<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  QuickStart</td>
<td>3</td>
</tr>
<tr>
<td>2  PyTorch Neuron</td>
<td>5</td>
</tr>
<tr>
<td>3  TensorFlow Neuron</td>
<td>415</td>
</tr>
<tr>
<td>4  Neuron Apache MXNet (Incubating)</td>
<td>843</td>
</tr>
<tr>
<td>5  Neuron Tutorials</td>
<td>1027</td>
</tr>
<tr>
<td>6  Neuron Inference Performance</td>
<td>1029</td>
</tr>
<tr>
<td>7  What’s New</td>
<td>1033</td>
</tr>
<tr>
<td>8  Neuron Features</td>
<td>1049</td>
</tr>
<tr>
<td>9  Developer Flows</td>
<td>1053</td>
</tr>
<tr>
<td>10 Containers</td>
<td>1075</td>
</tr>
<tr>
<td>11 Application Notes</td>
<td>1103</td>
</tr>
<tr>
<td>12 Neuron FAQ</td>
<td>1141</td>
</tr>
<tr>
<td>13 Setup Guide</td>
<td>1153</td>
</tr>
<tr>
<td>14 Neuron Compiler</td>
<td>1195</td>
</tr>
<tr>
<td>15 Neuron runtime</td>
<td>1217</td>
</tr>
<tr>
<td>16 Neuron Tools</td>
<td>1251</td>
</tr>
<tr>
<td>17 Release Details</td>
<td>1323</td>
</tr>
<tr>
<td>18 Roadmap</td>
<td>1327</td>
</tr>
<tr>
<td>19 Support</td>
<td>1329</td>
</tr>
<tr>
<td>Index</td>
<td>1337</td>
</tr>
</tbody>
</table>
AWS Neuron is the SDK for AWS Inferentia, the custom designed machine learning chips enabling high-performance deep learning inference applications on EC2 Inf1 instances. Neuron includes a deep learning compiler, runtime and tools that are natively integrated into TensorFlow, PyTorch and Apache MXNet (Incubating). With Neuron, you can develop, profile, and deploy high-performance inference applications on top of EC2 Inf1 instances.

Check Release Details, Neuron Performance page and What’s New in Neuron 1.16.3 (01/05/2022) release.

1. Compile your trained model for Neuron using PyTorch/TensorFlow/MXNet APIs or with Sagemaker Neo

2. Deploy the compiled model to EC2 inf1 instance/s and execute the model for inference
A typical Neuron developer flow includes compilation phase and then deployment (inference) on inf1 instance/s.

To quickly start developing with Neuron:

1. Setup your environment to run one of the Neuron tutorials on AWS ML accelerator instance:
   - pytorch-quickstart
   - tensorflow-quickstart
   - mxnet-quickstart
   You can also check Setup Guide for more options of installing Neuron.
   For Neuron containers setup please visit Containers.

2. Run a tutorial from one of the leading machine learning frameworks supported by Neuron:
   - PyTorch Tutorials
   - TensorFlow Tutorials
   - Neuron Apache MXNet (Incubating) Tutorials

3. Learn more about Neuron
   - Neuron Features
   - Application Notes
   - Inherentia Model Architecture Fit
   - Roadmap
   - Developer Flows
Customers can train their models anywhere and easily migrate their ML applications to Neuron and run their high-performance production predictions with Inferentia. Once a model is trained to the required accuracy, model is compiled to an optimized binary form, referred to as a Neuron Executable File Format (NEFF), and loaded by the Neuron runtime driver to execute inference input requests on the Inferentia chips. Developers have the option to train their models in fp16 or keep training in 32-bit floating point for best accuracy and Neuron will auto-cast the 32-bit trained model to run at speed of 16-bit using bfloat16.
Neuron is integrated into PyTorch, and provides you with a familiar environment to run inference using Inferentia based instances.

### 2.1 Installation Guide

#### 2.1.1 Install Neuron PyTorch

**Note:**
- Instructions in this page only apply to setting up Neuron components on Linux host running Ubuntu or Amazon Linux AMI.
- For an example of how to install Neuron components in a container, see [Docker environment setup](#) and our [Containers](#) documentation for more details.

- Develop on AWS ML accelerator instance
- Compile on compute instance
- Deploy on AWS ML accelerator instance

**Develop on AWS ML accelerator instance**

The simplest environment setup for model development installs all Neuron SDK components directly on an AWS ML accelerator instance: the Neuron framework extensions, compiler, runtime, and tools. This will allow you to compile, execute, and performance tune your model, all in the same instance. This is the recommended workflow when first starting to work with Neuron device or when optimizing a model.

**Important:**

**For successful installation or update to Neuron 1.16.0 and newer from previous releases:**

- Stop Neuron Runtime 1.x daemon (neuron-rtd) by running: `sudo systemctl stop neuron-rtd`
- Uninstall neuron-rtd by running: `sudo apt remove aws-neuron-runtime` or `sudo yum remove aws-neuron-runtime`
- Install or upgrade to latest Neuron driver (aws-neuron-dkms) by following the “Setup Guide” instructions.
- Visit *Introducing Neuron Runtime 2.x (libnrt.so)* for more information.

### PyTorch 1.9.1
### PyTorch 1.8.1
### PyTorch 1.7.1
### PyTorch 1.5.1
### Ubuntu AMI
### Amazon Linux AMI
### Ubuntu DLAMI
### Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
./etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y
```

(continues on next page)
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name "Python (Neuron PyTorch)"
ip install jupyter notebook
copy environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron neuron-cc[tensorflow] torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   → 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo yum install aws-neuron-tools -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name "Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron neuron-cc[tensorflow] torchvision
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron neuron-cc[tensorflow] torchvision
```

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo yum install aws-neuron-tools -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron neuron-cc[tensorflow] torchvision

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
debs https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
#   or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
```

(continues on next page)
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name "Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron PyTorch
pip install torch-neuron==1.8.1.* neuron-cc[tensorflow] torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo yum install aws-neuron-tools -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name "Python (Neuron PyTorch)"

pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.* neuron-cc[tensorflow] torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate PyTorch
source activate aws_neuron_pytorch_p36

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

#############################################################################
→

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

→

# Activate PyTorch
source activate aws_neuron_pytorch_p36

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

(continues on next page)```
# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name "Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.* neuron-cc[tensorflow] torchvision

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB
(continues on next page)
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo yum install aws-neuron-tools -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++ python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name "Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron PyTorch
pip install torch-neuron==1.7.1.* neuron-cc[tensorflow] torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   →'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
#      Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron PyTorch
pip install torch-neuron==1.7.1.* neuron-cc[tensorflow] torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   - `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo yum install aws-neuron-tools -y
export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

Install Neuron PyTorch
pip install torch-neuron==1.7.1.* neuron-cc[tensorflow] torchvision

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
./etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   {'sudo systemctl stop neuron-rtd'}

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

(continues on next page)
# Install Neuron Tools
```bash
sudo apt-get install aws-neuron-tools -y
export PATH=/opt/aws/neuron/bin:$PATH
```

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
```bash
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip
```

# Install Jupyter notebook kernel
```bash
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name "Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron PyTorch
```bash
pip install torch-neuron==1.5.1.* neuron-cc[tensorflow] torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB
```

# Update OS packages
```bash
sudo yum update -y
```

# Before installing or updating aws-neuron-dkms:
```bash
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   -> 'sudo systemctl stop neuron-rtd'
```
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
- DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

Install OS headers:
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

Install Neuron Driver:
sudo yum install aws-neuron-dkms -y

Warning: If Linux kernel is updated as a result of OS package update
Neuron driver (aws-neuron-dkms) should be re-installed after reboot

Install Neuron Tools:
sudo yum install aws-neuron-tools -y

export PATH=/opt/aws/neuron/bin:$PATH

Install Python venv and activate Python virtual environment to install Neuron pip packages:
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

Install Jupyter notebook kernel:
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name "Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels

Set Pip repository to point to the Neuron repository:
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

Install Neuron PyTorch:
pip install torch-neuron==1.5.1.* neuron-cc[tensorflow] torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
Framework will be installed/updated inside a Python environment

(continues on next page)
# Update OS packages
```
sudo apt-get update -y
```

# Before installing or updating aws-neuron-dkms:
```
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
```

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
```
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
```

# Install OS headers
```
sudo apt-get install linux-headers-$(uname -r) -y
```

# Install Neuron Driver
```
sudo apt-get install aws-neuron-dkms -y
```

# Warning: If Linux kernel is updated as a result of OS package update
```
#   Neuron driver (aws-neuron-dkms) should be re-installed after reboot.
```

# Install Neuron Tools
```
sudo apt-get install aws-neuron-tools -y
```

```
export PATH=/opt/aws/neuron/bin:$PATH
```

# Activate PyTorch
```
source activate aws_neuron_pytorch_p36
```

# Set Pip repository to point to the Neuron repository
```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron PyTorch
```
pip install torch-neuron==1.5.1.* neuron-cc[tensorflow] torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
```
# Framework will be installed/updated inside a Python environment
```

(continues on next page)
# Update OS packages
```bash
sudo yum update -y
```

# Before installing or updating aws-neuron-dkms:
```bash
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
→ 'sudo systemctl stop neuron-rtd'
```

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
```
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
```

# Install OS headers
```bash
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
```

# Install Neuron Driver
```bash
sudo yum install aws-neuron-dkms -y
```

# Warning: If Linux kernel is updated as a result of OS package update
```
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```

# Install Neuron Tools
```bash
sudo yum install aws-neuron-tools -y
```

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
```bash
source activate aws_neuron_pytorch_p36
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron PyTorch
```bash
pip install torch-neuron==1.5.1.* neuron-cc[tensorflow] torchvision
```
Compile on compute instance

If model compilation occurs outside the model deployment environment, you can install only the Neuron framework extensions and the compiler on any compute instance. This setup is helpful when compiling large complex models that require large amount of memory or during a CICD process where models are compiled in a separate step, prior to deployment.

PyTorch 1.9.1
PyTorch 1.8.1
PyTorch 1.7.1
PyTorch 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install # Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron neuron-cc[tensorflow] torchvision
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install # Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron neuron-cc[tensorflow] torchvision
```
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```python
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron neuron-cc[tensorflow] torchvision
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```python
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron neuron-cc[tensorflow] torchvision
```

Ubuntu AMI

Amazon Linux AMI

Ubuntu DLAMI

Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```python
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install --user pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

(continues on next page)
#Install Neuron PyTorch

pip install torch-neuron==1.8.1.* neuron-cc[tensorflow] torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.* neuron-cc[tensorflow] torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation

# Activate PyTorch
source activate aws_neuron_pytorch_p36
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation

# Activate PyTorch
source activate aws_neuron_pytorch_p36
```

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.* neuron-cc[tensorflow] torchvision
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.* neuron-cc[tensorflow] torchvision
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.* neuron-cc[tensorflow] torchvision
```

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.* neuron-cc[tensorflow] torchvision
```

**Ubuntu AMI**

**Amazon Linux AMI**

**Ubuntu DLAMI**

**Amazon Linux DLAMI**

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.  
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.* neuron-cc[tensorflow] torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.  
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.* neuron-cc[tensorflow] torchvision
```

(continues on next page)
# Install Neuron PyTorch

```bash
pip install torch-neuron==1.5.1.* neuron-cc[tensorflow] torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.* neuron-cc[tensorflow] torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

**Deploy on AWS ML accelerator instance**

During deployment it can be beneficial to reduce the number of components installed in the system. For use-cases where only inference is necessary (compilation is already complete), only the framework and runtime should be installed.

**Important:**

**For successful installation or update to Neuron 1.16.0 and newer from previous releases:**

- Stop Neuron Runtime 1.x daemon (`neuron-rtd`) by running: `sudo systemctl stop neuron-rtd`
- Uninstall `neuron-rtd` by running: `sudo apt remove aws-neuron-runtime` or `sudo yum remove aws-neuron-runtime`
- Install or upgrade to latest Neuron driver (`aws-neuron-dkms`) by following the “Setup Guide” instructions.
Visit *Introducing Neuron Runtime 2.x (libnrt.so)* for more information.

PyTorch 1.9.1
PyTorch 1.8.1
PyTorch 1.7.1
PyTorch 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
./etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -
# Update OS packages
sudo apt-get update -y
#########################################################################
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
#########################################################################
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
#########################################################################
# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y
# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y
#########################################################################
# Warning: If Linux kernel is updated as a result of OS package update
```

(continues on next page)
Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron PyTorch
pip install torch-neuron torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
# Install OS headers
sudo yum install kernel-devel-$\{(uname -r)\} kernel-headers-$\{(uname -r)\} -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
# Install OS headers
sudo apt-get install linux-headers-$\{$(uname -r)\} -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# \$('sudo systemctl stop neuron-rtd')

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron torchvision

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y
# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
export PATH=/opt/aws/neuron/bin:$PATH
# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
#Install Neuron PyTorch
pip install torch-neuron==1.8.1.* torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB
# Update OS packages
sudo yum update -y
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   ```
sudo systemctl stop neuron-rtd
   ```

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
```bash
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
```

# Install Neuron Driver
```
sudo yum install aws-neuron-dkms -y
```

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

```bash
export PATH=/opt/aws/neuron/bin:$PATH
```

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
```
sudo yum install -y python3 gcc-c++ python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip
```

# Set Pip repository to point to the Neuron repository
```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron PyTorch
```
pip install torch-neuron==1.8.1.* torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation
```

(continues on next page)
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   → 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate PyTorch
source activate aws_neuron_pytorch_p36

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   → 'sudo systemctl stop neuron-rtd'

(continues on next page)
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

#############################################################################
# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

#############################################################################

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

#############################################################################

# Activate PyTorch
source activate aws_neuron_pytorch_p36

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

#############################################################################

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
```

(continues on next page)
# To install or update to Neuron versions 1.16.0 and newer from previous releases:

- DO NOT skip `aws-neuron-dkms` install or upgrade step, you MUST install or upgrade to latest Neuron driver

```
# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y
```

Warning: If Linux kernel is updated as a result of OS package update

```
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```

```
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip
```

```
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

```
# Install Neuron PyTorch
pip install torch-neuron==1.7.1.* torchvision
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y
```

(continues on next page)
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   → 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.* torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
```bash
sudo apt-get update -y
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   → 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y
# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH
# Activate PyTorch
source activate aws_neuron_pytorch_p36
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
# Install Neuron PyTorch
pip install torch-neuron==1.7.1.* torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y
```

(continues on next page)
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   → 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.* torchvision

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
./etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
debs https://apt.repos.neuron.amazonaws.com $\{VERSION_CODENAME\} main
EOF

(continues on next page)
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous
# releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
# or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.1 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous
# releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
#   or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

(continues on next page)
#Install Neuron PyTorch

```
pip install torch-neuron==1.5.1.* torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# - 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.* torchvision
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   →'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.* torchvision
```
2.1.2 Update to latest Neuron PyTorch

Note:

- Instructions in this page only apply to setting up Neuron components on Linux host running Ubuntu or Amazon Linux AMI.
- For an example of how to install Neuron components in a container, see Docker environment setup and our Containers documentation for more details.

Develop on AWS ML accelerator instance

The simplest environment setup for model development installs all Neuron SDK components directly on an AWS ML accelerator instance: the Neuron framework extensions, compiler, runtime, and tools. This will allow you to compile, execute, and performance tune your model, all in the same instance. This is the recommended workflow when first starting to work with Neuron device or when optimizing a model.

Important:

For successful installation or update to Neuron 1.16.0 and newer from previous releases:

- Stop Neuron Runtime 1.x daemon (neuron-rtd) by running: sudo systemctl stop neuron-rtd
- Uninstall neuron-rtd by running: sudo apt remove aws-neuron-runtime or sudo yum remove aws-neuron-runtime
- Install or upgrade to latest Neuron driver (aws-neuron-dkms) by following the “Setup Guide” instructions.
- Visit Introducing Neuron Runtime 2.x (libnrt.so) for more information.

- Develop on AWS ML accelerator instance
- Compile on compute instance
- Deploy on AWS ML accelerator instance

PyTorch 1.9.1
PyTorch 1.8.1
PyTorch 1.7.1
PyTorch 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
#   or upgrade to latest Neuron driver

# Update OS headers
sudo apt-get install linux-headers-$(uname -r) -y
# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo apt-get install aws-neuron-tools -y
# Activate Python virtual environment where Neuron pip packages were installed
source pytorch_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron_neuron-cc[tensorflow] torchvision
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo yum update aws-neuron-tools -y

# Activate Python virtual environment where Neuron pip packages were installed
source pytorch_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron neuron-cc[tensorflow] torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
#   or upgrade to latest Neuron driver

# Update OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron neuron-cc[tensorflow] torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y
## Before installing or updating aws-neuron-dkms:

- `sudo systemctl stop neuron-rtd`

## To install or update to Neuron versions 1.16.0 and newer from previous releases:

- DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

### Update OS headers

```bash
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y
```

### Update Neuron Driver

```bash
sudo yum update aws-neuron-dkms -y
```

### Warning: If Linux kernel is updated as a result of OS package update

- Neuron driver (aws-neuron-dkms) should be re-installed after reboot

### Update Neuron Tools

```bash
sudo yum update aws-neuron-tools -y
```

### Activate PyTorch

```bash
source activate aws_neuron_pytorch_p36
```

### Set Pip repository to point to the Neuron repository

```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

### Update Neuron PyTorch

```bash
pip install --upgrade torch-neuron neuron-cc[tensorflow] torchvision
```

---

**Ubuntu AMI**

**Amazon Linux AMI**

**Ubuntu DLAMI**

**Amazon Linux DLAMI**

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Update OS packages
sudo apt-get update -y
```
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo apt-get install linux-headers-$\{\text{uname -r}\} -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Activate Python virtual environment where Neuron pip packages were installed
source pytorch_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron==1.8.1.* neuron-cc[tensorflow] torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

(continues on next page)
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   → 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo yum update aws-neuron-tools -y

# Activate Python virtual environment where Neuron pip packages were installed
source pytorch_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron==1.8.1.* neuron-cc[tensorflow] torchvision

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   → 'sudo systemctl stop neuron-rtd'

(continues on next page)
# To install or update to Neuron versions 1.16.0 and newer from previous releases:

- DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver.

# Update OS headers
```
sudo apt-get install linux-headers-$\{uname -r\} -y
```

# Update Neuron Driver
```
sudo apt-get install aws-neuron-dkms -y
```

# Warning: If Linux kernel is updated as a result of OS package update

- Neuron driver (aws-neuron-dkms) should be re-installed after reboot.

# Update Neuron Tools
```
sudo apt-get install aws-neuron-tools -y
```

# Activate PyTorch
```
source activate aws_neuron_pytorch_p36
```

# Set Pip repository to point to the Neuron repository
```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Update Neuron PyTorch
```
pip install --upgrade torch-neuron==1.8.1.* neuron-cc[tensorflow] torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
```
sudo yum update -y
```

# Before installing or updating aws-neuron-dkms:

- Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  ```
  sudo systemctl stop neuron-rtd
  ```
# To install or update to Neuron versions 1.16.0 and newer from previous releases:

- DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

### Update OS headers

```bash
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y
```

### Update Neuron Driver

```bash
sudo yum update aws-neuron-dkms -y
```

### Warning: If Linux kernel is updated as a result of OS package update

- Neuron driver (aws-neuron-dkms) should be re-installed after reboot

### Update Neuron Tools

```bash
sudo yum update aws-neuron-tools -y
```

### Activate PyTorch

```bash
source activate aws_neuron_pytorch_p36
```

### Set Pip repository to point to the Neuron repository

```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

### Update Neuron PyTorch

```bash
pip install --upgrade torch-neuron==1.8.1.* neuron-cc[tensorflow] torchvision
```

Ubuntu AMI

Amazon Linux AMI

Ubuntu DLAMI

Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
# Update OS headers
sudo apt-get install linux-headers-$\{(uname -r)\} -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Activate Python virtual environment where Neuron pip packages were installed
source pytorch_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron==1.7.1.* neuron-cc[tensorflow] torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#  'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

(continues on next page)
# Update OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo yum update aws-neuron-tools -y

# Activate Python virtual environment where Neuron pip packages were installed
source pytorch_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron==1.7.1.* neuron-cc[tensorflow] torchvision

---

# Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

(continues on next page)
# Update OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron==1.7.1.* neuron-cc[tensorflow] torchvision

## Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

(continues on next page)
# Update OS headers
sudo yum install kernel-devel-$\{\texttt{uname} \ -r\} \ kernel-headers-$\{\texttt{uname} \ -r\} \ -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms \ -y

#########
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after...
# reboot

# Update Neuron Tools
sudo yum update aws-neuron-tools \ -y

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron==1.7.1.* neuron-cc[tensorflow] torchvision

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update \ -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   \`sudo systemctl stop neuron-rtd\`

# To install or update to Neuron versions 1.16.0 and newer from previous...
# releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install...
# or upgrade to latest Neuron driver

(continues on next page)
# Update OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Update Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Activate Python virtual environment where Neuron pip packages were
# installed
source pytorch_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron==1.5.1.* neuron-cc[tensorflow] torchvision

---

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous
# releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
  # or upgrade to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

---

(continues on next page)
# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo yum update aws-neuron-tools -y

# Activate Python virtual environment where Neuron pip packages were installed
source pytorch_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron==1.5.1.* neuron-cc[tensorflow] torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo apt-get install linux-headers-$(uname -r) -y

(continues on next page)
# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron==1.5.1.* neuron-cc[tensorflow] torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

###########################################################################
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot.
###########################################################################

# Update Neuron Tools
sudo yum update aws-neuron-tools -y

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron==1.5.1.* neuron-cc[tensorflow] torchvision

## Compile on compute instance

If model compilation occurs outside the model deployment environment, you can install only the Neuron framework extensions and the compiler on any compute instance. This setup is helpful when compiling large complex models that require large amount of memory or during a CICD process where models are compiled in a separate step, prior to deployment.

PyTorch 1.9.1
PyTorch 1.8.1
PyTorch 1.7.1
PyTorch 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Activate Python virtual environment where Neuron pip packages were installed
source pytorch_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

(continues on next page)
# Update Neuron PyTorch

```bash
pip install --upgrade torch-neuron neuron-cc[tensorflow] torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Activate Python virtual environment where Neuron pip packages were installed
source pytorch_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron neuron-cc[tensorflow] torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron neuron-cc[tensorflow] torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron neuron-cc[tensorflow] torchvision
```

Ubuntu AMI

Amazon Linux AMI
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Activate Python virtual environment where Neuron pip packages were installed
source pytorch_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron==1.8.1.* neuron-cc[tensorflow] torchvision
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron==1.8.1.* neuron-cc[tensorflow] torchvision
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Activate PyTorch
source activate aws_neuron_pytorch_p36
```

(continues on next page)
# Set Pip repository to point to the Neuron repository
```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Update Neuron PyTorch
```
pip install --upgrade torch-neuron==1.8.1.* neuron-cc[tensorflow] torchvision
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Activate Python virtual environment where Neuron pip packages were installed
```
source pytorch_venv/bin/activate
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Activate Python virtual environment where Neuron pip packages were installed
```
source pytorch_venv/bin/activate
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment
```
source activate aws_neuron_pytorch_p36
```

(continues on next page)
# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Update Neuron PyTorch
```bash
pip install --upgrade torch-neuron==1.7.1.* neuron-cc[tensorflow] torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
```bash
source activate aws_neuron_pytorch_p36
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Update Neuron PyTorch
```bash
pip install --upgrade torch-neuron==1.7.1.* neuron-cc[tensorflow] torchvision
```

Ubuntu AMI

Amazon Linux AMI

Ubuntu DLAMI

Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Activate Python virtual environment where Neuron pip packages were installed
```bash
source pytorch_venv/bin/activate
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Update Neuron PyTorch
```bash
pip install --upgrade torch-neuron==1.5.1.* neuron-cc[tensorflow] torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Activate Python virtual environment where Neuron pip packages were installed
```bash
source pytorch_venv/bin/activate
```

(continues on next page)
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron==1.5.1.* neuron-cc[tensorflow] torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron==1.5.1.* neuron-cc[tensorflow] torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron==1.5.1.* neuron-cc[tensorflow] torchvision

**Deploy on AWS ML accelerator instance**

During deployment it can be beneficial to reduce the number of components installed in the system. For use-cases where only inference is necessary (compilation is already complete), only the framework and runtime should be installed.

**Important:**

**For successful installation or update to Neuron 1.16.0 and newer from previous releases:**

- Stop Neuron Runtime 1.x daemon (`neuron-rtd`) by running: `sudo systemctl stop neuron-rtd`
- Uninstall `neuron-rtd` by running: `sudo apt remove aws-neuron-runtime` or `sudo yum remove aws-neuron-runtime`
AWS Neuron

- Install or upgrade to latest Neuron driver (aws-neuron-dkms) by following the “Setup Guide” instructions.
- Visit *Introducing Neuron Runtime 2.x (libnrt.so)* for more information.

PyTorch 1.9.1
PyTorch 1.8.1
PyTorch 1.7.1
PyTorch 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```

(continues on next page)
# Activate Python virtual environment where Neuron pip packages were installed
source pytorch_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$ (uname -r) kernel-headers-$ (uname -r) -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate Python virtual environment where Neuron pip packages were installed
source pytorch_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   -> 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron torchvision
```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# - Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron torchvision

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo apt-get install linux-headers-$\{(uname -r)\} -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate Python virtual environment where Neuron pip packages were installed
source pytorch_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron==1.8.1.* torchvision
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Update OS packages
sudo yum update -y
```

(continues on next page)
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
```bash
sudo systemctl stop neuron-rtd
```

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
```bash
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
```

# Update Neuron Driver
```bash
sudo yum update aws-neuron-dkms -y
```

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```bash
source pytorch_venv/bin/activate
```

# Activate Python virtual environment where Neuron pip packages were installed
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Update Neuron PyTorch
```bash
pip install --upgrade torch-neuron==1.8.1.* torchvision
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
```bash
sudo apt-get update -y
```

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
```bash
sudo systemctl stop neuron-rtd
```
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron==1.8.1.* torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
# Update OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron==1.8.1.* torchvision

---

**Ubuntu AMI**

**Amazon Linux AMI**

**Ubuntu DLAMI**

**Amazon Linux DLAMI**

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
#   or upgrade to latest Neuron driver

(continues on next page)
# Update OS headers
```bash
sudo apt-get install linux-headers-$(uname -r) -y
```

# Update Neuron Driver
```bash
sudo apt-get install aws-neuron-dkms -y
```

```
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```

# Activate Python virtual environment where Neuron pip packages were installed
```bash
source pytorch_venv/bin/activate
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Update Neuron PyTorch
```bash
pip install --upgrade torch-neuron==1.7.1.* torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
```bash
sudo yum update -y
```

```
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
```

```
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
```

# Update OS headers
```bash
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
```

# Update Neuron Driver
```bash
sudo yum update aws-neuron-dkms -y
```

(continues on next page)
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot.

# Activate Python virtual environment where Neuron pip packages were installed
source pytorch_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron==1.7.1.* torchvision

## Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo apt-get install linux_headers-$ (uname -r) -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

### AWS Neuron

2.1. Installation Guide
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after

source activate aws_neuron_pytorch_p36

# Activate PyTorch

# Note: For a successful installation or update, execute each line of the instructions below separately or
# copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages

sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
#   or upgrade to latest Neuron driver

# Update OS headers

sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Update Neuron Driver

sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after

(continues on next page)
# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron==1.7.1.* torchvision

**Ubuntu AMI**

**Amazon Linux AMI**

**Ubuntu DLAMI**

**Amazon Linux DLAMI**

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate Python virtual environment where Neuron pip packages were installed
```
source pytorch_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron==1.5.1.* torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate Python virtual environment where Neuron pip packages were installed
source pytorch_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

(continues on next page)
# Update Neuron PyTorch

torch install --upgrade torch-neuron==1.5.1.* torchvısion

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo apt-get install linux-headers-$uname -r -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron==1.5.1.* torchvısion
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# -> 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron PyTorch
pip install --upgrade torch-neuron==1.5.1.* torchvision
```
2.1.3 Install previous Neuron PyTorch releases

Note:

• Instructions in this page only apply to setting up Neuron components on Linux host running Ubuntu or Amazon Linux AMI.

• For an example of how to install Neuron components in a container, see Docker environment setup and our Containers documentation for more details.

Install Neuron PyTorch (Neuron 1.16.2)

• Develop on AWS ML accelerator instance
• Compile on compute instance
• Deploy on AWS ML accelerator instance

Develop on AWS ML accelerator instance

The simplest environment setup for model development installs all Neuron SDK components directly on an AWS ML accelerator instance: the Neuron framework extensions, compiler, runtime, and tools. This will allow you to compile, execute, and performance tune your model, all in the same instance. This is the recommended workflow when first starting to work with Neuron device or when optimizing a model.

Important:
For successful installation or update to Neuron 1.16.0 and newer from previous releases:

• Stop Neuron Runtime 1.x daemon (neuron-rtd) by running: sudo systemctl stop neuron-rtd

• Uninstall neuron-rtd by running: sudo apt remove aws-neuron-runtime or sudo yum remove aws-neuron-runtime

• Install or upgrade to latest Neuron driver (aws-neuron-dkms) by following the “Setup Guide” instructions.

• Visit Introducing Neuron Runtime 2.x (libnrt.so) for more information.

PyTorch 1.9.1
PyTorch 1.8.1
PyTorch 1.7.1
PyTorch 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=2.0.327.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
```

(continues on next page)
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name
        "Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron PyTorch
pip install torch-neuron==1.9.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0
        torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-
        AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
# # To install or update to Neuron versions 1.16.0 and newer from previous
#   releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
#   or upgrade to latest Neuron driver

(continues on next page)
# Install OS headers
```
sudo yum install kernel-devel-$({uname -r} kernel-headers-$({uname -r}) -y
```
# Install Neuron Driver
```
# If you are downgrading from newer version , please remove existing package
-> using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.2.6.0 -y
```
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
-> reboot

# Install Neuron Tools
```
# If you are downgrading from newer version , please remove existing package
-> using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=2.0.327.0 -y
```
```
export PATH=/opt/aws/neuron/bin:$PATH
```
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
```
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip
```
# Instal Jupyter notebook kernel
```
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name
-> "Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels
```
# Set Pip repository to point to the Neuron repository
```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```
# Install Neuron PyTorch
```
pip install torch-neuron==1.9.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment
```
# Update OS packages
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous
# releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
# or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version , please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Tools
# If you are downgrading from newer version , please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=2.0.327.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.9.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0

Ubuntu AMI
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Configure Linux for Neuron repository updates
/etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$uname -r -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=2.0.327.0 -y
```

(continues on next page)
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name
"Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
# Install Neuron PyTorch
pip install torch-neuron==1.8.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous
# releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

```
# Install OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y
```

```
# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.2.6.0 -y
```

```
# Warning: If Linux kernel is updated as a result of OS package update, Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```

```
# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=2.0.327.0 -y
```

```
export PATH=/opt/aws/neuron/bin:$PATH
```

```
# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip
```

```
# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name "Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels
```

```
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

```
# Install Neuron PyTorch
pip install torch-neuron==1.8.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0 torchvision
```

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Update OS packages

```bash
sudo apt-get update -y
```

---

Before installing or updating aws-neuron-dkms:

- Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:

```bash
sudo systemctl stop neuron-rtd
```

---

To install or update to Neuron versions 1.16.0 and newer from previous releases:

- DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

---

# Install OS headers

```bash
sudo apt-get install linux-headers-$(uname -r) -y
```

# Install Neuron Driver

- If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'

```bash
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y
```

---

# Warning: If Linux kernel is updated as a result of OS package update Neuron driver (aws-neuron-dkms) should be re-installed after reboot

---

# Install Neuron Tools

- If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'

```bash
sudo apt-get install aws-neuron-tools=2.0.327.0 -y
```

```bash
export PATH=/opt/aws/neuron/bin:$PATH
```

# Activate PyTorch

```bash
source activate aws_neuron_pytorch_p36
```

# Set Pip repository to point to the Neuron repository

```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron PyTorch

```bash
pip install torch-neuron==1.8.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0 torchvision
```

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Update OS packages
```shell
sudo yum update -y
```

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
```shell
sudo systemctl stop neuron-rtd
```

# To install or update to Neuron versions 1.16.0 and newer from previous
# releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
# or upgrade to latest Neuron driver

# Install OS headers
```shell
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
```

# Install Neuron Driver
```shell
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package

clean install aws-neuron-dkms=2.2.6.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after,
```shell
export PATH=/opt/aws/neuron/bin:$PATH
```

# Install Neuron Tools
```shell
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package

clean install aws-neuron-tools=2.0.327.0 -y
```

# Activate PyTorch
```shell
source activate aws_neuron_pytorch_p36
```

# Set Pip repository to point to the Neuron repository
```shell
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron PyTorch
```shell
pip install torch-neuron==1.8.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0
```

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ~(VERSION_CODENAME) main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-
NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=2.0.327.0 -y

export PATH=/opt/aws/neuron/bin:$PATH
# Install Python venv and activate Python virtual environment to install Neuron pip packages.
```bash
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip
```

# Install Jupyter notebook kernel
```bash
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name "Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron PyTorch
```bash
pip install torch-neuron==1.7.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0 torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
```bash
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB
```

# Update OS packages
```bash
sudo yum update -y
```

# Before installing or updating aws-neuron-dkms:
```bash
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
```

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
```bash
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
```

(continues on next page)
# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=2.0.327.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name
"Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=2.0.327.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or
**AWS Neuron**

copy the contents of the code block into a script file and source its contents.

```
# Note: There is no DLAMI Conda environment for this framework version
#     Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous
# releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
#   or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$({uname} -r) kernel-headers-$({uname} -r) -y

# Install Neuron Driver
# If you are downgrading from newer version , please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Tools
# If you are downgrading from newer version , please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=2.0.327.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0
```
# Configure Linux for Neuron repository updates
/etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'

(continues on next page)
sudo apt-get install aws-neuron-tools=2.0.327.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name
  "Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0
  torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
- DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
```
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
```

# Install Neuron Driver
- If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
```
sudo yum install aws-neuron-dkms=2.2.6.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
- If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
```
sudo yum install aws-neuron-tools=2.0.327.0 -y
```

```
export PATH=/opt/aws/neuron/bin:$PATH
```

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
```
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip
```

# Install Jupyter notebook kernel
```
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name "Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels
```

# Set Pip repository to point to the Neuron repository
```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron PyTorch
```
pip install torch-neuron==1.5.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$\{$\text{uname -r}\}$ -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after a reboot

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=2.0.327.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=2.0.327.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0
```

2.1. Installation Guide
**Compile on compute instance**

If model compilation occurs outside the model deployment environment, you can install only the Neuron framework extensions and the compiler on any compute instance. This setup is helpful when compiling large complex models that require large amount of memory or during a CICD process where models are compiled in a separate step, prior to deployment.

PyTorch 1.9.1
PyTorch 1.8.1
PyTorch 1.7.1
PyTorch 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.9.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0
   --torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

(continues on next page)
# Install Neuron PyTorch

```shell
pip install torch-neuron==1.9.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0 torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

Ubuntu AMI

Amazon Linux AMI

Ubuntu DLAMI

Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

```shell
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
```

(continues on next page)
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0
→torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install # Neuron pip packages.

```bash
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0
→torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Activate PyTorch

```bash
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0
→torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Activate PyTorch

```bash
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

(continues on next page)
# Install Neuron PyTorch

```bash
pip install torch-neuron==1.8.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

Ubuntu AMI

Amazon Linux AMI

Ubuntu DLAMI

Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0

```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0

```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
AWS Neuron

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0
→torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0
→torchvision
# Install Python venv and activate Python virtual environment to install Neuron pip packages.
```
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip
```

# Set Pip repository to point to the Neuron repository
```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron PyTorch
```
pip install torch-neuron==1.5.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0
```

## Note:
For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
```
source activate aws_neuron_pytorch_p36
```

# Set Pip repository to point to the Neuron repository
```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron PyTorch
```
pip install torch-neuron==1.5.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0
```

## Note:
For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
```
source activate aws_neuron_pytorch_p36
```

# Set Pip repository to point to the Neuron repository
```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron PyTorch
```
pip install torch-neuron==1.5.1.2.0.468.0 neuron-cc[tensorflow]==1.8.2.0
```

## Note:
For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
Deploy on AWS ML accelerator instance

During deployment it can be beneficial to reduce the number of components installed in the system. For use-cases where only inference is necessary (compilation is already complete), only the framework and runtime should be installed.

Important:

For successful installation or update to Neuron 1.16.0 and newer from previous releases:

- Stop Neuron Runtime 1.x daemon (neuron-rtd) by running: `sudo systemctl stop neuron-rtd`
- Uninstall `neuron-rtd` by running: `sudo apt remove aws-neuron-runtime` or `sudo yum remove aws-neuron-runtime`
- Install or upgrade to latest Neuron driver (aws-neuron-dkms) by following the “Setup Guide” instructions.
- Visit Introducing Neuron Runtime 2.x (libnrt.so) for more information.

PyTorch 1.9.1
PyTorch 1.8.1
PyTorch 1.7.1
PyTorch 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
```

(continues on next page)
### To install or update to Neuron versions 1.16.0 and newer from previous releases:

- DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

```bash
# Install OS headers
sudo apt-get install linux Headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global. extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.9.1.2.0.468.0 torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-KEY-AMAZON-AWS-NEURON.PUB
```

(continues on next page)
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.9.1.2.0.468.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  → 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y
# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.9.1.2.0.468.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$uname-r kernel-headers-$uname-r -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.9.1.2.0.468.0 torchvision

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository

(continues on next page)
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron PyTorch
grep install torch-neuron==1.8.1.2.0.468.0 torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
    name=Neuron YUM Repository
    baseurl=https://yum.repos.neuron.amazonaws.com
    enabled=1
    metadata_expire=0
EOF

# Import the GPG key for Neuron repository updates
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```

(continues on next page)
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.2.0.468.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux Headers-$\{(uname -r)\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add `--allow-downgrades`
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

(continues on next page)
export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.2.0.468.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$({uname -r}) kernel-headers-$({uname -r}) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
(continues on next page)
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.2.0.468.0 torchvision

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y
```

(continues on next page)
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron PyTorch
pip install torch-neuron==1.7.1.2.0.468.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  sudo systemctl stop neuron-rtd

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.2.0.468.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   ->'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:

(continues on next page)
# DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

```bash
# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.2.0.468.0 torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
```

(continues on next page)
# Install OS headers
`sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y`

# Install Neuron Driver
 # If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
`sudo yum install aws-neuron-dkms=2.2.6.0 -y`

# Warning: If Linux kernel is updated as a result of OS package update
 # Neuron driver (aws-neuron-dkms) should be re-installed after reboot
export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
`source activate aws_neuron_pytorch_p36`

# Set Pip repository to point to the Neuron repository
`pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com`

#Install Neuron PyTorch
`pip install torch-neuron==1.7.1.2.0.468.0 torchvision`

---

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
`. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
`sudo apt-get update -y`

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
 `sudo systemctl stop neuron-rtd`

(continues on next page)
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.2.0.468.0 torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF

(continues on next page)
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++ python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.2.0.468.0 torchvision
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.2.0.468.0 torchvision
```

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.2.0.468.0 torchvision
```
Install Neuron PyTorch (Neuron 1.16.1)

- Develop on AWS ML accelerator instance
- Compile on compute instance
- Deploy on AWS ML accelerator instance

Develop on AWS ML accelerator instance

The simplest environment setup for model development installs all Neuron SDK components directly on an AWS ML accelerator instance: the Neuron framework extensions, compiler, runtime, and tools. This will allow you to compile, execute, and performance tune your model, all in the same instance. This is the recommended workflow when first starting to work with Neuron device or when optimizing a model.

Important:

For successful installation or update to Neuron 1.16.0 and newer from previous releases:

- Stop Neuron Runtime 1.x daemon (neuron-rtd) by running: sudo systemctl stop neuron-rtd
- Uninstall neuron-rtd by running: sudo apt remove aws-neuron-runtime or sudo yum remove aws-neuron-runtime
- Install or upgrade to latest Neuron driver (aws-neuron-dkms) by following the “Setup Guide” instructions.
- Visit Introducing Neuron Runtime 2.x (libnrt.so) for more information.

PyTorch 1.9.1
PyTorch 1.8.1
PyTorch 1.7.1
PyTorch 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-\n--NEURON.PUB | sudo apt-key add -
```
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=2.0.327.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name "Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.9.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0

 Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$\{$uname -r\} kernel-headers-$\{$uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.2.6.0 -y

(continues on next page)
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

#############################################################################

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=2.0.327.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name "Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.9.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

#############################################################################

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  'sudo systemctl stop neuron-rtd'

# Install Neuron PyTorch
pip install torch-neuron==1.9.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0 torchvision

(continues on next page)
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
- DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
```bash
sudo apt-get install linux-headers-$(uname -r) -y
```

# Install Neuron Driver
- If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
```bash
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
- If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
```bash
sudo apt-get install aws-neuron-tools=2.0.327.0 -y
```

# Activate PyTorch
```bash
export PATH=/opt/aws/neuron/bin:$PATH
```

# Activate PyTorch
```bash
source activate aws_neuron_pytorch_p36
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron PyTorch
```bash
pip install torch-neuron==1.9.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
- Framework will be installed/updated inside a Python environment

# Update OS packages
```bash
sudo yum update -y
```

# Before installing or updating aws-neuron-dkms:
- Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:'sudo systemctl stop neuron-rtd'
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=2.0.327.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.9.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0
torchvision

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Configure Linux for Neuron repository updates

```bash
/etc/os-release
```

```bash
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
```

```bash
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -
```

```bash
# Update OS packages
sudo apt-get update -y
```

```bash
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
```

```bash
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
```

```bash
# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y
```

```bash
# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y
```

```bash
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```

```bash
# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=2.0.327.0 -y
```

```bash
export PATH=/opt/aws/neuron/bin:$PATH
```

```bash
# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip
```

(continues on next page)
# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name
"Python (Neuron PyTorch)"

pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0
->torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y
```

(continues on next page)
# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
# sudo yum install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
# sudo yum install aws-neuron-tools=2.0.327.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
# sudo yum install -y python3 gcc-c++
# python3 -m venv pytorch_venv
# source pytorch_venv/bin/activate
# pip install -U pip

# Install Jupyter notebook kernel
# pip install ipykernel
# python -m ipykernel install --user --name pytorch_venv --display-name "Python (Neuron PyTorch)"
# pip install jupyter notebook
# pip install environment_kernels

# Set Pip repository to point to the Neuron repository
# pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
# pip install torch-neuron==1.8.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
# sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
# or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=2.0.327.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
→ 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
→ or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$\langle$uname -r$\rangle$ kernel-headers-$\langle$uname -r$\rangle$ -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
→ using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
→ reboot

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
→ using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=2.0.327.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0
→ torchvision

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
/etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=2.0.327.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv

(continues on next page)
source pytorch_venv/bin/activate

pip install --U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name
"Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0
--torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers

(continues on next page)
sudo yum install kernel-devel-$\{\text{uname -r}\}$ kernel-headers-$\{\text{uname -r}\}$ -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=2.0.327.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name "Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y
Before installing or updating aws-neuron-dkms:
- Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  ```bash
  sudo systemctl stop neuron-rtd
  ```

To install or update to Neuron versions 1.16.0 and newer from previous releases:
- DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

Install OS headers
```bash
sudo apt-get install linux-headers-$\{uname -r\} -y
```

Install Neuron Driver
```bash
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y
```

Warning: If Linux kernel is updated as a result of OS package update
- Neuron driver (aws-neuron-dkms) should be re-installed after reboot

Install Neuron Tools
```bash
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=2.0.327.0 -y
```

Export PATH
```bash
export PATH=/opt/aws/neuron/bin:$PATH
```

Activate PyTorch
```bash
source activate aws_neuron_pytorch_p36
```

Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

Install Neuron PyTorch
```bash
#Install Neuron PyTorch
pip install torch-neuron==1.7.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0
  torchvision
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

```bash
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=2.0.327.0 -y

export PATH=/opt/aws/neuron/bin:$PATH
# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0
```

Ubuntu AMI

Amazon Linux AMI
## AWS Neuron

### Ubuntu DLAMI

### Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=2.0.327.0 -y
```

(continues on next page)
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name
"Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron PyTorch
pip install torch-neuron==1.5.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0
	torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

###################################################

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
###################################################

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=2.0.327.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name "Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=2.0.327.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0
torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=2.0.327.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0
```

Chapter 2. PyTorch Neuron
Compile on compute instance

If model compilation occurs outside the model deployment environment, you can install only the Neuron framework extensions and the compiler on any compute instance. This setup is helpful when compiling large complex models that require a large amount of memory or during a CICD process where models are compiled in a separate step, prior to deployment.

PyTorch 1.9.1
PyTorch 1.8.1
PyTorch 1.7.1
PyTorch 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.9.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0.0 --torchvision
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

(continues on next page)
# Install Neuron PyTorch
pip install torch-neuron==1.9.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0
→ torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.9.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0
→ torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.9.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0
→ torchvision

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0
→torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0
→torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0
→torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Install Neuron PyTorch

```bash
pip install torch-neuron==1.8.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0
```  

Ubuntu AMI  
Amazon Linux AMI  
Ubuntu DLAMI  
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install  
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install  
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0

2.1. Installation Guide
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0

→torchvision
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0

→torchvision
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.2.0.392.0 neuron-cc[tensorflow]==1.7.3.0

→torchvision
```
Deploy on AWS ML accelerator instance

During deployment it can be beneficial to reduce the number of components installed in the system. For use-cases where only inference is necessary (compilation is already complete), only the framework and runtime should be installed.

Important:

For successful installation or update to Neuron 1.16.0 and newer from previous releases:

- Stop Neuron Runtime 1.x daemon (neuron-rtd) by running: `sudo systemctl stop neuron-rtd`
- Uninstall neuron-rtd by running: `sudo apt remove aws-neuron-runtime` or
  `sudo yum remove aws-neuron-runtime`
- Install or upgrade to latest Neuron driver (aws-neuron-dkms) by following the “Setup Guide” instructions.
- Visit Introducing Neuron Runtime 2.x (libnrt.so) for more information.

PyTorch 1.9.1
PyTorch 1.8.1
PyTorch 1.7.1
PyTorch 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  # 'sudo systemctl stop neuron-rtd'
```
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global. extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.9.1.2.0.392.0 torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB
# Update OS packages
```bash
sudo yum update -y
```

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
```bash
sudo systemctl stop neuron-rtd
```

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
```bash
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y
```

# Install Neuron Driver
```bash
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.2.6.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
```bash
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron PyTorch
```bash
pip install torch-neuron==1.9.1.2.0.392.0 torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.9.1.2.0.392.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Update OS packages
```bash
sudo yum update -y
```

# Before installing or updating aws-neuron-dkms:
# Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
```bash
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y
```

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
```bash
sudo yum install aws-neuron-dkms=2.2.6.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```bash
export PATH=/opt/aws/neuron/bin:$PATH
```

# Activate PyTorch
```bash
source activate aws_neuron_pytorch_p36
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron PyTorch
```bash
pip install torch-neuron==1.9.1.2.0.392.0 torchvision
```

---

**Ubuntu AMI**

**Amazon Linux AMI**

**Ubuntu DLAMI**

**Amazon Linux DLAMI**

**Note:** For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
 export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++ python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron PyTorch
pip install torch-neuron==1.8.1.2.0.392.0 torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-${uname -r} kernel-headers-${uname -r} -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```

(continues on next page)
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.2.0.392.0 torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$uname -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

(continues on next page)
export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.2.0.392.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Reboot
export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.2.0.392.0 torchvision

Ubuntu AMI

Amazon Linux AMI

Ubuntu DLAMI

Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux_headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
export PATH=/opt/aws/neuron/bin:$PATH
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
# Install Neuron PyTorch
pip install torch-neuron==1.7.1.2.0.392.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  sudo systemctl stop neuron-rtd
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

(continues on next page)
# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
# sudo yum install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.2.0.392.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.2.0.392.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

(continues on next page)
# Install OS headers

```bash
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y
```

# Install Neuron Driver

# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package

```bash
sudo yum install aws-neuron-dkms=2.2.6.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update

# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

```bash
export PATH=/opt/aws/neuron/bin:$PATH
```

# Activate PyTorch

```bash
source activate aws_neuron_pytorch_p36
```

# Set Pip repository to point to the Neuron repository

```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron PyTorch

```bash
pip install torch-neuron==1.7.1.2.0.392.0 torchvision
```

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
./etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'
```
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron Pytorch
pip install torch-neuron==1.5.1.2.0.392.0 torchvision

## Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF

(continues on next page)
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
#   or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.2.0.392.0 torchvision
**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$\{(uname -r)\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.2.0.392.0 torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package before installing the older package
sudo yum install aws-neuron-dkms=2.2.6.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.2.0.392.0 torchvision
```
Install Neuron PyTorch (Neuron 1.15.2)

- Develop on AWS ML accelerator instance
- Compile on compute instance
- Deploy on AWS ML accelerator instance

Develop on AWS ML accelerator instance

The simplest environment setup for model development installs all Neuron SDK components directly on an AWS ML accelerator instance: the Neuron framework extensions, compiler, runtime, and tools. This will allow you to compile, execute, and performance tune your model, all in the same instance. This is the recommended workflow when first starting to work with Neuron device or when optimizing a model.

PyTorch 1.8.1
PyTorch 1.7.1
PyTorch 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$({uname -r}) -y

# Install Neuron Driver
```

(continues on next page)
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# If Linux kernel is updated as a result of OS package update Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name "Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
EOF

(continued on next page)
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF

sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --display-name "Python (Neuron PyTorch)"

(continues on next page)
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0 -t torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package:
#   'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after:
#   'reboot'

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package:
#   'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package:
#   'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note:  For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after a
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '-Allow-downgrades'
  option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name
  "Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
  torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

################################################################################
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  'sudo systemctl stop neuron-rtd'
# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name
"Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$uname_r kernel-headers-$uname_r -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0 torchvision

Ubuntu AMI

Amazon Linux AMI
```
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.

(continues on next page)
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name "Python (Neuron PyTorch)"

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   →'sudo systemctl stop neuron-rtd'
#############################################################################
→##################################
# Install OS headers
sudo apt-get install linuxHeaders-$uname_r -y
# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
→option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y
#############################################################################
→#######
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
→reboot
#############################################################################
→#######
# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
→option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y
# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
→option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y
export PATH=/opt/aws/neuron/bin:$PATH
# Activate PyTorch
source activate aws_neuron_pytorch_p36
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
#Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
→torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment
# Update OS packages
sudo yum update -y

(continues on next page)
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  → 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package,
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package,
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package,
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0

Compile on compute instance

If model compilation occurs outside the model deployment environment, you can install only the Neuron framework extensions and the compiler on any compute instance. This setup is helpful when compiling large complex models that require large amount of memory or during a CICD process where models are compiled in a separate step, prior to deployment.

PyTorch 1.8.1
PyTorch 1.7.1
PyTorch 1.5.1
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

(continues on next page)
AWS Neuron

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
torchvision

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate

(continues on next page)
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0 torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0 torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Install Python venv and activate Python virtual environment to install # Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
→torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install # Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
→torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version # Framework will be installed/updated inside a Python environment

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
→torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0

Deploy on AWS ML accelerator instance

During deployment it can be beneficial to reduce the number of components installed in the system. For use-cases where only inference is necessary (compilation is already complete), only the framework and runtime should be installed.

PyTorch 1.8.1
PyTorch 1.7.1
PyTorch 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
./etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# Install OS headers

(continues on next page)
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
← option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

#############################################################################

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
←reboot
#############################################################################

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
← option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.5.2.1.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-
→AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

#############################################################################

(continues on next page)
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  →'sudo systemctl stop neuron-rtd'
#############################################################################
→##################################

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package:
  →using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

#############################################################################
→#######
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
#############################################################################
→#######

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package:
  →using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

(continues on next page)
# Install OS headers
```bash
sudo apt-get install linux-headers-$\{uname -r\} -y
```

# Install Neuron Driver
- If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
```bash
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y
```

---

# Warning: If Linux kernel is updated as a result of OS package update
- Neuron driver (aws-neuron-dkms) should be re-installed after reboot.

---

# Install Neuron Runtime server
- If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
```bash
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y
```

```bash
export PATH=/opt/aws/neuron/bin:$PATH
```

# Activate PyTorch
```bash
source activate aws_neuron_pytorch_p36
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron PyTorch
```bash
pip install torch-neuron==1.8.1.1.5.21.0 torchvision
```

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Update OS packages
sudo yum update -y
```

---

# Before installing or updating aws-neuron-dkms:
- Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
```bash
sudo systemctl stop neuron-rtd
```

---

# Install OS headers
```bash
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y
```

# Install Neuron Driver
- If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package

---

(continues on next page)
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 torchvision

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
sudo systemctl stop neuron-rtd

(continues on next page)
# Install OS headers
sudo apt-get install linux-headers-$\{(uname -r)\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 torchvision

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-
AWS-NEURON.PUB

# Update OS packages
sudo yum update -y
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   → 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++ python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 torchvision

---

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
# Install OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 torchvision

### Ubuntu AMI

### Amazon Linux AMI

### Ubuntu DLAMI

### Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
debs https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
eof
wget -qO - https://apt.repos.neuron.amazonaws.com/KEY-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

(continues on next page)
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.5.21.0 torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
EOF
# Install OS headers
sudo yum install kernel-devel-$\{(uname -r)\} kernel-headers-$\{(uname -r)\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.5.21.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.21.0 torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   → 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version , please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.5.21.0 torchvision

Install Neuron PyTorch (Neuron 1.15.1)

- Develop on AWS ML accelerator instance
- Compile on compute instance
- Deploy on AWS ML accelerator instance
Develop on AWS ML accelerator instance

The simplest environment setup for model development installs all Neuron SDK components directly on an AWS ML accelerator instance: the Neuron framework extensions, compiler, runtime, and tools. This will allow you to compile, execute, and performance tune your model, all in the same instance. This is the recommended workflow when first starting to work with Neuron device or when optimizing a model.

PyTorch 1.8.1
PyTorch 1.7.1
PyTorch 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y
# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```

(continues on next page)
# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
    → option to 'sudo apt-get install'
    sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
    → option to 'sudo apt-get install'
    sudo apt-get install aws-neuron-tools=1.7.25.0 -y

    export PATH=/opt/aws/neuron/bin:$PATH

    # Install Python venv and activate Python virtual environment to install
    # Neuron pip packages.
    sudo apt-get install -y python3-venv g++
    python3 -m venv pytorch_venv
    source pytorch_venv/bin/activate
    pip install -U pip

    # Install Jupyter notebook kernel
    pip install ipykernel
    python -m ipykernel install --user --name pytorch_venv --display-name
       →"Python (Neuron PyTorch)"
    pip install jupyter notebook
    pip install environment_kernels

    # Set Pip repository to point to the Neuron repository
    pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

    # Install Neuron PyTorch
    pip install torch-neuron==1.8.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0->torchvision

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-
AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

(continues on next page)
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  → 'sudo systemctl stop neuron-rtd'
# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
  → using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot
# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
  → using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y
# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
  → using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y
export PATH=/opt/aws/neuron/bin:$PATH
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip
# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name
  → "Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
  → torchvision
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# --> 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0 --torchvision

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or
# Configure Linux for Neuron repository updates

`. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`
# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

(continues on next page)
# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name "Python (Neuron PyTorch)"

pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron PyTorch
pip install torch-neuron==1.7.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  sudo systemctl stop neuron-rtd

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
  using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
→ using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
→ using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name
"Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# → 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add 'allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add 'allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add 'allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0

---

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

---

(continues on next page)
# Install OS headers
```bash
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y
```
# Install Neuron Driver
```bash
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y
```
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```bash
# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y
```
# Install Neuron Tools
```bash
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y
```
```bash
export PATH=/opt/aws/neuron/bin:$PATH
```
# Activate PyTorch
```bash
source activate aws_neuron_pytorch_p36
```
# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```
# Install Neuron PyTorch
```bash
pip install torch-neuron==1.7.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
```

Ubuntu AMI

Amazon Linux AMI

Ubuntu DLAMI

Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -
```
(continues on next page)
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neo
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name "Python (Neuron PyTorch)"

pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package

(continues on next page)
sudo yum install aws-neuron-tools=1.7.25.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name
"Python (Neuron PyTorch)"
ip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
"sudo systemct1 stop neuron-rtd"

# Install OS headers
sudo apt-get install linux-headers-$\langle$uname -r\rangle$ -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
\langle option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

(continues on next page)
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

## Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

## Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0

torchvision

---

# Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

## Note: There is no DLAMI Conda environment for this framework version

## Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0

Compile on compute instance

If model compilation occurs outside the model deployment environment, you can install only the Neuron framework extensions and the compiler on any compute instance. This setup is helpful when compiling large complex models that require large amount of memory or during a CICD process where models are compiled in a separate step, prior to deployment.

PyTorch 1.8.1
PyTorch 1.7.1
PyTorch 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
→torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
→torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
→torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
→ torchvision

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
→ torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
→ torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
torchvision
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
→torchvision
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
→torchvision
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
→torchvision
```
Deploy on AWS ML accelerator instance

During deployment it can be beneficial to reduce the number of components installed in the system. For use-cases where only inference is necessary (compilation is already complete), only the framework and runtime should be installed.

PyTorch 1.8.1
PyTorch 1.7.1
PyTorch 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```

(continues on next page)
# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url "https://pip.repos.neuron.amazonaws.com"

#Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

### Configure Linux for Neuron repository updates

```bash
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
```
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

(continues on next page)
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++ python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
```bash
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y
export PATH=/opt/aws/neuron/bin:$PATH
# Activate PyTorch
source activate aws_neuron_pytorch_p36
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Update OS packages
sudo yum update -y
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y
# Warning: If Linux kernel is updated as a result of OS package update
# Reboot
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```
sudo yum install aws-neuron-runtime=1.6.24.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 torchvision

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
./etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-
AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
# - Install OS headers
s  sudo yum install kernel-devel-${uname -r} kernel-headers-${uname -r} -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package

(continues on next page)
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$((uname -r)) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'

(continues on next page)
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# Install OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update

Neuron driver (aws-neuron-dkms) should be re-installed after reboot.

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 torchvision

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
./etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.21.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
#   'sudo systemctl stop neuron-runtime'

(continues on next page)
# Install OS headers
```bash
sudo yum install kernel-devel-$\{$uname -r\} kernel-headers-$\{$uname -r\} -y
```

# Install Neuron Driver
```bash
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y
```  

# Warning: If Linux kernel is updated as a result of OS package update
```bash
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```  

# Install Neuron Runtime server
```bash
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y
```  

```bash
export PATH=/opt/aws/neuron/bin:$PATH
```

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
```bash
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron PyTorch
```bash
pip install torch-neuron==1.5.1.5.21.0 torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
```bash
# Framework will be installed/updated inside a Python environment
```

# Update OS packages
```bash
sudo apt-get update -y
```

# Before installing or updating aws-neuron-dkms:
```bash
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   ->'sudo systemctl stop neuron-rtd'
```

(continues on next page)
# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
  option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
  reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
  option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.5.21.0 torchvision

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Neuron Driver
Install Neuron PyTorch (Neuron 1.15.0)

- Develop on AWS ML accelerator instance
- Compile on compute instance
- Deploy on AWS ML accelerator instance

Develop on AWS ML accelerator instance

The simplest environment setup for model development installs all Neuron SDK components directly on an AWS ML accelerator instance: the Neuron framework extensions, compiler, runtime, and tools. This will allow you to compile, execute, and performance tune your model, all in the same instance. This is the recommended workflow when first starting to work with Neuron device or when optimizing a model.

PyTorch 1.8.1
PyTorch 1.7.1
PyTorch 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
debug https://apt.repos.neuron.amazonaws.com $(VERSION_CODENAME) main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.20.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv

```
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name
"Python (Neuron PyTorch)"
# Install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
-->torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   "sudo systemctl stop neuron-rtd"
# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

(continues on next page)
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version , please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version , please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.20.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name "Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

(continues on next page)
# Install OS headers
```
sudo apt-get install linux-headers-$(uname -r) -y
```

# Install Neuron Driver
```
# If you are downgrading from newer version, please add '允许-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
```
# If you are downgrading from newer version, please add '允许-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y
```

# Install Neuron Tools
```
# If you are downgrading from newer version, please add '允许-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.20.0 -y
```

```
export PATH=/opt/aws/neuron/bin:$PATH
```

# Activate PyTorch
```
sudo systemctl stop neuron-rtd
```

# Set Pip repository to point to the Neuron repository
```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

```
pip install torch-neuron==1.8.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
torchvision
```

## Note
For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
sudo yum install kernel-devel-$\text{(uname -r)}$ kernel-headers-$\text{(uname -r)}$ -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

###
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot
###

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.20.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
debs https://apt.repos.neuron.amazonaws.com '${VERSION_CODENAME}' main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

(continues on next page)
# Update OS packages
```bash
sudo apt-get update -y
```

# Before installing or updating aws-neuron-dkms:
```bash
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`
```

# Install OS headers
```bash
sudo apt-get install linux-headers-$(uname -r) -y
```

# Install Neuron Driver
```bash
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update
```bash
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```

# Install Neuron Runtime server
```bash
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y
```

# Install Neuron Tools
```bash
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.20.0 -y
```

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
```bash
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
```
```bash
pip install -U pip
```

# Install Jupyter notebook kernel
```bash
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name "Python (Neuron PyTorch)"
```

```bash
pip install jupyter notebook
pip install environment_kernels
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

(continues on next page)
# Install Neuron PyTorch
```bash
pip install torch-neuron==1.7.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0<torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB
# Update OS packages
sudo yum update -y
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`
# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y
# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
```

(continues on next page)
sudo yum install aws-neuron-tools=1.7.20.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name
"Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

(continues on next page)
# Warning: If Linux kernel is updated as a result of OS package update
Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.20.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

(continues on next page)
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.20.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
./etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

(continues on next page)
# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.20.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name "Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Configure Linux for Neuron repository updates
```
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
```
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB
```
# Update OS packages
sudo yum update -y
```
```
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`
```
```
# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
```
```
# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
```
```
# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
```
```
# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
```
```
export PATH=/opt/aws/neuron/bin:$PATH
```
```
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
```
```
(continues on next page)
# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name
"Python (Neuron PyTorch)"
 pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0

---

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

(continues on next page)
# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
- option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.20.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron PyTorch
pip install torch-neuron==1.5.1.5.1.21.0 neuron-cc[tensorflow]==1.6.13.0
-torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   - 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version , please remove existing package
- using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version , please remove existing package
- using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y
# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.20.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0

## Compile on compute instance

If model compilation occurs outside the model deployment environment, you can install only the Neuron framework extensions and the compiler on any compute instance. This setup is helpful when compiling large complex models that require large amount of memory or during a CICD process where models are compiled in a separate step, prior to deployment.

PyTorch 1.8.1
PyTorch 1.7.1
PyTorch 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
→torchvision
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
→torchvision
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
→torchvision
```

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
→torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
→torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
→torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
  torchvision
```

Ubuntu AMI

Amazon Linux AMI

Ubuntu DLAMI

Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
  torchvision
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip
```

(continues on next page)
# Set Pip repository to point to the Neuron repository
```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron PyTorch
```
pip install torch-neuron==1.5.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
```

→ torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
```
source activate aws_neuron_pytorch_p36
```

# Set Pip repository to point to the Neuron repository
```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron PyTorch
```
pip install torch-neuron==1.5.1.1.5.21.0 neuron-cc[tensorflow]==1.6.13.0
```

→ torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

Deploy on AWS ML accelerator instance

During deployment it can be beneficial to reduce the number of components installed in the system. For use-cases where only inference is necessary (compilation is already complete), only the framework and runtime should be installed.

PyTorch 1.8.1
PyTorch 1.7.1
PyTorch 1.5.1
Ubuntu AMI
# Configure Linux for Neuron repository updates
/etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
debug https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
eof
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Install OS headers
sudo apt-get install linux-headers-$uname -r -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

(continues on next page)
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y

(continues on next page)
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install 
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron PyTorch
pip install torch-neuron==1.8.1.0.21.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or 
copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   -'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r$\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' 
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update 
# Neuron driver (aws-neuron-dkms) should be re-installed after
#reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' 
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  → 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package_
  → using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after_
→ reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package_
  → using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.21.0 torchvision

Ubuntu AMI
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip
```

(continues on next page)
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y

(continues on next page)
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install --upgrade pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add('--allow-downgrades')
#   option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add('--allow-downgrades')
#   option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

(continues on next page)
export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.21.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  # 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.5.21.0 torchvision

Ubuntu AMI

Amazon Linux AMI

Ubuntu DLAMI

Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after a reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.21.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# Install OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot.
# Install Neuron Runtime server

# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.21.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot
# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.21.0 torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# Install OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
```
sudo yum install aws-neuron-runtime=1.6.19.0 -y
export PATH=/opt/aws/neuron/bin:$PATH
# Activate PyTorch
source activate aws_neuron_pytorch_p36
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
# Install Neuron PyTorch
pip install torch-neuron==1.5.1.5.21.0 torchvision
```

Install Neuron PyTorch (Neuron 1.14.2)

- Develop on AWS ML accelerator instance
- Compile on compute instance
- Deploy on AWS ML accelerator instance

Develop on AWS ML accelerator instance

The simplest environment setup for model development installs all Neuron SDK components directly on an AWS ML accelerator instance: the Neuron framework extensions, compiler, runtime, and tools. This will allow you to compile, execute, and performance tune your model, all in the same instance. This is the recommended workflow when first starting to work with Neuron device or when optimizing a model.

PyTorch 1.8.1
PyTorch 1.7.1
PyTorch 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Update OS packages
```bash
sudo apt-get update -y
```

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
```bash
sudo systemctl stop neuron-rtd
```

# Install OS headers
```bash
sudo apt-get install linux-headers-$(uname -r) -y
```

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
```bash
sudo apt-get install aws-neuron-dkms=2.0.386.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
```bash
sudo apt-get install aws-neuron-runtime=1.6.9.0 -y
```

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
```bash
sudo apt-get install aws-neuron-tools=1.7.10.0 -y
```

```bash
export PATH=/opt/aws/neuron/bin:$PATH
```

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
```bash
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip
```

# Install Jupyter notebook kernel
```bash
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name "Python (Neuron PyTorch)"
```

```bash
pip install jupyter notebook
```

```bash
pip install environment_kernels
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

(continues on next page)
# Install Neuron PyTorch

```
pip install torch-neuron==1.8.1.15.12.0 neuron-cc[tensorflow]==1.5.5.0
torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   -> systemctl stop neuron-rtd

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.9.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
```

(continues on next page)
sudo yum install aws-neuron-tools=1.7.10.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name
"Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.12.0 neuron-cc[tensorflow]==1.5.5.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.9.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.10.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.12.0 neuron-cc[tensorflow]==1.5.5.0

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.386.0 -y

---

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after a reboot

---

(continues on next page)
# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.9.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.10.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.12.0 neuron-cc[tensorflow]==1.5.5.0

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
./etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$\{(uname -r)\} -y

(continues on next page)
# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
→ option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
→ reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
→ option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.9.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
→ option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.10.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name
→ "Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.12.0 neuron-cc[tensorflow]==1.5.5.0
→ torchvision

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
EOF
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF

sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-
AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.9.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.10.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name "Python (Neuron PyTorch)"

pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.12.0 neuron-cc[tensorflow]==1.5.5.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# Install OS headers
sudo apt-get install linux-headers-$uname -r -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.9.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.10.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.12.0 neuron-cc[tensorflow]==1.5.5.0

torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.9.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.10.0 -y
export PATH=/opt/aws/neuron/bin:$PATH
# Activate PyTorch
source activate aws_neuron_pytorch_p36
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.12.0 neuron-cc[tensorflow]==1.5.5.0

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.9.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.10.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name
"Python (Neuron PyTorch)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.5.12.0 neuron-cc[tensorflow]==1.5.5.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y
Before installing or updating aws-neuron-dkms:
- Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  ```bash
  sudo systemctl stop neuron-rtd
  ```

# Install OS headers
```bash
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
```

# Install Neuron Driver
- If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
  ```bash
  sudo yum install aws-neuron-dkms=2.0.386.0 -y
  ```

# Warning: If Linux kernel is updated as a result of OS package update
- Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
- If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
  ```bash
  sudo yum install aws-neuron-runtime=1.6.9.0 -y
  ```

# Install Neuron Tools
- If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
  ```bash
  sudo yum install aws-neuron-tools=1.7.10.0 -y
  ```

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
```bash
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip
```

# Install Jupyter notebook kernel
```bash
pip install ipykernel
python -m ipykernel install --user --name pytorch_venv --display-name "Python (Neuron PyTorch)"
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron PyTorch
(continues on next page)
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment
#
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.9.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.10.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

(continues on next page)
AWS Neuron

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.12.0 neuron-cc[tensorflow]==1.5.5.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment
#
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neutron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package:
#   using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after:
#   'reboot'

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package:
#   using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.9.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package:
#   using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.10.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

(continues on next page)
# Install Neuron PyTorch

```bash
# Install Python venv and activate Python virtual environment to install # Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.12.0 neuron-cc[tensorflow]==1.5.5.0
    torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

**Compile on compute instance**

If model compilation occurs outside the model deployment environment, you can install only the Neuron framework extensions and the compiler on any compute instance. This setup is helpful when compiling large complex models that require large amount of memory or during a CICD process where models are compiled in a separate step, prior to deployment.

PyTorch 1.8.1
PyTorch 1.7.1
PyTorch 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

```bash
# Install Python venv and activate Python virtual environment to install # Neuron pip packages.
sudo yum install -y python3 gcc-c+
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.12.0 neuron-cc[tensorflow]==1.5.5.0
    torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.12.0 neuron-cc[tensorflow]==1.5.5.0
→ torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.12.0 neuron-cc[tensorflow]==1.5.5.0
→ torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.12.0 neuron-cc[tensorflow]==1.5.5.0
→torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.12.0 neuron-cc[tensorflow]==1.5.5.0
→torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Activate PyTorch
```bash
source activate aws_neuron_pytorch_p36
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron PyTorch
```bash
pip install torch-neuron==1.7.1.1.5.12.0 neuron-cc[tensorflow]==1.5.5.0
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

Ubuntu AMI

Amazon Linux AMI

Ubuntu DLAMI

Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

Ubuntu AMI

```
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.12.0 neuron-cc[tensorflow]==1.5.5.0
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

Amazon Linux AMI

```
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.12.0 neuron-cc[tensorflow]==1.5.5.0
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1 neuron-cc[tensorflow]==1.5.5.0

Deploy on AWS ML accelerator instance

During deployment it can be beneficial to reduce the number of components installed in the system. For use-cases where only inference is necessary (compilation is already complete), only the framework and runtime should be installed.

PyTorch 1.8.1
PyTorch 1.7.1
PyTorch 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
eof
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.9.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.12.0 torchvision
```
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
  name=Neuron YUM Repository
  baseurl=https://yum.repos.neuron.amazonaws.com
  enabled=1
  metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  # 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Install Neuron Driver
# If you are downgrading from newer version , please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version , please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.9.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip
```

(continues on next page)
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.12.0 torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$\{(uname -r)\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.9.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.12.0 torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#     → 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.9.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.8.1.1.5.12.0 torchvision

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Configure Linux for Neuron repository updates

```
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -
```

# Update OS packages

```
sudo apt-get update -y
```

# Before installing or updating aws-neuron-dkms:

```
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  'sudo systemctl stop neuron-rtd'
```

# Install OS headers

```
sudo apt-get install linux-headers-$(uname -r) -y
```

# Install Neuron Driver

```
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.386.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update

```
# - Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```

# Install Neuron Runtime server

```
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.9.0 -y
```

```
export PATH=/opt/aws/neuron/bin:$PATH
```

# Install Python venv and activate Python virtual environment to install

```
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip
```

```
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

```
#Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.12.0 torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

```
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron Y UM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.9.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

(continues on next page)
# Install Neuron PyTorch

```
pip install torch-neuron==1.7.1.1.5.12.0 torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.9.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.12.0 torchvision
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or...
copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.9.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.7.1.1.5.12.0 torchvision
```

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or
# Configure Linux for Neuron repository updates
```bash
./etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -
```

# Update OS packages
```bash
sudo apt-get update -y
```

# Before installing or updating aws-neuron-dkms:
```bash
# Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'
```

# Install OS headers
```bash
sudo apt-get install linux-headers-$UNAME -r -y
```

# Install Neuron Driver
```bash
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.386.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update
```bash
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```

# Install Neuron Runtime server
```bash
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.9.0 -y
```

```bash
export PATH=/opt/aws/neuron/bin:$PATH
```

# Install Python venv and activate Python virtual environment to install
```bash
# Neuron pip packages.
sudo apt-get install -y python3-venv g++ python3 -m venv pytorch_venv
source pytorch_venv/bin/activate
pip install -U pip
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron PyTorch
```bash
pip install torch-neuron==1.5.1.15.12.0 torchvision
```
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.9.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
# sudo yum install -y python3 gcc-c++
# python3 -m venv pytorch_venv
# source pytorch_venv/bin/activate
# pip install -U pip
```

(continues on next page)
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.12.0 torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.9.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.12.0 torchvision
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version , please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version , please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.9.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron==1.5.1.1.5.12.0 torchvision
```
2.2 User Guide

2.2.1 PyTorch Tutorials

Before running a tutorial

You will run the tutorials on an inf1.6xlarge instance running Deep Learning AMI (DLAMI) to enable both compilation and deployment (inference) on the same instance. In a production environment we encourage you to try different instance sizes to optimize to your specific deployment needs.

Follow instructions at PyTorch Tutorial Setup before running a PyTorch tutorial on Inferentia. We recommend new users start with the ResNet-50 tutorial.

PyTorch Tutorial Setup

1. Launch an Inf1.6xlarge Instance:

   • Please follow the instructions at launch an Amazon EC2 Instance to Launch an Inf1 instance, when choosing the instance type at the EC2 console. Please make sure to select the correct instance type. To get more information about Inf1 instances sizes and pricing see Inf1 web page.

   • When choosing an Amazon Machine Image (AMI) make sure to select Deep Learning AMI with Conda Options. Please note that Neuron Conda environments are supported only in Ubuntu 18 DLAMI and Amazon Linux2 DLAMI, Neuron Conda environments are not supported in Amazon Linux DLAMI.

   • After launching the instance, follow the instructions in Connect to your instance to connect to the instance.

   [Note: You can also launch the instance from AWS CLI, please see AWS CLI commands to launch inf1 instances.]

2. Set up a development environment:

   [Important:

   For successful installation or update to Neuron 1.16.0 and newer from previous releases:

   • Stop Neuron Runtime 1.x daemon (neuron-rtd) by running: sudo systemctl stop neuron-rtd

   • Uninstall neuron-rtd by running: sudo apt remove aws-neuron-runtime or sudo yum remove aws-neuron-runtime

   • Install or upgrade to latest Neuron driver (aws-neuron-dkms) by following the “Setup Guide” instructions.

   • Visit Introducing Neuron Runtime 2.x (libnrt.so) for more information.

   • Enable or install PyTorch-Neuron:

     PyTorch 1.9.1]
PyTorch 1.8.1
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework
version
#    Framework will be installed/updated inside a Python environment
#
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y
export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36
```

(continues on next page)
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron neuron-cc[tensorflow] torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo yum install aws-neuron-tools -y
export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron neuron-cc[tensorflow] torchvision

Ubuntu DLAMI

Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

(continues on next page)
# Activate PyTorch

```
source activate aws_neuron_pytorch_p36
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest
→ DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest"
→ release" instruction on Neuron documentation

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by
→ calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from
→ previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you
→ MUST install or upgrade to latest Neuron driver

# Install OS headers
```
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
```

# Install Neuron Driver
```
sudo yum install aws-neuron-dkms -y
```

# Warning: If Linux kernel is updated as a result of OS package
→ update
# Neuron driver (aws-neuron-dkms) should be re-installed
→ after reboot

# Activate PyTorch
```
source activate aws_neuron_pytorch_p36
```

3. **Run tutorial in Jupyter notebook:**
   - Follow instruction at *Setup Jupyter notebook* to:
     1. Start the Jupyter Notebook on the instance
2. Run the Jupyter Notebook from your local browser

- Connect to the instance from the terminal, clone the Neuron Github repository to the Inf1 instance and then change the working directory to the tutorial directory:

  ```
  git clone https://github.com/aws/aws-neuron-sdk.git
  cd aws-neuron-sdk/src/examples/pytorch
  ```

- Locate the tutorial notebook file (.ipynb file) under `aws-neuron-sdk/src/examples/pytorch`

- From your local browser, open the tutorial notebook from the menu and follow the instructions.

**Computer Vision**

- ResNet-50 tutorial [html] [notebook]
- PyTorch YOLOv4 tutorial [html] [notebook]

**ResNet50 model for Inferentia**

**Introduction:**

In this tutorial we will compile and deploy a ResNet50 model for inference on Inferentia. This Jupyter notebook should run on an inf1.6xlarge instance. The inference part of this tutorial requires an inf1 instance, not the compilation stage. For simplicity we will run this tutorial on an inf1.6xlarge, but in real life scenarios the compilation should be done on a compute instance and the deployment on an inf1 instance to save costs.

In this tutorial we provide three main sections:

1. Compile the ResNet50 model and infer with a batch size of 1
2. Run the same compiled model on multiple NeuronCores using `torch.neuron.DataParallel` and dynamic batching
3. Compile the ResNet50 model with a batch size of 5 and run it on multiple NeuronCores using `torch.neuron.DataParallel` for optimal performance on Inferentia

Before running the following, verify that this Jupyter notebook is running the `conda_aws_neuron_pytorch_p36` kernel. You can select the kernel from the “Kernel -> Change Kernel” option on the top of this Jupyter notebook page.

**Install Dependencies:**

This tutorial requires the following pip packages:

- `torch>=1.8`
- `torch-neuron`
- `torchvision`
- `neuron-cc[tensorflow]`
These will be installed by default when configuring your environment using the Neuron PyTorch setup guide.

**Compile model for Neuron**

The following step will compile the ResNet50 model for Inferentia. This will take a few minutes. At the end of script execution, the compiled model is saved as `resnet50_neuron.pt` in your local directory.

```python
import torch
from torchvision import models, transforms, datasets
import torch_neuron

# Create an example input for compilation
image = torch.zeros([1, 3, 224, 224], dtype=torch.float32)

# Load a pretrained ResNet50 model
model = models.resnet50(pretrained=True)

# Tell the model we are using it for evaluation (not training)
model.eval()

# Analyze the model - this will show operator support and operator count
torch.neuron.analyze_model(model, example_inputs=[image])

# Compile the model using torch.neuron.trace to create a Neuron model
# that that is optimized for the Inferentia hardware
model_neuron = torch.neuron.trace(model, example_inputs=[image])

# The output of the compilation step will report the percentage of operators
# that
# are compiled to Neuron, for example:
# INFO:Neuron:Neuron successfully compiled 1 sub-graphs, Total fused
# subgraphs = 1, Percent of model sub-graphs successfully compiled = 100.0%
# INFO:Neuron:The neuron partitioner created 1 sub-graphs
# INFO:Neuron:Neuron successfully compiled 1 sub-graphs, Total fused
# subgraphs = 1, Percent of model sub-graphs successfully compiled = 100.0%

# We will also be warned if there are operators that are not placed on the
# Inferentia hardware

# Save the compiled model
model_neuron.save("resnet50_neuron.pt")
```

**Run inference on Inferentia**

We can use the compiled Neuron model to run inference on Inferentia.

In the following example, we preprocess a sample image for inference using the CPU model and Neuron model. We compare the predicted labels from the CPU model and Neuron model to verify that they are the same.

Important: Do not perform inference with a Neuron traced model on a non-Neuron supported instance, as the results will not be calculated properly.
Define a preprocessing function

We define a basic image preprocessing function that loads a sample image and labels, normalizes and batches the image, and transforms the image into a tensor for inference using the compiled Neuron model.

```python
import json
import os
from urllib import request

# Create an image directory containing a sample image of a small kitten
os.makedirs("./torch_neuron_test/images", exist_ok=True)
"./torch_neuron_test/images/kitten_small.jpg")

# Fetch labels to output the top classifications
request.urlretrieve("https://s3.amazonaws.com/deep-learning-models/image-models/imagenet_class_index.json", "imagenet_class_index.json")
idx2label = []

# Read the labels and create a list to hold them for classification
with open("imagenet_class_index.json", "r") as read_file:
    class_idx = json.load(read_file)
    idx2label = [class_idx[str(k)][1] for k in range(len(class_idx))]

def preprocess(batch_size=1, num_neuron_cores=1):
    # Define a normalization function using the ImageNet mean and standard deviation
    normalize = transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225])

    # Resize the sample image to [1, 3, 224, 224], normalize it, and turn it into a tensor
    eval_dataset = datasets.ImageFolder(os.path.dirname("./torch_neuron_test/"),
        transforms.Compose([
            transforms.Resize([224, 224]),
            transforms.ToTensor(),
            normalize,
        ])
    )
    image, _ = eval_dataset[0]
    image = torch.tensor(image.numpy()[:, np.newaxis, ...])

    # Create a "batched" image with enough images to go on each of the available NeuronCores
    # batch_size is the per-core batch size
    # num_neuron_cores is the number of NeuronCores being used
    batch_image = image
    for i in range(batch_size * num_neuron_cores - 1):
        batch_image = torch.cat([batch_image, image], 0)

    return batch_image
```

2.2. User Guide
Run inference using the Neuron model

We import the necessary python modules, load the torch-neuron compiled model, and run inference on Inferentia.

By default, the Neuron model will run on a single NeuronCore. In the next section, we will see how to run the Neuron model on multiple NeuronCores to fully saturate our hardware for optimal performance on Inferentia.

```python
# Get a sample image
image = preprocess()

# Run inference using the CPU model
output_cpu = model(image)

# Load the compiled Neuron model
model_neuron = torch.jit.load('resnet50_neuron.pt')

# Run inference using the Neuron model
output_neuron = model_neuron(image)

# Verify that the CPU and Neuron predictions are the same by comparing the top-5 results
top5_cpu = output_cpu[0].sort()[1][-5:]
top5_neuron = output_neuron[0].sort()[1][-5:]

# Lookup and print the top-5 labels
idx2label = ...
top5_labels_cpu = [idx2label[idx] for idx in top5_cpu]
top5_labels_neuron = [idx2label[idx] for idx in top5_neuron]
print("CPU top-5 labels: {}\n      ").format(top5_labels_cpu))
print("Neuron top-5 labels: {}\n      ").format(top5_labels_neuron))
```

Run Inference using torch.neuron.DataParallel

To fully leverage the Inferentia hardware we want to use all available NeuronCores. An inf1.xlarge and inf1.2xlarge have four NeuronCores, an inf1.6xlarge has 16 NeuronCores, and an inf1.24xlarge has 64 NeuronCores. For maximum performance on Inferentia hardware, we can use `torch.neuron.DataParallel` to utilize all available NeuronCores.

`torch.neuron.DataParallel` implements data parallelism at the module level by duplicating the Neuron model on all available NeuronCores and distributing data across the different cores for parallelized inference.

In the following section, we will run inference using the `torch.neuron.DataParallel` module to fully saturate the Inferentia hardware. We benchmark the model to collect throughput and latency statistics.

Note: `torch.neuron.DataParallel` is new with Neuron 1.16.0. Please ensure you are using the latest Neuron package to run the following sections.
Define a benchmarking function

We create a function that handles benchmarking the Neuron model to collect throughput and latency metrics.

```python
def benchmark(model, image):
    print('Input image shape is {}'.format(list(image.shape)))

    # The first inference loads the model so exclude it from timing
    results = model(image)

    # Collect throughput and latency metrics
    latency = []
    throughput = []

    # Run inference for 100 iterations and calculate metrics
    num_infers = 100
    for _ in range(num_infers):
        delta_start = time()
        results = model(image)
        delta = time() - delta_start
        latency.append(delta)
        throughput.append(image.size(0)/delta)

    # Calculate and print the model throughput and latency
    print("Avg. Throughput: {:.0f}, Max Throughput: {:.0f}".
          format(np.mean(throughput), np.max(throughput)))
    print("Latency P50: {:.0f}".format(np.percentile(latency, 50)*1000.0))
    print("Latency P90: {:.0f}".format(np.percentile(latency, 90)*1000.0))
    print("Latency P95: {:.0f}".format(np.percentile(latency, 95)*1000.0))
    print("Latency P99: {:.0f}".format(np.percentile(latency, 99)*1000.0))
```

Run Inference using torch.neuron.DataParallel

We create the `torch.neuron.DataParallel` module using the compiled Neuron model, get a sample image, and benchmark the parallelized model on Neuron.

```python
# Create a torch.neuron.DataParallel module using the compiled Neuron model
# By default, torch.neuron.DataParallel will use four cores on an inf1.xlarge
# or inf1.2xlarge, 16 cores on an inf1.6xlarge, and 24 cores on an inf1.24xlarge
model_neuron_parallel = torch.neuron.DataParallel(model_neuron)

# Get sample image with batch size=1 per NeuronCore
batch_size = 1

# For an inf1.xlarge or inf1.2xlarge, set num_neuron_cores = 4
# num_neuron_cores = 16
image = preprocess(batch_size=batch_size, num_neuron_cores=num_neuron_cores)

# Benchmark the model
benchmark(model_neuron_parallel, image)
```
Run inference with dynamic batch sizes

Batch size has a direct impact on model performance. The Inferentia chip is optimized to run with small batch sizes. This means that a Neuron compiled model can outperform a GPU model, even if running single digit batch sizes.

As a general best practice, we recommend optimizing your model’s throughput by compiling the model with a small batch size and gradually increasing it to find the peak throughput on Inferentia.

Dynamic batching is a feature that allows you to use tensor batch sizes that the Neuron model was not originally compiled against. This is necessary because the underlying Inferentia hardware will always execute inferences with the batch size used during compilation. Fixed batch size execution allows tuning the input batch size for optimal performance. For example, batch size 1 may be best suited for an ultra-low latency on-demand inference application, while batch size > 1 can be used to maximize throughput for offline inferencing. Dynamic batching is implemented by slicing large input tensors into chunks that match the batch size used during the `torch.neuron.trace` compilation call.

The `torch.neuron.DataParallel` class automatically enables dynamic batching on eligible models. This allows us to run inference in applications that have inputs with a variable batch size without needing to recompile the model.

In the following example, we use the same `torch.neuron.DataParallel` module to run inference using several different batch sizes. Notice that latency increases consistently as the batch size increases. Throughput increases as well, up until a certain point where the input size becomes too large to be efficient.

```python
batch_sizes = [2, 3, 4, 5, 6, 7]
for batch_size in batch_sizes:
    image = preprocess(batch_size=batch_size, num_neuron_cores=num_neuron_cores)
    # Benchmark the model for each input batch size
    benchmark(model_neuron_parallel, image)
```

Compile and Infer with different batch sizes on multiple NeuronCores

Dynamic batching using small batch sizes can result in sub-optimal throughput because it involves slicing tensors into chunks and iteratively sending data to the hardware. Using a larger batch size at compilation time can use the Inferentia hardware more efficiently in order to maximize throughput. You can test the tradeoff between individual request latency and total throughput by fine-tuning the input batch size.

In the following example, we recompile our model using a batch size of 5 and run the model using `torch.neuron.DataParallel` to fully saturate our Inferentia hardware for optimal performance.

```python
# Create an input with batch size 5 for compilation
batch_size = 5
image = torch.zeros([batch_size, 3, 224, 224], dtype=torch.float32)

# Recompile the ResNet50 model for inference with batch size 5
model_neuron = torch.neuron.trace(model, example_inputs=[image])

# Export to saved model
model_neuron.save("resnet50_neuron_b{}\(\)\).pt\(\)\).format(batch_size))

Run inference with batch size of 5 using the Neuron model compiled for a batch size of 5.
You can experiment with different batch size values to see what gives the best overall throughput on Inferentia.

**Evaluate YOLO v4 on Inferentia**

**Introduction**

This tutorial walks through compiling and evaluating YOLO v4 model implemented in PyTorch on Inferentia.

The tutorial has five main sections:

1. Define YOLO v4 model in PyTorch
2. Download the COCO 2017 evaluation dataset and define the data loader function
3. Build, Compile, and Save Neuron-Optimized YOLO v4 TorchScript
4. Evaluate Accuracy on the COCO 2017 Dataset
5. Benchmark COCO Dataset Performance of the Neuron-Optimized TorchScript

Before running the following verify this Jupyter notebook is running “conda_aws_neuron_pytorch_p36” kernel. You can select the Kernel from the “Kernel -> Change Kernel” option on the top of this Jupyter notebook page.

**Install Dependencies:**

This tutorial requires the following pip packages:

- torch-neuron
- torchvision
- pillow
- pycocotools
- neuron-cc[tensorflow]

Many of these packages will be installed by default when configuring your environment using the Neuron PyTorch setup guide. The additional dependencies must be installed here.

```bash
pip install --upgrade pillow pycocotools
```
Part 1: Define YOLO v4 model in PyTorch

The following PyTorch model definition is from https://github.com/Tianxiaomo/pytorch-YOLOv4/.

```python
import numpy as np
import torch
import torch.neuron
from torch import nn
import torch.nn.functional as F
import os
import warnings

# Setting up NeuronCore groups for inf1.6xlarge with 16 cores
nc_env = ', '.join(['1'] * n_cores)
warnings.warn("NEURONCORE_GROUP_SIZES is being deprecated, if your application is using NEURONCORE_GROUP_SIZES please see https://awsdocs-neuron.readthedocs-hosted.com/en/latest/release-notes/deprecation.html#announcing-end-of-support-for-neuroncore-group-sizes for more details.", DeprecationWarning)
os.environ['NEURONCORE_GROUP_SIZES'] = nc_env

class Mish(nn.Module):
    def __init__(self):
        super().__init__()

    def forward(self, x):
        x = x * (torch.tanh(torch.nn.functional.softplus(x)))

        return x

class Upsample(nn.Module):
    def __init__(self):
        super(Upsample, self).__init__()

    def forward(self, x, target_size, inference=False):
        assert (x.data.dim() == 4)

        if inference:

            return x.view(x.size(0), x.size(1), x.size(2), 1, x.size(3), 1) *
                  expand(x.size(0), x.size(1), x.size(2), target_size[2] //
                         x.size(2), x.size(3), target_size[3] // x.size(3)) *
                  contiguous().view(x.size(0), x.size(1), target_size[2],
                                     target_size[3])
            else:
                return F.interpolate(x, size=(target_size[2], target_size[3]),
                                     mode='nearest')

class Conv_Bn_Activation(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, stride, activation, bn=True, bias=False):
        super().__init__()
        pad = (kernel_size - 1) // 2
```

(continues on next page)
self.conv = nn.ModuleList()
    if bias:
        self.conv.append(nn.Conv2d(in_channels, out_channels, kernel_
˓→size, stride, pad))
    else:
        self.conv.append(nn.Conv2d(in_channels, out_channels, kernel_
˓→size, stride, pad, bias=False))
    if bn:
        self.conv.append(nn.BatchNorm2d(out_channels))
    if activation == "mish":  
        self.conv.append(Mish())
    elif activation == "relu":  
        self.conv.append(nn.ReLU(inplace=True))
    elif activation == "leaky":  
        self.conv.append(nn.LeakyReLU(0.1, inplace=True))
    elif activation == "linear":
        pass
    else:
        print("activate error !!! {} {} {}",format(sys._getframe().f_
˓→code.co_filename, sys._getframe().f_
˓→code.co_name, sys._getframe().f_lineno))

def forward(self, x):
    for l in self.conv:
        x = l(x)
    return x

class ResBlock(nn.Module):
    ""
    Sequential residual blocks each of which consists of two convolution layers.
    Args:
        ch (int): number of input and output channels.
        nblocks (int): number of residual blocks.
        shortcut (bool): if True, residual tensor addition is enabled.
    ""
    def __init__(self, ch, nblocks=1, shortcut=True):
        super().__init__()
        self.shortcut = shortcut
        self.module_list = nn.ModuleList()
        for i in range(nblocks):
            resblock_one = nn.ModuleList()
            resblock_one.append(Conv_Bn_Activation(ch, ch, 1, 1, 'mish'))
            resblock_one.append(Conv_Bn_Activation(ch, ch, 3, 1, 'mish'))
            self.module_list.append(resblock_one)

def forward(self, x):
    for module in self.module_list:
        h = x
        for res in module:
            h = res(h)
        x = x + h if self.shortcut else h
    return x
class DownSample1(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = Conv_Bn_Activation(3, 32, 3, 1, 'mish')
        self.conv2 = Conv_Bn_Activation(32, 64, 3, 2, 'mish')
        self.conv3 = Conv_Bn_Activation(64, 64, 1, 1, 'mish')
        # [route]
        # layers = -2
        self.conv4 = Conv_Bn_Activation(64, 64, 1, 1, 'mish')
        self.conv5 = Conv_Bn_Activation(64, 32, 1, 1, 'mish')
        self.conv6 = Conv_Bn_Activation(32, 64, 3, 1, 'mish')
        # [shortcut]
        # from=-3
        # activation = linear
        self.conv7 = Conv_Bn_Activation(64, 64, 1, 1, 'mish')
        # [route]
        # layers = -1, -7
        self.conv8 = Conv_Bn_Activation(128, 64, 1, 1, 'mish')

    def forward(self, input):
        x1 = self.conv1(input)
        x2 = self.conv2(x1)
        x3 = self.conv3(x2)
        # route -2
        x4 = self.conv4(x2)
        x5 = self.conv5(x4)
        x6 = self.conv6(x5)
        # shortcut -3
        x6 = x6 + x4
        x7 = self.conv7(x6)
        # [route]
        # layers = -1, -7
        x7 = torch.cat([x7, x3], dim=1)
        x8 = self.conv8(x7)
        return x8

class DownSample2(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = Conv_Bn_Activation(64, 128, 3, 2, 'mish')
        self.conv2 = Conv_Bn_Activation(128, 64, 1, 1, 'mish')
        # r -2
        self.conv3 = Conv_Bn_Activation(128, 64, 1, 1, 'mish')
        self.conv4 = Conv_Bn_Activation(64, 128, 3, 2, 'mish')
        # r -2
        self.conv5 = Conv_Bn_Activation(128, 128, 1, 1, 'mish')

        self.resblock = ResBlock(ch=64, nblocks=2)

        # s -3
        self.conv4 = Conv_Bn_Activation(64, 128, 3, 1, 'mish')
        # r -1 -10
        self.conv5 = Conv_Bn_Activation(128, 128, 1, 1, 'mish')

(continues on next page)
def forward(self, input):
    x1 = self.conv1(input)
    x2 = self.conv2(x1)
    x3 = self.conv3(x1)

    r = self.resblock(x3)
    x4 = self.conv4(r)

    x4 = torch.cat([x4, x2], dim=1)
    x5 = self.conv5(x4)
    return x5

class DownSample3(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = Conv_Bn_Activation(128, 256, 3, 2, 'mish')
        self.conv2 = Conv_Bn_Activation(256, 128, 1, 1, 'mish')
        self.conv3 = Conv_Bn_Activation(256, 128, 1, 1, 'mish')

        self.resblock = ResBlock(ch=128, nblocks=8)
        self.conv4 = Conv_Bn_Activation(128, 256, 1, 1, 'mish')
        self.conv5 = Conv_Bn_Activation(256, 256, 1, 1, 'mish')

    def forward(self, input):
        x1 = self.conv1(input)
        x2 = self.conv2(x1)
        x3 = self.conv3(x1)

        r = self.resblock(x3)
        x4 = self.conv4(r)

        x4 = torch.cat([x4, x2], dim=1)
        x5 = self.conv5(x4)
        return x5

class DownSample4(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = Conv_Bn_Activation(256, 512, 3, 2, 'mish')
        self.conv2 = Conv_Bn_Activation(512, 256, 1, 1, 'mish')
        self.conv3 = Conv_Bn_Activation(512, 256, 1, 1, 'mish')

        self.resblock = ResBlock(ch=256, nblocks=8)
        self.conv4 = Conv_Bn_Activation(256, 512, 1, 1, 'mish')
        self.conv5 = Conv_Bn_Activation(512, 512, 1, 1, 'mish')

    def forward(self, input):
        x1 = self.conv1(input)
        x2 = self.conv2(x1)
        x3 = self.conv3(x1)

        r = self.resblock(x3)
        x4 = self.conv4(r)

        x4 = torch.cat([x4, x2], dim=1)
        (continues on next page)
x5 = self.conv5(x4)
return x5

class DownSample5(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = Conv_Bn_Activation(512, 1024, 3, 2, 'mish')
        self.conv2 = Conv_Bn_Activation(1024, 512, 1, 1, 'mish')
        self.conv3 = Conv_Bn_Activation(1024, 512, 1, 1, 'mish')
        self.resblock = ResBlock(ch=512, nblocks=4)
        self.conv4 = Conv_Bn_Activation(512, 512, 1, 1, 'mish')
        self.conv5 = Conv_Bn_Activation(1024, 1024, 1, 1, 'mish')

    def forward(self, input):
        x1 = self.conv1(input)
        x2 = self.conv2(x1)
        x3 = self.conv3(x1)
        r = self.resblock(x3)
        x4 = self.conv4(r)
        x4 = torch.cat([x4, x2], dim=1)
        x5 = self.conv5(x4)
        return x5

class Neck(nn.Module):
    def __init__(self, inference=False):
        super().__init__()
        self.inference = inference
        self.conv1 = Conv_Bn_Activation(1024, 512, 1, 1, 'leaky')
        self.conv2 = Conv_Bn_Activation(512, 1024, 3, 1, 'leaky')
        self.conv3 = Conv_Bn_Activation(1024, 512, 1, 1, 'leaky')
        self.conv4 = Conv_Bn_Activation(2048, 512, 1, 1, 'leaky')
        self.conv5 = Conv_Bn_Activation(512, 1024, 3, 1, 'leaky')
        self.conv6 = Conv_Bn_Activation(1024, 512, 1, 1, 'leaky')
        self.conv7 = Conv_Bn_Activation(512, 256, 1, 1, 'leaky')
        self.upsample1 = Upsample()

    def forward(self, input):
        x1 = self.conv1(input)
        x2 = self.conv2(x1)
        x3 = self.conv3(x1)
        r = self.conv4(x3)
        x4 = self.conv5(r)
        x4 = torch.cat([x4, x2], dim=1)
        x5 = self.conv6(x4)
        return x5
self.conv13 = Conv_Bn_Activation(512, 256, 1, 1, 'leaky')
self.conv14 = Conv_Bn_Activation(256, 128, 1, 1, 'leaky')
# UP
self.upsample2 = Upsample()
# R 54
self.conv15 = Conv_Bn_Activation(256, 128, 1, 1, 'leaky')
# R -1 -3
self.conv16 = Conv_Bn_Activation(256, 128, 1, 1, 'leaky')
self.conv17 = Conv_Bn_Activation(128, 256, 3, 1, 'leaky')
self.conv18 = Conv_Bn_Activation(256, 128, 1, 1, 'leaky')
self.conv19 = Conv_Bn_Activation(128, 256, 3, 1, 'leaky')
self.conv20 = Conv_Bn_Activation(256, 128, 1, 1, 'leaky')

```python
def forward(self, input, downsample4, downsample3, inference=False):
x1 = self.conv1(input)
x2 = self.conv2(x1)
x3 = self.conv3(x2)
# SPP
m1 = self.maxpool1(x3)
m2 = self.maxpool2(x3)
m3 = self.maxpool3(x3)
spp = torch.cat([m3, m2, m1, x3], dim=1)
# SPP end
x4 = self.conv4(spp)
x5 = self.conv5(x4)
x6 = self.conv6(x5)
x7 = self.conv7(x6)
# UP
up = self.upsample1(x7, downsample4.size(), self.inference)
# R 85
x8 = self.conv8(downsample4)
# R -1 -3
x8 = torch.cat([x8, up], dim=1)
x9 = self.conv9(x8)
x10 = self.conv10(x9)
x11 = self.conv11(x10)
x12 = self.conv12(x11)
x13 = self.conv13(x12)
x14 = self.conv14(x13)
# UP
up = self.upsample2(x14, downsample3.size(), self.inference)
# R 54
x15 = self.conv15(downsample3)
# R -1 -3
x15 = torch.cat([x15, up], dim=1)
x16 = self.conv16(x15)
x17 = self.conv17(x16)
x18 = self.conv18(x17)
x19 = self.conv19(x18)
x20 = self.conv20(x19)
return x20, x13, x6
```

class Yolov4Head(nn.Module):

(continues on next page)
def __init__(self, output_ch, n_classes, inference=False):
    super().__init__()
    self.inference = inference

    self.conv1 = Conv_Bn_Activation(128, 256, 3, 1, 'leaky')
    self.conv2 = Conv_Bn_Activation(256, output_ch, 1, 1, 'linear',
    bn=False, bias=True)

    self.yolo1 = YoloLayer(
        anchor_mask=[0, 1, 2], num_classes=n_classes,
        anchors=[12, 16, 19, 36, 40, 28, 36, 75, 76,
        55, 72, 146, 142, 110, 192, 243, 459, 401],
        num_anchors=9, stride=8)

    # R -4
    self.conv3 = Conv_Bn_Activation(128, 256, 3, 2, 'leaky')

    # R -1 -16
    self.conv4 = Conv_Bn_Activation(512, 256, 1, 1, 'leaky')
    self.conv5 = Conv_Bn_Activation(256, 512, 3, 1, 'leaky')
    self.conv6 = Conv_Bn_Activation(512, 256, 1, 1, 'leaky')
    self.conv7 = Conv_Bn_Activation(256, 512, 3, 1, 'leaky')
    self.conv8 = Conv_Bn_Activation(512, 256, 1, 1, 'leaky')
    self.conv9 = Conv_Bn_Activation(256, 512, 3, 1, 'leaky')
    self.conv10 = Conv_Bn_Activation(512, output_ch, 1, 1, 'linear',
    bn=False, bias=True)

    self.yolo2 = YoloLayer(
        anchor_mask=[3, 4, 5], num_classes=n_classes,
        anchors=[12, 16, 19, 36, 40, 28, 36, 75, 76,
        55, 72, 146, 142, 110, 192, 243, 459, 401],
        num_anchors=9, stride=16)

    # R -4
    self.conv11 = Conv_Bn_Activation(256, 512, 3, 2, 'leaky')

    # R -1 -37
    self.conv12 = Conv_Bn_Activation(1024, 512, 1, 1, 'leaky')
    self.conv13 = Conv_Bn_Activation(512, 1024, 3, 1, 'leaky')
    self.conv14 = Conv_Bn_Activation(1024, 512, 1, 1, 'leaky')
    self.conv15 = Conv_Bn_Activation(512, 1024, 3, 1, 'leaky')
    self.conv16 = Conv_Bn_Activation(1024, 512, 1, 1, 'leaky')
    self.conv17 = Conv_Bn_Activation(512, 1024, 3, 1, 'leaky')
    self.conv18 = Conv_Bn_Activation(1024, output_ch, 1, 1, 'linear',
    bn=False, bias=True)

    self.yolo3 = YoloLayer(
        anchor_mask=[6, 7, 8], num_classes=n_classes,
        anchors=[12, 16, 19, 36, 40, 28, 36, 75, 76,
        55, 72, 146, 142, 110, 192, 243, 459, 401],
        num_anchors=9, stride=32)

def forward(self, input1, input2, input3):
    x1 = self.conv1(input1)
    x2 = self.conv2(x1)
    x3 = self.conv3(input1)
x3 = torch.cat([x3, input2], dim=1)
x4 = self.conv4(x3)
x5 = self.conv5(x4)
x6 = self.conv6(x5)
x7 = self.conv7(x6)
x8 = self.conv8(x7)
x9 = self.conv9(x8)
x10 = self.conv10(x9)

# R -4
x11 = self.conv11(x8)
# R -1 -37
x11 = torch.cat([x11, input3], dim=1)
x12 = self.conv12(x11)
x13 = self.conv13(x12)
x14 = self.conv14(x13)
x15 = self.conv15(x14)
x16 = self.conv16(x15)
x17 = self.conv17(x16)
x18 = self.conv18(x17)

if self.inference:
    y1 = self.yolo1(x2)
y2 = self.yolo2(x10)
y3 = self.yolo3(x18)

    return get_region_boxes([y1, y2, y3])
else:
    return [x2, x10, x18]

class Yolov4(nn.Module):
    def __init__(self, yolov4conv137weight=None, n_classes=80, inference=False):
        super().__init__()
        output_ch = (4 + 1 + n_classes) * 3

        # backbone
        self.down1 = DownSample1()
        self.down2 = DownSample2()
        self.down3 = DownSample3()
        self.down4 = DownSample4()
        self.down5 = DownSample5()

        # neck
        self.neek = Neck(inference)

        # yolov4conv137
        if yolov4conv137weight:
            _model = nn.Sequential(self.down1, self.down2, self.down3, self.
            down4, self.down5, self.neek)
            pretrained_dict = torch.load(yolov4conv137weight)

            model_dict = _model.state_dict()
            # 1. filter out unnecessary keys
pretrained_dict = {k1: v for (k, v), k1 in zip(pretrained_dict.items(), model_dict.items())}

# 2. overwrite entries in the existing state dict
model_dict.update(pretrained_dict)
_model.load_state_dict(model_dict)

# head
self.head = Yolov4Head(output_ch, n_classes, inference)

def forward(self, input):
    d1 = self.down1(input)
d2 = self.down2(d1)d3 = self.down3(d2)d4 = self.down4(d3)d5 = self.down5(d4)

    x20, x13, x6 = self.neek(d5, d4, d3)

    output = self.head(x20, x13, x6)
    return output

def yolo_forward_dynamic(output, conf_thresh, num_classes, anchors, num_anchors, scale_x_y, only_objectness=1, validation=False):
    # Output would be invalid if it does not satisfy this assert
    # assert (output.size(1) == (5 + num_classes) * num_anchors)
    # print(output.size())

    # Slice the second dimension (channel) of output into:
    # [ 2, 2, 1, num_classes, 2, 2, 1, num_classes, 2, 2, 1, num_classes ]
    # And then into
    # batch = output.size(0)
    # H = output.size(2)
    # W = output.size(3)

    bxy_list = []bwh_list = []det_confs_list = []cls_confs_list = []

    for i in range(num_anchors):
        begin = i * (5 + num_classes)end = (i + 1) * (5 + num_classes)

        bxy_list.append(output[:, begin : begin + 2])bwh_list.append(output[:, begin + 2 : begin + 4])det_confs_list.append(output[:, begin + 4 : begin + 5])cls_confs_list.append(output[:, begin + 5 : end])

    # Shape: [batch, num_anchors * 2, H, W]
    bxy = torch.cat(bxy_list, dim=1)
    # Shape: [batch, num_anchors * 2, H, W]
    bwh = torch.cat(bwh_list, dim=1)
# Shape: [batch, num_anchors, H, W]

det_confs = torch.cat(det_confs_list, dim=1)

# Shape: [batch, num_anchors * H * W]

det_confs = det_confs.view(output.size(0), num_anchors * output.size(2) * output.size(3))

# Shape: [batch, num_anchors * num_classes, H, W]

cls_confs = torch.cat(cls_confs_list, dim=1)

# Shape: [batch, num_anchors, num_classes, H * W]

cls_confs = cls_confs.view(output.size(0), num_anchors, num_classes, output.size(2) * output.size(3))

# Shape: [batch, num_anchors, num_classes, H * W] --> [batch, num_anchors * H * W, num_classes]

cls_confs = cls_confs.permute(0, 1, 3, 2).reshape(output.size(0), num_anchors * output.size(2) * output.size(3), num_classes)

# Apply sigmoid(), exp() and softmax() to slices

bxy = torch.sigmoid(bxy) * scale_x_y - 0.5 * (scale_x_y - 1)
bwh = torch.exp(bwh)
det_confs = torch.sigmoid(det_confs)

cls_confs = torch.sigmoid(cls_confs)

anchor_w = []
anchor_h = []

for i in range(num_anchors):
    anchor_w.append(anchors[i * 2])
    anchor_h.append(anchors[i * 2 + 1])

device = None
cuda_check = output.is_cuda

if cuda_check:
    device = output.get_device()

bx_list = []
by_list = []
bw_list = []
bb_list = []

# Apply C-x, C-y, P-w, P-h

for i in range(num_anchors):
    ii = i * 2
    # Shape: [batch, 1, H, W]
    bx = bxy[:, ii : ii + 1] + torch.tensor(grid_x, device=device, dtype=torch.float32)
# Shape: \([\text{batch}, 1, H, W]\)
by = bxy[:, ii + 1 : ii + 2] + torch.tensor(grid_y, device=device, dtype=torch.float32)  # grid_y.to(device=device, dtype=torch.float32)
# Shape: \([\text{batch}, 1, H, W]\)
bw = bwh[:, ii : ii + 1] * anchor_w[i]
# Shape: \([\text{batch}, 1, H, W]\)
bh = bwh[:, ii + 1 : ii + 2] * anchor_h[i]

bx_list.append(bx)
by_list.append(by)
bw_list.append(bw)
bh_list.append(bh)

# Figure out bboxes from slices

# Shape: \([\text{batch}, \text{num_anchors}, H, W]\)
bx = torch.cat(bx_list, dim=1)
# Shape: \([\text{batch}, \text{num_anchors}, H, W]\)
by = torch.cat(by_list, dim=1)
# Shape: \([\text{batch}, \text{num_anchors}, H, W]\)
bw = torch.cat(bw_list, dim=1)
# Shape: \([\text{batch}, \text{num_anchors}, H, W]\)
bh = torch.cat(bh_list, dim=1)

# Shape: \([\text{batch}, \text{2 * num_anchors}, H, W]\)
bx_bw = torch.cat((bx, bw), dim=1)
# Shape: \([\text{batch}, \text{2 * num_anchors}, H, W]\)
by_bh = torch.cat((by, bh), dim=1)

# normalize coordinates to \([0, 1]\)
bx_bw /= output.size(3)
by_bh /= output.size(2)

# Shape: \([\text{batch}, \text{num_anchors * H * W}, 1]\)
bx = bx_bw[:, :num_anchors].view(output.size(0), num_anchors * output.size(2) * output.size(3), 1)
by = by_bh[:, :num_anchors].view(output.size(0), num_anchors * output.size(2) * output.size(3), 1)
bw = bx_bw[:, num_anchors:].view(output.size(0), num_anchors * output.size(2) * output.size(3), 1)
bh = by_bh[:, num_anchors:].view(output.size(0), num_anchors * output.size(2) * output.size(3), 1)

bx1 = bx - bw * 0.5
by1 = by - bh * 0.5
bx2 = bx1 + bw
by2 = by1 + bh

# Shape: \([\text{batch}, \text{num_anchors * h * w}, 4]\) -> \([\text{batch}, \text{num_anchors * h * w}, 4]\)
boxes = torch.cat((bx1, by1, bx2, by2), dim=2).view(output.size(0), num_anchors * output.size(2) * output.size(3), 1, 4)
# boxes = boxes.repeat(1, 1, num_classes, 1)

(continues on next page)
# boxes: [batch, num_anchors * H * W, 1, 4]
# cls_confs: [batch, num_anchors * H * W, num_classes]
# det_confs: [batch, num_anchors * H * W]

det_confs = det_confs.view(output.size(0), num_anchors * output.size(2) -
> output.size(3), 1)
confs = cls_confs * det_confs

# boxes: [batch, num_anchors * H * W, 1, 4]
# confs: [batch, num_anchors * H * W, num_classes]
return boxes, confs

class YoloLayer(nn.Module):
    
    """Yolo layer
    model_out: while inference, is post-processing inside or outside the model
    true:outside
    """
    def __init__(self, anchor_mask=[], num_classes=0, anchors=[], num_-
> anchors=1, stride=32, model_out=False):
        super(YoloLayer, self).__init__()
        self.anchor_mask = anchor_mask
        self.num_classes = num_classes
        self.anchors = anchors
        self.num_anchors = num_anchors
        self.anchor_step = len(anchors) // num_anchors
        self.coord_scale = 1
        self.noobject_scale = 1
        self.object_scale = 5
        self.class_scale = 1
        self.thresh = 0.6
        self.stride = stride
        self.seen = 0
        self.scale_x_y = 1

        self.model_out = model_out

        def forward(self, output, target=None):
            if self.training:
                return output
            masked_anchors = []
            for m in self.anchor_mask:
                masked_anchors += self.anchors[m * self.anchor_step:(m + 1) * self.anchor_step]
            masked_anchors = [anchor / self.stride for anchor in masked_anchors]

            return yolo_forward_dynamic(output, self.thresh, self.num_classes, -
> masked_anchors, len(self.anchor_mask), scale_x_y=self.scale_x_y)

    def get_region_boxes(boxes_and_confs):
        # print('Getting boxes from boxes and confs ...')
        boxes_list = []
        confs_list = []

(continues on next page)
for item in boxes_and_confs:
    boxes_list.append(item[0])
    confs_list.append(item[1])

# boxes: [batch, num1 + num2 + num3, 1, 4]
# confs: [batch, num1 + num2 + num3, num_classes]
boxes = torch.cat(boxes_list, dim=1)
confs = torch.cat(confs_list, dim=1)

return boxes, confs

Part 2: Download the COCO 2017 evaluation dataset and define the data loader function

Download dataset

[:]
!curl -LO http://images.cocodataset.org/annotations/annotations_trainval2017._zip
!unzip -q val2017.zip
!unzip annotations_trainval2017.zip

[:]
!ls

Define data loader

[:]
import os
import json
import time
import torchvision
import torchvision.transforms as transforms
import torchvision.datasets as dset
from pycocotools.coco import COCO

def get_image_filenames(root=os.getcwd()):
    """
    Generate paths to the coco dataset image files.
    
    Args:
    root (str): The root folder contains.
    
    Yields:
    filename (str): The path to an image file.
    """
    image_path = os.path.join(root, 'val2017')
    for root, dirs, files in os.walk(image_path):
        for filename in files:
            yield os.path.join(image_path, filename)
def get_coco_dataloader(coco2017_root, transform, subset_indices=None):
    """Create the dataset loader and ground truth coco dataset.

    Arguments:
    coco2017_root (str): The root directory to load the data/labels from.
    transform (torchvision.Transform): A transform to apply to the
    images.
    subset_indices (list): Indices used to create a subset of the
dataset.

    Returns:
    loader (iterable): Produces transformed images and labels.
    cocoGt (pycocotools.coco.COCO): Contains the ground truth in coco
    format.
    label_info (dict): A mapping from label id to the human-readable
    name.
    """

    # Create the dataset
coco2017_img_path = os.path.join(coco2017_root, 'val2017')
coco2017_ann_path = os.path.join(coco2017_root, 'annotations/instances_val2017.json')

    # check the number of images in val2017 - Should be 5000
    num_files = len(list(get_image_filenames(coco2017_root)))
    print('
Number of images in val2017 = {}
'.format(num_files))

    # load annotations to decode classification results
    with open(coco2017_ann_path) as f:
        annotate_json = json.load(f)
    label_info = {label['id']: label['name']
                  for label in annotate_json['categories']}

    # initialize COCO ground truth dataset
cocoGt = COCO(coco2017_ann_path)

    # create the dataset using torchvision's coco detection dataset
coco_val_data = dset.CocoDetection(
    root=coco2017_img_path,
    annFile=coco2017_ann_path,
    transform=transform
    )

    if subset_indices is not None:
        # Create a smaller subset of the data for testing - e.g. to pinpoint
        # error at image 516
        coco_val_data = torch.utils.data.Subset(coco_val_data, subset_  
indices)

        # create the dataloader using torch dataloader
        loader = torch.utils.data.DataLoader(coco_val_data, batch_size=1,  
shuffle=False)

    return loader, cocoGt, label_info
Here 2 dataset loaders are created and the resulting data is displayed -
**orig_coco_val_data_loader**: Contains the original unmodified image
**coco_val_data_loader**: Contains images of a standardized size of 608x608 pixels

```python
[ ]: coco2017_root = './'
orig_coco_val_data_loader, _ = get_coco_dataloader(coco2017_root, )
    -- transforms.ToTensor())
    -- ToTensor()])
coco_val_data_loader, cocoGt, label_info = get_coco_dataloader(coco2017_root,
    -- transform)
image_orig, _ = next(iter(orig_coco_val_data_loader))
print(image_orig.shape)
image, image_info = next(iter(coco_val_data_loader))
image_id = image_info[0]["image_id"].item()
print(image.shape)
```

Define some helper functions for deployment (inference)

```python
[ ]: def postprocess(boxes, scores, score_threshold=0.05, iou_threshold=0.5):
    """
    Classifies and filters bounding boxes from Yolo V4 output.
    Performs classification, filtering, and non-maximum suppression to remove
    boxes that are irrelevant. The result is the filtered set of boxes, the
    associated label confidence score, and the predicted label.
    
    
    Args:
    scores (torch.Tensor): The categories scores for each box.
    score_threshold (float): Ignore boxes with scores below threshold.
    iou_threshold (float): Discards boxes with intersection above
    threshold.
    
    Returns:
    boxes (torch.Tensor): The filtered Yolo V4 bounding boxes.
    scores (torch.Tensor): The label score for each box.
    labels (torch.Tensor): The label for each box.
    """
    # shape: [n_batch, n_boxes, 1, 4] => [n_boxes, 4]  # Assumes n_batch
    -- size is 1
    boxes = boxes.squeeze()
    # shape: [n_batch, n_boxes, 80] => [n_boxes, 80]  # Assumes n_batch size
    -- is 1
    scores = scores.squeeze()
    # Classify each box according to the maximum category score
    score, column = torch.max(scores, dim=1)
```
(continues on next page)
# Filter out rows for scores which are below threshold
mask = score > score_threshold

# Filter model output data
boxes = boxes[mask]
score = score[mask]
idxs = column[mask]

# Perform non-max suppression on all categories at once. shape: [n_keep,]
keep = torchvision.ops.batched_nms(
    boxes=boxes,
    scores=score,
    idxs=idxs,
    iou_threshold=iou_threshold,
)

# The image category id associated with each column
categories = torch.tensor([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 27, 28, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 67, 70, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 84, 85, 86, 87, 88, 89, 90])

boxes = boxes[keep]  # shape: [n_keep, 4]
score = score[keep]  # shape: [n_keep,]
idxs = idxs[keep]
label = categories[idxs]  # shape: [n_keep,]

return boxes, score, label

def get_results_as_dict(boxes, scores, labels, image_orig):
    ""
    Transforms post-processed output into dictionary output.
    This translates the model coordinate bounding boxes (x1, y1, x2, y2)
    into a rectangular description (x, y, width, height) scaled to the
    original image size.
    Args:
    scores (torch.Tensor): The label score for each box.
    labels (torch.Tensor): The label for each box.
    image_orig (torch.Tensor): The image to scale the bounding boxes to.
    Returns:
    output (dict): The dictionary of rectangle bounding boxes.
    ""
    h_size, w_size = image_orig.shape[-2:]
x1 = boxes[:, 0] * w_size
y1 = boxes[:, 1] * h_size
(continues on next page)
\[ x_2 = \text{boxes}[:, 2] \times \text{w\_size} \]
\[ y_2 = \text{boxes}[:, 3] \times \text{h\_size} \]

width = \( x_2 - x_1 \)
height = \( y_2 - y_1 \)

boxes = \text{torch\_stack([x_l, y_l, width, height]).T}

return 

\{'boxes': boxes\_detach().\text{numpy()},
    'labels': labels\_detach().\text{numpy()},
    'scores': scores\_detach().\text{numpy()},
\}'

\[
\text{def prepare\_for\_coco\_detection(predictions):}
\]

\""\n\text{Convert dictionary model predictions into an expected COCO dataset format.}
\""\n
\text{Args:}

\text{predictions (dict): The list of box coordinates, scores, and labels.}

\text{Returns:}

\text{output (list[dict]): The list of bounding boxes.}

\""\n
coco_results = []

\text{for original\_id, prediction in predictions.items():}

\text{if len(prediction) == 0:}

\text{continue}

boxes = prediction["boxes"].\text{tolist()}
scores = prediction["scores"].\text{tolist()}
labels = prediction["labels"].\text{tolist()}

coco_results.extend(

\[
\{
    "image\_id": original\_id,
    "category\_id": labels[k],
    "bbox": box,
    "score": scores[k],
\}

\text{for k, box in enumerate(boxes)}

\}

\text{return coco_results} \]
Download pretrained checkpoint

```python
import requests

def download_file_from_google_drive(id, destination):
    URL = "https://docs.google.com/uc?export=download"
    session = requests.Session()
    response = session.get(URL, params={'id': id}, stream=True)
    token = get_confirm_token(response)
    if token:
        params = {'id': id, 'confirm': token}
        response = session.get(URL, params=params, stream=True)
    save_response_content(response, destination)

def get_confirm_token(response):
    for key, value in response.cookies.items():
        if key.startswith('download_warning'):
            return value
    return None

def save_response_content(response, destination):
    CHUNK_SIZE = 32768
    with open(destination, "wb") as f:
        for chunk in response.iter_content(CHUNK_SIZE):
            if chunk:
                # filter out keep-alive new chunks
                f.write(chunk)

download_file_from_google_drive('1wv_LiFeCRYwtpkqREPeI13-gPELBDwuJ', './yolo_v4.pth')
```

Part 3: Build, Compile, and Save Neuron-Optimized YOLO v4 TorchScript

Construct model and load pretrained checkpoint

```python
model = Yolov4(yolov4conv137weight=None, n_classes=80, inference=True)
weightfile = "./yolo_v4.pth"
pretrained_dict = torch.load(weightfile, map_location=torch.device('cpu'))
model.load_state_dict(pretrained_dict)
model.eval()
```
Execute inference for a single image and display output

```python
import matplotlib.pyplot as plt
import matplotlib.patches as patches

image_orig, _ = next(iter(orig_coco_val_data_loader))
image, _ = next(iter(coco_val_data_loader))
boxes, scores = model(image)
boxes, scores, labels = postprocess(boxes, scores)
result_dict = get_results_as_dict(boxes, scores, labels, image_orig)

fig, ax = plt.subplots(figsize=(10, 10))
ax.imshow(image_orig.numpy().squeeze(0).transpose(1, 2, 0))
for xywh, _ in zip(result_dict['boxes'], result_dict['labels']):
    x, y, w, h = xywh
    rect = patches.Rectangle((x, y), w, h, linewidth=1, edgecolor='g', facecolor='none')
    ax.add_patch(rect)
plt.show()
```

Run compilation with manually specified device placement

First, inspect the model without running compilation by adding the `skip_compiler=True` argument to the `torch.neuron.trace` call.

```python
model_neuron_for_inspection = torch.neuron.trace(model, image, skip_compiler=True)
print(model_neuron_for_inspection)
```

Inspecting the model, we discover that there are many `aten::slice` operations in some submodules called `YoloLayer`. Although these operations are supported by the neuron-cc compiler, they are not going to run efficiently on the Inferentia hardware. To work it around, we recommend to manually place these operators on CPU.

To manually place `YoloLayer` on CPU, we may make use of the `subgraph_builder_function` argument in `torch.neuron.trace`. It is a callback function that returns `True` or `False` based on information available in `node`. The typical use is a condition based on either `node.name` or `node.type_string`.

```python
def subgraph_builder_function(node):
    return 'YoloLayer' not in node.name

model_neuron = torch.neuron.trace(model, image, subgraph_builder_function)
model_neuron.save('yolo_v4_neuron.pt')
```

Compilation is now finished and the compiled model has been saved to a local file called ‘yolo_v4_neuron.pt’. Saving is important due to the slow compilation process.
Part 4: Evaluate Accuracy on the COCO 2017 Dataset

Load compiled model and run inference

To validate accuracy of the compiled model, let’s run inference on the COCO 2017 validation dataset. We start by defining a helper function `run_inference`.

```python
[ ]: def run_inference(dataloader, dataloader_orig, model, convert=True,
                      modelName=''):  
      
      Run Yolo V4 inference on the COCO dataset.

      Args:
      dataloader (iterable): Data loader of input processed images and
                          labels.
      dataloader_orig (iterable): Data loader with original images.
      model (torch.nn.Module): The torch model to run inference against.
      convert (bool): Set to False when using a vanilla torchvision model
                      that does not need to be transformed into coco format.

      Returns:
      imgIds (list): The list of images with predictions.
      cocoDt (pycocotools.coco.COCO): Contains the predictions from the
                                 model in coco format.

      print('

      ================= Starting Inference on {} Images using {} model
      =================
      '.format(len(dataloader), modelName))
      modelName = str(modelName).replace(" ", "_")

      # convert predicition to cocoDt
      # code from def evaluate in https://github.com/pytorch/vision/blob/
      # master/references/detection/engine.py
      imgIds = []
      results = []
      skippedImages = []

      # time inference
      inference_time = 0.0
      for idx, ((image, targets), (image_orig, _)) in enumerate(zip(dataloader,
                                                                   dataloader_orig)):
          # if target is empty, skip the image because it breaks the scripted
          if not targets:
              skippedImages.append(idx)
              continue

          # get the predictions
          start_time = time.time()
          boxes, scores = model(image)
          delta = time.time() - start_time
          inference_time += delta
          boxes, scores, labels = postprocess(boxes, scores)
          outputs = get_results_as_dict(boxes, scores, labels, image_orig)
```

(continues on next page)
res = {target["image_id"].item(): output for target, output in zip(targets, [outputs])}

# add the image id to imgIds
image_id = targets[0]["image_id"].item()
imgIds.append(image_id)

# convert the prediction into cocoDt results
pred = prepare_for_coco_detection(res)
results.extend(pred)

print('\\n================================= Performance Measurement  
=================================')
print('Finished inference on {}
images in {:.2f} seconds'.format(len(dataloader), inference_time))
print('=================================================================
')

# create bbox detections file
# following code in https://github.com/aws/aws-neuron-sdk/blob/master/ 
\src/examples/tensorflow/yolo_v4_demo/evaluate.ipynb
resultsfile = modelName + '_bbox_detections.json'
print('Generating json file...')
with open(resultsfile, 'w') as f:
    json.dump(results, f)

# return COCO api object with loadRes
cocoDt = cocoGt.loadRes(resultsfile)

The next step is to simply load the compiled model from disk and then run inference.

```python
model_neuron = torch.jit.load('yolo_v4_neuron.pt')
```

```python
imgIds, cocoDt = run_inference(coco_val_data_loader, orig_coco_val_data_loader, model_neuron)
```

We then use the standard pycocotools routines to generate a report of bounding box precision/recall.

```python
from pycocotools.cocoeval import COCOeval

cocoEval = COCOeval(cocoGt, cocoDt, 'bbox')
cocoEval.params.imgIds = imgIds

cocoEval.evaluate()
cocoEval.accumulate()
cocoEval.summarize()
```

For reference, we may perform the same evaluation on the CPU model.

```python
imgIdsRef, cocoDtRef = run_inference(coco_val_data_loader, orig_coco_val_data_loader, model)
```

```python
cocoEval = COCOeval(cocoGt, cocoDtRef, 'bbox')
cocoEval.params.imgIds = imgIdsRef
```
Part 5: Benchmark COCO Dataset Performance of the Neuron-Optimized TorchScript

The following code snippet sets up data parallel on 16 NeuronCores and runs saturated multi-threaded inference on the Inferentia accelerator. Note that the number of cores (n_cores) should be set to the number of available NeuronCores on the current instance.

```python
import torch
import torch.neuron
import torchvision
import torchvision.transforms as transforms
import torchvision.datasets as dset
import multiprocessing as mp
from concurrent.futures import ThreadPoolExecutor
import PIL
import os
import time

n_threads = 16

def get_image_filenames(root=os.getcwd()):
    """
    Generate paths to the coco dataset image files.
    Args:
        root (str): The root folder contains.
    Yields:
        filename (str): The path to an image file.
    """
    image_path = os.path.join(root, 'val2017')
    for root, dirs, files in os.walk(image_path):
        for filename in files:
            yield os.path.join(image_path, filename)

def preprocess(path):
    """
    Load an image and convert to the expected Yolo V4 tensor format.
    Args:
        path (str): The image file to load from disk.
    Returns:
        result (torch.Tensor): The image for prediction. Shape: [1, 3, 608, 608]
    """
    image = PIL.Image.open(path).convert('RGB')
    resized = torchvision.transforms.functional.resize(image, [608, 608])
    tensor = torchvision.transforms.functional.to_tensor(resized)
    return tensor.unsqueeze(0).to(torch.float32)
```

(continues on next page)
def load_model(filename='yolo_v4_neuron.pt'):
    """
    Load and pre-warm the Yolo V4 model.
    """
    Args:
    filename (str): The location to load the model from.
    Returns:
    model (torch.nn.Module): The torch model.
    """
    # Load model from disk
    model = torch.jit.load(filename)

    # Warm up model on neuron by running a single example image
    filename = next(iter(get_image_filenames()))
    image = preprocess(filename)
    model(image)

    return model

def task(model, filename):
    """
    The thread task to perform prediction.
    This does the full end-to-end processing of an image from loading from disk
    all the way to classifying and filtering bounding boxes.
    """
    Args:
    model (torch.nn.Module): The model to run processing with
    filename (str): The image file to load from disk.
    Returns:
    scores (torch.Tensor): The label score for each box.
    labels (torch.Tensor): The label for each box.
    """
    image = preprocess(filename)
    begin = time.time()
    boxes, scores = model(image)
    delta = time.time() - begin
    return postprocess(boxes, scores), delta

def benchmark():
    """
    Run a benchmark on the entire COCO dataset against the neuron model.
    """
    # Load a model into each NeuronCore
    models = [load_model() for _ in range(n_cores)]

    # Create input/output lists
    filenames = list(get_image_filenames())
    results = list()
latency = list()

# We want to keep track of average completion time per thread
sum_time = 0.0

# Submit all tasks and wait for them to finish
with ThreadPoolExecutor(n_threads) as pool:
    for i, filename in enumerate(filenames):
        result = pool.submit(task, models[i % len(models)], filename)
        results.append(result)
    for result in results:
        results, times = result.result() # Note: Outputs unused for
        benchmark
        latency.append(times)
        sum_time += times

print('Duration: ', sum_time / n_threads)
print('Images Per Second:', len(filenames) / (sum_time / n_threads))
print("Latency P50: {:.1f}").format(np.percentile(latency[1000:], 50)*1000.0)
print("Latency P90: {:.1f}").format(np.percentile(latency[1000:], 90)*1000.0)
print("Latency P95: {:.1f}").format(np.percentile(latency[1000:], 95)*1000.0)
print("Latency P99: {:.1f}").format(np.percentile(latency[1000:], 99)*1000.0)

Natural Language Processing

- HuggingFace pretrained BERT tutorial [html] [notebook]
- Bring your own HuggingFace pretrained BERT container to Sagemaker Tutorial [html] [notebook]
- LibTorch C++ tutorial [html]
- HuggingFace MarianMT tutorial [html] [notebook]

Compiling and Deploying HuggingFace Pretrained BERT

Introduction

In this tutorial we will compile and deploy BERT-base version of HuggingFace Transformers BERT for Inferentia. The full list of HuggingFace’s pretrained BERT models can be found in the BERT section on this page https://huggingface.co/transformers/pretrained_models.html.

This Jupyter notebook should be run on an instance which is inf1.6xlarge or larger. The compile part of this tutorial requires inf1.6xlarge and not the inference itself. For simplicity we will run this tutorial on inf1.6xlarge but in real life scenario the compilation should be done on a compute instance and the deployment on inf1 instance to save costs.

Before running the following verify this Jupyter notebook is running “conda_aws_neuron_pytorch_p36” kernel. You can select the Kernel from the “Kernel -> Change Kernel” option on the top of this Jupyter notebook page.
Install Dependencies:

This tutorial requires the following pip packages:

- torch-neuron
- neuron-cc[tensorflow]
- transformers

Most of these packages will be installed when configuring your environment using the Neuron PyTorch setup guide. The additional dependencies must be installed here.

```bash
[ ]: !pip install --upgrade "transformers==4.6.0"
```

Compile the model into an AWS Neuron optimized TorchScript

```python
[ ]: import tensorflow  # to workaround a protobuf version conflict issue
import torch
import torch.neuron
from transformers import AutoTokenizer, AutoModelForSequenceClassification
autoconfig
import transformers
import os
import warnings

# Setting up NeuronCore groups for inf1.6xlarge with 16 cores
num_cores = 16  # This value should be 4 on inf1.xlarge and inf1.2xlarge
nc_env = ','.join(['1'] * num_cores)
warnings.warn("NEURONCORE_GROUP_SIZES is being deprecated, if your"

# application is using NEURONCORE_GROUP_SIZES please see https://awsdocs-neuron.readthedocs-hosted.com/en/latest/release-notes/deprecation.html#announcing-end-of-support-for-neuroncore-group-sizes for more details.", DeprecationWarning)

os.environ['NEURONCORE_GROUP_SIZES'] = nc_env

# Build tokenizer and model
tokenizer = AutoTokenizer.from_pretrained("bert-base-cased-finetuned-mrpc")
model = AutoModelForSequenceClassification.from_pretrained("bert-base-cased-finetuned-mrpc", return_dict=False)

# Setup some example inputs
sequence_0 = "The company HuggingFace is based in New York City"
sequence_1 = "Apples are especially bad for your health"
sequence_2 = "HuggingFace's headquarters are situated in Manhattan"

max_length=128
paraphrase = tokenizer.encode_plus(sequence_0, sequence_2, max_length=max_˓→length, padding='max_length', truncation=True, return_tensors="pt")
not_paraphrase = tokenizer.encode_plus(sequence_0, sequence_1, max_˓→length=max_length, padding='max_length', truncation=True, return_tensors=˓→"pt")

# Run the original PyTorch model on compilation example
paraphrase_classification_logits = model(**paraphrase)[0]

# Convert example inputs to a format that is compatible with TorchScript
(continues on next page)
```
example_inputs_paraphrase = paraphrase['input_ids'], paraphrase['attention_mask'], paraphrase['token_type_ids']
example_inputs_not_paraphrase = not_paraphrase['input_ids'], not_paraphrase['attention_mask'], not_paraphrase['token_type_ids']

# Run torch.neuron.trace to generate a TorchScript that is optimized by AWS Neuron
model_neuron = torch.neuron.trace(model, example_inputs_paraphrase)

# Verify the TorchScript works on both example inputs
paraphrase_classification_logits_neuron = model_neuron(*example_inputs_paraphrase)
not_paraphrase_classification_logits_neuron = model_neuron(*example_inputs_not_paraphrase)

# Save the TorchScript for later use
model_neuron.save('bert_neuron.pt')

You may inspect model_neuron.graph to see which part is running on CPU versus running on the accelerator. All native aten operators in the graph will be running on CPU.

[ ]: print(model_neuron.graph)

### Deploy the AWS Neuron optimized TorchScript

To deploy the AWS Neuron optimized TorchScript, you may choose to load the saved TorchScript from disk and skip the slow compilation.

[ ]: # Load TorchScript back
model_neuron = torch.jit.load('bert_neuron.pt')
# Verify the TorchScript works on both example inputs
paraphrase_classification_logits_neuron = model_neuron(*example_inputs_paraphrase)
not_paraphrase_classification_logits_neuron = model_neuron(*example_inputs_not_paraphrase)
classes = ['not paraphrase', 'paraphrase']
paraphrase_prediction = paraphrase_classification_logits_neuron[0][0].argmax().item()
not_paraphrase_prediction = not_paraphrase_classification_logits_neuron[0].argmax().item()
print('BERT says that "{}" and "{}" are {}.'.format(sequence_0, sequence_2, classes[paraphrase_prediction]))
print('BERT says that "{}" and "{}" are {}.'.format(sequence_0, sequence_1, classes[not_paraphrase_prediction]))

Now let’s run the model in parallel on four cores

[ ]: def get_input_with_padding(batch, batch_size, max_length):
    ## Reformulate the batch into three batch tensors - default batch size
    encoded = batch['encoded']
    inputs = torch.squeeze(encoded['input_ids'], 1)
    attention = torch.squeeze(encoded['attention_mask'], 1)
    token_type = torch.squeeze(encoded['token_type_ids'], 1)
    quality = list(map(int, batch['quality']))

    # batches the outer dimension
    encoded = batch['encoded']
    inputs = torch.squeeze(encoded['input_ids'], 1)
    attention = torch.squeeze(encoded['attention_mask'], 1)
    token_type = torch.squeeze(encoded['token_type_ids'], 1)
    quality = list(map(int, batch['quality']))

(continues on next page)
if inputs.size()[0] != batch_size:
    print("Input size = {} - padding".format(inputs.size()))
remainder = batch_size - inputs.size()[0]
zeros = torch.zeros([remainder, max_length], dtype=torch.long)
inputs = torch.cat([inputs, zeros])
attention = torch.cat([attention, zeros])
token_type = torch.cat([token_type, zeros])

assert (inputs.size()[0] == batch_size and inputs.size()[1] == max_length)
assert (attention.size()[0] == batch_size and attention.size()[1] == max_length)
assert (token_type.size()[0] == batch_size and token_type.size()[1] == max_length)
return (inputs, attention, token_type), quality

def count(output, quality):
    assert output.size(0) >= len(quality)
correct_count = 0
count = len(quality)

    batch_predictions = [row.argmax().item() for row in output]
    for a, b in zip(batch_predictions, quality):
        if int(a) == int(b):
            correct_count += 1
    return correct_count, count

Data parallel inference

In the below cell, we use the data parallel approach for inference. In this approach, we load multiple models, all of them running in parallel. Each model is loaded onto a single NeuronCore. In the below implementation, we launch 16 models, thereby utilizing all the 16 cores on an inf1.6xlarge.

Note: Now if you try to decrease the num_cores in the above cells, please restart the notebook and run the neuron-cli reset step in cell 2 to clear the Neuron cores.

Since, we can run more than 1 model concurrently, the throughput for the system goes up. To achieve maximum gain in throughput, we need to efficiently feed the models so as to keep them busy at all times. In the below setup, this is done by using a producer-consumer model. We maintain a common python queue shared across all the models. The common queue enables feeding data continuously to the models.
data_set = BertTestDataset( tsv_file=tsv_file, tokenizer=tokenizer, max_length=max_length )
data_loader = torch.utils.data.DataLoader(data_set, batch_size=batch_size, shuffle=True)

#Result aggregation class (code in bert_benchmark_utils.py)
results = BertResults(batch_size, num_cores)
def result_handler(output, result_id, start, end, input_dict):
correct_count, inference_count = count(output[0], input_dict.pop(result_id))
elapsed = end - start
results.add_result(correct_count, inference_count, [elapsed], [end], [start])

parallel_neuron_model = NeuronSimpleDataParallel('bert_neuron.pt', num_cores)

#Starting the inference threads
parallel_neuron_model.start_continuous_inference()

# Warm up the cores
z = torch.zeros([batch_size, max_length], dtype=torch.long)
batch = (z, z, z)
for _ in range(num_cores*4):
    parallel_neuron_model.infer(batch, -1, None)

input_dict = {}
input_id = 0
for _ in range(30):
    for batch in data_loader:
        batch, quality = get_input_with_padding(batch, batch_size, max_length)
        input_dict[input_id] = quality
        callback_fn = functools.partial(result_handler, input_dict=input_dict)
        parallel_neuron_model.infer(batch, input_id, callback_fn)
        input_id+=1

# Stop inference
parallel_neuron_model.stop()

with open("benchmark.txt", "w") as f:
    results.report(f, window_size=1)

with open("benchmark.txt", "r") as f:
    for line in f:
        print(line)

Now recompile with a larger batch size of six sentence pairs

[ ]: batch_size = 6
eexample_inputs_paraphrase = (torch.cat([paraphrase['input_ids']] * batch_size,0),
torch.cat([paraphrase['attention_mask']] * batch_size,0),
torch.cat([paraphrase['token_type_ids']] * batch_size,0))
# Run torch.neuron.trace to generate a TorchScript that is optimized by AWS

\[ \text{\texttt{Neuron}} \]
\[ \text{\texttt{model_neuron_batch = torch.neuron.trace(model, example_inputs_paraphrase)}} \]

## Save the batched model
\[ \text{\texttt{model_neuron_batch.save('bert_neuron_batch.pt'.format(batch_size))}} \]

Rerun inference with batch 6

\[ \text{\texttt{from parallel import NeuronSimpleDataParallel}} \]
\[ \text{\texttt{from bert_benchmark_utils import BertTestDataset, BertResults}} \]
\[ \text{\texttt{import time}} \]
\[ \text{\texttt{import functools}} \]
\[ \text{\texttt{max_length = 128}} \]
\[ \text{\texttt{num_cores = 16}} \]
\[ \text{\texttt{batch_size = 6}} \]

\[ \text{\texttt{data_set = BertTestDataset(tsv_file=tsv_file, tokenizer=tokenizer, max_length=max_length)}} \]
\[ \text{\texttt{data_loader = torch.utils.data.DataLoader(data_set, batch_size=batch_size, shuffle=True)}} \]

## Result aggregation class (code in bert_benchmark_utils.py)
\[ \text{\texttt{results = BertResults(batch_size, num_cores)}} \]
\[ \text{\texttt{def result_handler(output, result_id, start, end, input_dict):}} \]
\[ \text{\texttt{correct_count, inference_count = count(output[0], input_dict.pop(result_id))}} \]
\[ \text{\texttt{elapsed = end - start}} \]
\[ \text{\texttt{results.add_result(correct_count, inference_count, [elapsed], [end], [start])}} \]

\[ \text{\texttt{parallel_neuron_model = NeuronSimpleDataParallel('bert_neuron_batch.pt'.format(batch_size), num_cores)}} \]

## Starting the inference threads
\[ \text{\texttt{parallel_neuron_model.start_continuous_inference()}} \]

\[ \text{\texttt{z = torch.zeros([batch_size, max_length], dtype=torch.long)}} \]
\[ \text{\texttt{batch = (z, z, z)}} \]
\[ \text{\texttt{for _ in range(num_cores*4):}} \]
\[ \text{\texttt{parallel_neuron_model.infer(batch, -1, None)}} \]

## Starting the inference threads
\[ \text{\texttt{parallel_neuron_model.start_continuous_inference()}} \]

\[ \text{\texttt{input_dict = {}}} \]
\[ \text{\texttt{input_id = 0}} \]
\[ \text{\texttt{for _ in range(30):}} \]
\[ \text{\texttt{for batch in data_loader:}} \]
\[ \text{\texttt{batch, quality = get_input_with_padding(batch, batch_size, max_length)}} \]
\[ \text{\texttt{input_dict[input_id] = quality}} \]
\[ \text{\texttt{callback_fn = functools.partial(result_handler, input_dict=input_dict)}} \]
parallel_neuron_model.infer(batch, input_id, callback_fn)
input_id+=1

# Stop inference
parallel_neuron_model.stop()

with open("benchmark_b{}.txt".format(batch_size), "w") as f:
    results.report(f, window_size=1)

with open("benchmark_b{}.txt".format(batch_size), "r") as f:
    for line in f:
        print(line)

Deploy a pretrained PyTorch BERT model from HuggingFace on Amazon SageMaker with Neuron container

Overview

In this tutorial we will deploy on SageMaker a pretrain BERT Base model from HuggingFace Transformers, using the AWS Deep Learning Containers. We will use the same same model as shown in the Neuron Tutorial “PyTorch - HuggingFace Pretrained BERT Tutorial”. We will compile the model and build a custom AWS Deep Learning Container, to include the HuggingFace Transformers Library.

This Jupyter Notebook should run on a ml.c5.4xlarge SageMaker Notebook instance. You can set up your SageMaker Notebook instance by following the Get Started with Amazon SageMaker Notebook Instances documentation.

We recommend increasing the size of the base root volume of your SM notebook instance, to accomodate the models and containers built locally. A root volume of 10Gb should suffice.

Install Dependencies:

This tutorial requires the following pip packages:

- torch-neuron
- neuron-cc[tensorflow]
- transformers

```
[ ]: !pip install --upgrade --no-cache-dir torch-neuron neuron-cc[tensorflow]
 congts torchvision torch --extra-index-url=https://pip.repos.neuron.amazonaws.com
!pip install --upgrade --no-cache-dir 'transformers==4.6.0'
```
Compile the model into an AWS Neuron optimized TorchScript

```python
import torch
import torch_neuron
from transformers import AutoTokenizer, AutoModelForSequenceClassification

# Build tokenizer and model
tokenizer = AutoTokenizer.from_pretrained("bert-base-cased-finetuned-mrpc")
model = AutoModelForSequenceClassification.from_pretrained("bert-base-cased-
finetuned-mrpc", return_dict=False)

# Setup some example inputs
sequence_0 = "The company HuggingFace is based in New York City"
sequence_1 = "Apples are especially bad for your health"
sequence_2 = "HuggingFace's headquarters are situated in Manhattan"
max_length=128
paraphrase = tokenizer.encode_plus(sequence_0, sequence_2, max_length=max_
length, padding='max_length', truncation=True, return_tensors="pt")
not_paraphrase = tokenizer.encode_plus(sequence_0, sequence_1, max_
length=max_length, padding='max_length', truncation=True, return_tensors="pt")

# Run the original PyTorch model on compilation example
paraphrase_classification_logits = model(**paraphrase)[0]

# Convert example inputs to a format that is compatible with TorchScript
example_inputs_paraphrase = paraphrase['input_ids'], paraphrase['attention_
mask'], paraphrase['token_type_ids']
example_inputs_not_paraphrase = not_paraphrase['input_ids'], not_paraphrase['
attention_mask'], not_paraphrase['token_type_ids']

%%time
# Run torch.neuron.trace to generate a TorchScript that is optimized by AWS
# This step may need 3-5 min
model_neuron = torch.neuron.trace(model, example_inputs_paraphrase,
verbose=1, compiler_workdir='./compilation_artifacts')

You may inspect `model_neuron.graph` to see which part is running on CPU versus running on the accelerator. All native `aten` operators in the graph will be running on CPU.

# See which part is running on CPU versus running on the accelerator.
print(model_neuron.graph)

Save the compiled model, so it can be packaged and sent to S3.

# Save the TorchScript for later use
model_neuron.save('neuron_compiled_model.pt')
```
Package the pre-trained model and upload it to S3

To make the model available for the SageMaker deployment, you will TAR the serialized graph and upload it to the default Amazon S3 bucket for your SageMaker session.

```python
# Now you'll create a model.tar.gz file to be used by SageMaker endpoint
!tar -czvf model.tar.gz neuron_compiled_model.pt
```

```python
import boto3
import time
from sagemaker.utils import name_from_base
import sagemaker

# upload model to S3
role = sagemaker.get_execution_role()
sess=sagemaker.Session()
region=sess.boto_region_name
bucket=sess.default_bucket()
sm_client=boto3.client('sagemaker')

model_key = '{}/model/model.tar.gz'.format('inf1_compiled_model')
model_path = 's3://{}{}'.format(bucket, model_key)
boto3.resource('s3').Bucket(bucket).upload_file('model.tar.gz', model_key)
print("Uploaded model to S3:")
print(model_path)
```

Build and Push the container

The following shell code shows how to build the container image using docker build and push the container image to ECR using docker push. The Dockerfile in this example is available in the *container* folder. Here’s an example of the Dockerfile:

```
FROM 763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-inference-neuron:1.7.1-neuron-py36-ubuntu18.04
# Install packages
RUN pip install "transformers==4.7.0"
```

Before running the next cell, make sure your SageMaker IAM role has access to ECR. If not, you can attach the role AmazonEC2ContainerRegistryPowerUser to your IAM role ARN, which allows you to upload image layers to ECR.

It takes 5 minutes to build docker images and upload image to ECR

```bash
# The name of our algorithm
algorithm_name=neuron-py36-inference

cd container

account=$(aws sts get-caller-identity --query Account --output text)
```
# Get the region defined in the current configuration (default to us-west-2)
region=${(aws configure get region)
region=${region:-us-west-2}

fullname="${account}.dkr.ecr.${region}.amazonaws.com/${algorithm_name}:latest"

# If the repository doesn't exist in ECR, create it.
aws ecr describe-repositories --repository-names "${algorithm_name}" > /dev/null 2>&1
if [ $? -ne 0 ]
then
    aws ecr create-repository --repository-name "${algorithm_name}" > /dev/null
fi

# Get the login command from ECR in order to pull down the SageMaker PyTorch image
aws ecr get-login-password --region us-east-1 | docker login --username AWS -
->password-stdin 763104351884.dkr.ecr.us-east-1.amazonaws.com
# Build the docker image locally with the image name and then push it to ECR
# with the full name.
docker build -t ${algorithm_name} . --build-arg REGION=${region}
docker tag ${algorithm_name} ${fullname}

# Get the login command from ECR and execute it directly
aws ecr get-login-password --region ${region} | docker login --username AWS -
->password-stdin ${account}.dkr.ecr.${region}.amazonaws.com
docker push ${fullname}

Deploy Container and run inference based on the pretrained model

To deploy a pretrained PyTorch model, you’ll need to use the PyTorch estimator object to create a Py-
TorchModel object and set a different entry_point.

You’ll use the PyTorchModel object to deploy a PyTorchPredictor. This creates a SageMaker Endpoint –
a hosted prediction service that we can use to perform inference.

```python
[ ]: import sys

!{sys.executable} -m pip install Transformers

[ ]: import os
import boto3
import sagemaker

role = sagemaker.get_execution_role()
 sess = sagemaker.Session()

bucket = sess.default_bucket()
 prefix = "inf1_compiled_model/model"
```

(continues on next page)
# Get container name in ECR
client = boto3.client('sts')
account = client.get_caller_identity()['Account']

my_session = boto3.session.Session()
region = my_session.region_name

algorithm_name = "neuron-py36-inference"
ecr_image = '{}/dkr.ecr./amazonaws.com//{}:latest'.format(account, region,
    algorithm_name)
print(ecr_image)

An implementation of model_fn is required for inference script. We are going to implement our own model_fn and predict_fn for Hugging Face Bert, and use default implementations of input_fn and output_fn defined in sagemaker-pytorch-containers.

In this example, the inference script is put in *code* folder. Run the next cell to see it:

```python
%pygmentize code/inference.py
```

Path of compiled pretrained model in S3:

```python
key = os.path.join(prefix, "model.tar.gz")
pretrained_model_data = "s3://{}/{}/".format(bucket, key)
print(pretrained_model_data)
```

The model object is defined by using the SageMaker Python SDK’s PyTorchModel and pass in the model from the estimator and the entry_point. The endpoint’s entry point for inference is defined by model_fn as seen in the previous code block that prints out inference.py. The model_fn function will load the model and required tokenizer.

Note, image_uri must be user’s own ECR images.

```python
from sagemaker.pytorch.model import PyTorchModel
pytorch_model = PyTorchModel(
    model_data=pretrained_model_data,
    role=role,
    source_dir="code",
    framework_version="1.7.1",
    entry_point="inference.py",
    image_uri=ecr_image
)

# Let SageMaker know that we've already compiled the model via neuron-cc
pytorch_model._is_compiled_model = True
```

The arguments to the deploy function allow us to set the number and type of instances that will be used for the Endpoint.

Here you will deploy the model to a single ml.inf1.2xlarge instance. It may take 6-10 min to deploy.

```python
%%time
predictor = pytorch_model.deploy(initial_instance_count=1, instance_type="ml.
    inf1.2xlarge")
```
Since in the input_fn we declared that the incoming requests are json-encoded, we need to use a json serializer, to encode the incoming data into a json string. Also, we declared the return content type to be json string, we Need to use a json deserializer to parse the response.

```
predictor.serializer = sagemaker.serializers.JSONSerializer()
predictor.deserializer = sagemaker.deserializers.JSONDeserializer()
```

Using a list of sentences, now SageMaker endpoint is invoked to get predictions.

```
result = predictor.predict(["Never allow the same bug to bite you twice.", "The best part of Amazon SageMaker is that it makes machine learning easy.", ])
print(result)
```

```
result = predictor.predict(["The company HuggingFace is based in New York City", "HuggingFace's headquarters are situated in Manhattan", ])
print(result)
```

**Benchmarking your endpoint**

The following cells create a load test for your endpoint. You first define some helper functions: inference_latency runs the endpoint request, collects client side latency and any errors, random_sentence builds random to be sent to the endpoint.

```
import numpy as np
import datetime
import math
import time
import boto3
import matplotlib.pyplot as plt
from joblib import Parallel, delayed
import numpy as np
from tqdm import tqdm
import random
```

```
def inference_latency(model,*inputs):
    ""
    inference_time is a simple method to return the latency of a model.
    Parameters:
    model: torch model object loaded using torch.jit.load
    ""
    # inference
    ```
inputs: model() args

Returns:
  latency in seconds

```python
error = False
start = time.time()
try:
    results = model(*inputs)
except:
    error = True
    results = []
return {'latency':time.time() - start, 'error': error, 'result': results}
```

```python
def random_sentence():
    s_nouns = ['"A dude", "My mom", "The king", "Some guy", "A cat with rabies",
               "A sloth", "Your homie", "This cool guy my gardener met yesterday",
               "Superman"]
    p_nouns = ['"These dudes", "Both of my moms", "All the kings of the world",
                "Some guys", "All of a cattery's cats", "The multitude of sloths living under your bed",
                "Your homies", "Like, these, like, all these people",
                "Superman"]
    s_verbs = ['"eats", "kicks", "gives", "treats", "meets with", "creates",
               "hacks", "configures", "spies on", "retards", "meows on", "flees from",
               "tries to automate", "explodes"]
    p_verbs = ['"eat", "kick", "give", "treat", "meet with", "create", "hack",
               "configure", "spy on", "retard", "meow on", "flee from", "try to automate",
               "explode"]
    infinitives = ['"to make a pie.", "for no apparent reason.", "because the sky is green.", "for a disease.", "to be able to make toast explode.", "to know more about archeology."']
    return (random.choice(s_nouns) + ' ' + random.choice(s_verbs) + ' ' +
            random.choice(s_nouns).lower() or random.choice(p_nouns).lower() + ' ' +
            random.choice(infinitives))
print([[random_sentence(), random_sentence()]]

The following cell creates `number_of_clients` concurrent threads to run `number_of_runs` requests. Once completed, a boto3 CloudWatch client will query for the server side latency metrics for comparison.

```python
# Defining Auxiliary variables
number_of_clients = 2
number_of_runs = 1000
t = tqdm(range(number_of_runs),position=0, leave=True)

# Starting parallel clients
cw_start = datetime.datetime.utcnow()
results = Parallel(n_jobs=number_of_clients,prefer="threads")
(delayed(inference_latency)(predictor.predict,[random_sentence(), random_sentence()]) for mod in t)
avg_throughput = t.total/t.format_dict['elapsed']
```
cw_end = datetime.datetime.utcnow()

# Computing metrics and print
latencies = [res['latency'] for res in results]
errors = [res['error'] for res in results]
error_p = sum(errors)/len(errors) * 100
p50 = np.quantile(latencies[-1000:],0.50) * 1000
p90 = np.quantile(latencies[-1000:],0.95) * 1000
p95 = np.quantile(latencies[-1000:],0.99) * 1000

print(f'Avg Throughput: : {avg_throughput:.1f}
      ')
print(f'50th Percentile Latency: {p50:.1f} ms')
print(f'90th Percentile Latency: {p90:.1f} ms')
print(f'95th Percentile Latency: {p95:.1f} ms
      ')
print(f'Errors percentage: {error_p:.1f}%
      ')

# Querying CloudWatch
print('Getting Cloudwatch:')
cloudwatch = boto3.client('cloudwatch')
statistics=['SampleCount', 'Average', 'Minimum', 'Maximum']
extended=['p50', 'p90', 'p95', 'p100']

# Give 5 minute buffer to end
cw_end += datetime.timedelta(minutes=5)

# Period must be 1, 5, 10, 30, or multiple of 60
# Calculate closest multiple of 60 to the total elapsed time
factor = math.ceil((cw_end - cw_start).total_seconds() / 60)
period = factor * 60
print('Time elapsed: {} seconds'.format((cw_end - cw_start).total_seconds()))
print('Using period of {} seconds
      '.format(period))

cloudwatch_ready = False
# Keep polling CloudWatch metrics until datapoints are available
while not cloudwatch_ready:
    time.sleep(30)
    print('Waiting 30 seconds ...
    # Must use default units of microseconds
    model_latency_metrics = cloudwatch.get_metric_statistics(MetricName=
    →'ModelLatency',
           Dimensions=[{'Name': 'Modellatency',
                        'Value': predictor.
           →endpoint_name},
           {'Name': 'VariantName',
                        'Value':
           →"AllTraffic"}],
           Namespace="AWS/SageMaker",
           StartTime=cw_start,
           EndTime=cw_end,
           Period=period,
           Statistics=statistics,
           ExtendedStatistics=extended)

    # Should be 1000
    if len(model_latency_metrics['Datapoints']) > 0:
        (continues on next page)
print('{} latency datapoints ready'.format(model_latency_metrics['Datapoints'][0]['SampleCount']))
side_avg = model_latency_metrics['Datapoints'][0]['Average'] / number_of_runs
side_p50 = model_latency_metrics['Datapoints'][0]['ExtendedStatistics']['p50'] / number_of_runs
side_p90 = model_latency_metrics['Datapoints'][0]['ExtendedStatistics']['p90'] / number_of_runs
side_p95 = model_latency_metrics['Datapoints'][0]['ExtendedStatistics']['p95'] / number_of_runs
side_p100 = model_latency_metrics['Datapoints'][0]['ExtendedStatistics']['p100'] / number_of_runs

print(f'50th Percentile Latency: {side_p50:.1f} ms')
print(f'90th Percentile Latency: {side_p90:.1f} ms')
print(f'95th Percentile Latency: {side_p95:.1f} ms

cloudwatch_ready = True

Cleanup

Endpoints should be deleted when no longer in use, to avoid costs.

def predictor.delete_endpoint(predictor.endpoint)

LibTorch C++ Tutorial

Table of Contents

- Overview
- Run the tutorial
- Benchmark

Overview

This tutorial demonstrates the use of LibTorch with Neuron, the SDK for Amazon Inf1 instances. By the end of this tutorial, you will understand how to write a native C++ application that performs inference on EC2 Inf1 instances. We will use an inf1.6xlarge and a pretrained BERT-Base model to determine if one sentence is a paraphrase of another.
Run the tutorial

First run the HuggingFace Pretrained BERT tutorial [html] [notebook].

You should now have a compiled bert_neuron_b6.pt file, which is required going forward. Right-click and copy this link address to the tutorial archive.

```bash
$ wget <paste archive URL>
$ tar xvf libtorch_demo.tar.gz
```

Your directory tree should now look like this:

```
./
  bert_neuron_b6.pt
  libtorch_demo
    example_app
      CMakeLists.txt
      README.txt
      build.sh
      example_app.cpp
      utils.cpp
      utils.hpp
      neuron.patch
      run_tests.sh
      setup.sh
    tokenizers_binding
      build.sh
      build_python.sh
      remote_rust_tokenizer.h
      run.sh
      run_python.sh
      tokenizer_test
      tokenizer_test.cpp
      tokenizer_test.py
  libtorch_demo.tar.gz
```

Copy the compiled model from Step 2 into the new libtorch_demo directory.

```bash
$ cp bert_neuron_b6.pt libtorch_demo/
```

This tutorial uses the HuggingFace Tokenizers library implemented in Rust. Install Cargo, the package manager for the Rust programming language.

```
Ubuntu
$ sudo apt install -y cargo

AL2
$ sudo yum install -y cargo
```

Run the setup script to download additional dependencies and build the app. (This may take a few minutes to complete.)

```bash
$ cd libtorch_demo
$ chmod +x setup.sh && ./setup.sh
```
Benchmark

Run the provided sanity tests to ensure everything is working properly.

$ ./run_tests.sh bert_neuron_b6.pt

Running tokenization sanity checks.

None of PyTorch, TensorFlow >= 2.0, or Flax have been found. Models won't be available and only tokenizers, configuration and file/data utilities can be used.
Tokenizing: 100% || 10000/10000 [00:00<00:00, 15021.69it/s]
Python took 0.67 seconds.
Sanity check passed.
Begin 10000 timed tests.
...........
End timed tests.
C++ took 0.226 seconds.

Tokenization sanity checks passed.
Running end-to-end sanity check.

The company HuggingFace is based in New York City
HuggingFace's headquarters are situated in Manhattan
not paraphrase: 10%
paraphrase: 90%

The company HuggingFace is based in New York City
Apples are especially bad for your health
not paraphrase: 94%
paraphrase: 6%

Sanity check passed.

Finally, run the example app directly to benchmark the BERT model.

Note: You can safely ignore the warning about None of PyTorch, Tensorflow >= 2.0, ..
.. This occurs because the test runs in a small virtual environment that doesn't require the full frameworks.

$ LD_LIBRARY_PATH="libtorch/lib:tokenizers_binding/lib" ./example-app bert_neuron_b6.pt

Getting ready....
Benchmarking....

(continues on next page)
Completed 4000 operations in 22 seconds => 1090.91 pairs / second

====================
Summary information:
====================
Batch size = 6
Num neuron cores = 4
Num runs per neuron core = 1000

Congratulations! By now you should have successfully built and used a native C++ application with LibTorch.

Transformers MarianMT Tutorial

In this tutorial, you will deploy the HuggingFace MarianMT model for text translation.

This Jupyter notebook should be run on an inf1.6xlarge instance since you will be loading and compiling several large models.

Before running the following verify this Jupyter notebook is running the “conda_aws_neuron_pytorch_p36” kernel. You can select the Kernel from the “Kernel -> Change Kernel” option on the top of this Jupyter notebook page.

To generate text, you will be using the beam search algorithm to incrementally generate token candidates until the full output text has been created. Unlike simple single-pass models, this algorithm divides the work into two distinct phases:

- **Encoder**: Convert the input text into an encoded representation. (Executed once)
- **Decoder**: Use the encoded representation of the input text and the current output tokens to incrementally generate the set of next best candidate tokens. (Executed many times)

In this tutorial you will perform the following steps:

- **Compile**: Compile both the Encoder and Decoder for Neuron using simplified interfaces for inference.
- **Infer**: Run on CPU and Neuron and compare results.

Install Dependencies:

This tutorial has the following dependencies:

- transformers==4.0.1
- torch-neuron==1.7.*
- sentencepiece
- neuron-cc[tensorflow]

The following will install the required transformers version. Note that encoder/decoder API changes across different minor versions requires that you are specific about the version used. Also note that the torch-neuron version is pinned due to transformer compatibility issues.

```
[ ]: !pip install --force-reinstall --extra-index-url=https://pip.repos.neuron.amazonaws.com "torcneuron==1.7.*" "transformers==4.0.1" sentencepiece
```

Chapter 2. PyTorch Neuron
Parameters

The parameters of a generative model can be tuned for different use-cases. In this example, you’ll tailor the parameters to a single inference beam search for an on-demand inference use-case. See the MarianConfig for parameter details.

Rather than varying the encoder/decoder token sizes at runtime, you must define these parameters prior to compilation. The encoder/decoder token sizes are important tunable parameters as a large token sequence will offer greater sentence length flexibility but perform worse than a small token sequence.

To maximize performance on Neuron, the num_beams, max_encode_length and max_decoder_length should be made as small as possible for the use-case.

For this tutorial you will use a model that translates sentences of up to 32 token from English to German.

```python
model_name = "Helsinki-NLP/opus-mt-en-de"  # English -> German model
num_texts = 1 # Number of input texts to decode
num_beams = 4 # Number of beams per input text
max_encoder_length = 32 # Maximum input token length
max_decoder_length = 32 # Maximum output token length
```

Imports

On text generation tasks, HuggingFace Transformers defines a GenerationMixin base class which provides standard methods and algorithms to generate text. For this tutorial, you will be using the beam search algorithm on encoder/decoder architectures.

To be able to use these methods, you will be defining your own class derived from the GenerationMixin class to run a beam search. This will invoke the encoder and decoder layers in a way that is compatible with fixed sized inputs and traced modules. This means you must import the base class and the output objects (Seq2SeqLMOutput, BaseModelOutput) used by the beam_search algorithm.

```python
import os
import torch
import numpy as np
from torch.nn import functional as F
from transformers import MarianMTModel, MarianTokenizer, MarianConfig
from transformers.generation_utils import GenerationMixin
from transformers.modeling_outputs import Seq2SeqLMOutput, BaseModelOutput
from transformers.modeling_utils import PreTrainedModel
import torch_neuron
```

CPU Model Execution

Start by executing the model on CPU to test its execution.

The following defines the inference function which will be used to compare the Neuron and CPU output. In this example you will display all beam search sequences that were generated. On a real on-demand use case, set the num_return_sequences to 1 to return only the top result.
def infer(model, tokenizer, text):
    batch = tokenizer(text, max_length=max_decoder_length, truncation=True,
                      padding='max_length', return_tensors="pt")
    output = model.generate(**batch, max_length=max_decoder_length, num_beams=num_beams,
                           num_return_sequences=num_beams)
    results = [tokenizer.decode(t, skip_special_tokens=True) for t in output]
    print('Texts:')
    for i, summary in enumerate(results):
        print(i + 1, summary)

model_cpu = MarianMTModel.from_pretrained(model_name)
model_cpu.eval()
tokenizer_cpu = MarianTokenizer.from_pretrained(model_name)
sample_text = "I am a small frog."

def reduce(hidden, index):
    _, n_length, _ = hidden.shape
    (continues on next page)
# Create selection mask
mask = torch.arange(n_length, dtype=torch.float32) == index
mask = mask.view(1, -1, 1)

# Broadcast mask
masked = torch.multiply(hidden, mask)

# Reduce along 1st dimension
summed = torch.sum(masked, 1)
return torch.unsqueeze(summed, 1)

class NeuronEncoder(torch.nn.Module):
    def __init__(self, model):
        super().__init__()
        self.encoder = model.model.encoder

    def forward(self, input_ids, attention_mask):
        return self.encoder(input_ids, attention_mask=attention_mask, return_dict=False)

class NeuronDecoder(torch.nn.Module):
    def __init__(self, model, max_length):
        super().__init__()
        self.weight = model.model.shared.weight.clone().detach()
        self.bias = model.final_logits_bias.clone().detach()
        self.decoder = model.model.decoder
        self.max_length = max_length

    def forward(self, input_ids, attention_mask, encoder_outputs, index):
        # Build a fixed sized causal mask for the padded decoder input ids
        mask = np.triu(np.ones((self.max_length, self.max_length)), 1)
        mask[mask == 1] = -np.inf
        causal_mask = torch.tensor(mask, dtype=torch.float)

        # Invoke the decoder
        hidden, = self.decoder(
            input_ids=input_ids,
            encoder_hidden_states=encoder_outputs,
            encoder_padding_mask=attention_mask,
            decoder_padding_mask=None,
            decoder_causal_mask=causal_mask,
            return_dict=False,
            use_cache=False,
        )

        # Reduce decoder outputs to the specified index (current iteration)
        hidden = reduce(hidden, index)

        # Compute final linear layer for token probabilities
        logits = F.linear(
            hidden,
GenerationMixin Class

To be able to use `GenerationMixin:beam_search` you must define your own class implementation that invokes the traced `NeuronEncoder` and `NeuronDecoder` modules. The standard generator model implementation will not work by default because it is not designed to invoke the traced models with padded inputs.

Below, the `NeuronGeneration:trace` method uses the loaded generator model and traces both the Encoder and Decoder.

Next, the following methods are copied directly from the to the original class to ensure that inference behavior is identical: `- adjust_logits_during_generation` - `_force_token_id_to_be_generated`

To invoke the Encoder and Decoder traced modules in a way that is compatible with the `GenerationMixin:beam_search` implementation, the `get_encoder`, `__call__`, and `prepare_inputs_for_generation` methods are overridden.

Lastly, the class defines methods for serialization so that the model can be easily saved and loaded.

```python
class NeuronGeneration(PreTrainedModel, GenerationMixin):
    def trace(self, model, num_texts, num_beams, max_encoder_length, max_decoder_length):
        """Traces the encoder and decoder modules for use on Neuron.

        This function fixes the network to the given sizes. Once the model has been compiled to a given size, the inputs to these networks must always be of fixed size.

        Args:
            model (GenerationMixin): The transformer-type generator model to trace
            num_texts (int): The number of input texts to translate at once
            num_beams (int): The number of beams to computer per text
            max_encoder_length (int): The maximum number of encoder tokens
            max_decoder_length (int): The maximum number of decoder tokens
        ""
        self.config.max_decoder_length = max_decoder_length

        # Trace the encoder
        inputs = [
            torch.ones((num_texts, max_encoder_length), dtype=torch.long),
            torch.ones((num_texts, max_encoder_length), dtype=torch.long),
        ]
        encoder = NeuronEncoder(model)
        self.encoder = torch_neuron.trace(encoder, inputs)
```

(continues on next page)
# Trace the decoder (with expanded inputs)
batch_size = num_texts * num_beams
inputs = (
    torch.ones((batch_size, max_decoder_length), dtype=torch.long),
    torch.ones((batch_size, max_encoder_length), dtype=torch.long),
    torch.ones((batch_size, max_encoder_length, model.config.d_˓
    model), dtype=torch.float),
    torch.tensor(0),
)
decoder = NeuronDecoder(model, max_decoder_length)
self.decoder = torch_neuron.trace(decoder, inputs)

# Beam Search Methods (Copied directly from transformers)

# Encoder/Decoder Invocation

def adjust_logits_during_generation(self, logits, cur_len, max_length):
    if cur_len == 1 and self.config.force_bos_token_to_be_generated:
        self._force_token_id_to_be_generated(logits, self.config.bos_˓
token_id)
    elif cur_len == max_length - 1 and self.config.eos_token_id ˓
is not None:
        self._force_token_id_to_be_generated(logits, self.config.eos_˓
token_id)
    return logits

@staticmethod
def _force_token_id_to_be_generated(scores, token_id) -> None:
    scores[:, [x for x in range(scores.shape[1]) if x != token_id]] = -float("inf")

# Encoder/Decoder Invocation

def prepare_inputs_for_generation(self, decoder_input_ids, encoder_outputs=None,
                                   attention_mask=None, **model_kwargs):
    # Pad the inputs for Neuron
    current_length = decoder_input_ids.shape[1]
    pad_size = self.config.max_decoder_length - current_length
    return dict(
        input_ids=F.pad(decoder_input_ids, (0, pad_size)),
        attention_mask=attention_mask,
        encoder_outputs=encoder_outputs.last_hidden_state,
        current_length=torch.tensor(current_length - 1),
    )
```python
def get_encoder(self):
    """Helper to invoke the encoder and wrap the results in the expected structure""
    def encode(input_ids, attention_mask, **kwargs):
        output, = self.encoder(input_ids, attention_mask)
        return BaseModelOutput(last_hidden_state=output,
    )
    return encode

def __call__(self, input_ids, attention_mask, encoder_outputs, current_length, **kwargs):
    """Helper to invoke the decoder and wrap the results in the expected structure""
    logits = self.decoder(input_ids, attention_mask, encoder_outputs, current_length)
    return Seq2SeqLMOutput(logits=logits)

# Serialization

@classmethod
def save_pretrained(cls, directory):
    if os.path.isfile(directory):
        print(f"Provided path ({directory}) should be a directory, not a file")
    return os.makedirs(directory, exist_ok=True)
    torch.jit.save(cls.encoder, os.path.join(directory, 'encoder.pt'))
    torch.jit.save(cls.decoder, os.path.join(directory, 'decoder.pt'))
    cls.config.save_pretrained(directory)

@classmethod
def from_pretrained(cls, directory):
    config = MarianConfig.from_pretrained(directory)
    obj = cls(config)
    obj.encoder = torch.jit.load(os.path.join(directory, 'encoder.pt'))
    obj.decoder = torch.jit.load(os.path.join(directory, 'decoder.pt'))
    return obj

@property
def device(self):
    return torch.device('cpu')
```
### Execution

Using everything together from above, now the process to deploy the model is as follows:

1. Compile the model
2. Serialize an artifact
3. Load the serialized artifact
4. Execute the model Neuron

```python
# This is the name of the folder where the artifacts will be stored on disk
neuron_name = 'NeuronMarianMT'

model_neuron = NeuronGeneration(model_cpu.config)

# 1. Compile the model
# Note: This may take a couple of minutes since both the encoder/decoder will be compiled
model_neuron.trace(
    model=model_cpu,
    num_texts=num_texts,
    num_beams=num_beams,
    max_encoder_length=max_encoder_length,
    max_decoder_length=max_decoder_length,
)

# 2. Serialize an artifact
# After this call you will have an `encoder.pt`, `decoder.pt` and `config.json` in the neuron_name folder
model_neuron.save_pretrained(neuron_name)

# 3. Load the serialized artifact
model_neuron = NeuronGeneration.from_pretrained(neuron_name)

# 4. Execute the model Neuron
infer(model_neuron, tokenizer_cpu, sample_text)
```

Comparing the Neuron execution to the original CPU implementation, you will see the exact same generated text.

```python
# CPU execution for comparison
infer(model_cpu, tokenizer_cpu, sample_text)
```
Appendix - BART (Mask Filling Task)

These NeuronGeneration class can be applied to the BART model for the task of filling in mask tokens.

```python
from transformers import BartForConditionalGeneration, BartTokenizer
bart_name = "facebook/bart-large"
bart_model = BartForConditionalGeneration.from_pretrained(bart_name, force_bos_token_to_be_generated=True)
bart_tokenizer = BartTokenizer.from_pretrained(bart_name)
bart_text = "UN Chief Says There Is No <mask> in Syria"

# CPU Execution
infer(bart_model, bart_tokenizer, bart_text)

# Neuron Execution
bart_neuron = NeuronGeneration(bart_model.config)
bart_neuron.trace(
    model=bart_model,
    num_texts=num_texts,
    num_beams=num_beams,
    max_encoder_length=max_encoder_length,
    max_decoder_length=max_decoder_length,
)
infer(bart_neuron, bart_tokenizer, bart_text)
```

Appendix - Pegasus (Summarization Task)

These NeuronGeneration class can be applied to the Pegasus model for summarization.

```python
from transformers import PegasusForConditionalGeneration, PegasusTokenizer
pegasus_name = 'google/pegasus-xsum'
pegasus_model = PegasusForConditionalGeneration.from_pretrained(pegasus_name)
pegasus_tokenizer = PegasusTokenizer.from_pretrained(pegasus_name)
pegasus_text = "PG&E stated it scheduled the blackouts in response to forecasts for high winds amid dry conditions. The aim is to reduce the risk of wildfires."

# CPU Execution
infer(pegasus_model, pegasus_tokenizer, pegasus_text)

# Neuron Execution
pegasus_neuron = NeuronGeneration(pegasus_model.config)
pegasus_neuron.trace(
    model=pegasus_model,
    num_texts=num_texts,
    num_beams=num_beams,
    max_encoder_length=max_encoder_length,
    max_decoder_length=max_decoder_length,
)
infer(pegasus_neuron, pegasus_tokenizer, pegasus_text)
```
Utilizing Neuron Capabilities

- BERT TorchServe tutorial [html]
- NeuronCore Pipeline tutorial [html] [notebook]

BERT TorchServe Tutorial

Table of Contents

- Overview
- Run the tutorial
- Setup TorchServe
- Run TorchServe
- Benchmark TorchServe

Overview

This tutorial demonstrates the use of TorchServe with Neuron, the SDK for Amazon Inf1 instances. By the end of this tutorial, you will understand how TorchServe can be used to serve a model backed by EC2 Inf1 instances. We will use a pretrained BERT-Base model to determine if one sentence is a paraphrase of another.

Run the tutorial

First run the HuggingFace Pretrained BERT tutorial [html] [notebook].

You should now have a compiled bert_neuron_b6.pt file, which is required going forward.

Open a shell on the instance you prepared earlier, create a new directory named torchserve. Copy your compiled model from the previous tutorial into this new directory.

```
$ cd torchserve
$ ls
```

```
bert_neuron_b6.pt
```

Prepare a new Python virtual environment with the necessary Neuron and TorchServe components. Use a virtual environment to keep (most of) the various tutorial components isolated from the rest of the system in a controlled way.

```
$ python3 -m venv env
$ . env/bin/activate
$ pip install -U pip
$ pip install torch-neuron 'neuron-cc[tensorflow]' --extra-index-url=https://pip.repos.neuron.amazonaws.com
$ pip install transformers==4.12.5 torchserve==0.5.0 torch-model-archiver==0.5.0
```

Install the system requirements for TorchServe.
$ sudo apt install openjdk-11-jdk
$ java -version

openjdk 11.0.11 2021-04-20
OpenJDK Runtime Environment (build 11.0.11+9-Ubuntu-0ubuntu2.18.04)
OpenJDK 64-Bit Server VM (build 11.0.11+9-Ubuntu-0ubuntu2.18.04, mixed mode, sharing)

$ javac -version

javac 11.0.11

Verify that TorchServe is now available.

$ torchserve --version

TorchServe Version is 0.5.0

Setup TorchServe

During this tutorial you will need to download various files onto your instance. The simplest way to accomplish this is to paste the download links provided above each file into a `wget` command. (We don’t provide the links directly because they are subject to change.) For example, right-click and copy the download link for `config.json` shown below.

Listing 1: config.json

```json
{
   "model_name": "bert-base-cased-finetuned-mrpc",
   "max_length": 128,
   "batch_size": 6
}
```

Now execute the following in your shell:

```
$ wget <paste link here>
$ ls
```

```
bert_neuron_b6.pt  config.json
```

Download the custom handler script that will eventually respond to inference requests.

Listing 2: handler_bert.py

```python
import os
import json
import sys
import logging

import torch, torch_neuron
from transformers import AutoTokenizer
from abc import ABC
from ts.torch_handler.base_handler import BaseHandler
```

(continues on next page)
# one core per worker

```python
os.environ['NEURONCORE_GROUP_SIZES'] = '1'
```

```
logger = logging.getLogger(__name__)
```

```python
class BertEmbeddingHandler(BaseHandler, ABC):
    
    Handler class for Bert Embedding computations.
    
    __init__(self):
        super(BertEmbeddingHandler, self).__init__()
        self.initialized = False

    initialize(self, ctx):
        self.manifest = ctx.manifest
        properties = ctx.system_properties
        self.device = 'cpu'
        model_dir = properties.get('model_dir')
        serialized_file = self.manifest['model']['serializedFile']
        model_pt_path = os.path.join(model_dir, serialized_file)

        # point sys.path to our config file
        with open('config.json') as fp:
            config = json.load(fp)
            self.max_length = config['max_length']
            self.batch_size = config['batch_size']
            self.classes = ['not paraphrase', 'paraphrase']

            self.model = torch.jit.load(model_pt_path)
            logger.debug(f'Model loaded from {model_dir}')
            self.model.to(self.device)
            self.model.eval()

            self.tokenizer = AutoTokenizer.from_pretrained(config['model_name'])
            self.initialized = True

    preprocess(self, input_data):
        
        Tokenization pre-processing
        
        input_ids = []
        attention_masks = []
        token_type_ids = []
        for row in input_data:
            seq_0 = row['seq_0'].decode('utf-8')
            seq_1 = row['seq_1'].decode('utf-8')
            logger.debug(f'Received text: 
```
input_ids.append(inputs['input_ids'])
attention_masks.append(inputs['attention_mask'])
token_type_ids.append(inputs['token_type_ids'])

batch = (torch.cat(input_ids, 0),
         torch.cat(attention_masks, 0),
         torch.cat(token_type_ids, 0))

return batch

def inference(self, inputs):
    """
    Predict the class of a text using a trained transformer model.
    """

    # sanity check dimensions
    assert(len(inputs) == 3)
    num_inferences = len(inputs[0])
    assert(num_inferences <= self.batch_size)

    # insert padding if we received a partial batch
    padding = self.batch_size - num_inferences
    if padding > 0:
        pad = torch.nn.ConstantPad1d((0, 0, 0, padding), value=0)
        inputs = [pad(x) for x in inputs]

    outputs = self.model(*inputs)[0]
    predictions = []
    for i in range(num_inferences):
        prediction = self.classes[outputs[i].argmax().item()]
        predictions.append([prediction])
        logger.debug("Model predicted: " + prediction)
    return predictions

def postprocess(self, inference_output):
    return inference_output

Next, we need to associate the handler script with the compiled model using torch-model-archiver. Run the following commands in your terminal:

```
$ mkdir model_store
$ MAX_LENGTH=$(jq '.max_length' config.json)
$ BATCH_SIZE=$(jq '.batch_size' config.json)
$ MODEL_NAME=bert-max_length$MAX_LENGTH-batch_size$BATCH_SIZE
$ torch-model-archiver --model-name "$MODEL_NAME" --version 1.0 --serialized-file ./bert_neuron_b6.pt --handler "./handler_bert.py" --extra-files "/config.json" --export-path model_store
```

**Note:** If you modify your model or a dependency, you will need to rerun the archiver command with the `-f` flag appended to update the archive.

The result of the above will be a mar file inside the model_store directory.
This file is essentially an archive associated with a fixed version of your model along with its dependencies (e.g. the handler code).

**Note:** The version specified in the `torch-model-archiver` command can be appended to REST API requests to access a specific version of your model. For example, if your model was hosted locally on port 8080 and named “bert”, the latest version of your model would be available at `http://localhost:8080/predictions/bert`, while version 1.0 would be accessible at `http://localhost:8080/predictions/bert/1.0`. We will see how to perform inference using this API in Step 6.

Create a custom `config` file to set some parameters. This file will be used to configure the server at launch when we run `torchserve --start`.

**Listing 3:** `torchserve.config`

```
# bind inference API to all network interfaces with SSL enabled
inference_address=http://0.0.0.0:8080
default_workers_per_model=1
```

**Note:** This will cause TorchServe to bind on all interfaces. For security in real-world applications, you’ll probably want to use port 8443 and enable SSL.

**Run TorchServe**

It’s time to start the server. Typically we’d want to launch this in a separate console, but for this demo we’ll just redirect output to a file.

```
$ torchserve --start --ncs --model-store model_store --ts-config torchserve.config 2>&1 >torchserve.log
```

Verify that the server seems to have started okay.

```
$ curl http://127.0.0.1:8080/ping
```

```
{
  "status": "Healthy"
}
```

**Note:** If you get an error when trying to ping the server, you may have tried before the server was fully launched. Check `torchserve.log` for details.

Use the Management API to instruct TorchServe to load our model.
$ \text{MAX\_BATCH\_DELAY}=5000 \ # \ ms \ timeout \ before \ a \ partial \ batch \ is \ processed

$ \text{INITIAL\_WORKERS}=4 \ # \ number \ of \ models \ that \ will \ be \ loaded \ at \ launch

$ \text{curl} \ -\text{X} \ \text{POST} \ \text{"http://localhost:8081/models?url=$MODEL\_NAME.mar&batch\_size= \text{\$BATCH\_SIZE}&initial\_workers=$INITIAL\_WORKERS&max\_batch\_delay=$MAX\_BATCH\_DELAY"}$

```
{
  "status": "Model \"bert-max_length128-batch_size6\" Version: 1.0
  \registered \with \4 \initial \workers"
}
```

**Note:** Any additional attempts to configure the model after the initial curl request will cause the server to return a 409 error. You'll need to stop/start/configure the server to realize any changes.

The **MAX\_BATCH\_DELAY** is a timeout value that determines how long to wait before processing a partial batch. This is why the handler code needs to check the batch dimension and potentially add padding. TorchServe will instantiate the number of model handlers indicated by **INITIAL\_WORKERS**, so this value controls how many models we will load onto Inferentia in parallel. This tutorial was performed on an inf1.xlarge instance (one Inferentia chip), so there are four NeuronCores available. If you want to control worker scaling more dynamically, see the docs.

**Warning:** If you attempt to load more models than NeuronCores available, one of two things will occur. Either the extra models will fit in device memory but performance will suffer, or you will encounter an error on your initial inference. You shouldn't set **INITIAL\_WORKERS** above the number of NeuronCores. However, you may want to use fewer cores if you are using the **NeuronCore Pipeline** feature.

It looks like everything is running successfully at this point, so it's time for an inference.

Create the **infer_bert.py** file below on your instance.

```python
import json
import concurrent.futures
import requests

with open('config.json') as fp:
    config = json.load(fp)
max_length = config['max_length']
batch_size = config['batch_size']
name = f'bert-max_length{max_length}-batch_size{batch_size}'

# dispatch requests in parallel
url = f'http://localhost:8080/predictions/{name}'
paraphrase = {'seq_0': "HuggingFace's headquarters are situated in Manhattan \""},
               'seq_1': "The company HuggingFace is based in New York City"
not_paraphrase = {'seq_0': paraphrase['seq_0'], 'seq_1': "This is total\"nonsense.'}

with concurrent.futures.ThreadPoolExecutor(max_workers=batch_size) as executor:
    (continues on next page)
```python
def worker_thread(worker_index):
    # we'll send half the requests as not_paraphrase examples for sanity
    data = paraphrase if worker_index < batch_size//2 else not_paraphrase
    response = requests.post(url, data=data)
    print(worker_index, response.json())
    
    for worker_index in range(batch_size):
        executor.submit(worker_thread, worker_index)
```

This script will send a `batch_size` number of requests to our model. In this example, we are using a model that estimates the probability that one sentence is a paraphrase of another. The script sends positive examples in the first half of the batch and negative examples in the second half.

Execute the script in your terminal.

```bash
$ python infer_bert.py
```

```
[{'paraphrase'},
 {'not paraphrase'},
 {'not paraphrase'},
 {'paraphrase'},
 {'not paraphrase'},
 {'paraphrase'}]
```

We can see that the first three threads (0, 1, 2) all report `paraphrase`, as expected. If we instead modify the script to send an incomplete batch and then wait for the timeout to expire, the excess padding results will be discarded.

**Benchmark TorchServe**

We’ve seen how to perform a single batched inference, but how many inferences can we process per second? A separate upcoming tutorial will document performance tuning to maximize throughput. In the meantime, we can still perform a simple naïve stress test. The code below will spawn 64 worker threads, with each thread repeatedly sending a full batch of data to process. A separate thread will periodically print throughput and latency measurements.

```
import os
import argparse
import time
import numpy as np
import requests
import sys
from concurrent import futures
import torch

parser = argparse.ArgumentParser()
parser.add_argument('--url', help='Torchserve model URL', type=str, default=f'http://127.0.0.1:8080/predictions/bert-max_length128-batch_size6')
parser.add_argument('--num_thread', type=int, default=64, help='Number of threads invoking the model URL')
parser.add_argument('--batch_size', type=int, default=6)
```

(continues on next page)
parser.add_argument('--sequence_length', type=int, default=128)
parser.add_argument('--latency_window_size', type=int, default=1000)
parser.add_argument('--throughput_time', type=int, default=300)
parser.add_argument('--throughput_interval', type=int, default=10)
args = parser.parse_args()

data = { 'seq_0': 'A completely made up sentence.', 'seq_1': 'Well, I suppose they are all made up.' }

live = True
num_infer = 0
latency_list = []

def one_thread(pred, feed_data):
    global latency_list
    global num_infer
    global live
    session = requests.Session()
    while True:
        start = time.time()
        result = session.post(pred, data=feed_data)
        latency = time.time() - start
        latency_list.append(latency)
        num_infer += 1
        if not live:
            break

def current_performance():
    last_num_infer = num_infer
    for _ in range(args.throughput_time // args.throughput_interval):
        current_num_infer = num_infer
        throughput = (current_num_infer - last_num_infer) / args.throughput_interval
        p50 = 0.0
        p90 = 0.0
        if latency_list:
            p50 = np.percentile(latency_list[-args.latency_window_size:], 50)
            p90 = np.percentile(latency_list[-args.latency_window_size:], 90)
            print('pid {} current throughput {:.3f}, latency p50={:.3f} p90={:.3f}'.format(os.getpid(), throughput, p50, p90))
    sys.stdout.flush()
    last_num_infer = current_num_infer
    time.sleep(args.throughput_interval)
    global live
    live = False

with futures.ThreadPoolExecutor(max_workers=args.num_thread+1) as executor:
    executor.submit(current_performance)
    for _ in range(args.num_thread):
        executor.submit(one_thread, args.url, data)

Run the benchmarking script.

$ python benchmark_bert.py

pid 26980: current throughput 0.0, latency p50=0.000 p90=0.000
pid 26980: current throughput 584.1, latency p50=0.099 p90=0.181

(continues on next page)
Congratulations! By now you should have successfully served a batched model over TorchServe.

Using NeuronCore Pipeline with PyTorch

In this tutorial you compile a pretrained BERT base model from HuggingFace Transformers, using the NeuronCore Pipeline feature of the AWS Neuron SDK. You benchmark model latency of the pipeline parallel mode and compare with the usual data parallel (multi-worker) deployment.

This tutorial is intended to run in an inf1.6xlarge, running the latest AWS Deep Learning AMI (DLAMI). The inf1.6xlarge instance size has AWS Inferentia chips for a total of 16 NeuronCores.

Before continuing, verify that this Jupyter notebook is running conda_aws_neuron_pytorch_p36 kernel of the DLAMI. You can select the Kernel from the “Kernel -> Change Kernel” option on the top of this Jupyter notebook page. If you are using your own AMI, follow these instructions to set up your environment.

Note: Do not execute this tutorial using “Run -> Run all cells” option.

Install Dependencies:

This tutorial requires the following pip packages:

• torch-neuron
• neuron-cc[tensorflow]
• transformers

Most of these packages will be installed when configuring your environment using the Neuron PyTorch setup guide. The additional HuggingFace Transformers dependency must be installed here.

```
[ ]: !pip install --upgrade "transformers==4.6.0"
```

Compiling a BERT base model for a single NeuronCore

To run a HuggingFace BERTModel on Inferentia, you only need to add a single extra line of code to the usual Transformers PyTorch implementation, after importing the torch_neuron framework.

Add the argument `return_dict=False` to the BERT transformers model so it can be traced with TorchScript. TorchScript is a way to create serializable and optimizable models from PyTorch code.

Enable padding to a maximum sequence length of 128, to test the model’s performance with a realistic payload size. You can adapt this sequence length to your application’s requirement.

You can adapt the original example on the BertModel forward pass docstring according to the following cell
import torch
import torch_neuron
from transformers import BertTokenizer, BertModel
from joblib import Parallel, delayed
import numpy as np
from tqdm import tqdm
import os
import time

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased', return_dict=False)

inputs = tokenizer("Hello, my dog is cute", return_tensors="pt", max_length=128, padding='max_length', truncation=True)

The one extra line required is the call to torch.neuron.trace() method. This call compiles the model and returns the forward method of the torch.nn.Model method, which you can use to run inference.

The compiled graph can be saved using the torch.jit.save function and restored using torch.jit.load function for inference on Inf1 instances. During inference, the previously compiled artifacts will be loaded into the Neuron Runtime for inference execution.

neuron_model = torch.neuron.trace(model, example_inputs=(inputs['input_ids'], inputs['attention_mask']), verbose=1)

Running the BERT base model on a single NeuronCore

With the model already available in memory, you can time one execution and check for the latency on the single inference call. You will load the model into Inferentia with a single inference call. A large “wall time” is expected when you first run the next cell, running the cell twice will show the actual inference latency:

outputs = neuron_model(*{inputs['input_ids'], inputs['attention_mask']})

You can also check for the throughput of the single model running on a single NeuronCore.

The sequential inference test (for loop) does not measure all the performance one can achieve in an instance with multiple NeuronCores. To improve hardware utilization you can run parallel inference requests over multiple model workers, which you'll test in the Data Parallel Bonus Section below.

outputs = neuron_model(*{inputs['input_ids'], inputs['attention_mask']})

Save the compiled model for later use:
Compiling a BERT base model for 16 NeuronCores

Our next step is to compile the same model for all 16 NeuronCores available in the inf1.6xlarge and check the performance difference when running pipeline parallel inferences.

Prior to compiling and executing the model, use the following cell to restart your IPython Kernel.

**Note:** If you run this notebook using Jupyter Notebooks, instead of Jupyterlab, you may need to restart the kernel using the “Kernel -> Restart” option, after running the next cell

```python
import IPython
# Automatically restarts kernel
IPython.Application.instance().kernel.do_shutdown(True)
```

After you get {'status': 'ok', 'restart': True}, reinstantiate your environment with the required libraries and the BertTokenizer and BertModel from Transformers.

```python
import torch
import torch_neuron
from transformers import BertTokenizer, BertModel
from joblib import Parallel, delayed
import numpy as np
from tqdm import tqdm
import os
import time

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased',return_dict=False)
inputs = tokenizer("Hello, my dog is cute",return_tensors="pt",max_length=128,padding='max_length',truncation=True)

To enable pipeline mode during compilation, you need only to add the compiler flag --neuroncore-pipeline-cores and set the number of desired cores. The cell below sets up a neuroncore_pipeline_cores string, which you can set for the available number of NeuronCores on the instance: *inf1.6xlarge* has 16 NeuronCores in 4 Inferentia chips.

```python
# Number of Cores in the Pipeline Mode
neuroncore_pipeline_cores = 16 # This string should be '4' on an inf1.xlarge
# Compiling for neuroncore-pipeline-cores='16'
neuron_pipeline_model = torch.neuron.trace(model,
    example_inputs = (inputs['input_ids'],inputs['attention_mask']),
    verbose=1,
    compiler_args = ['--neuroncore-pipeline-cores', str(neuroncore_pipeline_cores)]
)
```
Running the BERT base model on 16 NeuronCores

Next, time one execution and check for the latency on the single inference call over 16 cores. You will load the model into Inferentia with a single inference call. A large “wall time” is expected when you first run the next cell, running the cell twice will show the actual inference latency:

```py
%time
# The following line tests inference and should be executed on Inf1 instance family.
outputs = neuron_pipeline_model(*(inputs['input_ids'],inputs['attention_mask']
˓→))
```

Check also for the throughput of the single model running over a 16 NeuronCores.

The sequential inference test (for loop) does not measure all the performance one can achieve with Pipeline mode. As the inference runs in streaming fashion, at least 15 cores are waiting for a new call until the last one processes the first call. This results in low NeuronCore utilization. To improve hardware utilization you will require parallel inference requests, which you’ll test in the next section.

```py
for _ in tqdm(range(100)):
    outputs = neuron_pipeline_model(*(inputs['input_ids'],inputs['attention_mask']
˓→))
```

Load Testing the Pipeline Parallel Mode

To put the 16 NeuronCores group to test, a client has to run concurrent requests to the model. In this Notebook setup you achieve it by creating a thread pool with `Joblib.Parallel`, with all workers on the pool running one inference call.

You can define a new method called `inference_latency()` so that you measure the amount of time each inference calls take.

```py
def inference_latency(model,*inputs):

    """
    inference_time is a simple method to return the latency of a model inference.

    Parameters:
    model: torch model object loaded using torch.jit.load
    inputs: model() args

    Returns:
    latency in seconds
    """
    start = time.time()
    _ = model(*inputs)
    return time.time() - start
```

Use `tqdm` to measure total throughput of your experiment, with a nice side-effect of “cool progress bar!”.

The total throughput is expected to be high, so set your experiment range to a large number, here 30k inferences.

To calculate the latency statistics over the returned 30k list of latencies use `numpy.quantile()` method.
```python
import torch
import torch_neuron
from transformers import BertTokenizer
from joblib import Parallel, delayed
import numpy as np
import os
import time

def inference_latency(model, *inputs):
    """
    inference_time is a simple method to return the latency of a model.
    Parameters:
    model: torch model object loaded using torch.jit.load
    inputs: model() args
    """
    inference_time = Parallel(n_jobs=12, prefer="threads")
    latencies = Parallel(n_jobs=12, prefer="threads")
    inference_latencies = Parallel(n_jobs=12, prefer="threads")
    for i in tqdm(range(30000), position=0, leave=True):
        latency = Parallel(n_jobs=12, prefer="threads")
        latencies = Parallel(n_jobs=12, prefer="threads")
        inference_latencies = Parallel(n_jobs=12, prefer="threads")
        inference_time(model, *inputs, *inputs) for i in tqdm(range(30000), position=0, leave=True)
        p50 = np.quantile(latency[-10000:], 0.50) * 1000
        p95 = np.quantile(latency[-10000:], 0.95) * 1000
        p99 = np.quantile(latency[-10000:], 0.99) * 1000
        avg_throughput = t.total/t.format_dict['elapsed']
        print(f'Avg Throughput: {avg_throughput:.1f}')
        print(f'50th Percentile Latency: {p50:.1f} ms')
        print(f'95th Percentile Latency: {p95:.1f} ms')
        print(f'99th Percentile Latency: {p99:.1f} ms')

Save compile model for later use:
```
Returns:
    latency in seconds

```python
start = time.time()
_ = model(*inputs)
return time.time() - start
```

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
inputs = tokenizer("Hello, my dog is cute",return_tensors="pt",max_length=128,padding='max_length',truncation=True)

You use the 'NEURONCORE_GROUP_SIZES' environment variable to define NeuronCore groups that will each load a single model at runtime. Set the environment variable to the number of individual workers you want to test in parallel.

```
torch_neuron will load one model per NeuronCore group until it runs out of cores. At that point, if the Python process continues to spawn more model objects using `torch.jit.load`, `torch_neuron` will start stacking more than one model per core, until the Inferentia chip memory is full.

Inferentia is able to run inference over all the loaded models, but only one at a time. The Neuron Runtime takes care of dynamically switching the model context as requests come in, no extra worker process management required. Use 1 model per NeuronCore to achieve maximum performance.

The following cell creates a list with as many models as NeuronCore Groups and execute one single dummy inference to load the models into Inferentia.

```python
import warnings
# Number of data parallel workers
number_of_workers=16 # This number should be 4 on an inf1.xlarge

# Setting up a data parallel group
warnings.warn("NEURONCORE_GROUP_SIZES is being deprecated, if your application is using NEURONCORE_GROUP_SIZES please see https://awsdocs-neuron.readthedocs-hosted.com/en/latest/release-notes/deprecation.html#announcing-end-of-support-for-neuroncore-group-sizes for more details.", DeprecationWarning)
os.environ['NEURONCORE_GROUP_SIZES'] = ",".join(["1"]*number_of_workers)

# Loading 'number_of_workers' amount of models in Python memory
model_list = [torch.jit.load('bert-base-uncased-neuron.pt') for _ in range(number_of_workers)]

# Dummy inference to load models to Inferentia
_ = [mod(*(inputs['input_ids'],inputs['attention_mask'])) for mod in model_list]
```

Adapt the call to `joblib.Parallel()` iterating over a concatenated version of the `model_list`, to run 'round-robin' calls to each of the model workers.

```python
import warnings  # Number of data parallel workers
number_of_workers=16 # This number should be 4 on an inf1.xlarge

# Setting up a data parallel group
warnings.warn("NEURONCORE_GROUP_SIZES is being deprecated, if your application is using NEURONCORE_GROUP_SIZES please see https://awsdocs-neuron.readthedocs-hosted.com/en/latest/release-notes/deprecation.html#announcing-end-of-support-for-neuroncore-group-sizes for more details.", DeprecationWarning)
os.environ['NEURONCORE_GROUP_SIZES'] = ",".join(["1"]*number_of_workers)

# Loading 'number_of_workers' amount of models in Python memory
model_list = [torch.jit.load('bert-base-uncased-neuron.pt') for _ in range(number_of_workers)]

# Dummy inference to load models to Inferentia
_ = [mod(*(inputs['input_ids'],inputs['attention_mask'])) for mod in model_list]
```
For this model, despite the larger number of workers, the per-worker latency increases when running a single model per core, which in turn reduces the total throughput.

This behavior may not repeat if the model memory footprint or the input payload size changes, i.e. batch size > 1. We encourage you to experiment with the data parallel and pipeline parallel modes to optimize your application performance.

2.2.2 PyTorch-Neuron trace python API

The PyTorch-Neuron trace Python API provides a method to generate PyTorch models for execution on Inferentia, which can be serialized as TorchScript. It is analogous to torch.jit.trace() function in PyTorch.

torch_neuron.trace(model, example_inputs, **kwargs)

The torch_neuron.trace() method sends operations to the Neuron-Compiler (neuron-cc) for compilation and embeds compiled artifacts in a TorchScript graph.

Compilation can be done on any EC2 machine with sufficient memory and compute resources. c5.4xlarge or larger is recommended.

Options can be passed to Neuron compiler via the compile function. See Neuron compiler CLI Reference Guide for more information about compiler options.

This function partitions nodes into operations that are supported by Neuron and operations which are not. Operations which are not supported by Neuron are run on CPU. Graph partitioning can be controlled by the subgraph builder function, minimum segment size, and fallback parameters (See below). By default all supported operations are compiled and run on Neuron.

The compiled graph can be saved using the torch.jit.save() function and restored using torch.jit.load() function for inference on Inf1 instances. During inference, the previously compiled artifacts will be loaded into the Neuron Runtime for inference execution.

Required Arguments

Parameters

- **model** (Module, callable) – The functions that that will be run with example_inputs arguments. The arguments and return types must compatible with torch.jit.trace(). When a Module is passed to torch_neuron.trace(), only the forward() method is run and traced.

- **example_inputs** (tuple) – A tuple of example inputs that will be passed to the model while tracing. The resulting trace can be run with inputs of different types and shapes assuming the traced operations support those types and shapes. This
optional keyword arguments

parameters

- **compiler_args** *(list[str])* - List of strings representing neuron-cc compiler arguments. Note that these arguments apply to all subgraphs generated by allowlist partitioning. For example, use `compiler_args=['--neuroncore-pipeline-cores', '4']` to set number of NeuronCores per subgraph to 4. See Neuron compiler CLI Reference Guide for more information about compiler options.

- **compiler_timeout** *(int)* - Timeout in seconds for waiting neuron-cc to complete. Exceeding this timeout will cause a subprocess.TimeoutExpired exception.

- **compiler_workdir** *(str)* - Work directory used by neuron-cc. Useful for debugging and/or inspecting neuron-cc logs/IRs.

- **subgraph_builder_function** *(callable)* - A function which is evaluated on each node during graph partitioning. This takes in a torch graph operator node and returns a `bool` value of whether it should be included in the fused Neuron graph or not. By default the partitioner selects all operators which are supported by Neuron.

- **minimum_segment_size** *(int)* - A parameter used during partitioning. This specifies the minimum number of graph nodes which should be compiled into a Neuron graph (default=2). If the number of nodes is smaller than this size, the operations will run on CPU.

- **fallback** *(bool)* - A function parameter to turn off graph partitioning. Indicates whether to attempt to fall back to CPU operations if an operation is not supported by Neuron. By default this is `True`. If this is set to `False` and an operation is not supported by Neuron, this will fail compilation and raise an `AttributeError`.

- **dynamic_batch_size** *(bool)* - A flag to allow Neuron graphs to consume variable sized batches of data. Dynamic sizing is restricted to the 0th dimension of a tensor.

- **optimizations** *(list)* - A list of Optimization passes to apply to the model.

- ****kwargs** - All other keyword arguments will be forwarded directly to torch.jit.trace(). This supports flags like `strict=False` in order to allow dictionary outputs.

returns

The traced `ScriptModule` with embedded compiled neuron sub-graphs. Operations in this module will run on Neuron unless they are not supported by Neuron or manually partitioned to run on CPU.

Note that in torch<1.8 This would return a `ScriptFunction` if the input was function type.

return type  `ScriptModule, ScriptFunction`

**class** `torch_neuron.Optimization`

A set of optimization passes that can be applied to the model.
FLOAT32_TO_FLOAT16
A post-processing pass that converts all torch.float32 tensors to torch.float16 tensors. The advantage to this optimization pass is that input/output tensors will be type cast. This reduces the amount of data that will be copied to and from Inferentia hardware. The resulting traced model will accept both torch.float32 and torch.float16 inputs where the model used torch.float32 inputs during tracing. It is only beneficial to enable this optimization if the throughput of a model is highly dependent upon data transfer speed. This optimization is not recommended if the final application will use torch.float32 inputs since the torch.float16 type cast will occur on CPU during inference.

Example Usage

Function Compilation

```python
import torch
import torch_neuron

def foo(x, y):
    return 2 * x + y

# Run 'foo' with the provided inputs and record the tensor operations
traced_foo = torch.neuron.trace(foo, (torch.rand(3), torch.rand(3)))

# 'traced_foo' can now be run with the TorchScript interpreter or saved
# and loaded in a Python-free environment
torch.jit.save(traced_foo, 'foo.pt')
traced_foo = torch.jit.load('foo.pt')
```

Module Compilation

```python
import torch
import torch_neuron
import torch.nn as nn

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv = nn.Conv2d(1, 1, 3)

    def forward(self, x):
        return self.conv(x) + 1

n = Net()
n.eval()

inputs = torch.rand(1, 1, 3, 3)

# Trace a specific method and construct `ScriptModule` with
# a single `forward` method
neuron_forward = torch.neuron.trace(n.forward, inputs)

# Trace a module (implicitly traces `forward`) and constructs a
# `ScriptModule` with a single `forward` method
neuron_net = torch.neuron.trace(n, inputs)
```
Pre-Trained Model Compilation

The following is an example usage of the compilation Python API, with default compilation arguments, using a pretrained `torch.nn.Module`:

```python
import torch
import torch_neuron
from torchvision import models

# Load the model and set it to evaluation mode
model = models.resnet50(pretrained=True)
model.eval()

# Compile with an example input
image = torch.rand([1, 3, 224, 224])
model_neuron = torch.neuron.trace(model, image)
```

Compiling models with `torch.jit.trace` kwargs

This example uses the `strict=False` flag to compile a model with dictionary outputs. Similarly, any other keyword argument of `torch.jit.trace()` can be passed directly to `torch_neuron.trace()` so that it is passed to the underlying trace call.

```python
import torch
import torch_neuron
import torch.nn as nn

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv = nn.Conv2d(1, 1, 3)

    def forward(self, x):
        return {'conv': self.conv(x) + 1}

model = Model()
model.eval()

inputs = torch.rand(1, 1, 3, 3)
# use the strict=False kwarg to compile a model with dictionary outputs
# the model output format does not change
model_neuron = torch.neuron.trace(model, inputs, strict=False)
```

Dynamic Batching

This example uses the optional `dynamic_batch_size` option in order to support variable sized batches at inference time.

```python
import torch
import torch_neuron
from torchvision import models
```

(continues on next page)
# Load the model and set it to evaluation mode
model = models.resnet50(pretrained=True)
model.eval()

# Compile with an example input of batch size 1
image = torch.rand([1, 3, 224, 224])
model_neuron = torch.neuron.trace(model, image, dynamic_batch_size=True)

# Execute with a batch of 7 images
batch = torch.rand([7, 3, 224, 224])
results = model_neuron(batch)

## Manual Partitioning

The following example uses the optional `subgraph_builder_function` parameter to ensure that only a specific convolution layer is compiled to Neuron. The remaining operations are executed on CPU.

```python
import torch
import torch_neuron
import torch.nn as nn

class ExampleConvolutionLayer(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv = nn.Conv2d(1, 1, 3)

    def forward(self, x):
        return self.conv(x) + 1

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.layer = ExampleConvolutionLayer()

    def forward(self, x):
        return self.layer(x) * 100

def subgraph_builder_function(node) -> bool:
    """Select if the node will be included in the Neuron graph""

    # Node names are tuples of Module names.
    if 'ExampleConvolutionLayer' in node.name:
        return True
    # Ignore all operations not in the example convolution layer
    return False

model = Model()
model.eval()

inputs = torch.rand(1, 1, 3, 3)

# Log output shows that 'aten::_convolution' and 'aten::add' are compiled
# but 'aten::mul' is not. This will seamlessly switch between Neuron/CPU
# execution in a single graph.
```
2.2.3 torch.neuron.DataParallel API

The `torch.neuron.DataParallel()` Python API implements data parallelism on `ScriptModule` models created by the PyTorch-Neuron trace python API. This function is analogous to `DataParallel` in PyTorch. The Data Parallel Inference on Torch Neuron application note provides an overview of how `torch.neuron.DataParallel()` can be used to improve the performance of inference workloads on Inferentia.

```python
neuron_model = torch_neuron.trace(
    model,
    inputs,
    subgraph_builder_function=subgraph_builder_function
)
```

`torch.neuron.DataParallel(model, device_ids=None, dim=0)`
Applies data parallelism by replicating the model on available NeuronCores and distributing data across the different NeuronCores for parallelized inference.

By default, DataParallel will use all available NeuronCores allocated for the current process for parallelism. DataParallel will apply parallelism on `dim=0` if `dim` is not specified.

DataParallel automatically enables dynamic batching on eligible models if `dim=0`. Dynamic batching can be disabled using `torch.neuron.DataParallel().disable_dynamic_batching()`. If dynamic batching is not enabled, the batch size at compilation-time must be equal to the batch size at inference-time divided by the number of NeuronCores being used. Specifically, the following must be true when dynamic batching is disabled: `input.shape[dim] / len(device_ids) == compilation_input.shape[dim]`. DataParallel will throw a warning if dynamic batching cannot be enabled.

DataParallel will try load all of a model’s NEFFs onto a single NeuronCore, only if all of the NEFFs can fit on a single NeuronCore. DataParallel does not currently support models that have been compiled with `NeuronCore Pipeline`.

`torch.neuron.DataParallel()` requires PyTorch >= 1.8.

**Required Arguments**

- **model** (`ScriptModule`) – Model created by the PyTorch-Neuron trace python API to be parallelized.

**Optional Arguments**

- **device_ids** (`list`) – List of `int` or `nc:#` that specify the NeuronCores to use for parallelization (default: all NeuronCores). Refer to the `device_ids note` for a description of how `device_ids` indexing works.

- **dim** (`int`) – Dimension along which the input tensor is scattered across NeuronCores (default `dim=0`).

**Attributes**

- **num_workers** (`int`) – Number of worker threads used for multithreaded inference (default: `2 * number of NeuronCores`).
• **split_size** (*int*) – Size of the input chunks (default: max(1, input. shape[dim] // number of NeuronCores)).

```
torch.neuron.DataParallel().disable_dynamic_batching()
```

Disables automatic dynamic batching on the DataParallel module. See *Dynamic batching disabled* for example of how DataParallel can be used with dynamic batching disabled. Use as follows:

```
>>> model_parallel = torch.neuron.DataParallel(model_neuron)
>>> model_parallel.disable_dynamic_batching()
```

**Note:** `device_ids` uses per-process NeuronCore granularity and zero-based indexing. Per-process granularity means that each Python process “sees” its own view of the world. Specifically, this means that `device_ids` only “sees” the NeuronCores that are allocated for the current process. Zero-based indexing means that each Python process will index its allocated NeuronCores starting at 0, regardless of the “global” index of the NeuronCores. Zero-based indexing makes it possible to redeploy the exact same code unchanged in different process. This behavior is analogous to the `device_ids` argument in the PyTorch *DataParallel* function.

As an example, assume DataParallel is run on an inf1.6xlarge, which contains four Inferentia chips each of which contains four NeuronCores:

- If `NEURON_RT_VISIBLE_CORES` is not set, a single process can access all 16 NeuronCores. Thus specifying `device_ids=["nc:0"]` will correspond to chip0:core0 and `device_ids=["nc:14"]` will correspond to chip3:core2.

- However, if two processes are launched where: process 1 has `NEURON_RT_VISIBLE_CORES=0-6` and process 2 has `NEURON_RT_VISIBLE_CORES=7-15`, `device_ids=["nc:14"]` cannot be specified in either process. Instead, chip3:core2 can only be accessed in process 2. Additionally, chip3:core2 is specified in process 2 with `device_ids=["nc:7"]`. Furthermore, in process 1, `device_ids=["nc:0"]` would correspond to chip0:core0; in process 2 `device_ids=["nc:0"]` would correspond to chip1:core3.

**Examples**

The following sections provide example usages of the `torch.neuron.DataParallel()` module.

**Default usage**

The default DataParallel use mode will replicate the model on all available NeuronCores in the current process. The inputs will be split on `dim=0`.

```
import torch
import torch_neuron
from torchvision import models

# Load the model and set it to evaluation mode
model = models.resnet50(pretrained=True)
model.eval()

# Compile with an example input
image = torch.rand([1, 3, 224, 224])
model_neuron = torch.neuron.trace(model, image)
```

(continues on next page)
# Create the DataParallel module
model_parallel = torch.neuron.DataParallel(model_neuron)

# Create a batched input
batch_size = 5
image_batched = torch.rand([batch_size, 3, 224, 224])

# Run inference with a batched input
output = model_parallel(image_batched)

### Specifying NeuronCores

The following example uses the `device_ids` argument to use the first three NeuronCores for DataParallel inference.

```python
import torch
import torch_neuron
from torchvision import models

# Load the model and set it to evaluation mode
model = models.resnet50(pretrained=True)
model.eval()

# Compile with an example input
image = torch.rand([1, 3, 224, 224])
model_neuron = torch.neuron.trace(model, image)

# Create the DataParallel module, run on the first three NeuronCores
# Equivalent to model_parallel = torch.neuron.DataParallel(model_neuron,
device_ids=[0, 1, 2])
model_parallel = torch.neuron.DataParallel(model_neuron, device_ids=['nc:0',
'nc:1', 'nc:2'])

# Create a batched input
batch_size = 5
image_batched = torch.rand([batch_size, 3, 224, 224])

# Run inference with a batched input
output = model_parallel(image_batched)
```

### DataParallel with dim != 0

In this example we run DataParallel inference using four NeuronCores and `dim = 2`. Because `dim != 0`, dynamic batching is not enabled. Consequently, the DataParallel inference-time batch size must be four times the compile-time batch size. DataParallel will generate a warning that dynamic batching is disabled because `dim != 0`.

```python
import torch
import torch_neuron

# Create an example model
class Model(torch.nn.Module):

(continues on next page)
```python
def __init__(self):
    super().__init__()
    self.conv = torch.nn.Conv2d(3, 3, 3)

def forward(self, x):
    return self.conv(x) + 1

model = Model()
model.eval()

# Compile with an example input
image = torch.rand([1, 3, 8, 8])
model_neuron = torch.neuron.trace(model, image)

# Create the DataParallel module using 4 NeuronCores and dim = 2
model_parallel = torch.neuron.DataParallel(model_neuron, device_ids=[0, 1, 2, 3], dim=2)

# Create a batched input
# Note that image_batched.shape[dim] / len(device_ids) == image.shape[dim]
batch_size = 4 * 8
image_batched = torch.rand([1, 3, batch_size, 8])

# Run inference with a batched input
output = model_parallel(image_batched)
```

### Dynamic batching

In the following example, we use the `torch.neuron.DataParallel()` module to run inference using several different batch sizes without recompiling the Neuron model.

```python
import torch
import torch_neuron
from torchvision import models

# Load the model and set it to evaluation mode
model = models.resnet50(pretrained=True)
model.eval()

# Compile with an example input
image = torch.rand([1, 3, 224, 224])
model_neuron = torch.neuron.trace(model, image)

# Create the DataParallel module
model_parallel = torch.neuron.DataParallel(model_neuron)

# Create batched inputs and run inference on the same model
batch_sizes = [2, 3, 4, 5, 6]
for batch_size in batch_sizes:
    image_batched = torch.rand([batch_size, 3, 224, 224])

    # Run inference with a batched input
    output = model_parallel(image_batched)
```
Dynamic batching disabled

In the following example, we use `torch.neuron.DataParallel().disable_dynamic_batching()` to disable dynamic batching. We provide an example of a batch size that will not work when dynamic batching is disabled as well as an example of a batch size that does work when dynamic batching is disabled.

```python
import torch
import torch_neuron
from torchvision import models

# Load the model and set it to evaluation mode
model = models.resnet50(pretrained=True)
model.eval()

# Compile with an example input
image = torch.rand([1, 3, 224, 224])
model_neuron = torch.neuron.trace(model, image)

# Create the DataParallel module and use 4 NeuronCores
model_parallel = torch.neuron.DataParallel(model_neuron, device_ids=[0, 1, 2, 3], dim=0)

# Disable dynamic batching
model_parallel.disable_dynamic_batching()

# Create a batched input (this won't work)
batch_size = 8
image_batched = torch.rand([batch_size, 3, 224, 224])

# This will fail because dynamic batching is disabled and
# image_batched.shape[dim] / len(device_ids) != image.shape[dim]
output = model_parallel(image_batched)

# Create a batched input (this will work)
batch_size = 4
image_batched = torch.rand([batch_size, 3, 224, 224])

# This will work because
# image_batched.shape[dim] / len(device_ids) == image.shape[dim]
output = model_parallel(image_batched)
```

Full tutorial with `torch.neuron.DataParallel`

For an end-to-end tutorial that uses DataParallel, see the PyTorch Resnet Tutorial.
2.2.4 PyTorch Supported operators

Current operator lists may be generated with these commands inside python:

```python
import torch.neuron
print(*torch.neuron.get_supported_operations(), sep='\n')
```

**PyTorch Neuron Release [2.0.536.0]**

- The following are operators with limited support on Neuron. Unlike fully supported operators, these operators are not returned when using `torch_neuron.get_supported_operations()`. See each operator description for conditional support:
  - `aten::max_pool2d_with_indices` - Supported when indices outputs are not used by a downstream operation. This allows the operation to be compiled to Neuron when it is equivalent to an `aten::max_pool2d`.
  - `aten::max_pool3d_with_indices` - Supported when indices outputs are not used by a downstream operation. This allows the operation to be compiled to Neuron when it is equivalent to an `aten::max_pool3d`.
  - `aten::where` - Supported when used as a conditional selection (3-argument variant). Unsupported when used to generate a dynamic list of indices (1-argument variant). See `torch.where()`.

**PyTorch Neuron Release [2.0.318.0]**

- Added support for new operators:
  - `aten::empty_like`
  - `aten::log`
  - `aten::type_as`
  - `aten::movedim`
  - `aten::einsum`
  - `aten::argmax`
  - `aten::min`
  - `aten::argmin`
  - `aten::abs`
  - `aten::cos`
  - `aten::sin`
  - `aten::linear`
  - `aten::pixel_shuffle`
  - `aten::group_norm`
  - `aten::_weight_norm`
PyTorch Neuron Release [1.5.21.0]
No change

PyTorch Neuron Release [1.5.7.0]
Added:
-aten::erf
-prim::DictConstruct

PyTorch Neuron Release [1.4.1.0]
No change

PyTorch Neuron Release [1.3.5.0]
Added:
-aten::numel
-aten::ones_like
-aten::reciprocal
-aten::topk

PyTorch Neuron Release [1.2.16.0]
No change

PyTorch Neuron Release [1.2.15.0]
No change

PyTorch Neuron Release [1.2.3.0]
Added:
-aten::silu
-aten::zeros_like

PyTorch Neuron Release [1.1.7.0]
Added:
-aten::_shape_as_tensor
-aten::chunk
-aten::empty
-aten::masked_fill
PyTorch Neuron Release [1.0.24045.0]

Added:

-aten::__and__
-aten::bmm
-aten::clone
-aten::expand_as
-aten::fill_
-aten::floor_divide
-aten::full
-aten::hardtanh
-aten::hardtanh_
-aten::le
-aten::leaky_relu
-aten::lt
-aten::mean
-aten::ne
-aten::softplus
-aten::unbind
-aten::upsample_bilinear2d

PyTorch Neuron Release [1.0.1720.00]

Added:

-aten::constant_pad_nd
-aten::meshgrid

PyTorch Neuron Release [1.0.1532.0]

Added:

-aten::ones

PyTorch Neuron Release [1.0.1522.0]

- No change

PyTorch Neuron Release [1.0.1386.0]

Added the following instructions. Please note, primitives are included in this list from this release.

-aten::ceil
-aten::clamp
-aten::eq
-aten::exp
-aten::expand_as
-aten::flip
-aten::full_like
-aten::ge
-aten::gt

(continues on next page)
aten::log2
aten::log_softmax
aten::max
aten::neg
aten::relu
aten::rsqrt
aten::scalarImplicit
aten::sqrt
aten::squeeze
aten::stack
aten::sub
aten::sum
aten::true_divide
aten::upsample_nearest2d
prim::Constant
prim::GetAttr
prim::ImplicitTensorToNum
prim::ListConstruct
prim::ListUnpack
prim::NumToTensor
prim::TupleConstruct
prim::TupleUnpack

**PyTorch Neuron Release [1.0.1168.0]**

Added aten::ScalarImplicit

**PyTorch Neuron Release [1.0.1001.0]**

Added

aten::detach
aten::floor
aten::gelu
aten::pow
aten::sigmoid
aten::split

Removed (Reasons given alongside)

aten::embedding (does not meet performance criteria)
aten::erf (error function does not meet accuracy criteria)
aten::tf_dtype_from_torch (internal support function, not an operator)
PyTorch Neuron Release [1.0.825.0]

PyTorch Neuron Release [1.0.763.0]

• Please note. Starting with this release we will not publish primitives (prim::).

• Previous release inaccurately listed these operators as aten ops, they are not.

| aten::tf_broadcastable_slice |
| aten::tf_padding |

The following new operators are added in this release.

| aten::Int |
| aten::arange |
| aten::contiguous |
| aten::div |
| aten::embedding |
| aten::erf |
| aten::expand |
| aten::eye |
| aten::index_select |
| aten::layer_norm |
| aten::matmul |
| aten::mm |
| aten::permute |
| aten::reshape |
| aten::rsub |
| aten::select |
| aten::size |
| aten::slice |
| aten::softmax |
| aten::tf_dtype_from_torch |
| aten::to |
| aten::transpose |
| aten::unsqueeze |
| aten::view |
| aten::zeros |

These operators were already supported previously (removing the two that were included by mistake).

| aten::_convolution |
| aten::adaptive_avg_pool2d |
| aten::add |
| aten::add_ |
| aten::addmm |
| aten::avg_pool2d |
| aten::batch_norm |
| aten::cat |
| aten::dimension_value |
| aten::dropout |
| aten::flatten |
| aten::max_pool2d |
| aten::mul |
| aten::relu_ |
| aten::t |
| aten::tanh |
| aten::values |

(continues on next page)
PyTorch Neuron Release [1.0.672.0]

No change

PyTorch Neuron Release [1.0.552.0]

2.2.5 Troubleshooting Guide for Torch-Neuron

General Torch-Neuron issues

If you see an error about “Unknown builtin op: neuron::forward_1” like below, please ensure that import line “import torch_neuron” (to register the Neuron custom operation) is in the inference script before using torch.jit.load.

Unknown builtin op: neuron::forward_1.
Could not find any similar ops to neuron::forward_1. This op may not exist, or may not be currently supported in TorchScript.
TorchVision related issues

If you encounter an error like below, it is because latest torchvision version >= 0.7 is not compatible with Torch-Neuron 1.5.1. Please downgrade torchvision to version 0.6.1:

```python
E  AttributeError: module 'torch.jit' has no attribute '_script_if_tracing'
```

torch.jit.trace issues

The PyTorch-Neuron trace python API uses the PyTorch torchvision.models for execution on Inferentia. Due to that, to execute your PyTorch model on Inferentia it must be torch-jit-traceable, otherwise you need to make sure your model is torch-jit-traceable. You can try modifying your underlying PyTorch model code to make it traceable. If it’s not possible to change your model code, you can write a wrapper around your model that makes it torch-jit-traceable to compile it for Inferentia.

Please visit torchvision.models to review the properties that a model must have to be torch-jit-traceable. The PyTorch-Neuron trace API torchvision.models accepts **kwargs for torchvision.models. For example, you can use the strict=False flag to compile models with dictionary outputs.

Compiling models with outputs that are not torch-jit-traceable

To enable compilation of models with non torch-jit-traceable outputs, you can use a technique that involves writing a wrapper that converts the model’s output into a form that is torch-jit-traceable. You can then compile the wrapped model for Inferentia using torchvision.models.

The following example uses a wrapper to compile a model with non torch-jit-traceable outputs. This model cannot be compiled for Inferentia in its current form because it outputs a list of tuples and tensors, which is not torch-jit-traceable.

```python
import torch
import torchvision
import torch.nn as nn

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv = nn.Conv2d(1, 1, 3)

    def forward(self, x):
        a = self.conv(x) + 1
        b = self.conv(x) + 2
        c = self.conv(x) + 3
        # An output that is a list of tuples and tensors is not torch-jit-traceable
        return [(a, b), c]

model = Model()
model.eval()

inputs = torch.randn(1, 1, 3, 3)
```
To compile this model for Inferentia, we can write a wrapper around the model to convert its outputs into a tuple of tensors, which is torch-jit-traceable.

```python
class NeuronCompatibilityWrapper(nn.Module):
    def __init__(self):
        super(NeuronCompatibilityWrapper, self).__init__()
        self.model = Model()

    def forward(self, x):
        out = self.model(x)
        # An output that is a tuple of tuples and tensors is torch-jit-traceable
        return tuple(out)
```

Now, we can successfully compile the model for Inferentia using the `NeuronCompatibilityWrapper` wrapper as follows:

```python
model = NeuronCompatibilityWrapper()
model.eval()

# Compile the traceable wrapped model
model_neuron = torch.neuron.trace(model, inputs)
```

If the model’s outputs must be in the original form, a second wrapper can be used to transform the outputs after compilation for Inferentia. The following example uses the `OutputFormatWrapper` wrapper to convert the compiled model’s output back into the original form of a list of tuples and tensors.

```python
class OutputFormatWrapper(nn.Module):
    def __init__(self):
        super(OutputFormatWrapper, self).__init__()
        self.traceable_model = NeuronCompatibilityWrapper()

    def forward(self, x):
        out = self.traceable_model(x)
        # Return the output in the original format of Model()
        return list(out)
```

```python
model = OutputFormatWrapper()
model.eval()

# Compile the traceable wrapped model
model.traceable_model = torch.neuron.trace(model.traceable_model, inputs)
```
Compiling a submodule in a model that is not torch-jit-traceable

The following example shows how to compile a submodule that is part of a non torch-jit-traceable model. In this example, the top-level model `Outer` uses a dynamic flag, which is not torch-jit-traceable. However, the submodule `Inner` is torch-jit-traceable and can be compiled for Inferentia.

```python
import torch
import torch_neuron
import torch.nn as nn

class Inner(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv = nn.Conv2d(1, 1, 3)

    def forward(self, x):
        return self.conv(x) + 1

class Outer(nn.Module):
    def __init__(self):
        super().__init__()
        self.inner = Inner()

    def forward(self, x, add_offset: bool = False):
        base = self.inner(x)
        if add_offset:
            return base + 1
        return base

model = Outer()
inputs = torch.rand(1, 1, 3, 3)

# Compile the traceable wrapped submodule
model.inner = torch.neuron.trace(model.inner, inputs)

# TorchScript the model for serialization
script = torch.jit.script(model)
torch.jit.save(script, 'model.pt')

loaded = torch.jit.load('model.pt')
```

Alternatively, for usage scenarios in which the model configuration is static during inference, the dynamic flags can be hard coded in a wrapper to make the model torch-jit-traceable and enable compiling the entire model for Inferentia. In this example, we assume the `add_offset` flag is always `True` during inference, so we can hardcode this conditional path in the `Static` wrapper to remove the dynamic behavior and compile the entire model for Inferentia.

```python
class Static(nn.Module):
    def __init__(self):
        super().__init__()
        self.outer = Outer()

    def forward(self, x):
        # hardcode `add_offset=True`
        output = self.outer(x, add_offset=True)
        return output
```

(continues on next page)
model = Static()

# We can now compile the entire model because 'add_offset=True' is hardcoded in the Static wrapper
model_neuron = torch.neuron.trace(model, inputs)

2.3 Release notes

2.3.1 What's New

PyTorch Neuron release notes

- Known Issues and Limitations - Updated 08/12/2021
- PyTorch Neuron release [2.0.536.0]
- PyTorch Neuron release [2.0.468.0]
- PyTorch Neuron release [2.0.392.0]
- PyTorch Neuron release [2.0.318.0]
- [1.8.1.1.5.21.0]
- [1.8.1.1.5.7.0]
- [1.8.1.1.4.1.0]
- [1.7.1.1.3.5.0]
- [1.7.1.1.2.16.0]
- [1.7.1.1.2.15.0]
- [1.7.1.1.2.3.0]
- [1.1.7.0]
- [1.0.1978.0]
- [1.0.1721.0]
- [1.0.1532.0]
- [1.0.1522.0]
- [1.0.1386.0]
- [1.0.1168.0]
- [1.0.1001.0]
- [1.0.825.0]
- [1.0.763.0]
- [1.0.672.0]
- [1.0.627.0]
This document lists the release notes for the Pytorch-Neuron package.

**Known Issues and Limitations - Updated 08/12/2021**

The following are not torch-neuron limitations, but may impact models you can successfully torch.neuron.trace

- If you attempt to import torch.neuron from Python 3.5 you will see this error in 1.1.7.0 - please use Python 3.6 or greater:

```python
File ".../site-packages/torch_neuron/__init__.py", line 29
  f'Invalid dependency version torch=={torch.__version__}. '

SyntaxError: invalid syntax
```

- Torchvision has dropped support for Python 3.5
- HuggingFace transformers has dropped support for Python 3.5
- aten::max only correctly implements the simplest versions of that operator, the variants that return a tuple with arg max now return NotImplementedError during compilation
- There is a dependency between versions of torchvision and the torch package that customers should be aware of when compiling torchvision models. These dependency rules can be managed through pip. At the time of writing torchvision==0.6.1 matched the torch==1.5.1 release, and torchvision==0.8.2 matched the torch==1.7.1 release

**PyTorch Neuron release [2.0.536.0]**

Date: 01/05/2022

**New in this release**

- Added new operator support for specific variants of operations (See PyTorch Supported operators)
- Added optional optimizations keyword to torch_neuron.trace() which accepts a list of Optimization passes.

**PyTorch Neuron release [2.0.468.0]**

Date: 12/15/2021
New in this release

- Added support for `aten::cumsum` operation.
- Fixed `aten::expand` to correctly handle adding new dimensions.

PyTorch Neuron release [2.0.392.0]

Date: 11/05/2021

- Updated Neuron Runtime (which is integrated within this package) to `libnrt 2.2.18.0` to fix a container issue that was preventing the use of containers when `/dev/neuron0` was not present. See details here *Neuron Runtime 2.x Release Notes*.

PyTorch Neuron release [2.0.318.0]

Date: 10/27/2021

New in this release

- PyTorch Neuron 1.x now support Neuron Runtime 2.x (`libnrt.so` shared library) only.

  **Important:**
  - You must update to the latest Neuron Driver (`aws-neuron-dkms` version 2.1 or newer) for proper functionality of the new runtime library.
  - Read *Introducing Neuron Runtime 2.x (libnrt.so)* application note that describes *why we are making this change* and *how this change will affect the Neuron SDK* in detail.
  - Read *Migrate your application to Neuron Runtime 2.x (libnrt.so)* for detailed information of how to migrate your application.

- Introducing PyTorch 1.9.1 support (support for `torch==1.9.1`)
- Added `torch_neuron.DataParallel`, see ResNet-50 tutorial [html] and *Data Parallel Inference on Torch Neuron* application note.
- Added support for tracing on GPUs
- Added support for `ConvTranspose2d`
- Added support for new operators:
  - `aten::empty_like`
  - `aten::log`
  - `aten::type_as`
  - `aten::movedim`
  - `aten::einsum`
  - `aten::argmax`
  - `aten::min`
- `aten::argmin`
- `aten::abs`
- `aten::cos`
- `aten::sin`
- `aten::linear`
- `aten::pixel_shuffle`
- `aten::group_norm`
- `aten::_weight_norm`

• Added `torch_neuron.is_available()`

Resolved Issues

• Fixed a performance issue when using both the `dynamic_batch_size=True` trace option and `--neuron-core-pipeline` compiler option. Dynamic batching now uses OpenMP to execute pipeline batches concurrently.
• Fixed `torch_neuron.trace` issues:
  – Fixed a failure when the same submodule was traced with multiple inputs
  – Fixed a failure where some operations would fail to be called with the correct arguments
  – Fixed a failure where custom operators (torch plugins) would cause a trace failure
• Fixed variants of `aten::upsample_bilinear2d` when `scale_factor=1`
• Fixed variants of `aten::expand` using `dim=-1`
• Fixed variants of `aten::stack` using multiple different input data types
• Fixed variants of `aten::max` using indices outputs

[1.8.1.5.21.0]

Date: 08/12/2021

Summary

• Minor updates.

[1.8.1.5.7.0]

Date: 07/02/2021
Summary

• Added support for dictionary outputs using `strict=False` flag. See *Troubleshooting Guide for Torch-Neuron*.
• Updated `aten::batch_norm` to correctly implement the `affine` flag.
• Added support for `aten::erf` and `prim::DictConstruct`. See *PyTorch Supported operators*.
• Added dynamic batch support. See *PyTorch-Neuron trace python API*.

**[1.8.1.1.4.1.0]**

Date: 5/28/2021

Summary

• Added support for PyTorch 1.8.1
  – Models compatibility
    * Models compiled with previous versions of Neuron PyTorch (<1.8.1) are compatible with Neuron PyTorch 1.8.1.
    * Models compiled with Neuron PyTorch 1.8.1 are not backward compatible with previous versions of Neuron PyTorch (<1.8.1).
  – Updated tutorials to use Hugging Face Transformers 4.6.0.
  – Added a new set of forward operators (forward_v2)
  – Host memory allocation when loading the same model on multiple NeuronCores is significantly reduced
  – Fixed an issue where models would not deallocate all memory within a python session after being garbage collected.
  – Fixed a TorchScript/C++ issue where loading the same model multiple times would not use multiple NeuronCores by default.
• Fixed logging to no longer configure the root logger.
• Removed informative messages that were produced during compilations as warnings. The number of warnings reduced significantly.
• Convolution operator support has been extended to include ConvTranspose2d variants.
• Reduce the amount of host memory usage during inference.
[1.7.1.3.5.0]
Date: 4/30/2021

**Summary**

- ResNext models now functional with new operator support
- Convolution operator support has been extended to include most Conv1d and Conv3d variants
- New operator support. Please see *PyTorch Supported operators* for the complete list of operators.

[1.7.1.1.2.16.0]
Date: 3/4/2021

**Summary**

- Minor enhancements.

[1.7.1.1.2.15.0]
Date: 2/24/2021

**Summary**

- Fix for CVE-2021-3177.

[1.7.1.1.2.3.0]
Date: 1/30/2021

**Summary**

- Made changes to allow models with -inf scalar constants to correctly compile
- Added new operator support. Please see *PyTorch Supported operators* for the complete list of operators.
[1.1.7.0]

Date: 12/23/2020

Summary

• We are dropping support for Python 3.5 in this release
• `torch.neuron.trace` behavior will now throw a `RuntimeError` in the case that no operators are compiled for neuron hardware
• `torch.neuron.trace` will now display compilation progress indicators (dots) as default behavior (neuron-cc must updated to the December release to greater to see this feature)
• Added new operator support. Please see PyTorch Supported operators for the complete list of operators.
• Extended the BERT pretrained tutorial to demonstrate execution on multiple cores and batch modification, updated the tutorial to accomodate changes in the Hugging Face Transformers code for version 4.0
• Added a tutorial for torch-serve which extends the BERT tutorial
• Added support for PyTorch 1.7

[1.0.1978.0]

Date: 11/17/2020

Summary

• Fixed bugs in comparison operators, and added remaining variantes (eq, ne, gt, ge, lt, le)
• Added support for `prim::PythonOp` - note that this must be run on CPU and not Neuron. We recommend you replace this code with PyTorch operators if possible
• Support for a series of new operators. Please see PyTorch Supported operators for the complete list of operators.
• Performance improvements to the runtime library
• Correction of a runtime library bug which caused models with large tensors to generate incorrect results in some cases

[1.0.1721.0]

Date: 09/22/2020

Summary

• Fixed bugs in comparison operators, and added remaining variantes (eq, ne, gt, ge, lt, le)
• Added support for `prim::PythonOp` - note that this must be run on CPU and not Neuron. We recommend you replace this code with PyTorch operators if possible
• Support for a series of new operators. Please see PyTorch Supported operators for the complete list of operators.
• Performance improvements to the runtime library
• Correction of a runtime library bug which caused models with large tensors to generate incorrect results in some cases
Summary

• Various minor improvements to the Pytorch autopartitioner feature
• Support for the operators aten::constant_pad_nd, aten::meshgrid
• Improved performance on various torchvision models. Of note are resnet50 and vgg16

[1.0.1532.0]

Date: 08/08/2020

Summary

• Various minor improvements to the Pytorch autopartitioner feature
• Support for the aten:ones operator

[1.0.1522.0]

Date: 08/05/2020

Summary

Various minor improvements.

[1.0.1386.0]

Date: 07/16/2020

Summary

This release adds auto-partitioning, model analysis and PyTorch 1.5.1 support, along with a number of new operators

Major New Features

• Support for Pytorch 1.5.1
• Introduce an automated operator device placement mechanism in torch.neuron.trace to run subgraphs that contain operators that are not supported by the neuron compiler in native PyTorch. This new mechanism is on by default and can be turned off by adding argument fallback=False to the compiler arguments.
• Model analysis to find supported and unsupported operators in a model
Resolved Issues

[1.0.1168.0]

Date 6/11/2020

Summary

Major New Features

Resolved Issues

Known Issues and Limitations

[1.0.1001.0]

Date: 5/11/2020

Summary

Additional PyTorch operator support and improved support for model saving and reloading.

Major New Features

• Added Neuron Compiler support for a number of previously unsupported PyTorch operators. Please see `neuron-cc-ops-pytorch` for the complete list of operators.
• Add support for torch.neuron.trace on models which have previously been saved using torch.jit.save and then reloaded.

Resolved Issues

Known Issues and Limitations

[1.0.825.0]

Date: 3/26/2020
Summary

Major New Features

Resolved Issues

Known Issues and limitations

[1.0.763.0]

Date: 2/27/2020

Summary

Added Neuron Compiler support for a number of previously unsupported PyTorch operators. Please see PyTorch Supported operators for the complete list of operators.

Major new features

• None

Resolved issues

• None

[1.0.672.0]

Date: 1/27/2020

Summary

Major new features

Resolved issues

• Python 3.5 and Python 3.7 are now supported.
Known issues and limitations

Other Notes

[1.0.627.0]

Date: 12/20/2019

Summary

This is the initial release of torch-neuron. It is not distributed on the DLAMI yet and needs to be installed from the neuron pip repository.

Note that we are currently using a TensorFlow as an intermediate format to pass to our compiler. This does not affect any runtime execution from PyTorch to Neuron Runtime and Inferentia. This is why the neuron-cc installation must include [tensorflow] for PyTorch.

Major new features

Resolved issues

Known issues and limitations

Models TESTED

The following models have successfully run on neuron-inferentia systems

1. SqueezeNet
2. ResNet50
3. Wide ResNet50

Pytorch Serving

In this initial version there is no specific serving support. Inference works correctly through Python on Inf1 instances using the neuron runtime. Future releases will include support for production deployment and serving of models

Profiler support

Profiler support is not provided in this initial release and will be available in future releases
Automated partitioning

Automatic partitioning of graphs into supported and non-supported operations is not currently supported. A tutorial is available to provide guidance on how to manually partition a model graph. Please see pytorch-manual-partitioning-jn-tutorial

PyTorch dependency

Currently PyTorch support depends on a Neuron specific version of PyTorch v1.3.1. Future revisions will add support for 1.4 and future releases.

Trace behavior

In order to trace a model it must be in evaluation mode. For examples please see ResNet50 model for Inferentia

Six pip package is required

The Six package is required for the torch-neuron runtime, but it is not modeled in the package dependencies. This will be fixed in a future release.

Multiple NeuronCore support

If the num-neuroncores options is used the number of cores must be manually set in the calling shell environment variable for compilation and inference.

For example: Using the keyword argument compiler_args=[`--num-neuroncores`, `4`] in the trace call, requires NEURONCORE_GROUP_SIZES=4 to be set in the environment at compile time and runtime

CPU execution

At compilation time a constant output is generated for the purposes of tracing. Running inference on a non neuron instance will generate incorrect results. This must not be used. The following error message is generated to stderr:

```
Warning: Tensor output are ** NOT CALCULATED ** during CPU execution and only indicate tensor shape
```

Other notes

- Python version(s) supported:
  - 3.6
- Linux distribution supported:
  - DLAMI Ubuntu 18 and Amazon Linux 2 (using Python 3.6 Conda environments)
  - Other AMIs based on Ubuntu 18
  - For Amazon Linux 2 please install Conda and use Python 3.6 Conda environment
Warning: Starting with Neuron 1.14.0, Neuron Conda packages in Deep Learning AMI are no longer supported, for more information see blog announcing the end of support for Neuron conda packages.

Conda-PyTorch Release notes

This document lists the release notes for the Neuron Conda-Pytorch package.

Table of Contents

- [1.7.1.1.3.5.0]
- [1.7.1.1.2.16.0]
- [1.7.1.1.2.15.0]
- [1.7.1.1.2.3.0]
- [1.5.1.1.2.3.0]
- [1.7.1.1.1.7.0]
- [1.5.1.1.1.7.0]
- [1.5.1.1.0.1978.0]
- [1.5.1.1.0.1721.0-2.0.1017.0]
- [1.5.1.1.0.298.0-2.0.880.0]
- [1.5.1.1.0.258.0-2.0.871.0]
- [1.5.1.1.0.251.0-2.0.783.0]
- [1.3.0.1.0.215.0-2.0.633.0]
- [1.3.0.1.0.170.0-2.0.349.0]
- [1.3.0.1.0.90.0-2.0.62.0]
- [1.3.0.1.0.90.0-1.0.918.0]
- [1.3.0.1.0.41.0-1.0.737.0]
[1.7.1.1.3.5.0]

Date: 4/30/2021

Included Neuron Packages

neuron-cc-1.3.7.0
torch_neuron-1.7.1.1.3.5.0

[1.7.1.1.2.16.0]

Date: 3/4/2021

Included Neuron Packages

neuron-cc-1.2.7.0
torch_neuron-1.7.1.1.2.16.0

Resolved Issues

Minor internal improvements.

[1.7.1.1.2.15.0]

Date: 2/24/2021

Included Neuron Packages

neuron-cc-1.2.7.0
torch_neuron-1.7.1.1.2.15.0

Resolved Issues

Fix for CVE-2021-3177.

[1.7.1.1.2.3.0]

Date: 1/30/2021
Included Neuron Packages

neuron-cc-1.2.2.0
torch_neuron-1.7.1.1.2.3.0

Resolved Issues

Resolved the segmentation fault when enabling profiling using NEURON_PROFILE=<directory> environment variable for inference within a PyTorch-Neuron Conda environment (https://github.com/aws/aws-neuron-sdk/issues/230). This fix will be available in the next DLAMI release.

[1.5.1.1.2.3.0]

Date: 1/30/2021

Included Neuron Packages

neuron-cc-1.2.1.0
torch_neuron-1.5.1.1.2.3.0

Resolved Issues

Resolved the segmentation fault when enabling profiling using NEURON_PROFILE=<directory> environment variable for inference within a PyTorch-Neuron Conda environment (https://github.com/aws/aws-neuron-sdk/issues/230). This fix will be available in the next DLAMI release.

[1.7.1.1.1.7.0]

Date: 12/23/2020

Included Neuron Packages

neuron-cc-1.1.7.0
torch_neuron-1.7.1.1.1.7.0

Known Issues

When enabling profiling using NEURON_PROFILE=<directory> environment variable for inference within a PyTorch-Neuron Conda environment (such as the DLAMI aws_neuron_pytorch_p36 environment), running inference would result in segmentation fault (https://github.com/aws/aws-neuron-sdk/issues/230). The workaround is to reinstall the PyTorch package of the same version as installed. For example, if the installed PyTorch version is 1.7.1, please do:

```
pip install --no-deps --force-reinstall torch==1.7.1
```
Similarly, if the installed PyTorch version is 1.5.1,

```
pip install --no-deps --force-reinstall torch==1.5.1
```

[1.5.1.1.7.0]

Date: 12/22/2020

**Included Neuron Packages**

- neuron-cc-1.1.7.0
- torch_neuron-1.5.1.1.7.0

[1.5.1.0.1978.0]

Date: 11/17/2020

**Included Neuron Packages**

- neuron-cc-1.0.24045.0
- torch_neuron-1.5.1.0.1978.0

**Known Issues**

- Conda environment aws_neuron_pytorch_p36 of Conda DLAMI v36 cannot be updated to this latest (1.5.1.0.1978.0) PyTorch-Neuron Conda package using “conda update torch-neuron” command. To use the latest PyTorch-Neuron Conda package, please create a new Conda environment and install PyTorch-Neuron Conda package there using “conda install -c https://conda.repos.neuron.amazonaws.com torch-neuron”. This issue is fixed in Conda DLAMI v37.

- Conda environment aws_neuron_pytorch_p36 of Conda DLAMI v30 to v35 can be updated using the following commands:

  ```
  conda install --force torch-neuron=1.5.1.0.1978.0
  conda install --force numpy=1.18.1
  ```

[1.5.1.0.1721.0_2.0.1017.0]

Date: 09/22/2020
Included Neuron Packages

neuron-cc-1.0.20600.0
torch_neuron-1.0.1721.0

Resolved Issues

When TorchVision is updated to version >= 0.5, running Neuron compilation would crash with “Segmentation fault (core dumped)” error.

Known Issues

- When TorchVision is updated to version >= 0.5, running Neuron compilation would crash with “Segmentation fault (core dumped)” error. This issue is resolved with version 1.5.1.1.0.1721.0_2.0.1017.0 of PyTorch-Neuron Conda package (9/22/2020 release).
- When running PyTorch script in latest Torch-Neuron conda environment, you may see errors “AttributeError: module ‘numpy’ has no attribute ‘integer’” and “ModuleNotFoundError: No module named ‘numpy.core._multiarray_umath’”. This is due to older version of numpy. Please update numpy to version 1.18 using the command “conda install --force numpy=1.18.1”.
- Due to changes to PyTorch-Neuron Conda package content in this release, updating from aws_neuron_pytorch_p36 of Conda DLAMI (v35 or earlier) would require the following to update:

  conda install --force torch-neuron=1.5.1.1.0.1721.0
  conda install --force numpy=1.18.1

[1.5.1.1.0.298.0_2.0.880.0]

Date: 08/08/2020

Included Neuron Packages

neuron-cc-1.0.18001.0
torch_neuron-1.0.1532.0
torch_neuron_base-1.5.1.1.0.298.0

[1.5.1.1.0.258.0_2.0.871.0]

Date: 08/05/2020
Included Neuron Packages

neuron-cc-1.0.17937.0
torch_neuron-1.0.1522.0
torch_neuron_base-1.5.1.1.0.258.0

[1.5.1.0.251.0_2.0.783.0]

Date: 07/16/2020
Now supporting Python 3.7 Conda packages in addition to Python 3.6 Conda packages.

Included Neuron Packages

neuron-cc-1.0.16861.0
torch_neuron-1.0.1386.0
torch_neuron_base-1.5.1.1.0.251.0

[1.3.0.1.0.215.0-2.0.633.0]

Date 6/11/2020

Included Neuron Packages

neuron-cc-1.0.15275.0
torch_neuron-1.0.1168.0
torch_neuron_base-1.3.0.1.0.215.0

[1.3.0.1.0.170.0-2.0.349.0]

Date 5/11/2020

Included Neuron Packages

neuron-cc-1.0.12696.0
torch_neuron-1.0.1001.0
torch_neuron_base-1.3.0.1.0.170.0
[1.3.0.1.0.90.0_2.0.62.0]

Date 3/26/2020

Included Neuron Packages

neuron-cc-1.0.9410.0
torch_neuron-1.0.825.0
torch_neuron_base-1.3.0.1.0.90.0

[1.3.0.1.0.90.0-1.0.918.0]

Date: 2/27/2020

Included Neuron Packages

neuron_cc-1.0.7878.0
torch_neuron-1.0.763.0
torch_neuron_base-1.3.0.1.0.90.0

Known Issues and Limitations

conda-tensorflow-release-notes

[1.3.0.1.0.41.0-1.0.737.0]

Date: 1/27/2020

Included Neuron Packages

neuron-cc-1.0.6801.0
torch-neuron-1.0.672.0
torch-neuron-base-1.3.0.1.0.41.0

Known Issues and Limitations
Neuron is integrated into TensorFlow, and provides you with a familiar environment to run inference using Inferentia based instances.

Neuron supports both Tensorflow 2.x, an eager-execution-based deep learning framework, and TensorFlow 1.x, a static-graph-based deep learning framework.

### 3.1 Installation Guide

#### 3.1.1 Install Neuron TensorFlow

**Note:**

- Instructions in this page only apply to setting up Neuron components on Linux host running Ubuntu or Amazon Linux AMI.
- For an example of how to install Neuron components in a container, see Docker environment setup and our Containers documentation for more details.

- Develop on AWS ML accelerator instance
- Compile on compute instance
- Deploy on AWS ML accelerator instance

**Develop on AWS ML accelerator instance**

The simplest environment setup for model development installs all Neuron SDK components directly on an AWS ML accelerator instance: the Neuron framework extensions, compiler, runtime, and tools. This will allow you to compile, execute, and performance tune your model, all in the same instance. This is the recommended workflow when first starting to work with Neuron device or when optimizing a model.

**Important:**

**For successful installation or update to Neuron 1.16.0 and newer from previous releases:**

- Stop Neuron Runtime 1.x daemon (neuron-rtd) by running: `sudo systemctl stop neuron-rtd`
• Uninstall neuron-rtd by running: sudo apt remove aws-neuron-runtime or
  sudo yum remove aws-neuron-runtime

• Install or upgrade to latest Neuron driver (aws-neuron-dkms) by following the “Setup
  Guide” instructions.

• Visit *Introducing Neuron Runtime 2.x (libnrt.so)* for more information.

TensorFlow 2.5.1
TensorFlow 2.4.3
TensorFlow 2.3.4
TensorFlow 2.2.3
TensorFlow 2.1.4
TensorFlow 1.15.5
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

*Note:* For a successful installation or update, execute each line of the instructions below separately or
  copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y
```

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  # 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
  # or upgrade to latest Neuron driver

(continues on next page)
# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# --reboot

# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]

# Optional: Install Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
named=Neuron YUM Repository
EOF
```
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  ➔ sudo systemctl stop neuron-rtd

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo yum install aws-neuron-tools -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++ python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

(continues on next page)
# Instal Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron TensorFlow
pip install tensorflow-neuron[cc]

# Optional: Install Neuron TensorFlow model server
sudo yum install tensorflow-model-server-neuron -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
#   or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

(continued on next page)
# Install Neuron Tools
```bash
sudo apt-get install aws-neuron-tools -y
```

# Install Neuron TensorBoard
```bash
pip install tensorboard-plugin-neuron
```

```bash
export PATH=/opt/aws/neuron/bin:$PATH
```

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
```bash
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip
```

# Install Jupyter notebook kernel
```bash
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron TensorFlow
```bash
pip install tensorflow-neuron[cc]
```

Optional: Install Neuron TensorFlow model server
```bash
sudo apt-get install tensorflow-model-server-neuron -y
```

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
#############################################################################
# Install OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y
# Install Neuron Driver
sudo yum install aws-neuron-dkms -y
#############################################################################
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
#############################################################################
# Install Neuron Tools
sudo yum install aws-neuron-tools -y
# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron
export PATH=/opt/aws/neuron/bin:$PATH
# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip
# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]
# Optional: Install Neuron TensorFlow model server
sudo yum install tensorflow-model-server-neuron -y

Ubuntu AMI
Amazon Linux AMI
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron

export PATH=/opt/aws/neuron/bin:$PATH

(continues on next page)
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name
"Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.3.*

# Optional: Install Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron=2.4.3.2.0.4.0 -y

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  #  sudo systemctl stop neuron-rtd

# To install or update to Neuron versions 1.16.0 and newer from previous releases:

(continues on next page)
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

```
# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo yum install aws-neuron-tools -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.3.*

# Optional: Install Neuron TensorFlow model server
sudo yum install tensorflow-model-server-neuron=2.4.3.2.0.4.0 -y
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

#############################################################################

→

##################################
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
→'sudo systemctl stop neuron-rtd'

#############################################################################

→

###################################
# To install or update to Neuron versions 1.16.0 and newer from previous
→ releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
→ or upgrade to latest Neuron driver

#############################################################################

→

#######
# Warning: If Linux kernel is updated as a result of OS package update
→ reboot
# Neuron driver (aws-neuron-dkms) should be re-installed after,

#######

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

#############################################################################

→#

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after,
→reboot

#############################################################################

#######
# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name
→"Python (Neuron TensorFlow)"

(continues on next page)
pip install jupyter notebook
data install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.3.*

# Optional: Install Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron=2.4.3.2.0.4.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$ (uname -r) kernel-headers-$ (uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools

(continues on next page)
sudo yum install aws-neuron-tools -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.3.*

# Optional: Install Neuron TensorFlow model server
sudo yum install tensorflow-model-server-neuron=2.4.3.2.0.4.0 -y

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
./etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <EOF
deb https://apt.repos.neuron.amazonaws.com/${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.3.*

# Optional: Install Neuron TensorFlow model server
sudo yum install tensorflow-model-server-neuron=2.4.3.2.0.4.0 -y
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
        -> 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.4.*  

# Optional: Install Neuron TensorFlow model server  
sudo apt-get install tensorflow-model-server-neuron=2.3.4.2.0.4.0 -y

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates  
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF  
[neuron]  
name=Neuron YUM Repository  
baseurl=https://yum.repos.neuron.amazonaws.com  
enabled=1  
metadata_expire=0  
EOF  
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages  
sudo yum update -y
```

```
# To install or update to Neuron versions 1.16.0 and newer from previous releases:  
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver  
# Install OS headers  
sudo yum install kernel-devel-$uname -r) kernel-headers-$uname -r) -y

# Install Neuron Driver  
sudo yum install aws-neuron-dkms -y
```

```
# Warning: If Linux kernel is updated as a result of OS package update  
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot  
```

# Install Neuron Tools
sudo yum install aws-neuron-tools -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.4.*

# Optional: Install Neuron TensorFlow model server
sudo yum install tensorflow-model-server-neuron=2.3.4.2.0.4.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

(continues on next page)
# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.4.*

# Optional: Install Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron=2.3.4.2.0.4.0 -y

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Note: There is no DLAMI Conda environment for this framework version

(continues on next page)
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   → 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$({uname} -r) kernel-headers-$({uname} -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo yum install aws-neuron-tools -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"

(continues on next page)
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.4.*

# Optional: Install Neuron TensorFlow model server
sudo yum install tensorflow-model-server-neuron=2.3.4.2.0.4.0 -y

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

(continues on next page)
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot.

# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.*

# Optional: Install Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron=2.2.3.2.0.4.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

(continues on next page)
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo yum install aws-neuron-tools -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

(continues on next page)
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.*

# Optional: Install Neuron TensorFlow model server
sudo yum install tensorflow-model-server-neuron=2.2.3.2.0.4.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   -> 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install:
#   -> or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y

(continues on next page)
# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.*

# Optional: Install Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron=2.2.3.2.0.4.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
  # or upgrade to latest Neuron driver

(continues on next page)
# Install OS headers
sudo yum install kernel-devel-$\{\text{uname} -r\} \ kernel-headers-$\{\text{uname} -r\} -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Tools
sudo yum install aws-neuron-tools -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name
"Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.*

# Optional: Install Neuron TensorFlow model server
sudo yum install tensorflow-model-server-neuron=2.2.3.2.0.4.0 -y

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate

(continues on next page)
```
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.*

# Optional: Install Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron=2.1.4.2.0.4.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

(continues on next page)
# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

#############################################################################
→
####
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
→
####

# Install Neuron Tools
sudo yum install aws-neuron-tools -y

# Install Neuron TensorFlow
pip install tensorboard-plugin-neuron

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++ python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.*

# Optional: Install Neuron TensorFlow model server
sudo yum install tensorflow-model-server-neuron=2.1.4.2.0.4.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages

---

3.1. Installation Guide
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.*

# Optional: Install Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron=2.1.4.2.0.4.0 -y

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo yum install aws-neuron-tools -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron
```

(continues on next page)
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python3 -m ipykernel install --user --name tensorflow_venv --display-name
"Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.*

# Optional: Install Neuron TensorFlow model server
sudo yum install tensorflow-model-server-neuron=2.1.4.2.0.4.0 -y

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  
  # sudo systemctl stop neuron-rtd'

(continues on next page)
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
# Install OS headers
sudo apt-get install linux-headers-$\{(uname -r)\} -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Install Neuron TensorFlow
pip install tensorboard-plugin-neuron
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.* neuron-cc

# Optional: Install Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron=1.15.0.2.0.4.0 -y
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo yum install aws-neuron-tools -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron

export PATH=/opt/aws/neuron/bin:$PATH
```

(continues on next page)
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name
"""Python (Neuron TensorFlow)"

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.* neuron-cc

# Optional: Install Neuron TensorFlow model server
sudo yum install tensorflow-model-server-neuron=1.15.0.2.0.4.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version_
# may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release"_
# instruction on Neuron documentation

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous_
# releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install_
# or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
Note:  For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install OS headers
sudo yum install kernel-devel-$\langle$uname $-$r$\rangle$ kernel-headers-$\langle$uname $-$r$\rangle$ $-$y

# Install Neuron Driver
sudo yum install aws-neuron-dkms $-$y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after $-$reboot

# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version $-$may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" $-$instruction on Neuron documentation

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous $-$releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install $-$or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$\langle$uname $-$r$\rangle$ kernel-headers-$\langle$uname $-$r$\rangle$ $-$y

# Install Neuron Driver
sudo yum install aws-neuron-dkms $-$y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after $-$reboot
Compile on compute instance

If model compilation occurs outside the model deployment environment, you can install only the Neuron framework extensions and the compiler on any compute instance. This setup is helpful when compiling large complex models that require large amount of memory or during a CICD process where models are compiled in a separate step, prior to deployment.

TensorFlow 2.5.1
TensorFlow 2.4.3
TensorFlow 2.3.4
TensorFlow 2.2.3
TensorFlow 2.1.4
TensorFlow 1.15.5
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]

Ubuntu AMI
Amazon Linux AMI
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.3.*
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.3.*
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip
```

(continues on next page)
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.3.*

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.3.*
```

Ubuntu AMI

Amazon Linux AMI

Ubuntu DLAMI

Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.4.*
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

```
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.4.*
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.4.*
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Install Neuron TensorFlow

```bash
pip install tensorflow-neuron[cc]==2.3.4.*
```

## Ubuntu AMI

## Amazon Linux AMI

## Ubuntu DLAMI

## Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.*
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.*
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment
```

(continues on next page)
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.*

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.*
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.*

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.*

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.*

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

(continues on next page)
# Install Python venv and activate Python virtual environment to install # Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.*

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Install Python venv and activate Python virtual environment to install # Neuron pip packages.
sudo apt-get install --yes python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.* neuron-cc

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Install Python venv and activate Python virtual environment to install # Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

(continues on next page)
# Install Neuron TensorFlow

```bash
pip install tensorflow-neuron==1.15.5.* neuron-cc
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation

# Activate TensorFlow
source activate aws_neuron_tensorflow_p36
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation

# Activate TensorFlow
source activate aws_neuron_tensorflow_p36
```

**Deploy on AWS ML accelerator instance**

During deployment it can be beneficial to reduce the number of components installed in the system. For use-cases where only inference is necessary (compilation is already complete), only the framework and runtime should be installed.

**Important:**

**For successful installation or update to Neuron 1.16.0 and newer from previous releases:**

- Stop Neuron Runtime 1.x daemon (`neuron-rtd`) by running: `sudo systemctl stop neuron-rtd`
- Uninstall `neuron-rtd` by running: `sudo apt remove aws-neuron-runtime` or `sudo yum remove aws-neuron-runtime`
- Install or upgrade to latest Neuron driver (`aws-neuron-dkms`) by following the “Setup Guide” instructions.
- Visit *Introducing Neuron Runtime 2.x (libnrt.so)* for more information.

TensorFlow 2.5.1
TensorFlow 2.4.3
# Configure Linux for Neuron repository updates
```
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -
```

# Update OS packages
```
sudo apt-get update -y
```

# Before installing or updating aws-neuron-dkms:
```
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
```

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
```
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
```

# Install OS headers
```
sudo apt-get install linux-headers-$\{uname -r\} -y
```

# Install Neuron Driver
```
sudo apt-get install aws-neuron-dkms -y
```

# Warning: If Linux kernel is updated as a result of OS package update
```
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

(continues on next page)
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron

# Optional: Install Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron -y

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-
AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous
# releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
#   or upgrade to latest Neuron driver

(continues on next page)
# Install OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron

# Optional: Install Neuron TensorFlow model server
sudo yum install tensorflow-model-server-neuron -y

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   →'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

#########################################################################

Warning: If Linux kernel is updated as a result of OS package update
Neuron driver (aws-neuron-dkms) should be re-installed after reboot
#########################################################################

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
# sudo apt-get install -y python3-venv g++
# python3 -m venv tensorflow_venv
# source tensorflow_venv/bin/activate
# pip install -U pip

# Set Pip repository to point to the Neuron repository
# pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
# pip install tensorflow-neuron

# Optional: Install Neuron TensorFlow model server
# sudo apt-get install tensorflow-model-server-neuron -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

#########################################################################

Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
#########################################################################
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$\{(\text{uname} -r)\} kernel-headers-$\{(\text{uname} -r)\} -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver ('aws-neuron-dkms') should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$\{PATH\}

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron

# Optional: Install Neuron TensorFlow model server
sudo yum install tensorflow-model-server-neuron -y

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
. /etc/os-release

(continues on next page)
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
eof
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$ (uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.4.3.*

# Optional: Install Neuron TensorFlow model server

(continues on next page)
sudo apt-get install tensorflow-model-server-neuron=2.4.3.2.0.4.0 -y

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```plaintext
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-
AWS-NEURON.PUB
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
# or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
```
**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Install OS headers
sudo apt-get install linux-headers-$\{\text{uname -r}\} -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Install OS headers
sudo apt-get install linux-headers-$\{\text{uname -r}\} -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot
```

(continues on previous page)
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.4.3.*

# Optional: Install Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron=2.4.3.2.0.4.0 -y

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
#   or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$({uname -r}) kernel-headers-$({uname -r}) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

(continues on next page)
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after.

```bash
export PATH=/opt/aws/neuron/bin:$PATH
```

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
```bash
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron TensorFlow
```bash
pip install tensorflow-neuron==2.4.3.*
```

# Optional: Install Neuron TensorFlow model server
```bash
sudo yum install tensorflow-model-server-neuron=2.4.3.2.0.4.0 -y
```

Ubuntu AMI

Amazon Linux AMI

Ubuntu DLAMI

Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
./etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y
```

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

(continues on next page)
# To install or update to Neuron versions 1.16.0 and newer from previous releases:

- DO NOT skip `aws-neuron-dkms` install or upgrade step, you MUST install or upgrade to latest Neuron driver

```bash
# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y
```

---

### Warning: If Linux kernel is updated as a result of OS package update

Neuron driver (`aws-neuron-dkms`) should be re-installed after reboot

---

```bash
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.3.4.*

# Optional: Install Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron=2.3.4.2.0.4.0 -y
```

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB
```
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
#   or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.3.4.*

# Optional: Install Neuron TensorFlow model server
sudo yum install tensorflow-model-server-neuron=2.3.4.0.4.0 -y

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   - 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++ python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.3.4.*

# Optional: Install Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron=2.3.4.2.0.4.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
#   or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.3.4.*

# Optional: Install Neuron TensorFlow model server
sudo yum install tensorflow-model-server-neuron=2.3.4.2.0.4.0 -y
```
# Configure Linux for Neuron repository updates
/etc/os-release

```
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -
```

# Update OS packages

```
sudo apt-get update -y
```

# Before installing or updating aws-neuron-dkms:

```
# To install or update to Neuron versions 1.16.0 and newer from previous releases:

# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
```

# Install OS headers

```
sudo apt-get install linux-headers-$(uname -r) -y
```

# Install Neuron Driver

```
sudo apt-get install aws-neuron-dkms -y
```

# Warning: If Linux kernel is updated as a result of OS package update

```
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```

```
export PATH=/opt/aws/neuron/bin:$PATH
```

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.2.3.*

# Optional: Install Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron=2.2.3.2.0.4.0 -y

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y
```
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot.

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.2.3.*

# Optional: Install Neuron TensorFlow model server
sudo yum install tensorflow-model-server-neuron=2.2.3.2.0.4.0 -y

---

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

---

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

---

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

(continues on next page)
# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after a
# reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron TensorFlow
pip install tensorflow-neuron==2.2.3.*

# Optional: Install Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron=2.2.3.2.0.4.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# ‘sudo systemctl stop neuron-rtd’

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
# Install OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update, Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.2.3.*

# Optional: Install Neuron TensorFlow model server
sudo yum install tensorflow-model-server-neuron=2.2.3.2.0.4.0 -y

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
./etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  → 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.1.4.*

# Optional: Install Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron=2.1.4.2.0.4.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
nname=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-
AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip
```
(continues on next page)
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.1.4.*

# Optional: Install Neuron TensorFlow model server
sudo yum install tensorflow-model-server-neuron=2.1.4.2.0.4.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   -> 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv  
source tensorflow_venv/bin/activate  
pip install -U pip

# Set Pip repository to point to the Neuron repository  
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow  
pip install tensorflow-neuron==2.1.4.*  

# Optional: Install Neuron TensorFlow model server  
sudo apt-get install tensorflow-model-server-neuron=2.1.4.2.0.4.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version  
# Framework will be installed/updated inside a Python environment

# Update OS packages  
sudo yum update -y

# Before installing or updating aws-neuron-dkms:  
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:  
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:  
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers  
sudo yum install kernel-devel-${uname -r} kernel-headers-${uname -r} -y

# Install Neuron Driver  
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update  
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

(continues on next page)
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.1.4.*

# Optional: Install Neuron TensorFlow model server
sudo yum install tensorflow-model-server-neuron=2.1.4.2.0.4.0 -y

Ubuntu AMI

Amazon Linux AMI

Ubuntu DLAMI

Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
./etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  sudo systemctl stop neuron-rtd

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

(continues on next page)
# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Set PATH
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.*

# Optional: Install Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron=1.15.0.2.0.4.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
   
   → 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.*

# Optional: Install Neuron TensorFlow model server
sudo yum install tensorflow-model-server-neuron=1.15.0.2.0.4.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`
# To install or update to Neuron versions 1.16.0 and newer from previous releases:

- DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

```
# Install OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y
```

#####

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

#####

# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

## 3.1.2 Update to latest Neuron TensorFlow

**Note:**

- Instructions in this page only apply to setting up Neuron components on Linux host running Ubuntu or Amazon Linux AMI.
- For an example of how to install Neuron components in a container, see Docker environment setup and our Containers documentation for more details.

- **Develop on AWS ML accelerator instance**
- **Compile on compute instance**
- **Deploy on AWS ML accelerator instance**

### Develop on AWS ML accelerator instance

The simplest environment setup for model development installs all Neuron SDK components directly on an AWS ML accelerator instance: the Neuron framework extensions, compiler, runtime, and tools. This will allow you to compile, execute, and performance tune your model, all in the same instance. This is the recommended workflow when first starting to work with Neuron device or when optimizing a model.

**Important:**

For successful installation or update to Neuron 1.16.0 and newer from previous releases:
• Stop Neuron Runtime 1.x daemon (neuron-rtd) by running: `sudo systemctl stop neuron-rtd`
• Uninstall neuron-rtd by running: `sudo apt remove aws-neuron-runtime` or `sudo yum remove aws-neuron-runtime`
• Install or upgrade to latest Neuron driver (aws-neuron-dkms) by following the “Setup Guide” instructions.
• Visit Introducing Neuron Runtime 2.x (libnrt.so) for more information.

TensorFlow 2.5.1
TensorFlow 2.4.3
TensorFlow 2.3.3
TensorFlow 2.2.3
TensorFlow 2.1.4
TensorFlow 1.15.5
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo apt-get install linux-headers-$(/bin/uname -r) -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y
```

(continues on next page)
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Update Neuron TensorBoard
pip install --upgrade tensorboard-plugin-neuron

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]

# Optional: Update Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo yum update aws-neuron-tools -y

# Update Neuron TensorBoard
pip install --upgrade tensorboard-plugin-neuron

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]

# Optional: Update Neuron TensorFlow model server
sudo yum update tensorflow-model-server-neuron -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  sudo systemctl stop neuron-rtd

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
# Update OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Update Neuron TensorBoard
pip install --upgrade tensorboard-plugin-neuron

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]

# Optional: Update Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron -y

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# - 'sudo systemctl stop neuron-rtd'

(continues on next page)
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo yum update aws-neuron-tools -y

# Update Neuron TensorBoard
pip install --upgrade tensorboard-plugin-neuron

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]

# Optional: Update Neuron TensorFlow model server
sudo yum update tensorflow-model-server-neuron -y

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y
# Before installing or updating `aws-neuron-dkms`:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
`sudo apt-get install linux-headers-$\text{(uname -r)}$ -y`

# Update Neuron Driver
`sudo apt-get install aws-neuron-dkms -y`

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
`sudo apt-get install aws-neuron-tools -y`

# Update Neuron TensorBoard
`pip install --upgrade tensorboard-plugin-neuron`

# Activate Python virtual environment where Neuron pip packages were installed
`source tensorflow_venv/bin/activate`

# Set Pip repository to point to the Neuron repository
`pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com`

# Update Neuron TensorFlow
`pip install --upgrade tensorflow-neuron[cc]==2.4.3.*`

# Optional: Update Neuron TensorFlow model server
`sudo apt-get install tensorflow-model-server-neuron=2.4.3.2.0.4.0 -y`

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo yum update aws-neuron-tools -y
# Update Neuron TensorBoard
pip install --upgrade tensorboard-plugin-neuron

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]==2.4.3.*

# Optional: Update Neuron TensorFlow model server
sudo yum update tensorflow-model-server-neuron=2.4.3.2.0.4.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Update Neuron TensorFlow
pip install --upgrade tensorboard-plugin-neuron

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]==2.4.3.*

# Optional: Update Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron=2.4.3.2.0.4.0 -y
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  → 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
  → or upgrade to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo yum update aws-neuron-tools -y

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]==2.4.3.*

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]==2.4.3.*
```
# Optional: Update Neuron TensorFlow model server
```
sudo yum update tensorflow-model-server-neuron=2.4.3.2.0.4.0 -y
```

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
__main__ : error: tensorflow-2.3.3 is not a supported framework
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
__main__ : error: tensorflow-2.3.3 is not a supported framework
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
__main__ : error: tensorflow-2.3.3 is not a supported framework
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
__main__ : error: tensorflow-2.3.3 is not a supported framework
```

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Update OS packages
sudo apt-get update -y
```

(continues on next page)
# Before installing or updating aws-neuron-dkms:

- Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  
  `sudo systemctl stop neuron-rtd`

To install or update to Neuron versions 1.16.0 and newer from previous releases:

- DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

Update OS headers

```bash
sudo apt-get install linux-headers-$\langle$uname -r$\rangle$ -y
```

Update Neuron Driver

```bash
sudo apt-get install aws-neuron-dkms -y
```

Warning: If Linux kernel is updated as a result of OS package update Neuron driver (aws-neuron-dkms) should be re-installed after reboot

Update Neuron Tools

```bash
sudo apt-get install aws-neuron-tools -y
```

Update Neuron TensorFlow

```bash
pip install --upgrade tensorflow-neuron[cc]==2.2.3.*
```

Optional: Update Neuron TensorFlow model server

```bash
sudo apt-get install tensorflow-model-server-neuron=2.2.3.2.0.4.0 -y
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages

```bash
sudo yum update -y
```
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  → `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo yum update aws-neuron-tools -y

# Update Neuron TensorBoard
pip install --upgrade tensorboard-plugin-neuron

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]==2.2.3.*

# Optional: Update Neuron TensorFlow model server
sudo yum update tensorflow-model-server-neuron=2.2.3.2.0.4.0 -y

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   -> 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous
# releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
#   or upgrade to latest Neuron driver

# Update OS headers
sudo apt-get install linux_headers-$(uname -r) -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Update Neuron TensorBoard
pip install --upgrade tensorboard-plugin-neuron

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]==2.2.3.*

# Optional: Update Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron=2.2.3.2.0.4.0 -y
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  # 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
  # or upgrade to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$ (uname -r) kernel-headers-$ (uname -r) -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo yum update aws-neuron-tools -y

# Update Neuron TensorBoard
pip install --upgrade tensorboard-plugin-neuron

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]==2.2.3.*
```

(continues on next page)
# Optional: Update Neuron TensorFlow model server
sudo yum update tensorflow-model-server-neuron=2.2.3.2.0.4.0 -y

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
#   or upgrade to latest Neuron driver

# Update OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Update Neuron TensorBoard
pip install --upgrade tensorboard-plugin-neuron

# Activate Python virtual environment where Neuron pip packages were installed
```

(continues on next page)
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]==2.1.4.*

# Optional: Update Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron=2.1.4.2.0.4.0 -y

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   -> 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# - Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo yum update aws-neuron-tools -y

# Update Neuron TensorBoard
pip install --upgrade tensorboard-plugin-neuron
# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]==2.1.4.*

# Optional: Update Neuron TensorFlow model server
sudo yum update tensorflow-model-server-neuron=2.1.4.2.0.4.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Update Neuron TensorBoard
pip install --upgrade tensorboard-plugin-neuron

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]==2.1.4.*

# Optional: Update Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron=2.1.4.2.0.4.0 -y

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Update OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

(continues on next page)
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo yum update aws-neuron-tools -y

# Update Neuron TensorBoard
pip install --upgrade tensorboard-plugin-neuron

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]==2.1.4.*

# Optional: Update Neuron TensorFlow model server
sudo yum update tensorflow-model-server-neuron=2.1.4.2.0.4.0 -y

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  # 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

(continues on next page)
# Update OS headers
sudo apt-get install linux-headers-$\{(uname -r)\} -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

#############################################################################

→

#######
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
→reboot
→

#############################################################################

→

#######

# Update Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Update Neuron TensorBoard
pip install --upgrade tensorboard-plugin-neuron

# Activate Python virtual environment where Neuron pip packages were
→installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron==1.15.5.* neuron-cc

# Optional: Update Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron=1.15.0.2.0.4.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

#############################################################################

→

##################################
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
→'sudo systemd stop neuron-rtd'
→

#############################################################################

→

##################################

#############################################################################

→

###################################
# To install or update to Neuron versions 1.16.0 and newer from previous
→releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
→or upgrade to latest Neuron driver

(continues on next page)
# Update OS headers
```bash
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y
```

# Update Neuron Driver
```bash
sudo yum update aws-neuron-dkms -y
```

# Warning: If Linux kernel is updated as a result of OS package update
```bash
# Neuron driver (aws-neuron-dkms) should be re-installed after
reboot
```

# Update Neuron Tools
```bash
sudo yum update aws-neuron-tools -y
```

# Update Neuron TensorFlow
```bash
pip install --upgrade tensorboard-plugin-neuron
```

# Activate Python virtual environment where Neuron pip packages were installed
```bash
source tensorflow_venv/bin/activate
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Update Neuron TensorFlow
```bash
pip install --upgrade tensorflow-neuron==1.15.5.* neuron-cc
```

# Optional: Update Neuron TensorFlow model server
```bash
sudo yum update tensorflow-model-server-neuron=1.15.0.2.0.4.0 -y
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Update OS packages
sudo apt-get update -y
```

# Before installing or updating aws-neuron-dkms:
```bash
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
sudo systemctl stop neuron-rtd
```

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
```bash
```
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
```bash
sudo apt-get install linux-headers-$\{uname -r\} -y
```

# Update Neuron Driver
```bash
sudo apt-get install aws-neuron-dkms -y
```

# Warning: If Linux kernel is updated as a result of OS package update, the Neuron driver (aws-neuron-dkms) should be re-installed after a reboot.

# Update Neuron Tools
```bash
sudo apt-get install aws-neuron-tools -y
```

# Update Neuron TensorBoard
```bash
pip install --upgrade tensorboard-plugin-neuron
```

# Activate TensorFlow
```bash
source activate aws_neuron_tensorflow_p36
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Update Neuron TensorFlow
```bash
pip install --upgrade tensorflow-neuron==1.15.5.* neuron-cc
```

# Optional: Update Neuron TensorFlow model server
```bash
sudo apt-get install tensorflow-model-server-neuron=1.15.0.2.0.4.0 -y
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Update OS packages
sudo yum update -y
```

# Before installing or updating aws-neuron-dkms:
```bash
# Before installing or updating aws-neuron-dkms:
```

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
```bash
... (continues on next page)
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

```
# Update OS headers
sudo yum install kernel-devel-$\{(uname -r)\} kernel-headers-$\{(uname -r)\} -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y
```

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

```
# Update Neuron Tools
sudo yum update aws-neuron-tools -y

# Update Neuron TensorBoard
pip install --upgrade tensorboard-plugin-neuron

# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron==1.15.5.* neuron-cc

# Optional: Update Neuron TensorFlow model server
sudo yum update tensorflow-model-server-neuron=1.15.0.2.0.4.0 -y
```

### Compile on compute instance

If model compilation occurs outside the model deployment environment, you can install only the Neuron framework extensions and the compiler on any compute instance. This setup is helpful when compiling large complex models that require large amount of memory or during a CICD process where models are compiled in a separate step, prior to deployment.

- TensorFlow 2.5.1
- TensorFlow 2.4.3
- TensorFlow 2.3.3
- TensorFlow 2.2.3
- TensorFlow 2.1.4
- TensorFlow 1.15.5
- Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]
```

Ubuntu AMI

Amazon Linux AMI

Ubuntu DLAMI

Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]==2.4.3.*
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]==2.4.3.*
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

```python
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]==2.4.3.*

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

__main__ : error: tensorflow-2.3.3 is not a supported framework

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

__main__ : error: tensorflow-2.3.3 is not a supported framework
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

___main___ : error: tensorflow-2.3.3 is not a supported framework

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

___main___ : error: tensorflow-2.3.3 is not a supported framework

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]==2.2.3.*

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]==2.2.3.*

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
copy the contents of the code block into a script file and source its contents.

```bash
# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]==2.1.4.*
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]==2.1.4.*
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron[cc]==2.1.4.*
```

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron==1.15.5.* neuron-cc
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron==1.15.5.* neuron-cc
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron==1.15.5.* neuron-cc
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron==1.15.5.* neuron-cc
```

(continues on next page)
# Update Neuron TensorFlow

```
pip install --upgrade tensorflow-neuron==1.15.5.* neuron-cc
```

**Deploy on AWS ML accelerator instance**

During deployment it can be beneficial to reduce the number of components installed in the system. For use-cases where only inference is necessary (compilation is already complete), only the framework and runtime should be installed.

**Important:**

**For successful installation or update to Neuron 1.16.0 and newer from previous releases:**

- Stop Neuron Runtime 1.x daemon (`neuron-rtd`) by running: `sudo systemctl stop neuron-rtd`
- Uninstall `neuron-rtd` by running: `sudo apt remove aws-neuron-runtime` or `sudo yum remove aws-neuron-runtime`
- Install or upgrade to latest Neuron driver (`aws-neuron-dkms`) by following the “Setup Guide” instructions.
- Visit *Introducing Neuron Runtime 2.x (libnrt.so)* for more information.

TensorFlow 2.5.1
TensorFlow 2.4.3
TensorFlow 2.3.3
TensorFlow 2.2.3
TensorFlow 2.1.4
TensorFlow 1.15.5
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`
```

(continues on next page)
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo apt-get install linux-headers-$\{\text{uname} -r\} -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron

# Optional: Update Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron -y

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Update OS packages
sudo yum update -y

---

Chapter 3. TensorFlow Neuron

518
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
# Update OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
# Update Neuron Driver
sudo yum update aws-neuron-dkms -y
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron
# Optional: Update Neuron TensorFlow model server
sudo yum update tensorflow-model-server-neuron -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment
# Update OS packages
sudo apt-get update -y
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
#   'sudo systemctl stop neuron-rtd'

(continues on next page)
To install or update to Neuron versions 1.16.0 and newer from previous releases:

- DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

```
# Update OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y
```

Warning: If Linux kernel is updated as a result of OS package update Neuron driver (aws-neuron-dkms) should be re-installed after reboot

```
# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron

# Optional: Update Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron -y
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'
```

(continues on next page)
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
# Update OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron

# Optional: Update Neuron TensorFlow model server
sudo yum update tensorflow-model-server-neuron -y

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

---

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
- DO NOT skip `aws-neuron-dkms` install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
```
sudo apt-get install linux-headers-$\{uname -r\} -y
```

# Update Neuron Driver
```
sudo apt-get install aws-neuron-dkms -y
```

# Warning: If Linux kernel is updated as a result of OS package update, Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate Python virtual environment where Neuron pip packages were installed
```
source tensorflow_venv/bin/activate
```

# Set Pip repository to point to the Neuron repository
```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Update Neuron TensorFlow
```
pip install --upgrade tensorflow-neuron==2.4.3.*
```

# Optional: Update Neuron TensorFlow model server
```
sudo apt-get install tensorflow-model-server-neuron=2.4.3.2.0.4.0 -y
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Update OS packages
```
sudo yum update -y
```

# Before installing or updating aws-neuron-dkms:
- Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  ```
sudo systemctl stop neuron-rtd
  ```

---
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
# Update OS headers
sudo yum install kernel-devel-$\{(uname -r)\} kernel-headers-$\{(uname -r)\} -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron==2.4.3.*

# Optional: Update Neuron TensorFlow model server
sudo yum update tensorflow-model-server-neuron=2.4.3.2.0.4.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

(continues on next page)
# To install or update to Neuron versions 1.16.0 and newer from previous releases:

- DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers

```
sudo apt-get install linux-headers-$(uname -r) -y
```

# Update Neuron Driver

```
sudo apt-get install aws-neuron-dkms -y
```

WARNING: If Linux kernel is updated as a result of OS package update

# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate Python virtual environment where Neuron pip packages were installed

```
source tensorflow_venv/bin/activate
```

# Set Pip repository to point to the Neuron repository

```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Update Neuron TensorFlow

```
pip install --upgrade tensorflow-neuron==2.4.3.*
```

# Optional: Update Neuron TensorFlow model server

```
sudo apt-get install tensorflow-model-server-neuron=2.4.3.2.0.4.0 -y
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Note: There is no DLAMI Conda environment for this framework version

# Framework will be installed/updated inside a Python environment

# Update OS packages

```
sudo yum update -y
```

# Before installing or updating aws-neuron-dkms:

- Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:

```
sudo systemctl stop neuron-rtd
```

---

(continues on next page)
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
- DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

```
# Update OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y
```

---

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

---

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron==2.4.3.*

# Optional: Update Neuron TensorFlow model server
sudo yum update tensorflow-model-server-neuron=2.4.3.2.0.4.0 -y

---

Ubuntu AMI

Amazon Linux AMI

Ubuntu DLAMI

Amazon Linux DLAMI

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

---

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```python
__main__ : error: tensorflow-2.3.3 is not a supported framework
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```python
__main__ : error: tensorflow-2.3.3 is not a supported framework
```

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo apt-get install linux-headers-$uname -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```

(continues on next page)
# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron==2.2.3.*

# Optional: Update Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron=2.2.3.0.4.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate Python virtual environment where Neuron pip packages were installed

(continues on next page)
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron==2.2.3.*

# Optional: Update Neuron TensorFlow model server
sudo yum update tensorflow-model-server-neuron=2.2.3.0.4.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate Python virtual environment where Neuron pip packages were installed

(continues on next page)
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron==2.2.3.*

# Optional: Update Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron=2.2.3.2.0.4.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$uname -r$ kernel-headers-$uname -r$ -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate Python virtual environment where Neuron pip packages were installed

(continues on next page)
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron==2.2.3.*

# Optional: Update Neuron TensorFlow model server
sudo yum update tensorflow-model-server-neuron=2.2.3.2.0.4.0 -y

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron==2.1.4.*

# Optional: Update Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron=2.1.4.2.0.4.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname $uname -r -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron==2.1.4.*

# Optional: Update Neuron TensorFlow model server
sudo yum update tensorflow-model-server-neuron=2.1.4.2.0.4.0 -y

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo apt-get install linux-headers-$\{(uname -r)\} -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron==2.1.4.*

# Optional: Update Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron=2.1.4.2.0.4.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$ (uname -r) kernel-headers-$ (uname -r) -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron==2.1.4.*

# Optional: Update Neuron TensorFlow model server
sudo yum update tensorflow-model-server-neuron=2.1.4.2.0.4.0 -y

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron==1.15.5.*

# Optional: Update Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron=1.15.0.2.0.4.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate Python virtual environment where Neuron pip packages were installed
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron==1.15.5.*

# Optional: Update Neuron TensorFlow model server
sudo yum update tensorflow-model-server-neuron=1.15.0.2.0.4.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo apt-get install linux-headers-$\{(uname -r)\} -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron==1.15.5.*

# Optional: Update Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron=1.15.0.2.0.4.0 -y

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous
# releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
#   or upgrade to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after

# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron TensorFlow
pip install --upgrade tensorflow-neuron==1.15.5.*

# Optional: Update Neuron TensorFlow model server
sudo yum update tensorflow-model-server-neuron=1.15.0.2.0.4.0 -y
```
3.1.3 Install previous Neuron TensorFlow releases

Note:

- Instructions in this page only apply to setting up Neuron components on Linux host running Ubuntu or Amazon Linux AMI.
- For an example of how to install Neuron components in a container, see Docker environment setup and our Containers documentation for more details.

Install Neuron TensorFlow (Neuron 1.15.2)

- Develop on AWS ML accelerator instance
- Compile on compute instance
- Deploy on AWS ML accelerator instance

Develop on AWS ML accelerator instance

The simplest environment setup for model development installs all Neuron SDK components directly on an AWS ML accelerator instance: the Neuron framework extensions, compiler, runtime, and tools. This will allow you to compile, execute, and performance tune your model, all in the same instance. This is the recommended workflow when first starting to work with Neuron device or when optimizing a model.

TensorFlow 2.5.0
TensorFlow 2.4.2
TensorFlow 2.3.3
TensorFlow 2.2.3
TensorFlow 2.1.4
TensorFlow 1.15.5
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -
```
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name
   "Python (Neuron TensorFlow)"
ip install jupyter notebook
pip install environment_kernels
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.5.0.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.5.1.1.6.10.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
→ using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
→ using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install --y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name
→ "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.5.0.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
→ using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.5.1.1.6.10.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
→ 'sudo systemctl stop neuron-rtd'

(continues on next page)
# Install OS headers
```bash
sudo apt-get install linux-headers-$\text{(uname -r)}$ -y
```

# Install Neuron Driver
```bash
# If you are downgrading from newer version, please add '---allow-downgrades'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
```bash
# If you are downgrading from newer version, please add '---allow-downgrades'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y
```

# Install Neuron Tools
```bash
# If you are downgrading from newer version, please add '---allow-downgrades'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y
```

# Install Neuron TensorBoard
```bash
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH
```

# Activate TensorFlow
```bash
source activate
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url \(https://pip.repos.neuron.amazonaws.com\)
```

# Install Neuron TensorFlow
```bash
pip install tensorflow-neuron[cc]==2.5.0.1.6.10.0
```

# Optional: Install Neuron TensorFlow model server
```bash
# If you are downgrading from newer version, please add '---allow-downgrades'
sudo apt-get install tensorflow-model-server-neuron=2.5.1.1.6.10.0 -y
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Update OS packages
sudo yum update -y
```

(continues on next page)
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  → 'sudo systemctl stop neuron-rtd'
#############################################################################
→##################################
# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y
#############################################################################
→#######
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
#############################################################################
→#######
# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y
# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y
# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH
# Activate TensorFlow
source activate
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.5.0.1.6.10.0
# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.5.1.1.6.10.0 -y

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com $(VERSION_CODENAME) main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
```

(continues on next page)
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.2.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.4.1.1.6.10.0 -y

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
```

(continues on next page)
sudo yum install aws-neuron-dkms=2.1.5.0 -y

#------------------------------------------------------------------------------
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot
#------------------------------------------------------------------------------

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name
"Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.2.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.4.1.1.6.10.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.2.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.4.1.1.6.10.0 -y

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

```bash
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install TensorFlow
pip install tensorflow-neuron[cc]==2.4.2.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.4.1.1.6.10.0 -y
```
# Configure Linux for Neuron repository updates
/etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${{VERSION_CODENAME}} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard

(continues on next page)
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name
"Python (Neuron TensorFlow)"
ip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.3.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.3.0.1.6.10.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

(continues on next page)
# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.3.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package

(continues on next page)
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorFlow
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
```
pip install tensorflow-neuron[cc]==2.3.3.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.3.0.1.6.10.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate
# Set Pip repository to point to the Neuron repository

pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow

pip install tensorflow-neuron[cc]==2.3.3.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package

```
sudo yum install tensorflow-model-server-neuron=2.3.0.1.6.10.0 -y
```

Ubuntu AMI

Amazon Linux AMI

Ubuntu DLAMI

Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Configure Linux for Neuron repository updates
./etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'

```
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

```
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after...
```
# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
→ option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
→ option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name
→ "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
→ option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.2.2.1.6.10.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
(continues on next page)
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   -> sudo systemctl stop neuron-rtd

# Install OS headers
sudo yum install kernel-devel-$uname_r kernel-headers-$uname_r -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name
""
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.2.2.1.6.10.0 -y

---

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
sudo systemctl stop neuron-rtd

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools

---

(continues on next page)
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.2.2.1.6.10.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#  'sudo systemctl stop neuron-rtd'

# Install OS headers

(continues on next page)
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package, using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package, using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package, using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package, using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.2.2.1.6.10.0 -y
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
/etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com $(VERSION_CODENAME) main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard

(continues on next page)
```
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name
"Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.1.4.1.6.10.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#  'sudo systemctl stop neuron-rtd'

(continues on next page)
# Install OS headers
```
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
```

# Install Neuron Driver
```
# If you are downgrading from newer version, please remove existing package
→ using 'sudo yum remove' before installing the older package

sudo yum install aws-neuron-dkms=2.1.5.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
→ reboot
```
```

# Install Neuron Runtime server
```
# If you are downgrading from newer version, please remove existing package
→ using 'sudo yum remove' before installing the older package

sudo yum install aws-neuron-runtime=1.6.24.0 -y
```

# Install Neuron Tools
```
# If you are downgrading from newer version, please remove existing package
→ using 'sudo yum remove' before installing the older package

sudo yum install aws-neuron-tools=1.7.25.0 -y
```

# Install Neuron TensorBoard
```
pip install tensorboard-plugin-neuron==2.1.2.0
```

```
export PATH=/opt/aws/neuron/bin:$PATH
```

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
```
sudo yum install -y python3 gcc-c++ python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip
```

# Install Jupyter notebook kernel
```
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name
→ "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels
```

# Set Pip repository to point to the Neuron repository
```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron TensorFlow
```
pip install tensorflow-neuron[cc]==2.1.4.1.6.10.0
```

# Optional: Install Neuron TensorFlow model server
```
# If you are downgrading from newer version, please remove existing package
→ using 'sudo yum remove' before installing the older package
```

(continues on next page)
sudo yum install tensorflow-model-server-neuron=2.1.4.1.6.10.0 -y

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
```

(continues on next page)
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name
  "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.1.4.1.6.10.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

(continues on next page)
# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name
"Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.1.4.1.6.10.0 -y

Ubuntu AMI
Amazon Linux AMI

Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"

pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.10.0 neuron-cc==1.6.13.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=1.15.0.1.6.10.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
(continues on next page)
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
# pip install tensorflow-neuron==1.15.5.1.6.10.0 neuron-cc==1.6.13.0

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"

pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron=1.15.5.1.6.10.0 neuron-cc==1.6.13.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=1.15.0.1.6.10.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

(continues on next page)
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   → 'sudo systemctl stop neuron-rtd'
射手 phóng tên lửa
# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y
# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y
# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y
# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH
# Activate TensorFlow
source activate aws_neuron_tensorflow_p36
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.10.0 neuron-cc==1.6.13.0
# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=1.15.0.1.6.10.0 -y

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.10.0 neuron-cc==1.6.13.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=1.15.0.1.6.10.0 -y
Compile on compute instance

If model compilation occurs outside the model deployment environment, you can install only the Neuron framework extensions and the compiler on any compute instance. This setup is helpful when compiling large complex models that require large amount of memory or during a CICD process where models are compiled in a separate step, prior to deployment.

TensorFlow 2.5.0
TensorFlow 2.4.2
TensorFlow 2.3.3
TensorFlow 2.2.3
TensorFlow 2.1.4
TensorFlow 1.15.5
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install # Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.5.0.1.6.10.0
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install # Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```
# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.5.0.1.6.10.0

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.5.0.1.6.10.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.5.0.1.6.10.0
```

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.2.1.6.10.0
```
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.2.1.6.10.0
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.2.1.6.10.0
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.2.1.6.10.0
```

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install ~U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.3.1.6.10.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install ~U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.3.1.6.10.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.3.1.6.10.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.3.1.6.10.0

(continues on next page)
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.3.1.6.10.0

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.1.6.10.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.1.6.10.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.1.6.10.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.1.6.10.0

(continues on next page)
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.1.6.10.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install 
# Neuron pip packages.
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.1.6.10.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Install Python venv and activate Python virtual environment to install 
# Neuron pip packages.
sudo apt-get install -y python3-gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.1.6.10.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

(continues on next page)
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.1.6.10.0

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.10.0 neuron-cc==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Install Neuron TensorFlow

```
pip install tensorflow-neuron==1.15.5.1.6.10.0 neuron-cc==1.6.13.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.10.0 neuron-cc==1.6.13.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.10.0 neuron-cc==1.6.13.0
```

**Deploy on AWS ML accelerator instance**

During deployment it can be beneficial to reduce the number of components installed in the system. For use-cases where only inference is necessary (compilation is already complete), only the framework and runtime should be installed.

- TensorFlow 2.5.0
- TensorFlow 2.4.2
- TensorFlow 2.3.3
- TensorFlow 2.2.3
- TensorFlow 2.1.4
- TensorFlow 1.15.5
- Ubuntu AMI
- Amazon Linux AMI
- Ubuntu DLAMI
- Amazon Linux DLAMI
# Configure Linux for Neuron repository updates
`. /etc/os-release`  
`sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF`  
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main EOF`  
`wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -`  

# Update OS packages
`sudo apt-get update -y`  

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`  

# Install OS headers
`sudo apt-get install linux-headers-$(uname -r) -y`  

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
`sudo apt-get install aws-neuron-dkms=2.1.5.0 -y`  

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
`sudo apt-get install aws-neuron-runtime=1.6.24.0 -y`  

`export PATH=/opt/aws/neuron/bin:$PATH`  

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
`sudo apt-get install -y python3-venv g++`  
`python3 -m venv tensorflow_venv`  
`source tensorflow_venv/bin/activate`  
`pip install -U pip`  

# Set Pip repository to point to the Neuron repository
`pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com`  

(continues on next page)
# Install Neuron TensorFlow

```bash
pip install tensorflow-neuron==2.5.0.1.6.10.0
```

# Optional: Install Neuron TensorFlow model server

```bash
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.5.1.1.6.10.0 -y
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y
```

(continues on next page)
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.5.0.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.5.1.1.6.10.0 -y

## Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot.

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'

(continues on next page)
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.5.0.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.5.1.1.6.10.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH
# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.5.0.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.5.1.1.6.10.0 -y

Ubuntu AMI

Amazon Linux AMI

Ubuntu DLAMI

Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
/etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
(continues on next page)
## Neuron driver (aws-neuron-dkms) should be re-installed after reboot

### Install Neuron Runtime server
- If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'

```bash
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y
```

- Export PATH:

```bash
export PATH=/opt/aws/neuron/bin:$PATH
```

### Install Python venv and activate Python virtual environment to install Neuron pip packages.
- sudo apt-get install -y python3-venv g++
- python3 -m venv tensorflow_venv
- source tensorflow_venv/bin/activate
- pip install -U pip

### Set Pip repository to point to the Neuron repository
- pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

### Install Neuron TensorFlow
- pip install tensorflow-neuron==2.4.2.1.6.10.0

### Optional: Install Neuron TensorFlow model server
- If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'

```bash
sudo apt-get install tensorflow-model-server-neuron=2.4.1.1.6.10.0 -y
```

### Note:
For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

### Configure Linux for Neuron repository updates
```bash
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
```

- sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

- Update OS packages
```bash
sudo yum update -y
```

### Before installing or updating aws-neuron-dkms
- Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  ```bash
  sudo systemctl stop neuron-rtd
  ```

(continues on next page)
# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# WARNING: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.4.2.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.4.1.1.6.10.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.4.2.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.4.1.1.6.10.0 -y

---

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Update OS packages
sudo yum update -y

---

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

---

(continues on next page)
# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.4.2.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.4.1.1.6.10.0 -y

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
./etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

(continues on next page)
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add `--allow-downgrades` option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add `--allow-downgrades` option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.3.3.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add `--allow-downgrades` option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.3.0.1.6.10.0 -y

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Configure Linux for Neuron repository updates

```bash
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
```

```bash
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB
```

# Update OS packages

```bash
sudo yum update -y
```

# Before installing or updating aws-neuron-dkms:

```bash
'sudo systemctl stop neuron-rtd'
```

# Install OS headers

```bash
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y
```

# Install Neuron Driver

```bash
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server

```bash
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y
```

```bash
export PATH=/opt/aws/neuron/bin:$PATH
```

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.

```bash
sudo yum install --y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip
```

# Set Pip repository to point to the Neuron repository

```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron TensorFlow

(continues on next page)
pip install tensorflow-neuron==2.3.3.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.3.0.1.6.10.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.3.3.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.3.0.1.6.10.0 -y

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.3.3.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.3.0.1.6.10.0 -y
```
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.2.3.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
sudo apt-get install tensorflow-model-server-neuron=2.2.1.6.10.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# Install OS headers
sudo yum install kernel-devel-$ (uname -r) kernel-headers-$ (uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.2.3.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.2.2.1.6.10.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

(continues on next page)
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.2.3.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.2.2.1.6.10.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
pip install tensorflow
source tensorflow/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.2.3.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.2.2.1.6.10.0 -y

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
/etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudapt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudapt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudapt-get install'
sudapt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudapt-get install'
sudapt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudapt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.1.4.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudapt-get install'
sudapt-get install tensorflow-model-server-neuron=2.1.4.1.6.10.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Configure Linux for Neuron repository updates

```bash
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
```

```bash
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB
```

```bash
# Update OS packages
sudo yum update -y
```

# Before installing or updating aws-neuron-dkms:

- Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  ```bash
  sudo systemctl stop neuron-rtd
  ```

# Install OS headers

```bash
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
```

# Install Neuron Driver

- If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package

```bash
sudo yum install aws-neuron-dkms=2.1.5.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update

- Reboot

# Install Neuron Runtime server

- If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package

```bash
sudo yum install aws-neuron-runtime=1.6.24.0 -y
```

```bash
export PATH=/opt/aws/neuron/bin:$PATH
```

# Install Python venv and activate Python virtual environment to install

- Neuron pip packages.

```bash
sudo yum install -y python3 gcc-c++ python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip
```

# Set Pip repository to point to the Neuron repository

```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron TensorFlow

(continues on next page)
# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.1.4.1.6.10.0 -y

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.1.4.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.1.4.1.6.10.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.

(continues on next page)
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.1.4.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.1.4.1.6.10.0 -y

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after.
<-> reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
<-> option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
<-> option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=1.15.0.1.6.10.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
directory=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'
# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=1.15.0.1.6.10.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
→'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
→option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
→reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
→option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
→option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=1.15.0.1.6.10.0 -y

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
→'sudo systemctl stop neuron-rtd'
# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.10.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove before installing the older package
sudo yum install tensorflow-model-server-neuron=1.15.0.1.6.10.0 -y

Install Neuron TensorFlow (Neuron 1.15.1)

- Develop on AWS ML accelerator instance
- Compile on compute instance
- Deploy on AWS ML accelerator instance
Develop on AWS ML accelerator instance

The simplest environment setup for model development installs all Neuron SDK components directly on an AWS ML accelerator instance: the Neuron framework extensions, compiler, runtime, and tools. This will allow you to compile, execute, and performance tune your model, all in the same instance. This is the recommended workflow when first starting to work with Neuron device or when optimizing a model.

TensorFlow 2.5.0
TensorFlow 2.4.2
TensorFlow 2.3.3
TensorFlow 2.2.3
TensorFlow 2.1.4
TensorFlow 1.15.5
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

```

(continues on next page)
# Warning: If Linux kernel is updated as a result of OS package update
Neuron driver (aws-neuron-dkms) should be re-installed after reboot.

# Install Neuron Runtime server
If you are downgrading from newer version, please add ' --allow-downgrades'
option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
If you are downgrading from newer version, please add '
 --allow-downgrades'
option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name
"Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.5.0.1.6.8.0

# Optional: Install Neuron TensorFlow model server
If you are downgrading from newer version, please add '
--allow-downgrades'
option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.5.1.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF

sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version , please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version , please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version , please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

(continues on next page)
# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.5.0.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.5.1.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# - 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y
# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.5.0.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.5.1.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

(continues on next page)
# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.5.0.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.5.1.1.6.8.0 -y

### Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
```
# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '–allow-downgrades'
  \rightarrow option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
  \rightarrow reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '–allow-downgrades'
  \rightarrow option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '–allow-downgrades'
  \rightarrow option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name
  "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.2.1.6.8.0

# Optional: Install Neuron TensorFlow model server

(continues on next page)
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.4.1.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

(continues on next page)
# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.2.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.4.1.1.6.8.0 -y

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add 'allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

(continues on next page)
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot
#
# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y
#
# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y
#
# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH
#
# Activate TensorFlow
source activate
#
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
#
# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.2.1.6.8.0
#
# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.4.1.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y
#
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
#-
# Install OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

(continues on next page)
# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
```bash
sudo yum install aws-neuron-dkms=2.1.5.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
```bash
sudo yum install aws-neuron-runtime=1.6.24.0 -y
```

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
```bash
sudo yum install aws-neuron-tools=1.7.25.0 -y
```

# Install Neuron TensorBoard
```bash
pip install tensorboard-plugin-neuron==2.1.2.0
```
```bash
export PATH=/opt/aws/neuron/bin:$PATH
```

# Activate TensorFlow
```bash
source activate
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron TensorFlow
```bash
pip install tensorflow-neuron[cc]==2.4.2.1.6.8.0
```

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
```bash
sudo yum install tensorflow-model-server-neuron=2.4.1.1.6.8.0 -y
```

Ubuntu AMI
Amazon Linux AMI

Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
. /etc/os-release
```
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#  'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$\{(uname -r)\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel

(continues on next page)
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.3.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.3.0.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update...
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.3.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.3.0.1.6.8.0 -y

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Update OS packages
sudo apt-get update -y

# Update pip packages
pip install --upgrade pip

# Install Jupyter notebook kernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.3.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.3.0.1.6.8.0 -y
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
# Install OS headers
sudo apt-get install linux-headers-$\{\text{uname -r}\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.3.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.3.0.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorFlow
# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.3.0.1.6.8.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.3.1.6.8.0

Ubuntu AMI
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
debs https://apt.repos.neuron.amazonaws.com $(VERSION_CODENAME) main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0
```
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.2.2.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
# Note:  Can also use :
#   'sudo systemctl stop neuron-rtd'
# Note: Can also use :
#   'sudo systemctl stop neuron-rtd'
# Note: Can also use :
#   'sudo systemctl stop neuron-rtd'
# Note: Can also use :
#   'sudo systemctl stop neutron-rtd'

(continues on next page)
# Install OS headers
```bash
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y
```

# Install Neuron Driver
```bash
# If you are downgrading from newer version, please remove existing package, using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update, Neuron driver (aws-neuron-dkms) should be re-installed after reboot.

# Install Neuron Runtime server
```bash
# If you are downgrading from newer version, please remove existing package, using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y
```

# Install Neuron Tools
```bash
# If you are downgrading from newer version, please remove existing package, using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y
```

# Install Neuron TensorBoard
```bash
pip install tensorboard-plugin-neuron==2.1.2.0
```

```bash
export PATH=/opt/aws/neuron/bin:$PATH
```

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
```bash
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
```

# Install Jupyter notebook kernel
```bash
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron TensorFlow
```bash
pip install tensorflow-neuron[cc]==2.2.3.1.6.8.0
```

# Optional: Install Neuron TensorFlow model server
```bash
# If you are downgrading from newer version, please remove existing package, using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.2.2.1.6.8.0 -y
```
For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$uname -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

(continues on next page)
```
# Instal Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.2.2.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$ (uname -r) kernel-headers-$ (uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.2.2.1.6.8.0 -y

---

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

(continues on next page)
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.1.4.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
→ using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
→ using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name
"Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
→ using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.1.4.1.6.8.0 -y

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
→'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
→option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
→reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
→option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
→option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name
"Python (Neuron TensorFlow)"
ip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.1.6.8.0
# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
→ option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.1.4.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
→ 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package,
→ using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
→ reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package,
→ using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package,
→ using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name
  "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazons.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.1.4.1.6.8.0 -y

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
./etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com $(VERSION_CODENAME) main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-
  NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

########################################################################
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
########################################################################

(continues on next page)
# Install OS headers
```
sudo apt-get install linux-headers-$\{uname -r\} -y
```

# Install Neuron Driver
```
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot
```

# Install Neuron Runtime server
```
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y
```

# Install Neuron Tools
```
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y
```

# Install Neuron TensorBoard
```
pip install tensorboard-plugin-neuron==2.1.2.0
```

```
export PATH=/opt/aws/neuron/bin:$PATH
```

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
```
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip
```

# Install Jupyter notebook kernel
```
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name
"Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels
```

# Set Pip repository to point to the Neuron repository
```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron TensorFlow
```
pip install tensorflow-neuron==1.15.5.1.6.8.0 neuron-cc==1.6.13.0
```

# Optional: Install Neuron TensorFlow model server
```
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
```

(continues on next page)
sudo apt-get install tensorflow-model-server-neuron=1.15.0.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
```

(continues on next page)
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.8.0 neuron-cc==1.6.13.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=1.15.0.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$ (uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

(continues on next page)
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot.

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.8.0 neuron-cc==1.6.13.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=1.15.0.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package

(continues on next page)
Compile on compute instance

If model compilation occurs outside the model deployment environment, you can install only the Neuron framework extensions and the compiler on any compute instance. This setup is helpful when compiling large complex models that require large amount of memory or during a CICD process where models are compiled in a separate step, prior to deployment.

TensorFlow 2.5.0
TensorFlow 2.4.2
TensorFlow 2.3.3
TensorFlow 2.2.3
TensorFlow 2.1.4
TensorFlow 1.15.5
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.5.0.1.6.8.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.5.0.1.6.8.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.5.0.1.6.8.0
```
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.5.0.1.6.8.0
```

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.2.1.6.8.0
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.2.1.6.8.0
```
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.2.1.6.8.0
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.2.1.6.8.0
```

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.3.1.6.8.0
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Install Python venv and activate Python virtual environment to install # Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install ~U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.3.1.6.8.0

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Activate TensorFlow
source activate
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.3.1.6.8.0

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Activate TensorFlow
source activate
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.3.1.6.8.0

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install # Neuron pip packages.
sudo apt-get install -y python3-venv g++

(continues on next page)
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.1.6.8.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.1.6.8.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install ~U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.1.6.8.0

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install ~U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.1.6.8.0

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install ~U pip

# Set Pip repository to point to the Neuron repository
(continues on next page)
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.1.6.8.0

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.1.6.8.0

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.1.6.8.0

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Install Python venv and activate Python virtual environment to install # Neuron pip packages.
sudo apt-get install -y python3-venv g++ python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.8.0 neuron-cc==1.6.13.0
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Install Python venv and activate Python virtual environment to install # Neuron pip packages.
sudo yum install -y python3 gcc-c++ python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.8.0 neuron-cc==1.6.13.0
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.8.0 neuron-cc==1.6.13.0
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
**AWS Neuron**

```
# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.8.0 neuron-cc==1.6.13.0
```

### Deploy on AWS ML accelerator instance

During deployment it can be beneficial to reduce the number of components installed in the system. For use-cases where only inference is necessary (compilation is already complete), only the framework and runtime should be installed.

TensorFlow 2.5.0  
TensorFlow 2.4.2  
TensorFlow 2.3.3  
TensorFlow 2.2.3  
TensorFlow 2.1.4  
TensorFlow 1.15.5  
Ubuntu AMI  
Amazon Linux AMI  
Ubuntu DLAMI  
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`
#```

(continues on next page)
# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# --reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron TensorFlow
pip install tensorflow-neuron==2.5.0.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.5.1.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

(continues on next page)
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.5.0.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.5.1.1.6.8.0 -y

---

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

```
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
#   option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# 'reboot'

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
#   option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.5.0.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
#   option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.5.1.1.6.8.0 -y
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Update OS packages
sudo yum update -y
```

(continues on next page)
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.5.0.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.5.1.1.6.8.0 -y

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Configure Linux for Neuron repository updates

```bash
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -
# Update OS packages
sudo apt-get update -y
```

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.4.2.1.6.8.0

# Optional: Install Neuron TensorFlow model server

(continues on next page)
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-
AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.4.2.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.4.1.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH
# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.4.2.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
    option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.4.1.1.6.8.0 -y

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#    'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
    using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

###
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
###

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
    using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.4.2.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.4.1.1.6.8.0 -y

### Ubuntu AMI

### Amazon Linux AMI

### Ubuntu DLAMI

### Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
.

/etc/os-release

```bash
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null
```"""EOF

def https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF

```bash
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -
```"""

# Update OS packages
```bash
sudo apt-get update -y
```"""

# Before installing or updating aws-neuron-dkms:
# Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'

```bash
sudo apt-get install linux-headers-$\(uname -r\) -y
```"""

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
```bash
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y
```"""

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

```
(continues on next page)
# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.3.3.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.3.0.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
# ###

# Install OS headers
sudo yum install kernel-devel-$uname r kernel-headers-$uname -y
# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

#############################################################################
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
#############################################################################

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.3.3.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.3.0.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

#############################################################################
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
# For successful installation of aws-neuron-dkms, execute 'sudo systemctl restart neuron-rtd'
#############################################################################
# Install OS headers
`sudo apt-get install linux-headers-$(uname -r) -y`

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
`sudo apt-get install aws-neuron-dkms=2.1.5.0 -y`

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
`sudo apt-get install aws-neuron-runtime=1.6.24.0 -y`

`export PATH=/opt/aws/neuron/bin:$PATH`

# Activate TensorFlow
`source activate`

# Set Pip repository to point to the Neuron repository
`pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com`

# Install Neuron TensorFlow
`pip install tensorflow-neuron==2.3.3.1.6.8.0`

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
`sudo apt-get install tensorflow-model-server-neuron=2.3.0.1.6.8.0 -y`

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
`sudo yum update -y`

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# Install OS headers
`sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y`

(continues on next page)
# Install Neuron Driver

If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package.

```
sudo yum install aws-neuron-dkms=2.1.5.0 -y
```

Warning: If Linux kernel is updated as a result of OS package update, the Neuron driver (aws-neuron-dkms) should be re-installed after reboot.

```
# Install Neuron Runtime server

If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package.

```
sudo yum install aws-neuron-runtime=1.6.24.0 -y
```

Export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow

source activate

# Set Pip repository to point to the Neuron repository

```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron TensorFlow

```
pip install tensorflow-neuron==2.3.3.1.6.8.0
```

# Optional: Install Neuron TensorFlow model server

If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package.

```
sudo yum install tensorflow-model-server-neuron=2.3.0.1.6.8.0 -y
```

Ubuntu AMI

Amazon Linux AMI

Ubuntu DLAMI

Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Configure Linux for Neuron repository updates

./etc/os-release

```
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com $(VERSION_CODENAME) main
EOF

```
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -
```

# Update OS packages

```
sudo apt-get update -y
```

(continues on next page)
Before installing or updating aws-neuron-dkms:
# Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
-
sudo systemctl stop neuron-rtd

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y
# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
-
option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
-
reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
-
option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install --yes python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install --upgrade pip
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.2.3.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
-
option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.2.2.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version , please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version , please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron TensorFlow
pip install tensorflow-neuron==2.2.3.1.6.8.0
# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.2.1.6.8.0 -y

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

(continues on next page)
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.2.3.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.2.2.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.2.3.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.2.2.1.6.8.0 -y

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

(continues on next page)
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
# If you are downgrading from newer version, please add --allow-downgrades option to sudo apt-get install
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.1.4.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add --allow-downgrades option to sudo apt-get install
sudo apt-get install tensorflow-model-server-neuron=2.1.4.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

(continues on next page)
# Install OS headers
```bash
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
```

# Install Neuron Driver
```bash
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
```
```bash
sudo yum install aws-neuron-dkms=2.1.5.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
```bash
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
```
```bash
sudo yum install aws-neuron-runtime=1.6.24.0 -y
```

```bash
export PATH=/opt/aws/neuron/bin:$PATH
```

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
```bash
sudo yum install -y python3 gcc-c++
```
```bash
python3 -m venv tensorflow_venv
```
```bash
source tensorflow_venv/bin/activate
```
```bash
pip install -U pip
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron TensorFlow
```bash
pip install tensorflow-neuron==2.1.4.1.6.8.0
```

# Optional: Install Neuron TensorFlow model server
```bash
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
```
```bash
sudo yum install tensorflow-model-server-neuron=2.1.4.1.6.8.0 -y
```

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y
```

(continues on next page)
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.1.4.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.1.4.1.6.8.0 -y

---

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Note: There is no DLAMI Conda environment for this framework version

(continues on next page)
# Framework will be installed/updated inside a Python environment

first line of code

# Update OS packages
sudo yum update -y

different line of code

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   →'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
#   using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

different line of code

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

different line of code

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
#   using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.1.4.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
#   using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.1.4.1.6.8.0 -y
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -
# Update OS packages
sudo apt-get update -y
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  # 'sudo systemctl stop neuron-rtd'
# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y
# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
  option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
  option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y
export PATH=/opt/aws/neuron/bin:$PATH
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install --y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install --U pip
(continues on next page)
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=1.15.0.1.6.8.0 -y

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
```

(continues on next page)
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package

```bash
sudo yum install aws-neuron-runtime=1.6.24.0 -y
```

```bash
export PATH=/opt/aws/neuron/bin:$PATH
```

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
```bash
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
```

```bash
pip install -U pip
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron TensorFlow
```bash
pip install tensorflow-neuron==1.15.5.1.6.8.0
```

# Optional: Install Neuron TensorFlow model server
```bash
sudo yum install tensorflow-model-server-neuron=1.15.0.1.6.8.0 -y
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Update OS packages
sudo apt-get update -y
```

```bash
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
```

```bash
# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y
```

# Install Neuron Driver
```bash
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y
```

```bash
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```
# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=1.15.0.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y
export PATH=/opt/aws/neuron/bin:$PATH
# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=1.15.0.1.6.8.0 -y

### Install Neuron TensorFlow (Neuron 1.15.0)

- **Develop on AWS ML accelerator instance**
- **Compile on compute instance**
- **Deploy on AWS ML accelerator instance**

### Develop on AWS ML accelerator instance

The simplest environment setup for model development installs all Neuron SDK components directly on an AWS ML accelerator instance: the Neuron framework extensions, compiler, runtime, and tools. This will allow you to compile, execute, and performance tune your model, all in the same instance. This is the recommended workflow when first starting to work with Neuron device or when optimizing a model.

**TensorFlow**

- TensorFlow 2.5.0
- TensorFlow 2.4.2
- TensorFlow 2.3.3
- TensorFlow 2.2.3
- TensorFlow 2.1.4
- TensorFlow 1.15.5

**Ubuntu AMI**

- Ubuntu AMI
- Amazon Linux AMI
- Ubuntu DLAMI
- Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Configure Linux for Neuron repository updates

```
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com $(VERSION_CODENAME) main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -
```

# Update OS packages

```
sudo apt-get update -y
```

# Before installing or updating aws-neuron-dkms:

```
# Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
'sudo systemctl stop neuron-rtd'
```

# Install OS headers

```
sudo apt-get install linux-headers-$uname -r -y
```

# Install Neuron Driver

```
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update

```
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```

# Install Neuron Runtime server

```
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 =y
```

# Install Neuron Tools

```
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.20.0 -y
```

# Install Neuron TensorFlow

```
pip install tensorboard-plugin-neuron=2.1.2.0
```

```
export PATH=/opt/aws/neuron/bin:$PATH
```

# Install Python venv and activate Python virtual environment to install

```
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install --user pip
```

(continues on next page)
# Install Jupyter notebook kernel
pip install jupyter
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
ip install jupyter
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.5.0.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '('--allow-downgrades')
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.5.1.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

(continues on next page)
# Warning: If Linux kernel is updated as a result of OS package update

Neuron driver (aws-neuron-dkms) should be re-installed after reboot.

Reboot

# Install Neuron Runtime server

If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package.

```
sudo yum install aws-neuron-runtime=1.6.19.0 -y
```

# Install Neuron Tools

If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package.

```
sudo yum install aws-neuron-tools=1.7.20.0 -y
```

# Install Neuron TensorBoard

```
pip install tensorboard-plugin-neuron==2.1.2.0
```

Export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.

```
sudo yum install -y python3 gcc-c++
```

```
sudo python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
```

```
pip install -U pip
```

# Install Jupyter notebook kernel

```
pip install ipykernel
```

```
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
```

```
pip install jupyter notebook
```

```
pip install environment_kernels
```

# Set Pip repository to point to the Neuron repository

```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron TensorFlow

```
pip install tensorflow-neuron[cc]==2.5.0.1.6.8.0
```

# Optional: Install Neuron TensorFlow model server

If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package.

```
sudo yum install tensorflow-model-server-neuron=2.5.1.1.6.8.0 -y
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Update OS packages
sudo apt-get update -y
```

(continues on next page)
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   → 'sudo systemctl stop neuron-rtd'
#############################################################################
→##################################
# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y
# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
→option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y
#############################################################################
→##################################
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# →reboot
#############################################################################
→####
# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
→option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y
# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
→option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.20.0 -y
# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH
# Activate TensorFlow
source activate
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.5.0.1.6.8.0
# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
→option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.5.1.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Update OS packages
```
sudo yum update -y
```

# Before installing or updating aws-neuron-dkms:
```
# Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
→sudo systemctl stop neuron-rtd
```

# Install OS headers
```
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y
```

# Install Neuron Driver
```
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update
```
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```

# Install Neuron Runtime server
```
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y
```

# Install Neuron Tools
```
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.20.0 -y
```

# Install Neuron TensorBoard
```
pip install tensorboard-plugin-neuron==2.1.2.0
```

```
export PATH=/opt/aws/neuron/bin:$PATH
```

# Activate TensorFlow
```
source activate
```

# Set Pip repository to point to the Neuron repository
```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron TensorFlow
```
pip install tensorflow-neuron[cc]==2.5.0.1.6.8.0
```

# Optional: Install Neuron TensorFlow model server
```
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.5.1.1.6.8.0 -y
```

Ubuntu AMI
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.20.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0

(continues on next page)
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv
g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.2.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.4.1.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
#-----------------------------------------------------------------------------------
(continues on next page)
# Install OS headers
`sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y`

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
`sudo yum install aws-neuron-dkms=2.0.450.0 -y`

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
`sudo yum install aws-neuron-runtime=1.6.19.0 -y`

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
`sudo yum install aws-neuron-tools=1.7.20.0 -y`

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
pip install tensorflow_venv

source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.2.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
`sudo yum install tensorflow-model-server-neuron=2.4.1.1.6.8.0 -y`
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.20.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.2.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
```
sudo apt-get install tensorflow-model-server-neuron=2.4.1.1.6.8.0 -y

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.20.0 -y

# Install Neuron TensorFlow
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
```
pip install tensorflow-neuron[cc]==2.4.2.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.4.1.1.6.8.0 -y

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Configure Linux for Neuron repository updates
/etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y
# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
```
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.20.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name
"Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.3.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.3.0.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   -> 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version , please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version , please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version , please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.20.0 -y

# Install Neuron TensorFlow
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
deployme_to_aws.py
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.3.0.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.20.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.3.1.6.8.0

# Optional: Install tensorflow-neuron[cc] model server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.3.0.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.20.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.3.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.3.0.1.6.8.0 -y

Ubuntu AMI

Amazon Linux AMI

Ubuntu DLAMI

Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
```
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.20.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.2.2.1.6.8.0 -y
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Configure Linux for Neuron repository updates
```bash
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
```
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
```bash
sudo yum update -y
```

# Before installing or updating aws-neuron-dkms:
- Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  ```bash
  sudo systemctl stop neuron-rtd
  ```

# Install OS headers
```bash
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
```

# Install Neuron Driver
- If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
```bash
sudo yum install aws-neuron-dkms=2.0.450.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update
- Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
- If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
```bash
sudo yum install aws-neuron-runtime=1.6.19.0 -y
```

# Install Neuron Tools
- If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
```bash
sudo yum install aws-neuron-tools=1.7.20.0 -y
```

# Install Neuron TensorBoard
```bash
pip install tensorboard-plugin-neuron==2.1.2.0
```

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
```bash
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
```

(continues on next page)
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name
"""Python (Neuron TensorFlow)"

pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.2.2.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$uname -r -y

# Install Neuron Driver
# If you are downgrading from newer version, please add 'allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.20.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++ python3-m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.2.2.1.6.8.0 -y

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  `sudo systemctl stop neuron-rtd`

# Install OS headers
`sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y`

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
`sudo yum install aws-neuron-dkms=2.0.450.0 -y`

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
`sudo yum install aws-neuron-runtime=1.6.19.0 -y`

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
`sudo yum install aws-neuron-tools=1.7.20.0 -y`

# Install Neuron TensorBoard
`pip install tensorboard-plugin-neuron==2.1.2.0`

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
`sudo yum install -y python3 gcc-c++ python3 -m venv tensorflow_venv`
`source tensorflow_venv/bin/activate`
`pip install -U pip`

# Install Jupyter notebook kernel
`pip install ipykernel`
`python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
`pip install jupyter notebook`
`pip install environment_kernels`

# Set Pip repository to point to the Neuron repository
`pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com`

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.2.2.1.6.8.0 -y

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
/etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
```
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
→ option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.20.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name
"Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
→ option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.1.4.1.6.8.0 -y

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-
→ AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   → 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version , please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version , please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version , please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.20.0 -y

# Install Neuron TensorFlow
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"

pip install jupyter notebook
pip install environment_kernels

(continues on next page)
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.1.4.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# Install OS headers
sudo apt-get install linux-headers-$\{(uname -r)\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.20.0 -y
# Install Neuron TensorBoard

```bash
pip install tensorboard-plugin-neuron==2.1.2.0
```

```bash
export PATH=/opt/aws/neuron/bin:$PATH
```

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
```bash
sudo apt-get install -y python3-venv g++
pip install -U pip
```

```bash
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
```

# Install Jupyter notebook kernel
```bash
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron TensorFlow
```bash
pip install tensorflow-neuron[cc]==2.1.4.1.6.8.0
```

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
```bash
sudo apt-get install tensorflow-model-server-neuron=2.1.4.1.6.8.0 -y
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
```bash
sudo yum update -y
```

# Before installing or updating aws-neuron-dkms:
# Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# sudo systemctl stop neuron-rtd
```

# Install OS headers
```bash
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
```

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
```bash
```
sudo yum install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.20.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.1.4.1.6.8.0 -y
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
/etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.20.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
(continues on next page)
```
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.8.0 neuron-cc==1.6.13.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=1.15.0.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# Install OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

(continues on next page)
# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
→ using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

# WARNING: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
→ reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
→ using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
→ using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.20.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name
"Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.8.0 neuron-cc==1.6.13.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
→ using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=1.15.0.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

```bash
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.20.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.8.0 neuron-cc==1.6.13.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=1.15.0.1.6.8.0 -y
```
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.20.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.2.0
export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.8.0 neuron-cc==1.6.13.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
```

(continues on next page)
Compile on compute instance

If model compilation occurs outside the model deployment environment, you can install only the Neuron framework extensions and the compiler on any compute instance. This setup is helpful when compiling large complex models that require large amount of memory or during a CICD process where models are compiled in a separate step, prior to deployment.

TensorFlow 2.5.0
TensorFlow 2.4.2
TensorFlow 2.3.3
TensorFlow 2.2.3
TensorFlow 2.1.4
TensorFlow 1.15.5
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Install Python venv and activate Python virtual environment to install 
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.5.0.1.6.8.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Install Python venv and activate Python virtual environment to install 
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
```

(continues on next page)
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.5.0.1.6.8.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.5.0.1.6.8.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.5.0.1.6.8.0

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

(continues on next page)
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.2.1.6.8.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.2.1.6.8.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.2.1.6.8.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.4.2.1.6.8.0

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI

Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.3.1.6.8.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.3.1.6.8.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.3.1.6.8.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

```bash
# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.3.3.1.6.8.0
```

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.1.6.8.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.1.6.8.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
copy the contents of the code block into a script file and source its contents.

```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.1.6.8.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.2.3.1.6.8.0
```

Ubuntu AMI

Amazon Linux AMI

Ubuntu DLAMI

Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
```
source tensorflow_venv/bin/activate
dpip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.1.6.8.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.1.6.8.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.1.6.8.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install ~U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]==2.1.4.1.6.8.0

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install ~U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.8.0 neuron-cc==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.8.0 neuron-cc==1.6.13.0

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.8.0 neuron-cc==1.6.13.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.8.0 neuron-cc==1.6.13.0
```

**Deploy on AWS ML accelerator instance**

During deployment it can be beneficial to reduce the number of components installed in the system. For use-cases where only inference is necessary (compilation is already complete), only the framework and runtime should be installed.

- TensorFlow 2.5.0
- TensorFlow 2.4.2
- TensorFlow 2.3.3
- TensorFlow 2.2.3
- TensorFlow 2.1.4
- TensorFlow 1.15.5
- Ubuntu AMI
- Amazon Linux AMI
- Ubuntu DLAMI
- Amazon Linux DLAMI
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
./etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y
# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y
export PATH=/opt/aws/neuron/bin:$PATH
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

(continues on next page)
# Install Neuron TensorFlow

```bash
pip install tensorflow-neuron==2.5.0.1.6.8.0
```

# Optional: Install Neuron TensorFlow model server

# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'

```bash
sudo apt-get install tensorflow-model-server-neuron=2.5.1.1.6.8.0 -y
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y
```

(continues on next page)
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron TensorFlow
pip install tensorflow-neuron==2.5.0.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package,
# using 'sudo yum remove' before installing the older package.
sudo yum install tensorflow-model-server-neuron=2.5.1.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# Install OS headers
sudo apt-get install linux Headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot.

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y
export PATH=/opt/aws/neuron/bin:$PATH
# Activate TensorFlow
source activate
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
# Install Neuron TensorFlow
pip install tensorflow-neuron==2.5.0.1.6.8.0
# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.5.1.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot
# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

(continues on next page)
# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.5.0.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
sudo yum install tensorflow-model-server-neuron=2.5.1.1.6.8.0 -y

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.4.2.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.4.1.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
# Install OS headers
`sudo yum install kernel-devel-$\{$uname -r\} kernel-headers-$\{$uname -r\\} -y`

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
`sudo yum install aws-neuron-dkms=2.0.450.0 -y`

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# `sudo reboot`

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
`sudo yum install aws-neuron-runtime=1.6.19.0 -y`

`export PATH=/opt/aws/neuron/bin:$PATH`

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
`sudo yum install -y python3 gcc-c++ python3 -m venv tensorflow_venv`
`source tensorflow_venv/bin/activate`
`pip install -U pip`

# Set Pip repository to point to the Neuron repository
`pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com`

# Install Neuron TensorFlow
`pip install tensorflow-neuron==2.4.2.1.6.8.0`

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
`sudo yum install tensorflow-model-server-neuron=2.4.1.1.6.8.0 -y`

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Update OS packages
`sudo apt-get update -y`

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd'`
# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron TensorFlow
pip install tensorflow-neuron==2.4.2.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.4.1.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

(continues on next page)
# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

###############################################
### Warning: If Linux kernel is updated as a result of OS package update
### Neuron driver (aws-neuron-dkms) should be re-installed after reboot
###############################################

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.4.2.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.4.1.1.6.8.0 -y

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Update OS packages

```shell
sudo apt-get update -y
```

# Before installing or updating aws-neuron-dkms:

- Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:

```shell
sudo systemctl stop neuron-rtd
```

# Install OS headers

```shell
sudo apt-get install linux-headers-$\{uname -r\} -y
```

# Install Neuron Driver

- If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'

```shell
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update

- Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server

- If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'

```shell
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y
```

```shell
export PATH=/opt/aws/neuron/bin:$PATH
```

# Install Python venv and activate Python virtual environment to install Neuron pip packages.

```shell
sudo apt-get install -y python3-venv g++ python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
```

# Set Pip repository to point to the Neuron repository

```shell
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron TensorFlow

```shell
pip install tensorflow-neuron==2.3.3.1.6.8.0
```

# Optional: Install Neuron TensorFlow model server

- If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'

```shell
sudo apt-get install tensorflow-model-server-neuron=2.3.0.1.6.8.0 -y
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Configure Linux for Neuron repository updates
```
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
```
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
```
sudo yum update -y
```

# Before installing or updating aws-neuron-dkms:
```
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
```

# Install OS headers
```
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
```

# Install Neuron Driver
```
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
```
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y
```

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
```
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip
```

# Set Pip repository to point to the Neuron repository
```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron TensorFlow

(continues on next page)
pip install tensorflow-neuron==2.3.3.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
   using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.3.0.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
   option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
   reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
   option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.3.3.1.6.8.0

# Optional: Install Neuron TensorFlow model server
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.3.3.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.3.0.1.6.8.0 -y
```

3.1. Installation Guide 729
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv

(continues on next page)
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.2.3.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.2.2.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
  name=Neuron YUM Repository
  baseurl=https://yum.repos.neuron.amazonaws.com
  enabled=1
  metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

#############################################################################
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd
#############################################################################

# Install OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

#############################################################################
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot
#############################################################################
# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
→ using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.2.3.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
→ using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.2.2.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
→ 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$ (uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
→ option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

(continues on next page)
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.2.3.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.2.2.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$({uname -r}) kernel-headers-$({uname -r}) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package

(continues on next page)
sudo yum install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.2.3.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.2.2.1.6.8.0 -y

Ubuntu AMI

Amazon Linux AMI

Ubuntu DLAMI

Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
# Configure Linux for Neuron repository updates
./etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
debs http://apt.repos.neuron.amazonaws.com ${{VERSION_CODENAME}} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages

```bash
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after_
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.1.4.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.1.4.1.6.8.0 -y
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Configure Linux for Neuron repository updates

```bash
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<<EOF
[neuron]
name=Neuron YUM Repository
dirbase=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB
```

# Update OS packages

```bash
sudo yum update -y
```

# Before installing or updating aws-neuron-dkms:

```bash
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
```

# Install OS headers

```bash
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y
```

# Install Neuron Driver

```bash
# If you are downgrading from newer version , please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server

```bash
# If you are downgrading from newer version , please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y
```

```bash
export PATH=/opt/aws/neuron/bin:$PATH
```

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.

```bash
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip
```

# Set Pip repository to point to the Neuron repository

```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron TensorFlow

(continues on next page)
pip install tensorflow-neuron==2.1.4.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.1.4.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

(continues on next page)
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==2.1.4.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
→option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=2.1.4.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
→'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
→using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
→reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
→using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron TensorFlow
pip install tensorflow-neuron==2.1.4.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=2.1.4.1.6.8.0 -y

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
./etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#    'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$ (uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

(continues on next page)
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=1.15.0.1.6.8.0 -y

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
sudo systemctl stop neuron-rtd
# Install OS headers
sudo yum install kernel-devel-$\texttt{(uname -r)}$ kernel-headers-$\texttt{(uname -r)}$ -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update, Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=1.15.0.1.6.8.0 -y

\textbf{Note:} For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
→'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
→option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
→reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
→option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
→option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=1.15.0.1.6.8.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.6.8.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=1.15.0.1.6.8.0 -y

---

Install Neuron TensorFlow (Neuron 1.14.2)

- Develop on AWS ML accelerator instance
- Compile on compute instance
- Deploy on AWS ML accelerator instance
Develop on AWS ML accelerator instance

The simplest environment setup for model development installs all Neuron SDK components directly on an AWS ML accelerator instance: the Neuron framework extensions, compiler, runtime, and tools. This will allow you to compile, execute, and performance tune your model, all in the same instance. This is the recommended workflow when first starting to work with Neuron device or when optimizing a model.

TensorFlow 1.15.5
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y
# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
```
sudo apt-get install aws-neuron-runtime=1.6.9.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.10.0 -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron==2.1.0.0

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python3 -m ipykernel install --user --name tensorflow_venv --display-name
"Python (Neuron TensorFlow)"

pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.5.1.0 neuron-cc==1.5.5.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=1.15.0.1.5.1.0 -y

Note:  For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-
AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.9.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.10.0 -y

# Install Neuron TensorFlow
pip install tensorboard-plugin-neuron==2.1.0.0

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

(continues on next page)
# Set Pip repository to point to the Neuron repository
`pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com`

# Install Neuron TensorFlow
`pip install tensorflow-neuron==1.15.5.1.5.1.0 neuron-cc==1.5.5.0`

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
`sudo yum install tensorflow-model-server-neuron=1.15.0.1.5.1.0 -y`

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Update OS packages
`sudo apt-get update -y`

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
`sudo apt-get install linux-headers-$\{uname -r\} -y`

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
`sudo apt-get install aws-neuron-dkms=2.0.386.0 -y`

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
`sudo apt-get install aws-neuron-runtime=1.6.9.0 -y`

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
`sudo apt-get install aws-neuron-tools=1.7.10.0 -y`

# Install Neuron TensorBoard
`sudo pip install tensorboard-plugin-neuron==2.1.0.0`

---

(continues on next page)
export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.5.1.0 neuron-cc==1.5.5.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=1.15.0.1.5.1.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.9.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
Compile on compute instance

If model compilation occurs outside the model deployment environment, you can install only the Neuron framework extensions and the compiler on any compute instance. This setup is helpful when compiling large complex models that require large amount of memory or during a CICD process where models are compiled in a separate step, prior to deployment.

TensorFlow 1.15.5
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.5.1.0 neuron-cc==1.5.5.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=1.15.0.1.5.1.0 -y
```
**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.5.1.0 neuron-cc==1.5.5.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.5.1.0 neuron-cc==1.5.5.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.5.1.0 neuron-cc==1.5.5.0
```
Deploy on AWS ML accelerator instance

During deployment it can be beneficial to reduce the number of components installed in the system. For use-cases where only inference is necessary (compilation is already complete), only the framework and runtime should be installed.

TensorFlow 1.15.5

Ubuntu AMI

Amazon Linux AMI

Ubuntu DLAMI

Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'

(continues on next page)
### AWS Neuron

(continued from previous page)

```bash
sudo apt-get install aws-neuron-runtime=1.6.9.0 -y
export PATH=/opt/aws/neuron/bin:$PATH
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.5.1.0
# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=1.15.0.1.5.1.0 -y
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB
# Update OS packages
sudo yum update -y
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd'
#-# Install OS headers
sudo yum install kernel-devel-$NKNAME -y
# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
```

(continues on next page)
sudo yum install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.9.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.5.1.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install tensorflow-model-server-neuron=1.15.0.1.5.1.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.9.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron==1.15.5.1.5.1.0

# Optional: Install Neuron TensorFlow model server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install tensorflow-model-server-neuron=1.15.0.1.5.1.0 -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
sudo systemctl stop neuron-rtd

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.386.0 -y
### 3.2 User Guide

#### 3.2.1 TensorFlow Tutorials

**Before running a tutorial**

You will run the tutorials on an inf1.6xlarge instance running Deep Learning AMI (DLAMI) to enable both compilation and deployment (inference) on the same instance. In a production environment we encourage you to try different instance sizes to optimize to your specific deployment needs.

Follow instructions at [TensorFlow Tutorial Setup](#) before running a TensorFlow tutorial on Inferentia. We recommend new users start with the ResNet-50 tutorial.
TensorFlow Tutorial Setup

1. Launch an Inf1.6xlarge Instance:
   - Please follow the instructions at launch an Amazon EC2 Instance to Launch an Inf1 instance, when choosing the instance type at the EC2 console. Please make sure to select the correct instance type. To get more information about Inf1 instances sizes and pricing see Inf1 web page.
   - When choosing an Amazon Machine Image (AMI) make sure to select Deep Learning AMI with Conda Options. Please note that Neuron Conda environments are supported only in Ubuntu 18 DLAMI and Amazon Linux2 DLAMI, Neuron Conda environments are not supported in Amazon Linux DLAMI.
   - After launching the instance, follow the instructions in Connect to your instance to connect to the instance

   Note: You can also launch the instance from AWS CLI, please see AWS CLI commands to launch inf1 instances.

2. Set up a development environment:

   Important:

   For successful installation or update to Neuron 1.16.0 and newer from previous releases:

   - Stop Neuron Runtime 1.x daemon (neuron-rtd) by running: sudo systemctl stop neuron-rtd
   - Uninstall neuron-rtd by running: sudo apt remove aws-neuron-runtime or sudo yum remove aws-neuron-runtime
   - Install or upgrade to latest Neuron driver (aws-neuron-dkms) by following the “Setup Guide” instructions.
   - Visit Introducing Neuron Runtime 2.x (libnrt.so) for more information.

   • Enable or install TensorFlow-Neuron:
     TensorFlow 2.5.1
     TensorFlow 1.15.5
     Ubuntu DLAMI
     Amazon Linux DLAMI

   Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by
  # calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from
# previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you
  # MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package
# update
# Neuron driver (aws-neuron-dkms) should be re-installed
# after reboot

# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Install Neuron TensorFlow
pip install tensorboard-plugin-neuron

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment

# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --
  --display-name "Python (Neuron TensorFlow)"

(continues on next page)
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]

# Optional: Install Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron -y

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
# Install Neuron Tools
sudo yum install aws-neuron-tools -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]

# Optional: Install Neuron TensorFlow model server
sudo yum install tensorflow-model-server-neuron -y

---

Ubuntu DLAMI

Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation

---

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

(continues on next page)
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed

# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
# Install OS headers
```
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
```

# Install Neuron Driver
```
sudo yum install aws-neuron-dkms -y
```

# Warning: If Linux kernel is updated as a result of OS package update
Neuron driver (aws-neuron-dkms) should be re-installed after reboot.

# Activate TensorFlow
```
source activate aws_neuron_tensorflow_p36
```

3. Run tutorial in Jupyter notebook:
   - Follow instruction at Setup Jupyter notebook to:
     1. Start the Jupyter Notebook on the instance
     2. Run the Jupyter Notebook from your local browser
   - Connect to the instance from the terminal, clone the Neuron Github repository to the Inf1 instance and then change the working directory to the tutorial directory:
```
git clone https://github.com/aws/aws-neuron-sdk.git
cd aws-neuron-sdk/src/examples/tensorflow
```
   - Locate the tutorial notebook file (.ipynb file) under aws-neuron-sdk/src/examples/tensorflow
   - From your local browser, open the tutorial notebook from the menu and follow the instructions.

**Computer Vision**

- Tensorflow 1.x - OpenPose tutorial [html] [notebook]
- Tensorflow 1.x - ResNet-50 tutorial [html] [notebook]
- Tensorflow 1.x - YOLOv4 tutorial [html] [notebook]
- Tensorflow 1.x - YOLOv3 tutorial [html] [notebook]
- Tensorflow 1.x - SSD300 tutorial [html]
- Tensorflow 1.x - Keras ResNet-50 optimization tutorial [html] [notebook]
Running OpenPose on Inferentia

Note: this tutorial runs on tensorflow-neuron 1.x only

Introduction:

In this tutorial we will compile and deploy Openpose model for Inferentia. This jupyter notebook should run on an inf1.6xlarge instance for compilation and inference. The inference part of this tutorial requires inf1.6xlarge and not the compilation itself. For simplicity we will run this tutorial on a single instance but in real life scenario the compilation can be done on a compute c5.4xlarge instance and the deployment on the inf1 instance family.

In this tutorial we provide two main sections: 1. Compile the OpenPose model on inf1x6large. 2. Infer the same compiled model on inf1x6large.

Before running the following verify this Jupyter notebook is running “conda_aws_neuron_pytorch_p36” kernel. You can select the Kernel from the “Kernel -> Change Kernel” option on the top of this Jupyter notebook page.

Acknowledgement:

Many thanks to https://github.com/ildoonet for providing pretrained model as well as the image prepro-cessing/pose estimating infrastructure.

Download tensorflow pose net frozen graph.


Compile

Compile the pose net frozen graph into AWS Neuron compatible form. Network input image resolution is adjustable with argument –net_resolution (e. g., –net_resolution=656x368). The compiled model can accept arbitrary batch size input at runtime.

```python
    Usage: python convert_graph_opt.py /path/to/graph_opt.pb /path/to/graph_opt_neuron.pb

    #import argparse
    import numpy as np
    import tensorflow as tf
    from tensorflow.core.framework.tensor_shape_pb2 import TensorShapeProto
    import tensorflow.neuron as tfn

    def compile():
        #parser = argparse.ArgumentParser()
```
```python
# parser.add_argument('input_pb_path', help='Input serialized GraphDef
→protobuf')
# parser.add_argument('output_pb_path', help='Ouput serialized GraphDef
˓→protobuf')
# parser.add_argument('--net_resolution', default='656x368', help=
˓→'Network resolution in WxH format, e. g., --net_resolution=656x368')
# parser.add_argument('--debug_verify', action='store_true')
# args = parser.parse_args()

input_pb_path = './graph_opt.pb'
net_resolution = '656x368'
output_pb_path = './graph_opt_neuron_' + net_resolution + '.pb'
default_value = 'store_true'
dim_w, dim_h = net_resolution.split('x')
dim_w = int(dim_w)
dim_h = int(dim_h)

graph_def = tf.GraphDef()
with open(input_pb_path, 'rb') as f:
    graph_def.ParseFromString(f.read())
if debug_verify:
    np.random.seed(0)
    feed_dict = {'image:0': np.random.rand(1, dim_h, dim_w, 3)}
    output_name = 'Openpose/concat_stage7:0'
    with tf.Session(graph=tf.Graph()) as sess:
        tf.import_graph_def(graph_def, name='')
        result_reference = sess.run(output_name, feed_dict)

preprocessing_ops = {'preprocess_divide', 'preprocess_divide/y',
˓→'preprocess_subtract', 'preprocess_subtract/y'}
graph_def = nhwc_to_nchw(graph_def, preprocessing_ops)
graph_def = inline_float32_to_float16(graph_def, preprocessing_ops)
with tf.Session(graph=tf.Graph()) as sess:
    tf.import_graph_def(graph_def, name='')
    no_fuse_ops = preprocessing_ops.union({'Openpose/concat_stage7'})
    infer_graph = tfn.graph_util.inference_graph_from_session(
        sess, shape_feed_dict={'image:0': [1, dim_h, dim_w, 3]}, output_tensors=['Openpose/concat_stage7:0'],
        no_fuse_ops=no_fuse_ops, dynamic_batch_size=True,
    )
with open(output_pb_path, 'wb') as f:
    f.write(infer_graph.as_graph_def().SerializeToString())

if debug_verify:
    with tf.Session(graph=infer_graph) as sess:
        result_compiled = sess.run(output_name, feed_dict)
        np.testing.assert_allclose(result_compiled, result_reference,
            rtol=1e-2, atol=1e-3)

def inline_float32_to_float16(graph_def, preprocessing_ops):
    float32_enum = tf.float32.as_datatype_enum
    float16_enum = tf.float16.as_datatype_enum
    graph = tf.Graph()
    with graph.as_default():
        tf.import_graph_def(graph_def, name='')
```

(continues on next page)
graph_def = graph.as_graph_def()

for node in graph_def.node:
    if node.name in preprocessing_ops or node.op == 'Placeholder':
        cast_input_node_name = node.name
        continue
    if node.op == 'Const':
        if node.attr['dtype'].type == float32_enum:
            node.attr['dtype'].type = float16_enum
            tensor_def = node.attr['value'].tensor
            if tensor_def.dtype == float16_enum:
                const_np = np.frombuffer(tensor_def.tensor_content,
                                          dtype=np.float32).astype(np.float16)
                tensor_def.tensor_content = const_np.tobytes()
            elif len(tensor_def.float_val):
                const_np = np.array(tensor_def.float_val).astype(np.float16)
                tensor_def.float_val[:] = []
                tensor_def.half_val[:] = list(const_np)
            else:
                raise NotImplementedError
        elif 'T' in node.attr and node.attr['T'].type == float32_enum:
            node.attr['T'].type = float16_enum

    if node.name == cast_input_node_name:
        node.name = '{}_PreCastFloat32ToFlot16'.format(node.name)
        input_node = node
        break

cast_input_node = _gen_cast_node_def(cast_input_node_name, tf.float16, input_node)
output_node = graph_def.node[-1]
cast_output_node_name = output_node.name
output_node.name = '{}_PreCastFloat16ToFlot32'.format(output_node.name)
cast_output_node = _gen_cast_node_def(cast_output_node_name, tf.float32, output_node)
preprocessing_ops.add(input_node.name)

new_graph_def = tf.GraphDef()
new_graph_def.node.extend(graph_def.node)
new_graph_def.node.append(cast_input_node)
new_graph_def.node.append(cast_output_node)

with graph.as_default():
    tf.import_graph_def(new_graph_def, name='')
return graph.as_graph_def()


def nhwc_to_nchw(graph_def, preprocessing_ops):
    graph = tf.Graph()
    with graph.as_default():
        tf.import_graph_def(graph_def, name='')
        graph_def = graph.as_graph_def()

    node_name_to_node = {node.name: node for node in graph_def.node}
    for node in graph_def.node:
        if node.name in preprocessing_ops or node.op == 'Placeholder':
            transpose_input_node_name = node.name

(continues on next page)
continue
if node.op == 'Conv2D':
    node.attr['data_format'].s = b'NCHW'
    strides = node.attr['strides'].list.i
    strides[:] = [strides[0], strides[3], strides[1], strides[2]]
elif node.op == 'BiasAdd':
    if node.name != 'probs/BiasAdd':
        node.attr['data_format'].s = b'NCHW'
elif node.op == 'MaxPool':
    node.attr['data_format'].s = b'NCHW'
    ksize = node.attr['ksize'].list.i
    ksize[:] = [ksize[0], ksize[3], ksize[1], ksize[2]]
    strides = node.attr['strides'].list.i
    strides[:] = [strides[0], strides[3], strides[1], strides[2]]
elif node.op in {'Concat', 'ConcatV2'}:
    node_axes = node_name_to_node[node.input[-1]]
    node_axes.attr['value'].tensor.int_val[:] = [1]
for node in graph_def.node:
    if node.name == transpose_input_node_name:
        node.name = '{}_PreTransposeNHWC2NCHW'.format(node.name)
        input_node = node
        break
transpose_input_node, transpose_input_perm_node = _gen_transpose_def(transpose_input_node_name, [0, 3, 1, 2], input_node)

output_node = graph_def.node[-1]
transpose_output_node_name = output_node.name
output_node.name = '{}_PreTransposeNCHW2NHWC'.format(output_node.name)
transpose_output_node, transpose_output_perm_node = _gen_transpose_def(transpose_output_node_name, [0, 2, 3, 1], output_node)

preprocessing_ops.add(input_node.name)
preprocessing_ops.add(transpose_input_perm_node.name)
new_graph_def = tf.GraphDef()
new_graph_def.node.extend(graph_def.node)
new_graph_def.node.append(transpose_input_perm_node)
new_graph_def.node.append(transpose_input_node)
new_graph_def.node.append(transpose_output_perm_node)
new_graph_def.node.append(transpose_output_node)
graph = tf.Graph()
with graph.as_default():
    tf.import_graph_def(new_graph_def, name='')
return graph.as_graph_def()

def _gen_cast_node_def(name, target_dtype, input_node):
    cast_node = tf.NodeDef(name=name, op='Cast')
    cast_node.input.append(input_node.name)
    cast_node.attr['DstT'].type = target_dtype.as_datatype_enum
    cast_node.attr['SrcT'].type = input_node.attr['T'].type
    cast_node.attr['Truncate'].b = False
    return cast_node

def _gen_transpose_def(name, perm, input_node):
    perm_node = tf.NodeDef(name='//perm'.format(name), op='Const')
    perm_node.attr['dtype'].type = tf.int32.as_datatype_enum
    perm_node.attr['perm'].list.i[:] = perm
    return perm_node

3.2. User Guide 765
```python
tensor_def = perm_node.attr['value'].tensor
tensor_def.dtype = tf.int32.as_datatype_enum
tensor_def.tensor_shape.dim.append(TensorShapeProto.Dim(size=4))
tensor_def.tensor_content = np.array(perm, dtype=np.int32).tobytes()
transpose_node = tf.NodeDef(name=name, op='Transpose')
transpose_node.input.append(input_node.name)
transpose_node.input.append(perm_node.name)
transpose_node.attr['T'].type = input_node.attr['T'].type
transpose_node.attr['Tperm'].type = tf.int32.as_datatype_enum
return transpose_node, perm_node
```

```
[ ]: compile()

# Sample output will look like below:
# WARNING:tensorflow:From <ipython-input-3-27d3844cd753>:47: inference_graph_
# ˓→from_session (from tensorflow_neuron.python.graph_util) is deprecated and
# ˓→will be removed in a future version.
# Instructions for updating:
# Please refer to AWS documentation on Neuron integrated TensorFlow 2.0.
# INFO:tensorflow:Froze 0 variables.
# INFO:tensorflow:Converted 0 variables to const ops.
# INFO:tensorflow:fusing subgraph {subgraph neuron_op_ed41d2deb8c54255 with
# ˓→input tensors ["<tf.Tensor 'preprocess_subtract0/_0:0' shape=(1, 3, 368, 656) dtype=float16>"], output tensors ["<tf.Tensor 'Openpose/concat_stage7_PreCastFloat16ToFloat32:0' shape=(1, 46, 82, 57) dtype=float16>"]} with
# neuron-cc
# INFO:tensorflow:Number of operations in TensorFlow session: 474
# INFO:tensorflow:Number of operations after tf.neuron optimizations: 474
# INFO:tensorflow:Number of operations placed on Neuron runtime: 465
```

## Deploy

Using same instance to deploy the model. In case of different deployment instance, launch a deployment inf1 instance and copy the AWS Neuron optimized tensorflow frozen graph `graph_opt_neuron_656x368.pb` to the deployment inf1 instance. The smallest instance type inf1.xlarge is sufficient for this demo.

Your `graph_opt_neuron_656x368.pb` can now be plugged into https://github.com/ildoonet seemlessly if you have tensorflow-neuron installed. When it is used at runtime, please ensure that the image resolution is the same as compile-time image resolution, i.e., 656x368.

Measure performance on the compiled frozen graph using dummy inputs.
from tensorflow.python.client import session
from tensorflow.python.platform import tf_logging as logging

class measure_performance(ContextDecorator):
    """Convenient tool for performance measurements. Can be apply on tensorflow session.run, tf-serving unary gRPC calls, or a given custom function.
    Usage:
    To generate performance report for the entire Python or gRPC-client process, insert the following function call before running inferences:
    `tfn.measure_performance()`
    Then latency/throughput report will be generated when the process terminates.
    Alternatively, it is possible to use `tfn.measure_performance` programmatically as a context manager. Performance measurement will be done for all inferences happening under this context. Report will be displayed as INFO level log when exiting the context. It is also possible to obtain a JSON format report in Python.
    For example:
    ```
    with tfn.measure_performance() as perf:
        ... (run some inferences) ...
        report_json = perf.report()
        report_full_json = perf.report(verbosity=1)
        ...
    """
    def __init__(self, func=None, window_size=1):
        self.perf_tracker = PerformanceTracker(window_size)
        atexit.register(self.perf_tracker.report)
        self._original_run = session.Session.run
        self._original_grpc_call = None
        if callable(func):
            self.perf_tracker.register_func(self._track_performance(func))
        else:
            session.Session.run = self._track_performance(session.Session.run)
        try:
            import grpc
            from tensorflow_serving.apis import prediction_service_pb2_grpc
            dummy_stub = prediction_service_pb2_grpc.PredictionServiceStub(grpc.insecure_channel(''))
            self._grpc_callable_type = type(dummy_stub.Predict)
            self._original_grpc_call = self._grpc_callable_type.__call__
        except ImportError:
            pass
        if callable(self._original_grpc_call):
            self._grpc_callable_type.__call__ = self._track_performance(grpc._channel._UnaryUnaryMultiCallable.__call__)

(continued from previous page)
def __enter__(self):
    return self.perf_tracker

def __exit__(self, *exc):
    atexit.unregister(self.perf_tracker.report)
    self.perf_tracker.report()
    session.Session.run = self._original_run
    if self._original_grpc_call is not None:
        self._grpc_callable_type.__call__ = self._original_grpc_call
        return False

def _track_performance(self, func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        start = time.time()
        result = func(*args, **kwargs)
        end = time.time()
        self.perf_tracker.add_timestamps(start, end)
        return result
    return wrapper

class PerformanceTracker(ContextDecorator):
    description = 
    "Latency unit: second. Throughput unit: number of batched inferences per second. "
    "Reported throughput is a lower bound of the actual throughput as inferences spanning across window boundaries are not counted towards any of the windows. "
    "'Quiet' periods (i.e., window buckets where the inference function is not called) are not counted towards the reported average throughput."

    def __init__(self, window_size):
        self.window_size = window_size
        self.timestamps_list = []
        self._func = None

    def __call__(self, *args, **kwargs):
        return self._func(*args, **kwargs)

    def register_func(self, func):
        self._func = func

    def add_timestamps(self, start, end):
        self.timestamps_list.append([start, end])

    def report(self, verbosity=0):
        if self.timestamps_list:
            latency_list = [end - start for start, end in self.timestamps_list]
            latency_json = {
                'p50': percentile(latency_list, 50),
                'p90': percentile(latency_list, 90),
            }

            return latency_json
'p99': percentile(latency_list, 99),
'p100': percentile(latency_list, 100),
}

bucketed_timestamps = [self._get_bucket(start, end) for start,
end in self.timestamps_list]

counted_buckets = Counter(item for item in bucketed_timestamps
if item is not None)

bucket_throughputs = [(key, value / self.window_size) for key,
value in sorted(counted_buckets.items())]

busy_throughputs = list(OrderedDict((key, value) for key, value,
in bucket_throughputs).values())

throughput_json = {
'peak': max(busy_throughputs),
'median': percentile(busy_throughputs, 50),
'average': sum(busy_throughputs) / len(busy_throughputs),
}

if verbosity > 0:
    throughput_json['trend'] = busy_throughputs

report_json = {
    'pid': os.getpid(),
    'throughput': throughput_json,
    'latency': latency_json,
    'description': PerformanceTracker.description,
}

with _logging_show_info():
    logging.info('performance report:

{}}'.format(json.
 dumps(report_json, indent=4)))

return report_json

def _get_bucket(self, start, end):
    bucketed_start = math.floor(start / self.window_size) * self.window_ size
    bucketed_end = math.ceil(end / self.window_size) * self.window_size

    if bucketed_end - bucketed_start == self.window_size:
        return bucketed_start
    else:
        return None

def percentile(number_list, percent):
    pos_float = len(number_list) * percent / 100
    max_pos = len(number_list) - 1
    pos_floor = min(math.floor(pos_float), max_pos)
    pos ceil = min(math.ceil(pos_float), max_pos)

    number_list = sorted(number_list)

    if pos_float - pos_floor > 0.5:
        return number_list[pos ceil]
    else:
        return number_list[pos_floor]

@contextmanager

def _logging_show_info():
    try:
        verbosity = logging.get_verbosity()
        logging.set_verbosity(logging.INFO)
        yield
    finally:
        logging.set_verbosity(verbosity)
Below are the inputs for compiled frozen graph

```
pb_path is a /path/graph_opt_neuron_656x368.pb
tf_neuron = 8 ( Number of threads that work on each tensorflow session )
batch_size = 1 ( batch_size )
net_resolution ,default=656x368
num_inferences = 200
```

```python
import os
from concurrent import futures
import numpy as np
import tensorflow as tf
import tensorflow.neuron as tfn
import warnings

warnings.warn("NEURONCORE_GROUP_SIZES is being deprecated, if your
application is using NEURONCORE_GROUP_SIZES please \nsee https://awsdocs-neuron.readthedocs-hosted.com/en/latest/release-notes/
deprecation.html#announcing-end-of-support-for-neuroncore-group-sizes \nfor more details.", DeprecationWarning)

def run_with_dummy(sess, dummy_feed_dict, num_inferences):
    for _ in range(num_inferences):
        sess.run('Openpose/concat_stage7:0', dummy_feed_dict)

def main():
    os.environ['NEURONCORE_GROUP_SIZES'] = '16x1'
pb_path = './graph_opt_neuron_656x368.pb'
num_thread = 8
batch_size = 1
net_resolution = '656x368'
num_inferences = 200
dim_w, dim_h = net_resolution.split('x')
dim_w = int(dim_w)
dim_h = int(dim_h)
graph_def = tf.GraphDef()
with open(pb_path, 'rb') as f:
    graph_def.ParseFromString(f.read())

    with tfn.measure_performance() as perf:
        with tf.Session(graph=tf.Graph()) as sess:
            tf.import_graph_def(graph_def, name='')
            input_name = 'image:0'
            input_shape = sess.graph.get_tensor_by_name(input_name).shape.as_
list()
            input_shape[0] = batch_size
            input_shape[1] = dim_h
            input_shape[2] = dim_w
            dummy_feed_dict = {input_name: np.zeros(input_shape).astype(np.
float32)}

        with futures.ThreadPoolExecutor(max_workers=num_thread) as 
executor:
            fut_list = [executor.submit(run_with_dummy, sess, dummy_feed_ 
dict, num_inferences) for _ in range(num_thread)]
            res_list = [fut.result() for fut in fut_list]

(continues on next page)```
Running ResNet50 on Inferentia

**Note: this tutorial runs on tensorflow-neuron 1.x only**

**Introduction:**

In this tutorial we will compile and deploy ResNet50 model for Inferentia. In this tutorial we provide two main sections: 1. Compile the ResNet50 model. 2. Infer the same compiled model.

Before running the following verify this Jupyter notebook is running “conda_aws_neuron_tensorflow_p36” kernel. You can select the Kernel from the “Kernel -> Change Kernel” option on the top of this Jupyter notebook page.

Instructions of how to setup Neuron Tensorflow environment and run the tutorial as a Jupyter notebook are available in the Tensorflow Quick Setup

**Compile for Neuron**

A trained model must be compiled to Inferentia target before it can be deployed on Inferentia instances. In this step we compile the Keras ResNet50 model and export it as a SavedModel which is an interchange format for TensorFlow models. At the end of compilation, the compiled SavedModel is saved in resnet50_neuron local directory:

```python
import os
import time
import shutil
import tensorflow as tf
```
import tensorflow.neuron as tfn
import tensorflow.compat.v1.keras as keras
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input

# Create a workspace
WORKSPACE = './ws_resnet50'
os.makedirs(WORKSPACE, exist_ok=True)

# Prepare export directory (old one removed)
model_dir = os.path.join(WORKSPACE, 'resnet50')
compiled_model_dir = os.path.join(WORKSPACE, 'resnet50_neuron')
shutil.rmtree(model_dir, ignore_errors=True)
shutil.rmtree(compiled_model_dir, ignore_errors=True)

# Instantiate Keras ResNet50 model
keras.backend.set_learning_phase(0)
keras.backend.set_image_data_format('channels_last')
model = ResNet50(weights='imagenet')

# Export SavedModel
tf.saved_model.simple_save(
    session = keras.backend.get_session(),
    export_dir = model_dir,
    inputs = {'input': model.inputs[0]},
    outputs = {'output': model.outputs[0]})

# Compile using Neuron
tfn.saved_model.compile(model_dir, compiled_model_dir)

[[ ]]: !ls

**Deploy on Inferentia**

Using same instance to deploy the model. In case of different deployment instance, launch a deployment inf1 instance and copy compiled model to the deployment inf1 instance.

Download the example image, and install pillow module for inference on deployment instance

!pip install pillow  # Necessary for loading images
After downloading the example image, run the inference.

```python
import os
import time
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications import resnet50
tf.keras.backend.set_image_data_format('channels_last')

# Create input from image
img_sgl = image.load_img('kitten_small.jpg', target_size=(224, 224))
img_arr = image.img_to_array(img_sgl)
img_arr2 = np.expand_dims(img_arr, axis=0)
img_arr3 = resnet50.preprocess_input(img_arr2)

# Load model
COMPILED_MODEL_DIR = './ws_resnet50/resnet50_neuron/
predictor_inferentia = tf.contrib.predictor.from_saved_model(COMPILED_MODEL_DIR)

# Run inference
model_feed_dict={'input': img_arr3}
infa_rslts = predictor_inferentia(model_feed_dict);

# Display results
print(resnet50.decode_predictions(infa_rslts['output'], top=5)[0])
```

Working with YOLO v4 using AWS Neuron SDK

The /src/examples/tensorflow/yolo_v4_demo/evaluate.ipynb notebook contains an example on how to take an open source YOLO v4 models, and run it on AWS Inferentia.

Optimizing image pre-processing and post-processing for object detection models

End-to-end object detection pipelines usually contain image pre-post-processing operators that cannot run efficiently on Inferentia. DecodeJPEG and NonMaxSuppression are typical examples. In practice, we may simply place these operators on CPU using the AWS Neuron machine learning framework integration. However, Inferentia is such a high performance machine learning accelerator that, once the model successfully compiles and runs, these simple pre-post-processing operators can become the new performance bottleneck! In this tutorial, we explain some commonly used tensorflow techniques for optimizing the performance of these pre-post-processing operators so that we can fully unleash the potential of Inferentia.

1. Write JPEG decoding and image shifting/scaling as tensorflow operators.

In `yolo_v4_coco_saved_model.py`, you may find the following code snippet.
import tensorflow as tf
...

def YOLOv4(...
    ...
        x, image_shape = layers.Lambda(lambda t: preprocessor(t, input_→shape))(inputs)
        
            # cspdarknet53
            x = conv2d_unit(x, i32, 3, strides=1, padding='same')
    ...

def decode_jpeg_resize(input_tensor, image_size):
    tensor = tf.image.decode_png(input_tensor, channels=3)
    shape = tf.shape(tensor)
    tensor = tf.cast(tensor, tf.float32)
    tensor = tf.image.resize(tensor, image_size)
    tensor /= 255.0
    return tf.cast(tensor, tf.float16), shape

def preprocessor(input_tensor, image_size):
    with tf.name_scope('Preprocessor'):
        tensor = tf.map_fn(
            partial(decode_jpeg_resize, image_size=image_size), input_tensor,
            dtype=(tf.float16, tf.int32), back_prop=False, parallel_→iterations=16)
    return tensor

Comparing with the implementation in the original repo, our difference is the use of tf.image.
decode_png and tf.image.resize, along with a small number of scaling/casting operators. After
this modification, the generated tensorflow SavedModel now takes JPEG image raw bytes as input, in-
stead of a float32 array representing the image. When the image resolution is 608x608, this technique
effectively reduces the input image size from 4.4 MB to the size of a typical JPEG image, which can be as
little as hundreds of KB. When the tensorflow SavedModel is deployed through tensorflow/serving, this
technique can very effectively reduce the gRPC transfer overhead of input images.

2. Replace non-max suppression (NMS) operations by tf.image.
combined_non_max_suppression.

Another difference of our implementation is the treatment of non-max suppression, a commonly used
operation for removing redundant bounding boxes that overlap with other boxes. In an object detec-
tion scenario represented by the COCO dataset where the number of output classes is large, the hand-
fused `tf.image.combined_non_max_suppression <https://www.tensorflow.org/versions/r1.15/api_docs/python/tf/image/combined_non_max_suppression>`__ operator can parallelize multi-class
NMS on CPU in a very efficient manner. With proper use of this operator, the bounding box post-
processing step has a less chance of becoming the performance bottleneck in the end-to-end object detec-
tion pipeline.

The following sample code (from yolo_v4_coco_saved_model.py) demonstrates our method of
writing the bounding box post-processing step using efficient tensorflow operations.

... def filter_boxes(outputs):
    boxes_l, boxes_m, boxes_s, box_scores_l, box_scores_m, box_scores_s, image_shape = outputs
    boxes_l, box_scores_l = filter_boxes_one_size(boxes_l, box_scores_l)
    boxes_m, box_scores_m = filter_boxes_one_size(boxes_m, box_scores_m) (continues on next page)
boxes_s, box_scores_s = filter_boxes_one_size(boxes_s, box_scores_s)
boxes = tf.concat([boxes_l, boxes_m, boxes_s], axis=0)
box_scores = tf.concat([box_scores_l, box_scores_m, box_scores_s], axis=0)
image_shape_wh = image_shape[1::-1]
image_shape_whwh = tf.concat([image_shape_wh, image_shape_wh], axis=-1)
image_shape_whwh = tf.cast(image_shape_whwh, tf.float32)
boxes *= image_shape_whwh
boxes = tf.expand_dims(boxes, 0)
box_scores = tf.expand_dims(box_scores, 0)
boxes = tf.expand_dims(boxes, 2)
nms_boxes, nms_scores, nms_classes, valid_detections = tf.image.combined_non_max_suppression(
    boxes,
    box_scores,
    max_output_size_per_class=nms_top_k,
    max_total_size=nms_top_k,
    iou_threshold=nms_thresh,
    score_threshold=conf_thresh,
    pad_per_class=False,
    clip_boxes=False,
    name='CombinedNonMaxSuppression',
)
return nms_boxes[0], nms_scores[0], nms_classes[0]

def filter_boxes_one_size(boxes, box_scores):
    box_class_scores = tf.reduce_max(box_scores, axis=-1)
    keep = box_class_scores > conf_thresh
    boxes = boxes[keep]
    box_scores = box_scores[keep]
    return boxes, box_scores

def batch_yolo_out(outputs):
    with tf.name_scope('yolo_out'):
        b_output_lr, b_output_mr, b_output_sr, b_image_shape = outputs
        with tf.name_scope('process_feats'):
            b_boxes_l, b_box_scores_l = batch_process_feats(b_output_lr, anchors, masks[0])
        with tf.name_scope('process_feats'):
            b_boxes_m, b_box_scores_m = batch_process_feats(b_output_mr, anchors, masks[1])
        with tf.name_scope('process_feats'):
            b_boxes_s, b_box_scores_s = batch_process_feats(b_output_sr, anchors, masks[2])
        with tf.name_scope('filter_boxes'):
            b_nms_boxes, b_nms_scores, b_nms_classes = tf.map_fn(
                filter_boxes, [b_boxes_l, b_boxes_m, b_boxes_s, b_box_scores_l, b_box_scores_m, b_box_scores_s, b_image_shape],
                dtype=(tf.float32, tf.float32, tf.float32), back_prop=False, parallel_iterations=16)
    return b_nms_boxes, b_nms_scores, b_nms_classes

boxes_scores_classes = layers.Lambda(batch_yolo_out)([output_lr, output_mr, output_sr, image_shape])
For other advanced data input/output pipeline optimization techniques, please refer to https://www.tensorflow.org/guide/data#preprocessing_data.

**Evaluate YOLO v3 on Inferentia**

**Note: this tutorial runs on tensorflow-neuron 1.x only**

**Introduction**

This tutorial walks through compiling and evaluating YOLO v3 model on Inferentia using the AWS Neuron SDK.

In this tutorial we provide two main sections:

1. Download Dataset and Generate Pretrained SavedModel
2. Compile the YOLO v3 model.
3. Deploy the same compiled model.

Before running the following verify this Jupyter notebook is running “conda_aws_neuron_tensorflow_p36” kernel. You can select the Kernel from the “Kernel -> Change Kernel” option on the top of this Jupyter notebook page.

Instructions of how to setup Neuron Tensorflow environment and run the tutorial as a Jupyter notebook are available in the Tutorial main page Tensorflow-YOLO_v3 Tutorial

**Prerequisites**

This demo requires the following pip packages:

```python
import sys
!{sys.executable} -m pip install pillow matplotlib pycocotools==2.0.2 --force --extra-index-url=https://pip.repos.neuron.amazonaws.com
```

**Part 1: Download Dataset and Generate Pretrained SavedModel**

**Download COCO 2017 validation dataset**

We start by downloading the COCO validation dataset, which we will use to validate our model. The COCO 2017 dataset is widely used for object-detection, segmentation and image captioning.

```bash
!unzip -q val2017.zip
!unzip annotations_trainval2017.zip
```

```bash
!ls
```
Generate YOLO v3 tensorflow SavedModel (pretrained on COCO 2017 dataset)


```bash
$ run yolo_v3_coco_saved_model.py ./yolo_v3_coco_saved_model
```

This tensorflow SavedModel can be loaded as a tensorflow predictor. When a JPEG format image is provided as input, the output result of the tensorflow predictor contains information for drawing bounding boxes and classification results.

```python
import json
import tensorflow as tf
from PIL import Image
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# launch predictor and run inference on an arbitrary image in the validation dataset
yolo_pred_cpu = tf.contrib.predictor.from_saved_model('./yolo_v3_coco_saved_model')
image_path = './val2017/000000581781.jpg'
with open(image_path, 'rb') as f:
    feeds = {'image': [f.read()]}
results = yolo_pred_cpu(feeds)

# load annotations to decode classification result
with open('./annotations/instances_val2017.json') as f:
    annotate_json = json.load(f)
label_info = {idx+1: cat['name'] for idx, cat in enumerate(annotate_json['categories'])}

# draw picture and bounding boxes
fig, ax = plt.subplots(figsize=(10, 10))
ax.imshow(Image.open(image_path).convert('RGB'))
wanted = results['scores'][0] > 0.1
for xyxy, label_no_bg in zip(results['boxes'][0][wanted], results['classes'][0][wanted]):
    rect = patches.Rectangle((xywh[0], xywh[1]), xywh[2], xywh[3],
                              linewidth=1, edgecolor='g', facecolor='none')
    ax.add_patch(rect)
    rx, ry = rect.get_xy()
    rx = rx + rect.get_width() / 2.0
    ax.annotate(label_info[label_no_bg + 1], (rx, ry), color='w', backgroundcolor='g', fontsize=10,
                ha='center', va='center', bbox=dict(boxstyle='square,pad=0.01', fc='g', ec='none', alpha=0.5))
plt.show()
```
Part 2: Compile the Pretrained SavedModel for Neuron

We make use of the Python compilation API `tfn.saved_model.compile` that is available in `tensorflow-neuron<2`. For the purpose of reducing Neuron runtime overhead, it is necessary to make use of arguments `no_fuse_ops` and `minimum_segment_size`. Compiled model is saved in `./yolo_v3_coco_saved_model_neuron`.

```python
[ ]: import shutil
import tensorflow as tf
import tensorflow.neuron as tfn

def no_fuse_condition(op):
    return op.name.startswith('Preprocessor') or op.name.startswith('Postprocessor')

with tf.Session(graph=tf.Graph()) as sess:
    tf.saved_model.loader.load(sess, ['serve'], './yolo_v3_coco_saved_model')
    no_fuse_ops = [op.name for op in sess.graph.get_operations() if no_fuse_condition(op)]
shutil.rmtree('./yolo_v3_coco_saved_model_neuron', ignore_errors=True)
result = tfn.saved_model.compile('./yolo_v3_coco_saved_model', './yolo_v3_coco_saved_model_neuron',
    no_fuse_ops=no_fuse_ops,
    minimum_segment_size=100,
    batch_size=2,
    dynamic_batch_size=True,
)
print(result)

Deploy the model on Inferentia

Part 3: Evaluate Model Quality after Compilation

Define evaluation functions

We first define some handy helper functions for running evaluation on the COCO 2017 dataset.

```python
[ ]: import os
import json
import time
import numpy as np
import tensorflow as tf
from pycocotools.coco import COCO
from pycocotools.cocoeval import COCOeval

def cocoapi_eval(jsonfile, style, coco_gt=None, anno_file=None, max_dets=(100, 300, 1000)):
    """"""
Args:
style: COCOeval style, can be 'bbox', 'segm' and 'proposal'.
coco_gt: Whether to load COCOAPI through anno_file,
eg: coco_gt = COCO(anno_file)
anno_file: COCO annotations file.
max_dets: COCO evaluation maxDets.

"""
assert coco_gt is not None or anno_file is not None

if coco_gt is None:
coco_gt = COCO(anno_file)
print("Start evaluate...")
coco_dt = coco_gt.loadRes(jsonfile)
if style == 'proposal':
coco_eval = COCOeval(coco_gt, coco_dt, 'bbox')
coco_eval.params.useCats = 0
coco_eval.params.maxDets = list(max_dets)
else:
coco_eval = COCOeval(coco_gt, coco_dt, style)
coco_eval.evaluate()
coco_eval.accumulate()
coco_eval.summarize()
return coco_eval.stats

def bbox_eval(anno_file, bbox_list):
coco_gt = COCO(anno_file)

outfile = 'bbox_detections.json'
print('Generating json file...')
with open(outfile, 'w') as f:
    json.dump(bbox_list, f)

map_stats = cocoapi_eval(outfile, 'bbox', coco_gt=coco_gt)
return map_stats

def get_image_as_bytes(images, eval_pre_path):
    batch_im_id_list = []
    batch_im_name_list = []
    batch_img_bytes_list = []
    n = len(images)
    batch_im_id = []
    batch_im_name = []
    batch_img_bytes = []
    for i, im in enumerate(images):
        im_id = im['id']
        file_name = im['file_name']
        if i % eval_batch_size == 0 and i != 0:
            batch_im_id_list.append(batch_im_id)
            batch_im_name_list.append(batch_im_name)
            batch_img_bytes_list.append(batch_img_bytes)
            batch_im_id = []
            batch_im_name = []
            batch_img_bytes = []
        batch_im_id.append(im_id)
With open(os.path.join(eval_pre_path, file_name), 'rb') as f:
    batch_img_bytes.append(f.read())

return batch_im_id_list, batch_im_name_list, batch_img_bytes_list

def analyze_bbox(results, batch_im_id, _clsid2catid):
    bbox_list = []
    k = 0
    for boxes, scores, classes in zip(results['boxes'], results['scores'],
    results['classes']):
        if boxes is not None:
            im_id = batch_im_id[k]
            n = len(boxes)
            for p in range(n):
                clsid = classes[p]
                score = scores[p]
                xmin, ymin, xmax, ymax = boxes[p]
               catid = (_clsid2catid[int(clsid)])
                w = xmax - xmin + 1
                h = ymax - ymin + 1
                bbox = [xmin, ymin, w, h]
                # Round to the nearest 10th to avoid huge file sizes, as
                # COCO suggests
                bbox = [round(float(x) * 10) / 10 for x in bbox]
                bbox_res = {
                    'image_id': im_id,
                    'category_id': catid,
                    'bbox': bbox,
                    'score': float(score),
                }
                bbox_list.append(bbox_res)
                k += 1
    return bbox_list

Here is the actual evaluation loop. To fully utilize all four cores on one Inferentia, the optimal setup is to run multi-threaded inference using a ThreadPoolExecutor. The following cell is a multi-threaded adaptation of the evaluation routine at https://github.com/miemie2013/Keras-YOLOv4/blob/910c4c6f7265f3826c3ee0f784f94b4f6516bf/tools/cocotools.py#L97.

from concurrent import futures
def evaluate(yolo_predictor, images, eval_pre_path, anno_file, eval_batch_size, _clsid2catid):
    batch_im_id_list, batch_im_name_list, batch_img_bytes_list = get_image_as_bytes(images, eval_pre_path)
    # warm up
    yolo_predictor({'image': np.array(batch_img_bytes_list[0], dtype=object)})

    with futures.ThreadPoolExecutor(4) as exe:
        fut_im_list = []
        fut_list = []
        # (continues on next page)
Evaluate mean average precision (mAP) score

Here is the code to calculate mAP scores of the YOLO v3 model. The expected mAP score is around 0.328 if we use the pretrained weights.

```python
[ ]: yolo_pred = tf.contrib.predictor.from_saved_model('./yolo_v3_coco_saved_model_neuron')
val_coco_root = './val2017'
val_annotate = './annotations/instances_val2017.json'
eval_batch_size = 8
with open(val_annotate, 'r', encoding='utf-8') as f2:
    for line in f2:
(continues on next page)
Running SSD300 with AWS Neuron

Update 11/16: The model checkpoint link https://api.ngc.nvidia.com/v2/models/nvidia/ssdpyt_fp32/versions/1/files/nvidia_ssdpyt_fp32_20190225.pt is currently broken and the AWS Neuron team is working on providing an alternative source.

This demo shows a Neuron compatible SSD300 implementation that is functionally equivalent to open source SSD300 model. This demo uses TensorFlow-Neuron, PyTorch SSD300 model and checkpoint (https://pytorch.org/hub/nvidia_deeplearningexamples_ssd/) and also shows the performance achieved by the Inf1 instance.

Table of Contents

1. Launch EC2 instance and update AWS Neuron SDK software
2. Generating Neuron compatible SSD300 TensorFlow SavedModel
   • Convert open source PyTorch SSD300 model and checkpoint into Neuron compatible SSD300 TensorFlow SavedModel
3. Evaluate the generated SSD300 TensorFlow SavedModel for both accuracy and performance
   • Running threaded inference through the COCO 2017 validation dataset

Launch EC2 instances and update tensorflow-neuron and neuron-cc

For this demo, launch one inf1.xlarge EC2 instance. We recommend using the latest Ubuntu 18 Deep Learning AMI (DLAMI).

Please configure your ubuntu16/ubuntu18/yum repo following the steps in the Setup Guide in order to install tensorflow-model-server-neuron.

Generating Neuron compatible SSD300 TensorFlow SavedModel

First connect to your inf1.xlarge instance
Compile open source PyTorch SSD300 model and checkpoint into Neuron compatible
SSD300 TensorFlow SavedModel

In the same directory ssd300_demo, run the following:

1. Create venv and install dependencies

```
sudo apt update
sudo apt install g++ python3-dev python3-venv unzip
sudo apt install tensorflow-model-server-neuron
python3 -m venv env
source ./env/bin/activate
pip install pip setuptools --upgrade
pip install -r ./requirements.txt --extra-index-url=https://pip.repos.neuron.amazonaws.com
```

2. Clone NVIDIA’s DeepLearningExamples repo that contains PyTorch SSD300.

```
git clone https://github.com/NVIDIA/DeepLearningExamples.git
cd DeepLearningExamples
git checkout a644350589f9abc91b203f73e686a50f5d6f3e96
cd..
```

3. Download PyTorch SSD300 checkpoint file.

```
```


```
curl -LO http://images.cocodataset.org/zips/val2017.zip
unzip ./val2017.zip
curl -LO http://images.cocodataset.org/annotations/annotations_trainval2017.zip
unzip ./annotations_trainval2017.zip
```

5. Convert PyTorch SSD300 model and checkpoint into a Neuron-compatible TensorFlow SavedModel.

```
python ssd300_model.py --torch_checkpoint=./nvidia_ssdpyt_fp32_20190225.pt --output_saved_model=./ssd300_tf_neuron/1
```

This converts PyTorch SSD300 model and checkpoint to a Neuron-compatible TensorFlow SavedModel using tensorflow-neuron and neuron-cc. The compilation output is stored in ./ssd300_tf_neuron.

6. Launch the tensorflow-model-server-neuron gRPC server at default port 8500 in the background.

```
tensorflow_model_server_neuron --model_base_path=$(pwd)/ssd300_tf_neuron &
```

7. In client, evaluate the Neuron-compatible TensorFlow SavedModel for both accuracy and performance. Note that this client by default assumes a tensorflow-model-server-neuron listening at localhost:8500. On inf1.xlarge, the expected throughput is 100 images/second once the server is fully warmed up, and the expected mean average precision (mAP) is 0.253.

```
python ssd300_evaluation_client.py --val2017=./val2017 --instances_val2017_json=./annotations/instances_val2017.json
```
8. After running the demo, please cleanup resources allocated in Neuron runtime by gracefully killing the `tensorflow_model_server_neuron` process, e.g.,

```bash
killall tensorflow_model_server_neuron
```

Tensorflow ResNet 50 Optimization Tutorial

**Note: this tutorial runs on tensorflow-neuron 1.x only**

**Introduction:**

In this tutorial we provide three main sections:

- Take a Resnet 50 model and perform optimizations on it
- Compile the model with different batch sizes and Neuroncore Group sizes (read about Neuroncore Group sizes here: https://awsdocs-neuron.readthedocs-hosted.com/en/latest/neuron-guide/neuron-runtime/nrt-theory-of-operation.html#neuron-core-group)
- Run inference on our multiple compiled models to see which has the best throughput

Before running the following verify this Jupyter notebook is running “conda_aws_neuron_tensorflow_p36” kernel. You can select the Kernel from the “Kernel -> Change Kernel” option on the top of this Jupyter notebook page.

**Install Dependencies**

```bash
[pip install pillow # Necessary for loading images
!pip install 'tensorflow-neuron<2' --extra-index-url=https://pip.repos.neuron.amazonaws.com
```

**Compile**

The following example shows how to compile a FP16 ResNet50 network using various batching parameters to find the optimal solution. On inf1.6xlarge, run through the following steps to get a optimized Resnet 50 model. First, extract Keras ResNet50 FP32 (resnet50_fp32_keras.pb will be generated):

```python
import re
import argparse
import tensorflow as tf
import numpy as np
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input, decode_predictions
from google.protobuf import text_format
import tensorflow.python.saved_model

# set Keras global configurations
tf.keras.backend.set_learning_phase(0)
```

(continues on next page)
tf.keras.backend.set_image_data_format('channels_last')

float_type = 'float32'
float_type2 = 'fp32'
tf.keras.backend.set_floatx(float_type)

# load pre-trained model using Keras
model_name = 'resnet50_%s%sl.keras.backend.get_session()' % (float_type2)
model = ResNet50(weights='imagenet')

# various save files
frozen_file = model_name + '.pb'
opt_file = model_name + '_opt.pb'

# obtain parameters
model_input = model.input.name.replace(':0', '')
model_output = model.output.name.replace(':0', '')
batch, height, width, channels = model.input.shape
print("model, frozen file, optimized file, input size, input node, output node,")
print("%s, %s, %s, %dx%dx%d, %s, %s" %(model_name, frozen_file, opt_file, width, height, channels, model_input, model_output))

# obtain the TF session
sess = tf.compat.v1.keras.backend.get_session()

# save checkpoint files for freeze_graph
ckpt_file = '/tmp/' + model_name + '/' + model_name + '.ckpt'
graph_file = '/tmp/' + model_name + '/' + model_name + '.pb'
tf.compat.v1.train.Saver().save(sess, ckpt_file)
tf.io.write_graph(sess.graph.as_graph_def(), logdir='.', name=graph_file, as_text=False)
print(model_output)
with tf.compat.v1.Session(graph=tf.Graph()) as sess:
    saver = tf.compat.v1.train.import_meta_graph(ckpt_file + '.meta')
saver.restore(sess, ckpt_file)
output_graph_def = tf.compat.v1.graph_util.convert_variables_to_constants(sess, tf.compat.v1.get_default_graph().as_graph_def(), [model_output])
output_graph_def = tf.compat.v1.graph_util.remove_training_nodes(output_graph_def, protected_nodes=[model_output])
with open(frozen_file, 'wb') as f:
    f.write(output_graph_def.SerializeToString())

Optimize the extracted Keras ResNet50 FP32 graph for inference before casting (resnet50_fp32_keras_opt.pb will be generated) with the following transformations to the graph:

- Remove Identity and CheckNumerics nodes
- Fold FusedBatchNorm constants into previous Conv2D weights
- Fold other constants
- Strip unused nodes
- Sort by execution order
import copy
import string
from google.protobuf import text_format
from tensorflow.core.framework import node_def_pb2
from tensorflow.core.framework import attr_value_pb2
from tensorflow.python.framework import tensor_util
from tensorflow.tools.graph_transforms import TransformGraph

def clear_input(node):
    for i in range(len(node.input)):
        node.input.pop()

def replace_name(node, name):
    node.name = name

def replace_input(node, input_name, new_name):
    # node.input.replace(input_name, new_name)
    temp = []
    for i in node.input:
        temp.extend([new_name if i == input_name else i])
    clear_input(node)
    for i in temp:
        node.input.extend([i])

def swap_names(node1, node2):
    temp = node2.name
    node2.name = node1.name
    node1.name = temp

def get_const_node(const_node_name, const_by_name):
    name = re.sub("/read\$", "", const_node_name)
    return const_by_name[name]

def get_const_ndarray(const_node_name, const_by_name):
    name = re.sub("/read\$", "", const_node_name)
    node = const_by_name[name]
    return tf.make_ndarray(node.attr.get("value").tensor)

def adjust_bias_values(bias_node, fbn_node, const_by_name):
    bias_val = get_const_ndarray(bias_node.input[1], const_by_name)
    gamma_val = get_const_ndarray(fbn_node.input[1], const_by_name)
    mean_val = get_const_ndarray(fbn_node.input[3], const_by_name)
    variance_val = get_const_ndarray(fbn_node.input[4], const_by_name)
    new_bias = bias_val * gamma_val / np.sqrt(variance_val)
    new_tensor = tensor_util.make_tensor_proto(new_bias, new_bias.dtype, new_bias.shape)
    bias_const_node = get_const_node(bias_node.input[1], const_by_name)
    bias_const_node.attr["value"].CopyFrom(attr_value_pb2.AttrValue(tensor=new_tensor))

def MoveBiasAddAfterFusedBatchNorm(graphdef):
    """fold_batch_norm function of TransformGraph is unable to fold Keras ResNet50 because of BiasAdd between Conv2D and FusedBatchNorm (BiasAdd is not needed if FusedBatchNorm is used, but it exists in Keras ResNet50). Here, we move BiasAdd to after FusedBatchNorm, and adjust bias value by gamma/sqrt(variance)."""

(continues on next page)
sess = tf.compat.v1.Session(graph=tf.import_graph_def(graphdef))
output_graph_def = tf.compat.v1.GraphDef()
node_by_name = {}
const_by_name = {}
for node in graphdef.node:
    # Hack: use FusedBatchNormV2 so fold_batch_norm can recognize
    if node.op == "FusedBatchNormV3":
        node.op = "FusedBatchNorm"
        del(node.attr["U"])
        #import pdb; pdb.set_trace()
        copied_node = node_def_pb2.NodeDef()
        copied_node.CopyFrom(node)
        node_by_name[node.name] = copied_node
        skip_add_node = False
    # Switch Mul/BiasAdd in Keras RN50 so fold_batch_norm transform would work
    if node.op == "Const":
        const_by_name[node.name] = copied_node
    elif node.op.startswith("FusedBatchNorm"): inputs = node.input
    for i in inputs:
        input_node = node_by_name[i]
        if input_node.op == "BiasAdd":
            output_graph_def.node.remove(input_node)
            input_node_input0 = input_node.input[0]
            # Adjust bias values (multiply by scale/sqrt(variance))
            adjust_bias_values(input_node, node, const_by_name)
            # Hack: swap names to avoid changing input of activation
            swap_names(copied_node, input_node)
            # Fix inputs for these two ops
            replace_input(copied_node, i, input_node_input0)
            replace_input(input_node, input_node_input0, copied_node.name)
            # Fix order in node list
            output_graph_def.node.extend([copied_node])
        else:
            output_graph_def.node.extend([input_node])
        skip_add_node = True
    # Add maybe-modified nodes if not already done
    if not skip_add_node:
        output_graph_def.node.extend([copied_node])
return output_graph_def

def FoldFusedBatchNorm(graph_def):
    """Optimize training graph for inference:
    - Remove Identity and CheckNumerics nodes
    - Fold FusedBatchNorm constants into previous Conv2D weights
    - Fold other constants
    - Strip unused nodes
    - Sort by execution order
    ""
    transformed_graph_def = TransformGraph (graph_def,
        ['input_1'],
        ['probs/Softmax'],
        ['add_default_attributes'],
        'remove_nodes(op=Identity, op=CheckNumerics)',
        (continues on next page)
'fold_constants(ignore_errors=true)',
'fold_batch_norms',
'fold_old_batch_norms',
'strip_unused_nodes',
'sort_by_execution_order',
])
return transformed_graph_def

def load_graph(model_file):
    graph_def = tf.compat.v1.GraphDef()
    with open(model_file, "rb") as f:
        graph_def.ParseFromString(f.read())
    return graph_def

graph_orig = load_graph('resnet50_fp32_keras.pb')
graph_mod = MoveBiasAddAfterFusedBatchNorm(graph_orig)
graph_mod2 = FoldFusedBatchNorm(graph_mod)
with tf.io.gfile.GFile('resnet50_fp32_keras_opt.pb', "wb") as f:
    f.write(graph_mod2.SerializeToString())

Convert full graph to FP16 (resnet50_fp16_keras_opt.pb will be generated. This will take about a minute.

```python
from tensorflow.core.framework import graph_pb2
from tensorflow.python.platform import gfile

def ConvertFP32ToOther(graph_def):
    """Converts an FP32 network by casting all constants (weights) to a lower precision floating point type (FP16) and updating the dtypes everywhere.""
    cast_type = "float16"
    sess = tf.Session(graph=tf.import_graph_def(graph_def))
    output_graph_def = graph_pb2.GraphDef()
    dummy_tensor = sess.run(tf.constant([0.1]))
    dummy_tensor_proto = tensor_util.make_tensor_proto(dummy_tensor, dtype=cast_type, shape=dummy_tensor.shape)
    dummy_tensor32 = sess.run(tf.constant([0.1]))
    dummy_tensor_proto32 = tensor_util.make_tensor_proto(dummy_tensor, dtype=tf.float32, shape=dummy_tensor.shape)
    dt_float_type_attr = attr_value_pb2.AttrValue(type=dummy_tensor_proto32.dtype)
    dt_half_type_attr = attr_value_pb2.AttrValue(type=dummy_tensor_proto.dtype)
    for node in graph_def.node:
        output_node = node_def_pb2.NodeDef()
        output_node.CopyFrom(node)
        if (node.op == "Const"):
            if (node.attr["dtype"] == dt_float_type_attr):
                a = tensor_util.MakeNdarray(node.attr["value"].tensor)
                a = tf.cast(a, cast_type)
                a = sess.run(a)
                output_node.attr["dtype"].CopyFrom(dt_half_type_attr)
                output_node.attr["value"].CopyFrom(
                    tensor=tensor_util.make_tensor_proto(a,
                        dtype=cast_type, shape=a.shape))
            else:

(continues on next page)
if ("T" in node.attr.keys()):
    if (output_node.attr["T"] == dt_float_type_attr):
        output_node.attr["T"].CopyFrom(dt_half_type_attr)
if ("Tparams" in node.attr.keys()):
    if (output_node.attr["Tparams"] == dt_float_type_attr):
        output_node.attr["Tparams"].CopyFrom(dt_half_type_attr)
if ("dtype" in node.attr.keys()):
    if (node.attr["dtype"] == dt_float_type_attr):
        output_node.attr["dtype"].CopyFrom(dt_half_type_attr)
if ("SrcT" in node.attr.keys()):
    if (node.attr["SrcT"] == dt_float_type_attr):
        output_node.attr["SrcT"].CopyFrom(dt_half_type_attr)
if ("DstT" in node.attr.keys()):
    if (node.attr["DstT"] == dt_float_type_attr):
        output_node.attr["DstT"].CopyFrom(dt_half_type_attr)

output_graph_def.node.extend([output_node])
return output_graph_def

def load_graph(model_file):
    graph_def = tf.GraphDef()
    with open(model_file, "rb") as f:
        graph_def.ParseFromString(f.read())
    return graph_def

graph_f32 = load_graph('resnet50_fp32_keras_opt.pb')
graph_f16 = ConvertFP32ToOther(graph_f32)
output_xformed_graph_name = 'resnet50_fp16_keras_opt.pb'
with gfile.GFile(output_xformed_graph_name, "wb") as f:
    f.write(graph_f16.SerializeToString())

Run the compilation script to sweep through various batch sizes up to 5 and several NeuronCore Group sizes up to 16. The script calls the compilation script pb2sm_compile.py which tries to perform compilation. Some error messages are expected due to known issues (see Known Issues section in the tutorial). If you run all the configurations it will take about 45 minutes.

```bash
#!/usr/bin/env bash

echo "" > full_sweep.log
echo "" > full_sweep_results.txt

results=
for b in $(seq 1 5); do
    for i in 1 2 4 8 12 16; do
        python pb2sm_compile.py --batch_size=$b --neuroncore-pipeline-cores=4 $1 | tee -a full_sweep.log;
        results[$b]+="", " tail -l full_sweep.log"
    done
done

head="batch"
for i in 1 2 4 8 12 16; do
    head+="", nc${i}/"
```
done
echo $head | tee -a full_sweep_results.txt
for b in $(seq 1 5); do
    echo $b\${results[$b]/ } | tee -a full_sweep_results.txt
done
```

You should see some output like this:

```plaintext
INFO: Compilation finished in 95 seconds with 99.5% operations placed on Inferentia

1

*** Batch size 1, num NeuronCores 2 (input shape: (1, 224, 224, 3), saved model dir: rn50_fp16_compiled_b1_nc2) ***

INFO: Compilation finished in 95 seconds with 99.5% operations placed on Inferentia

1

*** Batch size 1, num NeuronCores 4 (input shape: (1, 224, 224, 3), saved model dir: rn50_fp16_compiled_b1_nc4) ***

INFO: Compilation finished in 95 seconds with 99.5% operations placed on Inferentia

1

... (outputs removed)

*** Batch size 5, num NeuronCores 16 (input shape: (5, 224, 224, 3), saved model dir: rn50_fp16_compiled_b5_nc16) ***

ERROR: Compilation finished in 120 seconds with less than 50% operations placed on Inferentia (0.0%)

INFO: Retry compilation without static weights

ERROR: Retry compilation finished in 137 seconds with less than 50% operations placed on Inferentia (0.0%)

0

The file full_sweep_results.txt shows a summary of the sweep results with latest Neuron 1/27/20 release (0 means compilation unsuccessful and 0 ops mapped to Inferentia and non-static weights, 2 means most ops mapped to Inferentia and using static weights):

<table>
<thead>
<tr>
<th>batch, nc1, nc2, nc4, nc8, nc12, nc16</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 1, 1, 1, 2, 2, 2</td>
</tr>
<tr>
<td>2, 1, 1, 0, 1, 2, 2</td>
</tr>
<tr>
<td>3, 1, 1, 1, 1, 1</td>
</tr>
<tr>
<td>4, 1, 1, 0, 1, 1</td>
</tr>
<tr>
<td>5, 1, 1, 0, 0, 0</td>
</tr>
</tbody>
</table>
```

Chapter 3. TensorFlow Neuron
Inference

Run inference over different batch sizes and Neuroncore groups to obtain throughput and latency results for ResNet50. To apply dynamic batching, the user batch size is set to 10x the compiled batch size, in order to keep input queue full and to amortize framework-to-Neuron overhead.

Note: The results are based on the Neuron v1.12.2 (Mar 4th 2021) release. These will continue improve as we increase Neuron performance.

```bash
[ ]: !cd ~/aws-neuron-sdk/src/examples/tensorflow/keras_resnet50/
  !echo "" > batch.log
  !for i in $(seq 1 5); do python infer_resnet50_keras_loadtest.py --batch_size=$i --neuroncore-pipeline-cores=1 | tee -a batch.log; done
  !for i in $(seq 1 5); do python infer_resnet50_keras_loadtest.py --batch_size=$i --neuroncore-pipeline-cores=2 | tee -a batch.log; done
  !for i in $(seq 1 5); do python infer_resnet50_keras_loadtest.py --batch_size=$i --neuroncore-pipeline-cores=4 | tee -a batch.log; done
  !for i in $(seq 1 5); do python infer_resnet50_keras_loadtest.py --batch_size=$i --neuroncore-pipeline-cores=8 | tee -a batch.log; done
  !for i in $(seq 1 5); do python infer_resnet50_keras_loadtest.py --batch_size=$i --neuroncore-pipeline-cores=12 | tee -a batch.log; done
  !for i in $(seq 1 5); do python infer_resnet50_keras_loadtest.py --batch_size=$i --neuroncore-pipeline-cores=16 | tee -a batch.log; done
```

The file batch.log now contains the results for each batch size. We can look at the throughput values to get an idea of which models are performing well. The output should look something like this:

The model best model configuration for throughput (if you run on an Inf1.6xlarge as suggested in the tutorial) is batch size 5 NeuronCore group size 1. Increasing batch size usually helps to increase throughput (up to a certain extent). Increasing the NeuronCore group size actually made throughput worse. The reason for that is that resnet50 is a relatively small model, it can easily fit on 1 neuron core. By forcing it to be split among multiple neuron cores, it actually just added more overhead than just having on a single core.

```bash
*** Compiled batch size 1, user batch size 10, num NeuronCores 1 (input shape: (10, 224, 224, 3), saved model dir: ./rn50_fp16_compiled_b1_nc1/1)

Instance type inf1.6xlarge with 16 NeuronCores
NEURON_MAX_NUM_INFERS (env): 4
NEURONCORE_GROUP_SIZES (env): 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
NUM THREADS: 32
NUM_LOOPS_PER_THREAD: 100
USER_BATCH_SIZE: 10
Throughput values collected:
[3160, 3200, 3210, 3210, 3170, 3170]
(rest of outputs removed)
```
Known Issues

Unable to compile with batch and num NeuronCores combination

For some combination of batch and number of NeuronCores setting, you may see an internal compiler error as below. Please see the sweep result above for Neuron 1/27/20 release. Furthermore, if using auto-casting to bfloat16 from FP32 network and batch size is larger than 1 would result in the same error.
Natural Language Processing

- Tensorflow 1.x - Running TensorFlow BERT-Large with AWS Neuron [html]
- Tensorflow 2.x - HuggingFace Pipelines distilBERT with Tensorflow2 Neuron [html] [notebook]

Running TensorFlow BERT-Large with AWS Neuron

This example shows a Neuron compatible BERT-Large implementation that is functionally equivalent to open source BERT-Large model. This demo uses TensorFlow-Neuron, BERT-Large weights fine tuned for MRPC and also shows the performance achieved by the Inf1 instance. For users who want to use public BERT SavedModels please also follow the steps described Using public BERT SavedModels.

Launch EC2 instances

For this demo, launch two EC2 instances:

- a c5.4xlarge instance for compiling the BERT-Large Model and
- an inf1.xlarge instance for running inference

For both of these instances choose the latest Ubuntu 18 Deep Learning AMI (DLAMI).

Compiling Neuron compatible BERT-Large

First connect to a c5.4xlarge instance and update tensorflow-neuron and neuron-cc
Update compilation EC2 instance

Update to the latest neuron software by executing the following commands:

```
source activate aws_neuron_tensorflow_p36
conda update tensorflow-neuron
conda update numpy
```

Note: if your tensorflow-neuron version on the inference instance is lower than 1.15.0.1.0.1333.0, you will need to run this demo on inf1.2xlarge instead of inf1.xlarge.

Compile open source BERT-Large saved model using Neuron compatible BERT-Large implementation

Neuron software works with TensorFlow saved models. Users should bring their own BERT-Large saved model for this section. This demo will run inference for the MRPC task and the saved model should be fine tuned for MRPC. Users who need additional help to fine-tune the model for MRPC or to create a saved model can refer to Appendix 1.

In the same conda environment and directory bert_demo scripts, run the following:

```
git clone https://github.com/aws/aws-neuron-sdk
cd ~/aws-neuron-sdk/src/examples/tensorflow/bert_demo/
export BERT_LARGE_SAVED_MODEL="/path/to/user/bert-large/savedmodel"
python bert_model.py --input_saved_model $BERT_LARGE_SAVED_MODEL --output(saved_model ./bert-saved-model-neuron --batch_size=6 --aggressive_optimizations
```

This compiles BERT-Large pointed to by $BERT_LARGE_SAVED_MODEL for an input size of 128 and batch size of 6. The compilation output is stored in bert-saved-model-neuron. Copy this to your Inf1 instance for inferencing.

The bert_model.py script encapsulates all the steps necessary for this process. For details on what is done by bert_model.py please refer to Appendix 2.

Running the inference demo

Connect to your inf1.xlarge instance and update tensorflow-neuron, aws-neuron-runtime and aws-neuron-tools.

Update inference EC2 instance

Update to the latest neuron software by executing the following commands:

```
source activate aws_neuron_tensorflow_p36
conda update tensorflow-neuron
conda update numpy
```
Launching the BERT-Large demo server

Copy the compiled model (bert-saved-model-neuron) from your c5.4xlarge to your inf1.xlarge instance. Place the model in the same directory as the bert_demo scripts. Then from the same conda environment launch the BERT-Large demo server:

```
cd ~/aws-neuron-sdk/src/examples/tensorflow/bert_demo/
python bert_server.py --dir bert-saved-model-neuron --batch 6 --parallel 4
```

This loads 4 BERT-Large models, one into each of the 4 NeuronCores found in an inf1.xlarge instance. For each of the 4 models, the BERT-Large demo server opportunistically stitches together asynchronous requests into batch 6 requests. When there are insufficient pending requests, the server creates dummy requests for batching.

Wait for the bert_server to finish loading the BERT-Large models to Inferentia memory. When it is ready to accept requests it will print the inferences per second once every second. This reflects the number of real inferences only. Dummy requests created for batching are not credited to inferentia performance. Once the inferences are done you can send a keyboard interrupt to print out the average throughput of your run.

Sending requests to server from multiple clients

Wait until the bert demo server is ready to accept requests. Then on the same inf1.xlarge instance, launch a separate linux terminal. From the bert_demo directory execute the following commands:

```
source activate aws_neuron_tensorflow_p36
cd ~/aws-neuron-sdk/src/examples/tensorflow/bert_demo/
for i in {1..96}; do python bert_client.py --cycle 128 & done
```

This spins up 96 clients, each of which sends 128 inference requests.

Printing latency metrics

After all your requests have been sent to your server you can run the following command:

```
python latency_printer.py
```

Using public BERT SavedModels

We are now providing a compilation script that has better compatibility with various flavors of BERT SavedModels generated from https://github.com/google-research/bert. Here are the current limitations:

1. You did not change modeling.py
2. BERT SavedModel is generated using estimator.export_saved_model
3. BERT SavedModel uses fixed sequence length 128 (you may check by saved_model_cli show --dir /path/to/user/bert/savedmodel --all)
4. neuron-cc version is at least 1.0.12000.0
5. aws-neuron-runtime version is at least 1.0.7000.0
6. The --batch_size argument specified in this script is at most 4
Example usage is shown below:

```bash
export BERT_LARGE_SAVED_MODEL="/path/to/user/bert-large/savedmodel"

cd ~/aws-neuron-sdk/src/examples/tensorflow/bert_demo/

python bert_no_model.py --input_saved_model $BERT_LARGE_SAVED_MODEL --output_saved_model ./bert-saved-model-neuron --batch_size=1
```

### Appendix 1

Users who need help finetuning BERT-Large for MRPC and creating a saved model may follow the instructions here.

Connect to the c5.4xlarge compilation EC2 instance you started above and download these three items:

1. clone this github repo.
2. download GLUE data as described here. Do not run the finetuning command.
3. download a desired pre-trained BERT-Large checkpoint from here. This is the model we will fine tune.

Next edit `run_classifier.py` in the cloned bert repo to apply the patch described in the following git diff.

```diff
diff --git a/run_classifier.py b/run_classifier.py
index 817b147..c9426bc 100644
--- a/run_classifier.py
+++ b/run_classifier.py
@@ -955,6 +955,18 @@
      drop_remainder=predict_drop_remainder)
     result = estimator.predict(input_fn=predict_input_fn)
     features = {
+      "input_ids": tf.placeholder(shape=[None, FLAGS.max_seq_length],
+                                  dtype=tf.int32, name='input_ids'),
+      "input_mask": tf.placeholder(shape=[None, FLAGS.max_seq_length],
+                                  dtype=tf.int32, name='input_mask'),
+      "segment_ids": tf.placeholder(shape=[None, FLAGS.max_seq_length],
+                                  dtype=tf.int32, name='segment_ids'),
+      "label_ids": tf.placeholder(shape=[None], dtype=tf.int32, name='label_ids'),
+      "is_real_example": tf.placeholder(shape=[None], dtype=tf.int32,
+                                          name='is_real_example'),
+     }
+     serving_input_fn = tf.estimator.export.build_raw_serving_input_receiver_fn(features)
     estimator._export_to_tpu = False  # !!important to add this
+     estimator.export_saved_model(
+      export_dir_base='./bert_classifier_saved_model',
+      serving_input_receiver_fn=serving_input_fn)

     output_predict_file = os.path.join(FLAGS.output_dir, "test_results.tsv")
     with tf.gfile.GFile(output_predict_file, "w") as writer:
```

**NOTE:** Users who are interested may refer to this link for additional background information on the patch but it is not necessary for running this demo.

Then from the bert_demo directory run the following:
source activate aws_neuron_tensorflow_p36
cd ~/aws-neuron-sdk/src/examples/tensorflow/bert_demo/
export BERT_REPO_DIR="/path/to/cloned/bert/repo/directory"
export GLUE_DIR="/path/to/glue/data/directory"
export BERT_BASE_DIR="/path/to/pre-trained/bert-large/checkpoint/directory"
./tune_save.sh

The a saved model will be created in $BERT_REPO_DIR/bert-saved-model/random_number/. Where, random_number is a random number generated for every run. Use this saved model to continue with the rest of the demo.

**Appendix 2**

For all BERT variants, we currently need to augment the standard Neuron compilation process for performance tuning. In the future, we intend to automate this tuning process. This would allow users to use the standard Neuron compilation process, which requires only a one line change in user source code. The standard compilation process is described *Running Neuron Apache MXNet (Incubating) ResNet50 on Inferentia*.

The augmented Neuron compilation process is encapsulated by the bert_model.py script, which performs the following things:

1. Define a Neuron compatible implementation of BERT-Large. For inference, this is functionally equivalent to the open source BERT-Large. The changes needed to create a Neuron compatible BERT-Large implementation is described in Appendix 3.
2. Extract BERT-Large weights from the open source saved model pointed to by –input_saved_model and associates it with the Neuron compatible model
3. Invoke TensorFlow-Neuron to compile the Neuron compatible model for Inferentia using the newly associated weights
4. Finally, the compiled model is saved into the location given by –output_saved_model

**Appendix 3**

The Neuron compatible implementation of BERT-Large is functionally equivalent to the open source version when used for inference. However, the detailed implementation does differ and here are the list of changes:

1. Data Type Casting: If the original BERT-Large an FP32 model, bert_model.py contains manually defined cast operators to enable mixed-precision. FP16 is used for multi-head attention and fully-connected layers, and fp32 everywhere else. This will be automated in a future release.
2. Remove Unused Operators: A model typically contains training operators that are not used in inference, including a subset of the reshape operators. Those operators do not affect inference functionality and have been removed.
3. Reimplementation of Selected Operators: A number of operators (mainly mask operators), has been reimplemented to bypass a known compiler issue. This will be fixed in a planned future release.
4. Manually Partition Embedding Ops to CPU: The embedding portion of BERT-Large has been partitioned manually to a subgraph that is executed on the host CPU, without noticeable performance impact. In near future, we plan to implement this through compiler auto-partitioning without the need for user intervention.
Utilizing Neuron Capabilities

- Tensorflow 1.x - NeuronCore Groups tutorial [html]
- Tensorflow 1.x - Using NEURON_RT_VISIBLE_CORES with TensorFlow Serving [html]

Using NEURON_RT_VISIBLE_CORES with TensorFlow Serving

TensorFlow serving allows customers to scale-up inference workloads across a network. Neuron TensorFlow Serving uses the same API as normal TensorFlow Serving with two differences: (a) the saved model must be compiled for Inferentia and (b) the entry point is a different binary named tensorflow_model_server_neuron. The binary is found at /usr/local/bin/tensorflow_model_server_neuron and is pre-installed in the DLAMI or installed with APT/YUM tensorflow-model-server-neuron package.

Install TensorFlow Model Server and Serving API

Follow the steps in the Setup Guide.

If using DLAMI and aws_neuron_tensorflow_p36 environment, you can skip the installation step below. Then ensure you install using either apt-get or yum:

```bash
sudo apt-get install tensorflow-model-server-neuron
```

or

```bash
sudo yum install tensorflow-model-server-neuron
```

Also, you would need TensorFlow Serving API (use –no-deps to prevent installation of regular tensorflow):

```bash
pip install --no-deps tensorflow_serving_api==1.15
```

For the example image preprocessing using Keras preprocessing, the Python Imaging Library Pillow is required:

```bash
pip install pillow
```

To workaround h5py issue https://github.com/aws/aws-neuron-sdk/issues/220:

```bash
pip install "h5py<3.0.0"
```

Export and Compile Saved Model

The following example shows graph construction followed by the addition of Neuron compilation step before exporting to saved model.

```python
import tensorflow as tf
import tensorflow.neuron

tf.keras.backend.set_learning_phase(0)
tf.keras.backend.set_image_data_format('channels_last')
```

(continues on next page)
model = tf.keras.applications.ResNet50(weights='imagenet')
sess = tf.keras.backend.get_session()
inputs = {'input': model.inputs[0]}
outputs = {'output': model.outputs[0]}

# save the model using tf.saved_model.simple_save
modeldir = "./resnet50/1"
tf.saved_model.simple_save(sess, modeldir, inputs, outputs)

# compile the model for Inferentia
neuron_modeldir = "./resnet50_inf1/1"
tf.neuron.saved_model.compile(modeldir, neuron_modeldir, batch_size=1)

**Serving Saved Model**

User can now serve the saved model with the tensorflow_model_server_neuron binary. To utilize multiple Neuron Cores, it is recommended to launch multiple tensorflow model servers that listen to the same gRPC port:

```
export NEURON_RT_VISIBLE_CORES=0  # important to set this environment variable before launching model servers
tensorflow_model_server_neuron --model_name=resnet50_inf1 \
  --model_base_path=$(pwd)/resnet50_inf1/ --port=8500
```

# then to run another server on a different neuron core open another window and run this, except this time set NEURON_RT_VISIBLE_CORES=1
# you can keep doing this up to the number of Neuron Cores on your machine

```
export NEURON_RT_VISIBLE_CORES=1
tensorflow_model_server_neuron --model_name=resnet50_inf1 \
  --model_base_path=$(pwd)/resnet50_inf1/ --port=8500
```

The compiled model is staged in Inferentia DRAM by the server to prepare for inference.

### Generate inference requests to the model server

Now run inferences via GRPC as shown in the following sample client code:

```
import numpy as np
import grpc
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.applications.resnet50 import decode_predictions
from tensorflow_serving.apis import predict_pb2
from tensorflow_serving.apis import prediction_service_pb2_grpc

if __name__ == '__main__':
    channel = grpc.insecure_channel('localhost:8500')
    stub = prediction_service_pb2_grpc.PredictionServiceStub(channel)
    img_file = tf.keras.utils.get_file(
        "./kitten_small.jpg",
        "https://raw.githubusercontent.com/awslabs/mxnet-model-server/master/docs/images/kitten_small.jpg")
```
img = load_img(img_file, target_size=(224, 224))
img_array = preprocess_input(img_to_array(img)[:, None, ...])
request = predict_pb2.PredictRequest()
request.model_spec.name = 'resnet50
request.inputs['input'].CopyFrom(
    tf.make_tensor_proto(img_array, shape=img_array.shape))
result = stub.Predict(request)
prediction = tf.make_ndarray(result.outputs['output'])
print(decode_predictions(prediction))

### 3.2.2 TensorFlow 2.x

**TensorFlow-Neuron 2.x Tracing API**

The Neuron tracing API enables tracing TensorFlow 2.x models for deployment on AWS Machine Learning Accelerators.

**Method**

tensorflow.neuron.trace

**Description**

Trace a `keras.Model` or a Python callable that can be decorated by `tf.function`, and return an AWS-Neuron-optimized `keras.Model` that can execute on AWS Machine Learning Accelerators. Tracing is ideal for `keras.Model` that accepts a list of `tf.Tensor` objects and returns a list of `tf.Tensor` objects. It is expected that users will provide example inputs, and the `trace` function will execute `func` symbolically and convert it to a `keras.Model`.

The returned `keras.Model` will support inference only. Attributes or variables held by the original function or `keras.Model` will be dropped.

The returned `keras.Model` can be exported as SavedModel and served using TensorFlow Serving. Please see tensorflow-serving for more information about exporting to saved model and serving using TensorFlow Serving.

Options can be passed to Neuron compiler via the environment variable `NEURON_CC_FLAGS`. For example, the syntax `env NEURON_CC_FLAGS="--neuroncore-pipeline-cores=4"` directs Neuron compiler to compile each subgraph to fit in the specified number of NeuronCores. This number can be less than the total available NeuronCores on an Inf1 instance. See `Neuron compiler CLI Reference Guide` for more information about compiler options.
Arguments

- **func**: The `keras.Model` or function to be traced.

- **example_inputs**: A `tf.Tensor` or a tuple/list/dict of `tf.Tensor` objects for tracing the function. When `example_inputs` is a `tf.Tensor` or a list of `tf.Tensor` objects, we expect `func` to have calling signature `func(example_inputs)`. Otherwise, the expectation is that inference on `func` is done by calling `func(**example_inputs)` when `example_inputs` is a dict. The case where `func` accepts mixed positional and keyword arguments is currently unsupported.

- **subgraph_builder_function**: (Optional) A callable with signature

  ```python
  subgraph_builder_function(node : NodeDef) -> bool  # NodeDef is defined in tensorflow/core/framework/node_def.proto
  ```

  that is used as a call-back function to determine which part of the tensorflow GraphDef given by tracing `func` will be placed on Machine Learning Accelerators.

  If `subgraph_builder_function` is not provided, then `trace` will automatically place operations on Machine Learning Accelerators or on CPU to maximize the execution efficiency.

  If it is provided, and `subgraph_builder_function(node)` returns `True`, and placing `node` on Machine Learning Accelerators will not cause deadlocks during execution, then `trace` will place `node` on Machine Learning Accelerators. If `subgraph_builder_function(node)` returns `False`, then `trace` will place `node` on CPU.

Returns

- An AWS-Neuron-optimized `keras.Model`.

Example Usage

```python
import tensorflow as tf
import tensorflow.neuron as tfn

input0 = tf.keras.layers.Input(3)
dense0 = tf.keras.layers.Dense(3)(input0)
model = tf.keras.Model(inputs=[input0], outputs=[dense0])
example_inputs = tf.random.uniform([1, 3])
model_neuron = tfn.trace(model, example_inputs)  # trace

model_dir = './model_neuron'
model_neuron.save(model_dir)
model_neuron_reloaded = tf.keras.models.load_model(model_dir)
```
Example Usage with Manual Device Placement Using `subgraph_builder_function`

```python
import tensorflow as tf
import tensorflow.neuron as tfn

input0 = tf.keras.layers.Input(3)
dense0 = tf.keras.layers.Dense(3)(input0)
reshape0 = tf.keras.layers.Reshape([1, 3])(dense0)
output0 = tf.keras.layers.Dense(2)(reshape0)
model = tf.keras.Model(inputs=[input0], outputs=[output0])
example_inputs = tf.random.uniform([1, 3])

def subgraph_builder_function(node):
    return node.op == 'MatMul'

model_neuron = tfn.trace(model, example_inputs,
                          subgraph_builder_function=subgraph_builder_function,
                          )
```

TensorFlow 2.x Accelerated Python APIs and Graph Ops

This page lists TensorFlow 2.x Python APIs and graph operators that are accelerated by AWS Neuron. The lists are not exhaustive. TensorFlow 2.x Python APIs or graph operators that are not listed here may still be accelerated if they are composed of accelerated primitives, or they will be executed on CPU without significant acceleration. The Neuron TensorFlow integration contains an automatic operator-device-placement mechanism that strives to maximize the execution efficiency of your deep learning models on AWS Machine Learning ASIC instances.

### Accelerated Python APIs

<table>
<thead>
<tr>
<th>Module</th>
<th>Accelerated Python API</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>tf</td>
<td>tf.abs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tf.add</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tf.add_n</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tf.broadcast_static_shape</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tf.cast</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tf.constant</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tf.convert_to_tensor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tf.cumsum</td>
<td>axis must be a compile-time constant.</td>
</tr>
<tr>
<td></td>
<td>tf.einsum</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tf.erf</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tf.exp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tf.identity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tf.matmul</td>
<td>Uses float16/bfloat16 matmul with float32 accumulation.</td>
</tr>
<tr>
<td></td>
<td>tf.maximum</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tf.minimum</td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Module</th>
<th>Accelerated Python API</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>tf.multiply</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.negative</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.range</td>
<td></td>
<td>start, limit and delta arguments must be compile-time</td>
</tr>
<tr>
<td></td>
<td></td>
<td>constants.</td>
</tr>
<tr>
<td>tf.realdiv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.reciprocal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.reduce_all</td>
<td>axis must be a compile-time constant.</td>
<td></td>
</tr>
<tr>
<td>tf.reduce_any</td>
<td>axis must be a compile-time constant.</td>
<td></td>
</tr>
<tr>
<td>tf.reduce_max</td>
<td>axis must be a compile-time constant.</td>
<td></td>
</tr>
<tr>
<td>tf.reduce_min</td>
<td>axis must be a compile-time constant.</td>
<td></td>
</tr>
<tr>
<td>tf.reduce_prod</td>
<td>axis must be a compile-time constant.</td>
<td></td>
</tr>
<tr>
<td>tf.reduce_sum</td>
<td>axis must be a compile-time constant.</td>
<td></td>
</tr>
<tr>
<td>tf.reshape</td>
<td>shape argument must be a compile-time constant.</td>
<td></td>
</tr>
<tr>
<td>tf.rsqrt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.scalar_mul</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.shape</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.shape_n</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.sigmoid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.size</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.slice</td>
<td>size must be a compile-time constant.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>In addition, either begin must be a compile-time constant</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or size must be non-negative.</td>
</tr>
<tr>
<td>tf.sqrt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.square</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.squared_difference</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.squeeze</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.stack</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.stop_gradient</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.strided_slice</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.tanh</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.tensordot</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.to_bfloat16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.to_float</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.truediv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.layers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>batch_normalization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.layers.dense</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.layers.flatten</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.nn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>batch_normalization</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>Module</th>
<th>Accelerated Python API</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>tf.nn.bias_add</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.nn.dropout</td>
<td></td>
<td>Always treated as tf.identity during inference.</td>
</tr>
<tr>
<td>tf.nn.fused_batch_norm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.nn.leaky_relu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.nn.relu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.nn.relu6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.nn.relu_layer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf.nn.softmax</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Accelerated graph operators**

Add
AddN
AddV2
BatchMatMul
BatchMatMulV2
BiasAdd
Cast
Const
Cumsum
Einsum
Erf
Exp
ExpandDims
FusedBatchNorm
FusedBatchNormV2
FusedBatchNormV3
Greater
Identity
LeakyRelu
MatMul
Max
Maximum
Minimum
Mean
Mul
Neg
Pack
RealDiv
Relu
Relu6
Reshape
Rsqrt
Sigmoid
Softmax
Split
SplitV
Sqrt
Square
SquaredDifference
Squeeze

(continues on next page)
StridedSlice
Sub
Sum
Tanh
Transpose
Unpack

The lists share many commonalities with Available TensorFlow Ops. Portions of this page are modifications based on work created and shared by Google and used according to terms described in the Creative Commons 4.0 Attribution License.

TensorFlow 2.x FAQ

- How do I get started with TensorFlow?
- What TensorFlow versions are supported by Neuron?
- What operators are supported?
- How do I compile my model?
- How do I deploy my model?
- Where can I find tutorials and examples?
- How to debug or profile my model?

How do I get started with TensorFlow?

The easiest entry point is the tutorials offered by the AWS Neuron team. For beginners, the HuggingFace Pipelines distilBERT Tutorial is a good place to start.

What TensorFlow versions are supported by Neuron?

The AWS Neuron provide well-tested tensorflow-neuron packages that work with a range of tensorflow official releases, as long as the version of tensorflow-neuron matches that of tensorflow. For example, you may install `tensorflow-neuron==2.3.3.1.0.9999.0` on top of `tensorflow==2.3.3` and expect them to work together.

Currently, tensorflow-neuron can work with tensorflow versions 2.1.4, 2.2.3, 2.3.3, 2.4.2, 2.5.0.

In a fresh Python environment, `pip install tensorflow-neuron` would bring in the highest version (2.5.0 as of 07/13/2021), which then pulls `tensorflow==2.5.0` into the current environment.

If you already have a particular version of tensorflow 2.x installed, then it is recommended to pay attention to the precise version of tensorflow-neuron and only install the desired one. For example, in an existing Python environment with `tensorflow==2.3.3` installed, you may install tensorflow-neuron by `pip install tensorflow-neuron==2.3.3`, which will reuse the existing tensorflow installation.
What operators are supported?

Due to fundamental backend design changes in the TensorFlow 2.x framework, the concept of “supported graph operators” is no longer well-defined. Please refer to *Accelerated Python APIs and graph operators* for a guide to the set of TensorFlow 2.x Python APIs and graph operators that can be accelerated by Neuron.

How do I compile my model?

It is achieved by a new public API called tfn.trace, which resembles the compilation API of AWS Neuron PyTorch integration. Programmatically, customers would be able to execute the following code.

```python
import tensorflow as tf
import tensorflow.neuron as tfn
...
model = tf.keras.Model(inputs=inputs, outputs=outputs)
model_neuron = tfn.trace(model, example_inputs)
model_neuron.save('./model_neuron_dir')
...
model_loaded = tf.saved_model.load('./model_dir')
predict_func = model_loaded['serving_default']
model_loaded_neuron = tfn.trace(predict_func, example_inputs2)
model_loaded_neuron.save('./model_loaded_neuron_dir')
...
```

How do I deploy my model?

Python tensorflow

Pre-compiled models can be saved and reloaded back into a Python environment using regular tensorflow model loading APIs, as long as tensorflow-neuron is installed.

```python
import tensorflow as tf
model = tf.keras.models.load_model('./model_loaded_neuron_dir')
example_inputs = ...
output = model(example_inputs)
```

tensorflow-serving

Pre-compiled models can be saved into SavedModel format via tensorflow SavedModel APIs.

```python
import tensorflow as tf
import tensorflow.neuron as tfn
...
model = tf.keras.Model(inputs=inputs, outputs=outputs)
model_neuron = tfn.trace(model, example_inputs)
tf.saved_model.save(model_neuron, './model_neuron_dir/1')
```
The generated SavedModel `./model_neuron_dir` can be loaded into tensorflow-model-server-neuron, which can be installed through apt or yum based on the type of the operating system. For example, on Ubuntu 18.04 LTS the following command installs and launches a tensorflow-model-server-neuron on a pre-compiled SavedModel.

```bash
sudo apt install tensorflow-model-server-neuron
# --model_base_path needs to be an absolute path
tensorflow_model_server_neuron --model_base_path=$(pwd)/model_neuron_dir
```

**Where can I find tutorials and examples?**

HuggingFace Pipelines distilBERT Tutorial is a good place to start.

**How to debug or profile my model?**

*AWS Neuron TensorBoard integration* provides visibility into what is happening inside of the Neuron runtime, and allows a more fine-grained (but also more hardware-aware) reasoning on where to improve the performance of machine learning applications.

### 3.2.3 TensorFlow 1.x

**TensorFlow-Neuron 1.x Compilation API**

The Neuron compilation API for TensorFlow 1.x enables compilation of saved model to an Inferentia target.

**Method**

```plaintext
tensorflow.neuron.saved_model.compile
```

**Description**

Within the graph or subgraph, the compile method selects and send Neuron-supported operations to Neuron-Compiler for compilation and saves the compiled artifacts in the graph. Uncompilable operations are kept as original operations for framework execution.

The compiled graph can be exported to saved model and served using TensorFlow Serving. Please see tensorflow-serving for more information about exporting to saved model and serving using TensorFlow Serving.

Options can be passed to Neuron compiler via the compile function. For example, the `"--neuroncore-pipeline-cores"` option directs Neuron compiler to compile each subgraph to fit in the specified number of NeuronCores. This number can be less than the total available NeuronCores on an Inf1 instance. See *Neuron compiler CLI Reference Guide* for more information about compiler options.
Arguments

- **model_dir**: The path of the original SavedModel.
- **new_model_dir**: The path to which the Neuron-optimized SavedModel will be stored.
- **batch_size**: (Optional) Positive integer representing batch size used in inference. The default value is 1.
- **model_shape_feed_dict**: (Optional) Dictionary \{str: list\} used for inferring tensor shapes. Keys should match model input names. Values are lists of positive integers representing model input tensor shapes.
- **model_feed_dict**: (Optional) Dictionary \{str: numpy.array\} used for inference. Useful for inferring tensor shapes. Keys should match model input names. Values are numpy arrays that can be fed as inputs to the SavedModel.
- **tags**: (Optional) Iterable of strings to identify the required MetaGraphDef. These should correspond to the tags used when saving the variables using the SavedModel save() API. Default is to use the first tag_set available in the SavedModel.
- **signature_def_key**: (Optional) String specifying the signature_def to use. Default is to use 'serving_default' or the first signature_def corresponding to tags.
- **minimum_segment_size**: (Optional) Integer indicating the minimum number of operations in an NeuronOp.
- **no_fuse_ops**: (Optional) None or iterable of strings (unordered) representing names of operations that are forcibly placed on CPU.
- **compiler_args**: (Optional) List of strings representing neuron-cc compiler arguments. Note that these arguments apply to all subgraphs generated by whitelist partitioning. For example, use compiler_args=['--neuroncore-pipeline-cores', '4'] to set number of NeuronCores per subgraph to 4. See Neuron compiler CLI Reference Guide for more information about compiler options.
- **compiler_workdir**: (Optional) String representing work directory of the neuron-cc compiler.

Returns

- Dictionary with operator counts before/after optimization.
- Operator count statistics are displayed to show original count, post-optimization count, and the number placed on Neuron runtime. For example:

```
INFO:tensorflow:Number of operations in TensorFlow session: 3978
INFO:tensorflow:Number of operations after tf.neuron optimizations: 555
INFO:tensorflow:Number of operations placed on Neuron runtime: 554
```
Example Usage

```python
import shutil
import tensorflow.neuron as tfn
saved_model_path = "<saved model path>"
compiled_saved_model_path = "<compiled saved model path>"
shutil.rmtree(compiled_saved_model_path, ignore_errors=True)
tfn.saved_model.compile(saved_model_path, compiled_saved_model_path)
```

TensorFlow 1.x Supported operators

To see a list of supported operators for TensorFlow 1.x, run the following command:

```
neuron-cc list-operators --framework TENSORFLOW
```

Neuron Compiler Release [1.7.3.0]

Added

- ArgMax
- ArgMin

Neuron Compiler Release [1.6.13.0]

No changes

Neuron Compiler Release [1.5.5.0]

No changes

Neuron Compiler Release [1.4.0.0]

No changes

Neuron Compiler Release [1.3.0.0]

Added

- Abs
- Cos
- DepthwiseConv2dNative
- Erf
- Rank
- Sin
- Size
Neuron Compiler Release [1.2.7.0]

No changes

Neuron Compiler Release [1.2.2.0]

Added

AdjustContrastv2
AdjustSaturation
BroadcastTo
Cholesky
Conv2DBackpropInput
Conv3D
CropAndResize
FloorDiv
HSVToRGB
InvertPermutation
L2Loss
Log1p
MatrixBandPart
MatrixDiag
MatrixSetDiag
MatrixTriangularSolve
MaxPool3D
MirrorPad
RGBToHSV
Range
SoftmaxCrossEntropyWithLogits
SquaredDifference
StopGradient
Unpack
UnsortedSegmentSum

Neuron Compiler Release [1.0.24045.0]

Added FloorDiv, Softplus, Unstack
Neuron Compiler Release [1.0.18001]
No changes

Neuron Compiler Release [1.0.16764]

Added:

- LogSoftmax
- Neg
- ResizeBilinear
- ResizeNearestNeighbor

Neuron Compiler Release [1.0.15275]

Added

- Neg

Removed

- Log

(was inadvertently advertised as supported)

Neuron Compiler Release [1.0.12696]
No changes

Neuron Compiler Release [1.0.9410]
No changes

Neuron Compiler Release [1.0.7878]
No changes

Neuron Compiler Release [1.0.6801]
No changes
Neuron Compiler Release [1.0.5939]

No changes

Neuron Compiler Release [1.0.5301]

No changes

Neuron Compiler Release [1.0.4680.0]

<table>
<thead>
<tr>
<th>Add</th>
<th>AddV2</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>AvgPool</td>
</tr>
<tr>
<td>BatchMatMul</td>
<td>BatchMatMulV2</td>
</tr>
<tr>
<td>BatchToSpaceND</td>
<td>BiasAdd</td>
</tr>
<tr>
<td>Cast</td>
<td>Ceil</td>
</tr>
<tr>
<td>Concat</td>
<td>ConcatV2</td>
</tr>
<tr>
<td>Const</td>
<td>Conv2D</td>
</tr>
<tr>
<td>Equal</td>
<td>Exp</td>
</tr>
<tr>
<td>ExpandDims</td>
<td>Fill</td>
</tr>
<tr>
<td>Floor</td>
<td>FusedBatchNorm</td>
</tr>
<tr>
<td>Greater</td>
<td>GreaterEqual</td>
</tr>
<tr>
<td>Identity</td>
<td>LRN</td>
</tr>
<tr>
<td>LeakyRelu</td>
<td>Less</td>
</tr>
<tr>
<td>LessEqual</td>
<td>Log</td>
</tr>
<tr>
<td>LogicalAnd</td>
<td>LogicalNot</td>
</tr>
<tr>
<td>LogicalOr</td>
<td>MatMul</td>
</tr>
<tr>
<td>Max</td>
<td>MaxPool</td>
</tr>
<tr>
<td>Maximum</td>
<td>Mean</td>
</tr>
<tr>
<td>Min</td>
<td>Minimum</td>
</tr>
<tr>
<td>Mul</td>
<td>NoOp</td>
</tr>
<tr>
<td>NotEqual</td>
<td>Pack</td>
</tr>
<tr>
<td>Pad</td>
<td>PadV2</td>
</tr>
</tbody>
</table>

(continues on next page)
### TensorFlow 1.x FAQ

- **How do I get started with TensorFlow?**
- **What TensorFlow versions are supported by Neuron?**
- **What operators are supported?**
- **How do I compile my model?**
- **How do I deploy my model?**
- **Where can I find tutorials and examples?**
- **How to debug or profile my model?**
How do I get started with TensorFlow?

The easiest entry point is the tutorials offered by the AWS Neuron team. For beginners, the ResNet50 tutorial is a good place to start.

What TensorFlow versions are supported by Neuron?

TensorFlow version 1.15.5

What operators are supported?

neuron-cc list-operators --framework TENSORFLOW provides a list of supported TensorFlow 1.x operators, and they are the operators that run on the machine learning accelerator. Note that operators not in this list are still expected to work with the supported operators in native TensorFlow together, although not accelerated by the hardware.

How do I compile my model?

tensorflow-neuron includes a public-facing compilation API called tfn.saved_model.compile. More can be found here TensorFlow-Neuron 1.x Compilation API.

How do I deploy my model?

Same way as deploying any tensorflow SavedModel. In Python TensorFlow, the easiest way is through the tf.contrib.predictor module. If a Python-free deployment is preferred for performance or some other reasons, tensorflow-serving is a great choice and the AWS Neuron team provides pre-built model server apt/yum packages named as tensorflow-model-server-neuron.

Where can I find tutorials and examples?

TensorFlow Tutorials is a great place to start with.

How to debug or profile my model?

At TensorFlow level, the v1 profiler is a great tool that provides operator-level breakdown of the inference execution time. Additionally, the AWS Neuron TensorBoard integration provides visibility into what is happening inside of the Neuron runtime, and allows a more fine-grained (but also more hardware-aware) reasoning on where to improve the performance of machine learning applications.
3.3 Release notes

3.3.1 What's New

TensorFlow 2.x

Tensorflow-Neuron 2.x Release Notes

- Known Issues and Limitations - updated 08/12/2021
- Tensorflow-Neuron 2.x release [2.0.4.0]
- Tensorflow-Neuron 2.x release [2.0.3.0]
- Tensorflow-Neuron 2.x release [1.6.8.0]

This document lists the release notes for the TensorFlow-Neuron 2.x packages.

**Known Issues and Limitations - updated 08/12/2021**

- Support on serialized TensorFlow 2.x custom operators is currently limited. Serializing some operators registered from tensorflow-text through TensorFlow Hub is going to cause failure in tensorflow.neuron.trace.

- Issue: When compiling large models, user might run out of memory and encounter this fatal error.

terminate called after throwing an instance of 'std::bad_alloc'

Solution: run compilation on a c5.4xlarge instance type or larger.

- Issue: When upgrading tensorflow-neuron with *pip install tensorflow-neuron --upgrade*, the following error message may appear, which is caused by *pip* version being too low.

Could not find a version that satisfies the requirement tensorflow<1.16.0,> →=1.15.0 (from tensorflow-neuron)

Solution: run a *pip install pip --upgrade* before upgrading tensorflow-neuron.

- Issue: Some Keras routines throws the following error:

AttributeError: 'str' object has no attribute 'decode'.

Solution: Please downgrade h5py by *pip install h5py<3*. This is caused by [https://github.com/tensorflow/tensorflow/issues/44467](https://github.com/tensorflow/tensorflow/issues/44467).
Tensorflow-Neuron 2.x release [2.0.4.0]

Date: 11/05/2021

• Updated Neuron Runtime (which is integrated within this package) to `libnrt 2.2.18.0` to fix a container issue that was preventing the use of containers when `/dev/neuron0` was not present. See details here *Neuron Runtime 2.x Release Notes*.

Tensorflow-Neuron 2.x release [2.0.3.0]

Date: 10/27/2021

**New in this release**

• TensorFlow Neuron 2.x now support Neuron Runtime 2.x (`libnrt.so` shared library) only.

**Important:**

– You must update to the latest Neuron Driver (`aws-neuron-dkms` version 2.1 or newer) for proper functionality of the new runtime library.

– Read *Introducing Neuron Runtime 2.x (libnrt.so)* application note that describes *why we are making this change* and *how this change will affect the Neuron SDK* in detail.

– Read *Migrate your application to Neuron Runtime 2.x (libnrt.so)* for detailed information of how to migrate your application.

• Updated Tensorflow 2.3.x from Tensorflow 2.3.3 to Tensorflow 2.3.4.

• Updated Tensorflow 2.4.x from Tensorflow 2.4.2 to Tensorflow 2.4.3.

• Updated Tensorflow 2.5.x from Tensorflow 2.5.0 to Tensorflow 2.5.1.

**Resolved Issues**

• Fix bug that can cause illegal compiler optimizations

• Fix bug that can cause dynamic-shape operators be placed on Neuron

Tensorflow-Neuron 2.x release [1.6.8.0]

Date: 08/12/2021
New in this release

• First release of TensorFlow 2.x integration, Neuron support now TensorFlow versions 2.1.4, 2.2.3, 2.3.3, 2.4.2, and 2.5.0.

• New public API tensorflow.neuron.trace: trace a TensorFlow 2.x keras.Model or a Python callable that can be decorated by tf.function, and return an AWS-Neuron-optimized keras.Model that can execute on AWS Machine Learning Accelerators.

Please note that TensorFlow 1.x SavedModel compilation API tensorflow.neuron.saved_model.compile is not supported in tensorflow-neuron 2.x. It continues to function in tensorflow-neuron 1.15.x.

• Included versions:
  – tensorflow-neuron-2.5.0.1.6.8.0
  – tensorflow-neuron-2.4.2.1.6.8.0
  – tensorflow-neuron-2.3.3.1.6.8.0
  – tensorflow-neuron-2.2.3.1.6.8.0
  – tensorflow-neuron-2.1.4.1.6.8.0

TensorFlow-Model-Server-Neuron 2.x Release Notes

This document lists the release notes for the TensorFlow-Model-Server-Neuron package.

TensorFlow Model Server Neuron 2.x release [2.0.4.0]

Date: 11/05/2021

• Updated Neuron Runtime (which is integrated within this package) to libnrt 2.2.18.0 to fix a container issue that was preventing the use of containers when /dev/neuron0 was not present. See details here Neuron Runtime 2.x Release Notes.

TensorFlow Model Server Neuron 2.x release [2.0.3.0]

Date: 10/27/2021
New in this release

- TensorFlow Model Server Neuron 2.x now support Neuron Runtime 2.x (libnrt.so shared library) only.

Important:
- You must update to the latest Neuron Driver (aws-neuron-dkms version 2.1 or newer) for proper functionality of the new runtime library.
- Read Introducing Neuron Runtime 2.x (libnrt.so) application note that describes why we are making this change and how this change will affect the Neuron SDK in detail.
- Read Migrate your application to Neuron Runtime 2.x (libnrt.so) for detailed information of how to migrate your application.

TensorFlow Model Server Neuron 2.x release [1.6.8.0]

Date: 08/12/2021

Summary

TensorFlow 2.x - tensorflow-model-server-neuron now support TensorFlow 2.x, tensorflow-model-server-neuron package versions 2.1.4, 2.2.2, 2.3.0, 2.4.1, and 2.5.1 support TensorFlow 2.x.

TensorFlow 1.x

Tensorflow-Neuron 1.x Release Notes

- Known Issues and Limitations - updated 08/12/2021
- Tensorflow-Neuron 1.x release [2.0.4.0]
- Tensorflow-Neuron 1.x release [2.0.3.0]
  - [1.15.5.1.5.1.0]
  - [1.15.5.1.4.0.0]
  - [1.15.5.1.3.3.0]
  - [1.15.5.1.2.9.0]
  - [1.15.5.1.2.8.0]
  - [1.15.5.1.2.2.0]
  - [1.15.4.1.1.3.0]
  - [1.15.4.1.0.2168.0]
  - [1.15.3.1.0.2043.0]
  - [1.15.3.1.0.1965.0]
This document lists the release notes for the TensorFlow-Neuron 1.x package.

**Known Issues and Limitations - updated 08/12/2021**

- Support on serialized TensorFlow 2.x custom operators is currently limited. Serializing some operators registered from tensorflow-text through TensorFlow Hub is going to cause failure in tensorflow.neuron.trace.

- Issue: When compiling large models, user might run out of memory and encounter this fatal error.

```
terminate called after throwing an instance of 'std::bad_alloc'
```

Solution: run compilation on a c5.4xlarge instance type or larger.

- Issue: When upgrading `tensorflow-neuron` with `pip install tensorflow-neuron --upgrade`, the following error message may appear, which is caused by pip version being too low.

```
Could not find a version that satisfies the requirement tensorflow<1.16.0,> <=1.15.0 (from tensorflow-neuron)
```

Solution: run `pip install pip --upgrade` before upgrading `tensorflow-neuron`.

- Issue: Some Keras routines throws the following error:

```
AttributeError: 'str' object has no attribute 'decode'.
```

Solution: Please downgrade `h5py` by `pip install 'h5py<3'`. This is caused by [https://github.com/tensorflow/tensorflow/issues/44467](https://github.com/tensorflow/tensorflow/issues/44467).
Tensorflow-Neuron 1.x release [2.0.4.0]

Date: 11/05/2021

• Updated Neuron Runtime (which is integrated within this package) to libnrt 2.2.18.0 to fix a container issue that was preventing the use of containers when /dev/neuron0 was not present. See details here Neuron Runtime 2.x Release Notes.

Tensorflow-Neuron 1.x release [2.0.3.0]

Date: 10/27/2021

New in this release

• TensorFlow Neuron 1.x now support Neuron Runtime 2.x (libnrt.so shared library) only.

Important:

– You must update to the latest Neuron Driver (aws-neuron-dkms version 2.1 or newer) for proper functionality of the new runtime library.

– Read Introducing Neuron Runtime 2.x (libnrt.so) application note that describes why are we making this change and how this change will affect the Neuron SDK in detail.

– Read Migrate your application to Neuron Runtime 2.x (libnrt.so) for detailed information of how to migrate your application.

Resolved Issues

• Fix neuron-cc argument handling bug when nothing can be compiled.

• Fixing the support of cast operators applied after constants, by Introducing support of constant-folding pass before Neuron auto-mixed-precision.

[1.15.5.1.5.1.0]

Date: 07/02/2021

New in this release

• Bug fixes regarding scalar inputs/outputs.

• Minor performance improvements when dynamic batch size is turned on or when model is small.
New in this release

- Reduce the amount of input/output data movement during inference.
- Improve parallelism for dynamic batch size inference by adopting a new sharding mechanism.
- Reduce the amount of host memory usage during inference.
- tfn.saved_model.compile now generates correct code when operator Split is used as output.
- tfn.saved_model.compile now properly reads input tensor shape information from SignatureDef proto.
- tfn.saved_model.compile now terminates properly when neuron-cc compiler argument is passed but there is no successful compilation.
- Fix bug on some wrong internal tensor names when neuron-cc compiler crashes.
- Other minor bug fixes.

[1.15.5.1.3.3.0]

Date: 05/01/2021

New in this release

1. Minor enhancements.

[1.15.5.1.2.9.0]

Date: 03/04/2021

New in this release

1. Minor enhancements.

[1.15.5.1.2.8.0]

Date: 02/24/2021
New in this release

1. Fix for CVE-2021-3177.

[1.15.5.1.2.2.0]

Date: 01/30/2021

New in this release

1. Bug fixes and internal refactor.
2. Bump tensorflow base package version to 1.15.5.
3. Introduced a new argument `convert_constants_to_variables` to the compilation API `tfn.saved_model.compile`. Setting it to `True` can address the issue of large constants consuming too much memory in the tensorflow runtime.

[1.15.4.1.1.3.0]

Date: 12/23/2020

New in this release

1. Improved logging during `tfn.saved_model.compile` to display neuron-cc compilation progress.
2. Small performance improvement in some edge cases by optimizing the NeuronCore-executable assignment mechanism.

[1.15.4.1.0.2168.0]

Date: 11/17/2020

New in this release

1. tensorflow-neuron is now a plugin package that can be used together with tensorflow>=1.15.0 built with `GLIBCXX_USE_CXX11_ABI=0`.
2. Improved logging during `tfn.saved_model.compile` to display neuron-cc logging file path, which is useful for tracking neuron-cc compilation progress.
3. Small performance improvement by utilizing shared memory more efficiently.
**New in this release**

1. tensorflow-neuron now automatically enables data parallel mode on four cores in one Inferentia. In `tensorflow-model-server-neuron`, most models can now fully utilize four cores automatically. In Python `tensorflow`, running threaded inference using `>=4` Python threads in the same `tensorflow` Session lead to full utilization of four cores.

2. tensorflow-neuron now tries to enable dynamic batch size automatically for a limited number of models, such as ResNet50.

3. Improved logging during `tfn.saved_model.compile` to display input/output information about subgraphs that are going to be compiled by `neuron-cc`.

**New in this release**

Various minor improvements.

**New in this release**

Various minor improvements.

This version contains a few bug fixes and user experience improvements.
Dependency change

1. Bump tensorflow base package version number to 1.15.3
2. Add tensorflow >= 1.15.0, < 1.16.0 as an installation dependency so that packages depending on tensorflow can be installed together with tensorflow-neuron without error

New Features

1. tensorflow-neuron now displays a summary of model performance when profiling is enabled by setting environment variable NEURON_PROFILE

Resolved Issues

1. Environment variable NEURON_PROFILE can now be set to a non-existing path which will be automatically created
2. Fixed a bug in tfn.saved_model.compile that causes compilation failure when dynamic_batch_size=True is specified on a SavedModel with unknown rank inputs.

[1.15.2.1.0.1796.0]

Date 6/11/2020

New in this release

This version contains a few bug fixes.

Major New Features

Resolved Issues

1. Fixed a bug related with device placement. Now models with device information hardcoded to GPU can be successfully compiled with tfn.saved_model.compile
2. Fixed a bug in tfn.saved_model.compile that causes models containing Reshape operators not functioning correctly when it is compiled with dynamic_batch_size=True
3. Fixed a bug in tfn.saved_model.compile that causes models containing Table related operators to initialize incorrectly after compilation.
Known Issues and limitations

[1.15.2.1.0.1572.0]

Date: 5/11/2020

New in this release

This version contains some bug fixes and new features.

Major New Features

• Tensorflow-Neuron is now built on TensorFlow 1.15.2 instead of TensorFlow 1.15.0

Resolved Issues

• Fixed a bug that caused Neuron runtime resources to not all be released when a tensorflow-neuron
process terminated with in-flight inferences
• Inference timeout value set at compile time is now correctly recognized at runtime

Known Issues and limitations

[1.15.0.1.0.1333.0]

Date: 3/26/2020

New in this release

Major New Features

• Improved performance between Tensorflow to Neuron runtime.

Resolved Issues

• Fixed a bug in Neuron runtime adaptor operator’s shape function when dynamic batch size inference
is enabled
• Framework method (tensorflow.neuron.saved-model.compile) improved handling of compiler time-
out termination by letting it clean up before exiting.
**Known Issues and limitations**

[1.15.0.1.0.1240.0]

Date: 2/27/2020

**New in this release**

**Major New Features**

- Enabled runtime memory optimizations by default to improve inference performance, specifically in cases with large input/output tensors
- `tfn.saved_model.compile` now displays warning message instead of “successfully compiled” if less than 30% of operators are mapped to Inferentia
- Improve error messages. Runtime failure error messages are now more descriptive and also provide instructions to restart neuron-rtd when necessary.

**Resolved Issues**

**Known Issues and Limitations**

- Issue: When compiling a large model, may encounter.

```
terminate called after throwing an instance of 'std::bad_alloc'
```

Solution: run compilation on c5.4xlarge instance type or larger.

**Other Notes**

[1.15.0.1.0.997.0]

Date: 1/27/2020

**New in this release**

**Major New Features**

- Added support for NCHW pooling operators in `tfn.saved_model.compile`.  

Resolved Issues

- Fixed GRPC transient status error issue.
- Fixed a graph partitioner issue with control inputs.

Known Issues and Limitations

- Issue: When compiling a large model, may encounter.

  ```
  terminate called after throwing an instance of 'std::bad_alloc'
  ```

  Solution: run compilation on c5.4xlarge instance type or larger.

Other Notes

[1.15.0.1.0.803.0]

Date: 12/20/2019

New in this release

Major New Features

Resolved Issues

- Improved handling of `tf.neuron.saved_model.compile` arguments

Known Issues and Limitations

Other Notes

[1.15.0.1.0.749.0]

Date: 12/1/2019

New in this release

Major New Features

Resolved Issues

- Fix race condition between model load and model unload when the process is killed
- Remove unnecessary GRPC calls when the process is killed
**Known Issues and Limitations**

- When compiling a large model, may encounter “terminate called after throwing an instance of 'std::bad_alloc'”. Solution: run compilation on c5.4xlarge instance type or larger.

- The pip package `wrapt` may have a conflicting version in some installations. This is seen when this error occurs:

```
ERROR: Cannot uninstall 'wrapt'. It is a distutils installed project and thus we cannot accurately determine which files belong to it which would lead to only a partial uninstall.
```

To solve this, you can update `wrapt` to the newer version:

```
python3 -m pip install wrapt --ignore-installed
python3 -m pip install tensorflow-neuron
```

Within a Conda environment:

```
conda update wrapt
conda update tensorflow-neuron
```

**Other Notes**

**[1.15.0.1.0.663.0]**

Date: 11/25/2019

**New in this release**

This version is available only in released DLAMI v26.0 and is based on TensorFlow version 1.15.0. Please *update* to latest version.

**Major New Features**

**Resolved Issues**

**Known Issues and Limits**

**Models Supported**

The following models have successfully run on neuron-inferentia systems

1. BERT_LARGE and BERT_BASE
2. Transformer
3. Resnet50 V1/V2
4. Inception-V2/V3/V4
Other Notes

- Python versions supported:
  - 3.5, 3.6, 3.7
- Linux distribution supported:
  - Ubuntu 18, Amazon Linux 2

TensorFlow-Model-Server-Neuron 1.x Release Notes

- TensorFlow Model Server Neuron 1.x release [2.0.4.0]
- TensorFlow Model Server Neuron 1.x release [2.0.3.0]
- [1.15.0.1.5.1.0]
- [1.15.0.1.4.0.0]
- [1.15.0.1.3.3.0]
- [1.15.0.1.2.9.0]
- [1.15.0.1.2.8.0]
- [1.15.0.1.2.2.0]
- [1.15.0.1.1.3.0]
- [1.15.0.1.0.2168.0]
- [1.15.0.1.0.2043.0]
- [1.15.0.1.0.1965.0]
- [1.15.0.1.0.1953.0]
- [1.15.0.1.0.1891.0]
- [1.15.0.1.0.1796.0]
- [1.15.0.1.0.1572.0]
- [1.15.0.1.0.1333.0]
- [1.15.0.1.0.1240.0]
- [1.15.0.1.0.997.0]
- [1.15.0.1.0.803.0]
- [1.15.0.1.0.749.0]
- [1.15.0.1.0.663.0]

This document lists the release notes for the TensorFlow-Model-Server-Neuron package.
TensorFlow Model Server Neuron 1.x release [2.0.4.0]

Date: 11/05/2021

- Updated Neuron Runtime (which is integrated within this package) to libnrt 2.2.18.0 to fix a container issue that was preventing the use of containers when /dev/neuron0 was not present. See details here Neuron Runtime 2.x Release Notes.

TensorFlow Model Server Neuron 1.x release [2.0.3.0]

Date: 10/27/2021

New in this release

- TensorFlow Model Server Neuron 1.x now support Neuron Runtime 2.x (libnrt.so shared library) only.

Important:

- You must update to the latest Neuron Driver (aws-neuron-dkms version 2.1 or newer) for proper functionality of the new runtime library.
- Read Introducing Neuron Runtime 2.x (libnrt.so) application note that describes why we are making this change and how this change will affect the Neuron SDK in detail.
- Read Migrate your application to Neuron Runtime 2.x (libnrt.so) for detailed information of how to migrate your application.

[1.15.0.1.5.1.0]

Date: 07/02/2021

Summary

No change. See tensorflow-neuron-release-notes for related TensorFlow-Neuron release notes.

[1.15.0.1.4.0.0]

Date: 05/24/2021
Summary

1. Remove SIGINT/SIGTERM handler and rely on mechanisms provided by Neuron runtime for resource cleanup.
2. Uncap protobuf size limit.

[1.15.0.1.3.3.0]
Date: 05/01/2021

Summary
No change. See tensorflow-neuron-release-notes for related TensorFlow-Neuron release notes.

[1.15.0.1.2.9.0]
Date: 03/04/2021

Summary
No change. See tensorflow-neuron-release-notes for related TensorFlow-Neuron release notes.

[1.15.0.1.2.8.0]
Date: 02/24/2021

Summary
No change. See tensorflow-neuron-release-notes for related TensorFlow-Neuron release notes.

[1.15.0.1.2.2.0]
Date: 01/30/2021

Summary
No change. See tensorflow-neuron-release-notes for related TensorFlow-Neuron release notes.
[1.15.0.1.3.0]

Date: 12/23/2020

Summary

No change. See tensorflow-neuron-release-notes for related TensorFlow-Neuron release notes.

[1.15.0.1.0.2168.0]

Date: 11/17/2020

Summary

No change. See tensorflow-neuron-release-notes for related TensorFlow-Neuron release notes.

[1.15.0.1.0.2043.0]

Date: 09/22/2020

Summary

No change. See tensorflow-neuron-release-notes for related TensorFlow-Neuron release notes.

[1.15.0.1.0.1965.0]

Date: 08/08/2020

Summary

No change. See tensorflow-neuron-release-notes for related TensorFlow-Neuron release notes.

[1.15.0.1.0.1953.0]

Date: 08/05/2020
**Summary**

No change. See tensorflow-neuron-release-notes for related TensorFlow-Neuron release notes.

**[1.15.0.1.0.1891.0]**

Date: 07/16/2020

**Summary**

No change. See tensorflow-neuron-release-notes for related TensorFlow-Neuron release notes.

**[1.15.0.1.0.1796.0]**

Date 6/11/2020

**Summary**

No change. See tensorflow-neuron-release-notes for related TensorFlow-Neuron release notes.

**[1.15.0.1.0.1572.0]**

Date 5/11/2020

**Summary**

No change. See tensorflow-neuron-release-notes for related TensorFlow-Neuron release notes.

**[1.15.0.1.0.1333.0]**

Date 3/26/2020

**Summary**

No change. See tensorflow-neuron-release-notes for related TensorFlow-Neuron release notes.
[1.15.0.1.0.1240.0]

Date 2/27/2020

Summary

No change. See tensorflow-neuron-release-notes for related TensorFlow-Neuron release notes.

[1.15.0.1.0.997.0]

Date 1/27/2019

Summary

No change. See tensorflow-neuron-release-notes for related TensorFlow-Neuron release notes.

[1.15.0.1.0.803.0]

Date 12/20/2019

Summary

No change. See tensorflow-neuron-release-notes for related TensorFlow-Neuron release notes.

[1.15.0.1.0.749.0]

Date 12/1/2019

Summary

No change. See tensorflow-neuron-release-notes for related TensorFlow-Neuron release notes.

[1.15.0.1.0.663.0]

Date 11/29/2019
Summary

This version is available only in released DLAMI v26.0. See TensorFlow-Neuron Release Notes. Please update to latest version.

Common

**Warning:** Starting with Neuron 1.14.0, Neuron Conda packages in Deep Learning AMI are no longer supported, for more information see blog announcing the end of support for Neuron conda packages.

Conda-TensorFlow Release Notes

This document lists the release notes for the Neuron Conda-TensorFlow package.

Table of Contents

- [1.15.5.1.3.3.0]
- [1.15.5.1.2.9.0]
- [1.15.5.1.2.8.0]
- [1.15.5.1.2.2.0]
- [1.15.4.1.1.3.0]
- [1.15.4.1.0.2168.0]
- [1.15.3.1.0.2043.0_2.0.894.0]
- [1.15.3.1.0.1965.0_2.0.778.0]
- [1.15.3.1.0.1953.0_2.0.769.0]
- [1.15.3.1.0.1891.0-2.0.706.0]
- [1.15.2.1.0.1782.0-2.0.593.0]
- [1.15.2.1.0.1572.0-2.0.329.0]
- [1.15.0.1.0.1333.0-2.0.63.0]
- [1.15.0.1.0.1240.0-1.0.918.0]
- [1.15.0.1.0.997.0-1.0.733.0]
- [1.15.0.1.0.803.0-1.0.611.0]
- [1.15.0.1.0.749.0-1.0.474.0]
- [1.15.0.1.0.663.0-1.0.298.0]
Included Neuron Packages

neuron_cc-1.3.7.0
tensorboard_plugin_neuron-2.0.29.0
tensorflow_neuron-1.15.5.1.3.3.0

Date: 4/30/2021

Included Neuron Packages

neuron_cc-1.2.7.0
tensorboard_neuron-1.15.0.1.2.6.0
tensorflow_neuron-1.15.5.1.2.9.0

Date: 3/4/2021

Included Neuron Packages

neuron_cc-1.2.7.0
tensorboard_neuron-1.15.0.1.2.6.0
tensorflow_neuron-1.15.5.1.2.8.0

Date: 2/24/2021

Included Neuron Packages

neuron_cc-1.2.2.0
tensorboard_neuron-1.15.0.1.2.0.0
tensorflow_neuron-1.15.5.1.2.2.0

Date: 1/30/2021
[1.15.4.1.3.0]

Date: 12/23/2020

Included Neuron Packages

neuron_cc-1.1.7.0
tensorboard_neuron-1.15.0.1.1.1.0
tensorflow_neuron-1.15.4.1.1.3.0

[1.15.4.1.0.2168.0]

Date: 11/17/2020

Included Neuron Packages

neuron_cc-1.0.24045.0
tensorboard_neuron-1.15.0.1.0.615.0
tensorflow_neuron-1.15.4.1.0.2168.0

[1.15.3.1.0.2043.0_2.0.894.0]

Date: 09/22/2020

Included Neuron Packages

neuron_cc-1.0.20600.0
tensorboard_neuron-1.15.0.1.0.600.0
tensorflow_neuron-1.15.3.1.0.2043.0

Known Issues

When running TensorFlow script in latest TensorFlow-Neuron conda environment, you may see errors “AttributeError: module ‘numpy’ has no attribute ‘integer’” and “ModuleNotFoundError: No module named ‘numpy.core._multiarray_umath’”. This is due to older version of numpy. Please update numpy to version 1.18 using the command “conda update numpy”.
[1.15.3.1.0.1965.0_2.0.778.0]

Date: 08/08/2020

**Included Neuron Packages**

neuron_cc-1.0.18001.0
tensorboard_neuron-1.15.0.1.0.570.0
tensorflow_neuron-1.15.3.1.0.1965.0

[1.15.3.1.0.1953.0_2.0.769.0]

Date: 08/05/2020

**Included Neuron Packages**

neuron_cc-1.0.17937.0
tensorboard_neuron-1.15.0.1.0.513.0
tensorflow_neuron-1.15.3.1.0.1889.0

[1.15.3.1.0.1891.0-2.0.706.0]

Date: 07/16/2020

Now supporting Python 3.7 Conda packages in addition to Python 3.6 Conda packages.

**Included Neuron Packages**

neuron_cc-1.0.16861.0
tensorboard_neuron-1.15.0.1.0.513.0
tensorflow_neuron-1.15.3.1.0.1891.0

[1.15.2.1.0.1782.0-2.0.593.0]

Date: 06/11/2020
**Included Neuron Packages**

neuron_cc-1.0.15275.0  
tensorboard_neuron-1.15.0.1.0.491.0  
tensorflow_neuron-1.15.0.1.0.1796.0

**[1.15.2.1.0.1572.0-2.0.329.0]**

Date 5/11/2020

**Included Neuron Packages**

neuron_cc-1.0.12696.0  
tensorboard_neuron-1.15.0.1.0.466.0  
tensorflow_neuron-1.15.2.1.0.1572.0

**[1.15.0.1.0.1333.0-2.0.63.0]**

Date 3/26/2020

**Included Neuron Packages**

neuron_cc-1.0.9410.0  
tensorflow_neuron-1.15.0.1.0.1333.0  
tensorboard_neuron-1.15.0.1.0.392.0

**[1.15.0.1.0.1240.0-1.0.918.0]**

Date 2/27/2020

**Included Neuron Packages**

neuron_cc-1.0.7668.0  
tensorflow_neuron-1.15.0.1.0.1240.0  
tensorboard_neuron-1.15.0.1.0.366.0
Included Neuron Packages

neuron-cc-1.0.6801.0
tensorflow-neuron-1.15.0.1.0.997.0
tensorboard-neuron-1.15.0.1.0.315.0

Included Neuron Packages

neuron-cc-1.0.5939.0
tensorflow-neuron-1.15.0.1.0.803.0
tensorboard-neuron-1.15.0.1.0.315.0

Included Neuron Packages

neuron-cc-1.0.5301.0
tensorflow-neuron-1.15.0.1.0.749.0
tensorboard-neuron-1.15.0.1.0.306.0

Known Issues and Limitations

This version is only available from the release DLAMI v26.0. Please see Known Issues to latest version.
Included Neuron Packages

neuron-cc-1.0.4680.0
tensorflow-neuron-1.15.0.1.0.663.0
tensorboard-neuron-1.15.0.1.0.280.0

Known Issues and Limitations

Please update to the latest conda package release.

```
source activate <conda environment>
conda update tensorflow-neuron
```

In TensorFlow-Neuron conda environment (aws_neuron_tensorflow_p36) of DLAMI v26.0, the installed numpy version prevents update to latest conda package version. Please do “conda install numpy=1.17.2 --yes --quiet” before “conda update tensorflow-neuron”. (See DLAMI with Neuron Release Notes).

```
source activate aws_neuron_tensorflow_p36
conda install numpy=1.17.2 --yes --quiet
conda update tensorflow-neuron
```
Neuron is integrated into MXNet, and provides you with a familiar environment to run inference using Inferentia based instances.

### 4.1 Installation Guide

#### 4.1.1 Install Neuron MXNet

**Note:**
- Instructions in this page only apply to setting up Neuron components on Linux host running Ubuntu or Amazon Linux AMI.
- For an example of how to install Neuron components in a container, see Docker environment setup and our Containers documentation for more details.

- **Develop on AWS ML accelerator instance**
- **Compile on compute instance**
- **Deploy on AWS ML accelerator instance**

**Develop on AWS ML accelerator instance**

The simplest environment setup for model development installs all Neuron SDK components directly on an AWS ML accelerator instance: the Neuron framework extensions, compiler, runtime, and tools. This will allow you to compile, execute, and performance tune your model, all in the same instance. This is the recommended workflow when first starting to work with Neuron device or when optimizing a model.

**Important:**

**For successful installation or update to Neuron 1.16.0 and newer from previous releases:**
- Stop Neuron Runtime 1.x daemon (neuron-rtd) by running: `sudo systemctl stop neuron-rtd`
- Uninstall neuron-rtd by running: `sudo apt remove aws-neuron-runtime` or `sudo yum remove aws-neuron-runtime`
• Install or upgrade to latest Neuron driver (aws-neuron-dkms) by following the “Setup Guide” instructions.

• Visit *Introducing Neuron Runtime 2.x (libnrt.so)* for more information.

MXNet 1.8.0
MXNet 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Configure Linux for Neuron repository updates
/etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$uname -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
```

(continues on next page)
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name mxnet_venv --display-name "Python (Neuron MXNet)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-\py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron neuron-cc

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

(continues on next page)
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  
  → 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$\{\text{uname -r}\} kernel-headers-$\{\text{uname -r}\} -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo yum install aws-neuron-tools -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++ python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name mxnet_venv --display-name "Python (Neuron MXNet)"

pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron neuron-cc

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
```

(continues on next page)
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-\py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron neuron-cc

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo yum install aws-neuron-tools -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
```

(continues on next page)
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron neuron-cc

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Configure Linux for Neuron repository updates
/etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y
```

# Warning: If Linux kernel is updated as a result of OS package update

(continues on next page)
Neuron driver (aws-neuron-dkms) should be re-installed after reboot.

# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install --python-version 3 pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name mxnet_venv --display-name "Python (Neuron MXNet)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.* neuron-cc

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd'
To install or update to Neuron versions 1.16.0 and newer from previous releases:

- DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

Install OS headers

```
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y
```

Install Neuron Driver

```
sudo yum install aws-neuron-dkms -y
```

Warning: If Linux kernel is updated as a result of OS package update, Neuron driver (aws-neuron-dkms) should be re-installed after reboot

Install Neuron Tools

```
sudo yum install aws-neuron-tools -y
```

Export PATH to point to Neuron bin directory

```
export PATH=/opt/aws/neuron/bin:$PATH
```

Install Python venv and activate Python virtual environment to install Neuron pip packages.

```
sudo yum install -y python3 gcc-c++ python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
```

Install Jupyter notebook kernel

```
pip install ipykernel
python -m ipykernel install --user --name mxnet_venv --display-name "Python (Neuron MXNet)"
pip install jupyter notebook
```

Install environment_kernels

```
pip install environment_kernels
```

Set Pip repository to point to the Neuron repository

```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

Install Neuron MXNet

```
pip install mxnet_neuron==1.5.1.* neuron-cc
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not include latest Neuron versions.

To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation.

Before installing or updating aws-neuron-dkms:
- Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  `sudo systemctl stop neuron-rtd`

To install or update to Neuron versions 1.16.0 and newer from previous releases:
- DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver.

Install OS headers:
```
sudo apt-get install linux-headers-$\{uname -r\} -y
```

Install Neuron Driver:
```
sudo apt-get install aws-neuron-dkms -y
```

Warning: If Linux kernel is updated as a result of OS package update or reboot, Neuron driver (aws-neuron-dkms) should be re-installed after.

Activate MXNet:
```
source activate aws_neuron_mxnet_p36
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate MXNet
source activate aws_neuron_mxnet_p36

## Compile on compute instance

If model compilation occurs outside the model deployment environment, you can install only the Neuron framework extensions and the compiler on any compute instance. This setup is helpful when compiling large complex models that require large amount of memory or during a CICD process where models are compiled in a separate step, prior to deployment.

- MXNet 1.8.0
- MXNet 1.5.1
- Ubuntu AMI
- Amazon Linux AMI
- Ubuntu DLAMI
- Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0--py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron neuron-cc

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0--py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron neuron-cc

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0--py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron neuron-cc

Note: For a successful installation or update, execute each line of the instructions below separately or...
copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-
˓→py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron neuron-cc

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.* neuron-cc

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.* neuron-cc

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation

# Activate MXNet
source activate aws_neuron_mxnet_p36
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation

# Activate MXNet
source activate aws_neuron_mxnet_p36
```

### Deploy on AWS ML accelerator instance

During deployment it can be beneficial to reduce the number of components installed in the system. For use-cases where only inference is necessary (compilation is already complete), only the framework and runtime should be installed.

**Important:**

**For successful installation or update to Neuron 1.16.0 and newer from previous releases:**

- Stop Neuron Runtime 1.x daemon (neuron-rtd) by running: `sudo systemctl stop neuron-rtd`
- Uninstall neuron-rtd by running: `sudo apt remove aws-neuron-runtime` or `sudo yum remove aws-neuron-runtime`
- Install or upgrade to latest Neuron driver (aws-neuron-dkms) by following the “Setup Guide” instructions.
- Visit *Introducing Neuron Runtime 2.x (libnrt.so)* for more information.

MXNet 1.8.0
MXNet 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -
# Update OS packages
sudo apt-get update -y
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
# Install OS headers
sudo apt-get install linux-headers-$\{(uname -r)\} -y
# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
export PATH=/opt/aws/neuron/bin:$PATH
# Install Python venv and activate Python virtual environment to install
```
(continues on next page)
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-
py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-
AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous
# releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
#   or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++ python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/awsmx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

#############################################################################

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

#############################################################################

export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

#############################################################################

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

#############################################################################

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

(continues on next page)
# Install OS headers
sudo yum install kernel-devel-$\{(uname -r)\} kernel-headers-$\{(uname -r)\} -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after_
# reboots

export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-
~py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-
NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
```
sudo apt-get install linux-headers-$\{uname -r\} -y
```

# Install Neuron Driver
```
sudo apt-get install aws-neuron-dkms -y
```

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver ('aws-neuron-dkms') should be re-installed after reboot

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
```
sudo apt-get install -y python3-venv g++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip
```

# Set Pip repository to point to the Neuron repository
```
config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron MXNet
```
pip install mxnet_neuron==1.5.1.*
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
```
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF

sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB
```

(continues on next page)
# Update OS packages

```bash
sudo yum update -y
```

# Before installing or updating `aws-neuron-dkms`

```bash
# Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
'sudo systemctl stop neuron-rtd'
```

# To install or update to Neuron versions 1.16.0 and newer from previous releases:

```bash
# DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
```

# Install OS headers

```bash
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
```

# Install Neuron Driver

```bash
sudo yum install aws-neuron-dkms -y
```

# Warning: If Linux kernel is updated as a result of OS package update

```bash
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```

```bash
export PATH=/opt/aws/neuron/bin:$PATH
```

# Install Python venv and activate Python virtual environment to install

```bash
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip
```

# Set Pip repository to point to the Neuron repository

```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

```bash
# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.*
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate MXNet
source activate aws_neuron_mxnet_p36

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
4.1.2 Update to latest Neuron MXNet

Note:

- Instructions in this page only apply to setting up Neuron components on Linux host running Ubuntu or Amazon Linux AMI.
- For an example of how to install Neuron components in a container, see Docker environment setup and our Containers documentation for more details.
Develop on AWS ML accelerator instance

The simplest environment setup for model development installs all Neuron SDK components directly on an AWS ML accelerator instance: the Neuron framework extensions, compiler, runtime, and tools. This will allow you to compile, execute, and performance tune your model, all in the same instance. This is the recommended workflow when first starting to work with Neuron device or when optimizing a model.

Important:

For successful installation or update to Neuron 1.16.0 and newer from previous releases:

- Stop Neuron Runtime 1.x daemon (neuron-rtd) by running: `sudo systemctl stop neuron-rtd`
- Uninstall neuron-rtd by running: `sudo apt remove aws-neuron-runtime` or `sudo yum remove aws-neuron-runtime`
- Install or upgrade to latest Neuron driver (aws-neuron-dkms) by following the “Setup Guide” instructions.
- Visit Introducing Neuron Runtime 2.x (libnrt.so) for more information.

MXNet 1.8.0
MXNet 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
# - or upgrade to latest Neuron driver
```

(continues on next page)
# Update OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Activate Python virtual environment where Neuron pip packages were installed
source mxnet_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron MXNet
pip install --upgrade mx_neuron neuron-cc

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo yum update aws-neuron-tools -y

# Activate Python virtual environment where Neuron pip packages were installed
source mxnet_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron MXNet
pip install --upgrade mx_neuron neuron-cc

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y
# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
#   or upgrade to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

(continues on next page)
Adapted from previous page)

------

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

------

# Update Neuron Tools
sudo yum update aws-neuron-tools -y

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron MXNet
pip install --upgrade mx_neuron neuron-cc

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

------

# Update OS packages
sudo apt-get update -y

------

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#       'sudo systemctl stop neuron-rtd'

------

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
#    or upgrade to latest Neuron driver

------

# Update OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Update Neuron Driver

(continues on next page)
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Activate Python virtual environment where Neuron pip packages were installed
source mxnet_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron MXNet
pip install --upgrade mxnet_neuron==1.5.1.* neuron-cc

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or update to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update Neuron Tools
sudo yum update aws-neuron-tools -y

# Activate Python virtual environment where Neuron pip packages were installed
source mxnet_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron MXNet
pip install --upgrade mxnet_neuron==1.5.1.* neuron-cc

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

(continues on next page)
# Update Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron MXNet
pip install --upgrade mxnet_neuron==1.5.1.* neuron-cc

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

**Warning:** If Linux kernel is updated as a result of OS package update
# - Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Update Neuron Tools
sudo yum update aws-neuron-tools -y

(continues on next page)
# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron MXNet
pip install --upgrade mxnet_neuron==1.5.1.* neuron-cc

---

**Compile on compute instance**

If model compilation occurs outside the model deployment environment, you can install only the Neuron framework extensions and the compiler on any compute instance. This setup is helpful when compiling large complex models that require large amount of memory or during a CICD process where models are compiled in a separate step, prior to deployment.

- **MXNet 1.8.0**
- **MXNet 1.5.1**
- **Ubuntu AMI**
- **Amazon Linux AMI**
- **Ubuntu DLAMI**
- **Amazon Linux DLAMI**

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Activate Python virtual environment where Neuron pip packages were installed
source mxnet_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron MXNet
pip install --upgrade mx_neuron neuron-cc

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Update Neuron MXNet
pip install --upgrade mx_neuron neuron-cc

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron MXNet
pip install --upgrade mx_neuron neuron-cc
```

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Activate Python virtual environment where Neuron pip packages were installed
source mxnet_venv/bin/activate

# Set Pip repository to point to the Neuron repository
```

(continues on next page)
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron MXNet
pip install --upgrade mxnet_neuron==1.5.1.* neuron-cc

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Activate Python virtual environment where Neuron pip packages were installed
source mxnet_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron MXNet
pip install --upgrade mxnet_neuron==1.5.1.* neuron-cc

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron MXNet
pip install --upgrade mxnet_neuron==1.5.1.* neuron-cc

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron MXNet
pip install --upgrade mxnet_neuron==1.5.1.* neuron-cc
Deploy on AWS ML accelerator instance

During deployment it can be beneficial to reduce the number of components installed in the system. For use-cases where only inference is necessary ( compilation is already complete), only the framework and runtime should be installed.

**Important:**

**For successful installation or update to Neuron 1.16.0 and newer from previous releases:**

- Stop Neuron Runtime 1.x daemon (neuron-rtd) by running: `sudo systemctl stop neuron-rtd`
- Uninstall neuron-rtd by running: `sudo apt remove aws-neuron-runtime` or `sudo yum remove aws-neuron-runtime`
- Install or upgrade to latest Neuron driver (aws-neuron-dkms) by following the “Setup Guide” instructions.
- Visit *Introducing Neuron Runtime 2.x (libnrt.so)* for more information.

MXNet 1.8.0
MXNet 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
# or upgrade to latest Neuron driver

# Update OS headers
sudo apt-get install linux-headers-$(uname -r) -y
```

(continues on next page)
# Update Neuron Driver

```
sudo apt-get install aws-neuron-dkms -y
```

---

Warning: If Linux kernel is updated as a result of OS package update, Neuron driver (aws-neuron-dkms) should be re-installed after reboot.

---

# Activate Python virtual environment where Neuron pip packages were installed

```
source mxnet_venv/bin/activate
```

# Set Pip repository to point to the Neuron repository

```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Update Neuron MXNet

```
pip install --upgrade mx_neuron
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Update OS packages
sudo yum update -y
```

---

Before installing or updating aws-neuron-dkms:

- Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:

```
sudo systemctl stop neuron-rtd
```

---

# To install or update to Neuron versions 1.16.0 and newer from previous releases:

- DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver.

---

# Update OS headers

```
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
```

# Update Neuron Driver

```
sudo yum update aws-neuron-dkms -y
```

---

(continues on next page)
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate Python virtual environment where Neuron pip packages were installed
source mxnet_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron MXNet
pip install --upgrade mx_neuron

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
"sudo systemctl stop neuron-rtd"

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron MXNet
pip install --upgrade mx_neuron

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate MXNet
source activate aws_neuron_mxnet_p36
# Set Pip repository to point to the Neuron repository
`pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com`

# Update Neuron MXNet
`pip install --upgrade mx_neuron`

**Ubuntu AMI**

**Amazon Linux AMI**

**Ubuntu DLAMI**

**Amazon Linux DLAMI**

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Update OS packages
`sudo apt-get update -y`

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
`sudo apt-get install linux-headers-$\{uname -r\} -y`

# Update Neuron Driver
`sudo apt-get install aws-neuron-dkms -y`

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate Python virtual environment where Neuron pip packages were installed
`source mxnet_venv/bin/activate`
```

(continues on next page)
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron MXNet
pip install --upgrade mxnet_neuron==1.5.1.*

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Update Neuron Driver
sudo yum update aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate Python virtual environment where Neuron pip packages were installed
source mxnet_venv/bin/activate

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron MXNet
pip install --upgrade mxnet_neuron==1.5.1.*
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Update OS headers
sudo apt-get install linux-headers-$\{(uname -r)\} -y

# Update Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Update Neuron MXNet
pip install --upgrade mxnet_neuron==1.5.1.*
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Update OS packages
sudo yum update -y
```

(continues on next page)
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  → 'sudo systemctl stop neuron-rtd'
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
# Update OS headers
  sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y
# Update Neuron Driver
  sudo yum update aws-neuron-dkms -y
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
# Activate MXNet
  source activate aws_neuron_mxnet_p36
# Set Pip repository to point to the Neuron repository
  pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
# Update Neuron MXNet
  pip install --upgrade mxnet_neuron==1.5.1.*

4.1.3 Install previous Neuron MXNet releases

Note:

- Instructions in this page only apply to setting up Neuron components on Linux host running Ubuntu or Amazon Linux AMI.
- For an example of how to install Neuron components in a container, see Docker environment setup and our Containers documentation for more details.
Install Neuron MXNet (Neuron 1.15.2)

- Develop on AWS ML accelerator instance
- Compile on compute instance
- Deploy on AWS ML accelerator instance

Develop on AWS ML accelerator instance

The simplest environment setup for model development installs all Neuron SDK components directly on an AWS ML accelerator instance: the Neuron framework extensions, compiler, runtime, and tools. This will allow you to compile, execute, and performance tune your model, all in the same instance. This is the recommended workflow when first starting to work with Neuron device or when optimizing a model.

MXNet 1.8.0
MXNet 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Configure Linux for Neuron repository updates
  . /etc/os-release
  sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
  deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
  EOF
  wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
  sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
  sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
  # If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
```

(continues on next page)
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

###
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
###

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++ python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name mxnet_venv --display-name "Python (Neuron MXNet)"

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0 neuron-cc==1.6.13.0

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
EOF

baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF

sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

echo "PATH=/opt/aws/neuron/bin:$PATH" >> /etc/profile

echo "export PATH=/opt/aws/neuron/bin:$PATH" >> /etc/profile

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name mxnet_venv --display-name "Python-(Neuron MXNet)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0 neuron-cc==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-\py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0 neuron-cc==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

(continues on next page)
# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-\py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0 neuron-cc==1.6.13.0

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-\NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$ (uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '-allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name mxnet_venv --display-name "Python (Neuron MXNet)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0 neuron-cc==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-
AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#  `sudo systemctl stop neuron-rtd`

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version , please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version , please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version , please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name mxnet_venv --display-name "Python
(Neuron MXNet)"
pip install jupyter notebook
pip install environment_kernels
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0 neuron-cc==1.6.13.0

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  # 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
# Install Neuron MXNet

```bash
pip install mxnet_neuron==1.5.1.1.6.5.0 neuron-cc==1.6.13.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# => 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0 neuron-cc==1.6.13.0
```
Compile on compute instance

If model compilation occurs outside the model deployment environment, you can install only the Neuron framework extensions and the compiler on any compute instance. This setup is helpful when compiling large complex models that require large amount of memory or during a CICD process where models are compiled in a separate step, prior to deployment.

MXNet 1.8.0
MXNet 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
  sudo apt-get install -y python3-venv g++
  python3 -m venv mxnet_venv
  source mxnet_venv/bin/activate
  pip install -U pip

# Set Pip repository to point to the Neuron repository
  pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
  wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-
      →py2.py3-none-manylinux2014_x86_64.whl
  pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
  pip install mx_neuron==1.8.0.1.3.4.0 neuron-cc==1.6.13.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
  sudo yum install -y python3 gcc-c++
  python3 -m venv mxnet_venv
  source mxnet_venv/bin/activate
  pip install -U pip

# Set Pip repository to point to the Neuron repository
  pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
  wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-
      →py2.py3-none-manylinux2014_x86_64.whl
```

(continues on next page)
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0 neuron-cc==1.6.13.0
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0 neuron-cc==1.6.13.0
```

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Install Python venv and activate Python virtual environment to install # Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0 neuron-cc==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install # Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0 neuron-cc==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0 neuron-cc==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0 neuron-cc==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
(continues on next page)
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.6.5.0 neuron-cc==1.6.13.0

## Deploy on AWS ML accelerator instance

During deployment it can be beneficial to reduce the number of components installed in the system. For use-cases where only inference is necessary (compilation is already complete), only the framework and runtime should be installed.

MXNet 1.8.0
MXNet 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$ (uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y
```

(continues on next page)
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
export PATH=/opt/aws/neuron/bin:$PATH
# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y
export PATH=/opt/aws/neuron/bin:$PATH
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
#Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-
py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-
AWS-NEURON.PUB
# Update OS packages
sudo yum update -y
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
#   'sudo systemctl stop neutron-rtd'
(continues on next page)
# Install OS headers
```
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
```

# Install Neuron Driver
```
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```
```
```

# Install Neuron Runtime server
```
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y
```

```
export PATH=/opt/aws/neuron/bin:$PATH
```

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
```
sudo yum install -y python3 gcc-c++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
```

```
pip install -U pip
```

```
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron MXNet
```
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-
py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
```

```
pip install mx_neuron==1.8.0.1.3.4.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
```
sudo apt-get update -y
```

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

(continues on next page)
# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-\npy2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
EOF
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF

sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-
AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$\{(uname -r)\} kernel-headers-$\{(uname -r)\} -y

# Install Neuron Driver
# If you are downgrading from newer version , please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version , please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++ python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0

**Note:** For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

```bash
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# `reboot`

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`
```
# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0

**Install Neuron MXNet (Neuron 1.15.1)**

- Develop on AWS ML accelerator instance
- Compile on compute instance
- Deploy on AWS ML accelerator instance
Develop on AWS ML accelerator instance

The simplest environment setup for model development installs all Neuron SDK components directly on an AWS ML accelerator instance: the Neuron framework extensions, compiler, runtime, and tools. This will allow you to compile, execute, and performance tune your model, all in the same instance. This is the recommended workflow when first starting to work with Neuron device or when optimizing a model.

MXNet 1.8.0
MXNet 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -
# Update OS packages
sudo apt-get update -y
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y
# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```

(continues on next page)
# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
→ option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
→ option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name mxnet_venv --display-name "Python
→ (Neuron MXNet)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-
→py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0 neuron-cc==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-
→AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

(continues on next page)
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   -> 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install --y python3 gcc-c++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name mxnet_venv --display-name "Python (Neuron MXNet)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-\--py3.none-manylinux2014_x86_64.whl
pip3 install aws_mx_cul10-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0 neuron-cc==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   →'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

(continues on next page)
# Install Neuron MXNet

```
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-
→py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0 neuron-cc==1.6.13.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Install NEURON Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Install NEURON Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install NEURON Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

# Activate MXNet
source activate aws_neuron_mxnet_p36
```

(continues on next page)
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-
py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0 neuron-cc==1.6.13.0

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-
NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# Install OS headers
sudo apt-get install linux-headers-$ (uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add ' --allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
```

(continues on next page)
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name mxnet_venv --display-name "Python
(Neuron MXNet)"

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0 neuron-cc==1.6.13.0

---

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

(continues on next page)
# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name mxnet_venv --display-name "Python (Neuron MXNet)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0 neuron-cc==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Update OS packages
`sudo apt-get update -y`

# Before installing or updating aws-neuron-dkms:
# Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#`sudo systemctl stop neuron-rtd`

# Install OS headers
`sudo apt-get install linux-headers-$(uname -r) -y`

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
`sudo apt-get install aws-neuron-dkms=2.1.5.0 -y`

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
#`reboot`

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
`sudo apt-get install aws-neuron-runtime=1.6.24.0 -y`

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
`sudo apt-get install aws-neuron-tools=1.7.25.0 -y`

`export PATH=/opt/aws/neuron/bin:$PATH`

# Activate MXNet
`source activate aws_neuron_mxnet_p36`

# Set Pip repository to point to the Neuron repository
`pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com`

# Install Neuron MXNet
`pip install mxnet_neuron==1.5.1.1.6.5.0 neuron-cc==1.6.13.0`

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  → 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
  → using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
  → reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
  → using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
  → using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.25.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.6.5.0 neuron-cc==1.6.13.0

## Compile on compute instance

If model compilation occurs outside the model deployment environment, you can install only the Neuron framework extensions and the compiler on any compute instance. This setup is helpful when compiling large complex models that require large amount of memory or during a CICD process where models are compiled in a separate step, prior to deployment.

MXNet 1.8.0
MXNet 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI

Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-˓
py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0 neuron-cc==1.6.13.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-˓
py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0 neuron-cc==1.6.13.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate MXNet
source activate aws_neuron_mxnet_p36
```

(continues on next page)
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0 neuron-cc==1.6.13.0

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0 neuron-cc==1.6.13.0

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0 neuron-cc==1.6.13.0

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0 neuron-cc==1.6.13.0
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0 neuron-cc==1.6.13.0
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0 neuron-cc==1.6.13.0
```
Deploy on AWS ML accelerator instance

During deployment it can be beneficial to reduce the number of components installed in the system. For use-cases where only inference is necessary (compilation is already complete), only the framework and runtime should be installed.

MXNet 1.8.0
MXNet 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
```
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-˓→py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-˓→AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
sudo systemctl stop neuron-rtd

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'

(continues on next page)
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

#################################################
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
#################################################

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

#################################################
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
#################################################

# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

(continues on next page)
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y
export PATH=/opt/aws/neuron/bin:$PATH
# Activate MXNet
source activate aws_neuron_mxnet_p36
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

---

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'
# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-
AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

(continues on next page)
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  `sudo systemctl stop neuron-rtd`

# Install OS headers
`sudo yum install kernel-devel-$({uname} -r) kernel-headers-$({uname} -r) -y`

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
`sudo yum install aws-neuron-dkms=2.1.5.0 -y`

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
`sudo yum install aws-neuron-runtime=1.6.24.0 -y`

`export PATH=/opt/aws/neuron/bin:$PATH`

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
`sudo yum install -y python3 gcc-c++ python3 -m venv mxnet_venv`
`source mxnet_venv/bin/activate`
`pip install -U pip`

# Set Pip repository to point to the Neuron repository
`pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com`

# Install Neuron MXNet
`pip install mxnet_neuron==1.5.1.1.6.5.0`

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
`sudo apt-get update -y`
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
#   └── 'sudo systemctl stop neuron-rtd'
#############################################################################
# Install OS headers
sudo apt-get install linux-headers-$\{$(uname -r)\} -y
# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.1.5.0 -y
#############################################################################
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot
#############################################################################
# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.24.0 -y
export PATH=/opt/aws/neuron/bin:$PATH
# Activate MXNet
source activate aws_neuron_mxnet_p36
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
#   └── 'sudo systemctl stop neuron-rtd'
#############################################################################
# Install OS headers
sudo yum install kernel-devel-$\{$(uname -r)\} kernel-headers-$\{$(uname -r)\} -y

(continues on next page)
# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.1.5.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.24.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0

---

Install Neuron MXNet (Neuron 1.15.0)

- **Develop on AWS ML accelerator instance**
- **Compile on compute instance**
- **Deploy on AWS ML accelerator instance**

---

Develop on AWS ML accelerator instance

The simplest environment setup for model development installs all Neuron SDK components directly on an AWS ML accelerator instance: the Neuron framework extensions, compiler, runtime, and tools. This will allow you to compile, execute, and performance tune your model, all in the same instance. This is the recommended workflow when first starting to work with Neuron device or when optimizing a model.

MXNet 1.8.0
MXNet 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
# Configure Linux for Neuron repository updates
./etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -
# Update OS packages
sudo apt-get update -y
#############################################################################
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
#############################################################################
# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y
# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y
#############################################################################
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot
#############################################################################
# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y
# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.20.0 -y
export PATH=/opt/aws/neuron/bin:$PATH
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv mxnet_venv
(continues on next page)
source mxnet_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name mxnet_venv --display-name "Python (Neuron MXNet)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0 neuron-cc==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

(continues on next page)
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.20.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name mxnet_venv --display-name "Python
(Neuron MXNet)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-
py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0 neuron-cc==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

(continues on next page)
Before installing or updating aws-neuron-dkms:
- Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  `sudo systemctl stop neuron-rtd`

# Install OS headers
```
sudo apt-get install linux-headers-$(uname -r) -y
```

# Install Neuron Driver
- If you are downgrading from newer version, please add `--allow-downgrades` option to 'sudo apt-get install'
```
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update
- Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
- If you are downgrading from newer version, please add `--allow-downgrades` option to 'sudo apt-get install'
```
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y
```

# Install Neuron Tools
- If you are downgrading from newer version, please add `--allow-downgrades` option to 'sudo apt-get install'
```
sudo apt-get install aws-neuron-tools=1.7.20.0 -y
```

```
export PATH=/opt/aws/neuron/bin:$PATH
```

# Activate MXNet
```
source activate aws_neuron_mxnet_p36
```

# Set Pip repository to point to the Neuron repository
```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron MXNet
```
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-\~py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0 neuron-cc==1.6.13.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment
```

---

932 Chapter 4. Neuron Apache MXNet (Incubating)
# Update OS packages
```bash
sudo yum update -y
```

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
```bash
sudo systemctl stop neuron-rtd
```

# Install OS headers
```bash
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
```

# Install Neuron Driver
```bash
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```

# Install Neuron Runtime server
```bash
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y
```

# Install Neuron Tools
```bash
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.20.0 -y
```

```bash
export PATH=/opt/aws/neuron/bin:$PATH
```

# Activate MXNet
```bash
source activate aws_neuron_mxnet_p36
```

# Set Pip repository to point to the Neuron repository
```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron MXNet
```bash
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
```
```bash
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
```
```bash
pip install mx_neuron==1.8.0.1.3.4.0 neuron-cc==1.6.13.0
```

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
./etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.20.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip
```
# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name mxnet_venv --display-name "Python → (Neuron MXNet)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0 neuron-cc==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.20.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name mxnet_venv --display-name "Python
(Neuron MXNet)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0 neuron-cc==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# 'sudo systemctl stop neuron-rtd'
# # Install OS headers
# sudo apt-get install linux-headers-$ (uname -r) -y

(continues on next page)
# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.20.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0 neuron-cc==1.6.13.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package

(continues on next page)
sudo yum install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.20.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0 neuron-cc==1.6.13.0

### Compile on compute instance

If model compilation occurs outside the model deployment environment, you can install only the Neuron framework extensions and the compiler on any compute instance. This setup is helpful when compiling large complex models that require large amount of memory or during a CICD process where models are compiled in a separate step, prior to deployment.

MXNet 1.8.0
MXNet 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
sudo apt-get install -y python3-venv g++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-
˓→py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0 neuron-cc==1.6.13.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-
˓→py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0 neuron-cc==1.6.13.0
```
**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-
˓py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0 neuron-cc==1.6.13.0
```

**Ubuntu AMI**

**Amazon Linux AMI**

**Ubuntu DLAMI**

**Amazon Linux DLAMI**

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0 neuron-cc==1.6.13.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0 neuron-cc==1.6.13.0
```

(continues on next page)
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0 neuron-cc==1.6.13.0

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0 neuron-cc==1.6.13.0

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0 neuron-cc==1.6.13.0

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

**Deploy on AWS ML accelerator instance**

During deployment it can be beneficial to reduce the number of components installed in the system. For use-cases where only inference is necessary (compilation is already complete), only the framework and runtime should be installed.

MXNet 1.8.0
MXNet 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Configure Linux for Neuron repository updates
# /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install --allow-downgrades python3-venv g++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install --allow-downgrades mx_neuron==1.8.0.1.3.4.0

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

(continues on next page)
# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-
py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' 
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after 
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' 
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

(continues on next page)
# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-‌py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   ->'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version , please remove existing package,
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version , please remove existing package,
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-
       →py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.4.0

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Configure Linux for Neuron repository updates
/etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-
       →NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y
```
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package,
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

(continues on next page)
# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$\$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.19.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.5.0

---

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# Install OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.450.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.19.0 -y
export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.6.5.0

**Install Neuron MXNet (Neuron 1.14.2)**

- **Develop on AWS ML accelerator instance**
- **Compile on compute instance**
- **Deploy on AWS ML accelerator instance**

**Develop on AWS ML accelerator instance**

The simplest environment setup for model development installs all Neuron SDK components directly on an AWS ML accelerator instance: the Neuron framework extensions, compiler, runtime, and tools. This will allow you to compile, execute, and performance tune your model, all in the same instance. This is the recommended workflow when first starting to work with Neuron device or when optimizing a model.

**MXNet 1.8.0**

**MXNet 1.5.1**

Ubuntu AMI

Amazon Linux AMI

Ubuntu DLAMI

Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-\ned
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  →'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
  →option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
#  Neuron driver (aws-neuron-dkms) should be re-installed after
  →reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
  →option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.9.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
  →option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.10.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Instal Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name mxnet_venv --display-name "Python
  →(Neuron MXNet)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-
  →py2.py3-none-manylinux2014_x86_64.whl
pip install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.0.0 neuron-cc==1.5.5.0
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# Install OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.9.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.10.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
```

(continues on next page)
# Instal Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name mxnet_venv --display-name "Python → (Neuron MXNet)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.0.0 neuron-cc==1.5.5.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# Install OS headers
sudo apt-get install linux-headers-$((uname -r)) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after:
# sudo apt-get install aws-neuron-dkms=2.0.386.0 -y

...
# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
→ option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.9.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
→ option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.10.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-
→py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.0.0 neuron-cc==1.5.5.0

Note: For a successful installation or update, execute each line of the instructions below separately or
copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
# →'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$ (uname -r) kernel-headers-$ (uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
→using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
→reboot
# Install Neuron Runtime server
# If you are downgrading from newer version , please remove existing package_,
→ using 'sudo yum remove' before installing the older package
```
sudo yum install aws-neuron-runtime=1.6.9.0 -y
```

# Install Neuron Tools
# If you are downgrading from newer version , please remove existing package_,
→ using 'sudo yum remove' before installing the older package
```
sudo yum install aws-neuron-tools=1.7.10.0 -y
```

```
export PATH=/opt/aws/neuron/bin:$PATH
```

# Activate MXNet
```
source activate aws_neuron_mxnet_p36
```

# Set Pip repository to point to the Neuron repository
```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron MXNet
```
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-
→ py2.py3-none-manylinux2014_x86_64.whl
pip install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.0.0 neuron-cc==1.5.5.0
```

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.9.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.10.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++ python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Instal Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name mxnet_venv --display-name "Python (Neuron MXNet)"

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.6.1.0 neuron-cc==1.5.5.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Configure Linux for Neuron repository updates

```
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
```

```
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB
```

# Update OS packages

```
sudo yum update -y
```

# Before installing or updating aws-neuron-dkms:

```
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'
```

# Install OS headers

```
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
```

# Install Neuron Driver

```
# If you are downgrading from newer version , please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.386.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update

```
# Neuron driver (aws-neuron-dkms) should be re-installed after re-boot
```

# Install Neuron Runtime server

```
# If you are downgrading from newer version , please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.9.0 -y
```

# Install Neuron Tools

```
# If you are downgrading from newer version , please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.10.0 -y
```

```
export PATH=/opt/aws/neuron/bin:$PATH
```

# Install Python venv and activate Python virtual environment to install Neuron pip packages.

```
sudo yum install -y python3 gcc-c++ python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip
```

(continues on next page)
# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name mxnet_venv --display-name "Python_{\downarrow}(Neuron MXNet)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.1.0 neuron-cc==1.5.5.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   `sudo systemctl stop neuron-rtd`

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after
# reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.9.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'
sudo apt-get install aws-neuron-tools=1.7.10.0 -y

(continues on next page)
export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.1.0 neuron-cc==1.5.5.0

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   sudo systemctl stop neuron-rtd

# Install OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.9.0 -y

# Install Neuron Tools
# If you are downgrading from newer version, please remove existing package
# using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-tools=1.7.10.0 -y

export PATH=/opt/aws/neuron/bin:$PATH
```

(continues on next page)
# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.1.0 neuron-cc==1.5.5.0

## Compile on compute instance

If model compilation occurs outside the model deployment environment, you can install only the Neuron framework extensions and the compiler on any compute instance. This setup is helpful when compiling large complex models that require large amount of memory or during a CICD process where models are compiled in a separate step, prior to deployment.

MXNet 1.8.0
MXNet 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-˓→py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mxnet_neuron==1.8.0.1.3.0.0 neuron-cc==1.5.5.0

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-\→py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.0.0 neuron-cc==1.5.5.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-\→py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.0.0 neuron-cc==1.5.5.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-\→py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.0.0 neuron-cc==1.5.5.0

Ubuntu AMI

4.1. Installation Guide
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.1.0 neuron-cc==1.5.5.0
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.1.0 neuron-cc==1.5.5.0
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.1.0 neuron-cc==1.5.5.0
```
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.6.1.0 neuron-cc==1.5.5.0
```

Deploy on AWS ML accelerator instance

During deployment it can be beneficial to reduce the number of components installed in the system. For use-cases where only inference is necessary (compilation is already complete), only the framework and runtime should be installed.

MXNet 1.8.0
MXNet 1.5.1
Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
./etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$uname -r -y
```

(continues on next page)
# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.9.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.0.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   -> 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package:
# -> using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after:
# -> reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package:
# -> using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.9.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-
   -py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.0.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Update OS packages

```bash
sudo apt-get update -y
```

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   ```
sudo systemctl stop neuron-rtd
```

# Install OS headers

```bash
sudo apt-get install linux-headers-$(uname -r) -y
```

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'

```bash
sudo apt-get install aws-neuron-dkms=2.0.386.0 -y
```

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades'
# option to 'sudo apt-get install'

```bash
sudo apt-get install aws-neuron-runtime=1.6.9.0 -y
```

```bash
export PATH=/opt/aws/neuron/bin:$PATH
```

# Activate MXNet

```bash
source activate aws_neuron_mxnet_p36
```

# Set Pip repository to point to the Neuron repository

```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron MXNet

```bash
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cul10-1.8.0-
py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cul10-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.0.0
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages

(continues on next page)
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   → 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package
#   using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
#   Neuron driver (aws-neuron-dkms) should be re-installed after
#   reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package
#   using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.9.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-
  →py2.py3-none-manylinux2014_x86_64.whl
pip install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron==1.8.0.1.3.0.0

Ubuntu AMI
Amazon Linux AMI
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Configure Linux for Neuron repository updates

```
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -
# Update OS packages
sudo apt-get update -y
```

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#    'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.9.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.1.0
```

Note: For a successful installation or update, execute each line of the instructions below separately or...
copy the contents of the code block into a script file and source its contents.

```bash
# Configure Linux for Neuron repository updates
sudo tee /etc/yum.repos.d/neuron.repo > /dev/null <<EOF
[neuron]
name=Neuron YUM Repository
baseurl=https://yum.repos.neuron.amazonaws.com
enabled=1
metadata_expire=0
EOF
sudo rpm --import https://yum.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please remove existing package, using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please remove existing package, using 'sudo yum remove' before installing the older package
sudo yum install aws-neuron-runtime=1.6.9.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv mxnet_venv
source mxnet_venv/bin/activate
pip install -U pip

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

(continues on next page)
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.1.0

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-dkms=2.0.386.0 -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Runtime server
# If you are downgrading from newer version, please add '--allow-downgrades' option to 'sudo apt-get install'
sudo apt-get install aws-neuron-runtime=1.6.9.0 -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.1.0
```
# Update OS packages
```
sudo yum update -y
```

# Before installing or updating aws-neuron-dkms:
```
- Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  `sudo systemctl stop neuron-rtd`
```

# Install OS headers
```
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y
```

# Install Neuron Driver
```
- If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
  `sudo yum install aws-neuron-dkms=2.0.386.0 -y`
```

# Warning: If Linux kernel is updated as a result of OS package update
```
- Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```

# Install Neuron Runtime server
```
- If you are downgrading from newer version, please remove existing package using 'sudo yum remove' before installing the older package
  `sudo yum install aws-neuron-runtime=1.6.9.0 -y`
```

```
export PATH=/opt/aws/neuron/bin:$PATH
```

# Activate MXNet
```
source activate aws_neuron_mxnet_p36
```

# Set Pip repository to point to the Neuron repository
```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

```
#Install Neuron MXNet
pip install mxnet_neuron==1.5.1.1.6.1.0
```

### 4.2 User Guide

#### 4.2.1 Neuron Apache MXNet (Incubating) Tutorials

**Before running a tutorial**

You will run the tutorials on an inf1.6xlarge instance running Deep Learning AMI (DLAMI) to enable both compilation and deployment (inference) on the same instance. In a production environment we encourage you to try different instance sizes to optimize to your specific deployment needs.

Follow instructions at [MXNet Tutorial Setup](#) before running an MXNet tutorial on Inferentia.)
MXNet Tutorial Setup

1. Launch an Inf1.6xlarge Instance:
   - Please follow the instructions at launch an Amazon EC2 Instance to Launch an Inf1 instance, when choosing the instance type at the EC2 console. Please make sure to select the correct instance type. To get more information about Inf1 instances sizes and pricing see Inf1 web page.
   - When choosing an Amazon Machine Image (AMI) make sure to select Deep Learning AMI with Conda Options. Please note that Neuron Conda environments are supported only in Ubuntu 18 DLAMI and Amazon Linux2 DLAMI, Neuron Conda environments are not supported in Amazon Linux DLAMI.
   - After launching the instance, follow the instructions in Connect to your instance to connect to the instance.

   Note: You can also launch the instance from AWS CLI, please see AWS CLI commands to launch inf1 instances.

2. Set up a development environment:

   Important:

   For successful installation or update to Neuron 1.16.0 and newer from previous releases:

   - Stop Neuron Runtime 1.x daemon (neuron-rtd) by running: `sudo systemctl stop neuron-rtd`
   - Uninstall `neuron-rtd` by running: `sudo apt remove aws-neuron-runtime` or `sudo yum remove aws-neuron-runtime`
   - Install or upgrade to latest Neuron driver (aws-neuron-dkms) by following the “Setup Guide” instructions.
   - Visit Introducing Neuron Runtime 2.x (libnrt.so) for more information.

   - Enable or install MXNet-Neuron:
     - MXNet 1.8.0
     - MXNet 1.5.1
     - Ubuntu DLAMI
     - Amazon Linux DLAMI

   Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

   ```
   # Note: There is no DLAMI Conda environment for this framework
   # version
   # Framework will be installed/updated inside a Python environment
   ```
   (continues on next page)
# Update OS packages

```bash
sudo apt-get update -y
```

# Before installing or updating aws-neuron-dkms:

- Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:

- DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers

```bash
sudo apt-get install linux-headers-$(uname -r) -y
```

# Install Neuron Driver

```bash
sudo apt-get install aws-neuron-dkms -y
```

# Warning: If Linux kernel is updated as a result of OS package update

Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools

```bash
sudo apt-get install aws-neuron-tools -y
```

```bash
export PATH=/opt/aws/neuron/bin:$PATH
```

# Activate MXNet

```bash
source activate aws_neuron_mxnet_p36
```

# Set Pip repository to point to the Neuron repository

```bash
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron MXNet

```bash
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
```

```bash
pip install mx_neuron neuron-cc
```

**Note:** For a successful installation or update, execute each line of the instructions below
separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework

# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo yum install aws-neuron-tools -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cul10_1.8.0_py2_py3_none_manylinux2014_x86_64.whl
```

(continues on next page)
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest
# DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest
# release" instruction on Neuron documentation

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by
#   calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from
# previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you
#   MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package
# update
# Neuron driver (aws-neuron-dkms) should be re-installed
# after reboot

# Activate MXNet
source activate aws_neuron_mxnet_p36
```

Note: For a successful installation or update, execute each line of the instructions below
separately or copy the contents of the code block into a script file and source its contents.

```bash
# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate MXNet
source activate aws_neuron_mxnet_p36
```

3. Run tutorial in Jupyter notebook:

- Follow instruction at *Setup Jupyter notebook* to:
  1. Start the Jupyter Notebook on the instance
  2. Run the Jupyter Notebook from your local browser
- Connect to the instance from the terminal, clone the Neuron Github repository to the Inf1 instance and then change the working directory to the tutorial directory:

```bash
git clone https://github.com/aws/aws-neuron-sdk.git
cd aws-neuron-sdk/src/examples/mxnet
```
- Locate the tutorial notebook file (.ipynb file) under *aws-neuron-sdk/src/*
examples/mxnet

- From your local browser, open the tutorial notebook from the menu and follow the instructions.

**Computer Vision**

- ResNet-50 tutorial [html] [notebook]
- Model Serving tutorial [html]
- Getting started with Gluon tutorial [html]

**Running Neuron Apache MXNet (Incubating) ResNet50 on Inferentia**

**Introduction:**

In this tutorial we will compile and deploy ResNet50 model for Inferentia. In this tutorial we provide two main sections:

1. Compile the ResNet50 model.
2. Infer the compiled model.

Before running the following verify this Jupyter notebook is running “conda_aws_neuron_mxnet_p36” kernel. You can select the Kernel from the “Kernel -> Change Kernel” option on the top of this Jupyter notebook page. Neuron supports Python module, Symbol APIs and the C predict API. The following quick start example uses the Symbol API.

**Warning**

This tutorial was tested on MXNet-1.5

MXNet-1.5 entered maintenance mode and require Neuron runtime 1.0, please see: **MXNet-1.5 enters maintainence mode**

To setup development environment for MXNet-1.5 see installation instructions for Neuron 1.15.1: **Neuron-1.15.1 MXNet install**

**Compile model on Neuron**

The following step will compile the resnet50 model. Compilation will take a few minutes on inf1.6xlarge. At the end of compilation, the files resnet-50_compiled-0000.params and resnet-50_compiled-symbol.json will be created in local directory.

```
import mxnet as mx
import numpy as np

path='http://data.mxnet.io/models/imagenet/
mx.test_utils.download(path+'resnet/50-layers/resnet-50-0000.params')
mx.test_utils.download(path+'resnet/50-layers/resnet-50-symbol.json')
sym, args, aux = mx.model.load_checkpoint('resnet-50', 0)
```

# Compile for Inferentia using Neuron

(continues on next page)
inputs = { "data" : mx.nd.ones([1,3,224,224], name='data', dtype='float32') }
sym, args, aux = mx.contrib.neuron.compile(sym, args, aux, inputs)

#save compiled model
mx.model.save_checkpoint("resnet-50_compiled", 0, sym, args, aux)

[ ]: !ls

**Deploy on Inferentia**

Using same instance to deploy the model.

[ ]: import mxnet as mx
import numpy as np

path='http://data.mxnet.io/models/imagenet/
mx.test_utils.download(path+'synset.txt')

img = mx.image.imread(fname) # convert into format (batch, RGB, width, height)
img = mx.image.imresize(img, 224, 224) # resize
img = img.transpose((2, 0, 1)) # Channel first
img = img.expand_dims(axis=0) # batchify
img = img.astype(dtype='float32')
sym, args, aux = mx.model.load_checkpoint('resnet-50_compiled', 0)
softmax = mx.nd.random_normal(shape=(1,))
args['softmax_label'] = softmax
args['data'] = img

# Inferentia context
ctx = mx.neuron()
exe = sym.bind(ctx=ctx, args=args, aux_states=aux, grad_req='null')

with open('synset.txt', 'r') as f:
    labels = [l.rstrip() for l in f]
exe.forward(data=img)
prob = exe.outputs[0].asnumpy() # print the top-5
prob = np.squeeze(prob)
a = np.argsort(prob)[:,::-1]
for i in a[0:5]:
    print('probability=%f, class=%s' %(prob[i], labels[i]))

# Sample output will look like below:
#probability=0.634792, class=n02123045 tabby, tabby cat
#probability=0.193601, class=n02123159 tiger cat
#probability=0.103627, class=n02124075 Egyptian cat
#probability=0.031604, class=n02127052 lynx, catamount
#probability=0.015892, class=n02129604 tiger, Panthera tigris
Tutorial: Neuron Apache MXNet (Incubating) Model Serving

This Neuron MXNet Model Serving (MMS) example is adapted from the MXNet vision service example which uses pretrained squeezenet to perform image classification: https://github.com/awslabs/multi-model-server/tree/master/examples/mxnet_vision.

Before starting this example, please ensure that Neuron-optimized MXNet version mxnet-neuron is installed along with Neuron Compiler.

Warning

If you are using MXNet-1.5, please note that MXNet-1.5 entered maintenance mode and require Neuron Runtime 1.x, please see Neuron support for Apache MXNet 1.5 enters maintenance mode. To setup development environment for MXNet-1.5 see installation instructions at Apache MxNet Setup.

If using DLAMI, you can activate the environment aws_neuron_mxnet_p36 and skip the installation part in the first step below.

1. First, install Java runtime and multi-model-server:

```
cd ~/
# sudo yum -y install -q jre # for AML2
sudo apt-get install -y -q default-jre # for Ubuntu
pip install multi-model-server
```

Download the example code:

```
git clone https://github.com/awslabs/multi-model-server
cd ~/multi-model-server/examples/mxnet_vision
```

2. Compile ResNet50 model to Inferentia target by saving the following Python script to compile_resnet50.py and run “python compile_resnet50.py”

```
from packaging import version
import numpy as np
import mxnet as mx

mxnet_version = version.parse(mx.__version__)
if mxnet_version >= version.parse("1.8"):
    import mx_neuron as neuron
else:
    from mxnet.contrib import neuron

path='http://data.mxnet.io/models/imagenet/
mx.test_utils.download(path+'resnet/50-layers/resnet-50-0000.params')
mx.test_utils.download(path+'resnet/50-layers/resnet-50-symbol.json')
mx.test_utils.download(path+'synset.txt')

nn_name = "resnet-50"

#Load a model
sym, args, auxs = mx.model.load_checkpoint(nn_name, 0)

#Define compilation parameters
# - input shape and dtype
inputs = {"data" : mx.nd.zeros([1,3,224,224], dtype='float32') } 
```

(continues on next page)
3. Prepare signature file `signature.json` to configure the input name and shape:

```json
{
  "inputs": [
    {
      "data_name": "data",
      "data_shape": [
        1,
        3,
        224,
        224
      ]
    }
  ]
}
```

4. Prepare `synset.txt` which is a list of names for ImageNet prediction classes:

```bash
curl -O https://s3.amazonaws.com/model-server/model_archive_1.0/examples/squeezenet_v1.1/synset.txt
```

5. Create custom service class following template in `model_server_template` folder:

```bash
cp -r ../model_service_template/* .
```

Edit `mxnet_model_service.py` to use the appropriate context.

Make the following change:

```python
from packaging import version
mxnet_version = version.parse(mx.__version__)
if mxnet_version >= version.parse("1.8"):
    import mx_neuron as neuron
self.mxnet_ctx = mx.neuron()
```

Comment out the existing context set:

```python
#self.mxnet_ctx = mx.cpu() if gpu_id is None else mx.gpu(gpu_id)
```

Also, comment out unnecessary data copy for `model_input` in `mxnet_model_service.py`:

```python
#model_input = [item.as_in_context(self.mxnet_ctx) for item in model_input]
```

6. Package the model with `model-archiver`:

```bash
cd ~/multi-model-server/examples
model-archiver --force --model-name resnet-50_compiled --model-path mxnet_vision --handler mxnet_vision_service:handle
```
7. Start MXNet Model Server (MMS) and load model using RESTful API. Please ensure that Neuron RTD is running with default settings (see Getting started: Installing and Configuring Neuron-RTD):

```bash
cd ~/multi-model-server/
multi-model-server --start --model-store examples
# Pipe to log file if you want to keep a log of MMS
curl -v -X POST "http://localhost:8081/models?initial_workers=1&max_workers=1&synchronous=true&url=resnet-50_compiled.mar"
sleep 10 # allow sufficient time to load model
```

Each worker requires NeuronCore Group that can accommodate the compiled model. Additional workers can be added by increasing max_workers configuration as long as there are enough NeuronCores available. Use `neuron-cli list-ncg` to see NeuronCore Groups being created.

8. Test inference using an example image:

```bash
curl -X POST http://127.0.0.1:8080/predictions/resnet-50_compiled -T kitten_small.jpg
```

You will see the following output:

```json
[
  {
    "probability": 0.6375716328620911,
    "class": "n02123045 tabby, tabby cat"
  },
  {
    "probability": 0.1692783385515213,
    "class": "n02123159 tiger cat"
  },
  {
    "probability": 0.12187337130308151,
    "class": "n02124075 Egyptian cat"
  },
  {
    "probability": 0.028840631246566772,
    "class": "n02127052 lynx, catamount"
  },
  {
    "probability": 0.019691042602062225,
    "class": "n02129604 tiger, Panthera tigris"
  }
]
```

9. To cleanup after test, issue a delete command via RESTful API and stop the model server:

```bash
curl -X DELETE http://127.0.0.1:8081/models/resnet-50_compiled
multi-model-server --stop
```
MXNet 1.8: Getting Started with Gluon Tutorial

Table of Contents

- Overview:
- Setting up your environment
- Run The Tutorial

Overview:

In this tutorial you will compile and deploy resnet-50 using the newly supported MXNet 1.8 and Gluon API on an Inf1 instance. This tutorial is only supported with MXNet 1.8.

Setting up your environment

To run this tutorial, please make sure you deactivate any existing MXNet conda environments you already using. Install MXNet 1.8 by following the instructions at MXNet Setup Guide. You would also need to change your kernel to use the correct Python environment setup earlier by clicking Kernel->Change Kernel->Python (Neuron MXNet)

Run The Tutorial

A trained model must be compiled to Inferentia target before it can run on Inferentia. In this step we compile a pre-trained ResNet50 and export it as a compiled MXNet checkpoint.

1. Create a file compile_resnet50.py with the content below and run it using python compile_resnet50.py. Compilation will take a few minutes on c5.4xlarge. At the end of compilation, the files resnet-50_compiled-0000.params and resnet-50_compiled-symbol.json will be created in local directory.

```python
import mxnet as mx
import mx_neuron as neuron
import numpy as np

path='http://data.mxnet.io/models/imagenet/
mx.test_utils.download(path+'resnet/50-layers/resnet-50-0000.params')
mx.test_utils.download(path+'resnet/50-layers/resnet-50-symbol.json')
block = mx.gluon.nn.SymbolBlock.imports('resnet-50-symbol.json',
    ['data', 'softmax_label'], 'resnet-50-0000.params', ctx=mx.cpu())
block.hybridize()

# Compile for Inferentia using Neuron
inputs = { "data" : mx.nd.ones([1,3,224,224], name='data', dtype='float32'),
    "softmax_label" : mx.nd.ones([1], name='data', dtype='float32') }
block = neuron.compile(block, inputs=inputs)

#save compiled model
block.export("resnet-50_compiled", 0, block)
```
2. Create an inference Python script named `infer_resnet50.py` with the following content:

```python
import numpy as np
import mxnet as mx
import mx_neuron as neuron

path='http://data.mxnet.io/models/imagenet/'
mx.test_utils.download(path+'synset.txt')

img = mx.image.imread(fname) # convert into format (batch, RGB, width, height)
img = mx.image.imresize(img, 224, 224) # resize
img = img.transpose((2, 0, 1)) # Channel first
img = img.expand_dims(axis=0) # batchify
img = img.astype(dtype='float32')
block = mx.gluon.nn.SymbolBlock.imports('resnet-50_compiled-symbol.json',
                                     ['data', 'softmax_label'], 'resnet-50_compiled-0000.params', ctx=mx.cpu())
softmax = mx.nd.random_normal(shape=(1,))
out = block(img, softmax).asnumpy()

with open('synset.txt', 'r') as f:
    labels = [l.rstrip() for l in f]
out = block(img, softmax).asnumpy()
prob = np.squeeze(out)
a = np.argsort(prob)[::-1]
for i in a[0:5]:
    print('probability=%f, class=%s' % (prob[i], labels[i]))
```

3. Run the script to see the inference results:

```
python infer_resnet50.py
```

```
probability=0.643591, class=n02123045 tabby, tabby cat
probability=0.184392, class=n02123159 tiger cat
probability=0.105063, class=n02124075 Egyptian cat
probability=0.030101, class=n02127052 lynx, catamount
probability=0.016112, class=n02129604 tiger, Panthera tigris
```
Natural Language Processing

- BERT tutorial [html]
- MXNet 1.8: Using data parallel mode tutorial [html] [notebook]

Tutorial: Apache MXNet (Incubating) BERT in a Jupyter notebook

Introduction

BERT (Bidirectional Encoder Representations from Transformers) is a Google Research project published in 2018 (https://arxiv.org/abs/1810.04805). BERT has a number of practical applications, it can be used for question answering, sequence prediction and sequence classification amongst other tasks.

This tutorial is using Jupyter notebooks to adapt the BERT-base model from https://github.com/dmlc/gluon-nlp, for the purpose of classifying sentences.

In this tutorial we will use a trained model, an inf1.2xlarge to compile the model to Inferentia using neuron-cc. We will use the same inf1.2xlarge to also run inference. The aim is to demonstrate how to compile, infer and measure performance.

In this tutorial we’ll also leverage the AWS Deep Learning AMI. This tutorial assumes you know how to configure your AWS CLI (https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-configure.html), which is required for the notebooks to run.

The tutorial assumes you are operating in us-east-1. It is possible to run in other regions, but you will need to choose a region where Inf1 instances are supported, and modify the setup script for MRPC or the training notebook where your S3 bucket is created.

Steps Overview

These steps will allow you to setup an environment for running Jupyter Notebooks, and in particular the tutorial on MXNet BERT on Inferentia, and access it via your notebook.

- Launch an EC2 Inf1 instance running the DLAMI (recommended instance: Inf1.2xlarge)
- Connect using ssh and local port forwarding
- Setup a virtual environment for the notebook to use as a kernel
- Fetch the notebook from github
- Start Jupyter and select the correct python virtual environment
- Execute the Notebook to compile a partitioned compute graph

Step 1: Launch EC2 instance

For this task we’ll use a inf1.2xlarge instance. Ensure it has the latest DLAMI. Refer to the Setup Guide for details.
Step 2: Connecting to your instance

In this tutorial we use a Jupyter notebook that runs via a browser on port 8888 by default. For simplicity we will use ssh port forwarding from your machine to the instance. The regular ssh command is:

```
ssh -i "<pem file>" <user>@<instance DNS name>
```

We will modify this base for to use:

```
ssh -i "<pem file>" <user>@<instance DNS name> -L 8888:127.0.0.1:8888
```

On an Ubuntu AMI the user will be ubuntu@, while on an AL2 the user will be ec2-user@

This additional argument forwards connections to port 8888 on your machine to the new Inf1 instance.

Now: ssh to the Inf1 instance

Step 3: Set up the Neuron Runtime environment & create a tutorial directory

If using Conda DLAMI version 27 and up, activate pre-installed MXNet-Neuron environment (using `source activate aws_neuron_mxnet_p36` command). Please update MXNet-Neuron environment by following update steps in Conda DLAMI.

To install in your own AMI, please see Setup Guide to setup virtual environment and install MXNet-Neuron (mxnet-neuron) and Neuron Compiler (neuron-cc) packages. In this tutorial we will use a python virtual environment.

```
# Make sure we are up to date
sudo apt update
sudo apt upgrade
```

Setup a new Python virtual environment:

```
python3 -m venv test_venv
source test_venv/bin/activate
pip install -U pip
```

Modify Pip repository configurations to point to the Neuron repository:

```
te $VIRTUAL_ENV/pip.conf > /dev/null <<EOF
[global]
extra-index-url = https://pip.repos.neuron.amazonaws.com
EOF
```

Install neuron packages:

```
pip install mxnet-neuron
pip install neuron-cc
pip install wget jupyter
```

Create a work directory:

```
mkdir -p notebook_tutorial
cd notebook_tutorial
```
Step 4: Fetch the notebook from GitHub

Run the following command to fetch the notebook into the current directory:

```
```

Step 5: Start Jupyter

From your ssh prompt run

```
# lets clear the old config
mv ~/.jupyter ~/.jupyter.old
mkdir -p ~/.jupyter
echo "c.NotebookApp.iopub_data_rate_limit = 10000000000" > ~/.jupyter/jupyter_notebook_config.py
#Start jupyter
jupyter notebook
```

You should see logging in your ssh session similar to:

```
[I 21:53:11.729 NotebookApp] Using EnvironmentKernelSpecManager...
[I 21:53:12.004 NotebookApp] [nb_conda] enabled
[I 21:53:12.004 NotebookApp] 0 active kernels
[I 21:53:12.004 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
```

If you copy and paste the link that looks like http://localhost:8888/?token=f9ad4086af3c91f33d5587781f9fd8143b4cafbf121a16 into your local browser the Notebook navigation pane should pop up.

This works because ssh is forwarding you local port 8888 through to the Inf1 instance port 8888 where the notebook is running. Note that our new conda environment is visible as “kernel” with the “conda_” prefix (highlighted)
Step 6: Start the notebook and select the correct kernel

- In notebook browser select “bert_mxnet.ipynb”
- This will pop up a new tab. In that tab use the menus:
  - Kernel → Change Kernel → Environment (my_notebook_env)
- Start reading through the self documenting notebook tutorial

Step 7: Terminate your instance

When done, don’t forget to terminate your instance through the AWS console to avoid ongoing charges

Appendix

- Try installing environment_kernels, if you see the following error while launching Jupyter notebook:

```
[C 06:39:39.153 NotebookApp] Bad config encountered during initialization:
[C 06:39:39.153 NotebookApp] The 'kernel_spec_manager_class' trait of
  →<notebook.notebookapp.NotebookApp object at 0x7f21309035c0> instance must
  →be a type, but 'environment_kernels.EnvironmentKernelSpecManager' could
  →not be imported
```

- If you do not see your conda enviroment in jupyter kernel list, try installing the kernel manually:

```
python -m ipykernel install --user --name my_notebook_env --display-name
  →"Python (my_notebook_env)"
```

Using Data Parallel Mode with Gluon MXNet

In this tutorial, you will compile a Gluon BERT model and run in data-parallel mode to completely utilize the NeuronCores. Here you will benchmark a multi-worker setup and compare it with a single worker.

This tutorial is intended only for MXNet-1.8.

In this tutorial, we will be using an inf1.2xlarge with the latest AWS Deep Learning AMI (DLAMI). The inf1.2xlarge instance has 1 AWS Inferentia Chip with 4 NeuronCores.

Setting up your environment

To run this tutorial, please make sure you deactivate any existing MXNet conda environments you already using. Install MXNet 1.8 by following the instructions at MXNet Setup Guide. You would also need to change your kernel to use the correct Python environment setup earlier by clicking Kernel->Change Kernel->Python (Neuron MXNet)
**Install dependencies**

We have to install gluon-nlp to get the BERT model. Run the following command to install:

```bash
[ ]: python -m pip install gluonnlp
```

**Compiling BERT Model**

Next, we compile the Gluon BERT model and save it. Once the model is compiled, we use the same model across the entire tutorial. In this tutorial, we will be using a BERT model with sequence length 32

```python
import os
import mxnet as mx
import mx_neuron
import gluonnlp as nlp

BERT_MODEL = 'bert_12_768_12'
BERT_DATA = 'book_corpus_wiki_en_uncased'
batch_size = 1
seq_len = 32
num_cores = 1
dtype = 'float32'
compiled_model_path = '{}/compiled.{}.compiled'.format(BERT_MODEL, batch_size, seq_len)

model, vocab = nlp.model.get_model(BERT_MODEL,
    dataset_name=BERT_DATA,
    use_classifier=False,
    use_decoder=False, ctx=mx.cpu())

# Create sample inputs for compilation
words = mx.nd.ones([batch_size, seq_len], name='words', dtype=dtype)
valid_len = mx.nd.ones([batch_size,], name='valid_len', dtype=dtype)
segments = mx.nd.ones([batch_size, seq_len], name='segments', dtype=dtype)
inputs = {'data0': words, 'data1': segments, 'data2': valid_len}

# Compiler Args ~~
options = {}
embeddingNames = ['bertmodel0_word_embed_embedding0_fwd',
    'bertmodel0_token_type_embed_embedding0_fwd',
    'bertencoder0_embedding0']
options.update({'force_incl_node_names': embeddingNames})
options.update({'flags': ['--fp32-cast matmult']})

# Compile and save ~~
model = mx_neuron.compile(model, inputs=inputs, **options)
model.export(compiled_model_path)
```
Data Parallel Mode

Data Parallel Mode is a setup in which you launch multiple copies of the same model, such that each model is running independently of the other. In other words, each model has its own resources to run inference.

On an inf1.2xlarge instance, we have 4 NeuronCores. Hence, we can launch 4 models such that each model is loaded on a single NeuronCore. This enables us to process 4 requests concurrently without linear increase in latency. As a result, the throughput of the system increases when compared to a single model inference. This would also allow us to utilize all the 4 NeuronCores on the instance.

Run through the next set of cells to see the difference in throughput as we scale from one model to 4 models running in parallel.

```python
import numpy as np

def get_sample_inputs(batch_size, seq_len):
    words = np.ones([batch_size, seq_len], dtype=np.float32)
    valid_len = np.ones([batch_size,], dtype=np.float32)
    segments = np.ones([batch_size, seq_len], dtype=np.float32)
    inputs = {'data0': words, 'data1': segments, 'data2': valid_len}
    return inputs
```

Next for comparison purposes, we run the setup with 1 worker. To do this, we set the num_cores=1. This would launch only 1 model running on a single NeuronCore. After running the below cell, note down the latency and throughput for the system.

```python
from parallel import NeuronSimpleDataParallel
from benchmark_utils import Results
import time
import functools
import os
import numpy as np
import warnings

num_cores = 1
batch_size=1

# Each worker process should use one core, hence we set
# os.environ['NEURONCORE_GROUP_SIZES'] = "1"
warnings.warn("NEURONCORE_GROUP_SIZES is being deprecated, if your
  →application is using NEURONCORE_GROUP_SIZES please \n  see https://awsdocs-neuron.readthedocs-hosted.com/en/latest/release-notes/
  →deprecation.html#announcing-end-of-support-for-neuroncore-group-sizes \n  for more details.", DeprecationWarning)

os.environ["NEURONCORE_GROUP_SIZES"] = "1"

#Result aggregation class (code in bert_benchmark_utils.py)
results = Results(batch_size, num_cores)

def result_handler(output, start, end):
    elapsed = end - start
    results.add_result([elapsed], [end], [start])

inputs = get_sample_inputs(batch_size, seq_len)
parallel_neuron_model = NeuronSimpleDataParallel(compiled_model_path, num_ →cores, inputs)
```

(continues on next page)
# Starting the inference threads
parallel_neuron_model.start_continuous_inference()

# Warm up the cores
for _ in range(num_cores*4):
    parallel_neuron_model.warmup(inputs)

# Need to run for high number of iterations to benchmark the models
for _ in range(1000):
    parallel_neuron_model.infer(inputs)
    # Passing the result_handler as a callback function
    parallel_neuron_model.add_result(result_handler)

# Stop inference
parallel_neuron_model.stop()

# Since we are using a multi-process execution with a shared queue, some
# inferences may still be in execution phase. Hence we need to wait till all the inputs
# are processed
# add_all_results() will collect all the results of requests which are in
# this state
parallel_neuron_model.add_all_results(result_handler)

with open("benchmark.txt", "w") as f:
    results.report(f, window_size=1)

with open("benchmark.txt", "r") as f:
    for line in f:
        print(line)

Now we run the setup with 4 workers. To do this, we set the num_cores=4. This would launch 4 model
running each running on individual NeuronCore. All the 4 models are running in individual processes, in
other words the models are running in parallel.

To feed the models efficiently, we use the producer-consumer setup, in which all processes running a
model act as consumers. All consumers are fed using a sharing input queue.

Now we run the below setup. You may notice, that the throughput increase by >2x when compared to a
single worker setup.

```python
[ ]: from parallel import NeuronSimpleDataParallel
from benchmark_utils import Results
import time
import functools
import os
import numpy as np

num_cores = 4
batch_size=1

# Important - please read:
# neuron-frameworks/tensorflow-neuron/tutorials/tutorial-tensorflow-
# NeuronCore-Group.html
# Each worker process should use one core, hence we set
# os.environ['NEURONCORE_GROUP_SIZES'] = "1"
```

(continues on next page)
os.environ["NEURONCORE_GROUP_SIZES"] = "1"

# Result aggregation class (code in bert_benchmark_utils.py)
results = Results(batch_size, num_cores)
def result_handler(output, start, end):
    elapsed = end - start
    results.add_result([elapsed], [end], [start])

inputs = get_sample_inputs(batch_size, seq_len)
parallel_neuron_model = NeuronSimpleDataParallel(compiled_model_path, num_cores, inputs)

# Starting the inference threads
parallel_neuron_model.start_continuous_inference()

# Warm up the cores
for _ in range(num_cores*4):
    parallel_neuron_model.warmup(inputs)

# Need to run for high number of iterations to benchmark the models
for _ in range(5000):
    parallel_neuron_model.infer(inputs)
    # Passing the result_handler as a callback function
    parallel_neuron_model.add_result(result_handler)

# Stop inference
parallel_neuron_model.stop()

with open("benchmark.txt", "w") as f:
    results.report(f, window_size=1)

with open("benchmark.txt", "r") as f:
    for line in f:
        print(line)

Utilizing Neuron Capabilities

- NeuronCore Groups tutorial [html] [notebook]
Neuron Apache MXNet (Incubating) - Configurations for NeuronCore Groups Using Resnet50

Introduction:

In this tutorial we will compile and deploy Resnet-50 model in parallel using the concept of NeuronCore Groups on an Inf1 instance. This Jupyter notebook should be run on an instance which is inf1.6xlarge or larger. For simplicity we will run this tutorial on inf1.6xlarge but in real life scenario the compilation should be done on a compute instance and the deployment on inf1 instance to save costs.

To explicitly specify the NeuronCore Groups, set environment variable NEURONCORE_GROUP_SIZES to a list of group sizes. The consecutive NeuronCore groups will be created by Neuron-RTD and be available to map the models to.

Note that in order to map a model to a group, the model must be compiled to fit within the group size. To limit the number of NeuronCores during compilation, use compiler_args dictionary with field "--neuroncore-pipeline-cores" set to the group size:

```
compile_args = {'--neuroncore-pipeline-cores' : 2}
sym, args, auxs = neuron.compile(sym, args, auxs, inputs, **compile_args)
```

In this tutorial we provide two main sections:

1. Compile the Resnet50 model for Neuron
2. Run inference using NeuronCore Groups

Warning

This tutorial was tested on MXNet-1.5

MXNet-1.5 entered maintenance mode and require Neuron runtime 1.0, please see : MXNet-1.5 enters maintainence mode

To setup development environment for MXNet-1.5 see installation instructions for Neuron 1.15.1 : Neuron-1.15.1 MXNet install

Compile model for Neuron

Model must be compiled to Inferentia target before it can be used on Inferentia. In the following we will compile the the flag, --neuroncore-pipeline-cores set to 2 and run it. The files resnet-50_compiled-0000.params and resnet-50_compiled-symbol.json will be created in local directory

```python
import mxnet as mx
import numpy as np
path='http://data.mxnet.io/models/imagenet/
mx.test_utils.download(path+'resnet/50-layers/resnet-50-0000.params')
mx.test_utils.download(path+'resnet/50-layers/resnet-50-symbol.json')
sym, args, aux = mx.model.load_checkpoint('resnet-50', 0)
# Compile for Inferentia using Neuron, fit to NeuronCore group size of 2
inputs = { "data" : mx.nd.ones([1,3,224,224], name='data', dtype='float32') }
compile_args = {'--neuroncore-pipeline-cores' : 2}
sym, args, aux = mx.contrib.neuron.compile(sym, args, aux, inputs, **compile_args)
```
Run inference using NeuronCore Groups

During inference, to subdivide the pool of a single Inferentia chip into three groups of 1, 2, and 1 NeuronCores, specify NEURONCORE_GROUP_SIZES as follows:

NEURONCORE_GROUP_SIZES='1,2,1'

Within the framework, the model can be mapped to group using ctx=mx.neuron(N) context where N is the group index within the NEURONCORE_GROUP_SIZES list.

```python
import os
import warnings

mx.test_utils.download(path+'synset.txt')
img = mx.image.imread(fname)  # convert into format (batch, RGB, width, height)
img = mx.image.imresize(img, 224, 224)  # resize
img = img.transpose((2, 0, 1))  # Channel first
img = img.expand_dims(axis=0)  # batchify
img = img.astype(dtype='float32')

sym, args, aux = mx.model.load_checkpoint('resnet-50_compiled', 0)
softmax = mx.nd.random_normal(shape=(1,))
args['softmax_label'] = softmax
args['data'] = img

warnings.warn("NEURONCORE_GROUP_SIZES is being deprecated, if your application is using NEURONCORE_GROUP_SIZES please see https://awsdocs-neuron.readthedocs-hosted.com/en/latest/release-notes/deprecation.html#announcing-end-of-support-for-neuroncore-group-sizes for more details.", DeprecationWarning)

os.environ['NEURONCORE_GROUP_SIZES'] = '1,2,1'

# Inferentia context - group index 1 (size 2) in NEURONCORE_GROUP_SIZES=1,2,1
ctx = mx.neuron(1)
exe = sym.bind(ctx=ctx, args=args, aux_states=aux, grad_req='null')

with open('synset.txt', 'r') as f:
    labels = [l.rstrip() for l in f]

exe.forward(data=img)
prob = exe.outputs[0].asnumpy()  # print the top-5
prob = np.squeeze(prob)
a = np.argsort(prob)[::-1]
for i in a[0:5]:
    print('probability=%f, class=%s' % (prob[i], labels[i]))
```

4.2. User Guide 993
You can experiment with different Neuron core group combinations and different models.

**Troubleshooting**

If not enough NeuronCores are provided, an error message will be displayed:

```python
 ←h:541: Check failed: rsp.status().code() == 0: Failed load model with
```

### 4.2.2 Neuron Apache MXNet (Incubating) Compilation Python API

The MXNet-Neuron compilation Python API provides a method to compile model graph for execution on Inferentia.

**Description**

Within the graph or subgraph, the compile method selects and sends Neuron-supported operations to Neuron-Compiler for compilation and saves the compiled artifacts in the graph. Uncompilable operations are kept as original operations for framework execution.

The compiled graph can be saved using the MXNet save_checkpoint and served using MXNet Model Serving. Please see [Tutorial: Neuron Apache MXNet (Incubating) Model Serving](#) for more information about exporting to saved model and serving using MXNet Model Serving.

Options can be passed to Neuron compiler via the compile function. For example, the "--neuroncore-pipeline-cores" option directs Neuron compiler to compile each subgraph to fit in the specified number of NeuronCores. This number can be less than the total available NeuronCores on an Inf1 instance. See [Neuron compiler CLI Reference Guide](#) for more information about compiler options.

For debugging compilation, use SUBGRAPH_INFO=1 environment setting before calling the compilation script. The extract subgraphs are preserved as hidden files in the run directory. For more information, see [Using Neuron GatherInfo Tool to collect debug and support information](#)

**MXNet 1.5**

**Method**

```python
from mxnet.contrib import neuron
neuron.compile(sym, args, aux, inputs, **compile_args)
```
Arguments

• **sym** - Symbol object loaded from symbol.json file
• **args** - args/params dictionary loaded from params file
• **aux** - aux/params dictionary loaded from params file
• **inputs** - a dictionary with key/value mappings for input name to input numpy arrays
• **kwargs** (optional) - a dictionary with key/value mappings for MXNet-Neuron compilation and Neuron Compiler options.
  
  – For example, to limit the number of NeuronCores per subgraph, use `compile_args={'--neuroncore-pipeline-cores': N}` where N is an integer representing the maximum number of NeuronCores per subgraph.

  – Additional compiler flags can be passed using `'flags' : [<flags>]` where is a comma separated list of strings. See *Using Neuron GatherInfo Tool to collect debug and support information* for example of passing debug flags to compiler.

  – Advanced option to exclude node names: `compile_args={'excl_node_names' : [<node names>]}` where is a comma separated list of node name strings.

Returns

• **sym** - new partitioned symbol
• **args** - modified args/params
• **auxs** - modified aux/params

Example Usage: Compilation

The following is an example usage of the compilation, with default compilation arguments:

```python
from mxnet.contrib import neuron
...
neuron.compile(sym, args=None, aux=None, inputs={"data" : img})
```

MXNet 1.8

Method

```python
import mx_neuron as neuron
neuron.compile(obj, args=None, aux=None, inputs=None, **compile_args)
```
Arguments

- **obj** - Symbol object loaded from symbol.json file or gluon.HybridBlock object
- **args** (optional) - args/params dictionary loaded from params file. Only needed in case of Symbol object
- **aux** (optional) - aux/params dictionary loaded from params file. Only needed in case of Symbol object
- **inputs** - a dictionary with key/value mappings for input name to input numpy arrays.
- **kwargs** (optional) - a dictionary with key/value mappings for MXNet-Neuron compilation and Neuron Compiler options.

  - For example, to limit the number of NeuronCores per subgraph, use
    `compile_args={'--neuroncore-pipeline-cores' : N}` where N is an integer representing the maximum number of NeuronCores per subgraph.

  - Additional compiler flags can be passed using 'flags' : [<flags>] where is a comma separated list of strings. See Using Neuron GatherInfo Tool to collect debug and support information for example of passing debug flags to compiler.

  - Advanced option to exclude node names: `compile_args={'excl_node_names' : [<node names>]}` where is a comma separated list of node name strings.

  - work_dir: relative or absolute path for storing compiler artifacts (including params and jsons) generated during compilation when SUBGRAPH_INFO=1.

Returns

- **(sym, args, auxs)** - for symbol object as input. sym, args and auxs are new partitioned symbol, modified args/params and modified aux/params repectively.
- **(obj)** - for gluon.HybridBlock object as input. obj is the partitioned and optimized gluon.Hybrid block object for Neuron backend.

Example Usage: Compilation

The following is an example usage of the compilation, with default compilation arguments for symbol object:

```python
import mx_neuron as neuron
...
neuron.compile(sym, args, aux, inputs={'data' : img})
```

The following is an example usage of the compilation, with default compilation arguments for gluon.HybridBlock object (only supported in MXNet-Neuron 1.8):

```python
import mx_neuron as neuron
...
neuron.compile(obj, inputs={'data' : img})
```
Example Usage: Extract Compilation Statistics

To extract operation counts, insert the following code after compile step (assume csym is the compiled MXNet symbol):

```python
import json

def sym_nodes(sym):
    return json.loads(sym.tojson())['nodes']

def count_ops(graph_nodes):
    return len([x['op'] for x in graph_nodes if x['op'] != 'null'])

def get_compile_stats(sym):
    cnt = count_ops(sym_nodes(sym))
    neuron_subgraph_cnt = 0
    neuron_compiled_cnt = 0
    for g in sym_nodes(sym):
        if g['op'] == '_neuron_subgraph_op':
            neuron_subgraph_cnt += 1
            for sg in g['subgraphs']:
                neuron_compiled_cnt += count_ops(sg['nodes'])
    return (cnt, neuron_subgraph_cnt, neuron_compiled_cnt)

original_cnt = count_ops(sym_nodes(sym))
post_compile_cnt, neuron_subgraph_cnt, neuron_compiled_cnt = get_compile_stats(csym)
print("INFO:mxnet: Number of operations in original model: ", original_cnt)
print("INFO:mxnet: Number of operations in compiled model: ", post_compile_cnt)
print("INFO:mxnet: Number of Neuron subgraphs in compiled model: ", neuron_subgraph_cnt)
print("INFO:mxnet: Number of operations placed on Neuron runtime: ", neuron_compiled_cnt)
```

```
INFO:mxnet: Number of operations in original model: 67
INFO:mxnet: Number of operations in compiled model: 4
INFO:mxnet: Number of Neuron subgraphs in compiled model: 2
INFO:mxnet: Number of operations placed on Neuron runtime: 65
```

4.2.3 Neuron Apache MXNet (Incubating) Supported operators

To see a list of supported operators for MXNet, run the following command:

```
neuron-cc list-operators --framework MXNET
```
Neuron Compiler Release [1.6.13.0]

Added

amp_cast
amp_multicast

Neuron Compiler Release [1.4.1.0]

No changes

Neuron Compiler Release [1.4.0.0]

No changes

Neuron Compiler Release [1.3.0.0]

No changes

Neuron Compiler Release [1.2.7.0]

No changes

Neuron Compiler Release [1.2.2.0]

No changes

Neuron Compiler Release [1.2.0.0]

Added

Deconvolution
LayerNorm
Pad
SwapAxis
_contrib_arange_like
_contrib_interleaved_matmul_encdec_qk
_contrib_interleaved_matmul_encdec_valatt
_contrib_interleaved_matmul_selfatt_qk
_contrib_interleaved_matmul_selfatt_valatt
arctan
broadcast_like
cos
erf
pad
sin
slice_axis
Neuron Compiler Release [1.0.24045.0]

Added _contrib_div_sqrt_dim, broadcast_axis

Neuron Compiler Release [1.0.18001.0]

No changes

Neuron Compiler Release [1.0.17937.0]

No changes

Neuron Compiler Release [1.0.16861.0]

Removed log (Was erroneously reported as added in previous release.)

Neuron Compiler Release [1.0.15275]

Added log

Neuron Compiler Release [1.0.12696]

No changes

Neuron Compiler Release [1.0.9410]

No changes

Neuron Compiler Release [1.0.7878]

No changes

Neuron Compiler Release [1.0.6801]

No changes

Neuron Compiler Release [1.0.5939]

no changes
Neuron Compiler Release [1.0.5301]

no changes

Neuron Compiler Release [1.0.4680.0]

```
Activation
BatchNorm
Cast
Concat
Convolution
Convolution_v1
Dropout
Flatten
FullyConnected
LeakyReLU
Pooling
Pooling_v1
RNN
Reshape
SequenceMask
SliceChannel
Softmax
UpSampling
__add_scalar__
__div_scalar__
__mul_scalar__
__pow_scalar__
__rdiv_scalar__
__rpow_scalar__
__rsub_scalar__
__sub_scalar__
_arange
_copy
__div_scalar
__equal_scalar
__full
__greater_equal_scalar
__greater_scalar
__lesser_equal_scalar
__lesser_scalar
__maximum
__maximum_scalar
__minimum
__minimum_scalar
__minus_scalar
__mul_scalar
__not_equal_scalar
__ones
__plus_scalar
__power_scalar
__rdiv_scalar
__rminus_scalar
__rnn_param_concat
__zeros
batch_dot
```

(continues on next page)
broadcast_add
broadcast_div
broadcast_equal
broadcast_greater
broadcast_greater_equal
broadcast_lesser
broadcast_lesser_equal
broadcast_maximum
broadcast_minimum
broadcast_mod
broadcast_mul
broadcast_not_equal
broadcast_sub
ceil
clip
concat
elemwise_add
elemwise_div
elemwise_mul
elemwise_sub
exp
expand_dims
flatten
floor
gather_nd
log
log_softmax
max
mean
min
negative
ones_like
relu
repeat
reshape
reshape_like
reverse
rsqrt
sigmoid
slice
slice_like
softmax
split
sqrt
square
squeeze
stack
sum
tanh
tile
transpose
where
zeros_like
4.2.4 Troubleshooting Guide for Neuron Apache MXNet (Incubating)

Table of Contents

• Inference Runtime Error
  – Out-of-memory error when calling Symbol API bind() too many times
  – Inference crashed with MXNetError: InferShapeKeyword argument name xyz not found
  – Inference crashed at mx.nd.waitall() with MXNetError: Check failed: bin.dtype() == mshadow::kUint8
  – Inference crashed with NRTD error 1002

• Multi-Model Server
  – Failed to create NEURONCORE Group with GRPC Error: Status Error: 14, Error message: “Connect Failed”
  – Multiple MMS workers die with “Backend worker process die.” message
  – MMS throws a “mxnet.base.MXNetError: array::at” error
  – MXNet Model Server is not able to clean up Neuron RTD states after model is unloaded
  – Pipeline mode is not able to execute inferences requests in parallel
  – Features only in MXNet-Neuron 1.5
  – Features only in MXNet-Neuron 1.8

Inference Runtime Error

Out-of-memory error when calling Symbol API bind() too many times

Important: NEURONCORE_GROUP_SIZES is being deprecated, if your application is using
NEURONCORE_GROUP_SIZES please see Migrate your application to Neuron Runtime 2.x (libnrt.so)
for more details.

If you see out-of-memory error when using Symbol API’s bind() function, please ensure that the bind() function is called once for each desired model instance. For example, on inf1.xlarge, use Symbol API to create 4 parallel instances of a model that was compiled to 1 NeuronCore (--neuroncore-pipeline-cores=1), each is bound to a different mx.neuron(i) context where i is the NeuronCore Group index ranging from 0 to 3. Then use 4 threads to feed the 4 instances in parallel. For example:

```python
NUM_PARALLEL = 4
os.environ['NEURONCORE_GROUP_SIZES'] = ','.join('1' for _ in range(NUM_PARALLEL))

data_iter = []
for i in range(NUM_PARALLEL):
    data_iter.append(mx.io.ImageRecordIter(path_imgrec=recfile_base, data_shape=(3, 224, 224), batch_size=1,
        prefetch_buffer=1, prefnum_parts=NUM_PARALLEL, part_index=i))
```

(continues on next page)
sym, args, auxs = mx.model.load_checkpoint('resnet-50_compiled', 0)

exec_list = []
for i in range(NUM_PARALLEL):
    exec = sym.bind(ctx=mx.neuron(i), args=args, aux_states=auxs, grad_req='null')
    exec_list.append(exec)

def single_thread_infer(i):
    for batch in data_iter[i]:
        img = batch.data[0]
        label = batch.label
        feed_dict = {'data': img}
        exe = exec_list[i]
        exe.copy_params_from(feed_dict)
        exe.forward()
        out = exe.outputs[0]

future_list = []
with futures.ThreadPoolExecutor(max_workers=NUM_PARALLEL) as executor:
    for i in range(NUM_PARALLEL):
        future_list.append(executor.submit(single_thread_infer, i))

Inference crashed with MXNetError: InferShapeKeyword argument name xyz not found

If you see MXNetError:

    →InferShapeKeyword argument name xyz not found.

This is followed by a list of “Candidate arguments”. This list shows all the input argument names that the model knows about, and ‘xyz’ is not in the list. To fix this, remove entry xyz from the feed dictionary.

Inference crashed at mx.nd.waitall() with MXNetError: Check failed: bin.dtype() == mshadow::kUint8

When executing Symbol API’s forward function followed by mx.nd.waitall(), where MXNetError exception occurs with ‘Check failed: bin.dtype() == mshadow::kUint8’.

Inference crashed with NRTD error 1002

During inference, the user may encounter an error with details “[NRTD:infer_wait] error: 1002”:

    →h:1175: Check failed: rsp_wait.status().code() == 0 || rsp_wait.status().
    →code() == 1003: Failed
    Infer Wait with Neuron-RTD Error. Neuron-RTD Status Code: 1002, details:
    →"[NRTD:infer_wait] error: 1002"
Runtime errors are listed in **Neuron Runtime return codes**. In particular, 1002 means that some invalid input has been submitted to infer, e.g. missing some of the input tensors, incorrect input tensor sizes. Please examine /var/log/syslog to see more details on the error. For example, you may see:

```
    nonhugetlb] Unexpected input size, for data00, expected: 2097152, received:
    33554432
```

This means that the input tensor size is larger than what the model was compiled for (i.e. the example input tensor shapes passed during compilation).

**Multi-Model Server**

Failed to create NEURONCORE Group with GRPC Error. Status Error: 14, Error message: “Connect Failed”

NOTE: This error only applies to MXNet 1.5.

If the client is unable to start workers and you get a message that MMS is unable to create NeuronCore Group, please check that Neuron RTD is running (neuron-rtd process).

```
{
    "code": 500,
    "type": "InternalServerException",
    "message": "Failed to start workers"
}
```

```
    -inferentia/.inferentia_util.h:218: Check failed: status.ok() Failed to
    -create NeuronCore Group with GRPC Error. Status Error: 14, Error message:
    -"Connect Failed"
```

**Multiple MMS workers die with “Backend worker process die.” message**

**Important:** **NEURONCORE_GROUP_SIZES** is being deprecated, if your application is using **NEURONCORE_GROUP_SIZES** please see **Migrate your application to Neuron Runtime 2.x (libnrt.so)** for more details.

If you run inference with MMS and get multiple messages “Backend worker process die”, please ensure that the number of workers (“intial_workers”) passed during load model is less than or equal to number of NeuronCores available divided by number of NeuronCores required by model.
As indicated in Performance Tuning, for greater flexibility user can use NEURONCORE_GROUP_SIZES to specify the groupings of NeuronCores into Neuron devices, each device consisting of one or more NeuronCores. Each worker would take a device. The total number of NeuronCores taken by all the workers should be less than or equal the total number of NeuronCores visible to neuron-rtd. This situation should be considered at full load (MMS scales up to max_workers). Additionally, to properly assign model to Neuron device, the environment NEURONCORE_GROUP_SIZES must be specified within the model server class (ie. mxnet_model_service.py in the example above). For example, add the following line within mxnet_model_service.py for model compiled to 1 NeuronCore:

```python
os.environ['NEURONCORE_GROUP_SIZES'] = '1'
```

More information about max_worker limit setting can be found at MMS Management API Documentation. For example, to run up to 4 workers in inf1.xlarge where 4 NeuronCores are available by default to Neuron-RTD, set max_workers to 4:

```bash
curl -v -X PUT "http://localhost:8081/models/squeezenet_v1.1_compiled?min_worker=1?max_worker=4"
```

**MMS throws a “mxnet.base.MXNetError: array::at” error**

If you see “mxnet.base.MXNetError: array::at” when running MMS please check that NDArray/Gluon API is not used as they are not supported in MXNet-Neuron. If you would like to use NDArray or Gluon API, please upgrade to MXNet 1.8.
MXNet Model Server is not able to clean up Neuron RTD states after model is unloaded

NOTE: This issue is resolved in version 1.5.1.1.188.0 released 11/17/2020 and only applies for MXNet 1.5.

MXNet Model Server is not able to clean up Neuron RTD states after model is unloaded (deleted) from model server. Restarting the model server may fail with “Failed to create NEURONCORE_GROUP” error:

```
 stole: 348: Check failed: code() = create_eg_rsp.status().code(): Failed to
 create NEURONCORE_GROUP with Neuron-RTD Error. Neuron-RTD Status Code: 9,
 details: ""
```

The workaround is to run “/opt/aws/neuron/bin/neuron-cli reset” to clear Neuron RTD states after all models are unloaded and server is shut down before restarting the model server.

Pipeline mode is not able to execute inferences requests in parallel

If you see that multiple executors in a neuron pipeline setup (one model compiled for more than one neuron-cores using –neuroncore-pipeline-cores option during compilation) are not running in parallel, please set the following MXNet’s environment variables before inference to allow mxnet to execute the CPU ops in parallel. Otherwise it will be sequential and stall the executors.

MXNET_CPU_WORKER_NTHREADS is used to do that. (https://mxnet.apache.org/versions/1.7.0/api/faq/env_var). Setting its value to __subgraph_opt_neuroncore__ in the compiled model json will ensure that all the executors (threads) can be run in parallel.

Features only in MXNet-Neuron 1.5

- Shared memory for IFMaps transfer to neuron runtime (has higher performance compared to GRPC mode)
- Neuron profiling using MXNet
Features only in MXNet-Neuron 1.8

- Gluon API support
- Library mode neuron runtime

4.3 Release notes

4.3.1 What’s New

Apache MXNet Neuron (Incubating) Release Notes

- Apache MXNet Neuron release [1.8.0.2.0.276.0]
- Apache MXNet Neuron release [1.8.0.2.0.271.0]
- [1.5.1.1.7.0.0]
- [1.5.1.1.6.5.0]
- [1.8.0.1.3.4.0]
- [1.5.1.1.6.1.0]
- [1.8.0.1.3.0.0]
- [1.8.0.1.2.1.0]
- [1.8.0.1.1.2.0]
- [1.5.1.1.4.x.x]
- [1.5.1.1.4.4.0]
- [1.5.1.1.3.8.0]
- [1.5.1.1.3.7.0]
- [1.5.1.1.3.2.0]
- [1.5.1.1.2.1.0]
- [1.5.1.1.1.88.0]
- [1.5.1.1.1.52.0]
- [1.5.1.1.1.1.0]
- [1.5.1.1.0.2101.0]
- [1.5.1.1.0.2093.0]
- [1.5.1.1.0.2033.0]
- [1.5.1.1.0.1900.0]
- [1.5.1.1.0.1596.0]
- [1.5.1.1.0.1498.0]
- [1.5.1.1.0.1401.0]
- [1.5.1.1.0.1325.0]
This document lists the release notes for MXNet-Neuron framework.

**Apache MXNet Neuron release [1.8.0.2.0.276.0]**

Date: 11/05/2021

- Updated Neuron Runtime (which is integrated within this package) to `libnrt 2.2.18.0` to fix a container issue that was preventing the use of containers when `/dev/neuron0` was not present. See details here [Neuron Runtime 2.x Release Notes](#).

**Apache MXNet Neuron release [1.8.0.2.0.271.0]**

**New in this release**

- MXNet Neuron 1.8 now support Neuron Runtime 2.x (`libnrt.so` shared library) only.

  **Important:**
  - You must update to the latest Neuron Driver (`aws-neuron-dkms` version 2.1 or newer) for proper functionality of the new runtime library.
  - Read [Introducing Neuron Runtime 2.x (`libnrt.so`)](#) application note that describes why we are making this change and how this change will affect the Neuron SDK in detail.
  - Read [Migrate your application to Neuron Runtime 2.x (`libnrt.so`)](#) for detailed information of how to migrate your application.

- Introducing Flexible Execution Groups (FlexEG) feature. See [Flexible Execution Group (FlexEG) in Neuron-MXNet](#) application note.

**Resolved Issues**

- Fixed a bug that prevented compilation of gluon models with multiple cpu and neuron nodes.
- Added more debug logic to help with profiling of model load timing.

**[1.5.1.1.7.0.0]**

**New in this release**

- MXNet 1.5 enters maintenance mode. Please visit [Neuron support for Apache MXNet 1.5 enters maintenance mode](#) for more information.
Resolved Issues

• Minor bug fixes.

[1.5.1.1.6.5.0]
Date 08/12/2021

Summary

Minor bug fixes and enhancements for MXNet 1.5 Neuron.

[1.8.0.1.3.4.0]
Date 08/12/2021

Summary

Minor bug fixes and enhancements for MXNet 1.8 Neuron.

[1.5.1.1.6.1.0]
Date 07/02/2021

Summary

 Minor bug fixes and enhancements for MXNet 1.5 Neuron.

[1.8.0.1.3.0.0]
Date 07/02/2021

Summary

Support for Autoloop, Cpredict API and minor bug fixes and enhancements for MXNet 1.8 Neuron.
AWS Neuron

Major New Features

• Added support for Autoloop feature for MXNet 1.8 Neuron.

Resolved Issues

• Added support for CPredict API.

[1.8.0.1.2.1.0]
Date 5/28/2021

Summary

Minor bug fixes and enhancements for MXNet 1.8 Neuron

Resolved Issues

• Added support for Neuron profiler

[1.8.0.1.1.2.0]
Date 4/30/2021

Summary

Initial release of Apache MXNet (Incubating) 1.8 for Neuron

Major New Features

• Gluon API and Neuron support for NLP BERT models
• Neuron is now a plugin
• Please note new API changes to support plugin mode: Neuron Apache MXNet (Incubating) Compilation Python API

[1.5.1.1.4.x.x]
Date 5/28/2021
Summary

• Minor enhancements.

[1.5.1.1.4.4.0]

Date 4/30/2021

Summary

• Resolve an issue with Neuron profiling.

Resolved Issues

• Issue: when Neuron profiling is enabled in MXNet-Neuron 1.5.1 (using NEURON_PROFILE=<dir>), and TensorBoard is used to read in the profiled data, user would see an error message "panic: runtime error: index out of range". This issue is resolved in this release.

[1.5.1.1.3.8.0]

Date 3/4/2021

Summary

Minor enhancements.

[1.5.1.1.3.7.0]

Date 2/24/2021

Summary

Fix for CVE-2021-3177.

[1.5.1.1.3.2.0]

Date 1/30/2021
Summary

Various minor improvements

[1.5.1.2.1.0]

Date 12/23/2020

Summary

Various minor improvements

[1.5.1.1.88.0]

Date 11/17/2020

Summary

This release includes the bug fix for MXNet Model Server not being able to clean up Neuron RTD states after model is unloaded (deleted) from model server.

Resolved Issues

- Issue: MXNet Model Server is not able to clean up Neuron RTD states after model is unloaded (deleted) from model server.
  - Workaround for earlier versions: run "/opt/aws/neuron/bin/neuron-cli reset" to clear Neuron RTD states after all models are unloaded and server is shut down.

[1.5.1.1.52.0]

Date 09/22/2020

Summary

Various minor improvements.
Major New Features

Resolved Issues

• Issue: When first importing MXNet into python process and subprocess call is invoked, user may get an OSError exception “OSError: [Errno 14] Bad address” during subprocess call (see https://github.com/apache/incubator-mxnet/issues/13875 for more details). This issue is fixed with a mitigation patch from MXNet for Open-MP fork race conditions.
  – Workaround for earlier versions: Export KMP_INIT_AT_FORK=false before running python process.

[1.5.1.1.1.0]

Date 08/08/2020

Summary

Various minor improvements.

Major New Features

Resolved Issues

[1.5.1.0.2101.0]

Date 08/05/2020

Summary

Various minor improvements.

Major New Features

Resolved Issues

[1.5.1.0.2093.0]

Date 07/16/2020
Summary

This release contains a few bug fixes and user experience improvements.

Major New Features

Resolved Issues

- User can specify NEURONCORE_GROUP_SIZES without brackets (for example, “1,1,1,1”), as can be done in TensorFlow-Neuron and PyTorch-Neuron.
- Fixed a memory leak when inferring neuron subgraph properties
- Fixed a bug dealing with multi-input subgraphs

[1.5.1.1.0.2033.0]

Date 6/11/2020

Summary

- Added support for profiling during inference

Major New Features

- Profiling can now be enabled by specifying the profiling work directory using NEURON_PROFILE environment variable during inference. For an example of using profiling, see tensorboard-neuron. (Note that graph view of MXNet graph is not available via TensorBoard).

Resolved Issues

Known Issues and Limitations

Other Notes

[1.5.1.1.0.1900.0]

Date 5/11/2020
Summary

Improved support for shared-memory communication with Neuron-Runtime.

Major New Features

• Added support for the BERT-Base model (base: L-12 H-768 A-12), max sequence length 64 and batch size of 8.
• Improved security for usage of shared-memory for data transfer between framework and Neuron-Runtime
• Improved allocation and cleanup of shared-memory resource
• Improved container support by automatic falling back to GRPC data transfer if shared-memory cannot be allocated by Neuron-Runtime

Resolved Issues

• User is unable to allocate Neuron-Runtime shared-memory resource when using MXNet-Neuron in a container to communicate with Neuron-Runtime in another container. This is resolved by automatic falling back to GRPC data transfer if shared-memory cannot be allocated by Neuron-Runtime.
• Fixed issue where some large models could not be loaded on inferentia.

Known Issues and Limitations

Other Notes

[1.5.1.0.1596.0]

Date 3/26/2020

Summary

No major changes or fixes

Major New Features

Resolved Issues

Known Issues and Limitations

Other Notes

[1.5.1.0.1498.0]

Date 2/27/2020
Summary

No major changes or fixes.

Major New Features

Resolved Issues

The issue(s) below are resolved:

• Latest pip version 20.0.1 breaks installation of MXNet-Neuron pip wheel which has py2.py3 in the wheel name.

Known Issues and Limitations

• User is unable to allocate Neuron-Runtime shared-memory resource when using MXNet-Neuron in a container to communicate with Neuron-Runtime in another container. To work-around, please set environment variable NEURON_RTD_USE_SHM to 0.

Other Notes

[1.5.1.1.0.1401.0]

Date 1/27/2020

Summary

No major changes or fixes.

Major New Features

Resolved Issues

• The following issue is resolved when the latest multi-model-server with version >= 1.1.0 is used with MXNet-Neuron. You would still need to use “/opt/aws/neuron/bin/neuron-cli reset” to clear all Neuron RTD states after multi-model-server is exited:
  – Issue: MXNet Model Server is not able to clean up Neuron RTD states after model is unloaded (deleted) from model server and previous workaround “/opt/aws/neuron/bin/neuron-cli reset” is unable to clear all Neuron RTD states.
Known Issues and Limitations

• Latest pip version 20.0.1 breaks installation of MXNet-Neuron pip wheel which has py2.py3 in the wheel name. This breaks all existing released versions. The error looks like:

ERROR: Could not find a version that satisfies the requirement mxnet-neuron
   (from versions: none)
ERROR: No matching distribution found for mxnet-neuron

• Work around: install the older version of pip using “pip install pip==19.3.1”.

Other Notes

[1.5.1.0.1325.0]

Date 12/1/2019

Summary

Major New Features

Resolved Issues

• Issue: Compiler flags cannot be passed to compiler during compile call. The fix: compiler flags can be passed to compiler during compile call using “flags” option followed by a list of flags.

• Issue: Advanced CPU fallback option is a way to attempt to improve the number of operators on Inferentia. The default is currently set to on, which may cause failures. The fix: This option is now off by default.

Known Issues and Limitations

• Issue: MXNet Model Server is not able to clean up Neuron RTD states after model is unloaded (deleted) from model server and previous workaround “/opt/aws/neuron/bin/neuron-cli reset” is unable to clear all Neuron RTD states.
  – Workaround: run “sudo systemctl restart neuron-rtd” to clear Neuron RTD states after all models are unloaded and server is shut down.
Other Notes

[1.5.1.0.1349.0]

Date 12/20/2019

Summary

No major changes or fixes. Released with other Neuron packages.

[1.5.1.0.1325.0]

Date 12/1/2019

Summary

Major New Features

Resolved Issues

• Issue: Compiler flags cannot be passed to compiler during compile call. The fix: compiler flags can be passed to compiler during compile call using “flags” option followed by a list of flags.

• Issue: Advanced CPU fallback option is a way to attempt to improve the number of operators on Inferentia. The default is currently set to on, which may cause failures. The fix: This option is now off by default.

Known Issues and Limitations

• Issue: MXNet Model Server is not able to clean up Neuron RTD states after model is unloaded (deleted) from model server and previous workaround “/opt/aws/neuron/bin/neuron-cli reset” is unable to clear all Neuron RTD states.
  – Workaround: run “sudo systemctl restart neuron-rtd” to clear Neuron RTD states after all models are unloaded and server is shut down.

Other Notes

[1.5.1.0.1260.0]

Date: 11/25/2019
Summary

This version is available only in released DLAMI v26.0 and is based on MXNet version 1.5.1. Please update to latest version.

Major new features

Resolved issues

Known issues and limitations

- Issue: Compiler flags cannot be passed to compiler during compile call.
- Issue: Advanced CPU fallback option is a way to attempt to improve the number of operators on Inferentia. The default is currently set to on, which may cause failures.
  - Workaround: explicitly turn it off by setting compile option op_by_op_compiler_retry to 0.
- Issue: Temporary files are put in current directory when debug is enabled.
  - Workaround: create a separate work directory and run the process from within the work directory.
- Issue: MXNet Model Server is not able to clean up Neuron RTD states after model is unloaded (deleted) from model server.
  - Workaround: run “/opt/aws/neuron/bin/neuron-cli reset” to clear Neuron RTD states after all models are unloaded and server is shut down.
- Issue: MXNet 1.5.1 may return inconsistent node names for some operators when they are the primary outputs of a Neuron subgraph. This causes failures during inference.
  - Workaround: Use the excl_node_names compilation option to change the partitioning of the graph during compile so that these nodes are not the primary output of a neuron subgraph.

```python
compile_args = { 'excl_node_names': ['node_name_to_exclude'] }
```

Models Supported

The following models have successfully run on neuron-inferentia systems

1. Resnet50 V1/V2
2. Inception-V2/V3/V4
3. Parallel-WaveNet
4. Tacotron 2
5. WaveRNN
Other Notes

- Python versions supported:
  - 3.5, 3.6, 3.7
- Linux distribution supported:
  - Ubuntu 18, Amazon Linux 2

Warning: Starting with Neuron 1.14.0, Neuron Conda packages in Deep Learning AMI are no longer supported, for more information see blog announcing the end of support for Neuron conda packages.

Neuron Apache MXNet (Incubating) Conda Release Notes

This document lists the release notes for the Neuron Conda-MxNet package.

Table of Contents

- [1.5.1.1.4.4.0]
- [1.5.1.1.3.8.0]
- [1.5.1.1.3.7.0]
- [1.5.1.1.3.2.0]
- [1.5.1.1.2.1.0]
- [1.5.1.1.1.88.0]
- [1.5.1.1.1.52.0_2.0.757.0]
- [1.5.1.1.1.0_2.0.651.0]
- [1.5.1.1.0.2101.0-2.0.631.0]
- [1.5.1.1.0.2093.0-2.0.579.0]
- [1.5.1.1.0.2033.0-2.0.489.0]
- [1.5.1.1.0.1900.0-2.0.274.0]
- [1.5.1.1.0.1596.0-2.0.56.0]
- [1.5.1.1.0.1498.0-1.0.918.0]
- [1.5.1.1.0.1401.0-1.0.737.0]
- [1.5.1.1.0.1349.0-1.0.611.0]
- [1.5.1.1.0.1325.0-1.0.474.0]
- [1.5.1.1.0.1260.0-1.0.298.0]
[1.5.1.4.4.0]

Date: 4/30/2021

Included Neuron Packages

neuron_cc-1.3.7.0
mxnet_neuron-1.5.1.1.4.4.0

Known Issues

• Issue: Running “conda update mxnet-neuron” in Conda DLAMI v37 and v38 would result in environment updates only but not update to the MXNet-Neuron Conda package.
  – Workaround: Running the same command “conda update mxnet-neuron” a second time would result in update to the MXNet-Neuron Conda package.

[1.5.1.3.8.0]

Date: 3/4/2021

Included Neuron Packages

neuron_cc-1.2.7.0
mxnet_neuron-1.5.1.1.3.8.0

Known Issues

• Issue: Running “conda update mxnet-neuron” in Conda DLAMI v37 and v38 would result in environment updates only but not update to the MXNet-Neuron Conda package.
  – Workaround: Running the same command “conda update mxnet-neuron” a second time would result in update to the MXNet-Neuron Conda package.

[1.5.1.3.7.0]

Date: 2/24/2021
Included Neuron Packages

neuron_cc-1.2.7.0
mxnet_neuron-1.5.1.1.3.7.0

Known Issues

• Issue: Running “conda update mxnet-neuron” in Conda DLAMI v37 and v38 would result in environment updates only but not update to the MXNet-Neuron Conda package.

  – Workaround: Running the same command “conda update mxnet-neuron” a second time would result in update to the MXNet-Neuron Conda package.

[1.5.1.1.3.2.0]

Date: 1/30/2021

Included Neuron Packages

neuron_cc-1.2.2.0
mxnet_neuron-1.5.1.1.3.2.0

Known Issues

• Issue: Running “conda update mxnet-neuron” in Conda DLAMI v37 and v38 would result in environment updates only but not update to the MXNet-Neuron Conda package.

  – Workaround: Running the same command “conda update mxnet-neuron” a second time would result in update to the MXNet-Neuron Conda package.

[1.5.1.1.2.1.0]

Date: 12/23/2020

Included Neuron Packages

neuron_cc-1.1.7.0
mxnet_neuron-1.5.1.1.2.1.0
Known Issues

- Issue: Running “conda update mxnet-neuron” in Conda DLAMI v37 and v38 would result in environment updates only but not update to the MXNet-Neuron Conda package.
  - Workaround: Running the same command “conda update mxnet-neuron” a second time would result in update to the MXNet-Neuron Conda package.

[1.5.1.1.1.88.0]

Date: 11/17/2020

Included Neuron Packages

neuron_cc-1.0.24045.0
mxnet_neuron-1.5.1.1.1.88.0

[1.5.1.1.52.0_2.0.757.0]

Date: 09/22/2020

Included Neuron Packages

neuron_cc-1.0.20600.0
mxnet_neuron-1.5.1.1.1.52.0

[1.5.1.1.1.0_2.0.651.0]

Date: 08/08/2020

Included Neuron Packages

neuron_cc-1.0.18001.0
mxnet_neuron-1.5.1.1.1.0

[1.5.1.0.2101.0-2.0.631.0]

Date: 08/05/2020
Included Neuron Packages

neuron_cc-1.0.17937.0
mxnet_neuron-1.5.1.1.0.2101.0

[1.5.1.1.0.2093.0-2.0.579.0]

Date: 07/16/2020
Now supporting Python 3.7 Conda packages in addition to Python 3.6 Conda packages.

Included Neuron Packages

neuron_cc-1.0.16861.0
mxnet_neuron-1.5.1.1.0.2093.0

[1.5.1.1.0.2033.0-2.0.489.0]

Date: 06/11/2020

Included Neuron Packages

neuron_cc-1.0.15275.0
mxnet-neuron=1.5.1.1.0.2033.0

[1.5.1.1.0.1900.0-2.0.274.0]

Date 5/11/2020

Included Neuron Packages

neuron-cc-1.0.12696.0
mxnet-neuron=1.5.1.1.0.1900.0

[1.5.1.1.0.1596.0-2.0.56.0]

Date 3/26/2020
Included Neuron Packages

neuron-cc-1.0.9410.0
mxnet-neuron=1.5.1.1.0.1596.0

[1.5.1.1.0.1498.0-1.0.918.0]

Date 2/27/2020

Included Neuron Packages

neuron-cc-1.0.7878.0
mxnet-neuron=1.5.1.1.0.1498.0

[1.5.1.1.0.1401.0-1.0.737.0]

Date 1/27/2020

Included Neuron Packages

neuron-cc-1.0.6801.0
mxnet-neuron-1.5.1.1.0.1401.0

[1.5.1.1.0.1349.0-1.0.611.0]

Date 12/20/2019

Included Neuron Packages

neuron-cc-1.0.5939.0
mxnet-neuron-1.5.1.1.0.1349.0

[1.5.1.1.0.1325.0-1.0.474.0]

Date 12/1/2019
Included Neuron Packages

neuron-cc-1.0.5301.0
mxnet-neuron-1.5.1.1.0.1325.0

Known Issues and Limitations

[1.5.1.1.0.1260.0-1.0.298.0]

Date: 11/25/2019
This version is only available from the release DLAMI v26.0. Please see Known Issues to latest version.

Included Neuron Packages

neuron-cc-1.0.4680.0
mxnet-neuron-1.5.1.1.0.1260.0

Known Issues and Limitations

Please update to the latest conda package:

```
source activate <conda environment>
conda update mxnet-neuron
```

For example, on Conda DLAMI:

```
source activate aws_neuron_tensorflow_p36
conda update mxnet-neuron
```
NEURON TUTORIALS

- PyTorch Tutorials
- TensorFlow Tutorials
- Neuron Apache MXNet (Incubating) Tutorials
- tools-tutorials
- Neuron Containers Tutorials
NEURON INFERENCE PERFORMANCE

Table of Contents
- Natural Language Processing
- Computer Vision

The following tables contain the reference inference performance for models in the Neuron Tutorials. Follow the links on each row to replicate similar results in your own environment. Refer to Setup Environment documentation to create a new environment based on the latest Neuron release.

Last update: October, 27th, 2021
# 6.1 Natural Language Processing

<table>
<thead>
<tr>
<th>Model Tutorial</th>
<th>Framework</th>
<th>Avg Throughput (sec)</th>
<th>Latency P50 (ms)</th>
<th>Latency P99 (ms)</th>
<th>Cost per 1M inferences</th>
<th>Application Type</th>
<th>Runtime</th>
<th>Batch Size</th>
<th>Model Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>HuggingFace Pretrained BERT</td>
<td>PyTorch 1.9.1</td>
<td>inf1.xlarge</td>
<td>22</td>
<td>23</td>
<td>$0.066</td>
<td>Batch</td>
<td>1.16.0</td>
<td>Data Parallel</td>
<td>fp32, bert-base-cased-finetuned-mrpc, sequence-length=128</td>
</tr>
<tr>
<td>Using NeuronCore Pipeline</td>
<td>PyTorch 1.9.1</td>
<td>inf1.6xlarge</td>
<td>6</td>
<td>6</td>
<td>$0.161</td>
<td>Real Time</td>
<td>1.16.0</td>
<td>Model Pipeline</td>
<td>fp32, bert-base-uncased, sequence-length=128</td>
</tr>
<tr>
<td>HuggingFace distilBERT with Tensorflow2</td>
<td>Tensorflow 2.5</td>
<td>inf1.6xlarge</td>
<td>125</td>
<td>125</td>
<td>$0.317</td>
<td>Batch</td>
<td>1.16.0</td>
<td>Data Parallel</td>
<td>fp32, distilbert-base-uncased-finetuned-sst-2-english, sequence-length=128</td>
</tr>
</tbody>
</table>

**Note:** Cost per 1M inferences is calculated using US East (N. Virginia) On-Demand hourly rate.

**Real Time** application refers to batch size 1 inference for minimal latency. **Batch** application refers to maximum throughput with minimum cost-per-inference.
## 6.2 Computer Vision

<table>
<thead>
<tr>
<th>Model Tutorial</th>
<th>Framework Type</th>
<th>Avg Throughput (/sec)</th>
<th>Latency P50 (ms)</th>
<th>Latency P99 (ms)</th>
<th>Cost per 1M Inferences</th>
<th>Application Type</th>
<th>Neuron Version</th>
<th>Run Mode</th>
<th>Batch Size</th>
<th>Model Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running OpenPose on Inferentia</td>
<td>Tensorflow 1.15</td>
<td>inf1.xlarge</td>
<td>62</td>
<td>69</td>
<td>$1.101</td>
<td>Real Time</td>
<td>1.16</td>
<td>Data Parallel</td>
<td>1</td>
<td>fp16</td>
</tr>
<tr>
<td>ResNet-50 optimization example</td>
<td>Tensorflow 1.15</td>
<td>inf1.xlarge</td>
<td>17</td>
<td>21</td>
<td>$0.002</td>
<td>Batch</td>
<td>1.16</td>
<td>Data Parallel</td>
<td>10</td>
<td>fp16</td>
</tr>
<tr>
<td>Resnet50 model for Inferentia</td>
<td>PyTorch 1.9.1</td>
<td>inf1.xlarge</td>
<td>22</td>
<td>23</td>
<td>$0.000</td>
<td>Batch</td>
<td>1.16</td>
<td>Data Parallel</td>
<td>10</td>
<td>fp32</td>
</tr>
<tr>
<td>Evaluate YOLO v4 on Inferentia</td>
<td>PyTorch 1.9.1</td>
<td>inf1.xlarge</td>
<td>42</td>
<td>61</td>
<td>$0.544</td>
<td>Real Time</td>
<td>1.16</td>
<td>Data Parallel</td>
<td>1</td>
<td>fp32</td>
</tr>
</tbody>
</table>

Note: **Cost per 1M inferences** is calculated using US East (N. Virginia) On-Demand hourly rate.

**Real Time** application refers to batch size 1 inference for minimal latency. **Batch** application refers to maximum throughput with minimum cost-per-inference.
CHAPTER
SEVEN

WHAT’S NEW

• Neuron 1.16.3 (01/05/2022)
• Neuron 1.16.2 (12/15/2021)
• Neuron 1.16.1 (11/05/2021)
• Neuron 1.16.0 (10/27/2021)
• Detailed release notes
• Previous Releases

7.1 Neuron 1.16.3 (01/05/2022)

Neuron 1.16.3 is a minor release. This release includes performance enhancements and operator support in PyTorch Neuron and minor bug fixes in Neuron Compiler.

7.2 Neuron 1.16.2 (12/15/2021)

Neuron 1.16.2 is a patch release. This release includes performance enhancements and minor bug fixes in Neuron Compiler and PyTorch Neuron.

7.3 Neuron 1.16.1 (11/05/2021)

Neuron 1.16.1 is a patch release. This release fixes a bug in Neuron Runtime that would have prevented users from launching a container that doesn’t use all of the Neuron Devices in the instance. If you are using Neuron within a container, please update to this new release by updating to latest Neuron ML framework package, Neuron Tools, and/or TensorFlow Neuron Model Server.

- To update to latest PyTorch 1.9.1: `pip install --upgrade torch-neuron neuron-cc[tensorflow] torchvision`
- To update to latest TensorFlow 2.5.1: `pip install --upgrade tensorflow-neuron[cc]`
- To update to latest TensorFlow 1.15.5: `pip install --upgrade tensorflow-neuron==1.15.5. neuron-cc`
- To update to latest MXNet 1.8.0: `pip install --upgrade mx_neuron neuron-cc`
7.4 Neuron 1.16.0 (10/27/2021)

Neuron 1.16.0 is a release that requires your attention. You must update to the latest Neuron Driver (aws-neuron-dkms version 2.1 or newer) for successful installation or upgrade.

This release introduces Neuron Runtime 2.x, upgrades PyTorch Neuron to PyTorch 1.9.1, adds support for new APIs (torch.neuron.DataParallel() and torch_neuron.is_available()), adds new features and capabilities (compiler --fast-math option for better fine-tuning of accuracy/performance and MXNet FlexEG feature), improves tools, adds support for additional operators, improves performance (Up to 20% additional throughput and up to 25% lower latency), and reduces model loading times. It also simplifies Neuron installation steps, and improves the user experience of container creation and deployment. In addition it includes bug fixes, new application notes, updated tutorials, and announcements of software deprecation and maintenance.

- Neuron Runtime 2.x
  - Introducing Neuron Runtime 2.x (libnrt.so) - In this release we are introducing Neuron Runtime 2.x. The new runtime is a shared library (libnrt.so), replacing Neuron Runtime 1.x which was a server daemon (neruon-rtd).

  Upgrading to libnrt.so is expected to improves throughput and latency, simplifies Neuron installation and upgrade process, introduces new capabilities for allocating NeuronCores to applications, streamlines container creation, and deprecates tools that are no longer needed. The new library-based runtime (libnrt.so) is directly integrated into Neuron’s ML Frameworks (with the exception of MXNet 1.5) and Neuron Tools packages. As a result, users no longer need to install/deploy the aws-neuron-runtime package.

  Important:
  - You must update to the latest Neuron Driver (aws-neuron-dkms version 2.1 or newer) for proper functionality of the new runtime library.
  - Read Introducing Neuron Runtime 2.x (libnrt.so) application note that describes why we are making this change and how this change will affect the Neuron SDK in detail.
  - Read Migrate your application to Neuron Runtime 2.x (libnrt.so) for detailed information of how to migrate your application.

- Performance
  - Updated performance numbers - Improved performance: Up to 20% additional throughput and up to 25% lower latency.

- Documentation resources
  - Improved Neuron Setup Guide.
  - New Introducing Neuron Runtime 2.x (libnrt.so) application note.
  - New Running inference on variable input shapes with bucketing application note.
  - New Mixed precision and performance-accuracy tuning application note.
  - New Data Parallel Inference on Torch Neuron application note.
  - New Flexible Execution Group (FlexEG) in Neuron-MXNet application note.
  - New Parallel Execution using NEURONCORE_GROUP_SIZES application note.
– Updated ResNet50 model for Inferentia tutorial to use torch.neuron.DataParallel().

• PyTorch
– PyTorch now supports Neuron Runtime 2.x only. Please visit Introducing Neuron Runtime 2.x (libnrt.so) for more information.
– Introducing PyTorch 1.9.1 support.
– Introducing new APIs: torch.neuron.DataParallel() (see Data Parallel Inference on Torch Neuron application note for more details) and torch_neuron.is_available().
– Introducing new operators support.
– For more information visit PyTorch Neuron

• TensorFlow 2.x
– TensorFlow 2.x now supports Neuron Runtime 2.x only. Please visit Introducing Neuron Runtime 2.x (libnrt.so) for more information.
– Updated Tensorflow 2.3.x from Tensorflow 2.3.3 to Tensorflow 2.3.4.
– Updated Tensorflow 2.4.x from Tensorflow 2.4.2 to Tensorflow 2.4.3.
– Updated Tensorflow 2.5.x from Tensorflow 2.5.0 to Tensorflow 2.5.1.
– Introducing new operators support
– For more information visit TensorFlow Neuron

• TensorFlow 1.x
– TensorFlow 1.x now supports Neuron Runtime 2.x only. Please visit Introducing Neuron Runtime 2.x (libnrt.so) for more information.
– Introducing new operators support.
– For more information visit TensorFlow Neuron

• MXNet 1.8
– MXNet 1.8 now supports Neuron Runtime 2.x only. Please visit Introducing Neuron Runtime 2.x (libnrt.so) for more information.
– Introducing Flexible Execution Groups (FlexEG) feature.
– MXNet 1.5 enters maintenance mode. Please visit Neuron support for Apache MXNet 1.5 enters maintenance mode for more information.
– For more information visit Neuron Apache MXNet (Incubating)

• Neuron Compiler
– Introducing the --fast-math option for better fine-tuning of accuracy/performance. See Mixed precision and performance-accuracy tuning
– For more information visit Neuron Compiler

• Neuron Tools
– Updates have been made to neuron-ls and neuron-top to improve the interface and utility of information provided.
– *neuron-monitor* has been enhanced to include additional information when used to monitor the latest Frameworks released with Neuron 1.16.0. See *Neuron Tools 2.x Release Notes*.

– *neuron-cli* is entering maintenance mode as its use is no longer relevant when using ML Frameworks with an integrated Neuron Runtime (libnrt.so).

– For more information visit *Neuron Tools*

• **Neuron Containers**

  – Starting with Neuron 1.16.0, installation of Neuron ML Frameworks now includes an integrated Neuron Runtime library. As a result, it is no longer required to deploy *neuron-rtd*. Please visit *Introducing Neuron Runtime 2.x (libnrt.so)* for information.

  – When using containers built with components from Neuron 1.16.0, or newer, please use *aws-neuron-dkms* version 2.1 or newer and the latest version of *aws-neuron-runtime-base*. Passing additional system capabilities is no longer required.

  – For more information visit *Containers*

• **Neuron Driver**

  – Support is added for Neuron Runtime 2.x (libnrt.so).

  – Memory improvements have been made to ensure all allocations are made with 4K alignments.

• **Software Deprecation**

  – *Announcing end of support for NEURONCORE_GROUP_SIZES*

  – *End of support for NeuronCore Groups (NCG)*

• **Software maintenance mode**

  – *Neuron Runtime 1.x (neuron-rtd) enters maintenance mode*

  – *Neuron support for Apache MXNet 1.5 enters maintenance mode*

  – *neuron-cli enters maintenance mode*
## 7.5 Detailed release notes

<table>
<thead>
<tr>
<th>Software</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>- Release Details</td>
</tr>
<tr>
<td></td>
<td>- Software deprecation</td>
</tr>
<tr>
<td></td>
<td>- Software maintenance mode</td>
</tr>
<tr>
<td>PyTorch</td>
<td>- PyTorch Neuron release notes</td>
</tr>
<tr>
<td></td>
<td>- PyTorch Supported operators</td>
</tr>
<tr>
<td>TensorFlow 2.x</td>
<td>- TensorFlow-Neuron 2.x Release Notes</td>
</tr>
<tr>
<td></td>
<td>- TensorFlow 2.x Accelerated Python APIs and Graph Ops</td>
</tr>
<tr>
<td></td>
<td>- TensorFlow-Model-Server-Neuron 2.x Release Notes</td>
</tr>
<tr>
<td>TensorFlow 1.x</td>
<td>- TensorFlow-Neuron 1.x Release Notes</td>
</tr>
<tr>
<td></td>
<td>- TensorFlow 1.x Supported operators</td>
</tr>
<tr>
<td></td>
<td>- TensorFlow-Model-Server-Neuron 1.x Release Notes</td>
</tr>
<tr>
<td>Apache MXNet (Incubating)</td>
<td>- Apache MXNet Neuron (Incubating) Release Notes</td>
</tr>
<tr>
<td></td>
<td>- Neuron Apache MXNet (Incubating) Supported operators</td>
</tr>
<tr>
<td>Compiler</td>
<td>- Neuron Compiler Release Notes</td>
</tr>
<tr>
<td></td>
<td>- Neuron Supported operators</td>
</tr>
<tr>
<td>Runtime</td>
<td>- Neuron Runtime 2.x Release Notes</td>
</tr>
<tr>
<td></td>
<td>- Neuron Driver 2.x Release Notes</td>
</tr>
<tr>
<td>Containers</td>
<td>- Neuron K8 Release Notes</td>
</tr>
<tr>
<td></td>
<td>- Neuron Containers Release Notes</td>
</tr>
<tr>
<td>Tools</td>
<td>- Neuron Tools 2.x Release Notes</td>
</tr>
<tr>
<td></td>
<td>- Neuron Plugin for TensorBoard Release Notes</td>
</tr>
<tr>
<td>Software Deprecation</td>
<td>- Software deprecation</td>
</tr>
<tr>
<td>Software Maintenance</td>
<td>- Software maintenance mode</td>
</tr>
</tbody>
</table>
7.6 Previous Releases

7.6.1 Previous Neuron Release Notes

- Neuron 1.15.2 (09/22/2021)
- Neuron 1.15.1 (08/30/2021)
- Neuron 1.15.0 (08/12/2021)
- Neuron 1.14.2 (07/26/2021)
- Neuron 1.14.1 (07/02/2021)
- Neuron 1.14.0 (05/28/2021)
- Neuron 1.14.0 (05/28/2021)
- Neuron 1.13.0 (05/01/2021)
- March 4, 2021 Release (Patch)
- February 24, 2021 Release (Patch)
- January 30, 2021 Release
- December 23, 2020 Release
- November 17, 2020 Release
- October 22, 2020 Release
- September 22, 2020 Release
- August 19, 2020 Release
- August 8, 2020 Release
- August 4, 2020 Release
- July 16, 2020 Release
- June 18, 2020 Release
- Jun 11, 2020 Release
- May 15, 2020 Release
- May 11, 2020 Release
- Mar 26, 2020 Release
- Feb 27, 2020 Release
- Jan 28, 2020 Release
Neuron 1.15.2 (09/22/2021)

Neuron 1.15.2 includes bug fixes for the tensorflow-model-server-neuron 2.5.1.1.6.8.0 package and several other bug fixes for tensorflow-neuron/tensorflow-model-server-neuron packages.

Neuron 1.15.1 (08/30/2021)

Neuron 1.15.1 includes bug fixes for the aws-neuron-dkms package and several other bug fixes for related packages.

Neuron 1.15.0 (08/12/2021)

Neuron 1.15.0 is the first release to support TensorFlow 2. In this release TensorFlow 2 supports language transformer base models like BERT. The TensorFlow 2 support will be enhanced in future releases to support additional models.

- **TensorFlow 2.x** - To get started with TensorFlow 2.x:
  - Run the TensorFlow 2 HuggingFace distilBERT Tutorial.
  - Read TensorFlow 2.x FAQ
  - See newly introduced TensorFlow-Neuron 2.x Tracing API.
  - See TensorFlow 2.x Accelerated Python APIs and Graph Ops.

- **Documentation**
  - New Inferentia Model Architecture Fit application note added in this release. This application note describes what types of deep learning model architectures perform well out of the box and provides guidance on techniques you can use to optimize your deep learning models for Inferentia.
  - New Neuron inference performance page provides performance information for popular models and links to test these models in your own environment. The data includes throughout and latency numbers, cost per inference, for both realtime and offline applications.
  - New TensorFlow 2 HuggingFace distilBERT Tutorial.
  - New Bring your own HuggingFace pretrained BERT container to Sagemaker Tutorial.

- **More information**
  - Tensorflow-Neuron 1.x Release Notes
  - Neuron Compiler Release Notes
  - TensorFlow-Model-Server-Neuron 1.x Release Notes

Neuron 1.14.2 (07/26/2021)

This release (Neuron 1.14.2), include bug fixes and minor enhancements to Neuron Runtime:

- Neuron Runtime - see Neuron Runtime 2.x Release Notes
Neuron 1.14.1 (07/02/2021)

This release (Neuron 1.14.1) include bug fixes and minor enhancements:

- Neuron PyTorch - This release adds “Dynamic Batching” feature support, see PyTorch-Neuron trace python API for more information, the release also add support for new operators and include additional bug fixes and minor enhancements, for more information see PyTorch Neuron release notes.
- Neuron Runtime - see Neuron Runtime 2.x Release Notes.

Neuron 1.14.0 (05/28/2021)


- Neuron PyTorch - Convolution operator support has been extended to include ConvTranspose2d variants.
- Neuron PyTorch - Updated tutorials to use Hugging Face Transformers 4.6.0.
- Neuron PyTorch - Additional performance enhancements, memory optimizations, and bug fixes. see PyTorch Neuron release notes.
- Neuron Compiler - New feature - Uncompressed NEFF format for faster loading models prior inference. Enable it by –enable-fast-loading-neuron-binaries. Some cases of large models may be detrimentally impacted as it will not be compressed but many cases will benefit.
- Neuron MXNet - Enhancements and minor bug fixes (MXNet 1.8), see Apache MXNet Neuron (Incubating) Release Notes.
- Neuron Tools - Minor bug fixes and enhancements.

Software Deprecation

- End of support for Neuron Conda packages in Deep Learning AMI, users should use pip upgrade commands to upgrade to latest Neuron version in DLAMI, see blog.
- End of support for Ubuntu 16, see documentation.
Neuron 1.14.0 (05/28/2021)


• Neuron PyTorch - First release of Neuron PyTorch 1.8.1.
• Neuron PyTorch - Convolution operator support has been extended to include ConvTranspose2d variants.
• Neuron PyTorch - Updated tutorials to use Hugging Face Transformers 4.6.0.
• Neuron PyTorch - Additional performance enhancements, memory optimizations, and bug fixes. see PyTorch Neuron release notes.
• Neuron Compiler - New feature - Uncompressed NEFF format for faster loading models prior inference. Enable it by –enable-fast-loading-neuron-binaries. Some cases of large models may be detrimentally impacted as it will not be compressed but many cases will benefit.
• Neuron Compiler - Additional performance enhancements, memory optimizations, and bug fixes, see Neuron Compiler Release Notes.
• Neuron TensorFlow - Performance enhancements, memory optimizations, and bug fixes. see Tensorflow-Neuron 1.x Release Notes.
• Neuron MXNet - Enhancements and minor bug fixes (MXNet 1.8), see Apache MXNet Neuron (Incubating) Release Notes.
• Neuron Tools - Minor bug fixes and enhancements.
• Software Deprecation
  – End of support for Neuron Conda packages in Deep Learning AMI, users should use pip upgrade commands to upgrade to latest Neuron version in DLAMI, see blog.
  – End of support for Ubuntu 16, see documentation.

Neuron 1.13.0 (05/01/2021)

This release introduces higher performance, updated framework support, new tutorials, and adding models and tools:

• Additional compiler improvements boost performance up to 20% higher throughput compared to previous release across model types.
• Improving usability for NLP models, with out-of-the-box 12x higher-throughput at 70% lower cost for Hugging Face Transformers pre-trained BERT Base models, see pytorch-tutorials-neuroncore-pipeline-pytorch.
• Upgrade Apache MXNet (Incubating) to 1.8, where Neuron is now a plugin, see Apache MXNet Neuron (Incubating) Release Notes.
• PyTorch ResNext models now functional with new operator support, see PyTorch Neuron release notes.
• PyTorch Yolov5 support, see PyTorch Neuron release notes.
• MXNet (Incubating): Gluon API and Neuron support for NLP BERT models, see Apache MXNet Neuron (Incubating) Release Notes.
• PyTorch Convolution operator support has been extended to include most Conv1d and Conv3d variants, please see PyTorch Supported operators for the complete list of operators.
• First release of Neuron plugin for TensorBoard, see Neuron Plugin for TensorBoard Release Notes.
Software Deprecation

- **End of support for Neuron Conda packages in Deep Learning AMI starting Neuron 1.14.0**
- **End of support for Ubuntu 16 starting Neuron 1.14.0**
- **End of support for classic TensorBoard-Neuron starting Neuron 1.13.0 and introducing Neuron Plugin for TensorBoard**

March 4, 2021 Release (Patch)

This release includes bug fixes and minor enhancements to the Neuron Runtime and Tools.

February 24, 2021 Release (Patch)

This release updates all Neuron packages and libraries in response to the Python Security issue CVE-2021-3177 as described here: [https://nvd.nist.gov/vuln/detail/CVE-2021-3177](https://nvd.nist.gov/vuln/detail/CVE-2021-3177). This vulnerability potentially exists in multiple versions of Python including 3.5, 3.6, 3.7. Python is used by various components of Neuron, including the Neuron compiler as well as Machine Learning frameworks including TensorFlow, PyTorch and Apache MXNet (Incubating). It is recommended that the Python interpreters used in any AMIs and containers used with Neuron are also updated.

Python 3.5 reached end-of-life, from this release Neuron packages will not support Python 3.5. Users should upgrade to latest DLAMI or upgrade to a newer Python versions if they are using other AMI.

January 30, 2021 Release

This release continues to improve the NeuronCore Pipeline performance for BERT models. For example, running BERT Base with the the neuroncore-pipeline-cores compile option, at batch=3, seqlen=32 using 16 Neuron Cores, results in throughput of up to 5340 sequences per second and P99 latency of 9ms using Tensorflow Serving.

This release also adds operator support and performance improvements for the PyTorch based DistilBert model for sequence classification.

December 23, 2020 Release

This release introduces a PyTorch 1.7 based torch-neuron package as a part of the Neuron SDK. Support for PyTorch model serving with TorchServe 0.2 is added and will be demonstrated with a tutorial. This release also provides an example tutorial for PyTorch based Yolo v4 model for Inferentia.

To aid visibility into compiler activity, the Neuron-extended Frameworks TensorFlow and PyTorch will display a new compilation status indicator that prints a dot (.) every 20 seconds to the console as compilation is executing.

Important to know:

1. This update continues to support the torch-neuron version of PyTorch 1.5.1 for backwards compatibility.
2. As Python 3.5 reached end-of-life in October 2020, and many packages including TorchVision and Transformers have stopped support for Python 3.5, we will begin to stop supporting Python 3.5 for frameworks, starting with PyTorch-Neuron version [1.1.7.0] in this release. You can continue to use older versions with Python 3.5.
November 17, 2020 Release

This release improves NeuronCore Pipeline performance. For example, running BERT Small, batch=4, seqlen=32 using 4 Neuron Cores, results in throughput of up to 7000 sequences per second and P99 latency of 3ms using Tensorflow Serving.

Neuron tools updated the NeuronCore utilization metric to include all inf1 compute engines and DMAs. Added a new neuron-monitor example that connects to Grafana via Prometheus. We’ve added a new sample script which exports most of neuron-monitor’s metrics to a Prometheus monitoring server. Additionally, we also provided a sample Grafana dashboard. More details at Neuron Tools.

ONNX support is limited and from this version onwards we are not planning to add any additional capabilities to ONNX. We recommend running models in TensorFlow, PyTorch or MXNet for best performance and support.

October 22, 2020 Release

This release adds a Neuron kernel mode driver (KMD). The Neuron KMD simplifies Neuron Runtime deployments by removing the need for elevated privileges, improves memory management by removing the need for huge pages configuration, and eliminates the need for running neuron-rtd as a sidecar container. Documentation throughout the repo has been updated to reflect the new support. The new Neuron KMD is backwards compatible with prior versions of Neuron ML Frameworks and Compilers - no changes are required to existing application code.

More details in the Neuron Runtime release notes at Neuron runtime.

September 22, 2020 Release

This release improves performance of YOLO v3 and v4, VGG16, SSD300, and BERT. As part of these improvements, Neuron Compiler doesn’t require any special compilation flags for most models. Details on how to use the prior optimizations are outlined in the neuron-cc Neuron Compiler Release Notes.

The release also improves operational deployments of large scale inference applications, with a session management agent incorporated into all supported ML Frameworks and a new neuron tool called neuron-monitor allows to easily scale monitoring of large fleets of Inference applications. A sample script for connecting neuron-monitor to Amazon CloudWatch metrics is provided as well. Read more about using neuron-monitor Neuron Monitor User Guide.

August 19, 2020 Release

Bug fix for an error reporting issue with the Neuron Runtime. Previous versions of the runtime were only reporting uncorrectable errors on half of the dram per Inferentia. Other Neuron packages are not changed.

August 8, 2020 Release

This release of the Neuron SDK delivers performance enhancements for the BERT Base model. Sequence lengths including 128, 256 and 512 were found to have best performance at batch size 6, 3 and 1 respectively using publically available versions of both Pytorch (1.5.x) and Tensorflow-based (1.15.x) models. The compiler option “-O2” was used in all cases.

A new Kubernetes scheduler extension is included in this release to improve pod scheduling on inf1.6xlarge and inf1.24xlarge instance sizes. Details on how the scheduler works and how to apply the scheduler can be found Neuron Kubernetes Scheduler Extension. Check the Neuron K8 Release Notes for details changes to k8 components going forward.
**August 4, 2020 Release**

Bug fix for a latent issue caused by a race condition in Neuron Runtime leading to possible crashes. The crash was observed under stress load conditions. All customers are encouraged to update the latest Neuron Runtime package (aws-neuron-runtime), version 1.0.8813.0 or newer. Other Neuron packages are being updated as well, but are to be considered non-critical updates.

**July 16, 2020 Release**

This release of Neuron SDK adds support for the OpenPose (posenet) Neural Network. An example of using Openpose for end to end inference is available Running OpenPose on Inferentia.

A new PyTorch auto-partitioner feature now automatically builds a Neuron specific graph representation of PyTorch models. The key benefit of this feature is automatic partitioning the model graph to run the supported operators on the NeuronCores and the rest on the host. PyTorch auto-partitioner is enabled by default with ability to disable if a manual partition is needed. More details PyTorch Neuron. The release also includes various bug fixes and increased operator support.

**Important to know:**

1. This update moves the supported version for PyTorch to the current release (PyTorch 1.5.1)
2. This release supports Python 3.7 Conda packages in addition to Python 3.6 Conda packages

**June 18, 2020 Release**

Point fix an error related to yum downgrade/update of Neuron Runtime packages. The prior release fails to successfully downgrade/update Neuron Runtime Base package and Neuron Runtime package when using Yum on Amazon Linux 2.

Please remove and then install both packages on AL2 using these commands:

```
# Amazon Linux 2
sudo yum remove aws-neuron-runtime-base
sudo yum remove aws-neuron-runtime
sudo yum install aws-neuron-runtime-base
sudo yum install aws-neuron-runtime
```

**Jun 11, 2020 Release**

This Neuron release provides support for the recent launch of EKS for Inf1 instance types and numerous other improvements. More details about how to use EKS with the Neuron SDK can be found in AWS documentation here.

This release adds initial support for OpenPose PoseNet for images with resolutions upto 400x400.

This release also adds a ‘-O2’ option to the Neuron Compiler. ‘-O2’ can help with handling of large tensor inputs.

In addition the Neuron Compiler increments the version of the compiled artifacts, called “NEFF”, to version 1.0. Neuron Runtime versions earlier than the 1.0.6905.0 release in May 2020 will not be able to execute NEFFs compiled from this release forward. Please see NEFF Support Table: for compatibility.

Stay up to date on future improvements and new features by following the Neuron SDK Roadmap.

Refer to the detailed release notes for more information on each Neuron component.
Important to know:

1. **Size of neural network.** The current Neuron compiler release has a limitation in terms of the size of neural network it could effectively optimize for. The size of neural network is influenced by a number of factors including: a) type of neural network (CNN, LSTM, MLP) , b) number of layers, c) sizes of input (dimension of the tensors, batch size, . . .). Using the Neuron Compiler ‘-O2’ option can help with handling of large tensor inputs for some models. If not used, Neuron limits the size of CNN models like ResNet to an input size of 480x480 fp16/32, batch size=4; LSTM models like GNMT to have a time step limit of 900; MLP models like BERT to have input size limit of sequence length=128, batch=8.

2. **INT8 data type is not currently supported by the Neuron compiler.**

3. Neuron does not support TensorFlow 2 or PyTorch 1.4.0.

**May 15, 2020 Release**

Point fix an error related to installation of the Neuron Runtime Base package. The prior release fails to successfully start Neuron Discovery when the Neuron Runtime package is not also installed. This scenario of running Neuron Discovery alone is critical to users of Neuron in container environments.

Please update the aws-neuron-runtime-base package:

```
# Ubuntu 18 or 16:
sudo apt-get update
sudo apt-get install aws-neuron-runtime-base

# Amazon Linux, Centos, RHEL
sudo yum update
sudo yum install aws-neuron-runtime-base
```

**May 11, 2020 Release**

This release provides additional throughput improvements to running inference on a variety of models; for example BERTLarge throughput has improved by an additional 35% compared to the previous release and with peak throughput of 360 seq/second on inf1.xlarge (more details [Running TensorFlow BERT-Large with AWS Neuron](Running%20TensorFlow%20BERT-Large%20with%20AWS%20Neuron)).

In addition to the performance boost, this release adds PyTorch, and MXNet framework support for BERT models, as well as expands container support in preparation to an upcoming EKS launch.

We continue to work on new features and improving performance further, to stay up to date follow this repository and our Neuron roadmap.

Refer to the detailed release notes for more information for each Neuron component.

Important to know:

1. **Size of neural network.** The current Neuron compiler release has a limitation in terms of the size of neural network it could effectively optimize for. The size of neural network is influenced by a number of factors including: a) type of neural network (CNN, LSTM, MLP) , b) number of layers, c) sizes of input (dimension of the tensors, batch size, . . .). As a result, we limit the sizes of CNN models like ResNet to have an input size limit of 480x480 fp16/32, batch size=4; LSTM models like GNMT to have a time step limit of 900; MLP models like BERT to have input size limit of sequence length=128, batch=8.

2. **INT8 data type is not currently supported by the Neuron compiler.**

3. Neuron does not support TensorFlow 2 or PyTorch 1.4.0.

7.6. Previous Releases
AWS Neuron

Mar 26, 2020 Release

This release supports a variant of the SSD object detection network, a SSD inference demo is available Running SSD300 with AWS Neuron.

This release also enhances our Tensorboard support to enable CPU-node visibility.

Refer to the detailed release notes for more information for each neuron component.

Important to know:

1. Size of neural network. The current Neuron compiler release has a limitation in terms of the size of neural network it could effectively optimize for. The size of neural network is influenced by a number of factors including: a) type of neural network (CNN, LSTM, MLP), b) number of layers, c) sizes of input (dimension of the tensors, batch size, . . .). As a result, we limit the sizes of CNN models like ResNet to have an input size limit of 480x480 fp16/32, batch size=4; LSTM models like GNMT to have a time step limit of 900; MLP models like BERT to have input size limit of sequence length=128, batch=8.

2. INT8 data type is not currently supported by the Neuron compiler.

3. Neuron does not support TensorFlow 2 or PyTorch 1.4.0.

Feb 27, 2020 Release

This release improves performance throughput by up to 10%, for example ResNet-50 on inf1.xlarge has increased from 1800 img/sec to 2040 img/sec, Neuron logs include more detailed messages and various bug fixes. Refer to the detailed release notes for more details.

We continue to work on new features and improving performance further, to stay up to date follow this repository, and watch the AWS Neuron developer forum.

Important to know:

1. Size of neural network. The current Neuron compiler release has a limitation in terms of the size of neural network it could effectively optimize for. The size of neural network is influenced by a number of factors including: a) type of neural network (CNN, LSTM, MLP), b) number of layers, c) sizes of input (dimension of the tensors, batch size, . . .). As a result, we limit the sizes of CNN models like ResNet to have an input size limit of 480x480 fp16/32, batch size=4; LSTM models like GNMT to have a time step limit of 900; MLP models like BERT to have input size limit of sequence length=128, batch=8.

2. Computer-vision object detection and segmentation models are not yet supported.

3. INT8 data type is not currently supported by the Neuron compiler.

4. Neuron does not support TensorFlow 2 or PyTorch 1.4.0.
Jan 28, 2020 Release

This release brings significant throughput improvements to running inference on a variety of models; for example Resnet50 throughput is increased by 63% (measured 1800 img/sec on inf1.xlarge up from 1100/sec, and measured 2300/sec on inf1.2xlarge). BERTbase throughput has improved by 36% compared to the re:Invent launch (up to 26100seq/sec from 19200seq/sec on inf1.24xlarge), and BERTlarge improved by 15% (230 seq/sec, compared to 200 running on inf1.2xlarge). In addition to the performance boost, this release includes various bug fixes as well as additions to the GitHub with Neuron Features diving deep on how Neuron performance features work and overall improved documentation following customer input.

We continue to work on new features and improving performance further, to stay up to date follow this repository, and watch the AWS Neuron developer forum.

Important to know:

1. Size of neural network. The current Neuron compiler release has a limitation in terms of the size of neural network it could effectively optimize for. The size of neural network is influenced by a number of factors including: a) type of neural network (CNN, LSTM, MLP) , b) number of layers, c) sizes of input (dimension of the tensors, batch size, . . . ). As a result, we limit the sizes of CNN models like ResNet to have an input size limit of 480x480 fp16/32, batch size=4; LSTM models like GNMT to have a time step limit of 900; MLP models like BERT to have input size limit of sequence length=128, batch=8.

2. Computer-vision object detection and segmentation models are not yet supported.

3. INT8 data type is not currently supported by the Neuron compiler.

4. Neuron does not support TensorFlow 2 or PyTorch 1.4.0.

Neuron SDK Release Notes Structure

The Neuron SDK is delivered through commonly used package mananagers (e.g. PIP, APT and YUM). These packages are then themselves packaged into Conda packages that are integrated into the AWS DLAMI for minimal developer overhead.

The Neuron SDK release notes follow a similar structure, with the core improvements and known-issues reported in the release notes of the primary packages (e.g. Neuron-Runtime or Neuron-Compiler release notes), and additional release notes specific to the package-integration are reported through their dedicated release notes (e.g. Conda or DLAMI release notes).
Neuron Technical and Application notes provide additional information that help users better utilize Neuron and achieve better performance.

### 8.1 Neuron Batching

Batching refers to the process of grouping multiple inference requests together and processing them as a group. Batching is typically used as an optimization for throughput at the expense of higher latency. Batching is usually implemented on a layer-by-layer basis, which allows for each set of weights in a given layer to be reused for each inference in the batch before needing to retrieve additional new weights. This enables Neuron to better amortize the cost of reading weights from the external memory (i.e. read weights from the memory once, and use them in multiple calculations), and thus improve the overall hardware efficiency.

The concept of batched inference is illustrated in the example below, with a single NeuronCore performing batched computation of a 3 layer neural network with a batch-size of 4. The NeuronCore reads weights from the external memory, and then performs the corresponding computations for all 4 inference-requests, before reading the next set of weights, thus better amortizing the cost of reading the weights from the memory.

Neuron uses an ahead-of-time compiler, so to enable batching in Neuron, a model should be explicitly compiled for a target batch-size by setting the input-tensor batch dimension accordingly. Users are encouraged to evaluate multiple batch sizes, in order to determine the optimal latency/throughput deployment-point (which is model/application dependent).

During inference, dynamic batching can be used to process a larger client-side inference batch-size, and allow the framework to automatically break up the user-batch into smaller batch sizes, to match the compiled batch-size. This technique increases the achievable throughput by hiding the framework-to-neuron overhead, and amortizing it over a larger batch size. To enable dynamic batching in TensorFlow, user would set the argument `dynamic_batch_size=True` during call to `tfn.saved_model.compile` method.
For example, the TensorFlow code snippet below enables batching, with dynamic-batching and a batch-size of N=4 when compiling a model to Inferentia target:

```python
import numpy as np
import tensorflow.neuron as tfn

# To change the batch size, change the first dimension in example_input
batch_sz = 4
example_input = np.zeros([batch_sz, 224, 224, 3], dtype='float16')

# Note: You can add custom compilation flags such as for mixed precision
# here. None are needed for this model.
compiler_args = []

# TensorFlow code snippet for dynamic batching
import tensorflow as tf
import tensorflow.neuron as tfn
predictor = tf.contrib.predictor.from_saved_model("rn50_fp16_compiled/1")
rt_batch_sz_list = [1, 4, 7, 8, 1024]
for rt_batch_sz in rt_batch_sz_list:
    example_input = np.zeros([rt_batch_sz, 224, 224, 3], dtype='float16')
    model_feed_dict = {'input_1:0': example_input}
    result = predictor(model_feed_dict)
```

The following TensorFlow code snippet shows that the model can accept inference requests with arbitrary batch size:

```python
import tensorflow as tf
import tensorflow.neuron as tfn

# TensorFlow code snippet for dynamic batching
import tensorflow as tf
import tensorflow.neuron as tfn
predictor = tf.contrib.predictor.from_saved_model("rn50_fp16_compiled/1")
rt_batch_sz_list = [1, 4, 7, 8, 1024]
for rt_batch_sz in rt_batch_sz_list:
    example_input = np.zeros([rt_batch_sz, 224, 224, 3], dtype='float16')
    model_feed_dict = {'input_1:0': example_input}
    result = predictor(model_feed_dict)
```

**Note:** Depending on the neural network size, Neuron will have a maximum batch size that works optimally on Inferentia. If an unsupported batch size is used, an internal compiler error message will be displayed. A simple way to explore optimal batch size for your specific model is to increment the batch size from 1 upward, one at a time, and test application performance.

### 8.2 NeuronCore Pipeline

The Neuron software feature referred to a NeuronCore Pipeline refers to the process of sharding a computation graph across multiple NeuronCores, caching the model parameters in each core’s on-chip memory (cache), and then streaming inference requests across the cores in a pipelined manner. Based on the number of NeuronCores selected, the model might get seamlessly sharded across up-to 16 Inferentia devices (i.e. 64 NeuronCores). This enables users to optimize for both throughput and latency, as it enables the NeuronCores to process neural-networks with locally cached data and avoid the cost of accessing external memory.
One benefit to this approach is that NeuronCore Pipeline can typically hit maximal hardware efficiency without the need for batching (e.g. BERT, ResNet50).

For maximal performance, users should choose an instance-size that can cache the entire model by using sufficient NeuronCores. Inf1 instance types have different number of Inferentia devices, each of which has 4 NeuronCores, as shown here https://aws.amazon.com/ec2/instance-types/inf1/

To enable the NeuronCore Pipeline optimization, the compiler should be invoked with the following flags: `--neuroncore-pipeline-cores` N. The number of NeuronCores is typically chosen to be the minimal number that can fit the entire model, which is currently done through a trial-and-error process (compiling to different number of cores and looking for compilation success/failure message). This process will be automated in the future. A simple formula to help define the number of NeuronCores that may be an appropriate choice is

```
neuroncore-pipeline-cores = 4 * round( number-of-weights-in-model/(2 * 10^7) )
```

This allocates a set of NeuronCores based on the size of the given model’s weights and normalizes to multiples of 4 so it uses full Inferentias.

The code snippet below shows how to compile a model with NeuronCore Pipeline for 16 NeuronCores (instance size inf1.6xlarge).

```python
import numpy as np
import tensorflow.neuron as tfn

example_input = np.zeros([1,224,224,3], dtype='float16')
tfn.saved_model.compile("rn50_fp16",
    "rn50_fp16_compiled/1",
    model_feed_dict={'input_1:0': example_input },
    compiler_args = ["--neuroncore-pipeline-cores", '16'])
```
A typical Neuron developer flow includes compilation phase and then deployment (inference) on inf1 instance/s. You can develop on Neuron using one of the following combinations of developer flows:

9.1 Compile with Framework API and Deploy on EC2 Inf1

Table of Contents

- Description
- Setup Environment
  - 1. Launch an Inf1 Instance
  - 2. Set up a development environment
    * Enable PyTorch-Neuron
    * Enable TensorFlow-Neuron
    * Enable Apache MXNet (Incubating)
  - 3. Set up Jupyter notebook
9.1.1 Description

You can use a single inf1 instance as a development environment to compile and deploy Neuron models. In this developer flow, you provision an EC2 inf1 instance using a Deep Learning AMI (DLAMI) and execute the two steps of the development flow in the same instance. The DLAMI comes pre-packaged with the Neuron frameworks, compiler, and required runtimes to complete the flow. Development happens through Jupyter Notebooks or using a secure shell (ssh) connection in terminal. Follow the steps bellow to setup your environment.

**Note:** Model compilation can be executed on a non-inf1 instance for later deployment. Follow the same EC2 Developer Flow Setup using other instance families and leverage Amazon Simple Storage Service (S3) to share the compiled models between different instances.

9.1.2 Setup Environment

1. **Launch an Inf1 Instance**

   - Please follow the instructions at launch an Amazon EC2 Instance to Launch an Inf1 instance, when choosing the instance type at the EC2 console. Please make sure to select the correct instance type. To get more information about Inf1 instances sizes and pricing see Inf1 web page.
   - When choosing an Amazon Machine Image (AMI) make sure to select Deep Learning AMI with Conda Options. Please note that Neuron Conda environments are supported only in Ubuntu 18 DLAMI and Amazon Linux2 DLAMI, Neuron Conda environments are not supported in Amazon Linux DLAMI.
   - After launching the instance, follow the instructions in Connect to your instance to connect to the instance

   **Note:** You can also launch the instance from AWS CLI, please see AWS CLI commands to launch inf1 instances.
2. Set up a development environment

Enable PyTorch-Neuron

Important:

For successful installation or update to Neuron 1.16.0 and newer from previous releases:

- Stop Neuron Runtime 1.x daemon (neuron-rtd) by running: 
  ```
  sudo systemctl stop neuron-rtd
  ```
- Uninstall neuron-rtd by running: 
  ```
  sudo apt remove aws-neuron-runtime or sudo yum remove aws-neuron-runtime
  ```
- Install or upgrade to latest Neuron driver (aws-neuron-dkms) by following the “Setup Guide” instructions.
- Visit *Introducing Neuron Runtime 2.x (libnrt.so)* for more information.

PyTorch 1.9.1
PyTorch 1.8.1
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$ (uname -r) -y

# Install Neuron Driver
```

(continues on next page)
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y
export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron neuron-cc[tensorflow] torchvision

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo yum install aws-neuron-tools -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron neuron-cc[tensorflow] torchvision

Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not
---include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

(continues on next page)
# Activate PyTorch

```bash
source activate aws_neuron_pytorch_p36
```

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

---

```bash
# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
```bash
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y
```n

# Install Neuron Driver
```bash
sudo yum install aws-neuron-dkms -y
```

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate PyTorch
```bash
source activate aws_neuron_pytorch_p36
```n

---
Enable TensorFlow-Neuron

Important:

For successful installation or update to Neuron 1.16.0 and newer from previous releases:

- Stop Neuron Runtime 1.x daemon (neuron-rtd) by running: `sudo systemctl stop neuron-rtd`
- Uninstall neuron-rtd by running: `sudo apt remove aws-neuron-runtime` or `sudo yum remove aws-neuron-runtime`
- Install or upgrade to latest Neuron driver (aws-neuron-dkms) by following the “Setup Guide” instructions.
- Visit Introducing Neuron Runtime 2.x (libnrt.so) for more information.

TensorFlow 2.5.1
TensorFlow 1.15.5
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y
```

(continues on next page)
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python
→(Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]

# Optional: Install Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
To install or update to Neuron versions 1.16.0 and newer from previous releases:
- DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
```
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y
```

# Install Neuron Driver
```
sudo yum install aws-neuron-dkms -y
```

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
```
sudo yum install aws-neuron-tools -y
```

# Install Neuron TensorBoard
```
pip install tensorboard-plugin-neuron
```

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
```
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip
```

# Install Jupyter notebook kernel
```
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels
```

# Set Pip repository to point to the Neuron repository
```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

# Install Neuron TensorFlow
```
pip install tensorflow-neuron[cc]
```

# Optional: Install Neuron TensorFlow model server
```
sudo yum install tensorflow-model-server-neuron -y
```

Ubuntu DLAMI

Amazon Linux DLAMI

9.1. Compile with Framework API and Deploy on EC2 Inf1
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate TensorFlow
source activate aws_neuron_tensorflow_p36
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
Enable Apache MXNet (Incubating)

Important:

For successful installation or update to Neuron 1.16.0 and newer from previous releases:

- Stop Neuron Runtime 1.x daemon (neuron-rtd) by running: `sudo systemctl stop neuron-rtd`
- Uninstall neuron-rtd by running: `sudo apt remove aws-neuron-runtime` or `sudo yum remove aws-neuron-runtime`
- Install or upgrade to latest Neuron driver (aws-neuron-dkms) by following the “Setup Guide” instructions.
- Visit *Introducing Neuron Runtime 2.x (libnrt.so)* for more information.

MXNet 1.8.0
MXNet 1.5.1
Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron neuron-cc

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo yum install aws-neuron-tools -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron neuron-cc

Ubuntu DLAMI

Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
AWS Neuron

# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not
---include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction
---on Neuron documentation

###################################################################################################

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo
---systemctl stop neuron-rtd'

###################################################################################################

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or
---upgrade to latest Neuron driver

###################################################################################################

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

###################################################################################################

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

###################################################################################################

# Activate MXNet
source activate aws_neuron_mxnet_p36

Note:  For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

(continues on next page)
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

```bash
# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y
```

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

```bash
# Activate MXNet
source activate aws_neuron_mxnet_p36
```

## 3. Set up Jupyter notebook

To develop from a Jupyter notebook see *Jupyter Notebook QuickStart*

You can also run a Jupyter notebook as a script, first enable the ML framework Conda or Python environment of your choice and see *Running Jupyter Notebook as script* for instructions.

### 9.2 Compile with Sagemaker Neo and Deploy on Sagemaker Hosting

**Table of Contents**

- Description
- Setup Environment
9.2.1 Description

You can use SageMaker Neo to compile models for deployment on SageMaker Hosting using ml.inf1 instances. In this developer flow, you provision a Sagemaker Notebook instance to train, compile and deploy your model using the SageMaker Python SDK. Follow the steps bellow to setup your environment.

9.2.2 Setup Environment

1. Create an Amazon SageMaker Notebook Instance:

   Follow the instructions in Get Started with Notebook Instances

   The Notebook instance created provides the required Python SDK for training, compiling and deploying models with Amazon SageMaker.

2. Compile a model using the Amazon SageMaker SDK:

   Refer to Supported Instances Types and Frameworks for information on the framework versions currently supported by Amazon SageMaker Neo on AWS Inferentia.

   More information about compiling and deploying models with Amazon SageMaker Neo can be found on Use Neo to Compile a Model

9.3 Bring Your Own Neuron Container to Sagemaker Hosting

Table of Contents

- Description
- Setup Environment
9.3.1 Description

You can use a SageMaker Notebook or an EC2 instance to compile models and build your own containers for deployment on SageMaker Hosting using ml.inf1 instances. In this developer flow, you provision a Sagemaker Notebook or an EC2 instance to train and compile your model to Inferentia. Then you deploy your model to SageMaker Hosting using the SageMaker Python SDK. Follow the steps below to setup your environment. Once your environment is set you’ll be able to follow the BYOC HuggingFace pretrained BERT container to Sagemaker Tutorial.

9.3.2 Setup Environment

1. Create a Compilation Instance: If using an EC2 instance for compilation you can use an Inf1 instance to compile and test a model. Follow these steps to launch an Inf1 instance:
   - Please follow the instructions at launch an Amazon EC2 Instance to Launch an Inf1 instance, when choosing the instance type at the EC2 console. Please make sure to select the correct instance type. To get more information about Inf1 instances sizes and pricing see Inf1 web page.
   - Select your Amazon Machine Image (AMI) of choice, please note that Neuron support Ubuntu 18 AMI or Amazon Linux 2 AMI, you can also choose Ubuntu 18 or Amazon Linux 2 Deep Learning AMI (DLAMI)
   - After launching the instance, follow the instructions in Connect to your instance to connect to the instance

   If using an SageMaker Notebook for compilation, follow the instructions in Get Started with Notebook Instances to provision the environment.

   It is recommended that you start with an ml.c5.4xlarge instance for the compilation. Also, increase the volume size of your SageMaker notebook instance, to accommodate the models and containers built locally. A volume of 10GB is sufficient.

   **Note:** To compile the model in the SageMaker Notebook instance, you’ll need to update the conda environments to include the Neuron Compiler and Neuron Framework Extensions. Follow the installation guide on the section How to update to latest Neuron packages in DLAMI Conda Environments? to update the environments.
2. **Set up the environment to compile a model, build your own container and deploy**: To compile your model on EC2 or SageMaker Notebook, follow the *Set up a development environment* section on the EC2 *Setup Environment* documentation.

Refer to *Adapting Your Own Inference Container* documentation for information on how to bring your own containers to SageMaker Hosting.

Make sure to add the `AmazonEC2ContainerRegistryPowerUser` role to your IAM role ARN, so you’re able to build and push containers from your SageMaker Notebook instance.

**Note:** The container image can be created using *How to Build a Neuron Container*.

### 9.4 Deploy Neuron Container on EC2

**Table of Contents**

- *Description*
- *Setup Environment*

#### 9.4.1 Description

You can use the Neuron version of the AWS Deep Learning Containers to run inference on inf1 instances. In this developer flow, you provision an EC2 inf1 instance using a Deep Learning AMI (DLAMI), pull the container image with the Neuron version of the desired framework, and run the container as a server for the already compiled model. This developer flow assumes the model has already been compiled through a *compilation developer flow*.
9.4.2 Setup Environment

1. **Launch an Inf1 Instance**
   - Please follow the instructions at launch an Amazon EC2 Instance to Launch an Inf1 instance, when choosing the instance type at the EC2 console. Please make sure to select the correct instance type. To get more information about Inf1 instances sizes and pricing see Inf1 web page.
   - Select your Amazon Machine Image (AMI) of choice, please note that Neuron support Ubuntu 18 AMI or Amazon Linux 2 AMI, you can also choose Ubuntu 18 or Amazon Linux 2 Deep Learning AMI (DLAMI)
   - After launching the instance, follow the instructions in Connect to your instance to connect to the instance

2. Once you have your EC2 environment set according to Docker environment setup, you can build and run a Neuron container using the How to Build a Neuron Container section above.

Note: Prior to running the container, make sure that the Neuron runtime on the instance is turned off, by running the command:

```
sudo service neuron-rtd stop
```
1. The model has already been compiled through *Compilation with Framework API on EC2 instance* or through *Compilation with Sagemaker Neo*.

2. You already set up your container to retrieve it from storage.

### 9.5.2 Setup Environment

1. **Set up an Amazon ECS cluster**: Follow the instructions on [Setting up Amazon ECS for Deep Learning Containers](#).

2. **Define an Inference Task**: Use the instruction on the [DLC Inference on ECS Tutorial](#) to define a task and create a service for the appropriate framework.

   When creating tasks for inf1 instances on ECS, be aware of the considerations and requirements listed in [Working with inference workloads on Amazon ECS](#).

3. Use the container image created using *How to Build a Neuron Container* as the image in your task definition.

   **Note:** Before deploying your task definition to your ECS cluster, make sure to push the image to ECR. Refer to [Pushing a Docker image](#) for more information.

### 9.6 Deploy Neuron Container on Elastic Kubernetes Service (EKS)

**Table of Contents**

- **Description**
- **Setup Environment**
  - Self-managed Kubernetes

#### 9.6.1 Description

![Diagram of EKS setup](image)

- **AWS Cloud**
- **EKS inf1 node 1**
- **EKS inf1 node 2**
- **Auto Scaling group**
- **Inference Service Load Balancer**
- **AWS EKS Control Plane**

**Kubernetes Pod**

- **Deep Learning Container**
- **Kubernetes Pod**
- **Deep Learning Container**

---

1072 Chapter 9. Developer Flows
You can use the Neuron version of the AWS Deep Learning Containers to run inference on Amazon Elastic Kubernetes Service (EKS). In this developer flow, you set up an EKS cluster with Inf1 instances, create a Kubernetes manifest for your inference service and deploy it to your cluster. This developer flow assumes:

1. The model has already been compiled through Compilation with Framework API on EC2 instance or through Compilation with Sagemaker Neo.
2. You already set up your container to retrieve it from storage.

### 9.6.2 Setup Environment

1. **Install pre-requisites:** Follow these instruction to install or upgrade the `eksctl` command line utility on your local computer.

   Follow these instruction to install `kubectl` in the same computer. `kubectl` is a command line tool for working with Kubernetes clusters.

2. **Follow the instructions in this EKS documentation link to set up AWS Inferentia on your EKS cluster.** Using the YML deployment manifest shown in the same link, replace the `image` in the `containers` specification with the one you built using How to Build a Neuron Container above.

   Before deploying your task definition to your EKS cluster, make sure to push the image to ECR. Refer to Pushing a Docker image for more information.

#### Self-managed Kubernetes

Please refer to tutorial-k8s-env-setup-for-neuron. In Deploy a TensorFlow Resnet50 model as a Kubernetes service, the container image referenced in the YML manifest is created using How to Build a Neuron Container.
CONTAINERS

It is recommended to deploy Neuron inside a preconfigured Deep Learning Container (DLC) from AWS. Running Neuron inside a container on Inf1 requires Docker version 18 (or newer) and a base AMI with aws-neuron-runtime-base and aws-neuron-dkms installed. It’s possible to also use a Neuron container on any instance type without the base and dkms package, but this is typically limited to compilation and development when running on instances without an Inf1 Device (inferentia). DLC images for Neuron can be obtained from here.

Documentation is organized based on the target deployment environment and use case. In most cases, it is recommended to use a preconfigured Deep Learning Container from AWS. Each DLC is pre-configured to have a recent version of Neuron components installed and is specific to the chosen ML Framework you want.

10.1 Quick Start

10.1.1 Docker environment setup

Introduction

A Neuron application can be deployed using docker containers. This tutorial describes how to configure docker to expose Inferentia devices to containers.

Once the environment is setup, a container can be started with `AWS_NEURON_VISIBLE_DEVICES` environment variable to specify desired set of Inferentia devices to be exposed to the container. `AWS_NEURON_VISIBLE_DEVICES` is a set of contiguous comma-separated inferentia logical ids. To find out the available logical ids on your instance, run the neuron-ls tool. For example, on inf1.6xlarge instance with 4 inferentia devices, you may set `AWS_NEURON_VISIBLE_DEVICES="2,3"` to expose the last two devices to a container. When running neuron-ls inside a container, you will only see the set of exposed Inferentias. For example:

```
docker run --env AWS_NEURON_VISIBLE_DEVICES="0" neuron-test neuron-ls
```

Would produce the following output:

```
+--------------+---------+--------+-----------+-----------+------+------+
| PCI BDF     | LOGICAL | NEURON | MEMORY   | MEMORY   | EAST | WEST |
| ID          | CORES   | CHANNEL 0 | CHANNEL 1 | | |
+--------------+---------+--------+-----------+-----------+------+------+
| 0000:00:1f.0 | 0       | 4      | 4096 MB   | 4096 MB   | 0    | 0    |
+--------------+---------+--------+-----------+-----------+------+------+
```
AWS Neuron

Steps:

This tutorial starts from a fresh Ubuntu 18

Step 1: Install Neuron driver and aws-neuron-runtime-base on the Linux host

Important: This step should run on the Linux host and not inside the container.

Configure Linux for Neuron repository updates, install Neuron driver and the aws-neuron-runtime-base package.

```bash
# Configure Linux for Neuron repository updates
. /etc/os-release
sudo tee /etc/apt/sources.list.d/neuron.list > /dev/null <<EOF
deb https://apt.repos.neuron.amazonaws.com ${VERSION_CODENAME} main
EOF
wget -qO - https://apt.repos.neuron.amazonaws.com/GPG-PUB-KEY-AMAZON-AWS-NEURON.PUB | sudo apt-key add -
# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$\(uname -r\) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

sudo apt-get install aws-neuron-runtime-base -y
```
Step 2: install oci-add-hooks dependency on the Linux host

**Important:** This step should run on the Linux host and not inside the container.

oci-add-hooks is an OCI runtime with the sole purpose of injecting OCI prestart, poststart, and poststop hooks into a container config.json before passing along to an OCI compatible runtime. oci-add-hooks is used to inject a hook that exposes Inferentia devices to the container.

```
sudo apt install -y golang &&
export GOPATH=$HOME/go &&
go get github.com/joeshaw/json-lossless &&
cd /tmp/ &&
git clone https://github.com/awslabs/oci-add-hooks &&
cd /tmp/oci-add-hooks &&
make build &&
sudo cp /tmp/oci-add-hooks/oci-add-hooks /usr/local/bin/
```

Step 3: setup Docker to use oci-neuron OCI runtime.

oci-neuron is a script representing OCI compatible runtime. It wraps oci-add-hooks, which wraps runc. In this step, we configure docker to point at oci-neuron OCI runtime. Install dockerIO:

```
sudo apt install -y docker.io
sudo usermod -aG docker $USER
```

Logout and log back in to refresh membership. Place daemon.json Docker configuration file supplied by Neuron SDK in default location. This file specifies oci-neuron as default docker runtime:

```
sudo cp /opt/aws/neuron/share/docker-daemon.json /etc/docker/daemon.json
sudo service docker restart
```

If the docker restart command fails, make sure to check if the docker systemd service is not masked. More information on this can be found here: [https://stackoverflow.com/a/37640824](https://stackoverflow.com/a/37640824)

Verify docker:

```
docker run hello-world
```

Expected result:

```
Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash
```

(continues on next page)
Build a docker image using provided dockerfile libmode-dockerfile, and use to verify whitelisting:

```
docker build . -f Dockerfile.app -t neuron-test
```

Then run:

```
docker run --env AWS_NEURON_VISIBLE_DEVICES="0" neuron-test neuron-ls
```

Expected result:

<table>
<thead>
<tr>
<th>PCI BDF</th>
<th>LOGICAL</th>
<th>NEURON</th>
<th>MEMORY</th>
<th>MEMORY</th>
<th>EAST</th>
<th>WEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000:00:1f.0</td>
<td>0</td>
<td>4</td>
<td>4096 MB</td>
<td>4096 MB</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### 10.1.2 How to Build a Neuron Container

#### Introduction

This document explains how to build a Neuron Container using an existing Dockerfile.

#### Pre-requisites

1. Docker version 18 or newer is configured according to *Docker environment setup*
2. Inf1 instance with available *Neuron Devices*
3. If running a serving application such as tensorflow-model-server, torchserve or multi-model-server, make sure the appropriate ports that the server listens to are exposed using EXPOSE in the Dockerfile or the arguments `-p 80:8080` on the `docker run` command.

#### Build and Run the Application Container

Follow the steps below for creating neuron application containers. If there were already existing containers that are packaged as per *Packaging Container Applications using Neuron Runtime 1.x* refer the *Migration to Neuron Runtime 2.x (libnrt.so)*

1. Build the container using libmode-dockerfile
2. Run the container locally:

```
docker run -it --name pt17 -p 80:8080 -e "AWS_NEURON_VISIBLE_DEVICES=ALL" neuron-→container:pytorch neuron-top
```
Important to know

Devices

There are currently two ways to specify Neuron Devices to a container.

1. The docker native way is to use `–device /dev/neuron#` for each of the Neuron Devices intended to be passed. When using `–device` option `ALL/all` is not supported.

   ```
   docker run --device=/dev/neuron0 --device=/dev/neuron1
   ```

2. If you install the `aws-neuron-runtime-base` package, you will have an OCI hook that also supports use of a container environment variable `AWS_NEURON_VISIBLE_DEVICES=<ALL | csv of devices>`, which intends to make things easier for multi device scenarios. Following are some examples

   ```
   docker run -e "AWS_NEURON_VISIBLE_DEVICES=0,1"
   docker run -e "AWS_NEURON_VISIBLE_DEVICES=ALL"
   ```

3. Multiple container applications running in the same host can share the devices but the cores cannot be shared. This is similar to running multiple applications in the host.

10.2 Deploy a Neuron Container

10.2.1 Deploy Neuron Container on EC2
AWS Neuron

Description

You can use the Neuron version of the AWS Deep Learning Containers to run inference on inf1 instances. In this developer flow, you provision an EC2 inf1 instance using a Deep Learning AMI (DLAMI), pull the container image with the Neuron version of the desired framework, and run the container as a server for the already compiled model. This developer flow assumes the model has already been compiled through a compilation developer flow.

Setup Environment

1. **Launch an Inf1 Instance**
   - Please follow the instructions at launch an Amazon EC2 Instance to Launch an Inf1 instance, when choosing the instance type at the EC2 console. Please make sure to select the correct instance type. To get more information about Inf1 instances sizes and pricing see Inf1 web page.
   - Select your Amazon Machine Image (AMI) of choice, please note that Neuron support Ubuntu 18 AMI or Amazon Linux 2 AMI, you can also choose Ubuntu 18 or Amazon Linux 2 Deep Learning AMI (DLAMI)
   - After launching the instance, follow the instructions in Connect to your instance to connect to the instance

2. Once you have your EC2 environment set according to Docker environment setup, you can build and run a Neuron container using the How to Build a Neuron Container section above.

Note: Prior to running the container, make sure that the Neuron runtime on the instance is turned off, by running the command:

```
sudo service neuron-rtd stop
```
10.2.2 Deploy Neuron Container on Elastic Container Service (ECS)

Table of Contents

- Description
- Setup Environment

Description

You can use the Neuron version of the AWS Deep Learning Containers to run inference on Amazon Elastic Container Service (ECS). In this developer flow, you set up an ECS cluster with inf1 instances, create a task description for your inference service and deploy it to your cluster. This developer flow assumes:

1. The model has already been compiled through Compilation with Framework API on EC2 instance or through Compilation with Sagemaker Neo.
2. You already set up your container to retrieve it from storage.

Setup Environment

1. **Set up an Amazon ECS cluster:** Follow the instructions on Setting up Amazon ECS for Deep Learning Containers.
2. **Define an Inference Task:** Use the instruction on the DLC Inference on ECS Tutorial to define a task and create a service for the appropriate framework.
   
   When creating tasks for inf1 instances on ECS, be aware of the considerations and requirements listed in Working with inference workloads on Amazon ECS.
3. Use the container image created using How to Build a Neuron Container as the image in your task definition.

**Note:** Before deploying your task definition to your ECS cluster, make sure to push the image to ECR. Refer to Pushing a Docker image for more information.
10.2.3 Deploy Neuron Container on Elastic Kubernetes Service (EKS)

Table of Contents

• Description
• Setup Environment
  – Self-managed Kubernetes

Description

You can use the Neuron version of the AWS Deep Learning Containers to run inference on Amazon Elastic Kubernetes Service (EKS). In this developer flow, you set up an EKS cluster with Inf1 instances, create a Kubernetes manifest for your inference service and deploy it to your cluster. This developer flow assumes:

1. The model has already been compiled through Compilation with Framework API on EC2 instance or through Compilation with Sagemaker Neo.
2. You already set up your container to retrieve it from storage.

Setup Environment

1. **Install pre-requisites:** Follow these instruction to install or upgrade the eksctl command line utility on your local computer.

   Follow these instruction to install kubectl in the same computer. kubectl is a command line tool for working with Kubernetes clusters.

2. **Follow the instructions in this EKS documentation link to set up AWS Inferentia on your EKS cluster.**

   Using the YML deployment manifest shown in the same link, replace the image in the containers specification with the one you built using How to Build a Neuron Container above.

   Before deploying your task definition to your EKS cluster, make sure to push the image to ECR. Refer to Pushing a Docker image for more information.
Self-managed Kubernetes

Please refer to tutorial-k8s-env-setup-for-neuron. In *Deploy a TensorFlow Resnet50 model as a Kubernetes service*, the container image referenced in the YML manifest is created using *How to Build a Neuron Container*.

10.2.4 Bring Your Own Neuron Container to Sagemaker Hosting

**Table of Contents**

- Description
- Setup Environment

**Description**

You can use a SageMaker Notebook or an EC2 instance to compile models and build your own containers for deployment on SageMaker Hosting using ml.inf1 instances. In this developer flow, you provision a SageMaker Notebook or an EC2 instance to train and compile your model to Inferentia. Then you deploy your model to SageMaker Hosting using the SageMaker Python SDK. Follow the steps below to setup your environment. Once your environment is set, you’ll be able to follow the *BYOC HuggingFace pretrained BERT container to Sagemaker Tutorial*.

**Setup Environment**

1. **Create a Compilation Instance**: If using an **EC2 instance for compilation** you can use an Inf1 instance to compile and test a model. Follow these steps to launch an Inf1 instance:
   
   - Please follow the instructions at [launch an Amazon EC2 Instance](#) to Launch an Inf1 instance, when choosing the instance type at the EC2 console. Please make sure to select the correct instance type. To get more information about Inf1 instances sizes and pricing see [Inf1 web page](#).
   
   - Select your Amazon Machine Image (AMI) of choice, please note that Neuron support Ubuntu 18 AMI or Amazon Linux 2 AMI, you can also choose Ubuntu 18 or Amazon Linux 2 Deep Learning AMI (DLAMI)
• After launching the instance, follow the instructions in Connect to your instance to connect to the instance

If using an SageMaker Notebook for compilation, follow the instructions in Get Started with Notebook Instances to provision the environment.

It is recommended that you start with an ml.c5.4xlarge instance for the compilation. Also, increase the volume size of you SageMaker notebook instance, to accommodate the models and containers built locally. A volume of 10GB is sufficient.

Note: To compile the model in the SageMaker Notebook instance, you’ll need to update the conda environments to include the Neuron Compiler and Neuron Framework Extensions. Follow the installation guide on the section How to update to latest Neuron packages in DLAMI Conda Environments? to update the environments.

2. Set up the environment to compile a model, build your own container and deploy: To compile your model on EC2 or SageMaker Notebook, follow the Set up a development environment section on the EC2 Setup Environment documentation.

Refer to Adapting Your Own Inference Container documentation for information on how to bring your own containers to SageMaker Hosting.

Make sure to add the AmazonEC2ContainerRegistryPowerUser role to your IAM role ARN, so you’re able to build and push containers from your SageMaker Notebook instance.

Note: The container image can be created using How to Build a Neuron Container.

10.3 Tutorials

10.3.1 Neuron Kubernetes Scheduler Extension

This document describes how the Neuron K8 scheduler extension works and how to use it in your cluster. The scheduler is required for scheduling pods that require more than one Neuron device resource. Please use this scheduler when working with inf1.6xlarge and inf1.24xlarge.

The k8-neuron-scheduler extends the default scheduler in these two ways:

1. Filter out nodes with non-contiguous device ids.
2. Enforces allocation of contiguous device ids for the PODs requiring it.

Flow Diagram

+-------------+
<p>| POD          |
| with         |
|aws.amazon.  |
| com/neuron:2 |
| Manifest     |</p>
<table>
<thead>
<tr>
<th>Request</th>
</tr>
</thead>
</table>
(continues on next page)
ENV: AWS_NEURON_VISIBLE_DEVICES: 2,3

1. neuron-device-plugin returns the list of Neuron devices to kublet
2. Kubelet advertises the Device list to K8s API server (in turn to kube-scheduler)
3. POD Request for neuron devices [Kube-Scheduler picks up the POD creation request]
4. kube-scheduler calls the neuron-scheduler-extn filter function with list of nodes and POD Specification
5. neuron-scheduler-extn scans through the nodes and filters out nodes with non contiguous devices and returns the nodes that are capable of supporting the given POD specification
6. kube-scheduler calls the neuron-scheduler-extn bind function with pod and node
7. neuron-scheduler-extn updates the POD annotation with allocated neuron device Ids (contiguous)
8. neuron-scheduler-extn sends the bind request to kublet of the selected node
9. Kubelet calls the Alloc function of the neuron-device-plugin
10. neuron-device-plugin queries the POD Annotation for allocated device Ids
11. neuron-device-plugin exports the visible devices to container runtime

Neuron Components

1. k8s-neuron-scheduler - scheduler extension that handles filter and bind request. ECR: 790709498068.dkr.ecr.us-east-1.amazonaws.com/neuron-scheduler:latest. It is deployed to a cluster using the provided:
   k8s-neuron-scheduler.yml
2. k8s-neuron-scheduler-configmap.yml - ConfigMap to register scheduler extension with Kube-scheduler.
3. k8s-neuron-device-plugin - manages neuron devices. ECR: 79070948068.dkr.ecr.us-east-1.amazonaws.com/neuron-device-plugin:latest. It is deployed to a cluster using the provided:
   k8s-neuron-device-plugin.yml
4. k8s-neuron-device-plugin-rbac.yml - configuration to enable permissions for device plugin to update the node and Pod annotations

Installation

For EKS, please follow the EKS documentation. If you are using Kops or similar follow these steps:

1. Enable the kube-scheduler with option to use configMap for scheduler policy. In your cluster.yml Please update the spec section with the following

   spec:
     kubeScheduler:
       usePolicyConfigMap: true

2. Launch the cluster

   kops create -f cluster.yml
   kops create secret --name neuron-test-1.k8s.local sshpublickey admin -i ~/.ssh/id_rsa.pub
   kops update cluster --name neuron-test-1.k8s.local --yes
3. Apply the k8s-neuron-scheduler-configmap.yml [Registers neuron-scheduler-extension with kube-scheduler]

```
kubectl apply -f k8s-neuron-scheduler-configmap.yml
```

4. Launch the neuron-scheduler-extension

```
kubectl apply -f k8s-neuron-scheduler.yml
```

5. Apply k8s-neuron-device-plugin-rbac.yml

```
kubectl apply -f k8s-neuron-device-plugin-rbac.yml
```

6. Apply the k8s-neuron-device-plugin.yml

```
kubectl apply -f k8s-neuron-device-plugin.yml
```

**Sample logs:**

<table>
<thead>
<tr>
<th>NAMESPACE</th>
<th>NAME</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>kube-system</td>
<td>dns-controller-865fd96754-s5x2p</td>
<td>1h</td>
</tr>
<tr>
<td></td>
<td>1 Running 0</td>
<td>12h</td>
</tr>
<tr>
<td>kube-system</td>
<td>etcd-manager-events-ip-172-20-92-213.us-west-2.compute.internal</td>
<td>1h</td>
</tr>
<tr>
<td></td>
<td>1 Running 0</td>
<td>12h</td>
</tr>
<tr>
<td>kube-system</td>
<td>etcd-manager-main-ip-172-20-92-213.us-west-2.compute.internal</td>
<td>1h</td>
</tr>
<tr>
<td></td>
<td>1 Running 0</td>
<td>12h</td>
</tr>
<tr>
<td>kube-system</td>
<td>k8s-neuron-scheduler-546bb6b45-k4x6s</td>
<td>1h</td>
</tr>
<tr>
<td></td>
<td>1 Running 0</td>
<td>11h</td>
</tr>
<tr>
<td>kube-system</td>
<td>kops-controller-h7t4s</td>
<td>1h</td>
</tr>
<tr>
<td></td>
<td>1 Running 0</td>
<td>12h</td>
</tr>
<tr>
<td>kube-system</td>
<td>kube-apiserver-ip-172-20-92-213.us-west-2.compute.internal</td>
<td>1h</td>
</tr>
<tr>
<td></td>
<td>1 Running 1</td>
<td>12h</td>
</tr>
<tr>
<td>kube-system</td>
<td>kube-controller-manager-ip-172-20-92-213.us-west-2.compute.internal</td>
<td>1h</td>
</tr>
<tr>
<td></td>
<td>1 Running 0</td>
<td>12h</td>
</tr>
<tr>
<td>kube-system</td>
<td>kube-dns-autoscaler-594dc84b5-bkgjl</td>
<td>1h</td>
</tr>
<tr>
<td></td>
<td>1 Running 0</td>
<td>12h</td>
</tr>
<tr>
<td>kube-system</td>
<td>kube-dns-b84c667f4-5qv86</td>
<td>3h</td>
</tr>
<tr>
<td></td>
<td>3 Running 0</td>
<td>12h</td>
</tr>
<tr>
<td>kube-system</td>
<td>kube-dns-b84c667f4-8x75m</td>
<td>3h</td>
</tr>
<tr>
<td></td>
<td>3 Running 0</td>
<td>12h</td>
</tr>
<tr>
<td>kube-system</td>
<td>kube-proxy-ip-172-20-75-104.us-west-2.compute.internal</td>
<td>1h</td>
</tr>
<tr>
<td></td>
<td>1 Running 0</td>
<td>11h</td>
</tr>
<tr>
<td>kube-system</td>
<td>kube-proxy-ip-172-20-92-213.us-west-2.compute.internal</td>
<td>1h</td>
</tr>
<tr>
<td></td>
<td>1 Running 0</td>
<td>11h</td>
</tr>
<tr>
<td>kube-system</td>
<td>kube-proxy-ip-172-20-95-42.us-west-2.compute.internal</td>
<td>1h</td>
</tr>
<tr>
<td></td>
<td>1 Running 0</td>
<td>12h</td>
</tr>
<tr>
<td>kube-system</td>
<td>kube-scheduler-ip-172-20-92-213.us-west-2.compute.internal</td>
<td>1h</td>
</tr>
<tr>
<td></td>
<td>1 Running 8</td>
<td>12h</td>
</tr>
<tr>
<td>kube-system</td>
<td>neuron-device-plugin-daemonset-7511q</td>
<td>1h</td>
</tr>
<tr>
<td></td>
<td>1 Running 0</td>
<td>11h</td>
</tr>
<tr>
<td>kube-system</td>
<td>neuron-device-plugin-daemonset-9wfnl</td>
<td>1h</td>
</tr>
<tr>
<td></td>
<td>1 Running 0</td>
<td>11h</td>
</tr>
</tbody>
</table>
10.4 Release Notes

10.4.1 What's New

Neuron K8 Release Notes

- **Introduction**

- **Neuron K8 release [1.7.3.0]**
  - [1.6.22.0]
  - [1.6.15.0]
  - [1.6.7.0]
  - [1.6.0.0]
  - [1.5.3.0]
  - [1.4.1.0]
  - [1.3.2.0]
  - [1.2.0.0]
  - [1.1.23.0]
  - [1.1.17.0]
  - [1.0.11000.0]

**Introduction**

This document lists the current release notes for AWS Neuron Kubernetes (k8) components. Neuron K8 components include a device plugin and a scheduler extension to assist with deployment and management of inf1 nodes within Kubernetes clusters. Both components are offered as pre-built containers in Public ECR and ready for deployment.

- **Device Plugin:** public.ecr.aws/neuron/neuron-device-plugin:1.6.22.0
- **Neuron Scheduler:** public.ecr.aws/neuron/neuron-scheduler:1.6.22.0

It's recommended to pin the version of the components used and to never use the “latest” tag. To get the list of image tags, please refer to these notes or check the image tags on the repo directly.

To Pull the Images from ECR:

```
docker pull public.ecr.aws/neuron/neuron-device-plugin:1.6.22.0
docker pull public.ecr.aws/neuron/neuron-scheduler:1.6.22.0
```
Neuron K8 release [1.7.3.0]

Date: 10/27/2021

Summary

Minor updates

[1.6.22.0]

Date: 08/30/2021

Summary

Minor updates.

[1.6.15.0]

Date: 08/06/2021

Summary

Minor updates.

[1.6.7.0]

Date: 07/26/2021

Summary

Minor internal enhancements.

[1.6.0.0]

Date: 07/02/2021
Summary

Minor internal enhancements.

[1.5.3.0]
Date: 05/01/2021

Summary

Minor internal enhancements.

[1.4.1.0]
Date: 01/30/2021

Summary

Minor internal enhancements.

[1.3.2.0]
Date: 12/23/2020

Summary

Minor internal enhancements.

[1.2.0.0]
Date: 11/17/2020

Summary

Minor internal enhancements.
[1.1.23.0]

Date: 10/22/2020

Summary


[1.1.17.0]

Date: 09/22/2020

Summary

Minor internal enhancements.

[1.0.11000.0]

Date: 08/08/2020

Summary

First release of the Neuron K8 Scheduler extension.

Major New Features

• New scheduler extension is provided to ensure that kubelet is scheduling pods on inf1 with contiguous device ids. Additional details about the new scheduler are provided Neuron Kubernetes Scheduler Extension, including instructions on how to apply it.
  – NOTE: The scheduler is only required when using inf1.6xlarge and/or inf1.24xlarge
• With this release the device plugin now requires RBAC permission changes to get/patch NODE/POD objects. Please apply the k8s-neuron-device-plugin-rbac.yml before using the new device plugin.

Resolved Issues

• Scheduler is intended to address https://github.com/aws/aws-neuron-sdk/issues/110
Neuron Containers Release Notes

- Neuron 1.16.0

Neuron 1.16.0

Date: 10/27/2021

New in this release

- Starting with Neuron 1.16.0, use of Neuron ML Frameworks now comes with an integrated Neuron Runtime as a library, as a result it is no longer needed to deploy neuron-rtd. Please visit Containers for more information.
- When using containers built with components from Neuron 1.16.0, or newer, please use aws-neuron-dkms version 2.1 or newer and the latest version of aws-neuron-runtime-base. Passing additional system capabilities is no longer required.

10.5 Resources for Neuron Runtime 1.x Users

10.5.1 Resources for Neuron Runtime 1.x Users

Migration to Neuron Runtime 2.x (libnrt.so)

Please refer this section only if neuron containers were already setup as per Packaging Container Applications using Neuron Runtime 1.x and are updating to the Neuron SDK software version 1.16.0 and beyond.

Application and Neuron Runtime in the same container

Follow the steps in Running Application Container

Application and Neuron Runtime in different container

1. Upgrade your application container as per Build and Run the Application Container section above.
2. With the Neuron Runtime library in the container there is no need to run a separate runtime container. You can now stop the runtime container.
Application in container and Neuron Runtime directly on host

1. Upgrade your application container as per Build and Run the Application Container section above.
2. With the Neuron Runtime library in the container there is no need to run the host runtime. You can now stop the host runtime with `sudo systemctl stop neuron-rtd` or `sudo killall neuron-rtd`.

Packaging Container Applications using Neuron Runtime 1.x

This document describes three ways to configure the Neuron SDK and run Neuron applications using containers. The goal is to ensure that the Neuron Runtime is able to control the Neuron Devices on the host instance and communicate with your application running inside a container.

The three different configurations enable you to chose between:
- Packaging your application and the runtime in a single container [Recommended].
- Packaging the application and runtime in separate containers.
- Packaging your application in a container and using the runtime from the host OS.

All three configurations are visualized below and can be implemented using docker alone on an EC2 Inf1 instance, or via a container orchestration services such as EKS and ECS.

Pre-requisites

1. Docker version 18 or newer is configured according to Docker environment setup for Neuron on EC2
2. Inf1 instance with available Neuron Devices
3. If running Runtime inside a container, system capability (docker run –cap-add IPC_LOCK). Refer Docker environment setup for Neuron on EC2
4. The UDS file that must be mounted and appropriate directories are setup
5. If running tensorflow-model-server/multi-model-server/torchserve etc then make sure appropriate ports that these servers are listening to are exposed using the EXPOSE in dockerfile or docker run -p 80:8080
Recommended - Packaging Application and Neuron Runtime in the same container

This is **recommended packaging mode**, as you benefit from:

- Not having to mount the UDS file from the host on to the container. The application and runtime use the default UDS local to the container.
- Full control of the Neuron Devices used by the application in the container.

1. Build the container using `app-rt-same-dockerfile`
2. The above docker file copies `dockerd-entrypoint-app-rt-same`
3. Run the containers

```
docker run -it --name pt17 -p 80:8080 --cap-add IPC_LOCK -e "AWS_NEURON_VISIBLE_DEVICES=ALL" neuron-container:pytorch neuron-top
```

**Note:** Since runtime is running inside the container the Neuron Devices needs to be mounted inside the container with the argument `-e “AWS_NEURON_VISIBLE_DEVICES=ALL”`

Alternative 1 - Packaging Application and Neuron Runtime in different container

This is **alternative packaging mode**. Should be used only when the recommended mode is not achievable, due to reasons such as:

- There are already application only/runtime only containers running.
- Would not want to mount devices to application container and localize them to the runtime container.
- Separate out the resources for application and runtime.

1. Build the runtime docker image using `neuron-runtime-dockerfile`
2. Run the runtime container

```
docker run --device=/dev/neuron0 --cap-add IPC_LOCK -v /run/:/run neuron-rtd
```

**Note:** Since runtime is running inside the container the Neuron Devices needs to be mounted inside the container with the argument `--device=/dev/neuron0`

3. Build the container using `app-rt-diff-dockerfile`
4. Run the application containers

```
docker run -it -v /run/:/run neuron-container:pytorch neuron-top
```

**Note:** Since runtime is not part of this container no need to mount Neuron Devices in this container
Alternative 2 - Packaging Application in container and Neuron Runtime directly on host

This is alternative packaging mode. Should be used only when the recommended mode is not achievable, due to reasons such as:

- Runtime is already running on the host
- Require multiple applications process and containers to access the runtime

1. Run the runtime software - refer Getting started: Installing and Configuring Neuron-RTD
2. Build the container using app-rt-diff-dockerfile
3. Run the application containers

```
docker run -it -v /run/:/run neuron-container:pytorch neuron-top
```

Note: Since runtime is not part of this container no need to mount neuron devices in this container

Important to know

Devices

There are currently two ways to specify Neuron Devices to a container.

1. The docker native way is to use –device /dev/neuron# for each of the Neuron Devices intended to be passed. When using –device option ALL/all is not supported.

```
docker run --device=/dev/neuron0 --device=/dev/neuron1
```

2. If you install the aws-neuron-runtime-base package, you will have an OCI hook that also supports use of a container environment variable AWS_NEURON_VISIBLE_DEVICES=<ALL | csv of devices>, which intends to make things easier for multi device scenarios. Following are some examples

```
docker run -e "AWS_NEURON_VISIBLE_DEVICES=0,1"
docker run -e "AWS_NEURON_VISIBLE_DEVICES=ALL"
```

UDS

1. The aws-neuron-runtime software is a grpc server (neuron-rtd) that listens on unix:/run/neuron.sock by default.

   Please refer the Neuron runtime that shows how the default can be changed.

2. The framework/app also by default sends grpc requests to uds unix:/run/neuron.sock. This can be changed by the environment variable NEURON_RTD_ADDRESS.

3. The docker run command below assumes the defaults are used. If using non-default uds then make the appropriate changes in the mount.

   - Default UDS

```
docker run -it neuron-container:pytorch
```

   - Non-default UDS - Mount /run in host to /tmp in container.
Neuron SDK Containers

Neuron provides several components in ECR that are updated on each release of the relevant Neuron component. When using these containers, you MUST specify the version you are interested in using. Avoid use of any “latest” tag as it will move without your approval. The latest tag is only provided in some cases for convenience.

Each ECR is maintained in only the us-east-1 and us-west-2 regions.

- Neuron Kubernetes Device Plugin:
  790709498068.dkr.ecr.<region>.amazonaws.com/neuron-device-plugin:<version>

- Neuron Kubernetes Scheduler Extension:
  790709498068.dkr.ecr.<region>.amazonaws.com/neuron-scheduler:<version>

- (deprecated) Neuron Runtime Deamon container, aka “the sidecar”:
  790709498068.dkr.ecr.<region>.amazonaws.com/neuron-rtd:<version>

Containers Tutorials & Examples

Docker environment setup for Neuron on EC2

Table of Contents

- Overview
- Run the tutorial
  - Step 1: Set up Neuron on the host instance
  - Step 2: Install and run docker daemon
  - Step 3: Build and Run a Neuron Docker image

Overview

This tutorial demonstrates how to configure Docker on an EC2 instance to expose Inferentia devices to containers.

By the end of this tutorial you will be able to run the Neuron Runtime inside a container and specify the desired set of Inferentia devices using the Docker flag `--device=/dev/neuron#`, where # is a available Neuron Device.

You will use an inf1.2xlarge to test your Docker configuration for Inferentia.

To find out the available neuron devices on your instance, use the command `ls /dev/neuron*`.

When running neuron-ls inside a container, you will only see the set of exposed Inferentias. For example:

```
docker run --device=/dev/neuron0 neuron-test neuron-ls
```

Would produce the following output:
Run the tutorial

Start by setting up an environment as described in Setup Environment section. While selecting the instance AMI you can select an Amazon Linux AMI, instead of the AWS Deep Learning AMI.

Step 1: Set up Neuron on the host instance

Once you are able to connect to your instance, start by installing the Neuron Kernel Modules as described in pytorch-pip-al2.

The neuron-rtd will run on the host after installation. Stop the neuron-rtd service before starting a containerized neuron-rtd. This is needed to allow assignment of devices to containers:

```bash
sudo service neuron-rtd stop
```

Step 2: Install and run docker daemon

Install Docker in your Amazon Linux 2 instance using the command:

```bash
sudo yum -y install docker
sudo usermod -aG docker $USER
```

If running on a Ubuntu AMI, refer to the official Docker installation documentation.

Logout and log back in to refresh membership. To verify your docker installation, run a simple `hello-work` Docker container with:

```bash
docker run hello-world
```

Expected result:

```
Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub. (amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash
```

(continues on next page)
AWS Neuron

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/

Step 3: Build and Run a Neuron Docker image

Using DockerFile for the runtime only

Build a docker image using provided dockerfile neuron-runtime-dockerfile and use to verify whitelisting:

Then run:

docker run --device=/dev/neuron0 neuron-test neuron-ls

Expected result:

| NEURON | NEURON | NEURON | CONNECTED | PCI | RUNTIME | RUNTIME | RUNTIME |
| DEVICE | CORES | MEMORY | DEVICES | BDF | ADDRESS | PID | VERSION |
|--------+--------+--------+-----------+--------------+---------+---------+---------|
| 0 | 4 | 8 GB | 1 | 0000:00:1c.0 | NA | 6 | NA |
|--------+--------+--------+-----------+--------------+---------+---------+---------|

Using DLC Neuron Image

Login to DLC repo with the following command:

aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 763104351884.dkr.ecr.us-east-1.amazonaws.com

Pull the docker image for your framework of choice. Images can be found here. The following examples pulls a TensorFlow Neuron inference image:

docker pull 763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-inference-neuron:1.15.5-neuron-py37-ubuntu18.04

After download finishes, the expected result for the command docker images is:

763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-inference-neuron:1.15.5-neuron-py37-ubuntu18.04 44c7584a6115 5 days ago 3.03GB

With the image available locally, you can Tag it and run it. The bellow example shows the docker run... command for the models used of tensorflow-serving tutorial.

docker tag 763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-inference-neuron:1.15.5-neuron-py37-ubuntu18.04 tf-dlc

docker run -it --name tf -p 8500:8500 --device=/dev/neuron0 --net=host --cap-add IPC_LOCK --mount type=bind,source=<saved_model_location>,target=/models/<model_name> -e MODEL_NAME=<model_name> tf-dlc
Building your own container

A sample Dockerfile for for torch-neuron can be found here torch-neuron-dockerfile. This Dockerfile requires an entrypoint script, to start the runtime with the `docker run` command.

Download the Dockerfile and the entrypoint script first, then build the image with the following command:

```
docker build . -f Dockerfile.torch-neuron -t torch-neuron
```

You can change the Neuron framework installation on the container and add your own application code, by modifying lines 35 to 40 on the torch-neuron-dockerfile.

Run the following command to execute `neuron-top` on the container:

```
docker run -it --device=/dev/neuron0 --cap-add IPC_LOCK torch-neuron neuron-top
```

Deploy a TensorFlow Resnet50 model as a Kubernetes service

This tutorial uses Resnet50 model as a teaching example on how to deploy an inference application using Kubernetes on the Inf1 instances.

Prerequisite:

- tutorial-k8s-env-setup-for-neuron: to setup k8s support on your cluster.
- Inf1 instances as worker nodes with attached roles allowing:
  - ECR read access policy to retrieve container images from ECR: `arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly`
  - S3 access to retrieve saved_model from within tensorflow serving container.

Deploy a TensorFlow Serving application image

A trained model must be compiled to an Inferentia target before it can be deployed on Inferentia instances. To continue, you will need a Neuron optimized TensorFlow model saved in Amazon S3. If you don’t already have a SavedModel, please follow the tutorial for creating a Neuron compatible ResNet50 model and upload the resulting SavedModel to S3.

ResNet-50 is a popular machine learning model used for image classification tasks. For more information about compiling Neuron models, see The AWS Inferentia Chip With DLAMI in the AWS Deep Learning AMI Developer Guide.

The sample deployment manifest manages a pre-built inference serving container for TensorFlow provided by AWS Deep Learning Containers. Inside the container is the AWS Neuron Runtime and the TensorFlow Serving application. A complete list of pre-built Deep Learning Containers optimized for Neuron is maintained on GitHub under Available Images. At start-up, the DLC will fetch your model from Amazon S3, launch Neuron TensorFlow Serving with the saved model, and wait for prediction requests.
AWS Neuron

The number of Neuron devices allocated to your serving application can be adjusted by changing the `aws.amazon.com/neuron` resource in the deployment yaml. Please note that communication between TensorFlow Serving and the Neuron runtime happens over GRPC, which requires passing the `IPC_LOCK` capability to the container.

1. Create a file named `rn50_deployment.yaml` with the contents below. Update the region-code and model path to match your desired settings. The model name is for identification purposes when a client makes a request to the TensorFlow server. This example uses a model name to match a sample ResNet50 client script that will be used in a later step for sending prediction requests.

Note:

1. Replace the s3 bucket name in `model_base_path` arg in the file with the location of the where the saved model was stored in s3.
2. In the image: add the appropriate location of the DLC tensorflow image

```yaml
kind: Deployment
apiVersion: apps/v1
metadata:
  name: k8s-neuron-test
  labels:
    app: k8s-neuron-test
    role: master
spec:
  replicas: 2
  selector:
    matchLabels:
      app: k8s-neuron-test
      role: master
  template:
    metadata:
      labels:
        app: k8s-neuron-test
        role: master
    spec:
      containers:
      - name: k8s-neuron-test
        image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-inference-neuron:1.15.4-neuron-py37-ubuntu18.04
        command:
          - /usr/local/bin/entrypoint.sh
        args:
          - --port=8500
          - --rest_api_port=9000
          - --model_name=resnet50_neuron
          - --model_base_path=s3://your-bucket-of-models/resnet50_neuron/
        ports:
          - containerPort: 8500
          - containerPort: 9000
        imagePullPolicy: IfNotPresent
        env:
          - name: AWS_REGION
            value: "us-east-1"
          - name: S3_USE_HTTPS
            value: "1"
          - name: S3_VERIFY_SSL
```

(continues on next page)
value: "0"
- name: S3_ENDPOINT
  value: s3.us-east-1.amazonaws.com
- name: AWS_LOG_LEVEL
  value: "3"

resources:
  limits:
    cpu: 4
    memory: 4Gi
  aws.amazon.com/neuron: 1
  requests:
    cpu: "1"
    memory: 1Gi

securityContext:
  capabilities:
    add:
    - IPC_LOCK

2. Deploy the model.

```bash
kubectl apply -f rn50_deployment.yaml
```

3. Create a file named `rn50_service.yaml` with the following contents. The HTTP and gRPC ports are opened for accepting prediction requests.

```yaml
customresourceDefinition
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: k8s-neuron-test
  labels:
    app.kubernetes.io/name: k8s-neuron-test
spec:
  scope: Namespaced
  group: k8s-neuron-test
  names:
    kind: Service
    plural: services
    shortName: svc
    singular: service

kind: Service
apiVersion: v1
metadata:
  name: k8s-neuron-test
  labels:
    app: k8s-neuron-test
spec:
  type: ClusterIP
  ports:
    - name: http-tf-serving
      port: 8500
      targetPort: 8500
    - name: grpc-tf-serving
      port: 9000
      targetPort: 9000
  selector:
    app: k8s-neuron-test
    role: master
```

4. Create a Kubernetes service for your TensorFlow model Serving application.

```bash
kubectl apply -f rn50_service.yaml
```
Make predictions against your TensorFlow Serving service

1. To test locally, forward the gRPC port to the `k8s-neuron-test` service.

   ```bash
   kubectl port-forward service/k8s-neuron-test 8500:8500 &
   ```

2. Create a Python script called `tensorflow-model-server-infer.py` with the following content. This script runs inference via gRPC, which is service framework.

   ```python
   import numpy as np
   import grpc
   import tensorflow as tf
   from tensorflow.keras.preprocessing import image
   from tensorflow.keras.applications.resnet50 import preprocess_input
   from tensorflow_serving.apis import predict_pb2
   from tensorflow_serving.apis import prediction_service_pb2_grpc
   from tensorflow.keras.applications.resnet50 import decode_predictions

   if __name__ == '__main__':
       channel = grpc.insecure_channel('localhost:8500')
       stub = prediction_service_pb2_grpc.PredictionServiceStub(channel)
       img = image.load_img(img_file, target_size=(224, 224))
       img_array = preprocess_input(image.img_to_array(img)[None, ...])
       request = predict_pb2.PredictRequest()
       request.model_spec.name = 'resnet50_inf1'
       request.inputs['input'].CopyFrom(
           tf.make_tensor_proto(img_array, shape=img_array.shape))
       result = stub.Predict(request)
       prediction = tf.make_ndarray(result.outputs['output'])
       print(decode_predictions(prediction))
   ```

3. Run the script to submit predictions to your service.

   ```bash
   python3 tensorflow-model-server-infer.py
   ```

   Your output should look like the following:

   ```python
   [[(u'n02123045', u'tabby', 0.68817204), (u'n02127052', u'lynx', 0.12701613), (u'n02123159', u'tiger_cat', 0.08736559), (u'n02124075', u'Egyptian_cat', 0.009240591), (u'n02128757', u'snow_leopard', 0.009240591)]]
   ```
11.1 Performance

11.1.1 General

Performance Tuning

**Important:** NeuronCore Groups (NCG) is deprecated, please see *End of support for NeuronCore Groups (NCG)* and *Migrate your application to Neuron Runtime 2.x (libnrt.so)* for more details.

This guide is intended to provide the reader with an in-depth understanding on how to optimize neural network performance on Inferentia for both throughput and latency. For simplicity, the guide uses TensorFlow and ResNet-50 model as a teaching example to learn how choosing between different compile-time optimizations (e.g. Batching and NeuronCore Pipeline), as well as model-serving optimizations (e.g. multi-threading and dynamic-batching) improves inference performance.

The following guides are considered prerequisites for this tutorial:

- *Running ResNet50 on Inferentia*
- tensorflow-serving-neurocore-group
- *Neuron Batching*
- *NeuronCore Pipeline*

**Batching and pipelining (technical background)**

Neuron provides developers with various performance optimization features.

Two of the most widely used features are batching and pipelining. Both techniques aim to keep the data close to the compute engines, but achieve this data locality in different ways. In batching it is achieved by loading the data into an on-chip cache and reusing it multiple times for multiple different model-inputs, while in pipelining this is achieved by caching all model parameters into the on-chip cache across multiple NeuronCores and streaming the calculation across them.

As a general rule of thumb, batching is preferred for applications that aim to optimize throughput and cost at the expense of latency, while pipelining is preferred for applications with high-throughput requirement under a strict latency budget.
Compiling for batching optimization

To enable the batching optimization, we first need to compile the model for a target batch-size. This is done by specifying the batch size in the input tensor’s batch dimension during compilation. Users are encouraged to evaluate multiple batch sizes in order to determine the optimal latency/throughput deployment-point, which is application dependent.

For example, the code snippet below enables batching on a ResNet50 model, with a batch-size of 5:

```python
import numpy as np
import tensorflow.neuron as tfn

# To change the batch size, change the first dimension in example_input
batch_size = 5
example_input = np.zeros([batch_size,224,224,3], dtype='float16')

tfn.saved_model.compile("rn50_fp16",
    "rn50_fp16_compiled/1",
    model_feed_dict={'input_1:0': example_input },
    dynamic_batch_size=True)
```

Note: Depending on the neural network size, Neuron will have a maximum batch size that works optimally on Inferentia. If an unsupported batch size is used, an internal compiler error message will be displayed. A simple way to explore optimal batch size for your specific model is to increment the batch size from 1 upward, one at a time, and test application performance.

Compiling for pipeline optimization

With NeuronCore Pipeline mode, Neuron stores the model parameters onto the Inferentias’ local caches, and streams the inference requests across the available NeuronCores, as specified by the `--neuroncore-pipeline-cores` compiler argument. For example, to compile the model to fit pipeline size of four Inferentia devices (16 NeuronCores) available in the inf1.6xlarge instance size:

```python
import numpy as np
import tensorflow.neuron as tfn

compiler_args = ['--neuroncore-pipeline-cores', '16']
example_input = np.zeros([1,224,224,3], dtype='float16')
tfn.saved_model.compile("rn50_fp16",
    "rn50_fp16_compiled/1",
    model_feed_dict={'input_1:0': example_input },
    compiler_args=compiler_args)
```

The minimum number of NeuronCores needed to run a compiled model can be found using Neuron Check Model tool. Please see Neuron Check Model.
Model-serving inference optimizations

In order to fully realize the maximum throughput of the compiled model (for either batching and pipelining), users need to launch multiple host CPU threads to feed inputs into the Neuron pipeline. The number of threads need to be larger than the specified maximum number of NeuronCores.

Additionally, dynamic batching (framework optimization currently supported only by TensorFlow-Neuron) can be used to process a larger client-side inference batch-size and the framework automatically breaks up the user-batch into smaller batch sizes to match the compiled batch-size. This technique increases the achievable throughput by hiding the framework-to-neuron overhead, and amortizing it over a larger batch size. To use dynamic batching, set the argument `--dynamic_batch_size=True` during compilation and send larger inference batch size (user inference batch size) that is equal to a multiple of the compiled batch size.

Both of methods can be applied together if that shows improvement in performance. However, multi-threading is always needed as a first step to achieve high throughput. You may need to experiment in order to find the right optimization settings for your application.

By default, the framework sets the number of outstanding inference requests to the total number of NeuronCores plus three. This can be changed by setting the NEURON_MAX_NUM_INFERS environment variable. For example, if the compiled model includes some CPU partitions (as when Neuron compiler decided some operations are more efficient to execute on CPU), the number of threads should be increased to account for the additional compute performed on the CPU. Note that the available instance host memory size should be taken into consideration to avoid out-of-memory errors. As above, you need to experiment in order to find the right optimization settings for your application.

---

**Note:** By default the framework allocates NeuronCore Group size to match the size of the compiled model. The size of the model is the number of NeuronCores limit passed to compiler during compilation (`--neuroncore-pipeline-cores` option). For more information see tensorflow-serving-neurocore-group.

---

Other considerations

Mixed Precision

You can find more details about performance and accuracy trade offs in *Mixed precision and performance-accuracy tuning*.

Operator support

The Neuron Compiler maintains an evolving list of supported operators for each framework: *Neuron Supported operators*

AWS Neuron handles unsupported operators by partitioning the graph into subgraph, and executing them on different targets (e.g. NeuronCore partition, CPU partition). If the entire model can run on Inferentia (i.e. all operators are supported), then the model will be compiled into a single subgraph which will be executed by a NeuronCore Group.
AWS Neuron

Debug

You can examine the post-compiled model to view the compilation results using the Neuron plugin for TensorBoard. See Visualize graphs executed on Neuron.

ResNet-50 optimization example

For an example demonstrating the concepts described here, see Tensorflow ResNet 50 Optimization Tutorial

Mixed precision and performance-accuracy tuning

- Neuron Hardware
- Performance-accuracy tradeoffs for models trained in FP32
- Compiler casting options
  - --fast-math option

The Neuron Compiler supports machine learning models with FP32, FP16 and BF16 (Bfloat16) tensors and operators. The Neuron hardware supports a mix of 32 and 16 bit datatypes. The available auto-cast methods and their performance / accuracy trade-offs are explained in this document.

Neuron Hardware

The Neuron hardware supports matrix multiplication using FP16 or BF16 on its Matmult Engine, and accumulations using FP32. Similarly, operators such as activations or vector operations are supported using FP16, BF16 and FP32. Neuron supports tensor transpose in two ways - by fast matrix multiplication in FP16/BF16 or by slower byte-by-byte data movements.

Performance-accuracy tradeoffs for models trained in FP32

Models that are trained using FP32 data types can be deployed on Neuron through ahead of time compilation using the Neuron Compiler.

By default, the Neuron Compiler will cast all FP32 tensors, weights and operations to BF16. Only partial sums are left in FP32. The default, casting will generate the highest performance for a FP32 trained model.

Using the --fast-math CLI option, you can choose the right tradeoff between performance and accuracy. The tradeoff usually is between achieving high performance or optimal accuracy, and decision what settings to use will be application specific.

It is recommended that the you start with compiling the model to achieve the high performance (default), you can then test the accuracy of the application and, if needed, try the next higher precision casting option until the desired accuracy and performance are achieved. A typical flow can be:

1. You can compile without options (default) or with --fast-math all which will optimize for performance.
2. If accuracy is not sufficient you can try --fast-math fp32-cast-matmult
3. If accuracy is not sufficient you can try --fast-math fp32-cast-matmult no-fast-relayout
4. If accuracy is not sufficient you can try --fast-math none which will optimize for accuracy.
Between step 2 and step 3, and between step 3 and step 4 you have additional options that can provide different level of accuracy and which are explained in the below section.

Note that compiler has to preserve the input/output (i/o) tensor types requested by Framework, therefore no casting is done on the i/o tensors. Additional speedup can be obtained by casting them in the Framework prior compilation.

To learn how to use compiler command line interface (CLI) options with your application’s framework, please see PyTorch-Neuron trace python API, TensorFlow-Neuron 1.x Compilation API and TensorFlow-Neuron 2.x Tracing API.

**Compiler casting options**

**--fast-math option**

The `--fast-math` option is intended to replace the `--fp32-cast` option. It is recommended to to start using or migrating to `--fast-math` option. The `--fast-math` option provides the same level of functionality as the `--fp32-cast` option in addition to the following:

- The `--fast-math` option introduces the `no-fast-relayout` option to enable lossless transpose operation. This was not possible with the `--fp32-cast` option.
- The `--fast-math` option provides finer control than the `--fp32-cast` option. The transpose operation and the cast operation are controlled independently:
  - `no-fast-relayout` and `fast-relayout` provide control for the transpose operation.
  - `fp32-cast-*` provide control for casting.

See the detailed list of the options in *Neuron compiler CLI Reference Guide*.

**Parallel Execution using NEURONCORE_GROUP_SIZES**

**Important:** NEURONCORE_GROUP_SIZES is being deprecated, if your application is using NEURONCORE_GROUP_SIZES please see *Migrate your application to Neuron Runtime 2.x (libnrt.so)* for more details.

**Introduction**

Inf1 instances are available with a different number of Inferentia chips, each Inferentia chip is combined of 4 NeuronCores and an Inf1 instance includes 4 to 64 NeuronCores depending on the instance size. NeuronCores can be combined into NeuronCore Groups (NCGs). This guide will show you how to load one or more compiled models into different NeuronCore Groups using your framework of choice.
Data Parallel Execution

The same compiled model can run in parallel on an Inf1 instance by loading it into separate NeuronCore Groups, thus setting up a data parallel execution. You can load multiple models into the same NCG, but only one of them will be active and execute inferences at any given time.

To define your NeuronCore Groups, you set the environment variable `NEURONCORE_GROUP_SIZES` with a comma separated list of number of cores in each group.

Running multiple models using single process

To run multiple models using single process, you set the environment variable `NEURONCORE_GROUP_SIZES` with a comma separated list of number of cores in each group.

You can set `NEURONCORE_GROUP_SIZES` environment variable at runtime:

```bash
#!/bin/bash
export NEURONCORE_GROUP_SIZES=2,4,3,4
python your_neuron_application.py
```

Or from within your python process running your models (NOTE: you can only set it once in the same process at the beginning of the script):

```python
#!/usr/bin/env python
import os

# Set Environment
os.environ['NEURONCORE_GROUP_SIZES']='2,4,3,4'

# Load models and run inferences ...
```

The above examples allow you to load 4 models into 4 NeuronCore Groups within one process. For example, if there are 4 models A, B, C, D compiled to 2, 4, 3, and 4 NeuronCores respectively, you directly load the models A, B, C, D in sequence within your TensorFlow or PyTorch Neuron process. This example requires an inf1.6xlarge instance with 16 NeuronCores, as the total number of NeuronCores within the NeuronCore Groups is 13.

In MXNet, the mapping from models to NeuronCore group is controlled by context `mx.neuron(device_id)` where `device_id` is the NeuronCore group ID. In the example above, you map model A to `mx.neuron(0)` context, model B to `mx.neuron(1)` context, model C to `mx.neuron(2)` context and model D to `mx.neuron(3)` context.

For PyTorch:

```python
# Set Environment
os.environ['NEURONCORE_GROUP_SIZES']='2,4,3,4'

# Load models (PT)
model0 = torch.jit.load(model0_file)  # loaded into the first group of NC0-NC1
model1 = torch.jit.load(model1_file)  # loaded into the second group of NC2-NC5
model2 = torch.jit.load(model2_file)  # loaded into the third group of NC6-NC8
model3 = torch.jit.load(model3_file)  # loaded into the fourth group of NC9-NC12

# run inference by simply calling the loaded model
results0 = model0(inputs0)
results1 = model1(inputs1)
results2 = model2(inputs2)
results3 = model3(inputs3)
```
For TensorFlow 2.x:

```python
# Set Environment
os.environ['NEURONCORE_GROUP_SIZES']='2,4,3,4'

# Load models (TF2)
model0 = tf.keras.models.load_model(model0_file)  # loaded into the first group of NC0-NC1
model1 = tf.keras.models.load_model(model1_file)  # loaded into the second group of NC2-NC5
model2 = tf.keras.models.load_model(model1_file)  # loaded into the third group of NC6-NC8
model3 = tf.keras.models.load_model(model1_file)  # loaded into the fourth group of NC9-NC12

# run inference by simply calling the loaded model
results0 = model0(inputs0)
results1 = model1(inputs1)
results2 = model2(inputs2)
results3 = model3(inputs3)
```

For MXNet 2.x:

```python
# Set Environment
os.environ['NEURONCORE_GROUP_SIZES']='2,4,3,4'

# Load models (MXNet)
# loaded into the first group of NC0-NC1
sym, args, aux = mx.model.load_checkpoint(mx_model0_file, 0)
model0 = sym.bind(ctx=mx.neuron(0), args=args, aux_states=aux, grad_req='null')  # loaded into the second group of NC2-NC5
sym, args, aux = mx.model.load_checkpoint(mx_model1_file, 0)
model1 = sym.bind(ctx=mx.neuron(1), args=args, aux_states=aux, grad_req='null')  # loaded into the third group of NC6-NC8
sym, args, aux = mx.model.load_checkpoint(mx_model2_file, 0)
model2 = sym.bind(ctx=mx.neuron(2), args=args, aux_states=aux, grad_req='null')  # loaded into the fourth group of NC9-NC12
sym, args, aux = mx.model.load_checkpoint(mx_model3_file, 0)
model3 = sym.bind(ctx=mx.neuron(3), args=args, aux_states=aux, grad_req='null')

# run inference by simply calling the loaded model
results0 = model0.forward(data=inputs0)
results1 = model1.forward(data=inputs1)
results2 = model2.forward(data=inputs2)
results3 = model3.forward(data=inputs3)
```

You can identify the NeuronCore Groups using the `neuron-cli` command line tool: [This example needs updating to show similar groups as defined above]

```
$ neuron-cli list-ncg
Device count 4 NC count 16
Found 4 NCG's
+--------+----------+--------------------+----------------+
| NCG ID | NC COUNT | DEVICE START INDEX | NC START INDEX |
+--------+----------+--------------------+----------------+
| 1 | 2 | 0 | 0 |
| 2 | 4 | 0 | 2 |
| 3 | 3 | 1 | 2 |
```

(continues on next page)
Running multiple models using multiple processes

You can also run multiple models in parallel processes, when you set `NEURONCORE_GROUP_SIZES` per process:

```
$ NEURONCORE_GROUP_SIZES=2 python your_1st_neuron_application.py
$ NEURONCORE_GROUP_SIZES=2 python your_2nd_neuron_application.py
```

The first process automatically selects a first set of 2 unused NeuronCores for its new group. The second process automatically selects a new set of 2 unused NeuronCores for its new group.
Running multiple models on the same NeuronCore Group

You can load more than one model in a NeuronCore Group within one process. The Neuron runtime will handle switching from one model to the next model within the NeuronCore Group when the next model is run within the application. In TensorFlow or PyTorch, simply load the additional models after the initial number of models have been loaded, to fill the NeuronCore Groups associated with the process.

For PyTorch:

```python
# Set Environment
os.environ['NEURONCORE_GROUP_SIZES']='2'

# Load models (PT)
model0 = torch.jit.load(model0_file)  # loaded into the first group of NC0-NC1
model1 = torch.jit.load(model1_file)  # loaded into the first group of NC0-NC1

# run inference by simply calling the loaded model
results0 = model0(inputs0)
results1 = model1(inputs1)
```

For TensorFlow 2.x:

```python
# Set Environment
os.environ['NEURONCORE_GROUP_SIZES']='2'

# Load models (TF2)
model0 = tf.keras.models.load_model(model0_file)  # loaded into the first group of NC0-NC1
model1 = tf.keras.models.load_model(model1_file)  # loaded into the first group of NC0-NC1

# run inference by simply calling the loaded model
results0 = model0(inputs0)
results1 = model1(inputs1)
```

In MXNet, use context `mx.neuron(neuroncore_group_id)` and use the same NeuronCore Group ID for the additional models. The additional models must have been compiled to fit into same or smaller NeuronCore Group size(s).

```python
# Set Environment
os.environ['NEURONCORE_GROUP_SIZES']='2'

# Load models (MXNet)
# loaded into the first group of NC0-NC1
sym, args, aux = mx.model.load_checkpoint(mx_model0_file, 0)
model0 = sym.bind(ctx=mx.neuron(0), args=args, aux_states=aux, grad_req='null')
# loaded into the first group of NC0-NC1
sym, args, aux = mx.model.load_checkpoint(mx_model1_file, 0)
model1 = sym.bind(ctx=mx.neuron(0), args=args, aux_states=aux, grad_req='null')

# run inference by simply calling the loaded model
results0 = model0.forward(data=inputs0)
results1 = model1.forward(data=inputs1)
```

The total `NEURONCORE_GROUP_SIZES` across all processes cannot exceed the number of NeuronCores visible to a framework (which is bound to the Neuron Runtime Daemon managing the Inferentias to be used). For example, on an inf1.xlarge with default configurations where the total number of NeuronCores visible to TensorFlow-Neuron is 4, you can launch one process with `NEURONCORE_GROUP_SIZES=2` (pipelined) and another process...
with \texttt{NEURONCORE\_GROUP\_SIZES=1,1} (data-parallel).

Examples using \texttt{NEURONCORE\_GROUP\_SIZES} include:

- \texttt{PyTorch example}
- \texttt{MXNet example}

**Auto Model Replication (Experimental for TensorFlow-Neuron only)**

The Auto Model Replication feature in TensorFlow-Neuron enables you to load the model once and the data parallel replication would happen automatically. This reduces framework memory usage as you are not loading the same model multiple times. This feature is experimental and available in TensorFlow-Neuron only.

To enable Auto Model Replication, set \texttt{NEURONCORE\_GROUP\_SIZES} to \texttt{Nx1} where \texttt{N} is the desired replication count (the number of NeuronCore groups, each group has size 1). For example, \texttt{NEURONCORE\_GROUP\_SIZES=8x1} would automatically replicate the single-NeuronCore model 8 times.

```python
os.environ[\'NEURONCORE\_GROUP\_SIZES\'] \= \'4x1\'
```

or

```bash
NEURONCORE\_GROUP\_SIZES=4x1 python3 application.py
```

When \texttt{NEURONCORE\_GROUP\_SIZES} is not set, the default is \texttt{4x1} where a single-NeuronCore model is replicated 4 times on any sized inf1 machine.

This feature is only available for models compiled with \texttt{neuroncore-pipeline-cores} set to 1 (default).

You will still need to use threads in the scaffolding code to feed the loaded replicated model instance in order to achieve high throughput.

Example of auto model replication: *Running OpenPose on Inferentia*

**FAQ**

**Can I mix data parallel and NeuronCore Pipeline?**

Yes. You can compile the model using \texttt{neuroncore-pipeline-cores} option. This tells the compiler to set compilation to the specified number of cores for \textit{NeuronCore Pipeline}. The Neuron Compiler will return a NEFF which fits within this limit. See the \textit{Neuron compiler CLI Reference Guide} on how to use this option.

For example, on an inf1.2xlarge, you can load two model instances, each compiled with \texttt{neuroncore-pipeline-cores} set to 2, so that they can run in parallel. The model instances can be loaded from different saved models or from the same saved model.
Can I have a mix of multiple models in one NCG and single model in another NCG?

Currently, you can do this in MXNet by setting up two NCGs, then load for example multiple models in one NCG using context mx.neuron(0), and load single model in the second NCG using context mx.neuron(1). You can also load single model in the first NCG and multiple models in the second NCG. For example:

```python
# Set Environment
os.environ['NEURONCORE_GROUP_SIZES']='2,4'

# Load models (MXNet)
# loaded into the first group of NC0-NC1
sym, args, aux = mx.model.load_checkpoint(mx_model0_file, 0)
model0 = sym.bind(ctx=mx.neuron(0), args=args, aux_states=aux, grad_req='null')

# loaded into the second group of NC2-NC5
sym, args, aux = mx.model.load_checkpoint(mx_model1_file, 0)
model1 = sym.bind(ctx=mx.neuron(1), args=args, aux_states=aux, grad_req='null')

# loaded into the second group of NC2-NC5
sym, args, aux = mx.model.load_checkpoint(mx_model2_file, 0)
model2 = sym.bind(ctx=mx.neuron(1), args=args, aux_states=aux, grad_req='null')

# loaded into the second group of NC2-NC5
sym, args, aux = mx.model.load_checkpoint(mx_model3_file, 0)
model3 = sym.bind(ctx=mx.neuron(1), args=args, aux_states=aux, grad_req='null')

# run inference by simply calling the loaded model
results0 = model0.forward(data=inputs0)
results1 = model1.forward(data=inputs1)
results2 = model2.forward(data=inputs2)
results3 = model3.forward(data=inputs3)
```

Loading multiple models in one NCG and single model in another NCG is currently not supported in TensorFlow and PyTorch.

### 11.1.2 PyTorch Neuron

**Running inference on variable input shapes with bucketing**

- **Introduction**
- **Applications that benefit from bucketing**
- **Implementing bucketing**
  - Creating bucketed models
  - Running inference with bucketing
- **Examples**
  - Computer vision bucketing
  - End-to-end computer vision bucketing example
  - Natural language processing bucketing
  - End-to-end natural language processing bucketing example
Introduction

With Inferentia, the shape of every input must be fixed at compile time. For applications that require multiple input sizes, we recommend using padding or bucketing techniques. Padding requires you to compile your model with the largest expected input size and pad every input to this maximum size. If the performance of your model using padding is not within your targets, you can consider implementing bucketing.

This guide introduces bucketing, a technique to run inference on inputs with variable shapes on Inferentia. The following sections explain how bucketing can improve the performance of inference workloads on Inferentia. It covers an overview of how bucketing works and provides examples of using bucketing in computer vision and natural language processing applications.

Applications that benefit from bucketing

Bucketing refers to compiling your model multiple times with different target input shapes to create “bucketed models.” Creating bucketed models provides an overview on selecting the input shapes that you use to create bucketed models. At inference time, each input is padded until its shape matches the next largest bucket shape. The padded input is then passed into the corresponding bucketed model for inference. By compiling the same model with multiple different input shapes, the amount of input padding is reduced compared to padding every input to the maximum size in your dataset. This technique minimizes the compute overhead and improves inference performance compared to padding every image to the maximum shape in your dataset.

Bucketing works best when multiple different bucketed models are created to efficiently cover the full range of input shapes. You can fine-tune the model performance by experimenting with different bucket sizes that correspond to the distribution of input shapes in your dataset.

Bucketing can only be used if there is an upper bound on the shape of the inputs. If necessary, an upper bound on the input shape can be enforced using resizing and other forms of preprocessing.

The upper bound on the number of bucketed models that you use is dictated by the total size of the compiled bucketed models. Each Inferentia chip has 8GB of DRAM, or 2GB of DRAM per NeuronCore. An inf1.xlarge and inf1.2xlarge have 1 Inferentia chip, an inf1.6xlarge has 4 Inferentia chips, and an inf1.24xlarge has 16 Inferentia chips. Thus, you should limit the total size of all bucketed models to around 8GB per Inferentia chip or 2GB per NeuronCore. The following formula provides an approximation for the number of compiled bucketed models you can fit on each NeuronCore:

\[
\text{number-of-buckets} = \text{round}(10^9 / \text{number-of-weights-in-model})
\]

We recommend using neuron-top to monitor the memory usage on your inf1 instance as you load multiple bucketed models.

Implementing bucketing

Implementing bucketing consists of two main parts: creating multiple bucketed models at compile-time and running inference using the bucketed models on (padded) inputs. The following sections describe how to implement bucketing to run inference in applications that have variable input shapes.
Creating bucketed models

Before running inference, models should be compiled for different input shapes that are representative of the input dataset. The input shapes that are used to compile the models determine the bucket shapes that are used during inference. The bucket shapes should be chosen to minimize the amount of padding on each new input. Additionally, there should always be a bucket that’s large enough to handle the maximum input shape in the dataset. The limit on the number of compiled bucketed models that can be used is described in this section.

Running inference with bucketing

At inference time, each input should be padded to match the size of the next largest bucket, such that the height and width (or sequence length) of the padded input equals the size of the bucket. Then, the padded input should be passed into the corresponding bucket for inference. If necessary, it’s important to remove and/or crop any aberrant predictions that occur in the padded region. For example, in object detection applications, bounding box predictions that occur in the padded regions should be removed to avoid erroneous predictions.

Examples

The following sections provide examples of applying the bucketing technique to run inference in applications that have variable input shapes.

Computer vision bucketing

As an example of implementing bucketing for computer vision models, consider an application where the height and width of images in dataset are uniformly distributed between [400, 400] and [800, 800]. Given that every input shape between [400, 400] and [800, 800] is equally likely, it could make sense to create bucketed models that divide up the range of input shapes into equally sized chunks. For example, we could create bucketed models for the input shapes [500, 500], [600, 600], [700, 700], and [800, 800].

As an example of running inference with bucketing, let’s assume that we created bucketed models for the input shapes [500, 500], [600, 600], [700, 700], and [800, 800]. If we receive an input with shape [640, 640], we would pad the input to the next largest bucket, [700, 700], and use this bucket for inference. If we receive an input with shape [440, 540], we would need to pad the input to the bucket size, [600, 600], and use this bucket for inference.

As another example of creating bucketed models, consider a computer vision application where the dataset is not uniformly distributed. As before, let’s assume the input shapes range between [400, 400] to [800, 800]. Now, let’s assume the data shape distribution is bimodal, such that [540, 540] and [720, 720] are the two most common input shapes. In this example, it might make sense to create bucketed models for input shapes [540, 540], [720, 720], and [800, 800] to target the most common shapes while still including the entire range of input shapes.

End-to-end computer vision bucketing example

In this example, we run inference in a computer vision application that has variable shaped images that range in shape from [400, 400] to [800, 800]. We create bucketed models for the input shapes [500, 500], [600, 600], [700, 700], and [800, 800] to handle the variable input shapes.

```python
import numpy as np
import torch
from torchvision import models
import torch_neuron
```
# Load the model and set it to evaluation mode
model = models.resnet50(pretrained=True)
model.eval()

# Define the bucket sizes that will be used for compilation and inference
bucket_sizes = [(500, 500), (600, 600), (700, 700), (800, 800)]

# Create the bucketed models by compiling a model for each bucket size
buckets = {}
for bucket_size in bucket_sizes:
    # Create an example input that is the desired bucket size
    h, w = bucket_size
    image = torch.rand([1, 3, h, w])
    # Compile with the example input to create the bucketed model
    model_neuron = torch.neuron.trace(model, image)
    # Run a warm up inference to load the model into Inferentia memory
    model_neuron(image)
    # Add the bucketed model based on its bucket size
    buckets[bucket_size] = model_neuron

def get_bucket_and_pad_image(image):
    # Determine which bucket size to use
    oh, ow = image.shape[-2:]
    target_bucket = None
    for bucket_size in bucket_sizes:
        # Choose a bucket that's larger in both the height and width dimensions
        if oh <= bucket_size[0] and ow <= bucket_size[1]:
            target_bucket = bucket_size
            break
    # Pad the image to match the size of the bucket
    h_delta = target_bucket[0] - oh
    w_delta = target_bucket[1] - ow
    b_pad = h_delta  # Bottom padding
    l_pad = 0  # Left padding
    t_pad = 0  # Top padding
    r_pad = w_delta  # Right padding

    # Pad the height and width of the image
    padding_amounts = (l_pad, r_pad, t_pad, b_pad)
    image_padded = torch.nn.functional.pad(image, padding_amounts, value=0)
    return image_padded, target_bucket

# Run inference on inputs with different shapes
for _ in range(10):
    # Create an image with a random height and width in range [400, 400] to [800, 800]
    h = int(np.random.uniform(low=400, high=800))
    w = int(np.random.uniform(low=400, high=800))
    image = torch.rand(1, 3, h, w)
Natural language processing bucketing

As an example of implementing bucketing for natural language processing models, consider an application where the lengths of tokenized sequences in a dataset are uniformly distributed between 0 and 128 tokens. Given that every tokenized sequence length between 0 and 128 is equally likely, it might make sense to create bucketed models that divide up the range of tokenized sequence lengths into equally sized chunks. For example, we could create bucketed models for tokenized sequence lengths 64 and 128.

As an example of running inference with bucketing, let’s assume that we created bucketed models for the input tokenized sequence lengths 64 and 128. If we receive a tokenized sequence with length 55, we would need to pad it to the bucket size 64 and use this bucket for inference. If we receive a tokenized sequence with length 112, we would need to pad it to the bucket size 128 and use this bucket for inference.

End-to-end natural language processing bucketing example

In this example, we run inference in a natural language processing application that has variable length tokenized sequences that range from 0 to 128. We create bucketed models for lengths 64 and 128 to handle the variable input lengths.

```python
import numpy as np
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import torch_neuron

# Build tokenizer and model
tokenizer = AutoTokenizer.from_pretrained("bert-base-cased-finetuned-mrpc")
model = AutoModelForSequenceClassification.from_pretrained("bert-base-cased-finetuned-mrpc", return_dict=False)
model.eval()

# Define the bucket sizes that will be used for compilation and inference
bucket_sizes = [64, 128]

# Create the bucketed models by compiling a model for each bucket size
buckets = {}
for bucket_size in bucket_sizes:
    # Setup some example inputs
    sequence_0 = "The company HuggingFace is based in New York City"
    sequence_1 = "HuggingFace’s headquarters are situated in Manhattan"

    # Create an example input that is the desired bucket size
    paraphrase = tokenizer.encode_plus(sequence_0,
                                         sequence_1,
                                         max_length=bucket_size,
                                         padding='max_length',
                                         truncation=True,
                                         return_tensors='pt')
```
return_tensors="pt")

# Convert example inputs to a format that is compatible with TorchScript tracing
example_inputs_paraphrase = paraphrase['input_ids'], paraphrase['attention_mask'],
  paraphrase['token_type_ids']

# Compile with the example input to create the bucketed model
model_neuron = torch.neuron.trace(model, example_inputs_paraphrase)

# Run a warm up inference to load the model into Inferentia memory
model_neuron(*example_inputs_paraphrase)

# Add the bucketed model based on its bucket size
buckets[bucket_size] = model_neuron

def get_bucket_and_pad_paraphrase(paraphrase):
  # Determine which bucket size to use
  inputs = paraphrase['input_ids']
  attention = paraphrase['attention_mask']
  token_type = paraphrase['token_type_ids']
  paraphrase_len = inputs.shape[1]
  target_bucket = None

  for bucket_size in bucket_sizes:
    if paraphrase_len <= bucket_size:
      target_bucket = bucket_size
      break

  # Pad the paraphrase to match the size of the bucket
  delta = target_bucket - paraphrase_len
  zeros = torch.zeros([1, delta], dtype=torch.long)
  inputs = torch.cat([inputs, zeros], dim=1)
  attention = torch.cat([attention, zeros], dim=1)
  token_type = torch.cat([token_type, zeros], dim=1)

  paraphrase_padded = inputs, attention, token_type
  return paraphrase_padded, target_bucket

# Create two sample sequences
sequence_0 = (
  "The only other bear similar in size to the polar bear is the "
  "Kodiak bear, which is a subspecies of the brown bear. Adult male "
  "polar bears weigh 350-700 kg and measure 2.4-3 meters in total "
  "length. All bears are short-tailed, the polar bear's tail is "
  "relatively the shortest amongst living bears."
)

sequence_1 = (
  "Around the Beaufort Sea, however, mature males reportedly "
  "average 450 kg. Adult females are roughly half the size of males "
  "and normally weigh 150-250 kg, measuring 1.8-2.4 meters in length. "
  "The legs are stocky and the ears and tail are small."
)

# Run inference on inputs with different shapes
# We create the variable shapes by randomly cropping the sequences
for _ in range(10):
  # Get random sequence lengths between 0 and 128
  paraphrase_len = int(np.random.uniform(128))

  # Crop the paraphrase
paraphrase_cropped = tokenizer.encode_plus(sequence_0,
sequence_1,
max_length=paraphrase_len,
padding='max_length',
truncation=True,
return_tensors="pt")

# Determine bucket and pad the paraphrase
paraphrase_padded, target_bucket = get_bucket_and_pad_paraphrase(paraphrase_cropped)

# Use the corresponding bucket to run inference
output = buckets[target_bucket](paraphrase_padded)

---

### Data Parallel Inference on Torch Neuron

- **Introduction**
- **Data parallel inference**
  - `torch.neuron.DataParallel`
    - NeuronCore selection
    - Batch dim
    - Dynamic batching
    - Performance optimizations
- **Examples**
  - Default usage
  - Specifying NeuronCores
  - `DataParallel` with dim != 0
  - Dynamic batching
  - Dynamic batching disabled
  - Full tutorial with `torch.neuron.DataParallel`

---

**Introduction**

This guide introduces `torch.neuron.DataParallel()`, a Python API that implements data parallelism on ScriptModule models created by the PyTorch-Neuron trace python API. The following sections explain how data parallelism can improve the performance of inference workloads on Inferentia, including how `torch.neuron.DataParallel()` uses dynamic batching to run inference on variable input sizes. It covers an overview of the `torch.neuron.DataParallel()` module and provides a few example data parallel applications.
Data parallel inference

Data Parallelism is a form of parallelization across multiple devices or cores, referred to as nodes. Each node contains the same model and parameters, but data is distributed across the different nodes. By distributing the data across multiple nodes, data parallelism reduces the total execution time of large batch size inputs compared to sequential execution. Data parallelism works best for smaller models in latency sensitive applications that have large batch size requirements.

torch.neuron.DataParallel

To fully leverage the Inferentia hardware, we want to use all available NeuronCores. An inf1.xlarge and inf1.2xlarge have four NeuronCores, an inf1.6xlarge has 16 NeuronCores, and an inf1.24xlarge has 64 NeuronCores. For maximum performance on Inferentia hardware, we can use `torch.neuron.DataParallel()` to utilize all available NeuronCores.

`torch.neuron.DataParallel()` implements data parallelism at the module level by replicating the Neuron model on all available NeuronCores and distributing data across the different cores for parallelized inference. This function is analogous to `DataParallel` in PyTorch. `torch.neuron.DataParallel()` requires PyTorch >= 1.8.

The following sections provide an overview of some of the features of `torch.neuron.DataParallel()` that enable maximum performance on Inferentia.

NeuronCore selection

By default, DataParallel will try to use all NeuronCores allocated to the current process to fully saturate the Inferentia hardware for maximum performance. It is more efficient to make the batch dimension divisible by the number of NeuronCores. This will ensure that NeuronCores are not left idle during parallel inference and the Inferentia hardware is fully utilized.

In some applications, it is advantageous to use a subset of the available NeuronCores for DataParallel inference. DataParallel has a `device_ids` argument that accepts a list of `int` or `nc:` that specify the NeuronCores to use for parallelization. See `Specifying NeuronCores` for an example of how to use `device_ids` argument.

Batch dim

DataParallel accepts a `dim` argument that denotes the batch dimension used to split the input data for distributed inference. By default, DataParallel splits the inputs on `dim = 0` if the `dim` argument is not specified. For applications with a non-zero batch dim, the `dim` argument can be used to specify the inference-time input batch dimension. `DataParallel with dim ! = 0` provides an example of data parallel inference on inputs with batch dim = 2.

Dynamic batching

Batch size has a direct impact on model performance. The Inferentia chip is optimized to run with small batch sizes. This means that a Neuron compiled model can outperform a GPU model, even if running single digit batch sizes.

As a general best practice, we recommend optimizing your model’s throughput by compiling the model with a small batch size and gradually increasing it to find the peak throughput on Inferentia.

Dynamic batching is a feature that allows you to use tensor batch sizes that the Neuron model was not originally compiled against. This is necessary because the underlying Inferentia hardware will always execute inferences with the batch size used during compilation. Fixed batch size execution allows tuning the input batch size for optimal
performance. For example, batch size 1 may be best suited for an ultra-low latency on-demand inference application, while batch size > 1 can be used to maximize throughput for offline inferencing. Dynamic batching is implemented by slicing large input tensors into chunks that match the batch size used during the `torch_neuron.trace()` compilation call.

The `torch.neuron.DataParallel()` class automatically enables dynamic batching on eligible models. This allows us to run inference in applications that have inputs with a variable batch size without needing to recompile the model. See Dynamic batching for an example of how DataParallel can be used to run inference on inputs with a dynamic batch size without needing to recompile the model.

Dynamic batching using small batch sizes can result in sub-optimal throughput because it involves slicing tensors into chunks and iteratively sending data to the hardware. Using a larger batch size at compilation time can use the Inferentia hardware more efficiently in order to maximize throughput. You can test the tradeoff between individual request latency and total throughput by fine-tuning the input batch size.

Automatic batching in the DataParallel module can be disabled using the `disable_dynamic_batching()` function as follows:

```python
>>> model_parallel = torch.neuron.DataParallel(model_neuron)
>>> model_parallel.disable_dynamic_batching()
```

If dynamic batching is disabled, the compile-time batch size must be equal to the inference-time batch size divided by the number of NeuronCores. DataParallel with dim != 0 and Dynamic batching disabled provide examples of running DataParallel inference with dynamic batching disabled.

### Performance optimizations

The DataParallel module has a `num_workers` attribute that can be used to specify the number of worker threads used for multithreaded inference. By default, `num_workers = 2 * number of NeuronCores`. This value can be fine tuned to optimize DataParallel performance.

DataParallel has a `split_size` attribute that dictates the size of the input chunks that are distributed to each Neuron-Core. By default, `split_size = max(1, input.shape[dim] // number of NeuronCores)`. This value can be modified to optimally match the inference input chunk size with the compile-time batch size.

### Examples

The following sections provide example usages of the `torch.neuron.DataParallel()` module.

### Default usage

The default DataParallel use mode will replicate the model on all available NeuronCores in the current process. The inputs will be split on dim=0.

```python
import torch
import torch_neuron
from torchvision import models

# Load the model and set it to evaluation mode
model = models.resnet50(pretrained=True)
model.eval()

# Compile with an example input
image = torch.rand([1, 3, 224, 224])
```
model_neuron = torch.neuron.trace(model, image)

# Create the DataParallel module
model_parallel = torch.neuron.DataParallel(model_neuron)

# Create a batched input
batch_size = 5
image_batched = torch.rand([batch_size, 3, 224, 224])

# Run inference with a batched input
output = model_parallel(image_batched)

Specifying NeuronCores

The following example uses the `device_ids` argument to use the first three NeuronCores for DataParallel inference.

```python
import torch
import torch_neuron
from torchvision import models

# Load the model and set it to evaluation mode
model = models.resnet50(pretrained=True)
model.eval()

# Compile with an example input
image = torch.rand([1, 3, 224, 224])
model_neuron = torch.neuron.trace(model, image)

# Create the DataParallel module, run on the first three NeuronCores
# Equivalent to model_parallel = torch.neuron.DataParallel(model_neuron, device_ids=[0, 1, 2])
model_parallel = torch.neuron.DataParallel(model_neuron, device_ids=['nc:0', 'nc:1', 'nc:2'])

# Create a batched input
batch_size = 5
image_batched = torch.rand([batch_size, 3, 224, 224])

# Run inference with a batched input
output = model_parallel(image_batched)
```

DataParallel with dim != 0

In this example we run DataParallel inference using four NeuronCores and `dim = 2`. Because `dim != 0`, dynamic batching is not enabled. Consequently, the DataParallel inference-time batch size must be four times the compile-time batch size. DataParallel will generate a warning that dynamic batching is disabled because `dim != 0`.

```python
import torch
import torch_neuron

# Create an example model
class Model(torch.nn.Module):
    def __init__(self):
```
super().__init__()
    self.conv = torch.nn.Conv2d(3, 3, 3)

def forward(self, x):
    return self.conv(x) + 1

model = Model()
model.eval()

# Compile with an example input
image = torch.rand([1, 3, 8, 8])
model_neuron = torch.neuron.trace(model, image)

# Create the DataParallel module using 4 NeuronCores and dim = 2
model_parallel = torch.neuron.DataParallel(model_neuron, device_ids=[0, 1, 2, 3], dim=2)

# Create a batched input
# Note that image_batched.shape[dim] / len(device_ids) == image.shape[dim]
batch_size = 4 * 8
image_batched = torch.rand([1, 3, batch_size, 8])

# Run inference with a batched input
output = model_parallel(image_batched)

## Dynamic batching

In the following example, we use the `torch.neuron.DataParallel()` module to run inference using several different batch sizes without recompiling the Neuron model.

```python
import torch
import torch_neuron
from torchvision import models

# Load the model and set it to evaluation mode
model = models.resnet50(pretrained=True)
model.eval()

# Compile with an example input
image = torch.rand([1, 3, 224, 224])
model_neuron = torch.neuron.trace(model, image)

# Create the DataParallel module
model_parallel = torch.neuron.DataParallel(model_neuron)

# Create batched inputs and run inference on the same model
batch_sizes = [2, 3, 4, 5, 6]
for batch_size in batch_sizes:
    image_batched = torch.rand([batch_size, 3, 224, 224])

    # Run inference with a batched input
    output = model_parallel(image_batched)
```

11.1. Performance
Dynamic batching disabled

In the following example, we use `torch.neuron.DataParallel.disable_dynamic_batching()` to disable dynamic batching. We provide an example of a batch size that will not work when dynamic batching is disabled as well as an example of a batch size that does work when dynamic batching is disabled.

```python
import torch
import torch_neuron
from torchvision import models

# Load the model and set it to evaluation mode
model = models.resnet50(pretrained=True)
model.eval()

# Compile with an example input
image = torch.rand([1, 3, 224, 224])
model_neuron = torch.neuron.trace(model, image)

# Create the DataParallel module and use 4 NeuronCores
model_parallel = torch.neuron.DataParallel(model_neuron, device_ids=[0, 1, 2, 3], dim=0)

# Disable dynamic batching
model_parallel.disable_dynamic_batching()

# Create a batched input (this won't work)
batch_size = 8
image_batched = torch.rand([batch_size, 3, 224, 224])

# This will fail because dynamic batching is disabled and
# image_batched.shape[dim] / len(device_ids) != image.shape[dim]
# output = model_parallel(image_batched)

# Create a batched input (this will work)
batch_size = 4
image_batched = torch.rand([batch_size, 3, 224, 224])

# This will work because
# image_batched.shape[dim] / len(device_ids) == image.shape[dim]
output = model_parallel(image_batched)
```

Full tutorial with torch.neuron.DataParallel

For an end-to-end tutorial that uses DataParallel, see the PyTorch Resnet Tutorial.
11.1.3 Apache MXNet Neuron

Flexible Execution Group (FlexEG) in Neuron-MXNet

Introduction

Inf1 instances are available with a different number of Inferentia chips, each Inferentia chip is combined of 4 Neuron-Cores and an Inf1 instance includes 4 to 64 NeuronCores depending on the instance size. With Neuron Runtime 1.x (neuron-rtd server), NeuronCores could be combined into NeuronCore Groups (NCG), which were basic scheduling units of compiled neural network in Neuron. Creation of desired sized NCGs was done at the start of the application and could not be modified afterwards.

Starting with Neuron SDK 1.16.0, and with the introduction of Neuron Runtime 2.x, Neuron MXNet 1.8 introduces Flexible Execution Groups (FlexEG) feature. With FlexEG, you do not have to create NCGs at the start of the process, instead you will set the index of the first NeuronCore you want to load models onto, and FlexEG feature will enable the flexibility of loading models onto any available NeuronCore on the inf1 instance starting from the first NeuronCore you set. This guide will show you how to efficiently utilize NeuronCores using FlexEG feature in NeuronMXNet.

FlexEG

With the introduction of FlexEG, you don’t need to create NCGs and can load models onto a group of consecutive NeuronCores by providing the index of the first NeuronCore in the group. Neuron runtime takes care of figuring out the number of NeuronCores required for the given compiled model and loads the model using the required number of cores (sequentially starting with the NeuronCore index provided by the user).

For example, assuming that you have an Inf1.6xl machine and there are 4 models A, B, C, D compiled to 2, 4, 3, and 4 NeuronCores respectively, you can map any model to any core by context

```python
mx.neuron(neuron_core_index)
```

where

```
neuron_core_index
```

is the NeuronCore index (0,1,2,3,4 ... ).

In the example below, you map model A to `mx.neuron(0)` context, model B to `mx.neuron(2)` context, model C to `mx.neuron(6)` context and model D to `mx.neuron(9)` context.

![Diagram of NeuronCores and models](image)

The above configuration is achieved by using application code similar to below:

```python
# Load models (MXNet)
# loaded into the 2 cores starting with core 0
sym, args, aux = mx.model.load_checkpoint(mx_model0_file, 0)
model0 = sym.bind(ctx=mx.neuron(0), args=args, aux_states=aux, grad_req='null')
```

(continues on next page)
Since there is no NCG creation at the start of the process, you can load the same four models but in a different configuration by changing the context being used for inference. For example, you could map model C to `mx.neuron(0)` context, model A to `mx.neuron(3)` context, model D to `mx.neuron(5)` context and model B to `mx.neuron(9)` context.

Migration from NeuronCore Groups to FlexEG

NeuronCore Groups are defined by setting the environment variable `NEURONCORE_GROUP_SIZES` with a comma separated list of number of cores in each group. In this mode of operation, number of devices (defined in `NEURONCORE_GROUP_SIZES`) are grouped together to create a single entity.

`NEURONCORE_GROUP_SIZES` environment variable is set at runtime:

```bash
#!/bin/bash
export NEURONCORE_GROUP_SIZES=2,4,3,4
python your_neuron_application.py
```

NeuronCore groups are created once at the start of the application and cannot be modified/re-created till the application process runs. The above flow creates 4 neuron devices with 2,4,3 and 4 devices each. In order to get the same configuration as the example from before, you map model A to `mx.neuron(0)` context, model B to `mx.neuron(1)` context, model C to `mx.neuron(2)` context and model D to `mx.neuron(3)` context.

This can be achieved programmatically as shown below:
So comparing to FlexEG, we see that in case of NCGs neuron context requires the index of the execution group, while in FlexEG neuron context requires the NeuronCore index of the first NeuronCore on which the model is supposed to be loaded and executed. For example, with `NEURONCORE_GROUP_SIZES='2,4,3,4'` loads the model on execution group 1 which effectively loads the model on the 2nd NCG group which has 4 Neuron-Cores.
**Best practices when using FlexEG**

FlexEG gives the user most flexibility in terms of accessing cores and loading models on specific cores. With this the users can effortlessly load and execute new models on NeuronCores without closing the application. Here we shall outline some of the best practices that should be kept in mind while using FlexEG.

**Choosing starting core**

FlexEG tries to use the required number of cores (based on the input model) starting with the core index provided by the user. Incase the system, doesnt have the required number of cores after the starting core index, model load will fail. For example: We have a model X which needs 2 cores and an inf1.xl machine with 4 NeuronCores (NeuronCore indexes are: 0, 1, 2 and 3). As the model needs at least 2 cores, valid start indexes for this model are: 0, 1, 2. However if the user gives 3 as the neuron context, then there are no 2 cores available starting from core 3. So it will fail.

**Performance vs. Flexibility tradeoff**

While using data parallel model of operation (were models are executed in parallel), for optimal performance the user should make sure that the models are not sharing any cores. That is because NeuronCores can execute one model at a time, when two or more models are executed on the same core (assuming that they are already loaded), it executes the first model, stops it, starts the second model and then executes it. This is called model switching and involves additional overhead and prevents execution on model in parallel. For example: assuming that you have an Inf1.6xl machine and there are 4 models A, B, C, D compiled to 2, 4, 3, and 4 NeuronCores respectively. Loading model A to \texttt{mx.neuron(0)} context, model B to \texttt{mx.neuron(2)} context, model C to \texttt{mx.neuron(6)} context and model D to \texttt{mx.neuron(9)} context is a good configuration because no two models are sharing NeuronCores and thus can be executed in parallel. However, Loading model A to \texttt{mx.neuron(0)} context, model B to \texttt{mx.neuron(2)} context, model C to \texttt{mx.neuron(5)} context and model D to \texttt{mx.neuron(9)} context is a not a good configuration as models B and C share NeuronCore 5 and thus cannot be executed in parallel.
11.2 Models

11.2.1 Inferentia Model Architecture Fit

- Introduction
- Model architectures that run well on Inferentia
  - Model enablement guidelines
    - Operator coverage
    - Variable input size
    - Control Flow
    - Dynamic shapes

Introduction

This section describes what types of deep learning Architectures perform well out of the box on Inferentia. It provides guidance on how Neuron maps operations to Inferentia, and discuss techniques you can use to optimize your deep learning models for Inferentia.

AWS Neuron, the SDK of Inferentia, enables you to deploy a wide range of pretrained deep learning models on AWS machine learning (ML) chips. Neuron includes a deep learning compiler, a runtime and tools natively integrated into popular ML frameworks like TensorFlow, PyTorch and Apache MXNet (Incubating).

Model architectures that run well on Inferentia

Many popular models used in today’s leading AI applications run out-of-the box on Inferentia. The following models are examples of model types that perform well on Inferentia:

- Language Models:
  - Transformers based Natural Language Processing/Understanding (NLP/NLU) such as HuggingFace Transformers BERT, distilBERT, XLM-BERT, Roberta and BioBert. To get started with NLP models you can refer to Neuron PyTorch, TensorFlow and MXNet NLP tutorials.
  - Generative language models like MarianMT, Pegasus and Bart.
- Computer Vision Models
  - Image classification models like Resnet, Resnext and VGG
  - Object detection models like Yolo v3/v4 and v5, and SSD
- Recommender engines models that include Embeddings and MLP layers.
Model enablement guidelines

The following points provide guidelines for deploying a model that doesn’t fit into one of the above categories or when deploying your own custom models. We encourage you to compile and run the model on Inferentia and contact us for support, if needed.

Operator coverage

Neuron has wide support for operator types for popular model types. That said, with Neuron Auto partition feature it is not required that all operators are supported by Neuron to successfully deploy a model on Inferentia.

Prior to compilation, the Neuron extension in the given Framework will examine the supported operators in the model and then partition the model graph, creating subgraph(s) that contain the unsupported operators that will execute within the framework on the CPU instance, or subgraph(s) that contain the supported operators that will execute within the accelerator on Inferentia.

While many models perform very well with subgraphs running on CPU, especially if the operations map well to CPU execution, it is possible that the performance will not meet your application needs. In such cases, we encourage you to contact us for further optimization.

Variable input size

With Neuron, the input size shape is fixed at compile time. If your application requires multiple input sizes, we recommend using padding or bucketing techniques. Padding requires you to compile your models to the largest expected input size, and test your application performance. If performance is not within your targets, you can consider implementing a bucketing scheme. With bucketing, you compile your model to a few input size categories that represent the range of possible input sizes. With some applications, bucketing will help optimize compute utilization compared to padding, especially if small input sizes are more frequent than large input sizes. If the varying input dimension is the batch size, dynamic batching can be used in TensorFlow-Neuron and PyTorch-Neuron to do inference using larger batch size than the compiled batch size, (see Neuron Batching).

Control Flow

Models that contain control flow operators (see Transformers MarianMT Tutorial) may require specific handling to ensure successful compilation with Neuron.

Dynamic shapes

Currently it is required that all tensor shapes (dimension sizes) in the compute-graph are known at compilation time. Model compilation with shapes that cannot be determined at compile time will fail.

For additional resources see:

- Neuron public roadmap
- Getting Started
- List of supported operators:
  - PyTorch supported operators
  - TensorFlow supported operators
  - MXNet supported operators
11.3 Neuron Components

11.3.1 Introducing Neuron Runtime 2.x (libnrt.so)

- What are we changing?
- Why are we making this change?
- How will this change affect the Neuron SDK?
  - Neuron Driver
  - Neuron Runtime
  - Neuron framework extensions
  - TensorFlow model server
  - Neuron tools
- How will this change affect me?
  - Neuron installation and upgrade
  - Migrate your application to Neuron Runtime 2.x (libnrt.so)
- Troubleshooting
  - Application fails to start
  - Application fails to start although I installed latest aws-neuron-dkms
  - Application unexpected behavior when upgrading to release Neuron 1.16.0 or newer
  - Application unexpected behavior when downgrading to releases before Neuron 1.6.0 (from Neuron 1.16.0 or newer)
  - Neuron Core is in use
- Frequently Asked Questions (FAQ)
  - Do I need to recompile my model to run it with Neuron Runtime 2.x (libnrt.so)?
  - Do I need to change my application launch command?
  - Can libnrt.so and neuron-rtd co-exist in the same environment?
  - Are there Neuron framework versions that will not support Neuron Runtime 2.x (libnrt.so)?

What are we changing?

Starting with Neuron 1.16.0 release, Neuron Runtime 1.x (neuron-rtd) is entering maintenance mode and is replaced by Neuron Runtime 2.x, a shared library named (libnrt.so). For more information on Runtime 1.x see Neuron Runtime 1.x (neuron-rtd) enters maintenance mode.

Upgrading to libnrt.so simplifies Neuron installation and upgrade process, introduces new capabilities for allocating NeuronCores to applications, streamlines container creation, and deprecates tools that are no longer needed.

This document describes the capabilities of Neuron Runtime 2.x in detail, provides information needed for successful installation and upgrade, and provides information needed for successful upgrade of Neuron applications using Neuron Runtime 1.x (included in releases before Neuron 1.16.0) to Neuron Runtime 2.x (included in releases Neuron 1.16.0 or newer).
Why are we making this change?

Before Neuron 1.16.0, Neuron Runtime was delivered as a daemon (neuron-rtd), and communicated with Neuron framework extensions through a gRPC interface. neuron-rtd was packaged as an rpm or debian package (aws-neuron-runtime) and required a separate installation step.

Starting with Neuron 1.16.0, Neuron Runtime 2.x is delivered as a shared library (libnrt.so) and is directly linked to Neuron framework extensions. libnrt.so is packaged and installed as part of Neuron framework extensions (e.g. Neuron TensorFlow, Neuron PyTorch or Neuron MXNet), and does not require a separate installation step. Installing Neuron Runtime as part of the Neuron framework extensions simplifies installation and improves the user experience. In addition, since libnrt.so is directly linked to Neuron framework extensions, it enables faster communication between the Neuron Runtime and Neuron Frameworks by eliminating the gRPC interface overhead.

For more information please see How will this change affect the Neuron SDK? and Migrate your application to Neuron Runtime 2.x (libnrt.so).

How will this change affect the Neuron SDK?

Neuron Driver

You need to use latest Neuron Driver. For successful installation and upgrade to Neuron 1.16.0 or newer, you must install or upgrade to Neuron Driver (aws-neuron-dkms) version 2.1.5.0 or newer. Neuron applications using Neuron 1.16.0 will fail if they do not detect Neuron Driver version 2.1.5.0 or newer. For installation and upgrade instructions see Setup Guide.

To see details of Neuron component versions please see Release Details.

Important:

For successful installation or update to Neuron 1.16.0 and newer from previous releases:

- Stop Neuron Runtime 1.x daemon (neuron-rtd) by running: sudo systemctl stop neuron-rtd
- Uninstall neuron-rtd by running: sudo apt remove aws-neuron-runtime or sudo yum remove aws-neuron-runtime
- Install or upgrade to latest Neuron Driver (aws-neuron-dkms) by following the Setup Guide instructions.

Neuron Runtime

- Installation Starting from Neuron 1.16.0, Neuron releases will no longer include the aws-neuron-runtime packages, and the Neuron Runtime will be part of the Neuron framework extension of choice (Neuron TensorFlow, Neuron PyTorch or Neuron MXNet). Installing any Neuron framework package will install the Neuron Runtime library (libnrt.so).
  - For installation and upgrade instructions see Setup Guide.
- Configuring Neuron Runtime Before Neuron 1.16.0, configuring Neuron Runtime 1.x was performed through configuration files (e.g. /opt/aws/neuron/config/neuron-rtd.config). Starting from Neuron 1.16.0, configuring Neuron Runtime 2.x can be done through environment variables, see Neuron Runtime Configuration for details.
• **Starting and Stopping Neuron Runtime** Before introducing libnrt.so, neuron-rtd ran as a daemon that communicated through a gRPC interface. Whenever neuron-rtd took ownership of a Neuron device, it continued owning that device until it was stopped. This created the need to stop neuron-rtd in certain cases. With the introduction of libnrt.so, stopping and starting the Neuron Runtime is no longer needed as it runs inside the context of the application. With Neuron Runtime 2.x, the act of starting and stopping a Neuron application will cause libnrt.so to automatically claim or release the ownership of the required Neuron devices.

• **NeuronCore Groups (NCG) deprecation** Before the introduction of Neuron Runtime 2.x, NeuronCore Group (NCG) has been used to define an execution group of one or more NeuronCores where models can be loaded and executed. It also provided separation between processes. With the introduction of Neuron Runtime 2.x, the strict separation of NeuronCores into groups is no longer needed and NeuronCore Groups (NCG) is deprecated. see End of support for NeuronCore Groups (NCG) for more information.

• **Running multiple Neuron Runtimes** Before the introduction of libnrt.so, you needed to run multiple neuron-rtd daemons to allocate Neuron devices for each neuron-rtd using configuration files. After the introduction of libnrt.so, you will no longer need to run multiple neuron-rtd daemons to allocate Neuron devices to specific Neuron application. With libnrt.so allocation of NeuronCores (Neuron device include multiple NeuronCores) to a particular application is done by using NEURON_RT_VISIBLE_CORES environment variable, for example:

```
NEURON_RT_VISIBLE_CORES=0-3 myapp1.py
NEURON_RT_VISIBLE_CORES=4-11 myapp2.py
```

See Neuron Runtime Configuration for details.

• **Logging** Similar to Neuron Runtime 1.x, Neuron Runtime 2.x logs to syslog (verbose logging). To make debugging easier, Neuron Runtime 2.x also logs to the console (error-only logging). Refer to Neuron Runtime Configuration to see how to increase or decrease logging verbosity.

• **Multi-process access to NeuronCores** With the introduction of libnrt.so, it’s no longer possible to load models on the same NeuronCore from multiple processes. Access to the same NeuronCore should be done from the same process. Instead you can load models on the same NeuronCore using multiple threads from the same process.

**Note:** For optimal performance of multi-model execution, each NeuronCore should execute single model.

• **Neuron Runtime architecture** Neuron Runtime 2.x is delivered as a shared library (libnrt.so) and is directly linked to Neuron framework extensions. libnrt.so is packaged and installed as part of Neuron framework extensions (e.g. Neuron TensorFlow, Neuron PyTorch or Neuron MXNet), and does not require a separate installation step. Installing Neuron Runtime as part of the Neuron framework extensions simplifies installation and improves the user experience. In addition, since libnrt.so is directly linked to Neuron framework extensions, it enables faster communication between the Neuron Runtime and Neuron Frameworks by eliminating the gRPC interface overhead.
Neuron framework extensions

Starting from Neuron 1.16.0, Neuron framework extensions (Neuron TensorFlow, Neuron PyTorch or Neuron MXNet) will be packaged together with libnrt.so. It is required to install the aws-neuron-dkms Driver version 2.1.5.0 or newer for proper operation. The neuron-rtd daemon that was installed in previous releases no longer works starting with Neuron 1.16.0.

To see details of Neuron component versions please see Release Details.

TensorFlow model server

Starting from Neuron 1.16.0, Neuron TensorFlow model server will be packaged together with libnrt.so and will expect aws-neuron-dkms version 2.1.5.0 or newer for proper operation.

Note: The Neuron TensorFlow model server included in Neuron 1.16.0 should run from the directory in which it was installed, as it will not run properly if copied to a different location due to its dependency on libnrt.so.

Neuron tools

• neuron-cli - Starting from Neuron 1.16.0, neuron-cli enters maintenance mode, see neuron-cli enters maintenance mode for more information.

• neuron-top - Starting from Neuron 1.16.0, neuron-top has a new user interface, see Neuron Top User Guide for more information.

• neuron-monitor - neuron-monitor was updated to support Neuron Runtime 2.x (libnrt.so)
  – See Migrating to Neuron Monitor 2.x for a list of changes between Neuron Monitor 2.x and Neuron Monitor 1.0
  – See Neuron Monitor 2.x Backwards Compatibility with Neuron Runtime 1.x for how you can use Neuron Monitor 2.x with Neuron Runtime 1.x (neuron-rtd).

How will this change affect me?

Neuron installation and upgrade

As explained in “How will this change affect the Neuron SDK?”, starting from Neuron 1.16.0, libnrt.so requires the latest Neuron Driver (aws-neuron-dkms), in addition there is no longer the need to install aws-neuron-runtime. To install Neuron or upgrade to latest Neuron version, please follow the installation and upgrade instructions below:

• Neuron PyTorch
  – Install Neuron PyTorch.
  – Update to latest Neuron PyTorch.

• Neuron TensorFlow
  – Install Neuron TensorFlow.
  – Update to latest Neuron TensorFlow.
• Neuron MXNet
  – Install Neuron MXNet.
  – Update to latest Neuron MXNet.

Migrate your application to Neuron Runtime 2.x (libnrt.so)

For a successful migration of your application to Neuron 1.16.0 or newer from previous releases, please make sure you perform the following:

1. **Prerequisite** Please read “How will this change affect the Neuron SDK?” section.

2. **Make sure you are not using Neuron Runtime 1.x (aws-neuron-runtime)**
   - Remove any code that install aws-neuron-runtime from any CI/CD scripts.
   - Stop neuron-rtd by running: `sudo systemctl stop neuron-rtd`
   - Uninstall neuron-rtd by running: `sudo apt remove aws-neuron-runtime` or `sudo yum remove aws-neuron-runtime`

3. **Upgrade to your Neuron Framework of choice:**
   - Update to latest Neuron PyTorch.
   - Update to latest Neuron TensorFlow.
   - Update to latest Neuron MXNet.

4. **If you have a code that start and/or stop neuron-rtd**
   Remove any code that start or stop neuron-rtd from any CI/CD scripts.

5. **Application running multiple neuron-rtd**
   If your application runs multiple processes and required running multiple neuron-rtd daemons:
   - Remove the code that runs multiple neuron-rtd daemons.
   - Instead of allocating N euron devices to neuron-rtd through configuration files, use `NEURON_RT_VISIBLE_CORES` environment variable to allocate NeuronCores. See **Neuron Runtime Configuration** for details.

   If your application uses `NEURONCORE_GROUP_SIZES`, see next item.

   **Note:** `NEURON_RT_VISIBLE_CORES` environment variable enables you to allocate NeuronCores to an application. Allocating NeuronCores improves application granularity because Neuron device include multiple NeuronCores.

6. **Application running multiple processes using NEURONCORE_GROUP_SIZES**
   - Please consider using `NEURON_RT_VISIBLE_CORES` introduced in Neuron 1.16.0 release instead of `NEURONCORE_GROUP_SIZES` as it is being deprecated,
   see **Neuron Runtime Configuration** for details.
   - Your application behavior will remain the same as before if you do not set `NEURON_RT_VISIBLE_CORES`.
   - If you are considering migrating to `NEURON_RT_VISIBLE_CORES`, please use the following guidelines:
– For TensorFlow applications or PyTorch applications make sure that
NEURONCORE_GROUP_SIZES is unset, or that NEURONCORE_GROUP_SIZES allocate
the same or less number of NeuronCores allocated by NEURON_RT_VISIBLE_CORES.
– For MXNet applications, setting NEURONCORE_GROUP_SIZES and
NEURON_RT_VISIBLE_CORES environment variables at the same time is not supported.
Please use NEURON_RT_VISIBLE_CORES only.

7. Application running multiple processes accessing same NeuronCore If your application accesses the same
NeuronCore from multiple processes, this is no longer possible with libnrt.so. Instead, please modify your application to access the same NeuronCore from multiple threads.

Note: For optimal performance of multi-model execution, each NeuronCore should execute a single model.

8. Neuron Tools

• If you are using Neuron Monitor, see Migrating to Neuron Monitor 2.x for details.
• If you are using neuron-cli please remove any call to neuron-cli. For more information, see
neuron-cli enters maintenance mode.

9. Containers If your application is running within a container, and it previously executed neuron-rtd
within the container, you need to re-build your container so it will not include or install
aws-neuron-runtime. See Containers and Migration to Neuron Runtime 2.x (libnrt.so) for details.

Troubleshooting

Application fails to start

Description

Starting from Neuron 1.16.0 release, Neuron Runtime (libnrt.so) requires Neuron Driver 2.0 or greater
(aws-neuron-dkms). Neuron Runtime requires Neuron Driver(aws-neuron-dkms package) to access Neuron
devices.

If aws-neuron-dkms is not installed then the application will fail with an error message on console and syslog that
look like the following:

```
NRT:nrt_init Unable to determine Neuron Driver version. Please check aws-neuron-
--dkms package is installed.
```

If an old aws-neuron-dkms is installed then the application will fail with an error message on console and syslog
that look like the following:

```
NRT:nrt_init This runtime requires Neuron Driver version 2.0 or greater. Please
--upgrade aws-neuron-dkms package.
```
Solution

Please follow the installation steps in Setup Guide to install `aws-neuron-dkms`.

**Application fails to start although I installed latest `aws-neuron-dkms`**

**Description**

Starting from *Neuron 1.16.0* release, Neuron Runtime (`libnrt.so`) require *Neuron Driver 2.0 or greater* (`aws-neuron-dkms`). If an old `aws-neuron-dkms` is installed, the application will fail. You may try to install `aws-neuron-dkms` and still face application failure, this may happen because the `aws-neuron-dkms` installation failed as a result of `neuron-rtd` daemon that is still running.

**Solution**

- **Stop** `neuron-rtd` by running: `sudo systemctl stop neuron-rtd`
- **Uninstall** `neuron-rtd` by running: `sudo apt remove aws-neuron-runtime or sudo yum remove aws-neuron-runtime`
- **Install** `aws-neuron-dkms` by following steps in Setup Guide

**Application unexpected behavior when upgrading to release *Neuron 1.16.0* or newer**

**Description**

When upgrading to release *Neuron 1.16.0* or newer from previous releases, the OS may include two different versions of *Neuron Runtime*: the `libnrt.so` shared library and `neuron-rtd` daemon. This can happen if the user didn't stop `neuron-rtd` daemon or didn't make sure to uninstall the existing Neuron version before upgrade. In this case the user application may behave unexpectedly.

**Solution**

If the OS includes two different versions of *Neuron Runtime*, `libnrt.so` shared library and `neuron-rtd` daemon:

- **Before running applications that use** `neuron-rtd`, **restart** `neuron-rtd` by calling `sudo systemctl restart neuron-rtd`.
- **Before running applications linked with** `libnrt.so`, **stop** `neuron-rtd` by calling `sudo systemctl stop neuron-rtd`.

11.3. Neuron Components 1137
Application unexpected behavior when downgrading to releases before Neuron 1.6.0 (from Neuron 1.16.0 or newer)

Description

When upgrading to release Neuron 1.16.0 or newer from previous releases, and then downgrading back to releases before Neuron 1.6.0, the OS may include two different versions of Neuron Runtime: the libnrt.so shared library and neuron-rtd daemon. This can happen if the user didn’t make sure to uninstall the existing Neuron version before upgrade or downgrade. In this case the user application may behave unexpectedly.

Solution

If the OS include two different versions of Neuron Runtime, libnrt.so shared library and neuron-rtd daemon:

- Before running applications that use neuron-rtd, restart neuron-rtd by calling sudo systemctl restart neuron-rtd.
- Before running applications linked with libnrt.so, stop neuron-rtd by calling sudo systemctl stop neuron-rtd.

Neuron Core is in use

Description

A Neuron Core can’t be shared between two applications. If an application started using a Neuron Core all other applications trying to use the NeuronCore would fail during runtime initialization with the following message in the console and in syslog:

```
ERROR  NRT:nrt_allocate_neuron_cores NeuronCore(s) not available - Requested:ncl-ncl Available:0
```

Solution

Terminate the the process using NeuronCore and then try launching the application again.

Frequently Asked Questions (FAQ)

Do I need to recompile my model to run it with Neuron Runtime 2.x (libnrt.so)?

No.
Do I need to change my application launch command?

No.

Can libnrt.so and neuron-rtd co-exist in the same environment?

Although we recommend upgrading to the latest Neuron release, we understand that for a transition period you may continue using neuron-rtd for old releases. If you are using Neuron Framework (PyTorch, TensorFlow or MXNet) from releases before Neuron 1.16.0:

- Install the latest Neuron Driver (aws-neuron-dkms)
- For development, we recommend using different environments for Neuron Framework (PyTorch, TensorFlow or MXNet) from releases before Neuron 1.16.0 and for Neuron Framework (PyTorch, TensorFlow or MXNet) from Neuron 1.16.0 and newer, if that is not possible, please make sure to stop neuron-rtd before executing models using Neuron Framework (PyTorch, TensorFlow or MXNet) from Neuron 1.16.0 and newer.
- For deployment, when you are ready to upgrade, please upgrade to Neuron Framework (PyTorch, TensorFlow or MXNet) from Neuron 1.16.0 and newer. see Migrate your application to Neuron Runtime 2.x (libnrt.so) for more information.

**Warning**: Executing models using Neuron Framework (PyTorch, TensorFlow or MXNet) from Neuron 1.16.0 and newer in an environment where neuron-rtd is running may cause undefined behavior. Please make sure to stop neuron-rtd before executing models using Neuron Framework (PyTorch, TensorFlow or MXNet) from Neuron 1.16.0 and newer.

Are there Neuron framework versions that will not support Neuron Runtime 2.x (libnrt.so)?

All supported Neuron PyTorch and TensorFlow framework extensions in addition to Neuron MXnet 1.8.0 framework extensions support Neuron Runtime 2.x.

Neuron MxNet 1.5.1 does not support Neuron Runtime 2.x (libnrt.so) and has now entered maintenance mode. Please see Neuron support for Apache MXNet 1.5 enters maintenance mode for details.
12.1 General Neuron FAQs

- What ML models types and operators are supported by AWS Neuron?
- Why is a compiler needed, and how do I use it?
- I am using a ML framework today – what will change for me to use this?
- What is a NeuronCore Pipeline? and How do I take advantage of it?
- NeuronCores, NeuronCore Groups and NeuronCore Pipelines: What do they do?
- Can I use TensorFlow networks from tfhub.dev as-is? if not, what should I do?

12.1.1 What ML models types and operators are supported by AWS Neuron?

AWS Neuron includes a compiler that converts your trained machine learning models to a binary object for execution (aka Neuron Executable File Format or a NEFF file in short). The Neuron compiler supports many commonly used machine learning operators used in computer vision, natural language processing, recommender engines and more. A list of supported ML operators and supported inputs are in Neuron Supported operators.

It’s important to mention that to get good performance doesn’t require all of the model operators to run on the chip. In many cases, some of the operators will continue to run on the instance CPUs, like the case of embeddings or image pre-processing, and will still provide a compelling end to end performance. We call this approach auto-partitioning, where the Neuron compiler optimizes the model execution based on operators that are most suitable to run on the CPU or the chip.

We constantly add more operators based on customers’ feedback.

12.1.2 Why is a compiler needed, and how do I use it?

The Neuron compiler converts from a framework level Neural Network graph, with operators like convolution and pooling, into a hardware-specific instruction set, builds the schedule for execution of these instructions, and converts the model parameters into format that the chip can consume. The supported input formats include TensorFlow, PyTorch, and MXNet. The output from the compiler is a Neuron Executable File Format (NEFF) artifact. The NEFF contains a combination of binary code, the model parameters, and additional meta-data needed by the Neuron runtime and profiler.
12.1.3 I am using a ML framework today – what will change for me to use this?

To use Inferentia within the Inf1 instances, the developer need to perform one-time compilation of the pre-trained model to generate a NEFF, and use this as the inference model in fleet of Inf1 instances.

- TensorFlow Neuron
- PyTorch Neuron
- Neuron Apache MXNet (Incubating)

12.1.4 What is a NeuronCore Pipeline ? and How do I take advantage of it?

A NeuronCore Pipeline is a unique technique to shard a specific Neural Network across multiple NeuronCores, to take advantage of the large on-chip cache instead of moving data in and out of external memory. The result is an increased throughput and reduce latency typically important for real-time inference applications. All Inf1 instances support it, and the Inf1 instances with multiple Inferentia accelerators, such as inf1.6xlarge or inf1.24xlarge support it thanks to the fast chip-to-chip interconnect.

Developers can choose to use NeuronCore Pipeline mode during compile stage, with an opt-in flag. Neuron Compiler provides further details.

12.1.5 NeuronCores, NeuronCore Groups and NeuronCore Pipelines: What do they do?

Each Inferentia chip has four compute engines called NeuronCores. A NeuronCore Group is a way to aggregate NeuronCores to increase hardware utilization and assign models with the right compute sizing for a specific application. If you want to run multiple models in parallel, you can assign different models to separate NeuronCore Groups. A model compiled to use multiple NeuronCores in a NeuronCore Pipeline can be assigned to a NeuronCore Group with enough NeuronCores to load into. Finally- it is also possible for sets of Inferentia devices to be mapped to separate Neuron Runtimes. Neuron Features section has more information and examples.

12.1.6 Can I use TensorFlow networks from tfhub.dev as-is ? if not, what should I do?

Yes. Models format can be imported into TensorFlow, either as a standard model-server, in which case it appears as a simple command line utility, or via the Python based TensorFlow environment. The primary additional step needed is to compile the model into Inferentia NEFF format.

12.2 Getting started with Neuron FAQs

- How can I get started?
- How do I select which Inf1 instance size to use?
12.2.1 How can I get started?

You can start your workflow by training your model in one of the popular ML frameworks using EC2 GPU instances, or alternatively download a pre-trained model. Once the model is trained to your required accuracy, you can use the ML frameworks’ API to invoke Neuron, to re-target (i.e. compile) the model for execution on Inferentia. The compilation is done once and its artifacts can then be deployed at scale. Once compiled, the binary can be loaded into one or more chips to start service inference calls.

In order to get started quickly, you can use AWS Deep Learning AMIs that come pre-installed with ML frameworks and the Neuron SDK. For a fully managed experience, you can use Amazon SageMaker to seamlessly deploy and accelerate your production models on ml.inf1 instances.

For customers who use popular frameworks like TensorFlow, Apache MXNet (Incubating) and PyTorch, a guide to help you get started with frameworks is available at:

- TensorFlow Neuron
- PyTorch Neuron
- Neuron Apache MXNet (Incubating)

You can also visit QuickStart.

12.2.2 How do I select which Inf1 instance size to use?

The decision as to which Inf1 instance size to use is based upon the application and its performance/cost targets. To assist, the Neuron Plugin for TensorBoard will show actual results when executed on a given instance. A guide to this process is available here: Neuron Plugin for TensorBoard.

As a rule of thumb, we encourage you to start with inf1.xlarge and test your model. For example, many computer vision models require pre/post processing that consume CPU resources and such models will get higher throughput on the inf1.2xlarge that provides higher ratio of vCPU/Chip.

We encourages you try out all the Inf1 instance sizes with your specific models, until you find the best size for your application needs.

12.3 Troubleshooting FAQs

- Performance is not what I expect it to be, what’s the next step?
- Do I need to worry about size of model and size of inferentia memory? what problems can I expect to have?
- How can I debug / profile my inference request?
- How to reporting Bugs/Feature Requests
- Contributing via Pull Requests
- How to find contributions to work on
- What is the code of conduct
- How to notify for a security issue
- What is the licensing
12.3.1 Performance is not what I expect it to be, what’s the next step?

Please check our performance-optimization section on performance tuning and other notes on how to use pipelining and batching to improve performance!

12.3.2 Do I need to worry about size of model and size of inferentia memory? what problems can I expect to have?

Errors like this will be logged and can be found as shown *Using Neuron GatherInfo Tool to collect debug and support information.*

12.3.3 How can I debug / profile my inference request?

See *Neuron Plugin for TensorBoard*

**Contributing Guidelines FAQs**

Whether it’s a bug report, new feature, correction, or additional documentation, we greatly value feedback and contributions from our community.

Please read through this document before submitting any issues or pull requests to ensure we have all the necessary information to effectively respond to your bug report or contribution.

12.3.4 How to reporting Bugs/Feature Requests

We welcome you to use the GitHub issue tracker to report bugs or suggest features.

When filing an issue, please check existing open, or recently closed, issues to make sure somebody else hasn’t already reported the issue. Please try to include as much information as you can. Details like these are incredibly useful:

- A reproducible test case or series of steps
- The version of our code being used
- Any modifications you’ve made relevant to the bug
- Anything unusual about your environment or deployment

12.3.5 Contributing via Pull Requests

Contributions via pull requests are much appreciated. Before sending us a pull request, please ensure that:

1. You are working against the latest source on the *master* branch.
2. You check existing open, and recently merged, pull requests to make sure someone else hasn’t addressed the problem already.
3. You open an issue to discuss any significant work - we would hate for your time to be wasted.

To send us a pull request, please:

1. Fork the repository.
2. Modify the source; please focus on the specific change you are contributing. If you also reformat all the code, it will be hard for us to focus on your change.
3. Ensure local tests pass.
4. Commit to your fork using clear commit messages.
5. Send us a pull request, answering any default questions in the pull request interface.
6. Pay attention to any automated CI failures reported in the pull request, and stay involved in the conversation.

GitHub provides additional documentation on forking a repository and creating a pull request.

### 12.3.6 How to find contributions to work on

Looking at the existing issues is a great way to find something to contribute on. As our projects, by default, use the default GitHub issue labels (enhancement/bug/duplicate/help wanted/invalid/question/wontfix), looking at any ‘help wanted’ issues is a great place to start.

### 12.3.7 What is the code of conduct

This project has adopted the Amazon Open Source Code of Conduct. For more information see the Code of Conduct FAQ or contact opensource-codeofconduct@amazon.com with any additional questions or comments.

### 12.3.8 How to notify for a security issue

If you discover a potential security issue in this project we ask that you notify AWS/Amazon Security via our vulnerability reporting page. Please do **not** create a public github issue.

### 12.3.9 What is the licensing

See the `https://github.com/aws/aws-neuron-sdk/blob/master/LICENSE-DOCUMENTATION` and `https://github.com/aws/aws-neuron-sdk/blob/master/LICENSE-SUMMARY-DOCS-SAMPLES` files for our project’s licensing. We will ask you to confirm the licensing of your contribution.

We may ask you to sign a Contributor License Agreement (CLA) for larger changes.

### 12.4 Contributing Guidelines FAQs

- **How to report Bugs/Feature Requests**
- **Contributing via Pull Requests**
- **How to find contributions to work on**
- **What is the code of conduct**
- **How to notify for a security issue**
- **What is the licensing**

Whether it’s a bug report, new feature, correction, or additional documentation, we greatly value feedback and contributions from our community.

Please read through this document before submitting any issues or pull requests to ensure we have all the necessary information to effectively respond to your bug report or contribution.
12.4.1 How to report Bugs/Feature Requests

We welcome you to use the GitHub issue tracker to report bugs or suggest features. When filing an issue, please check existing open, or recently closed, issues to make sure somebody else hasn’t already reported the issue. Please try to include as much information as you can. Details like these are incredibly useful:

- A reproducible test case or series of steps
- The version of our code being used
- Any modifications you’ve made relevant to the bug
- Anything unusual about your environment or deployment

12.4.2 Contributing via Pull Requests

Contributions via pull requests are much appreciated. Before sending us a pull request, please ensure that:

1. You are working against the latest source on the master branch.
2. You check existing open, and recently merged, pull requests to make sure someone else hasn’t addressed the problem already.
3. You open an issue to discuss any significant work - we would hate for your time to be wasted.

To send us a pull request, please:

1. Fork the repository.
2. Modify the source; please focus on the specific change you are contributing. If you also reformat all the code, it will be hard for us to focus on your change.
3. Ensure local tests pass.
4. Commit to your fork using clear commit messages.
5. Send us a pull request, answering any default questions in the pull request interface.
6. Pay attention to any automated CI failures reported in the pull request, and stay involved in the conversation.

GitHub provides additional document on forking a repository and creating a pull request.

12.4.3 How to find contributions to work on

Looking at the existing issues is a great way to find something to contribute on. As our projects, by default, use the default GitHub issue labels (enhancement/bug/duplicate/help wanted/invalid/question/wontfix), looking at any ‘help wanted’ issues is a great place to start.

12.4.4 What is the code of conduct

This project has adopted the Amazon Open Source Code of Conduct. For more information see the Code of Conduct FAQ or contact opensource-codeofconduct@amazon.com with any additional questions or comments.
12.4.5 How to notify for a security issue

If you discover a potential security issue in this project we ask that you notify AWS/Amazon Security via our vulnerability reporting page. Please do not create a public github issue.

12.4.6 What is the licensing

See the `https://github.com/aws/aws-neuron-sdk/blob/master/LICENSE-DOCUMENTATION` and `https://github.com/aws/aws-neuron-sdk/blob/master/LICENSE-SUMMARY-DOCS-SAMPLES` files for our project’s licensing. We will ask you to confirm the licensing of your contribution.

We may ask you to sign a Contributor License Agreement (CLA) for larger changes.

12.5 Neuron Compiler FAQs

- Where can I compile to Neuron?
- My current Neural Network is based on FP32, how can I use it with Neuron?
- What are some of the important compiler defaults I should be aware of?
- Which operators does Neuron support?
- Any operators that Neuron doesn’t support?
- Will I need to recompile again if I updated runtime/driver version?
- I have a NEFF binary, how can I tell which compiler version
- How long does it take to compile?

12.5.1 Where can I compile to Neuron?

The one-time compilation step from the standard framework-level model to NEFF binary may be performed on any EC2 instance or even on-premises.

We recommend using a high-performance compute server of choice (C5 or z1d instance types), for the fastest compile times and ease of use with a prebuilt DLAMI. Developers can also install Neuron in their own environments; this approach may work well for example when building a large fleet for inference, allowing the model creation, training and compilation to be done in the training fleet, with the NEFF files being distributed by a configuration management application to the inference fleet.

12.5.2 My current Neural Network is based on FP32, how can I use it with Neuron?

Developers who want to train their models in FP32 for best accuracy can compile and deploy them with Neuron. The Neuron compiler automatically converts FP32 to internally supported datatypes, such as FP16 or BF16. You can find more details about FP32 data type support and performance and accuracy tuning in Mixed precision and performance-accuracy tuning. The Neuron compiler preserves the application interface - FP32 inputs and outputs. Transferring such large tensors may become a bottleneck for your application. Therefore, you can improve execution time by casting the inputs and outputs to FP16 or BF16 in the ML framework prior to compilation for Inferentia.
12.5.3 What are some of the important compiler defaults I should be aware of?

The default optimization level is –O2. The compiler compiles the input graph for a single NeuronCore by default. Using the The “neuroncore-pipeline-cores” option directs the compiler to partition so as to run on a specified number of NeuronCores. This number can be less than the total available NeuronCores on an instance. See Performance Tuning for more information.

12.5.4 Which operators does Neuron support?

see Neuron Supported operators.

You can also use the “neuron-cc list-operators” command on the cli to list the operators. See neuron-cc-list-operators

If your model contains operators missing from the above list, and you can’t reach your performance goals, please post a message on the Neuron developer forum or open a github issue to let us know.

12.5.5 Any operators that Neuron doesn’t support?

Models with control-flow and dynamic shapes are not supported. You will need to partition the model using the framework prior to compilation. See the Neuron Compiler.

12.5.6 Will I need to recompile again if I updated runtime/driver version?

The compiler and runtime are committed to maintaining compatibility for major version releases with each other. The versioning is defined as major.minor, with compatibility for all versions with the same major number. If the versions mismatch, an error notification is logged and the load will fail. This will then require the model to be recompiled.

12.5.7 I have a NEFF binary, how can I tell which compiler version generated it?** We will bring a utility out to help with this soon.

12.5.8 How long does it take to compile?

It depends on the model and its size and complexity, but this generally takes a few minutes.

12.6 Neuron runtime FAQ

- Where can I find information about Neuron Runtime 2.x (libnrt.so)
- What will happen if I will upgrade Neuron Framework without upgrading latest kernel mode driver?
- Do I need to recompile my model to use the Runtime Library?
- Do I need to change my application launch command?
- How do I restart/start/stop the Neuron Runtime?
- How do I know which runtimes are associated with which Neuron Device(s)?
- What about RedHat or other versions of Linux and Windows?
12.6.1 Where can I find information about Neuron Runtime 2.x (libnrt.so)

See *Introducing Neuron Runtime 2.x (libnrt.so)* for detailed information about Neuron Runtime 2.x (libnrt.so).

12.6.2 What will happen if I will upgrade Neuron Framework without upgrading latest kernel mode driver?

Application start would fail with the following error message:

```bash
```

12.6.3 Do I need to recompile my model to use the Runtime Library?

No. Runtime 2.x supports all the models compiled with Neuron Compiler 1.x.

12.6.4 Do I need to change my application launch command?

No.

12.6.5 How do I restart/start/stop the Neuron Runtime?

Since Neuron Runtime is a library, starting/stopping application would result in starting/stopping the Neuron Runtime.

12.6.6 How do I know which runtimes are associated with which Neuron Device(s)?

`neuron-ls` and `neuron-top` can be used to find out applications using Neuron Devices.

12.6.7 What about RedHat or other versions of Linux and Windows?

We don’t officially support it yet.

12.6.8 How can I use Neuron in a container based environment? Does Neuron work with ECS and EKS?

ECS and EKS support is coming soon. Containers can be configured as shown [here](#).
12.6.9 How can I take advantage of multiple NeuronCores to run multiple inferences in parallel?

Examples of this for TensorFlow are found here here

12.7 Roadmap FAQ

- Why did you build this?
- What do the roadmap categories mean?
- Why are there no dates on your roadmap?
- Is everything on the roadmap?
- How can I provide feedback or ask for more information?
- How can I request a feature be added to the roadmap?
- Can I “+1” existing issues?

12.7.1 Why did you build this?

A: We know that our customers are making decisions and plans based on what we are developing, and we want to provide them with the right visibility to what we are working on, as well as the opportunity to provide direct feedback.

12.7.2 What do the roadmap categories mean?

- Roadmap Requests - Requests we received and we are considering to add to the roadmap, this is a great phase to give us feedback and let us know if you need this feature as well.
- Working on it - In progress, we might still be working through the implementation details, or scoping stuff out. This is a great phase to give us feedback as to how you want to see something implemented. We’ll benefit from your specific use cases here.
- Completed - Feature complete and supported by Neuron.

12.7.3 Why are there no dates on your roadmap?

A: We are not providing exact target dates for releases because we prioritize operational excellence, security and quality over hitting a specific date. If you have an urgent need for a feature, please contact us directly at aws-neuron-support@amazon.com.
12.7.4 Is everything on the roadmap?

A: We are focusing on upgrades for existing features, as well as building new features. We will keep adding features and capabilities to this roadmap as time progresses.

12.7.5 How can I provide feedback or ask for more information?

A: When in doubt, please create an issue or post a question on the AWS Neuron support forum.

12.7.6 How can I request a feature be added to the roadmap?

A: We encourage you to open an issue. All community-submitted issues will be reviewed by the roadmap maintainers.

12.7.7 Can I “+1” existing issues?

A: We strongly encourage you to do so, as it helps us understand which issues will have the widest impact. You can navigate to the issue details page and add a reaction (thumbs up). There are six types of reactions supported (thumbs down “-1”, confused, heart, watching, laugh, hooray, and thumbs up +1).
13.1 PyTorch Setup

- Fresh install
- Update to latest release
- Install previous releases

13.2 TensorFlow Setup

- Fresh install
- Update to latest release
- Install previous releases

13.3 Apache MxNet Setup

- Fresh install
- Update to latest release
- Install previous releases

13.4 Troubleshooting

- neuron-setup-troubleshooting
13.5 Additional Setup Resources

13.5.1 Common Developer Flows

*Developer Flows* section describe the common flows to develop with Neuron.

More information about DLAMI and Inferentia be found also at the DLAMI Documentation.

13.5.2 Deep Learning AMI (DLAMI)

Neuron packages are installed within Conda environments in AWS Deep Learning AMI (DLAMI) with Conda, and DLAMI is the recommended AMI to use with Neuron SDK.

For more information about Neuron and DLAMI:

**Deep Learning AMI (DLAMI) and Neuron versions Matrix**

<table>
<thead>
<tr>
<th>DLAMI version</th>
<th>Default Neuron version</th>
<th>How to upgrade to latest Neuron</th>
</tr>
</thead>
<tbody>
<tr>
<td>v46.0</td>
<td>v1.14.0 (May 2021)</td>
<td><a href="#">Neuron Pip Packages within DLAMI Conda Environments FAQ</a></td>
</tr>
<tr>
<td>v43.0</td>
<td>v1.12.2 (Mar 2021)</td>
<td><a href="#">Neuron Pip Packages within DLAMI Conda Environments FAQ</a></td>
</tr>
<tr>
<td>v42.2</td>
<td>v1.12.2 (Mar 2021)</td>
<td><a href="#">Neuron Pip Packages within DLAMI Conda Environments FAQ</a></td>
</tr>
<tr>
<td>v41.0</td>
<td>v1.12.1 (Feb 2021)</td>
<td><a href="#">Neuron Pip Packages within DLAMI Conda Environments FAQ</a></td>
</tr>
<tr>
<td>v40.0</td>
<td>v1.11.0 (Dec 2020)</td>
<td><a href="#">Neuron Pip Packages within DLAMI Conda Environments FAQ</a></td>
</tr>
<tr>
<td>v39.0</td>
<td>v1.11.0 (Dec 2020)</td>
<td><a href="#">Neuron Pip Packages within DLAMI Conda Environments FAQ</a></td>
</tr>
<tr>
<td>v38.0</td>
<td>v1.10.0 (Nov 2020)</td>
<td><a href="#">Neuron Pip Packages within DLAMI Conda Environments FAQ</a></td>
</tr>
<tr>
<td>v37.0</td>
<td>v1.10.0 (Nov 2020)</td>
<td><a href="#">Neuron Pip Packages within DLAMI Conda Environments FAQ</a></td>
</tr>
</tbody>
</table>

Neuron Pip Packages within DLAMI Conda Environments FAQ

<table>
<thead>
<tr>
<th>Table of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>• How to update to latest Neuron packages in DLAMI Conda Environments?</td>
</tr>
<tr>
<td>• What DLAMI versions include Neuron Conda environments?</td>
</tr>
<tr>
<td>• What version of Neuron packages are included in latest DLAMI version?</td>
</tr>
<tr>
<td>• Should I update to latest Neuron packages?</td>
</tr>
<tr>
<td>• How to know which DLAMI version I am running?</td>
</tr>
</tbody>
</table>
How to update to latest Neuron packages in DLAMI Conda Environments?

If the DLAMI Conda Environments do not include the latest Neuron packages, update the packages as follows:

- To upgrade Neuron PyTorch:

  ```bash
  source activate aws_neuron_pytorch_p36
  pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
  pip install --upgrade torch-neuron neuron-cc[tensorflow] torchvision
  ```

- To upgrade Neuron TensorFlow:

  ```bash
  source activate aws_neuron_tensorflow_p36
  pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
  pip install --upgrade tensorflow-neuron tensorboard-neuron neuron-cc
  ```

- To upgrade Neuron MXNet:

  ```bash
  source activate aws_neuron_mxnet_p36
  pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
  pip install --upgrade mxnet-neuron neuron-cc
  ```

**Note:** To avoid breaking an existing DLAMI environment, backup your DLAMI environment by creating an AMI from the existing DLAMI environment. Follow instructions at Create an AMI from an Amazon EC2 Instance to save the DLAMI before updating the Neuron Conda packages or upgrading to the latest DLAMI.

What DLAMI versions include Neuron Conda environments?

Starting with the DLAMI v26.0, the Deep Learning AMI with Conda Options include Neuron Conda packages.

Starting with Neuron SDK 1.14.0, pip packages (Neuron pip packages) are used to install Neuron SDK framework in DLAMI conda environments. To upgrade Neuron SDK framework DLAMI users should use pip upgrade commands instead of conda update commands. Instructions are in How to update to latest Neuron packages in DLAMI Conda Environments?. For more information, see https://aws.amazon.com/blogs/developer/neuron-conda-packages-eol/.

**Note:** Only Ubuntu 18 and Amazon Linux2 DLAMI are supported (Amazon Linux and Ubuntu 16 are not supported).

What version of Neuron packages are included in latest DLAMI version?

Both the DLAMI and Neuron have a monthly release cadence. When there is a new DLAMI release, it will include the latest Neuron Conda packages at the release time. This means that the latest DLAMI version include either the latest Neuron packages or the previous. See Deep Learning AMI (DLAMI) and Neuron versions Matrix for latest DLAMI information.
**Should I update to latest Neuron packages?**

Update to the latest Neuron packages if the tutorial or the machine learning application you intend to run require a feature or bug fix from the latest Neuron version. See *What’s New* for information on the latest Neuron version.

**How to know which DLAMI version I am running?**

You see the version of the running DLAMI by inspecting the README file on the user’s home folder, or at the start of a new terminal session. In the example below the DLAMI version is 35.0

```
cat ~/README
```

```
ubuntu@ip-172-31-88-188:~/$ cat ~/README
=============================================================================
___|__|_ )
| ( / Deep Learning AMI (Ubuntu 18.04) Version 35.0
___\___|___|
=============================================================================  
```

**DLAMI with Neuron Release Notes**

For more information about using Neuron with Conda and Base DLAMI, please see https://docs.aws.amazon.com/dlami/latest/devguide/tutorial-inferentia.html.

**Table of Contents**

- [DLAMI v43.0]
- [DLAMI v42.2]
- [DLAMI v41]
- [DLAMI v40]
- [DLAMI v39]
- [DLAMI v38]
- [DLAMI v37]
- [DLAMI v36]
- [DLAMI v35]
- [DLAMI v34]
- [DLAMI v33]
- [DLAMI v32]
- [DLAMI v31]
- [DLAMI v30.1 (Amazon Linux 2)]
- [DLAMI v30.0 (Ubuntu 16 and Ubuntu 18)]
- [DLAMI v29.0]
- [DLAMI v28.0]
Resolved Issues

Versions of Neuron packages included:

conda package mxnet-neuron-1.5.1.1.3.8.0 (includes neuron_cc-1.2.7.0)
conda package tensorflow-neuron-1.15.5.1.2.9.0 (includes neuron_cc-1.2.7.0)
conda package torch-neuron-1.7.1.1.2.16.0 (includes neuron_cc-1.2.7.0)
aws-neuron-dkms-1.4.5.0.
aws-neuron-runtime-base-1.4.8.0
aws-neuron-runtime-1.4.12.0
aws-neuron-tools-1.4.12.0
tensorflow-model-server-neuron-1.15.0.1.2.9.0

Resolved Issues

Versions of Neuron packages included:

conda package mxnet-neuron-1.5.1.1.3.8.0 (includes neuron_cc-1.2.7.0)
conda package tensorflow-neuron-1.15.5.1.2.9.0 (includes neuron_cc-1.2.7.0)
conda package torch-neuron-1.7.1.1.2.16.0 (includes neuron_cc-1.2.7.0)
aws-neuron-dkms-1.4.5.0.
aws-neuron-runtime-base-1.4.8.0
aws-neuron-runtime-1.4.12.0
aws-neuron-tools-1.4.12.0
tensorflow-model-server-neuron-1.15.0.1.2.9.0
[DLAMI v41]

Resolved Issues

Versions of Neuron packages included:

conda package mxnet-neuron-1.5.1.1.3.7.0 (includes neuron_cc-1.2.7.0)
conda package tensorflow-neuron-1.15.5.1.2.8.0 (includes neuron_cc-1.2.7.0)
conda package torch-neuron-1.7.1.1.2.15.0 (includes neuron_cc-1.2.7.0)
aws-neuron-dkms-1.4.1.0.
aws-neuron-runtime-base-1.4.2.0
aws-neuron-runtime-1.4.3.0
aws-neuron-tools-1.4.2.0
tensorflow-model-server-neuron-1.15.0.1.2.8.0

[DLAMI v40]

Resolved Issues

Versions of Neuron packages included:

conda package mxnet-neuron-1.5.1.1.2.1.0 (includes neuron_cc-1.1.7.0)
conda package tensorflow-neuron-1.15.4.1.1.3.0 (includes neuron_cc-1.1.7.0)
conda package torch-neuron-1.7.1.1.1.7.0 (includes neuron_cc-1.1.7.0)
aws-neuron-dkms-1.3.2.0.
aws-neuron-runtime-base-1.3.2.0
aws-neuron-runtime-1.3.1.0
aws-neuron-tools-1.3.1.0
tensorflow-model-server-neuron-1.15.0.1.1.3.0

[DLAMI v39]

Resolved Issues

Versions of Neuron packages included:

conda package mxnet-neuron-1.5.1.1.2.1.0 (includes neuron_cc-1.1.7.0)
conda package tensorflow-neuron-1.15.4.1.1.3.0 (includes neuron_cc-1.1.7.0)
conda package torch-neuron-1.7.1.1.1.7.0 (includes neuron_cc-1.1.7.0)
aws-neuron-dkms-1.3.2.0.
aws-neuron-runtime-base-1.3.2.0
aws-neuron-runtime-1.3.1.0
aws-neuron-tools-1.3.1.0
tensorflow-model-server-neuron-1.15.0.1.1.3.0

[DLAMI v38]

Resolved Issues

Versions of Neuron packages included:

conda package mxnet-neuron-1.5.1.1.1.88.0 (includes neuron_cc-1.0.24045.0)
conda package tensorflow-neuron-1.15.4.1.0.2168.0 (includes neuron_cc-1.0.24045.0)
conda package torch-neuron-1.5.1.1.0.1978.0 (includes neuron_cc-1.0.24045.0)
aws-neuron-dkms-1.2
aws-neuron-runtime-base-1.2.0.0
aws-neuron-runtime-1.2.5.0
aws-neuron-tools-1.2.7.0
tensorflow-model-server-neuron-1.15.0.1.0.2168.0

[DLAMI v37]

Resolved Issues

- Conda DLAMI v36 aws_neuron_pytorch_p36 environment has old versions of compiler and framework (https://github.com/aws/aws-neuron-sdk/issues/205)
- Unable to update Conda DLAMI v36 aws_neuron_pytorch_p36 environments (https://github.com/aws/aws-neuron-sdk/issues/206)

Versions of Neuron packages included:

conda package mxnet-neuron-1.5.1.1.1.88.0 (includes neuron_cc-1.0.24045.0)
conda package tensorflow-neuron-1.15.4.1.0.2168.0 (includes neuron_cc-1.0.24045.0)
conda package torch-neuron-1.5.1.1.0.1978.0 (includes neuron_cc-1.0.24045.0)
aws-neuron-dkms-1.2
aws-neuron-runtime-base-1.2.0.0
aws-neuron-runtime-1.2.5.0
aws-neuron-tools-1.2.7.0
tensorflow-model-server-neuron-1.15.0.1.0.2168.0

13.5. Additional Setup Resources
[DLAMI v36]

Versions of Neuron packages included:

conda package mxnet-neuron-1.5.1.1.1.52.0_2.0.757.0 (includes neuron_cc-1.0.20600.0)
conda package tensorflow-neuron-1.15.3.1.0.2043.0_2.0.894.0 (includes neuron_cc-1.0.20600.0)
conda package torch-neuron-1.5.1.1.0.298.0_2.0.880.0 (includes neuron-cc-1.0.18001.0)
aws-neuron-dkms-1.1
aws-neuron-runtime-base-1.1.1.0
aws-neuron-runtime-1.1.1402.0
aws-neuron-tools-1.1.228.0
tensorflow-model-server-neuron-1.15.0.1.0.2043.0

[DLAMI v35]

Versions of Neuron packages included:

conda package mxnet-neuron-1.5.1.1.1.52.0_2.0.757.0
conda package tensorflow-neuron-1.15.3.1.0.2043.0_2.0.894.0
conda package torch-neuron-1.5.1.1.0.298.0_2.0.880.0
aws-neuron-runtime-base-1.0.8126.0
aws-neuron-runtime-1.0.9592.0
aws-neuron-tools-1.0.11054.0
tensorflow-model-server-neuron-1.15.0.1.0.2043.0

[DLAMI v34]

Versions of Neuron packages included:

conda package mxnet-neuron-1.5.1.1.1.1.0_2.0.651.0
conda package tensorflow-neuron-1.15.3.1.0.1965.0_2.0.778.0
conda package torch-neuron-1.5.1.1.0.298.0_2.0.880.0
aws-neuron-runtime-base-1.0.7803.0
aws-neuron-runtime-1.0.9197.0
aws-neuron-tools-1.0.10616.0
tensorflow-model-server-neuron-1.15.0.1.0.1965.0
[DLAMI v33]

Versions of Neuron packages included:

conda package mxnet-neuron-1.5.1.1.1.0.2.0.651.0
conda package tensorflow-neuron-1.15.3.1.0.1965.0.2.0.778.0
conda package torch-neuron-1.5.1.1.0.298.0.2.0.880.0
aws-neuron-runtime-base-1.0.7803.0
aws-neuron-runtime-1.0.8896.0
aws-neuron-tools-1.0.10272.0
tensorflow-model-server-neuron-1.15.0.1.0.1965.0

[DLAMI v32]

Versions of Neuron packages included:

conda package mxnet-neuron-1.5.1.1.0.2.0.631.0
conda package tensorflow-neuron-1.15.3.1.0.1953.0.2.0.769.0
conda package torch-neuron-1.5.1.1.0.258.0.2.0.871.0
aws-neuron-runtime-base-1.0.7618.0
aws-neuron-runtime-1.0.8813.0
aws-neuron-tools-1.0.10182.0
tensorflow-model-server-neuron-1.15.0.1.0.1953.0

[DLAMI v31]

Versions of Neuron packages included:

conda package mxnet-neuron-1.5.1.1.0.2.0.579.0
conda package tensorflow-neuron-1.15.3.1.0.1891.0.2.0.706.0
conda package torch-neuron-1.5.1.1.0.251.0.2.0.783.0
aws-neuron-runtime-base-1.0.7395.0
aws-neuron-runtime-1.0.8032.0
aws-neuron-tools-1.0.9171.0
tensorflow-model-server-neuron-1.15.0.1.0.1796.0

13.5. Additional Setup Resources
[DLAMI v30.1 (Amazon Linux 2)]

Versions of Neuron packages included:

conda package mxnet-neuron-1.5.1.1.0.2033.0_2.0.489.0
conda package tensorflow-neuron-1.15.2.1.0.1796.0_2.0.593.0
conda package torch-neuron-1.3.0.1.0.215.0_2.0.633.0
aws-neuron-k8-plugin-1.0.9171.0
aws-neuron-runtime-base-1.0.7395.0
aws-neuron-runtime-1.0.8032.0
aws-neuron-tools-1.0.9171.0
tensorflow-model-server-neuron-1.15.0.1.0.1796.0

[DLAMI v30.0 (Ubuntu 16 and Ubuntu 18)]

Versions of Neuron packages included:

conda package mxnet-neuron-1.5.1.1.0.2033.0_2.0.489.0
conda package tensorflow-neuron-1.15.2.1.0.1796.0_2.0.593.0
conda package torch-neuron-1.3.0.1.0.215.0_2.0.633.0
aws-neuron-k8-plugin-1.0.9171.0
aws-neuron-runtime-base-1.0.7295.0
aws-neuron-runtime-1.0.7865.0
aws-neuron-tools-1.0.9043.0
tensorflow-model-server-neuron-1.15.0.1.0.1796.0

[DLAMI v29.0]

Versions of Neuron packages included:

conda package mxnet-neuron-1.5.1.1.0.1498.0_1.0.918.0
conda package tensorflow-neuron-1.15.0.1.0.1240.0_1.0.918.0
conda package torch-neuron-1.3.0.1.0.170.0_2.0.349.0
aws-neuron-runtime-base-1.0.7173.0
aws-neuron-runtime-1.0.6905.0
aws-neuron-tools-1.0.8550.0
tensorflow-model-server-neuron-1.15.0.1.0.1572.0
[DLAMI v28.0]

Versions of Neuron packages included:

conda package mxnet-neuron-1.5.1.1.0.918.0
conda package tensorflow-neuron-1.15.0.1.0.1240.0_1.0.918.0
conda package torch-neuron-1.3.0.1.0.90.0_1.0.918.0
aws-neuron-runtime-base-1.0.6554.0
aws-neuron-runtime-1.0.6222.0
aws-neuron-tools-1.0.6554.0
tensorflow-model-server-neuron-1.15.0.1.0.1333.0

[DLAMI v27.0]

This DLAMI release incorporates all content in the releases for Neuron up to and including the Feb 27, 2020 SDK release set.

Versions of Neuron packages included:

conda package mxnet-neuron-1.5.1.1.0.918.0
conda package tensorflow-neuron-1.15.0.1.0.1240.0_1.0.918.0
conda package torch-neuron-1.3.0.1.0.90.0_1.0.918.0
aws-neuron-runtime-base-1.0.5832.0
aws-neuron-runtime-1.0.5795.0
aws-neuron-tools-1.0.5832.0
tensorflow-model-server-neuron-1.15.0.1.0.1240.0

Resolved issues

• To update Conda package in Conda DLAMI v27.0 and up, simply do “conda update tensorflow-neuron” within Conda environment aws_neuron_tensorflow_p36. There’s no need to install Numpy version 1.17.2 as in DLAMI v26.0.

Updating

• It is strongly encouraged to update all packages to most recent release. If using Conda environments, please use “conda update” instead of “pip install” within the respective environment:
Base and Conda DLAMI on Ubuntu:

```bash
sudo apt-get update
sudo apt-get install aws-neuron-runtime-base
sudo apt-get install aws-neuron-runtime
sudo apt-get install aws-neuron-tools
sudo apt-get install tensorflow-model-server-neuron
```

Base and Conda DLAMI on Amazon Linux:

```bash
sudo yum install aws-neuron-runtime-base
sudo yum install aws-neuron-runtime
sudo yum install aws-neuron-tools
sudo yum install tensorflow-model-server-neuron
```

Conda DLAMI:

```
# MXNet-Neuron Conda environment
source activate aws_neuron_mxnet_p36
conda update mxnet-neuron

# TensorFlow-Neuron Conda environment
source activate aws_neuron_tensorflow_p36
conda update tensorflow-neuron

# PyTorch-Neuron Conda environment
source activate aws_neuron_pytorch_p36
conda update torch-neuron
```

[DLAMI v26.0]

NOTE: It is strongly encouraged to update all packages to most recent release. If using Conda environments, please use “conda update” instead of “pip install” within the respective environment:

**Supported Operating Systems:**

Amazon Linux 2
Ubuntu 16
Ubuntu 18
Versions of Neuron packages included:

conda package mxnet-neuron-1.5.1.1.0.1260.0_1.0.298.0
conda package tensorflow-neuron-1.15.0.1.0.663.0_1.0.298.0
aws-neuron-runtime-base-1.0.3657.0
aws-neuron-runtime-1.0.4109.0
aws-neuron-tools-1.0.3657.0
tensorflow-model-server-neuron-1.15.0.1.0.663.0

Known Issues

Installation Guidelines

Base and Conda DLAMI on Ubuntu:

```
sudo apt-get update
sudo apt-get install aws-neuron-runtime-base
sudo apt-get install aws-neuron-runtime
sudo apt-get install aws-neuron-tools
sudo apt-get install tensorflow-model-server-neuron
```

Base and Conda DLAMI on Amazon Linux:

```
sudo yum install aws-neuron-runtime-base
sudo yum install aws-neuron-runtime
sudo yum install aws-neuron-tools
sudo yum install tensorflow-model-server-neuron
```

Conda DLAMI:

```
# MXNet-Neuron Conda environment
source activate aws_neuron_mxnet_p36
conda update mxnet-neuron

# TensorFlow-Neuron Conda environment (DLAMI v26)
source activate aws_neuron_tensorflow_p36
conda install numpy=1.17.2 --yes --quiet
conda update tensorflow-neuron
```

- In TensorFlow-Neuron conda environment (aws_neuron_tensorflow_p36), the installed numpy version prevents update to latest conda package version. Please do “conda install numpy=1.17.2 –yes –quiet” before “conda update tensorflow-neuron”.
- When using the Conda DLAMI, use the above conda commands to update packages, not pip.
- When doing `conda update aws_neuron_tensorflow` in the `aws_neuron_tensorflow_p36` environment or when using pip install, you will see the following warning which can be ignored: “neuron-cc has requirement numpy<=1.17.2,>=1.13.3, but you’ll have numpy 1.17.4 which is incompatible.”
• Customers experiencing 404 errors from https://yum.repos.neuron.amazonaws.com during yum updates will need to remake their yum HTTP caches as shown in the code below this bullet. It’s also encouraged to configure the Neuron repository for immediate metadata expiration to avoid the 404 errors in the future as shown here:

Setup Guide

```
# refresh yum HTTP cache:
sudo yum makecache
```

• If using Base DLAMI and installing tensorflow-neuron outside of Conda or virtual environment, the package ‘wrapt’ may cause an error during installation using Pip. In this case an error like this will occur:

```
ERROR: Cannot uninstall 'wrapt'. It is a distutils installed project and thus we cannot accurately determine which files belong to it which would lead to only a partial uninstall.
```

• To resolve this, execute:

```
python3 -m pip install wrapt --ignore-installed
python3 -m pip install tensorflow-neuron
```

• The tensorflow-neuron conda package comes with:
  – TensorBoard-Neuron for Neuron v1.12.2 release and earlier (tensorflow-neuron<=1.15.5.1.2.9.0)
  – the Neuron plugin for TensorBoard for Neuron v1.13.0 release and later
• There is no standalone tensorboard-neuron or tensorboard-plugin-neuron package at this time.

For more information, please see Known Issues and Limitations - updated 08/12/2021.

13.5.3 Deep Learning (DL) Containers

For containerized applications, it is recommended to use the neuron-rtd container, more details at Containers.

13.5.4 Jupyter notebook setup

Jupyter Notebook QuickStart

Table of Contents

• SSH Tunnel to the inf1 instance
• Starting the Jupyter Notebook on the instance
• Running the Jupyter Notebook from your local browser
• Troubleshooting
SSH Tunnel to the inf1 instance

The Jupyter notebook can be run via a browser on port 8888 by default. For simplicity we will use ssh port forwarding from your machine to the instance.

```
ssh -i "<pem file>" <user>@<instance DNS name> -L 8888:127.0.0.1:8888
```

On an Ubuntu image the user will be ubuntu@, while on AL2 you should use ec2-user@

This additional argument forwards connections to port 8888 on your machine to the new inf1 instance.

Starting the Jupyter Notebook on the instance

From your ssh prompt on the inf1 instance run

```
jupyter notebook
```

You should see logging in your ssh session similar to:

```
[I 21:53:11.729 NotebookApp] Using EnvironmentKernelSpecManager...
[I 21:53:12.002 NotebookApp] [nb_conda] enabled
[I 21:53:12.004 NotebookApp] 0 active kernels
[I 21:53:12.004 NotebookApp] The Jupyter Notebook is running at:
[I 21:53:12.004 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
```

Copy/paste this URL into your browser when you connect for the first time, to login with a token: http://localhost:8888/?token=f9ad4086afdf3c91f33d5587781f9fd8143b4cafbbf121a16

```
```

13.5. Additional Setup Resources
Running the Jupyter Notebook from your local browser

If you copy and paste the link that looks like http://localhost:8888/?token=f9ad4086af3d333d558781f9fd8143b4cafbf3121a16&token=f9ad4086af3d333d558781f9fd8143b4cafbf3121a16 into your local browser the Notebook navigation pane should pop up.

This works because ssh is forwarding you local port 8888 through to the Inf1 instance port 8888 where the notebook is running. Note that our new conda environment is visible as “kernel” with the “conda_” prefix (highlighted)

1) In notebook browser select the tutorial.

2) This will pop up a new tab. In that tab use the menus:

Kernel → Change Kernel → Environment (conda_my_notebook_env)

3) Start reading through the self documenting notebook tutorial

Troubleshooting

If your jupyter notebook does not start please try the following:

mv ~/.jupyter ~/.jupyter.old
mkdir -p ~/.jupyter
echo "c.NotebookApp.ipub_data_rate_limit = 10000000000" > ~/.jupyter/jupyter_notebook_config.py
conda install nb_conda_kernels
jupyter notebook

Running Jupyter Notebook as script

Converting the Jupyter Notebook and running

Go into the aws-neuron-sdk repository directory containing the Jupyter Notebook (.ipynb file),

cd aws-neuron-sdk/src/examples/<framework like pytorch, tensorflow, etc>

The Jupyter Notebook (.ipynb) can be converted to python script using jupyter-nbconvert. For example,

jupyter nbconvert --to script tutorial_pretrained_bert.ipynb

and can be run in the virtual env (if needed),

# if not already in the virtual env,
source activate <virtual env>
# Run the converted script
python <tutorial.py>
13.5.5 Launching Inf1 Instance from AWS CLI

AWS CLI commands to launch inf1 instances

```
# Launch instance
# The following are the different Deep Learning AMIs to get started and is recommended
# for the tutorials.
# "Deep Learning AMI (Amazon Linux)"
# "Deep Learning AMI (Amazon Linux 2)"
# "Deep Learning AMI (Ubuntu 18.04)"
#
# You can get the latest AMI ID for any of the above ones using the following command
AWS_REGION="<aws region name like us-east-1>"
AMIID=$(aws ec2 describe-images --filters "Name=name,Values=Deep Learning Base AMI
→(Ubuntu 18.04)" --query 'sort_by(Images, &CreationDate)[].[Name,ImageId]'
→$AWS_REGION --output text | tail -n 1 | awk '{print $(NF)}')
INSTANCE_ID=$(aws ec2 run-instances --image-id $AMIID --count 1 --instance-type <inf1.
→xlarge type> --key-name MyKeyPair --region $AWS_REGION [--subnet-id <subnet id>]
→python -c 'import sys, json; print(json.load(sys.stdin)
→"Instances""][0]["InstanceId"]')
echo "Instance ID of launched instance" $INSTANCE_ID

# Wait for few seconds to a minute for the instance to get created and have public
→DNS/ip.

# The following command will get the public DNS name of the launched instance to which
# you can then log in to using your key pair.
INSTANCE_PUBLIC_DNS=$(aws ec2 describe-instances --instance-id $INSTANCE_ID --region
→$AWS_REGION | python -c 'import sys, json; print(json.load(sys.stdin)
→"Reservations""][0]["Instances""][0]["PublicDnsName"]')
echo "DNS name of the launched instance" $INSTANCE_PUBLIC_DNS

# Wait for couple of minutes for the instance to be ready and then login:
ssh -i <key.pem> <ubuntu/ec2-user>@$INSTANCE_PUBLIC_DNS
```

13.5.6 Tensorboard

Install Neuron Plugin for TensorBoard

The Neuron plugin for TensorBoard is available starting with Neuron v1.13.0.

To install the Neuron plugin, first enable ML framework Conda environment of your choice, by running one of the following:

- Enable PyTorch-Neuron Conda environment:
  - PyTorch 1.9.1
  - PyTorch 1.8.1
  - Ubuntu DLAMI
  - Amazon Linux DLAMI
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install
#   or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron neuron-cc[tensorflow] torchvision
copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  →'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo yum install aws-neuron-tools -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron neuron-cc[tensorflow] torchvision

Ubuntu DLAMI

Amazon Linux DLAMI

13.5. Additional Setup Resources 1171
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate PyTorch
source activate aws_neuron_pytorch_p36
```

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  → 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate PyTorch
source activate aws_neuron_pytorch_p36

• Enable TensorFlow-Neuron Conda enviroment:
  TensorFlow 2.5.1
  TensorFlow 1.15.5
  Ubuntu DLAMI
  Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
To install or update to Neuron versions 1.16.0 and newer from previous releases:

- DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

Install OS headers
```
sudo apt-get install linux-headers-$\{uname -r\} -y
```

Install Neuron Driver
```
sudo apt-get install aws-neuron-dkms -y
```

Warning: If Linux kernel is updated as a result of OS package update
Neuron driver (aws-neuron-dkms) should be re-installed after reboot

Install Neuron Tools
```
sudo apt-get install aws-neuron-tools -y
```

Install Neuron TensorBoard
```
pip install tensorboard-plugin-neuron
```

Export PATH
```
export PATH=/opt/aws/neuron/bin:$PATH
```

Install Python venv and activate Python virtual environment to install Neuron pip packages.
```
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip
```

Install Jupyter notebook kernel
```
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels
```

Set Pip repository to point to the Neuron repository
```
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
```

Install Neuron TensorFlow
```
pip install tensorflow-neuron[cc]
```

Optional: Install Neuron TensorFlow model server
```
sudo apt-get install tensorflow-model-server-neuron -y
```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo yum install aws-neuron-tools -y

# Install Neuron TensorBoard
pip install tensorflow-plugin-neuron
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel

(continues on next page)
python -m ipykernel install --user --name tensorflow_venv --display-name "Python → (Neuron TensorFlow)"

pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

#Install Neuron TensorFlow
pip install tensorflow-neuron[cc]

# Optional: Install Neuron TensorFlow model server
sudo yum install tensorflow-model-server-neuron -y

Ubuntu DLAMI

Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation
# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
# Install OS headers
sudo yum install kernel-devel-$uname -r) kernel-headers-$uname -r) -y
# Install Neuron Driver
sudo yum install aws-neuron-dkms -y
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
# Activate TensorFlow
source activate aws_neuron_tensorflow_p36
```

- Enable MXNet-Neuron Conda enviroment:
  - MXNet 1.8.0
  - MXNet 1.5.1
  - Ubuntu DLAMI
  - Amazon Linux DLAMI
AWS Neuron

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or update to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
   -py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron neuron-cc
```
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo yum install aws-neuron-tools -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
   -py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron_neuron-cc

Ubuntu DLAMI
```

13.5. Additional Setup Resources 1179
# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Activate MXNet
source activate aws_neuron_mxnet_p36

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate MXNet
source activate aws_neuron_mxnet_p36

Then run the following:
If you are using the DLAMI TensorFlow-Neuron Conda environment, please run the following to update TensorBoard before installing the Neuron plugin.

```
pip install "tensorboard<=2.4.0" --force-reinstall
```

Modify Pip repository configurations to point to the Neuron repository:

```
tee $VIRTUAL_ENV/pip.conf > /dev/null <<EOF
[global]
extra-index-url = https://pip.repos.neuron.amazonaws.com
EOF
```

```
pip install tensorboard-plugin-neuron
```
Install Neuron TensorBoard (Deprecated)

**Warning:** TensorBoard-Neuron is deprecated and no longer compatible with Neuron tools version 1.5 and higher. Neuron tools version 1.5 is first introduced in Neuron v1.13.0 release. Please use the Neuron plugin for TensorBoard instead.

To install Tensorboard, first enable ML framework Conda environment of your choice, by running one of the following:

- Enable PyTorch-Neuron Conda enviroment:
  - PyTorch 1.9.1
  - PyTorch 1.8.1
  - Ubuntu DLAMI
  - Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
```

(continues on next page)
# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y

export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron neuron-cc[tensorflow] torchvision

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$uname -r kernel-headers-$uname -r -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
# Install Neuron Tools
sudo yum install aws-neuron-tools -y
export PATH=/opt/aws/neuron/bin:$PATH

# Activate PyTorch
source activate aws_neuron_pytorch_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron PyTorch
pip install torch-neuron neuron-cc[tensorflow] torchvision

Ubuntu DLAMI
Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
#   'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux Headers-$ (uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

### Warning: If Linux kernel is updated as a result of OS package update
Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling:
  → 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
  sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
  sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate PyTorch
  source activate aws_neuron_pytorch_p36

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

- Enable TensorFlow-Neuron Conda environment:

  TensorFlow 2.5.1
Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

```bash
# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or
#   upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y

# Install Neuron TensorFlow
pip install tensorboard-plugin-neuron
export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install
# Neuron pip packages.
sudo apt-get install -y python3-venv g++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
```

(continues on next page)
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"

# Install Jupyter notebook
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]

# Optional: Install Neuron TensorFlow model server
sudo apt-get install tensorflow-model-server-neuron -y

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
(continues on next page)
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo yum install aws-neuron-tools -y

# Install Neuron TensorBoard
pip install tensorboard-plugin-neuron

export PATH=/opt/aws/neuron/bin:$PATH

# Install Python venv and activate Python virtual environment to install Neuron pip packages.
sudo yum install -y python3 gcc-c++
python3 -m venv tensorflow_venv
source tensorflow_venv/bin/activate
pip install -U pip

# Install Jupyter notebook kernel
pip install ipykernel
python -m ipykernel install --user --name tensorflow_venv --display-name "Python (Neuron TensorFlow)"
pip install jupyter notebook
pip install environment_kernels

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron TensorFlow
pip install tensorflow-neuron[cc]

# Optional: Install Neuron TensorFlow model server
sudo yum install tensorflow-model-server-neuron -y

Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.
# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Install OS headers
sudo yum install kernel-devel-$(uname -r) kernel-headers-$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate TensorFlow
source activate aws_neuron_tensorflow_p36

• Enable Neuron Conda environment for Neuron Apache MXNet (Incubating):
  MXNet 1.8.0
  MXNet 1.5.1
  Ubuntu DLAMI
  Amazon Linux DLAMI

**Note:** For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo apt-get update -y

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$\{uname -r\} -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot
##################################################################################################

# Install Neuron Tools
sudo apt-get install aws-neuron-tools -y
export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
   ->py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron neuron-cc

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Note: There is no DLAMI Conda environment for this framework version
# Framework will be installed/updated inside a Python environment

# Update OS packages
sudo yum update -y

##################################################################################################

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo
#   ->systemctl stop neuron-rtd'

##################################################################################################

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or
#   ->upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$ (uname -r) kernel-headers-$ (uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

##################################################################################################

(continues from previous page)
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Install Neuron Tools
sudo yum install aws-neuron-tools -y
export PATH=/opt/aws/neuron/bin:$PATH

# Activate MXNet
source activate aws_neuron_mxnet_p36

# Set Pip repository to point to the Neuron repository
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com

# Install Neuron MXNet
wget https://aws-mx-pypi.s3-us-west-2.amazonaws.com/1.8.0/aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip3 install aws_mx_cu110-1.8.0-py2.py3-none-manylinux2014_x86_64.whl
pip install mx_neuron neuron-cc

Ubuntu DLAMI
Amazon Linux DLAMI

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo apt-get install linux-headers-$(uname -r) -y

# Install Neuron Driver
sudo apt-get install aws-neuron-dkms -y
# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate MXNet
source activate aws_neuron_mxnet_p36

Note: For a successful installation or update, execute each line of the instructions below separately or copy the contents of the code block into a script file and source its contents.

# Neuron is pre-installed on Deep Learning AMI (DLAMI), latest DLAMI version may not include latest Neuron versions
# To update to latest Neuron version, follow "Update to latest release" instruction on Neuron documentation

# Before installing or updating aws-neuron-dkms:
# - Stop any existing Neuron runtime 1.0 daemon (neuron-rtd) by calling: 'sudo systemctl stop neuron-rtd'

# To install or update to Neuron versions 1.16.0 and newer from previous releases:
# - DO NOT skip 'aws-neuron-dkms' install or upgrade step, you MUST install or upgrade to latest Neuron driver

# Install OS headers
sudo yum install kernel-devel-$\$(uname -r) kernel-headers-$\$(uname -r) -y

# Install Neuron Driver
sudo yum install aws-neuron-dkms -y

# Warning: If Linux kernel is updated as a result of OS package update
# Neuron driver (aws-neuron-dkms) should be re-installed after reboot

# Activate MXNet
source activate aws_neuron_mxnet_p36

Then run the following:
pip install tensorboard-neuron

- Installing `tensorflow-neuron<=1.15.5.1.2.9.0` will automatically install `tensorboard-neuron` as a dependency. The final version, 1.15.5.1.2.9.0, is part of Neuron v1.12.2 release.

- To verify `tensorboard-neuron` is installed correctly, run `tensorboard_neuron -h | grep run_neuron_profile`. If nothing is shown, please retry installation with the `--force-reinstall` option.
The Neuron Compiler is an Ahead-of-Time (AoT) compiler that accepts Machine Learning models in various formats (TensorFlow, MXNet, PyTorch, XLA HLO) and optimizes them to run on AWS Inferentia chips.

AoT compilation requires that dynamic tensor shapes (dimension sizes) of all tensors in the compute-graph are known at compilation time, in order for the compiler to make informed decisions. Model compilation with shapes that cannot be determined at compile time will fail.

It is common for developers to train models in FP32 to avoid the challenges of low-precision training (e.g. loss-scaling, etc). However, during inference, developers typically look for the most performant and cost-effective methods. In order to address these two requirements, Neuron supports FP32 auto-casting, details for which can be found in Mixed precision and performance-accuracy tuning.

The Neuron compiler is used within an integrated framework, such as TensorFlow, PyTorch or MXNet. From that framework, ML models are sent to the compiler and the results then sent back. This compile step will often be done on a compilation server. The resulting compiler artifact is called a NEFF file (Neuron Executable File Format). NEFF files are loaded into an Inferentia device, using the Neuron Runtime. The NEFF is actually encapsulated inside a saved version of that model, which the framework can use.

14.1 Neuron compiler CLI Reference Guide

This document describes the command line interface of the Neuron compiler. This reference is not relevant for applications that run neuron-cc from within a machine learning framework (TensorFlow-Neuron for example) since these options are passed from the framework directly to neuron-cc.

Using neuron-cc on the command line may be desirable for applications that do not use a framework, or customize existing frameworks. It is also possible to supply CLI commands to the framework as options to be passed through to the compiler.

14.1.1 Usage

Optional parameters are shown in square brackets. See the individual framework guides for the correct syntax.
Neuron Compiler CLI

**neuron-cc** [options] <command> [parameters]

Common options for the Neuron CLI:

- **--verbose** (string) default="WARN":
  
  Valid values:
  - DEBUG
  - INFO
  - WARN
  - ERROR

Use **neuron-cc** <command> --help for information on a specific command.

**Available Commands:**

- compile
- list-operators

**neuron-cc compile [parameters]**

Compile a model for use on the AWS Inferentia Machine Learning Accelerator.

```bash
```

**Compile Parameters:**

- `<file names>`: Input containing model specification. The number of arguments required varies between frameworks:
  
  - **TENSORFLOW**: A local filename or URI of a TensorFlow Frozen GraphDef (.pb); or the name of a local directory containing a TensorFlow SavedModel.

  
  - **MXNET**: List of local filenames or URIs where input architecture .json file and parameter .param file are stored. These contains information related to the architecture of your graph and associated parameters, respectively.

- **--framework** (string): Framework in which the model was trained.

  Valid values:
  - TENSORFLOW
  - MXNET
  - XLA

- **--neuroncore-pipeline-cores** (int) (default=1): Number of neuron cores to be used in “NeuronCore Pipeline” mode. This is different from data parallel deployment (same model on multiple neuron cores). Refer to Runtime/Framework documentation for data parallel deployment options.
Compile for the given number of neuron cores so as to leverage NeuronCore Pipeline mode.

**Note:** This is not used to define the number of Neuron Cores to be used in a data parallel deployment (i.e., the same model on multiple Neuron Cores). That is a runtime/framework configuration choice.

- **--output** (string) (default="out.neff"): Filename where compilation output (NEFF archive) will be recorded.
- **--io-config** (string): Configuration containing the names and shapes of input and output tensors.

The io-config can be specified as a local filename, a URI, or a string containing the io-config itself.

The io-config must be formatted as a JSON object with two members “inputs” and “outputs”. “inputs” is an object mapping input tensor names to an array of shape and data type. “outputs” is an array of output tensor names. Consider the following example:

```json
{
  "inputs": {
    "input0:0": [[1,100,100,3], "float16"],
    "input1:0": [[1,100,100,3], "float16"]
  },
  "outputs": ["output:0"]
}
```

- **--enable-fast-loading-neuron-binaries**: Write the compilation output (NEFF archive) in uncompressed format which results in faster loading of the archive during inference.
- **--enable-fast-context-switch**: Optimize for faster model switching rather than inference latency. This results in overall faster system performance when your application switches between models frequently on the same neuron core (or set of cores). The optimization triggered by this option for example defers loading some weight constants until the start of inference.
- **--fast-math**: Controls tradeoff between performance and accuracy for fp32 operators. See more suggestions on how to use this option with the below arguments in Mixed precision and performance-accuracy tuning.
  - all (Default): enables all optimizations that improve performance. This option can potentially lower precision/accuracy.
  - none: Disables all optimizations that improve performance. This option will provide best precision/accuracy.
  - Tensor transpose options
    * fast-relayout: Only enables fast relayout optimization to improve performance by using the matrix multiplier for tensor transpose. The data type used for the transpose is either FP16 or BF16, which is controlled by the `fp32-cast-xxx` keyword.
    * no-fast-relayout: Disables fast relayout optimization which ensures that tensor transpose is bit-accurate (lossless) but slightly slower.
  - Casting options
    * fp32-cast-all (Default): Cast all FP32 operators to BF16 to achieve highest performance and preserve dynamic range. Same as setting `--fp32-cast all`.
    * fp32-cast-all-fp16: Cast all FP32 operators to FP16 to achieve speed up and increase precision versus BF16. Same setting as `--fp32-cast all-fp16`.  

* fp32-cast-matmult: Only cast FP32 operators that use Neuron Matmult engine to BF16 while using FP16 for matmult-based transpose to get better accuracy. Same as setting --fp32-cast matmult.

* fp32-cast-matmult-bf16: Cast only FP32 operators that use Neuron Matmult engine (including matmult-based transpose) to BF16 to preserve dynamic range. Same as setting --fp32-cast matmult-bf16.

* fp32-cast-matmult-fp16: Cast only FP32 operators that use Neuron Matmult engine (including matmult-based transpose) to fp16 to better preserve precision. Same as setting --fp32-cast matmult-fp16.

**Important:**
- all and none are mutually exclusive
- all is equivalent to using fp32-cast-all fast-relayout (best performance)
- none is equivalent to using fp32-cast-matmult-fp16 no-fast-relayout (best accuracy)
- fp32-cast-* options are mutually exclusive
- fast-relayout and no-fast-relayout are mutually exclusive
- The fp32-cast-* and --fast-relayout options will overwrite the default behavior in all and none.
- For backward compatibility, the --fp32-cast option has higher priority over --fast-math. It will overwrite the FP32 casting options in any of the --fast-math options if --fp32-cast option is present explicitly.

* --fp32-cast: Refine the automatic casting of fp32 tensors. This is being replaced by a newer --fast-math.

**Important:**
- --fp32-cast option is being deprecated and --fast-math will replace it in future releases.
- --fast-math is introducing the no-fast-relayout option to enable lossless transpose operation.

The --fp32-cast is an interface for controlling the performance and accuracy tradeoffs. Many of the --fast-math values invoke (override) it.

- all (default): Cast all FP32 operators to BF16 to achieve speed up and preserve dynamic range.
- matmult: Cast only FP32 operators that use Neuron Matmult engine to BF16 while using fp16 for matmult-based transpose to get better accuracy.
- matmult-fp16: Cast only FP32 operators that use Neuron Matmult engine (including matmult-based transpose) to fp16 to better preserve precision.
- matmult-bf16: Cast only FP32 operators that use Neuron Matmult engine (including matmult-based transpose) to BF16 to preserve dynamic range.
- all-fp16: Cast all FP32 operators to FP16 to achieve speed up and better preserve precision.
Log Levels:

Logs at levels “trace”, “debug”, and “info” will be written to STDOUT.
Logs at levels “warn”, “error”, and “fatal” will be written to STDERR.

Exit Status

0 - Compilation succeeded

>0 - An error occurred during compilation.

Examples

Compiling a saved TensorFlow model:

```
neuron-cc compile test_graph_tfmatmul.pb --framework TENSORFLOW --io-config test_graph_tfmatmul.config
```

Compiling a MXNet model:

```
neuron-cc compile lenet-symbol.json lenet-0001.params --framework MXNET --neuroncore-pipeline-cores 2 --output file.neff
```

Compiling an XLA HLO:

```
neuron-cc compile bert-model.hlo --framework XLA --output file.neff
```

`neuron-cc list-operators [parameters]`

Returns a newline ('n') separated list of operators supported by the NeuronCore.

- **TENSORFLOW**: Operators will be formatted according to the value passed to the associated REGISTER_OP(“OperatorName”) macro.
  
  See [https://www.tensorflow.org/guide/create_op#define_the_op_interface](https://www.tensorflow.org/guide/create_op#define_the_op_interface) for more information regarding operator registration in TensorFlow.

- **MXNET**: Operator names will be formatted according to the value passed to the associated NNVM_REGISTER_OP(operator_name) macro.

- **XLA**: Operator names will be formatted according to the value used by XLA compiler in XlaBuilder.
  
  See [https://www.tensorflow.org/xla/operation_semantics](https://www.tensorflow.org/xla/operation_semantics) for more information regarding XLA operator semantics in XLA interface.

```
neuron-cc list-operators --framework <value>
```

- **--framework** (string): Framework in which the operators were registered.

  Valid values:
  - TENSORFLOW
  - MXNET
  - XLA

Exit Status

0 - Call succeeded
AWS Neuron

>0 - An error occurred

Example

```bash
$ neuron-cc list-operators --framework TENSORFLOW
AddN
AdjustContrastv2
CheckNumbers
...
```

14.2 What’s New

14.2.1 Neuron Compiler Release Notes

- Introduction
- Known issues and limitations - updated 10/27/2021
- Neuron Compiler release [1.8.5.0]
- Neuron Compiler release [1.8.2.0]
- Neuron Compiler release [1.7.3.0]
- [1.6.13.0]
- [1.5.5.0]
- [1.4.0.0]
- [1.3.0.0]
- [1.2.7.0]
- [1.2.2.0]
- [1.1.7.0]
- [1.0.24045.0]
- [1.0.20600.0]
- [1.0.18001.0]
- [1.0.17937.0]
- [1.0.16861.0]
- [1.0.15275.0]
- [1.0.12696.0]
- [1.0.9410.0]
- [1.0.7878.0]
- [1.0.6801.0]
- [1.0.5939.0]
- [1.0.5301.0]
- [1.0.4680.0]
Introduction

This document lists the release notes for AWS Neuron compiler. The Neuron Compiler is an ahead-of-time compiler that ensures Neuron will optimally utilize the Inferentia chips.

Operator-support for each input format is provided directly from the compiler.

```
neuron-cc list-operators --framework {TENSORFLOW | MXNET | XLA}
```

The supported operators are also listed here:

Tensorflow: TensorFlow 1.x Supported operators

Pytorch: PyTorch Supported operators

XLA: neuron-cc-ops-xla

Apache MXNet (Incubating): Neuron Apache MXNet (Incubating) Supported operators

Known issues and limitations - updated 10/27/2021

- **TensorFlow 2.x** - In this release supported operators are limited to BERT-like models, specifically no conv2d or reduce-window operators are available.

- **Control flow** Neuron only supports control flow operators which are static at compile time. For example static length RNN, top-k, sort.

- **Data layout** The Neuron compiler supports multiple data layout format (NCHW, NHWC, ...). Non-CNHW input/output data-layouts will require Neuron to insert additional transpose operations, causing a degradation in performance.

- **Primary inputs in NeuronCore Pipeline mode** When a neural network is executed in NeuronCore Pipeline mode, only the first operator in a neural network can receive primary inputs from the host.

- **Reduce data type** INT8 data type is not currently supported by the Neuron compiler.

- **NeuronCore Pipeline**: NeuronCorePipeline mode provides low-latency and high-throughput for small batch sizes. We recommend to start testing with batch=1 and gradually increase batch size to fine tune your model throughput and latency performance.

- **Large input tensors** support varies by model. On some models the large input tensors (eg 1024x1024) may result in lower performance or exceeding hardware or compile-time limits, especially on models where the large input tensor is used by many downstream operators. Workarounds may include use of smaller batch, see Neuron Batching

- **Conv2d operator** is mapped to Inferentia except for specific cases of extremely large tensors and specific parameters.

- **Conv3d operator** performance is limited when the operator has small number of input channels (< 64).

- FP64 and INT64 input and output tensors are not supported. Please cast to FP32/INT32 in the machine learning framework, prior compiling for Neuron.
Neuron Compiler release [1.8.5.0]

Date: 01/05/2022

New in this release

• Minor bug fixes.

Neuron Compiler release [1.8.2.0]

Date: 12/15/2021

New in this release

• Performance enhancements as a result of improved layout and DMA optimizations.
  • Minor bug fixes.

Neuron Compiler release [1.7.3.0]

Date: 10/27/2021

New in this release

• The compiler’s list-operators command can now display the supported TensorFlow 2.x operators.
  • Support added for new operators in TensorFlow 1.x - ArgMax and ArgMin.
  • Introducing the --fast-math option for better fine-tuning of accuracy/performance. See *Mixed precision and performance-accuracy tuning*.

[1.6.13.0]

Date 08/12/2021

New in this release

• TensorFlow 2.x - First support of TensorFlow 2.x. The support is limited to operators in BERT-like models and was tested with Huggingface BERT small, base, large and DistillBert.
Resolved issues

- Fixed compiler backend issue in Tensor_tensor argument distance, github #269

[1.5.5.0]
Date 07/02/2021

Summary

- Robustness and performance improvements.

New in this release

- Added --enable-fast-context-switch to optimize for faster model switching rather than inference latency.
- Deprecated support for ONNX
- Improved robustness of Conv3d
- Corrected compilation error “too many instructions” in DLRM model

[1.4.0.0]
Date 5/28/2021

Summary

- Performance improvements, and usability improvements.

New in this release

- Added uncompressed NEFF format for faster loading models prior inference. Enable it by --enable-fast-loading-
  neuron-binaries. Some cases of large models may be detrimentally impacted as it will not be compressed but
  many cases will benefit.
- Corrected compilation error in specific arguments of ResizeBilinear operator

[1.3.0.0]
Date 4/30/2021
Summary

- Performance improvements, new operators, and usability improvements.

New in this release

- Improved performance of batched CNN models like resnet50 with the default compiler options by 10%.
- Improved performance of bert base sequence 128 batch 6 by upto 16%
- Added support for group and depth wise convolution (with limited performance when the number of input channels is small).
- Added more detailed debug names to support for tensorboard.

Resolved Issues

- Corrected potential race condition in overwriting tiles of output tensors.
- Fixed various issues in pipelined inference by enabling fine grain partitioning by default.

[1.2.7.0]

Date 2/24/2021

Summary

Fix for CVE-2021-3177.

[1.2.2.0]

Date 1/30/2021

Summary

Added support for multiple new operators (see operators list) for Tensorflow and MXNET. Improved inference performance of language, object recognition models on single as well as multiple pipelined cores using neuroncore-pipeline.

New in this release

- The following models are now supported: Resnext 224x224, specific BERT variations applied to natural language processing and translation.
- A number of new operators is now supported on Inferentia, see the full lists TensorFlow 1.x Supported operators and Neuron Apache MXNet (Incubating) Supported operators
- Improved inference performance on yolov4 BERT base sequence 64 (on 16 pipelined cores) and openpose 184.
Resolved Issues

• Corrected a random failure to compile Resnet50 batch 5
• Corrected numerical inaccuracy in RSQRT and related operators for tensors with very large values (> 1e20)

[1.1.7.0]
Date 12/23/2020

Summary

Added support for PyTorch Yolo V4, a new Framework-visible progress bar and improved inference performance. We continue to streamline the compiler usability by removing the need for options passed to control behavior. We are aiming to remove the need for such options entirely. Some tutorials have been updated to reflect this, but Resnet50 remains in need of these options to achieve maximum performance. Other usability improvements have been added, such as the compiler progress bar. As always, please let us know if there are other areas that we can improve.

New in this release

• Pytorch Yolo V4 is now supported.
• Added a compiler progress bar when compilation is invoked from the Framework. This allows the user to see that progress continues as compilation proceeds, which is useful when compilation takes several minutes. A dot is printed every 20 seconds.
• Improved inference performance of Tensorflow BERT base seq 256 batch 3 by 10%.

Resolved Issues

• Resolved issue with depthwise convolution that manifests as a type check error

[1.0.24045.0]
Date 11/17/2020

Summary

Improved performance for pipelined execution (NeuronCore Pipeline).
New in this release

• NeuronCore Pipeline: improved partitioning to enable better static weights loading to cache.

Resolved Issues

• --static-weights : No longer needed. As this is shown in some examples, please remove the option since the compiler now performs this auto-detection by default.

• --num-neuroncores renamed to --neuroncore-pipeline-cores. The prior option form is still functional (backwards compatible) and will be removed in future releases.

• --batching_en: Resolved compilation failure of ResNet50 FP32 batch 1 on Ubuntu16 when “--batching_en” was used.

[1.0.20600.0]

Date 9/22/2020

Summary

Various performance improvements - both compilation time and inference speed of object recognition models.
• Compiler optimization ‘-O2’ option is now enabled by default.

New in this release

• Improved inference performance of YOLO v3, YOLO v4, VGG16, SSD300. BERT models were improved by an additional 10%.

• Modified such that -O2 is now the default behavior and does not need to be specified. Note: some tutorials still explicitly specify “-O2”. These will be modified in forthcoming updates.

Resolved Issues

• Sped up compilation of large models that were taking hours to sub-40 minute.

[1.0.18001.0]

Date 8/08/2020
Summary
Various performance improvements.

New in this release
Improved performance of BERT base with -O2

Resolved Issues
- n/a

[1.0.17937.0]
Date 8/05/2020

Summary
Various improvements.

[1.0.16861.0]
Date 7/16/2020

Summary
This release has some bug fixes and some functional and performance improvements to support compilation of several neural networks.

New in this release
This release
- Supports compilation of PoseNet, tested for images of specific resolutions upto 736.
- Update the -O2 with a new memory allocator to reduce spilling to DRAM
- Improved performance of the ‘-O2’ on BERT base, and openpose pose network.
Resolved Issues

• Resolved compilation error in Vgg16 batch 1

Other Notes

• Some versions of Inception network may fail to compile in Tensorflow on Ubuntu 16 in conda environment. The symptom is neuron-cc backend data race error. As a workaround use Ubuntu 18, Amazon Linux 2, or virtual env, or use neuron-cc with flag -O2.

**Warning:** *Starting with Neuron 1.14.0, Ubuntu 16 is no longer supported*

[1.0.15275.0]

Date 6/11/2020

Summary

This release has some bug fixes and some functional and performance improvements to support compilation of several neural networks.

New in this release

This release

• Supports compilation of PoseNet for images of specific resolutions up to 400x400.
• Improves performance of resnet152.
• Supports a new command line option `-O2` that can help with handling of large tensor inputs for certain models.
• Increase NEFF versions to 1.0. This means new NEFFs compiled from this release forward are not compatible with older versions of Neuron Runtime prior to May, 2020 (1.0.6905.0) release. Please update the Neuron Runtime when using NEFF version 1.0.

Resolved Issues

• Compilation issues on prosotron encoder, decoder neural networks.

Other Notes

Dependencies

• This version creates NEFF 1.0 thus may require update of neuron-rtd if older than May 2020 release.

  dmlc_nvmm==1.0.2574.0    dmlc_topi==1.0.2574.0    dmlc_tvm==1.0.2574.0    inferentia_hwm==1.0.1362.0
  islpy==2018.2
Summary

Bug fixes and some functional and performance improvements to several neural networks.

New in this release

- This version supports compilation of unmodified Tensorflow BERT with batch size 1, 4, 6 for input sequence 128.
- Improved Tensorflow BERT batch 4 sequence 128 performance to 45% of the accelerator peak (from 34%).
- Support for MXNET BERT base batch 8 compilation
- Support for TF Resnet152 batch 2 compilation
- Most compiler messages are migrated from cout to logging mechanisms with verbosity control

Resolved Issues

- Fixed failure to compile unmodified Tensorflow BERT model for small batches
- Fixed run-to-run-variability in OneHot operator implementation
- Robustness improvements for ParallelWavenet and transformer decoder networks

Other Notes

Dependencies

dmlc_nvm==1.0.2356.0
dmlc_topi==1.0.2356.0
dmlc_tvm==1.0.2356.0
inherentia_hwm==1.0.1294.0
islpy==2018.2
AWS Neuron

Summary

Bug fixes and some functional and performance improvements to several neural networks.

New in this release

- Support compilation of modified SSD-300 (Running SSD300 with AWS Neuron)
- Improved inference performance in natural language processing networks (such as prosotron encoder) by 45%

Resolved Issues

- Eliminated redundant fp32 to bfloat16 cast on input and output tensors

Known issues and limitations

- See previous releases.

Other Notes

- Added support for faster iteration on recurrent networks (aka auto-loop)

Dependencies

```bash
dmlc_nnvm==1.0.2049.0
dmlc_topi==1.0.2049.0
pip install --upgrade dmlc_tvm==1.0.2049.0
inferentia_hwm==1.0.897.0
islpy==2018.2
```

[1.0.7878.0]

Date 2/27/2020

Summary

Bug fixes and minor performance improvements.
New in this release
None

Resolved Issues

• Corrected image resize operator functionality
• Compiler internal enhancements made that will benefit models such as BERT

Known issues and limitations

• See previous releases.

Other Notes

Dependencies

dmlc_nnvm-1.0.1826.0
dmlc_topi-1.0.1826.0
dmlc_tvm-1.0.1826.0
inferentia_hwm-1.0.897.0
islpy-2018.2

[1.0.6801.0]

Date 1/27/2020

Summary

Bug fixes and some performance enhancement related to data movement for BERT-type neural networks.

New in this release
None

Resolved Issues

• Improved throughput for operators processed in the Neuron Runtime CPU. As an example: execution of 4 single NeuronCore NEFF models of ResNet50 v2 float16 batch = 5 in parallel on an inf1.1xlarge sped up by 30%.
• Corrected shape handling in Gather(TensorFlow)/Take(MXNet) operators that are processed by the Neuron Runtime in the Neuron Runtime vCPU, which resolves a possible crash in Neuron Compiler when compiling models with these operators with some shapes.
• Added support for TensorFlow OneHot operator (as a Neuron Runtime CPU operator).
• Added more internal checking for compiler correctness with newly defined error messages for this case.
AWS Neuron

"Internal ERROR: Data race between Op1 'Name1(...) [...] and Op2 'Name2(...) [...]"

- Fixed out-of-memory issue introduced in 1.0.5939.0 such that some large models (BERT) compiled on instances with insufficient host memory would cause the runtime to crash with an invalid NEFF. This is actually a compiler error, but due to additional script layers wrapping this in the Running TensorFlow BERT-Large with AWS Neuron, this would have likely been seen as a runtime error like this:

```
˓→kernel failed: Invalid argument: neff is invalid
[{{node bert/NeuronOp}}]
```

**Known issues and limitations**

See previous release notes. Some tutorials show use of specific compiler options and flags, these are needed to help provide guidance to the compiler to achieve best performance in specific cases. Please do not use in cases other than as shown in the specific tutorial as results may not be defined. These options should be considered experimental and will be removed over time.

**Other Notes**

**Dependencies**

dmlc_nvm-1.0.1619.0
dmlc_topi-1.0.1619.0
dmlc_tvm-1.0.1619.0
inferentia_hwm-1.0.839.0
islpy-2018.2

[1.0.5939.0]

Date 12/20/2019

**Summary**

Bug fixes and some performance enhancement for NeuronCore Pipeline.

**New in this release**

**Resolved Issues**

- Fixed pipeline execution on more than 10 NeuronCores
- Improved NeuronCores Pipeline execution by improving data exchange efficiency between NeuronCores
- Added warning for unaligned memory access
- Fixed handling of cast on input FP32 tensor
• Improved handling of data layouts and transpose
• Improved dead-code elimination
• Improved efficiency of compute engine synchronization
• Improved efficiency of data transfers within the Neuron code

**Known issues and limitations**

See previous release notes. Some tutorials show use of specific compiler options and flags, these are needed to help provide guidance to the compiler to achieve best performance in specific cases. Please do not use in cases other than as shown in the specific tutorial as results may not be defined. These options should be considered experimental and will be removed over time.

**Other Notes**

**Dependencies**

• dmlc_nnvm-1.0.1416.0
• dmlc_topi-1.0.1416.0
• dmlc_tvm-1.0.1416.0
• inferentia_hwm-1.0.720.0
• islpy-2018.2

[1.0.5301.0]

Date 12/1/2019

**Summary**

**New in this release**

**Resolved Issues**

• Added warning for unsupported operators and convolution sizes
• Added warning for unsupported layout / upsampling
• Added support for Relu6, AddV2, BatchMatmulV2 operators
• Added support for default MXNet outputs in –io-config
• Improved performance of batched inference for convolutional networks
• Fixed MatMult column size 1
• Fixed bf16 constant loading
• Fixed Conv2D tile accumulation
Known Issues and Limitations

See previous release notes. Resolved issues are shown in Resolved Issues.

Other Notes

Please install g++ on AMIs without g++ pre-installed (i.e. server AMIs):

```bash
# Ubuntu
sudo apt-get install -y g++
```

```bash
# Amazon Linux
sudo yum install -y gcc-c++
```

Supported Python versions:

- 3.5, 3.6, 3.7

Supported Linux distributions:

- Ubuntu 16, Ubuntu 18, Amazon Linux 2

Dependencies

- dmlc_nvm-1.0.1328.0
- dmlc_topi-1.0.1328.0
- dmlc_tvm-1.0.1328.0
- inferentia_hwm-1.0.674.0
- islpy-2018.2

[1.0.4680.0]

Date: 11/25/2019

New in this release

N/A, this is the first release.

Resolved issues

N/A, this is the first release.
Known issues and limitations

1. **Control flow** Inferentia has a limited support for control flow. In general, Neuron can only support control flow operators which are static at compile time, i.e. static length RNN, top-k, sort, ...

2. **Size of neural network** The size of neural network is influenced by a) type of neural network (CNN, LSTM, MLP), b) number of layers, c) sizes of input (dimension of the tensors, batch size, ...). The current Neuron compiler release has a limitation in terms of the size of neural network it could effectively optimize. As a result, we limit CNN models (e.g. ResNet) to have an input size of up to 480x480 FP16, batch size of 4; LSTM models (e.g. GNMT) are limited to a time step limit of up to 900; MLP models (like BERT) are limited up to sequence-length equal 128, batch=8.

3. **Data layout** The Neuron compiler supports multiple data layout format (NCHW, NHWC, ...). Non-CNHW input/output data-layouts will require Neuron to insert additional transpose operations, causing a degradation in performance.

4. **Object detection models** Computer-vision object detection and segmentation models are not supported by the current release.

5. **Reduce data type** INT8 data type is not currently supported by the Neuron compiler.

6. **Tensor residency** When a sub-graph that is executed on the host is communicating with a sub-graph that is executing on Neuron cores, tensors are copied via the communication queues between the host and Inferentia memory for each inference, which may result in end-to-end performance degradation.

7. **Primary inputs in NeuronCore Pipeline mode** When a neural network is executed in NeuronCore Pipeline mode, only the first operator in a neural network can receive primary inputs from the host.

Other Notes

**Dependencies**

- nnvm: dmlc_nvm-1.0.1219.0
- topi: dmlc_topi-1.0.1219.0
- tvm: dmlc_tvm-1.0.1219.0
- hwm: inferentia_hwm-1.0.602.0
- islpy: islpy-2018.2+aws2018.x.73.0

**14.2.2 Neuron Supported operators**
Neuron runtime consists of kernel driver and C/C++ libraries which provides APIs to access Inferentia devices. Machine learning frameworks (TensorFlow, PyTorch and Apache Mxnet) uses Neuron runtime to execute models on the Neuron Cores. Neuron runtime load compiled deep learning models, also referred to as Neuron Executable File Format (NEFF) to the Inferentia chips to execute inference requests. Runtime is optimized for high-throughput and low-latency to meet customers ML applications requirements. In typical environment, Neuron runtime is transparent for users. User application will communicate with Neuron runtime through framework (TensorFlow, PyTorch, MxNet) API.

First generation of Neuron runtime (Neuron Runtime 1.x) was delivered as daemon (neuron-rtd) which provided GRPC API to load and execute a ML model. Runtime 1.x was available before Neuron 1.16.0.

Second generation of Neuron runtime (Neuron Runtime 2.x) is available starting Neuron 1.16.0 and is delivered as a shared library (libnrt.so), libnrt.so is installed together with the framework of choice, it improves the performance and ease of use by providing C++ library to load and execute ML models.

Visit Introducing Neuron Runtime 2.x (libnrt.so) for more information.

15.1 Neuron Runtime 2.x

15.1.1 Neuron Runtime Configuration

Runtime is responsible of executing ML models on Neuron devices and it determines which NeuronCore will execute which model and how to execute it. User application should configure the Runtime to change the default behavior. Runtime can be configured through environmental variables, in most cases Neuron framework extensions will take care of the proper configuration in other cases the user may need to explicitly configure the runtime to achieve the desired behavior.

This guide provides an overview of the different environment variables available to configure Neuron runtime behavior.
Table 1: Environment Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Expected Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEURON_RT_VISIBLE_CORES</td>
<td>Range of specific NeuronCores needed by the process</td>
<td>Integer range</td>
<td>Any value or range between 0 to Max NeuronCore in the system.</td>
<td>None</td>
</tr>
<tr>
<td>NEURON_RT_LOG_LOCATION</td>
<td>Runtime log location</td>
<td>string</td>
<td>console or syslog</td>
<td>console</td>
</tr>
<tr>
<td>NEURON_RT_LOG_LEVEL</td>
<td>Runtime log verbose level</td>
<td>string</td>
<td>ERROR, WARNING, INFO, DEBUG, TRACE</td>
<td>ERROR</td>
</tr>
<tr>
<td>NEURON_RT_EXEC_TIMEOUT</td>
<td>Timeout for execution in seconds</td>
<td>Integer</td>
<td>0 to INT_MAX</td>
<td>2</td>
</tr>
<tr>
<td>NEURON_RT_VALIDATE_HASH</td>
<td>Validate NEFF contents before loading into accelerator</td>
<td>Boolean</td>
<td>TRUE or FALSE</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

**NeuronCore Allocation with NEURON_RT_VISIBLE_CORES**

**Important:** NEURONCORE_GROUP_SIZES is being deprecated, if your application is using NEURONCORE_GROUP_SIZES please see Migrate your application to Neuron Runtime 2.x (libnrt.so) for more details.

By default, Neuron Runtime initializes all the cores present in the system and reserves them for the current process.

**Note:** Once a NeuronCore is reserved for a process, it can't be used by another process at all, until the process reserving that NeuronCore dies.

For parallel processing, it is necessary multiple processes need to use different NeuronCores. For this purpose `NEURON_RT_VISIBLE_CORES` can be used which controls what NeuronCores the process would reserve. This variable takes a NeuronCore index or an inclusive range.

For example, if an application(myapp.py) requires one NeuronCore, then it can be started with NEURON_RT_VISIBLE_CORES=0 to use only NeuronCore 0. To do parallel processing, multiple processes can be started (without any change to application) with different NEURON_RT_VISIBLE_CORES values. Here is an example which runs myapp.py on inf1.xl parallely by using different NeuronCores available.

| NEURON_RT_VISIBLE_CORES=0 myapp.py |
| NEURON_RT_VISIBLE_CORES=1 myapp.py |
| NEURON_RT_VISIBLE_CORES=2 myapp.py |
| NEURON_RT_VISIBLE_CORES=3 myapp.py |

Another example, where myapp2.py requires 3 NeuronCores and being run on inf1.6xl. In the following example, the first instance of myapp2 would use NeuronCores 0, 1 and 2, then next instance would use 3, 4, and 4 and so on.

| NEURON_RT_VISIBLE_CORES=0-2 myapp2.py |
| NEURON_RT_VISIBLE_CORES=3-5 myapp2.py |
| NEURON_RT_VISIBLE_CORES=6-8 myapp2.py |
| NEURON_RT_VISIBLE_CORES=9-11 myapp2.py |
| NEURON_RT_VISIBLE_CORES=12-14 myapp2.py |
Notes

1. Number of NeuronCores in a inferentia device is 4
2. Number of inferentia is depends on the instance size.
3. The NeuronCore index in NEURON_RT_VISBILE_CORES starts from 0 and ends at (number of NeuronDe-
   vices * number of NeuronCores) - 1.

Logging and debug-ability

By default, Neuron Runtime logs to syslog with verbose level of INFO and only ERROR s are logged in console. The
following code snippet shows ways to increase/decrease the log level.

```bash
NEURON_RT_LOG_LEVEL=INFO myapp.py  # Sets the log level for syslog and console.
NEURON_RT_LOG_LOCATION=console NEURON_RT_LOG_LEVEL=QUIET myapp.py  # Completely
→disables console logging.
```

By default, Neuron Runtime expects the NeuronCore to complete execution of any model with in 2 seconds. If
NeuronCore didn't complete the execution within 2 seconds then runtime would fail the execution with timeout er-
ror. Most of the models takes few milliseconds to complete so 2 seconds(2000 milliseconds) is more than ade-
quate. However if your model is expected to run more than 2 seconds then you can increase the timeout with NEU-
RON_RT_EXEC_TIMEOUT.

```bash
NEURON_RT_EXEC_TIMEOUT=5 myapp.py  # increases the timeout to 5 seconds
```

Checksum

To execute a model(NEFF), Neuron Runtime needs to load the NEFF file into NeuronCore and run. Neuron Runtime
provides a way to do checksum validation on each NEFF file while loading to validate the file is not corrupted. This
option is off by default to avoid performance penalty during model load time(~50%).

```bash
NEURON_RT_VALIDATE_HASH=true myapp1.py  # enables model checksum validation while
→loading
NEURON_RT_VALIDATE_HASH=false myapp2.py  # disables(default) model checksum
→validation while loading
```

15.1.2 Troubleshooting Neuron Runtime

This document aims to provide more information on how to fix issues you might encounter while using the Neuron
Runtime 2.x or above. For each issue we will provide an explanation of what happened and what can potentially
correct the issue.

If your issue is not listed below or you have a more nuanced problem, contact us via issues posted to this repo, the
AWS Neuron developer forum, or through AWS support.
Neuron Driver installation fails

aws-neuron-dkms is a driver package which needs to be compiled during installation. The compilation requires kernel headers for the instance’s kernel. `uname -r` can be used to find kernel version in the instance. In some cases, the installed kernel headers might be newer than the instance’s kernel itself.

Please look at the aws-neuron-dkms installation log for message like the following:

```
Building for 4.14.193-149.317.amzn2.x86_64
Module build for kernel 4.14.193-149.317.amzn2.x86_64 was skipped since the kernel headers for this kernel does not seem to be installed.
```

If installation log is not available, check whether the module is loaded.

```
$ lsmod | grep neuron
```

If the above has no output then that means `aws-neuron-dkms` installation is failed.

**Solution**

1. Stop all applications using the NeuronCores.
2. Uninstall `aws-neuron-dkms`
   
   ```
   sudo apt remove aws-neuron-dkms  
   or  
   sudo yum remove aws-neuron-dkms
   ```
3. Install kernel headers for the current kernel
   
   ```
   sudo apt install -y linux-headers-$(uname -r)  
   or  
   sudo yum install -y kernel-devel-$(uname -r) kernel-headers-$(uname -r)
   ```
4. Install `aws-neuron-dkms`
   
   ```
   sudo apt install aws-neuron-dkms  
   or  
   sudo yum install aws-neuron-dkms
   ```

Application fails to start

Neuron Runtime requires Neuron Driver(aws-neuron-dkms package) to access Neuron devices. If the driver is not installed then Neuron Runtime wont able to access the Neuron devices and will fail with an error message in console and syslog.

If `aws-neuron-dkms` is not installed then the error message will be like the following:

```
```

If `aws-neuron-dkms` is installed but does not support the latest runtime then the error message will be like the following:

```
```
Solution

Please follow the installation steps in *Setup Guide* to install `aws-neuron-dkms`.

**Neuron Core is in use**

A Neuron Core cannot be shared between two applications. If an application started using a Neuron Core all other applications trying to use the Neuron Core would fail during runtime initialization with the following message in the console and in syslog:

```
2021-Aug-27 23:22:12.0323 28078:28078 ERROR NRT:nrt_allocate_neuron_cores → NeuronCore(s) not available - Requested:nc1-nc1 Available:0
```

**Solution**

Terminate the process using Neuron Core and then try launching the application again.

**Unsupported NEFF Version**

While loading a model (NEFF), Neuron Runtime checks the version compatibility. If the version the NEFF is incompatible with Runtime then it would fail the model load with following error message:

```
NEFF version mismatch supported: 1.1 received: 2.0
```

**Solution**

Use compatible versions of Neuron Compiler and Runtime. Updating to the latest version of both Neuron Compiler and Neuron Runtime is the simplest solution. If updating one of the two is not an option, please refer to the *Neuron Runtime 2.x Release Notes* of the Neuron Runtime to determine NEFF version support.

**Insufficient Memory**

While loading a model (NEFF), Neuron Runtime reserves both device and host memory for storing weights, ifmap and ofmap of the Model. The memory consumption of each model is different. If Neuron Runtime is unable to allocate memory then the model load would fail with the following message in syslog:

```
kernel: [XXXXX] neuron:mc_alloc: device mempool [0:0] total 1073741568 occupied → 960539030 needed 1272 available 768
```
Solution

As the error is contextual to what’s going on with your instance, the exact next step is unclear. Try unloading some of the loaded models which will free up device DRAM space. If this is still a problem, moving to a larger Inf1 instance size with additional NeuronCores may help.

Insufficient number of NeuronCores

The NEFF requires more NeuronCores than available on the instance.

Check for error messages in syslog similar to:

```
NRT: 26638:26638 ERROR TDRV:db_vtpb_get_mla_and_tpb Could not find
--> VNC id n
NRT: 26638:26638 ERROR NMGR:dlr_kelf_stage Failed to
--> create shared io
NRT: 26638:26638 ERROR NMGR:stage_kelf_models Failed to stage
--> graph: kelf-a.json to NeuronCore
NRT: 26638:26638 ERROR NMGR:kmgr_load_nn_post_metrics Failed to load
--> NN: xxxxxxx, err: 2
```

Solution

The NeuronCores may be in use by models you are not actively using. Ensure you’ve unloaded models you’re not using and terminated unused applications. If this is still a problem, moving to a larger Inf1 instance size with additional NeuronCores may help.

Numerical Error

Neuron Devices will detect any NaN generated during execution and report it. If Neuron Runtime sees NaNs are generated then it would fail the execution request with Numerical Error with the following message:

```
nrtd[nnnnn]: .... Error notifications found on NC .... INFER_ERROR_SUBTYPE_NUMERICAL
```

Solution

This usually an indication of either error in the model or error in the input.

Report issue to Neuron by posting the relevant details on GitHub issues.
15.1.3 What’s New

Neuron Runtime 2.x Release Notes

- **NEFF Support Table:**
  - *Neuron Runtime 2.x (libnrt.so) release [2.2.18.0]*
  - *Neuron Runtime 2.x (libnrt.so) release [2.2.15.0]*

**NEFF Support Table:**

Use this table to determine the version of Runtime that will support the version of NEFF you are using. NEFF version is determined by the version of the Neuron Compiler.

<table>
<thead>
<tr>
<th>NEFF Version</th>
<th>Runtime Version Range</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.6</td>
<td>*</td>
<td>All versions of RT support NEFF 0.6</td>
</tr>
<tr>
<td>1.0</td>
<td>&gt;= 1.0.6905.0</td>
<td>Starting support for 1.0 NEFFs</td>
</tr>
<tr>
<td>2.0</td>
<td>&gt;= 1.6.5.0</td>
<td>Starting support for 2.0 NEFFs</td>
</tr>
</tbody>
</table>

**Neuron Runtime 2.x (libnrt.so) release [2.2.18.0]**

Date: 11/05/2021

- Resolved an issue that affect the use of Neuron within container. In previous Neuron Runtime release (libnrt.so.2.2.15.0), when /dev/neuron0 was not used by the application, Neuron Runtime attempted and failed to initialize /dev/neuron0 because user didn’t pass /dev/neuron0 to the container. This Neuron Runtime release (libnrt.so.2.2.18.0) allows customers to launch containers with specific NeuronDevices other than /dev/neuron0.

**Neuron Runtime 2.x (libnrt.so) release [2.2.15.0]**

Date: 10/27/2021

**New in this release**

- *First release of Neuron Runtime 2.x* - In this release we are introducing Neuron Runtime 2.x which is a shared library named (libnrt.so) and replacing Neuron Runtime 1.x server (neruon-rtd). Upgrading to libnrt.so improves throughput and latency, simplifies Neuron installation and upgrade process, introduces new capabilities for allocating NeuronCores to applications, streamlines container creation, and deprecates tools that are no longer needed. The new library-based runtime (libnrt.so) is integrated into Neuron’s ML Frameworks (with the exception of MXNet 1.5) and Neuron Tools packages directly - users no longer need to install/deploy the aws-neuron-runtime package.

**Important:**

- You must update to the latest Neuron Driver (aws-neuron-dkms version 2.1 or newer) for proper functionality of the new runtime library.
Neuron Driver 2.x Release Notes

- Neuron Driver release [2.2.6.0]
- Neuron Driver release [2.1]

Neuron Driver release [2.2.6.0]

Date: 10/27/2021

New in this release

- Memory improvements made to ensure all allocations are made with 4K alignments.

Resolved issues

- No longer delays 1s per NeuronDevice when closing Neuron Tools applications.
- Fixes a Ubuntu 20 build issue

Neuron Driver release [2.1]

- Support is added for Neuron Runtime 2.x (libnrt.so).
- Support for previous releases of Neuron Runtime 1.x is continued with Driver 2.x releases.

15.2 Neuron Runtime 1.x

15.2.1 Neuron Runtime 1.x

Neuron Runtime 1.x was available before Neuron 1.16.0, it is a userspace application that provides developers flexibility to deploy their inference applications, and optimize for high-throughput and low-latency to meet their specific application needs. Neuron runtime takes compiled models, also referred to as Neuron Executable File Format (NEFF), and loads them to the Inferentia chips to execute inference requests.

Neuron runtime provides the ability to control where a model is deployed to, and how the inferences are executed in the system. For example, using runtime commands developers can assign different models to separate NeuronCore Groups in a flexible and scalable way. This allows to run the same or multiple models to maximize the hardware utilization to ensure it fits their specific application requirements.
To get started, read the Neuron runtime *Getting started: Installing and Configuring Neuron-RTD* guide. If your application is container-based, learn how to integrate it with containers by referring to our *Containers Section*.

The Neuron runtime is prebuilt into the AWS DLAMI, but developers can also install it in their own environments, which can be custom AMIs or containers.

Looking for support? Please checkout our *Troubleshooting Neuron Runtime* doc or contact us directly by filing an issue.

**Getting started: Installing and Configuring Neuron-RTD**

In this getting started guide you will learn how to install Neuron runtime, and configure it for inference.

**Step 1: Launch an Inf1 Instance and Install runtime packages**

1. Select an AMI of your choice, which may be Ubuntu 16.x, Ubuntu 18.x, or Amazon Linux 2 based. Refer to the *Setup Guide* for details.
2. Select an Inf1 instance size of your choice (see https://aws.amazon.com/ec2/instance-types/inf1/)

**Step 2: Configure Neuron-RTD**

You can choose your Neuron-RTD mode, either select to run a single instance of the Neuron runtime, or multiple instances which may be desired to provide your application capabilities like isolation or load balancing.

**Single Neuron-RTD**

The default configuration sets up a single Neuron-RTD daemon for all present Neuron devices in the instance. With the default configuration:

1. Runtime API server listens on a single UDS endpoint `unix:/run/neuron.sock`
2. A single runtime daemon(multi threaded) handles all the inference requests.

**Multiple Neuron-RTD**

Multiple runtime daemon might be preferred in some cases for isolation or for load balancing.

When configuring multiple Neuron-RTD, a configuration file needs to be created to specify the API server endpoint (UDP or TCP port) and logical device id it should manage.

The following steps explains configuring four Neuron-RTD on an inf1.6xl instance and let each daemon to manage 1 Neuron device.
AWS Neuron

Identify logical IDs of Neuron Devices

Use `neuron-ls` to enumerate the set of Neuron Devices available in the system.

```
<table>
<thead>
<tr>
<th>NEURON</th>
<th>NEURON</th>
<th>NEURON</th>
<th>CONNECTED</th>
<th>PCI</th>
<th>RUNTIME</th>
<th>RUNTIME</th>
<th>RUNTIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVICE</td>
<td>CORES</td>
<td>MEMORY</td>
<td>DEVICES</td>
<td>BDF</td>
<td>ADDRESS</td>
<td>PID</td>
<td>VERSION</td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
<td>--------</td>
<td>-----------</td>
<td>----------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>8 GB</td>
<td>1</td>
<td>0000:00:1c.0</td>
<td>NA</td>
<td>12410</td>
<td>NA</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>8 GB</td>
<td>2, 0</td>
<td>0000:00:1d.0</td>
<td>NA</td>
<td>12410</td>
<td>NA</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8 GB</td>
<td>3, 1</td>
<td>0000:00:1e.0</td>
<td>NA</td>
<td>12410</td>
<td>NA</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>8 GB</td>
<td>2</td>
<td>0000:00:1f.0</td>
<td>NA</td>
<td>12410</td>
<td>NA</td>
</tr>
</tbody>
</table>
```

`neuron-rtd` can manage one or more devices. Select contiguous Neuron devices to be managed by a single `neuron-rtd`.

**Create a configuration file for each instance**

Create a configuration file for each `neuron-rtd` you wish to launch, with one or more Neuron Devices desired to be mapped to that `neuron-rtd` instance, and the listening port for it.

```
sudo tee /opt/aws/neuron/bin/nrtd0.json > /dev/null << EOF
{
"name": "nrtd0",
"server_port": "unix:/run/neuron.sock0",
"devices": [0]
}
EOF

dsudo tee /opt/aws/neuron/bin/nrtd1.json > /dev/null << EOF
{
"name": "nrtd1",
"server_port": "unix:/run/neuron.sock1",
"devices": [1]
}
EOF

dsudo tee /opt/aws/neuron/bin/nrtd2.json > /dev/null << EOF
{
"name": "nrtd2",
"server_port": "unix:/run/neuron.sock2",
"devices": [2]
}
EOF

dsudo tee /opt/aws/neuron/bin/nrtd3.json > /dev/null << EOF
{
"name": "nrtd3",
"server_port": "unix:/run/neuron.sock3",
"devices": [3]
}
EOF

sudo chmod 755 /opt/aws/neuron/bin/nrtd0.json
sudo chmod 755 /opt/aws/neuron/bin/nrtd1.json
```

(continues on next page)
AWS Neuron

(continued from previous page)

Start the services

Stop the default service

Start the new services

Verify the services are up and running. This example shows one of the Neuron-RTD daemons (Neuron-RTD0):

```
sudo systemctl status neuron-rtd@nrtd0
neuron-rtd@nrtd0.service - Neuron Runtime Daemon nrtd0
    Loaded: loaded (/lib/systemd/system/neuron-rtd@.service; disabled; vendor preset: → enabled)
    Active: active (running) since Wed 2019-11-13 00:24:25 UTC; 8s ago
Main PID: 32446 (neuron-rtd)
    Tasks: 14 (limit: 4915)
    CGroup: /system.slice/system-neuron\x2drtd.slice/neuron-rtd@nrtd0.service
[32446 /opt/aws/neuron/bin/neuron-rtd -i nrtd0 -c /opt/aws/neuron/config/ → neuron-rtd.config
Nov 13 00:23:39 ip-10-1-255-226 neuron-rtd[32446]: nrtd[32446]: [TDRV:reset_mla] → Resetting 0000:00:1f.0
Nov 13 00:23:39 ip-10-1-255-226 nrtd[32446]: [TDRV:reset_mla] Resetting 0000:00:1f.0
Nov 13 00:24:00 ip-10-1-255-226 neuron-rtd[32446]: nrtd[32446]: [hal] request seq: 3, → cmd: 1 timed out
Nov 13 00:24:00 ip-10-1-255-226 nrtd[32446]: nrtd[32446]: [TDRV:tdrv_init_one_ → mla_phase2] Initialized Inferentia: 0000:00:1f.0
Nov 13 00:24:25 ip-10-1-255-226 nrtd[32446]: [TDRV:tdrv_init_one_mla_phase2] → Initialized Inferentia: 0000:00:1f.0
Nov 13 00:24:25 ip-10-1-255-226 nrtd[32446]: [TDRV:reset_mla] Resetting 0000:00:1f.0
Nov 13 00:24:25 ip-10-1-255-226 neuron-rtd[32446]: nrtd[32446]: [TDRV:tdrv_init_one_ → mla_phase2] Initialized Inferentia: 0000:00:1f.0
Nov 13 00:24:25 ip-10-1-255-226 nrtd[32446]: [TDRV:tdrv_init_one_mla_phase2] → Initialized Inferentia: 0000:00:1f.0
Nov 13 00:24:25 ip-10-1-255-226 neuron-rtd[32446]: nrtd[32446]: [TDRV:reset_mla] Resetting 0000:00:1f.0
```

15.2. Neuron Runtime 1.x
Using the Neuron-RTD services with ML Frameworks

When multiple Neuron-RTD instances are launched with user configurations as described in previous sections, NEURON_RTD_ADDRESS environment variable setting is required to use the framework. To use a particular Neuron-RTD instance with a framework, set the environment variable NEURON_RTD_ADDRESS to the socket address of the Neuron-RTD instance. For example, to run a framework process with nrtd2 in the example above, do: .. code::

   bash
   NEURON_RTD_ADDRESS=unix:/run/neuron.sock2 python <framework script>

When a single Neuron-RTD instance is launched with default configuration, no special environment variable setting is required to use the framework.

Neuron-RTD Configurable Parameters

This guide provides an overview of the different parameters available to configure Neuron runtime behavior.

Global Runtime Configuration

These parameters are defined in neuron-rtd.config and affect global runtime configuration. Note that Neuron runtime must be restarted after changes to the configuration file for them to take effect.

Model Directory Caching:

One of the most time consuming stages in model loading is the unpackaging of the NEFF file to a temporary directory for the runtime to digest. To mitigate this cost for repeated loads of the same model, caching can be turned on by giving an integer value to the `model_cache_count` field in neuron-rtd.config to set a threshold on the number of unpacked model directories that the runtime can keep around. Keyed based on NEFF UUID, the runtime will check for an existing mapping to a cached directory and reuse it if found. The cache employs simple LRU eviction when full.

Per-Model GRPC Load Parameters:

These are optional parameters that can accompany a model `load()` API call to set certain behaviors for that specific model. Note that some of these parameters can also have a default value configurable in neuron-rtd.config that will apply to every model that does not provide that parameter during `load()`.

Per-inference timeout:

The maximum amount of time in seconds spent waiting for each inference to complete can be configured by passing an integer value to the `timeout` parameter. If the timeout is reached, the runtime will immediately return TIMEOUT (error code 5) regardless of the eventual status of the inference.

The default timeout value is 2 seconds. It can be modified in the neuron-rtd.config file.
Inference queue size:

More than one inference request could be posted concurrently up to the inference queue size limit. Having inference requests in the queue allows the runtime to prepare the next set of inputs while the previous inference is running on the hardware thus increasing the overall throughput.

Interface queue size can be adjusted for each model by passing an integer value to the ‘ninfer’ parameter. A global inference queue size default can be specified in the neuron-rtd.config file. The default value is 4.

Input staging location:

Inference inputs can be configured to be staged in either host or device memory prior to starting an inference by passing a boolean flag to the io_data_host parameter. A value of true stages the data in host memory, while false stages the data in the device. Bandwidth is much higher from the device than the host to the chip, so staging on the device can be beneficial for models with large input loads that would otherwise cause a bottleneck during transfer. Note that this does introduce an extra step during inference posting to transfer the input to the device, so it may negatively affect single-inference latency. This option is most useful when paired with concurrent, pipelined inferences (with an appropriate ninfer value) so that inference execution in hardware can hide the extra overhead of staging.

A global default can be specified in the config file with the io_dma_data_host flag. Default value on installation is false.

Neuron Runtime return codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NERR_OK</td>
<td>Successful completion.</td>
</tr>
<tr>
<td>1</td>
<td>NERR_FAIL</td>
<td>Non specific failure, e.g. low level interface to the hardware error.</td>
</tr>
<tr>
<td>2</td>
<td>NERR_INVALID</td>
<td>Invalid/corrupted NEFF file, bad Neuron instruction, invalid access to Neuron memory, etc.</td>
</tr>
<tr>
<td>3</td>
<td>NERR_INVALID_HANDLE</td>
<td>Bad handle was passed to the request, e.g. for a Neural Network that has not been loaded or has been previously unloaded.</td>
</tr>
<tr>
<td>4</td>
<td>NERR_RESOURCE</td>
<td>Failed to allocate a resource for requested operation, for example: not enough TDRAM to load a Neural Network, not enough Host memory to perform an operation.</td>
</tr>
<tr>
<td>5</td>
<td>NERR_TIMEOUT</td>
<td>Request timed out.</td>
</tr>
<tr>
<td>6</td>
<td>NERR_HW_ERROR</td>
<td>RT failed to initialize, after a number of failed attempts to start RT stays up and returns the error in response to every GRPC call. Common causes: bad hugepages configuration (insufficient number of reserved pages on the system), failure to initialize Neuron device.</td>
</tr>
<tr>
<td>7</td>
<td>NERR_QUEUE_FULL</td>
<td>Not enough space in the inference input queue. The number of submitted and not completed inference requests is greater that what’s been configured during NN load. This is a transient error, inference requests can be submitted after some of the in-flight inferences have completed.</td>
</tr>
<tr>
<td>9</td>
<td>NERR_RESOURCE_NC</td>
<td>Not enough available NCs to load a Neural Network.</td>
</tr>
<tr>
<td>10</td>
<td>NERR_UNSUPPORTED_VERSION</td>
<td>NN load failed because the version of NEFF is not supported.</td>
</tr>
<tr>
<td>1000</td>
<td>NERR_INFER_PENDING</td>
<td>Inference has not completed yet.</td>
</tr>
<tr>
<td>1002</td>
<td>NERR_INFER_BAD_INPUT</td>
<td>Bad input has been submitted to infer, e.g. missing some of the input tensors, incorrect input tensor sizes.</td>
</tr>
<tr>
<td>1003</td>
<td>NERR_INFER_COMPLETED_WITH_NUM_ERR</td>
<td>Inference is completed. Numerical errors were encountered while executing the inference.</td>
</tr>
<tr>
<td>1004</td>
<td>NERR_INFER_COMPLETED_WITH_ERR</td>
<td>Inference is completed. Non-numerical errors were encountered while executing the inference.</td>
</tr>
</tbody>
</table>

15.2. Neuron Runtime 1.x

1229
Troubleshooting Neuron Runtime

This document aims to provide more information on how to fix issues you might encounter while using the Neuron Runtime 1.1 or above. For each issue we will provide an explanation of what happened and what can potentially correct the issue.

If you haven’t read it already, please familiarize yourself with our Getting started: Installing and Configuring Neuron-RTD documentation and usage examples. If your issue is still not resolved or you have a more nuanced problem, contact us via issues posted to this repo, the AWS Neuron developer forum, or through AWS support.

Topics

What is going wrong?
Runtime installation failed
Neuron Runtime services fail to start
Load model failure
Inferences are failing

Additional helpers:
Getting started: Installing and Configuring Neuron-RTD
Neuron Runtime return codes

Runtime installation failed

Refer to the Setup Guide for details.

Neuron Runtime service fails to start

If the neuron-rtd service is failing to start, you may be experiencing failure due to (1) a conflict with another instance of neuron-rtd, (2) neuron driver(aws-neuron-dkms) package is not installed.

Neuron Driver is not installed

Neuron Runtime requires Neuron Driver(aws-neuron-dkms) to access inf1 devices. If the driver is not installed then Neuron Runtime service wont start.

systemctl status command can be used to check whether neuron-rtd is active or not. If Neuron Driver is not installed then output would look similar to the following
Please follow the installation steps in Setup Guide to install aws-neuron-dkms package and then restart runtime using `sudo systemctl restart neuron-rtd` command.

**Neuron Driver installation fails**

aws-neuron-dkms is a driver package which needs to be compiled during installation. The compilation requires kernel headers for the instance’s kernel. `uname -r` can be used to find kernel version in the instance. In some cases, the installed kernel headers might be newer than the instance’s kernel itself.

Please look at the aws-neuron-dkms installation log for message like the following:

```
Building for 4.14.193-149.317.amzn2.x86_64
Module build for kernel 4.14.193-149.317.amzn2.x86_64 was skipped since the
kernel headers for this kernel does not seem to be installed.
```

If installation log is not available, check whether the module is loaded.

```
$ lsmod | grep neuron
```

If the above has no output then that means aws-neuron-dkms installation is failed.

1. Uninstall `aws-neuron-dkms` `sudo apt remove aws-neuron-dkms` or `sudo yum remove aws-neuron-dkms`
2. Install kernel headers for the current kernel `sudo apt install -y linux-headers-$\{uname -r\}` or `sudo yum install -y kernel-devel-$\{uname -r\} kernel-headers-$\{uname -r\}`
3. Install `aws-neuron-dkms` `sudo apt install aws-neuron-dkms` or `sudo yum install aws-neuron-dkms`
4. Restart runtime using `sudo systemctl restart neuron-rtd` command.

**Another Instance of Runtime is Running**

A new instance of neuron-rtd cannot start if another neuron-rtd is already running and bound to the same Neuron devices. Please read on for how to detect this scenario, but if you’re having trouble configuring two or more runtimes on the same Inf1 instance, see detailed config instructions at Multiple Neuron-RTD.

Check for error messages in syslog similar to:

```
Oct 16 01:07:00 xxxxxxxx kernel: [ 7638.723761] neuron:ncdev_device_init: device_˓→inuse by pid:9428
```

Terminate the current neuron-rtd that is already running before starting the new instance.

```
sudo systemctl stop neuron-rtd
```
Load Model Failure

There are a variety of reasons for a model load to fail. The most common ones are listed below. If the solutions below are insufficient, please reach out to the Neuron team by posting the relevant details on GitHub issues.

Neff couldn’t be extracted

Host ran out of disk space while trying to extract the NEFF object

Syslog will show an error similar to the following:

```
 nrtd[nnnnn]: .... Failed to untar (tar -xsvf /tmp/neff.XXXXX -C /tmp/neff.YYYYY > /dev/null)
```

Increase /tmp space by removing unused files or taking other measures to increase the available disk size under /tmp.

Unsupported NEFF Version

The version of the NEFF file is incompatible with the version of Neuron Runtime that has received it. The NEFF is generated by the compiler and the Neuron Runtime is intended to support multiple NEFF versions; however, this may require updating the runtime to gain support for newer NEFF formats.

Check for error messages in syslog similar to:

```
 nrtd[nnnnn]: .... NEFF version mismatch supported: 1.1 received: 2.0
```

Error Code: 10

Use compatible versions of Neuron Compiler and Runtime. Updating to the latest version of both Neuron Compiler and Neuron Runtime is the simplest solution. If updating one of the two is not an option, please refer to the Neuron Runtime 2.x Release Notes of the Neuron Runtime to determine NEFF version support.

Invalid NEFF

Validation is performed on the NEFF file before attempting to load it. When that validation fails, it usually indicates that the compiler produced an invalid NEFF (possible bug in the Neuron Compiler).

Check for error messages in syslog similar to:

```
 nrtd[nnnn]: .... Failed .... neff.json
 nrtd[nnnn]: .... Failed/Unsupported/Invalid .... NEFF
 nrtd[nnnn]: .... Wrong NEFF file size
 nrtd[nnnn]: .... NEFF upload failed
```

Error Code: 2

Try recompiling with the latest version of Neuron Compiler. If that does not work, report issue to Neuron by posting the relevant details on GitHub issues.
Bad Memory Access by NEFF

To ensure the execution of the NEFF, neuron-rtd is monitoring for illegal and unaligned access to Inferentia memory. When this occurs, the NEFF will fail to load.

Check for error messages in syslog similar to:

```
nrtd[nnnn]:.... address ... must be X byte aligned
```

Error Code: 2

Report issue to Neuron by posting the relevant details on GitHub issues.

Insufficient resources

Loading the NEFF requires more host or Inferentia resources (usually memory on the host or Inferentia) then available on the instance. This issue may be contextual in that other applications or models consumed the needed resources before the current NEFF could be loaded.

Check for error messages in syslog similar to:

```
kernel: [XXXXX] neuron:mc_alloc: device mempool [0:0] total 1073741568 occupied
˓→960539030 needed 1272 available 768
```

Error Code: 4

As the error is contextual to what’s going on with your instance, the exact next step is unclear. Try unloading some of the loaded models which will free up device DRAM space. If this is still a problem, moving to a larger Inf1 instance size with additional NeuronCores may help.

Insufficient number of NeuronCores available to load a NEFF

The NEFF requires more NeuronCores than available on the instance.

Check for error messages in syslog similar to:

```
nrtd[nnnn]:.... Requested number of NCs: X exceeds the total available number: Y
nrtd[nnnn]:.... Insufficient number of VNCs: X, required: Y
```

Error Code: 9

The NeuronCores may be in use by models you are not actively using. Ensure you’ve unloaded models you’re not using and deleted unused NCGroups. If this is still a problem, moving to a larger Inf1 instance size with additional NeuronCores may help.
Inferences are failing

Wrong Model Id

An inference request had a model id that is invalid or not in a running state.
Check for error messages in syslog similar to:

```
nrtd[nnnn]:....Failed to find model: 10001
```

Ensure your application is only inferring against models that are running on the Inferentia.

Bad or incorrect inputs

NEFF contains information of the number of input feature maps required by the model. If inputs to the model don’t match the expected number/size of the input, inference will fail.
Mismatch in either the number of expected inputs or the size of the inputs.
Check for error messages in syslog similar to:

```
nrtd[nnnn]: .... Wrong number of ifmaps, expected: X, received: Y
nrtd[nnnn]: .... Invalid data length for [input:0], received X, expected Y
```

Ensure the correct number of inputs and correct sizes are used.

Numerical errors on the Model

The inference generated NaNs during execution, which is usually an indication of model or input errors.
Check for error messages in syslog similar to:

```
nrtd[nnnn]: .... Error notifications found on NC .... INFER_ERROR_SUBTYPE_NUMERICAL
```

Report issue to Neuron by posting the relevant details on GitHub issues.

Inference Timeout

It’s possible that the Neuron Compiler built a NEFF with errors, e.g. the NEFF might describe incorrect internal data flows or contain incorrect instruction streams. When this happens, it will potentially result in a timeout during inference.
Check for error messages in syslog similar to:

```
nrtd[nnnn]: .... Error: DMA completion timeout in ....
```

Error Code: 5
Report issue to Neuron by posting the relevant details on GitHub issues.
Neuron runtime FAQ

- How does Neuron connect to all the Inferentia chips in an Inf1 instance?
- Where can I get logging and other telemetry information?
- What about RedHat or other versions of Linux?
- What about Windows?
- How can I use Neuron in a container based environment? Does Neuron work with ECS and EKS?
- How can I take advantage of multiple NeuronCores to run multiple inferences in parallel?

How does Neuron connect to all the Inferentia chips in an Inf1 instance?

By default, a single runtime process will manage all assigned Inferentias, including running the Neuron Core Pipeline mode. If needed, you can configure multiple KRT processes each managing a separate group of Inferentia chips. For more details please refer to nrt-overview.

Where can I get logging and other telemetry information?

See this document on how to collect logs: Using Neuron GatherInfo Tool to collect debug and support information.

What about RedHat or other versions of Linux?

We don't officially support it yet.

What about Windows?

Windows is not supported at this time.

How can I use Neuron in a container based environment? Does Neuron work with ECS and EKS?

ECS and EKS support is coming soon. Containers can be configured as shown here.

How can I take advantage of multiple NeuronCores to run multiple inferences in parallel?

Examples of this for TensorFlow are found here here.
Neuron Runtime 1.x Release Notes

This document lists the current release notes for AWS Neuron Runtime. Neuron Runtime software manages runtime aspects of executing inferences on Inferentia chips. Details on the configuration and use is available in Getting started: Installing and Configuring Neuron-RTD

Table of Contents

- NEFF Support Table:
  - [1.6.24.0]
  - [1.6.19.0]
  - [1.6.9.0]
  - [1.6.5.0]
  - [1.5.0.0]
  - [1.4.17.0]
  - [1.4.12.0]
  - [1.4.9.0]
  - [1.4.3.0]
  - [1.3.1.0]
  - [1.2.5.0]
  - [1.1.1402.0]
  - [1.0.9592.0]
  - [1.0.9197.0]
  - [1.0.8896.0]
  - [1.0.8813.0]
  - [1.0.8444.0]
  - [1.0.8032.0]
  - [1.0.7865.0]
  - [1.0.6905.0]
  - [1.0.6222.0]
  - [1.0.5795.0]
  - [1.0.5236.0]
  - [1.0.4751.0]
  - [1.0.4492.0]
  - [1.0.4109.0]
NEFF Support Table:

Use this table to determine the version of Runtime that will support the version of NEFF you are using. NEFF version is determined by the version of the Neuron Compiler.

<table>
<thead>
<tr>
<th>NEFF Version</th>
<th>Runtime Version Range</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.6</td>
<td>*</td>
<td>All versions of RT support NEFF 0.6</td>
</tr>
<tr>
<td>1.0</td>
<td>&gt;= 1.0.6905.0</td>
<td>Starting support for 1.0 NEFFs</td>
</tr>
<tr>
<td>2.0</td>
<td>&gt;= 1.6.5.0</td>
<td>Starting support for 2.0 NEFFs</td>
</tr>
</tbody>
</table>

[1.6.24.0]
Date: 08/30/2021

Improvements

- Minor updates.

[1.6.19.0]
Date: 08/12/2021

Improvements

- Minor updates.

[1.6.9.0]
Date: 07/26/2021

Improvements

- Bug fixes and minor enhancements.

[1.6.5.0]
Date: 07/02/2021

15.2. Neuron Runtime 1.x
Improvements

- Bug fixes and minor enhancements.

[1.5.0.0]
Date: 05/28/2021

Improvements

- Neuron driver now reserves memory to avoid memory allocation failure during bursty model loads.
- Fixed memory corruption issue where application might quit unexpectedly.
- General performance enhancements.

[1.4.17.0]
Date: 05/01/2021

Improvements

- Bug fixes and minor enhancements.

[1.4.12.0]
Date: 03/04/2021

Improvements

- Bug fixes and minor enhancements.

[1.4.9.0]
Date: 02/24/2021

Improvements

- Fix for CVE-2021-3177.
[1.4.3.0]

Date: 01/30/2021

Improvements

• Model load time has been improved by approximately 10% after changing runtime to avoid disk access.
• Improved return code when invalid/incomplete neffs are passed to runtime.

[1.3.1.0]

Date: 12/23/2020

Improvements

• Model load time has been improved. The model loading speed up could be up to 50% depending on the size of the model.

Resolved Issues

• Incorrect error code returned when a model fails to load due to the lack of resources.
• Restarting Neuron Runtime causes a memory leak in the Neuron kernel module.

[1.2.5.0]

Date: 11/17/2020

Major New Features

• Removed limitations on intermediate tensors in networks compiled for NeuronCore Pipeline. Previously, NeuronCores executing the pipeline could pass their outputs no further then to the NeuronCores on the same or the next Inferetia on an instance. This limitation is removed and a NeuronCore can now pass its outputs to any other NeuronCore in the NeuronCore Pipeline. This feature allows for deeper pipelines utilizing more NeuronCores that can result in better performance.

Resolved Issues

• Reloading Neuron Kernel Mode Driver causes memory leak
• Memory pool initialization can reference NULL pointer in case of a failure.
• A network fails to load on Inferetia with “Incorrect number of inputs” error. In some cases the Neuron Compiler could determine that a network input is a constant. The compiler then optimizes the input away to improve the performance. This action could create a mismatch between the inputs to the network submitted by a framework and the inputs expected by Inferentia causing errors during load.
[1.1.1402.0]

Date: 10/22/2020

**Major New Features**

This release introduces Neuron Kernel Mode Driver (KMD) as a new package `aws-neuron-dkms`. Neuron KMD removes the following requirements for Neuron Runtime:

- Passing of CAP_SYS_ADMIN to Neuron Runtime.
- User management of huge page system resources
- Execution of Neuron Runtime in a “sidecar” container.

This packages is required for regular operation of Neuron Runtime; hence it is marked as dependency for `aws-neuron-runtime-base` see Setup Guide for detailed installation steps.

**Resolved Issues**

- NEFF is container of files. When NEFF is generated on some host the content files permissions are inherited causing NEFF load failure in the inf1 instances. Fixed it by removing file permissions before loading it.

[1.0.9592.0]

Date: 09/22/2020

**Major New Features**

- n/a

**Improvements**

- The “handshake” API can be used between a framework, such as TensorFlow, and neuron-rtd. The API establishes a unique “session-id” (see the next item) and facilitates version exchange between a framework and neuron-rtd. Version information is used to improve logging and troubleshooting.
- The API for neural networks loading and for shared memory allocation have been enhanced to allow an optional “session id” to be passed in load/allocate requests. Session ids are used to associate a framework process with the networks and the shared memory segments used by the process. Neuron-rtd can optionally monitor framework processes and automatically unload all neural networks loaded by the process and free its shared memory when the process terminates.
Resolved Issues

- querying Neuron statistics could cause neuron-rtd to crash
- SRAM parity errors are not reported
- Under stress “queue full” error can be returned when submitting an inference request even when neuron-rtd has room for one more request

[1.0.9197.0]
Date: 08/19/2020
Summary
Bug fix only.
Major New Features
- n/a
Resolved Issues
- get-hw-counters API was returning ECC error counters for only one half of the Inferentia DRAM.

[1.0.8896.0]
Date: 08/08/2020
Summary
Bug fix only.
Major New Features
- n/a
Resolved Issues
- Fixed a crash in neuron-rtd when multiple clients attempt to load models at the same time.
Summary

Patching a bug from prior versions that could lead to crashes under load.

Major New Features

• n/a

Resolved Issues

• Fixed a race condition in the runtime that was leading to crashes in some cases of load testing.

[1.0.8444.0]

Date: 07/16/2020

Major New Features

• n/a

Improvements

• Improved performance of the Neural Networks with large input tensors.

Resolved Issues

• neuron-rtd crashes when “Unload All” API is called multiple times.

• In some cases neuron-compiler optimizes access to the input tensors. Because of this optimization inference requests fail with an error message indicating the mismatch between expected and supplied number of input tensors.

• In some cases NEFF can use more DMA rings than is supported by neuron-rtd. A Neural Network load fails to load with an error message indicating the failure to allocate a DMA ring.
Other Notes

- Renamed and combined Neuron device memory errors counters. Four counters - ddr0_ecc_corr, ddr0_ecc_uncorr, ddr1_ecc_corr, ddr1_ecc_uncorr were combined into two counters - mem_ecc_corr and mem_ecc_uncorr.

[1.0.8032.0]

Date: 6/18/2020

Major New Features

- n/a

Improvements

- n/a

Resolved Issues

- In the versions of aws-neuron-runtime-base and aws-neuron-runtime, yum downgrade/update removed the service unit files. This results in neuron-discovery and neuron-rtd start failures.

Please update the Neuron Runtime ingredients on AL2 by first removing the old package and installing the latest:

```
# Amazon Linux 2
sudo yum remove aws-neuron-runtime-base
sudo yum remove aws-neuron-runtime
sudo yum install aws-neuron-runtime-base
sudo yum install aws-neuron-runtime
```

[1.0.7865.0]

Date: 6/11/2020

Major New Features

- n/a
Improvements

- Improved Neuron device memory allocation to accommodate Neural Networks that operate on large tensors.
- Log the version of the NEFF file during Neural Network load to aid troubleshooting.

Resolved Issues

- An inference request with missing IFMAP tensors is allowed to execute and produces undefined results.
- neuron-rtd service is not stopped and is not removed when aws-neuron-runtime package is uninstalled.

Known Issues and Limitations

- A model might fail to load due to insufficient number of huge memory pages made available to Neuron-RTD.
  - Workaround: manually increase the amount of huge memory pages available to Neuron runtime by following the instructions here. (Requires a restart of the runtime daemon and a possible change to system-wide configs.)

[1.0.6905.0]

Date: 5/11/2020

Major New Features

- Support is added for NEFF 1.0.

Improvements

- A new API for unloading all loaded Neural Networks and for freeing all Inferentia resources. The API is used by ML frameworks in cases when an ML application needs to be restarted to bring Inferentias to their initial state.
- Improved inference error handling and improved verbosity of error notifications.
- Internal changes aimed to improve performance optimization work and debuggability.

Resolved Issues

- Latency of Neural Networks loading had degraded in 1.0.6222.0 release. The issue has been resolved.
Known Issues and Limitations

- A model might fail to load due to insufficient number of huge memory pages made available to Neuron-RTD.
  - Workaround: manually increase the amount of huge memory pages available to Neuron runtime by following the instructions here. (Requires a restart of the runtime daemon and a possible change to system-wide configs.)

[1.0.6222.0]

Date: 3/26/2020

Major New Features

N/A

Improvements

- Inferentia memory utilization has improved, allowing larger number of Neural Networks to be loaded simultaneously. The increased capacity could be up to 25% depending on the networks.
- Added an API to read performance counters for a single Neuron Core. Used internally by neuron-top, which comes with the aws-neuron-tools package.
- Added Neural Network caching. Caching of previously loaded Neural Networks in host memory can significantly speed up (up to 10x) the subsequent loading of the same networks, for example when using multiple Neuron Cores in data-parallel mode.

Resolved Issues

- Occassional neuron-rt service crashes when service was being shutdown.

Known Issues and Limitations

- A model might fail to load due to insufficient number of huge memory pages made available to Neuron-RTD.
  - Workaround: manually increase the amount of huge memory pages available to Neuron runtime by following the instructions here. (Requires a restart of the runtime daemon and a possible change to system-wide configs.)

[1.0.5795.0]

Date: 2/27/2020
Major New Features

• Added API to unload all models available via “neuron-cli reset”.

Improvements

• Neural Network Load and Neural Network Infer interfaces return descriptive error messages on failure.

• Throughput of Neural Networks running in NeuronCore Pipeline mode has improved by 10-50% (network dependent) by reducing contention among NeuronCores.

• Improved CPU utilization of neuron-rt daemon by completely removing one polling thread from neuron-rt.

Resolved Issues

• Neural Networks containing CPU partitions only do not load correctly.

• Insufficient logging makes it hard to identify Neural Network loading failure when multiple networks are loaded in parallel.

Known Issues and Limitations

• A model might fail to load due to insufficient number of huge memory pages made available to Neuron-RTD.
  
  – Workaround: manually increase the amount of huge memory pages available to Neuron runtime by following the instructions here. (Requires a restart of the runtime daemon and a possible change to system-wide configs.)

[1.0.5236.0]

Date: 1/27/2020

Major New Features

N/A

Improvements

• Improved neuron-rtd startup time on inf1.6xl and inf1.24xl. Neuron-rtd startup now takes the same amount of time on all instance sizes.

• Improved inference latency for Neural Networks that fully execute on Inferentia (have no on-CPU nodes). The exact latency improvement is network dependent and is estimated to be 50-100us per inference.

• Neural Network load GRPC returns descriptive error message when the load fails.

• Changed default behavior of neuron-rtd to drop elevated privileges after runtime initialization. During initialization elevated privileges are necessary to allow bus enumeration and shared memory with frameworks.

• Error log is automatically displayed on the console if the installation of aws-neuron-runtime fails.
Resolved Issues

- minor bug fixes

Known Issues and Limitations

- A model might fail to load due to insufficient number of huge memory pages made available to Neuron-RTD. A manual reconfiguration and Neuron-RTD restart is required for increasing the amount of huge memory pages available to Neuron-RTD.
  - Workaround: manually increase the amount of huge memory pages available to Neuron runtime by following the instructions here. (Requires a restart of the runtime daemon and a possible change to system-wide configs.)

- Neuron-RTD does not return verbose error messages when an inference fails. Detailed error messages are only available in syslog.
  - Workaround: manually search syslog file for Neuron-RTD error messages.

[1.0.4751.0]

Date: 12/20/2019

Major New Features

N/A

Improvements

- Improved neuron-rtd startup time on inf1.24xl
- Reduced inference submission overhead (improved inference latency)
- Made the names and the UUIDs of loaded models available to neuron-tools

Resolved Issues

The following issues have been resolved:

- File I/O errors are not checked during model load
- Memory leak during model unload
- Superfluous error message are logged while reading neuron-rtd configuration file
- neuron-rtd –version command does not work
Known Issues and Limitations

- A model might fail to load due to insufficient number of huge memory pages made available to Neuron-RTD. A manual reconfiguration and Neuron-RTD restart is required for increasing the amount of huge memory pages available to Neuron-RTD.
  - Workaround: manually increase the amount of huge memory pages available to Neuron runtime by following the instructions here: (Requires a restart of the runtime daemon and a possible change to system-wide configs.)
- Neuron-RTD does not return verbose error messages when a model load or an inference fails. Detailed error messages are only available in syslog.
  - Workaround: manually search syslog file for Neuron-RTD error messages.

Other Notes

[1.0.4492.0]

Date: 12/1/2019

Major New Features

N/A

Resolved Issues

The following issues have been resolved:

- Neuron-RTD fails to initialize all NeuronCores on Inf1.24x1 Inferentia instances
- On some instances neuron-discovery requires packages (pciutils)
- An inference request might timeout or return a failure when a NeuronCore Pipeline model is loaded on any instance larger than Inf1.xl or Inf1.2xla
- Loading of a model fails when NeuronCore Pipeline inputs are consumed by NeuronCores beyond the first 4 NeuronCores used by the model
- Neuron-RTD logging to stdout does not work
- Incorrect DMA descriptors validation. While loading a model; descriptors are allowed to point beyond allocated address ranges. This could cause the model load failure or produce incorrect numerical results
- NeuronCore statistics are read incorrectly
Known Issues and Limitations

- A model might fail to load due to insufficient number of huge memory pages made available to Neuron-RTD. A manual reconfiguration and Neuron-RTD restart is required for increasing the amount of huge memory pages available to Neuron-RTD.
  - Workaround: manually increase the amount of huge memory pages available to Neuron runtime by following the instructions here: (Requires a restart of the runtime daemon and a possible change to system-wide configs.)
- Neuron-RTD does not return verbose error messages when a model load or an inference fails. Detailed error messages are only available in syslog.
  - Workaround: manually search syslog file for Neuron-RTD error messages.
- Neuron-RTD takes 6 minutes to start on Inf1.24xl instance.

Other Notes

[1.0.4109.0]

Date: 11/25/2019

Summary

This document lists the current release notes for AWS Neuron runtime. Neuron runtime software manages runtime aspects of executing inferences on Inferentia chips. It runs on Ubuntu 16, Ubuntu 18 and Amazon Linux 2.

Major new features

N/A, this is the first release.

Major Resolved issues

N/A, this is the first release.

Known issues and limitations

- Neuron-RTD fails to initialize all NeuronCores on Inf1.24xl Inferentia instances.
  - Workarounds: update to next release
- On some instances neuron-discovery requires packages (pciutils)
  - Workaround: install explicitly
- An inference request might timeout or return a failure when a NeuronCore Pipeline model is loaded on any instance larger than Inf1.xl or Inf1.2xla
  - Workarounds: update to the next release
• Loading of a model fails when NeuronCore Pipeline inputs are consumed by NeuronCores beyond the first 4 NeuronCores used by the model. A model can be compiled to run on multiple NeuronCores spread across multiple Inferentias. The model’s inference inputs (ifmaps) can be consumed by one or more NeuronCores, depending on a model. If a model requires inputs going to NeuronCores beyond the first 4 the loading of the model will fail.
  – Workarounds: update to the next release
• Neuron-RTD logging to stdout does not work
  – Workarounds: update to the next release
• Incorrect DMA descriptors validation. While loading a model; descriptors are allowed to point beyond allocated address ranges. This could cause the model load failure or produce incorrect numerical results.
  – Workarounds: update to the next release
• NeuronCore statistics are read incorrectly
  – Workarounds: update to the next release
• A model might fail to load due to insufficient number of huge memory pages made available to Neuron-RTD. A manual reconfiguration and Neuron-RTD restart is required for increasing the amount of huge memory pages available to Neuron-RTD.
  – Workarounds: manually increase the amount of huge memory pages available to Neuron runtime by following the instructions here: ** This requires a restart of the runtime daemon.
• Neuron-RTD does not return verbose error messages when a model load or an inference fails. Detailed error messages are only available in syslog.
  – Workarounds: manually search syslog file for Neuron-RTD error messages.

Other Notes

• DLAMI v26.0 users are encouraged to update to the latest Neuron release by following these instructions: https://github.com/aws/aws-neuron-sdk/blob/master/release-notes/dlami-release-notes.md
Neuron includes a set of tools and capabilities to help developers monitor and optimize their Neuron based inference applications. Neuron tools can be incorporated into scripts to automate Neuron devices operation and health monitoring, and include discover and usage utilities, data-path profiling tools, and visualization utilities. Using a TensorBoard plugin you can inspect and profile graphs execution.

16.1 Neuron Tools 2.x

16.1.1 Neuron Plugin for TensorBoard

Table of Contents

- Overview
- Setup Environment
- Compile the neural network
- Enable profiling
- Launch TensorBoard
- View results in TensorBoard
- Visualize graphs executed on Neuron
  - Show how the graph was partition to run on NeuronCores
  - Inspect which operators consumes the most time
  - Check out Neuron support operators for each framework
  - Filter view by device
  - Expand/collapse subgraphs and view operator details
- Viewing the Neuron profile data
  - See performance summary
  - Get a breakdown of time spent per NeuronCore
  - Get a breakdown of time spent per operator
Overview

This guide is for developers who want to better understand how their model is executed using Neuron SDK through TensorBoard.

TensorBoard is an open source visualization tool for machine learning projects. It allows for custom plugins, allowing for new ways to visualize information within TensorBoard.

The Neuron plugin for TensorBoard is focused on helping users better understand the performance of their machine learning workload using Neuron SDK. It is compatible with TensorBoard versions 1.15 and higher, and supported for Neuron tools version 1.5 and higher. Neuron tools version 1.5 is introduced in Neuron v1.13.0 release.

Neuron Plugin for TensorBoard provides visualizations and profiling results for graphs executed on NeuronCores.

Note: Graph visualization is currently only supported for TensorFlow-Neuron. Support for MXNet-Neuron and PyTorch-Neuron visualization will be added in a future release.

Setup Environment

- pytorch-quickstart
- tensorflow-quickstart
- mxnet-quickstart

Compile the neural network

3. Refer to the following guides on how to compile a graph using Neuron SDK.
   - TensorFlow-Neuron
     - Running ResNet50 on Inferentia
   - PyTorch-Neuron:
     - “Compile model for Neuron” in PyTorch-Neuron Resnet50 Tutorial
   - MXNet-Neuron:
     - Running Neuron Apache MXNet (Incubating) ResNet50 on Inferentia

Enable profiling

In this step, we enable Neuron profile data collection and collect results from executing an inference.

4.1. To start profiling the neural network and collect inference traces, create a directory where profile data will be dumped and set the NEURON_PROFILE environment variable. In this example, we will assume this directory is $HOME/profile

```
mkdir -p $HOME/profile
export NEURON_PROFILE=$HOME/profile
```

4.2. Ensure Neuron Tools are executable by setting the PATH environment variable.

```
export PATH=/opt/aws/neuron/bin:$PATH
```
4.3. Execute inference!

**Note:** Please run the inference script outside of Jupyter notebook. Profiling in Jupyter notebook is not supported at this time.

**Note:** Please ensure the inference script executes only one inference, as profiling results are currently only supported for a single inference.

For more info on how to execute inference, refer to the following guides:

- **TensorFlow-Neuron**
  - *Running ResNet50 on Inferentia*
- **PyTorch-Neuron**
  - “Run inference on Single Core” in *ResNet50 model for Inferentia*
- **MXNet-Neuron**
  - *Running Neuron Apache MXNet (Incubating) ResNet50 on Inferentia*

4.4. Check if profiling results were successfully saved. In the directory pointed to by `NEURON_PROFILE` environment variable set in Step 4.1, there should be at least two files, one with the `.neff` extension and one with the `.ntff` extension. For TensorFlow-Neuron users, the graph file (`.pb`) will also be in this directory.

```
ls $NEURON_PROFILE
```

**Launch TensorBoard**

In this step, we will process the Neuron profile data and launch TensorBoard.

5.1. Install the Neuron plugin for Tensorboard.

If you are using the DLAMI TensorFlow-Neuron Conda environment, please run the following to update TensorBoard before installing the Neuron plugin.

```
pip install "tensorboard<=2.4.0" --force-reinstall
```

Modify Pip repository configurations to point to the Neuron repository:

```
tee $VIRTUAL_ENV/pip.conf > /dev/null <<EOF
[global]
extra-index-url = https://pip.repos.neuron.amazonaws.com
EOF

pip install tensorboard-plugin-neuron
```

5.2. After collecting the raw profile data, we need to post-process it to create the log files used by the Neuron plugin. This can be done when launching TensorBoard by passing an extra flag `--run_neuron_profiler`. Using this flag will create the directory specified by `--logdir` and populate it with Neuron plugin data. Please note that the `NEURON_PROFILE` environment variable set in Step 4.1 must still point to the same directory as before.

```
tensorboard --logdir results --run_neuron_profiler
```
Note: If using TensorBoard >= 2.5, please use the --load_fast=false option when launching. tensorboard --logdir results --run_neuron_profiler --load_fast=false

5.3. After you see the following message, TensorBoard is ready to use. By default, TensorBoard will be launched at localhost:6006 on the Deployment Instance.

... Running neuron-profile
Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind_all
TensorBoard 2.4.1 at http://localhost:6006/ (Press CTRL+C to quit)

View results in TensorBoard

In this step, we will view the Neuron plugin for TensorBoard from a browser on your local development machine.

6.1. Connect to the Deployment Instance while enabling port forwarding. In this example, we assume TensorBoard has been launched using the default address localhost:6006 on the Deployment Instance.

```bash
# if Ubuntu-based AMI
ssh -i <PEM key file> ubuntu@<instance DNS> -L 6006:localhost:6006

# if AL2-based AMI
ssh -i <PEM key file> ec2-user@<instance DNS> -L 6006:localhost:6006
```

6.2. In a browser, visit .

6.3. In the top navigation bar, switch from Graphs to Neuron. If it does not show up, please wait a while and refresh the page while the plugin loads. If the issue persists, check the Inactive dropdown list on the right and check for Neuron.

6.4. If TensorBoard failed to find the generated logs, you will see the following message:
In this case, please check the console output on the Deployment Instance where TensorBoard was launched for any warnings or error messages, and make sure the version of the `aws-neuron-tools` package is compatible.

**Visualize graphs executed on Neuron**

**Show how the graph was partition to run on NeuronCores**

To view how the graph was partitioned to run on NeuronCores, select “Device” under “Graph Color Schemes” in the left navigation bar.

Each operator will be colored according to the device used. In this example, light blue indicates an operator was executed on CPU, and orange indicates the operator was executed on NeuronCores. Operators that are white may have been optimized by the Neuron compiler and fused into another operation.
Inspect which operators consumes the most time

You can also view how long each operator took by changing to the “Compute time” color scheme.

This view will show time taken by each layer and will be colored according to how much relative time the layer took to compute. A lighter shade of red means that a relatively small portion of compute time was spent in this layer, while a darker red shows that more compute time was used.

Check out Neuron support operators for each framework

The “Compatibility” color scheme allows you to better understand what operators are currently supported by the Neuron compiler - green for compatible ops, red for incompatible ops, and yellow for subgraphs that contain both compatible and incompatible ops.
Filter view by device

Additionally, you can choose to filter by CPU and NeuronCores, which will only color ops that match the selected device(s).

Expand/collapse subgraphs and view operator details

Each rectangular node in the graph represents a subgraph that can be expanded or collapse by clicking on the name. Operators will be represented by ellipses, and can be clicked to reveal more information on that operator, such as inputs and execution device.

The Expand All and Collapse All buttons can be used to expand or collapse every subgraph. When using these features, the positioning of the graph may change when redrawing the new graph. Try using Reset Position button and zoom out by scrolling if the graph appears to be missing.
Viewing the Neuron profile data

On the right side of the Neuron plugin, information on the profiled inference will be displayed.

See performance summary

First is the “Neuron Performance Summary,” which gives a quick overview on how Neuron executed the graph, including information on the number of NeuronCores and both on-NeuronCore time and on-CPU time.

Get a breakdown of time spent per NeuronCore

Next, the “Neuron Execution” will give more details on how a graph was partitioned for Neuron. Each entry in the table will show the order it was executed in, what type of device was used, the compute time (in microseconds), and the percentage of total time spent. To dive deeper into subgraphs, you can check the “Show Details” box to display the breakdown per NeuronCore.
Get a breakdown of time spent per operator

The “Op Time Table” section shows the cycle count per operator, much like the “Compute time” coloring for graph visualization. This table can be sorted by clicking the column names, and searched using the provided text box in the top right corner. Due to Neuron compiler optimizations, some of the compute may not be associated with any specific operator and will be categorized as unknown. Additionally, time spent moving data to and from NeuronCores will fall under (ND_ENGINE_LOAD).
16.1.2 Neuron Monitor User Guide

- **Overview**
- **Using neuron-monitor**
  - Configuration file example
  - Neuron applications tagging
  - JSON objects and fields in the configuration file
  - Neuron Runtime-level metric groups
  - System-wide metric groups
- **Execution model**
- **The JSON output format**
- **Neuron application level metric groups**
  - neuroncore_counters
  - inference_stats
  - memory_used
  - neuron_runtime_vcpu_usage
- **System level metric groups**
  - neuron_hw_counters
  - vcpu_usage
  - memory_info
- **Companion scripts**
  - neuron-monitor-cloudwatch.py
  - neuron-monitor-prometheus.py

**Overview**

**neuron-monitor** collects metrics and stats from the Neuron Applications running on the system and streams the collected data to `stdout` in JSON format. It is provided as part of the `aws-neuron-tools` package.

These metrics and stats are organized into **metric groups** which can be configured by providing a configuration file as described in **Using neuron-monitor**.

When running, **neuron-monitor** will:

- Collect the data for the metric groups which, based on the elapsed time since their last update, need to be updated
- Take the newly collected data and consolidate it into a large report
- Serialize that report to JSON and stream it to `stdout` from where it can be consumed by other tools - such as the sample `neuron-monitor-cloudwatch.py` and `neuron-monitor-prometheus.py` scripts.
- Wait until at least one metric group needs to be collected and repeat this flow
Using neuron-monitor

neuron-monitor CLI

neuron-monitor [parameters]
neuron-monitor accepts the following optional parameters:

- **--verbose** (int) default=0: Can be 0 to 4, and controls the amount of debugging and verbose information sent to stderr; 0: no output, 4: maximum verbosity
- **-c, --config-file** (string): Allows specifying a valid path to a neuron-monitor JSON configuration file

Example:

```
neuron-monitor -c monitor.conf
```

Not specifying any configuration file will enable collecting all the metric groups with a period of 5 seconds for all currently running Neuron applications.

Configuration file example

Example of a configuration file which enables all available **metric groups** for every running Neuron application, with a global update period of 1 second and sets an update period of 2 seconds for the "neuron_hw_counters" metric group:

```
{
  "period": "1s",
  "neuron_runtimes": [
    {
      "tag_filter": ".*",
      "metrics": [
        {
          "type": "neuroncore_counters"
        },
        {
          "type": "memory_used"
        },
        {
          "type": "neuron_runtime_vcpu_usage"
        },
        {
          "type": "inference_stats"
        }
      ]
    }
  ],
  "system_metrics": [
    {
      "type": "vcpu_usage"
    },
    {
      "type": "memory_info"
    },
    {
      "period": "2s",
      "type": "neuron_hw_counters"
    }
  ]
}
```

(continues on next page)
Neuron applications tagging

In order to make application monitoring easier, Neuron applications can be tagged with a 32 character string which identifies that app. Tagging is done using the `NEURON_PROCESS_TAG` environment variable.

For example: `NEURON_PROCESS_TAG=my_app_1 python run_inferences.py` will associate the `my_app_1` tag with that Python application. If `NEURON_PROCESS_TAG` is not specified, the application’s PID will be used as a TAG.

This tag will be used by neuron-monitor to filter Neuron applications.

JSON objects and fields in the configuration file

- "neuron_runtimes" - array of objects specifying which Neuron Applications to monitor and what metric groups are enabled for each of them
  - "tag_filter" - a regex which will be used to filter Neuron applications tags in order to determine if they will be monitored (optional)
  - "metrics" - array of objects specifying which metric groups to capture for this Neuron application
    - "type" - type of metric group
- "period" - this field applies to metric group objects and sets the amount of time between two updates for that metric group
  - if can be specified as part of the root and/or neuron_runtime objects where it applies to all their children, and/or as part of a metric group object
  - if there’s no period specified, a default value of 5 seconds will be used
- "system_metrics" - array of objects specifying which system level metric groups are enabled

Neuron Runtime-level metric groups

- `neuroncore_counters` - NeuronCore related metrics
- `memory_used` - data on the amount of memory used by the Neuron application
- `vcpu_usage` - Neuron application vCPU utilization data
- `inference_stats` - Neuron application inference stats, including error count and latency
System-wide metric groups

- `vcpu_usage` - system-wide vCPU usage
- `memory_info` - system-wide memory usage
- `neuron_hw_counters` - counters for correctable and uncorrectable memory ecc events

Execution model

neuron-monitor waits for one or more metric groups to be up for update, then collects the corresponding data, consolidates it into a report which is streamed to stdout as a JSON and goes back to waiting.

The JSON output format

Whenever the report gets updated, a complete JSON is written to stdout. This is its structure:

```json
{
    "neuron_runtime_data": [
        {
            "pid": 0,
            "address": "",
            "neuron_runtime_tag": "my_app_1",
            "error": "",
            "report": {
                "neuroncore_counters": {
                    "...
                },
                "inference_stats": {
                    "...
                }
            }
        }
    ]
}
```

(continues on next page)
• "neuron_runtime_data" is an array containing one entry per each Neuron application which passes the filter specified in the settings file
  - "pid" is the pid of this Neuron application
  - "neuron_runtime_tag" is the configured tag for the Neuron application
  - "error" specifies any error that occurred when collecting data from this Neuron application
  - "report" will contain the results for the Neuron application-level metric groups; their formats are described below

• "system_data" has a similar structure to "neuron_runtime_data"'s "report" but only contains system-level metric groups (not associated to any Neuron application)

Regardless of the configuration, the following two JSON objects are always present in the output:

**instance_data** Contains information about the instance on which neuron-monitor is running.

```
"instance_data": {
  "instance_name": "My_Instance",
  "instance_id": "i-0011223344556677a",
  "instance_type": "inf1.xlarge",
  "instance_availability_zone": "us-west-2b",
  "instance_availability_zone_id": "usw2-az2",
  "instance_region": "us-west-2",
  "ami_id": "ami-0011223344556677b",
  "subnet_id": "subnet-112233ee",
  "error": ""
}
```

Depending on when the instance was launched, the following fields might not be available:

• instance_availability_zone_id: available only for instances launched in 2020-08-24 and later
instance_region: available only for instances launched on 2020-08-24 and later

instance_name: available only if instance_region is set and aws-cli tools are installed
e error will contain an error string if getting one of the fields, except those mentioned above, resulted in an error.

neuron_hardware_info Contains basic information about the Neuron hardware.

```
"neuron_hardware_info": {
  "neuron_device_count": 16,
  "neuroncore_per_device_count": 4,
  "error": ""
}
```

neuron_device_count: number of available Neuron Devices

neuroncore_per_device_count: number of NeuronCores present on each Neuron Device

e error: will contain an error string if any occurred when getting this information (usually due to the Neuron Driver not being installed or not running).

Each metric group requested in the settings file will get an entry in the resulting output. The general format for such an entry is:

```
"metric_group": {
  "period": 1.015, // Actual captured period, in seconds
  "error": "", // Error, if any occurred, otherwise an empty string
  [...] // Metric group specific data
}
```

Neuron application level metric groups

neuroncore_counters

```
"neuroncore_counters": {
  "period": 1.000113182,
  "neuroncores_in_use": {
    "0": {
      "neuroncore_utilization": 42.01,
    },
    "1": {
      "neuroncore_utilization": 42.02,
    },
    "2": {
      "neuroncore_utilization": 42.03,
    },
    "3": {
      "neuroncore_utilization": 42.04,
    },
  },
  "error": ""
}
```

- "neuroncores_in_use" is an object containing data for all the NeuronCores that were active when the data was captured, indexed by NeuronCore index: "neuroncore_index": { neuroncore_data }
  - "neuroncore_utilization" - NeuronCore utilization, in percent, during the captured period
- "error" - string containing any error that occurred when collecting the data
inference_stats

```
"inference_stats": {
    "period": 1.030613214,
    "error_summary": {
        "generic": 0,
        "numerical": 0,
        "transient": 0,
        "model": 0,
        "runtime": 0,
        "hardware": 0
    },
    "inference_summary": {
        "completed": 123,
        "completed_with_err": 0,
        "completed_with_num_err": 0,
        "timed_out": 0,
        "incorrect_input": 0,
        "failed_to_queue": 0
    },
    "latency_stats": {
        "total_latency": {
            "p0": 0.01100001,
            "p1": 0.01100002,
            "p25": 0.01100004,
            "p50": 0.01100008,
            "p75": 0.01100010,
            "p99": 0.01100012,
            "p100": 0.01100013
        },
        "device_latency": {
            "p0": 0.01000001,
            "p1": 0.01000002,
            "p25": 0.01000004,
            "p50": 0.01000008,
            "p75": 0.01000010,
            "p99": 0.01000012,
            "p100": 0.01000013
        }
    },
    "error": ""
},
```

- "error_summary" is an object containing the error counts for the captured period indexed by their type
  - "generic" - generic inference errors
  - "numeric" - NAN inference errors
  - "transient" - recoverable errors, such as ECC corrections
  - "model" - model-related errors
  - "runtime" - Neuron Runtime / Library errors
  - "hardware" - hardware errors such as uncorrectable ECC issues

- "inference_summary" is an object containing all inference outcome counts for the captured period indexed by their type
  - "completed" - inferences completed successfully
- "completed_with_err" - inferences that ended in an error other than numeric
- "completed_with_num_err" - inferences that ended in a numeric error
- "timed_out" - inferences that took longer than the Neuron Runtime configured timeout value
- "incorrect_input" - inferences that failed to start due to incorrect input being provided
- "failed_to_queue" - inference requests that were rejected due to Neuron Runtime not being able to queue them

- "latency_stats" contains two objects containing latency percentiles, in seconds, for the data captured for inferences executed during the captured period. If there are no inferences being executed during this time, the two objects will be null (i.e. "total_latency": null)
  - "total_latency" - percentiles, in seconds, representing latency for an inference as measured by the Neuron Runtime
  - "device_latency" - percentiles, in seconds, representing time spent by an inference exclusively on the Neuron device
- "error" - string containing any error that occurred when collecting the data

memory_used

```
"memory_used": {
  "period": 1.030366715,
  "neuron_runtime_used_bytes": {
    "host": 1000000,
    "neuron_device": 2000000
  }
},
"loaded_models": [ {
  "name": "my_model",
  "uuid": "aaaaaaaaaaaaabbbbbbbbb0000000009999999999",
  "model_id": 10234,
  "is_running": true,
  "memory_used_bytes": {
    "host": 250000,
    "neuron_device": 500000
  }
},
"subgraphs": {
  "sg00": {
    "memory_used_bytes": {
      "host": 250000,
      "neuron_device": 500000
    },
    "neuroncore_index": 2,
    "neuron_device_index": 0
  }
}, ...
],
"error": ""
```

- "memory_used" summarizes the amount of memory used by the Neuron application
  - "neuron_runtime_used_bytes" - current amount of memory used by the Neuron application
all memory usage objects contain these two fields:

* "host" - host DRAM usage in bytes
* "neuron_device" - Neuron device DRAM usage in bytes

"loaded_models" - array containing objects representing loaded models

- "name" - name of the model
- "uuid" - unique id for the model
- "model_id" - Neuron application-assigned ID for this model
- "is_running" - true if this model is currently started, false otherwise
- "memory_used_bytes" - total memory usage for the model
- "subgraphs" - object containing all the subgraph for the model indexed by their name:
  "subgraph_name": { subgraph_data }
  * "memory_used_bytes" - memory usage for this subgraph
  * "neuroncore_index" - NeuronCore index with which the subgraph is associated
  * "neuron_device_index" - Neuron device index on which the subgraph is loaded

"error" - string containing any error that occurred when collecting the data

```json
"neuron_runtime_vcpu_usage": {
  "period": 1.030604818,
  "vcpu_usage": {
    "user": 42.01,
    "system": 12.34
  },
  "error": ""
}
```

"vcpu_usage" - object showing vCPU usage in percentages for the Neuron application during the captured period

- "user" - percentage of time spent in user code by this Neuron Application
- "system" - percentage of time spent in kernel code by this Neuron application

"error" - string containing any error that occurred when collecting the data

System level metric groups

```
"neuron_hw_counters": {
  "period": 1.030359284,
  "neuron_devices": [
    {
      "neuron_device_index": 0,
      "mem_ecc_corrected": 0,
      "mem_ecc_uncorrected": 0,
    }
  ]
}
```

(continues on next page)
• "neuron_devices" - array containing ECC data for all Neuron devices
  - "neuron_device_index" - Neuron device index
  - "mem_ecc_corrected" - number of corrected ECC events in the Neuron device’s DRAM
  - "mem_ecc_uncorrected" - number of uncorrected ECC events in the Neuron device’s DRAM
  - "sram_ecc_uncorrected" - number of uncorrected ECC events in the Neuron device’s SRAM

• "error" - string containing any error that occurred when collecting the data

vcpu_usage

```json
"vcpu_usage": {
  "period": 0.999974868,
  "average_usage": {
    "user": 32.77,
    "nice": 0,
    "system": 22.87,
    "idle": 39.36,
    "io_wait": 0,
    "irq": 0,
    "soft_irq": 0
  },
  "usage_data": {
    "0": {
      "user": 34.41,
      "nice": 0,
      "system": 27.96,
      "idle": 37.63,
      "io_wait": 0,
      "irq": 0,
      "soft_irq": 0
    },
    "1": {
      "user": 56.84,
      "nice": 0,
      "system": 28.42,
      "idle": 14.74,
      "io_wait": 0,
      "irq": 0,
      "soft_irq": 0
    },
    [...]
  },
  "context_switch_count": 123456,
  "error": ""
}
```

• each vCPU usage object contains the following fields:
- "user" - percentage of time spent in user code
- "nice" - percentage of time spent executing niced user code
- "system" - percentage of time spent executing kernel code
- "idle" - percentage of time spent idle
- "io_wait" - percentage of time spent waiting for IO operations
- "irq" - percentage of time spent servicing hardware interrupts
- "soft_irq" - percentage of time spent servicing software interrupts

- "average_usage" - contains the average usage across all vCPUs during the captured period
- "usage_data" - contains per vCPU usage during the captured period
- "context_switch_count" - contains the number of vCPU context switches during the captured period
- "error" - string containing any error that occurred when collecting the data

```
memory_info

"memory_info": {
    "period": 5.346411129,
    "memory_total_bytes": 49345835008,
    "memory_used_bytes": 16042344448,
    "swap_total_bytes": 0,
    "swap_used_bytes": 0,
    "error": "",
}
```

- "memory_total_bytes" - total size of the host memory, in bytes
- "memory_used_bytes" - amount of host memory in use, in bytes
- "swap_total_bytes" - total size of the host swap file, in bytes
- "swap_used_bytes" - amount of swap memory in use, in bytes

### Companion scripts

neuron-monitor is installed with two example Python companion script: `neuron-monitor-cloudwatch.py` and `neuron-monitor-prometheus.py`.

**neuron-monitor-cloudwatch.py**

It requires Python3 and the boto3 Python module. It is installed to: `/opt/aws/neuron/bin/neuron-monitor-cloudwatch.py`.
Using `neuron-monitor-cloudwatch.py`

```bash
euron-monitor | neuron-monitor-cloudwatch.py --namespace <namespace> --region <region>
euron-monitor | neuron-monitor-cloudwatch.py --namespace neuron_monitor_test --region us-west-2
```

For example:

```bash
euron-monitor | neuron-monitor-cloudwatch.py --namespace neuron_monitor_test --region us-west-2
```

**neuron-monitor-prometheus.py**

It requires Python3 and the [Prometheus](https://prometheus.io) client Python module. It is installed to: `/opt/aws/neuron/bin/` `neuron-monitor-prometheus.py`

**Using `neuron-monitor-prometheus.py`**

```bash
euron-monitor | neuron-monitor-prometheus.py --port <port>
euron-monitor | neuron-monitor-prometheus.py --port 8008
```

For example:

```bash
euron-monitor | neuron-monitor-prometheus.py --port 8008
```

The default value for `--port` is 8000.

If your data visualization framework is Grafana, we provided a [Grafana dashboard](https://grafana.com) which integrates with Prometheus and this script.

### 16.1.3 Neuron Top User Guide

- **Overview**
- **Using `neuron-top`**
  - Command line arguments
  - User interface

**Overview**

`neuron-top` provides useful information about NeuronCore and vCPU utilization, memory usage, loaded models, and Neuron applications.

**Note:** If you are parsing `neuron-top` output in your automation environment, you can now replace it with `neuron-monitor` ([Neuron Monitor User Guide](https://aws.amazon.com)) which outputs data in a standardized, easier to parse JSON format.
Using neuron-top

Command line arguments

Launching `neuron-top` with no arguments will show data for all ML Applications running with Neuron Runtime 2.x. `neuron-top` will also show data for any Neuron Runtime 1.x (`neuron-rtd` daemon) running on the default GRPC address - either `$NEURON_RTD_ADDRESS` or `unix:/run/neuron.sock` if that environment variable is not defined.

If more than one `neuron-rtd` daemons are running, each of their GRPC addresses needs to be specified in the command line:

```
neuron-top unix:/run/neuron2.sock unix:/run/neuron3.sock
```

User interface

The user interface is divided in 4 sections. The data shown in these sections only applies to the currently selected Neuron application:

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neuroncore Utilization</td>
<td>Shows NeuronCore utilization for the currently selected Neuron application.</td>
</tr>
<tr>
<td>VCPU and Memory Info</td>
<td>Shows total system vCPU usage, runtime vCPU usage, and runtime memory usage.</td>
</tr>
<tr>
<td>Error Info</td>
<td>Lists error details.</td>
</tr>
</tbody>
</table>

- The **Neuroncore Utilization** section shows the NeuronCore utilization for the currently selected Neuron application.
- The **VCPU and Memory Info** section shows:
  - Total system vCPU usage - the two percentages are user% and system%
  - Runtime vCPU usage - same breakdown
  - Runtime Memory Host - amount of host memory used by the Application and total available

```
```

1272 Chapter 16. Neuron Tools
Runtime Memory Device - amount of device memory used by the Application

- Loaded Models is a tree view which can be expanded/collapsed. The columns are:
  - Model ID - an Application-level identifier for this model instance
  - Host Memory - amount of host memory used by the model, displayed hierarchically, where the ‘parent’ value is the sum of its ‘children’
  - Device Memory - amount of device memory used by the model, displayed just like the Host Memory

**Note:** The up/down/left/right keys can be used to navigate the tree view. The ‘x’ key expands/collapses the entire tree.

The bottom bar shows which application is currently displayed by highlighting its tag using a yellow font and marking it using a pair of ‘>’, ‘<’ characters. For neuron-rtd daemons, the GRPC address will be shown here instead of the tag:

```
Model ID: neuron-process-1

Model ID: neuron-process-2
```

**Note:** The ‘1’-‘9’ keys select the current application. ‘a’/‘d’ selects the previous/next application on the bar.

### 16.1.4 Neuron LS User Guide

To identify number of Neuron Devices in a given instance use the `neuron-ls` command. `neuron-ls` will also show which processes are using each Device, including the command used to launch each of those processes.

If you are running Neuron Runtime 1.x neuron-rtd daemons, run the command as `sudo` to get their addresses and version numbers.

**neuron-ls CLI**

**neuron-ls [options]**

Available options:

- `--wide`, `-w`: Displays the table in a wider format.
- `--show-all-procs`, `-a`: Show all processes using the Neuron Devices, including processes that aren’t using Neuron Runtime 2.x such as `neuron-monitor` or `neuron-ls` itself.

**Note:** If you’re running Neuron Runtime 1.x neuron-rtd daemons, use `sudo` with the `neuron-ls` command to obtain their address and version information. Without `sudo`, the neuron-rtd processes will be considered generic processes using one or more Devices and will only be displayed when running neuron-ls with the `--show-all-procs` option.
### AWS Neuron Examples

```
$ neuron-ls
+--------+--------+--------+-----------+--------------+-------+-----------------------
| NEURON | NEURON | NEURON | CONNECTED | PCI | PID |
| COMMAND | RUNTIME | | | | |
| DEVICE | CORES | MEMORY | DEVICES | BDF | | |
| | | | | | | |
+--------+--------+--------+-----------+--------------+-------+-----------------------
| 0 | 4 | 8 GB | 1 | 0000:00:1c.0 | 23518 | neuron-app01 infer --
| | | | | | | | input-data-direct... 2.0.0 |
| 1 | 4 | 8 GB | 2, 0 | 0000:00:1d.0 | 23595 | neuron-app01 infer --
| | | | | | | | input-data-direct... 2.0.0 |
| 2 | 4 | 8 GB | 3, 1 | 0000:00:1e.0 | 23608 | neuron-app02 infer --
| | | | | | | | input-data-direct... 2.0.0 |
| 3 | 4 | 8 GB | 2 | 0000:00:1f.0 | NA | NA |
+--------+--------+--------+-----------+--------------+-------+-----------------------
```

```
$ neuron-ls --wide
+--------+--------+--------+-----------+--------------+-------+-----------------------
| NEURON | NEURON | NEURON | CONNECTED | PCI | PID |
| COMMAND | RUNTIME | | | | |
| DEVICE | CORES | MEMORY | DEVICES | BDF | | |
| | | | | | | |
+--------+--------+--------+-----------+--------------+-------+-----------------------
| 0 | 4 | 8 GB | 1 | 0000:00:1c.0 | 23518 | neuron-app01 infer --
| | | | | | | | input-data-directory ~/my_input_data --inference-count 5... 2.0.0 |
| 1 | 4 | 8 GB | 2, 0 | 0000:00:1d.0 | 23595 | neuron-app01 infer --
| | | | | | | | input-data-directory ~/my_input_data --inference-count 5... 2.0.0 |
| 2 | 4 | 8 GB | 3, 1 | 0000:00:1e.0 | 23608 | neuron-app02 infer --
| | | | | | | | input-data-directory ~/my_input_data --inference-count 5... 2.0.0 |
| 3 | 4 | 8 GB | 2 | 0000:00:1f.0 | NA | NA |
+--------+--------+--------+-----------+--------------+-------+-----------------------
```
$ neuron-ls --show-all-procs

<table>
<thead>
<tr>
<th>NEURON DEVICE</th>
<th>NEURON CORES</th>
<th>NEURON MEMORY</th>
<th>CONNECTED DEVICES</th>
<th>PCI BDF</th>
<th>PID</th>
</tr>
</thead>
<tbody>
<tr>
<td>neuron-app01</td>
<td>4</td>
<td>8 GB</td>
<td>1</td>
<td>0000:00:1c.0</td>
<td>23518</td>
</tr>
<tr>
<td>neuron-app02</td>
<td>4</td>
<td>8 GB</td>
<td>1</td>
<td>0000:00:1d.0</td>
<td>23595</td>
</tr>
<tr>
<td>neuron-app02</td>
<td>4</td>
<td>8 GB</td>
<td>1</td>
<td>0000:00:1e.0</td>
<td>23608</td>
</tr>
<tr>
<td>neuron-monitor</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>neuron-monitor</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>neuron-monitor</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

The above output is taken from an Inf1.6xlarge instance.

- **NEURON DEVICE**: Logical ID assigned to the Neuron Device.
- **NEURON CORES**: Number of NeuronCores present in the Neuron Device.
- **NEURON MEMORY**: Amount DRAM memory in Neuron Device.
- **CONNECTED DEVICES**: Logical ID of Neuron Devices connected to this Neuron Device.
- **PCI BDF**: PCI Bus Device Function (BDF) ID of the device.
- **PID**: ID of the process using this Neuron Device.
- **COMMAND**: Command used to launch the process using this Neuron Device.
- **RUNTIME VERSION**: Version of Neuron Runtime (if applicable) for the application using this Neuron Device.
16.1.5 What's New

Neuron Tools 2.x Release Notes

This document lists the release notes for AWS Neuron tools. Neuron tools are used for debugging, profiling and gathering inferentia system information.

Table of Contents

- Neuron Tools release [2.0.327.0]
- Neuron Tools release [2.0.277.0]

Neuron Tools release [2.0.327.0]

Date: 11/05/2021

- Updated Neuron Runtime (which is integrated within this package) to libnrt 2.2.18.0 to fix a container issue that was preventing the use of containers when /dev/neuron0 was not present. See details here Neuron Runtime 2.x Release Notes.

Neuron Tools release [2.0.277.0]

Date: 10/27/2021

New in this release

- Tools now support applications built with Neuron Runtime 2.x (libnrt.so).

Important:

- You must update to the latest Neuron Driver (aws-neuron-dkms version 2.1 or newer) for proper functionality of the new runtime library.

- Read Introducing Neuron Runtime 2.x (libnrt.so) application note that describes why we are making this change and how this change will affect the Neuron SDK in detail.

- Read Migrate your application to Neuron Runtime 2.x (libnrt.so) for detailed information of how to migrate your application.

- Updates have been made to neuron-1s and neuron-top to significantly improve the interface and utility of information provided.

- Expands neuron-monitor to include additional information when used to monitor latest Frameworks released with Neuron 1.16.0.

neuron_hardware_info Contains basic information about the Neuron hardware.

```json
"neuron_hardware_info": {
    "neuron_device_count": 16,
    "neuroncore_per_device_count": 4,
    (continues on next page)
```
- `neuron_device_count`: number of available Neuron Devices
- `neuroncore_per_device_count`: number of NeuronCores present on each Neuron Device
- `error`: will contain an error string if any occurred when getting this information (usually due to the Neuron Driver not being installed or not running).

- `neuron-cli` entering maintenance mode as it’s use is no longer relevant when using ML Frameworks with an integrated Neuron Runtime (libnrt.so). see Neuron support for Apache MXNet 1.5 enters maintenance mode for more information.

- For more information visit Neuron Tools

Neuron Plugin for TensorBoard Release Notes

Table of Contents

- **Known Issues and Limitations** - Updated 5/28/2021
- Neuron Plugin for TensorBoard release [2.2.0.0]
  - [2.1.2.0]
  - [2.1.0.0]
  - [2.0.29.0]
  - [1.15.0.1.2.6.0]
  - [1.15.0.1.1.1.0]
  - [1.15.0.1.0.615.0]
  - [1.15.0.1.0.600.0]
  - [1.15.0.1.0.570.0]
  - [1.15.0.1.0.513.0]
  - [1.15.0.1.0.491.0]
  - [1.15.0.1.0.466.0]
  - [1.15.0.1.0.392.0]
  - [1.15.0.1.0.366.0]
  - [1.15.0.1.0.315.0]
  - [1.15.0.1.0.306.0]
  - [1.15.0.1.0.280.0]
Known Issues and Limitations - Updated 5/28/2021

The following are not limitations in the Neuron plugin, but may affect your ability to use TensorBoard.

• In the DLAMI TensorFlow-Neuron Conda environment, attempting to run tensorboard with the existing version of tensorboard will fail. Please update the tensorboard version before installing the Neuron plugin by running `pip install tensorboard --force-reinstall`.

```
Traceback (most recent call last):
  File "/home/ubuntu/anaconda3/envs/aws_neuron_tensorflow_p36/bin/tensorboard", line 7, in <module>
      from tensorflow.tensorboard.tensorboard import main
ModuleNotFoundError: No module named 'tensorflow.tensorboard'
```

Neuron Plugin for TensorBoard release [2.2.0.0]

Date: 10/27/2021

New in this release

• Neuron Plugin for TensorBoard now support applications built with Neuron Runtime 2.x (libnrt.so).

_Important:_

– You must update to the latest Neuron Driver (aws-neuron-dkms version 2.1 or newer) for proper functionality of the new runtime library.

– Read _Introducing Neuron Runtime 2.x (libnrt.so)_ application note that describes _why are we making this change_ and _how this change will affect the Neuron SDK_ in detail.

– Read _Migrate your application to Neuron Runtime 2.x (libnrt.so)_ for detailed information of how to migrate your application.

[2.1.2.0]

Date: 8/12/2021

Summary

• Adds support for Neuron Tensorflow 2.5+
[2.1.0.0]

Date: 5/28/2021

Summary

• No major changes or fixes. Released with other Neuron packages.

[2.0.29.0]

Date: 4/30/2021

Summary

• First release Neuron plugin for TensorBoard. Check out it out here: Neuron Plugin for TensorBoard.
  – The Neuron plugin is now compatible with TensorBoard 2.0 and higher, in addition to TensorBoard 1.15
  – Provides a centralized place to better understand execution using Neuron SDK.
  – Continues support visualization for TensorFlow graphs, with support for PyTorch and MXNet coming in future releases.

• Neuron plugin for TensorBoard is supported for Neuron tools >= 1.5, which is first introduced in Neuron v1.13.0 release

• TensorBoard-Neuron is deprecated, and only supported for Neuron tools <= 1.4.12.0. The final version, 1.4.12.0 is part of Neuron v1.12.2 release.

[1.15.0.1.2.6.0]

Date: 2/24/2021

Summary

• Fix for CVE-2021-3177.

[1.15.0.1.1.0]

Date: 12/23/2020
Summary

- Minor internal improvements.

[1.15.0.1.0.615.0]

Date: 11/17/2020

Summary

- Fix issue with viewing chrome trace in Neuron profile plugin in Chrome 80+.

Resolved Issues

- Updated dependencies to polyfill missing APIs used by chrome trace in newer browser versions.

[1.15.0.1.0.600.0]

Date: 09/22/2020

Summary

- Minor internal improvements.

[1.15.0.1.0.570.0]

Date: 08/08/2020

Summary

- Minor internal improvements.

[1.15.0.1.0.513.0]

Date: 07/16/2020
Summary

- Minor internal improvements.

[1.15.0.1.0.491.0]

Date 6/11/2020

Summary

Fix issue where utilization was missing in the op-profile view.

Resolved Issues

- The op-profile view in the Neuron Profile plugin now correctly shows the overall NeuronCore utilization.

[1.15.0.1.0.466.0]

Date 5/11/2020

Summary

Fix potential installation issue when installing both tensorboard and tensorboard-neuron.

Resolved Issues

- Added tensorboard as a dependency in tensorboard-neuron. This prevents the issue of overwriting tensorboard-neuron features when tensorboard is installed after tensorboard-neuron.

Other Notes

[1.15.0.1.0.392.0]

Date 3/26/2020

Summary

Added ability to view CPU node latency in the Graphs plugin and the Neuron Profile plugins.
Major New Features

- Added an aggregate view in addition to the current Neuron subgraph view for both the Graphs plugin and the Neuron Profile plugin.
- When visualizing a graph executed on a Neuron device, CPU node latencies are available when coloring the graph by “Compute time” using the “neuron_profile” tag.
- The Neuron Profile plugin now has an overview page to compare time spent on Neuron device versus on CPU.

Other Notes

- Requires Neuron-RTD config option “enable_node_profiling” to be set to “true”

[1.15.0.1.0.366.0]

Date 02/27/2020

Summary

Reduced load times and fixed crashes when loading large models for visualization.

Resolved Issues

- Enable large attribute filtering by default
- Reduced load time for graphs with attributes larger than 1 KB
- Fixed a fail to load graphs with many large attributes totaling more than 1 GB in size

[1.15.0.1.0.315.0]

Date 12/20/2019

Summary

No major changes or fixes. Released with other Neuron packages.

[1.15.0.1.0.306.0]

Date 12/1/2019
Summary

Major New Features

Resolved Issues

Known Issues & Limits

Same as prior release

Other Notes

[1.15.0.1.0.280.0]

Date 11/29/2019

Summary

Initial release packaged with DLAMI.

Major New Features

N/A, initial release.


Resolved Issues

N/A - first release

Known Issues & Limits

- Must install TensorBoard-Neuron by itself, or after regular TensorBoard is installed. If regular Tensorboard is installed after TensorBoard-Neuron, it may overwrite some needed files.
- Utilization missing in Op Profile due to missing FLOPs calculation (see overview page instead)
- Neuron Profile plugin may not immediately show up on launch (try reloading the page)
- Graphs with NeuronOps may take a long time to load due to attribute size
- Instructions that cannot be matched to a framework layer/operator name show as “" (blank)
- CPU Usage section in chrome-trace is not applicable
- Debugger currently supports TensorFlow only
- Visualization requires a TensorFlow-compatible graph
Other Notes

16.2 Model Helper Tools

16.2.1 Neuron Check Model

Overview

Neuron Check Model tool provides user with basic information about the compiled and uncompiled model’s operations without the use of TensorBoard-Neuron. For additional visibility into the models, please see Neuron Plugin for TensorBoard.

Neuron Check Model tool scans the user’s uncompiled model and provides a table of the operations within the uncompiled model. By default, the table shows each operation type and number of instances of that type within model, and whether the type is supported in Neuron. If –show_names option is specified, the table shows each operation by name and whether the type of that operation is supported in Neuron.

If the model is already compiled, the tool also provides the table of operations as for uncompiled model. The table include the Neuron subgraph type and number of instances of that type, along with operations that have not been compiled to Neuron. Additionally, the tool displays a message showing the minimum number of NeuronCores required to run the model, followed by another table which shows the list of Neuron subgraphs by name and the number of pipelined NeuronCores used by each subgraph. More information about NeuronCore pipeline can be found in NeuronCore Pipeline. If –expand_subgraph option is specified, the operations within each subgraph are printed below the subgraph information.

Neuron Check Model tool is currently available for TensorFlow and MXNet. To check PT model, please use torch.neuron.analyze_model function as shown in PyTorch-Neuron Getting Started tutorial ResNet50 model for Inferentia

TensorFlow-Neuron Check Model

The following example shows how to run TensorFlow-Neuron Check Model tool with TensorFlow ResNet50 tutorial.

1. Start with the TensorFlow ResNet50 tutorial at Running ResNet50 on Inferentia and do the first three steps of the tutorial. Please stay in the Python environment that you setup during the tutorial.

2. Install needed tensorflow_hub package and download the tool:

```bash
pip install tensorflow_hub
python tf_neuron_check_model.py -h
```

usage: tf_neuron_check_model.py [-h] [-show_names] [-|--expand_subgraph] model_path

positional arguments:
  model_path : a TensorFlow SavedModel directory (currently supporting TensorFlow v1 SaveModel only).

optional arguments:
  -h, --help : show this help message and exit
  --show_names : list operation by name instead of summarizing by type
                 (caution: this option will generate many lines of output

(continues on next page)
3. After step 3 of the TensorFlow ResNet50 tutorial, you can check the uncompiled model to see Neuron supported operations (currently supporting TensorFlow v1 SaveModel only):

```
$ python tf_neuron_check_model.py ws_resnet50/resnet50/
```

* The following table shows the supported and unsupported operations within this uncompiled model.
* Each line shows an operation type, the number of instances of that type within model,
* and whether the type is supported in Neuron.
* Some operation types are excluded from table because they are no-operations or training-related operations:

```
 "VarHandleOp",
 "ShardedFilename", "SaveV2",
 "MergeV2Checkpoints", "RestoreV2"]
```

<table>
<thead>
<tr>
<th>Op Type</th>
<th>Num Instances</th>
<th>Neuron Supported ?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pad</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>RandomUniform</td>
<td>54</td>
<td>Yes</td>
</tr>
<tr>
<td>Sub</td>
<td>54</td>
<td>Yes</td>
</tr>
<tr>
<td>Mul</td>
<td>54</td>
<td>Yes</td>
</tr>
<tr>
<td>Add</td>
<td>54</td>
<td>Yes</td>
</tr>
<tr>
<td>Conv2D</td>
<td>53</td>
<td>Yes</td>
</tr>
<tr>
<td>BiasAdd</td>
<td>54</td>
<td>Yes</td>
</tr>
<tr>
<td>FusedBatchNormV3</td>
<td>53</td>
<td>Yes</td>
</tr>
<tr>
<td>Relu</td>
<td>49</td>
<td>Yes</td>
</tr>
<tr>
<td>MaxPool</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>AddV2</td>
<td>16</td>
<td>Yes</td>
</tr>
<tr>
<td>Fill</td>
<td>56</td>
<td>Yes</td>
</tr>
<tr>
<td>Mean</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>MatMul</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>Softmax</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>Pack</td>
<td>1</td>
<td>Yes</td>
</tr>
</tbody>
</table>

* Total inference operations: 504
* Total Neuron supported inference operations: 504
* Percent of total inference operations supported by Neuron: 100.0

4. You can also check the compiled model to see the number of pipeline NeuronCores for each subgraph:

```
$ python tf_neuron_check_model.py ws_resnet50/resnet50_neuron/
```

* Found 1 Neuron subgraph(s) (NeuronOp(s)) in this compiled model.
* Use this tool on the original uncompiled model to see Neuron supported operations.
* The following table shows all operations, including Neuron subgraphs.
* Each line shows an operation type, the number of instances of that type within model,
* and whether the type is supported in Neuron.
* Some operation types are excluded from table because they are no-operations or training-related operations:
AWS Neuron

(continued from previous page)


Op Type       Num Instances Neuron Supported ?
------ ------------- ------------------
NeuronOp 1 Yes

* Please run this model on Inf1 instance with at least 1 NeuronCore(s).
* The following list show each Neuron subgraph with number of pipelined NeuronCores used by subgraph
* (and subgraph operations if --expand_subgraph is used):

Subgraph Name Num
__Pipelined NeuronCores
------------ -------
-- ------------
conv5_block3_3_bn/FusedBatchNormV3/ReadVariableOp/neuron_op_d6f098c01c780733 1

5. When showing subgraph information, you can use --expand_subgraph to show operation types in each subgraph:

$ python tf_neuron_check_model.py ws_resnet50/resnet50_neuron/ --expand_subgraph

(output truncated to show subgraph information only)

Subgraph Name  Num
__Pipelined NeuronCores
------------ -------
-- ------------
conv5_block3_3_bn/FusedBatchNormV3/ReadVariableOp/neuron_op_d6f098c01c780733 1

Op Type       Num Instances
------ -------------
MatMul 1
Relu 49
Add 16
FusedBatchNorm 53
BiasAdd 54
Conv2D 53
Pad 2
Mean 1
MaxPool 1
Softmax 1

6. Use --show_names to see full operation names (caution: this option will generate many lines of output for a large model):

$ python tf_neuron_check_model.py ws_resnet50/resnet50_neuron/ --show_names

* Found 1 Neuron subgraph(s) (NeuronOp(s)) in this compiled model.
* Use this tool on the original uncompiled model to see Neuron supported operations.
* The following table shows all operations, including Neuron subgraphs.
* Each line shows an operation name and whether the type of that operation is supported in Neuron.
* Some operation types are excluded from table because they are no-operations or training-related operations:

(continues on next page)
### MXNet-Neuron Check Model

The following example shows how to run MXNet-Neuron Check Model tool with MXNet ResNet50 tutorial.

1. Start with the MXNet ResNet50 tutorial at [Running Neuron Apache MXNet (Incubating) ResNet50 on Inferentia](#) and do the first three steps of the tutorial. Please stay in the Python environment that you setup during the tutorial.

2. Download the tool:

   ```bash
   python mx_neuron_check_model.py -h
   ```

   **usage:** `mx_neuron_check_model.py [-h] [--show_names] [--expand_subgraph] model_path
   
   **positional arguments:**
   
   `model_path` path prefix to MXNet model (the part before -symbol.json)

   **optional arguments:**
   
   `-h, --help` show this help message and exit
   
   `--show_names` list operation by name instead of summarizing by type
   
   (caution: this option will generate many lines of output for a large model).
   
   `--expand_subgraph` show subgraph operations.

3. After step 3 of MXNet ResNet50 tutorial, you can check the uncompiled model to see Neuron supported operations:

   ```bash
   $ python mx_neuron_check_model.py resnet-50
   ```

---

### 16.2. Model Helper Tools

---
The following table shows the supported and unsupported operations within this uncompiled model.
* Each line shows an operation type, the number of instances of that type within model,
* and whether the type is supported in Neuron.
* Some operation types are excluded from table because they are no-operations or training-related operations:
['null']

<table>
<thead>
<tr>
<th>Op Type</th>
<th>Num Instances</th>
<th>Neuron Supported?</th>
</tr>
</thead>
<tbody>
<tr>
<td>BatchNorm</td>
<td>51</td>
<td>Yes</td>
</tr>
<tr>
<td>Convolution</td>
<td>53</td>
<td>Yes</td>
</tr>
<tr>
<td>Activation</td>
<td>50</td>
<td>Yes</td>
</tr>
<tr>
<td>Pooling</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>elemwise_add</td>
<td>16</td>
<td>Yes</td>
</tr>
<tr>
<td>Flatten</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>FullyConnected</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>SoftmaxOutput</td>
<td>1</td>
<td>No</td>
</tr>
</tbody>
</table>

Total inference operations: 175
Total Neuron supported inference operations: 174
Percent of total inference operations supported by Neuron: 99.4

4. You can also check the compiled model to see the number of pipeline NeuronCores for each subgraph:

```bash
$ python mx_neuron_check_model.py resnet-50_compiled
```

* Found 1 Neuron subgraph(s) (_neuron_subgraph_op(s)) in this compiled model.
* Use this tool on the original uncompiled model to see Neuron supported operations.
* The following table shows all operations, including Neuron subgraphs.
* Each line shows an operation type, the number of instances of that type within model,
* and whether the type is supported in Neuron.
* Some operation types are excluded from table because they are no-operations or training-related operations:
['null']

<table>
<thead>
<tr>
<th>Op Type</th>
<th>Num Instances</th>
<th>Neuron Supported?</th>
</tr>
</thead>
<tbody>
<tr>
<td>_neuron_subgraph_op</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>SoftmaxOutput</td>
<td>1</td>
<td>No</td>
</tr>
</tbody>
</table>

* Please run this model on Inf1 instance with at least 1 NeuronCore(s).
* The following list show each Neuron subgraph with number of pipelined NeuronCores used by subgraph
* (and subgraph operations if --expand_subgraph is used):

<table>
<thead>
<tr>
<th>Subgraph Name</th>
<th>Num Pipelined NeuronCores</th>
</tr>
</thead>
<tbody>
<tr>
<td>_neuron_subgraph_op0</td>
<td>1</td>
</tr>
</tbody>
</table>

5. When showing subgraph information, you can use --expand_subgraph to show operation types in each subgraph:

```bash
$ python mx_neuron_check_model.py resnet-50_compiled --expand_subgraph
```

(output truncated to show subgraph information only)
### Subgraph Name Num Pipelined NeuronCores

<table>
<thead>
<tr>
<th>Subgraph Name</th>
<th>Num Pipelined NeuronCores</th>
</tr>
</thead>
<tbody>
<tr>
<td>_neuron_subgraph_op0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Op Type</th>
<th>Num Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>BatchNorm</td>
<td>51</td>
</tr>
<tr>
<td>Convolution</td>
<td>53</td>
</tr>
<tr>
<td>Activation</td>
<td>50</td>
</tr>
<tr>
<td>Pooling</td>
<td>2</td>
</tr>
<tr>
<td>elemwise_add</td>
<td>16</td>
</tr>
<tr>
<td>Flatten</td>
<td>1</td>
</tr>
<tr>
<td>FullyConnected</td>
<td>1</td>
</tr>
</tbody>
</table>

6. Use `--show_names` to see full operation names (caution: this option will generate many lines of output for a large model):

```sh
$ python mx_neuron_check_model.py resnet-50_compiled --show_names
```

* Found 1 Neuron subgraph(s) (_neuron_subgraph_op(s)) in this compiled model.
* Use this tool on the original uncompiled model to see Neuron supported operations.
* The following table shows all operations, including Neuron subgraphs.
* Each line shows an operation name and whether the type of that operation is
  supported in Neuron.
* Some operation types are excluded from table because they are no-operations or
  training-related operations:

  ```
  ['null']
  ```

<table>
<thead>
<tr>
<th>Op Name</th>
<th>Op Type</th>
<th>Neuron Supported ?</th>
</tr>
</thead>
<tbody>
<tr>
<td>_neuron_subgraph_op0</td>
<td>_neuron_subgraph_op</td>
<td>Yes</td>
</tr>
<tr>
<td>softmax</td>
<td>SoftmaxOutput</td>
<td>No</td>
</tr>
</tbody>
</table>

* Please run this model on Inf1 instance with at least 1 NeuronCore(s).
* The following list show each Neuron subgraph with number of pipelined NeuronCores,
  used by subgraph
  (and subgraph operations if `--expand_subgraph` is used):

<table>
<thead>
<tr>
<th>Subgraph Name</th>
<th>Num Pipelined NeuronCores</th>
</tr>
</thead>
<tbody>
<tr>
<td>_neuron_subgraph_op0</td>
<td>1</td>
</tr>
</tbody>
</table>

### 16.2.2 Using Neuron GatherInfo Tool to collect debug and support information

#### Overview

The Neuron GatherInfo tool `neuron-gatherinfo.py` can assist in automating the collection and packaging of information from Neuron SDK tools that is useful to both user and AWS for issue resolution. The tool gathers log files and other system information. If being used to supply that info to AWS, the tool will redact proprietary and confidential information. The GatherInfo tool is supplied in source code form - available here: [Neuron Gatherinfo](neuron-gatherinfo/)

The tool enables developers to gather compiler and inference/runtime logs. Additionally, the common usage is from within one of the supported ML frameworks that have been integrated with Neuron, and information can be captured from those compile/runtime environments using the frameworks.
Steps Overview:

1. Obtain a copy of neuron-gatherinfo.py from Neuron Gatherinfo
2. Install into a location in your $PATH or into a location from where you can launch the script
3. Use with compile and/or runtime environments

Neuron-CC information gathering

Step 1: Re-run the compile steps for your workload with increased verbosity or debug levels

- For TensorFlow-Neuron, change the Python code as shown. Note that ‘compiler-workdir’ is expected to be an empty directory to prevent files from other runs from interfering with the information gathering. The call to the compile function has to be augmented with the verbose and the **compiler_workdir **arguments. In addition, please capture the stdout messages into a file (for example, by redirecting the stdout to a file)

```python
tfn.saved_model.compile(model_dir, compiled_model_dir, compiler_args=['--verbose', '2', '--pipeline', 'compile', 'SaveTemps'], compiler_workdir='./compiler-workdir')
```

- For Neuron Apache MXNet (Incubating), add compiler arguments as shown below and run the compilation process from an empty workdir:

```python
import mxnet as mx
import os
from packaging import version
mxnet_version = version.parse(mx.__version__)
if mxnet_version >= version.parse("1.8"):
    import mx_neuron as neuron
else:
    from mxnet.contrib import neuron
...
```

```python
os.environ['SUBGRAPH_INFO'] = '1'
compile_args = { '--verbose' : 2, '--pipeline' : 'compile', 'flags' : ['SaveTemps'] }
csym, cargs, cauxs = neuron.compile(sym, args, auxs, inputs=inputs, **compile_args)
```

Step 2: Run neuron-gatherinfo.py to gather information to share

The output result will be a tar.gz file.

Neuron Runtime information gathering

Step 1: EXECUTE inference steps for your workload with increased verbosity or debug levels

In the case of runtime information, the tool `neuron-dump.py` is used by **neuron-gatherinfo.py **to gather that information. Make sure that you have the neuron tools package (aws-neuron-tools) installed.
**Step 2: Run neuron-gatherinfo.py to gather information to share**

The output result will be a tar.gz file.

**Tool Usage Reference**

Run neuron-gatherinfo.py using the “--help” option:

```bash
bash $ ~/bin/neuron-gatherinfo.py --help
usage: neuron-gatherinfo.py [-h] [--additionalfileordir ADDFLDIR] [-c CCDIR]
                          [-i] [-f FILTERFILE] [-m] -o OUTDIR [-r RTDIR] -s
                          STDOUT [-v]

Usage: /home/user/bin/neuron-gatherinfo.py [options]
This program is used to gather information from this system for analysis
and debugging

optional arguments:
  -h, --help            show this help message and exit
  --additionalfileordir ADDFLDIR
                        Additional file or directory that the user wants to
                        provide in the archive. The user can sanitize this
                        file or directory before sharing
  -c CCDIR, --compileroutdir CCDIR
                        Location of the neuron-cc generated files
  -i, --include         By default, only the lines containing (grep) patterns
                        like 'nrtd|neuron|kernel:' from the syslog are copied.
                        Other lines are excluded. Using this option allows the
                        timestamp section of other lines to be included. The
                        rest of the contents of the line itself are elided.
                        Providing the timestamp section may provide time
                        continuity while viewing the copied syslog file
  -f FILTERFILE, --filter FILTERFILE
  -m, --modeldata       By using this option, the entire compiler work
                        directory's contents will be included (excluding the
                        .pb files, unless an additional option is used). This
                        would include model information, etc. The files that
                        are included, by default, are these: graph_def.neuron-
                        cc.log, all_metrics.csv, hh-tr-operand-
                        tensortensor.json
  -o OUTDIR, --out OUTDIR
                        The output directory where all the files and other
                        information will be stored. The output will be stored
                        as an archive as well as the actual directory where
                        all the contents are copied. This will allow a simple
                        audit of the files, if necessary. *** N O T E ***:
                        Make sure that this directory has enough space to hold
                        the files and resulting archive
  -r RTDIR, --runtimeoutdir RTDIR
                        Location of the neuron runtime generated files
  -s STDOUT, --stdout STDOUT
                        The file where the stdout of the compiler run was
                        saved
  -v, --verbose         Verbose mode displays commands executed and any
                        additional information which may be useful in
                        debugging the tool itself
```

16.2. Model Helper Tools
Examples

Example 1: no ML model information gathered (default behavior)

In this case, the tool will archive just the default information gathering:

```bash
$ sudo ~/bin/neuron-gatherinfo.py -o compile-and-run-info-for-debugging-no-model-info -i --verbose -s stdout-from-compile_resnet50.out -c compiler-workdir
```


<SNIP>

```
*****
Archive created at: 
/home/user/tutorials-3/compile-and-run-info-for-debugging-no-model-info/neuron-gatherinfo.tar.gz
From directory: 
/home/user/tutorials-3/compile-and-run-info-for-debugging-no-model-info/neuron-gatherinfo
*****
```

Example 2: model ML information gathered using the “—modeldata” option

In this case, the tool will archive the compiler work directory in addition to the default information gathering:

```bash
$ sudo ~/bin/neuron-gatherinfo.py -o compile-and-run-info-for-debugging-no-model-info -i --verbose -s stdout-from-compile_resnet50.out -c compiler-workdir --modeldata
```

<SNIP>

```
```

```
```

```
```

```
```

```
```

<SNIP>

```
*****
Archive created at: 
/home/user/tutorials-3/compile-and-run-info-for-debugging/neuron-gatherinfo.tar.gz
```

(continues on next page)
From directory:
/home/user/tutorials-3/compile-and-run-info-for-debugging/neuron-gatherinfo
******

**********************
Based on your command line option, we're also packaging these files:

graph_def.neuron-cc.log
all_metrics.csv
hh-tr-operand-tensortensor.json

And this directory: /home/user/tutorials-3/compiler-workdir
**********************

16.3 Resources for Neuron Runtime 1.x Users

16.3.1 Resources for Neuron Runtime 1.x Users

Migrating to Neuron Monitor 2.x

The new version of neuron-monitor is very similar to its predecessor, with a few differences.

- **Configuration file changes**
- **Output JSON structure changes**

**Configuration file changes**

- The "hw_counters" metric group has been renamed to "neuron_hw_counters" and is now part of the "system_metrics" category.

```
...
"system_metrics": [
...
    {  
        "period": "2s",
        "type": "neuron_hw_counters"
    }
]
```

- Each entry in the "neuron_runtimes" array can either have a "tag_filter" which is a regex that will be used to filter Neuron applications based on their tag; or an "address" field which is used to specify a neuron-rtd daemon GRPC address that should be monitored. For more details on backwards compatibility, read

```
{
    "period": "1s",
    "neuron_runtimes": [
```
Output JSON structure changes

- The "hw_counters" metric group has been renamed to "neuron_hw_counters" and is now part of the "system_metrics" category.

```json
"system_metrics": [
    {
        "type": "vcpu_usage"
    },
    {
        "type": "memory_info"
    },
    {
        "type": "neuron_hw_counters"
    }
]
```

- Each "neuron_runtime_data" object now contains 3 new properties: "pid", "address" and "neuron_runtime_tag". The "neuron_runtime_index" property has been removed:

```json
{
    "neuron_runtime_data": [
        {
            "pid": 0,
            "address": 
            "neuron_runtime_tag": "my_app_1",
            ...
        }
    ]
}
```

- The "loaded_models" array has been removed from the objects representing "neuroncore_counters" entries since the same information can be found in the "memory_used" group.

- A new "model" error category has been added:

```json
"error_summary": {
    "generic": 0,
    "numerical": 0,
    "transient": 0,
    "model": 0,
    "runtime": 0,
}
```
Neuron Monitor 2.x Backwards Compatibility with Neuron Runtime 1.x

neuron-Monitor 2.x can also monitor neuron-rtd daemons by adding an entry in the configuration file for each of them and specifying their GRPC address instead of a "tag_filter". These entries can coexist with entries for Neuron applications (which use the "tag_filter" field):

```
{
  "period": "1s",
  "neuron_runtimes": [
    {
      "address": "unix:/run/neuron.sock",
      "metrics": [
        ...
      ]
    },
    {
      "tag_filter": ".*",
      "metrics": [
        ...
      ]
    }
  ],
  "system_metrics": [
    ...
  ]
}
```

neuron-rtd entries in the output JSON will have a non-empty "address" field and the tag will contain its GRPC address:

```
{
  "neuron_runtime_data": [
    {
      "pid": 0,
      "address": "unix:/run/neuron.sock",
      "neuron_runtime_tag": "unix:/run/neuron.sock",
      "error": "",
      "report": {
        "neuroncore_counters": {
          [
          ...
        
      ]
    },
```
neuron-monitor collects metrics and stats from the Neuron Runtimes running on the system and streams the collected data to stdout in JSON format.

These metrics and stats are organized into metric groups which can be configured by providing a configuration file as described in Using neuron-monitor

When running, neuron-monitor will:

- Collect the data for the metric groups which, based on the elapsed time since their last update, need to be updated
- Take the newly collected data and consolidate it into a large report
- Serialize that report to JSON and stream it to stdout from where it can be consumed by other tools - such as the sample neuron-monitor-cloudwatch.py and neuron-monitor-prometheus.py scripts.
- Wait until at least one metric group needs to be collected and repeat this flow

Using neuron-monitor

neuron-monitor takes the following 2 optional arguments:

```
--verbose= Verbosity level (default: 0)
-c, --config-file= Path to configuration file
```

- --verbose Verbosity level, where Verbosity level can be 0 to 4, controls the amount of debugging and verbose information sent to stderr; 0: no output, 4: maximum verbosity, default is 0
- -c, --config-file path, where path is a valid path to a neuron-monitor JSON configuration file

Example:

```
neuron-monitor -c monitor.conf
```

Not specifying any option will enable collecting all the metric groups with a period of 5 seconds.

Example of a configuration file which enables all available metric groups for a single Neuron Runtime with a global update period of 1 second and sets an update period of 2 seconds for the "hw_counters" metric group:

```
{
    "period": "1s",
    "neuron_runtimes": [
        {
            "address": "unix:/run/neuron.sock",
            "metrics": [
                {
                    "type": "neuroncore_counters"
                },
                {
                    "type": "memory_used"
                },
                {
                    "type": "neuron_runtime_vcpu_usage"
                },
                {
                    "type": "inference_stats"
                }
            ]
        }
    ]
}
```

(continues on next page)
```json
{
  "neuron_runtimes": [
    {
      "address": "...",
      "metrics": [
        {
          "type": "vcpu_usage"
        },
        {
          "type": "memory_info"
        }
      ]
    }
  ],
  "system_metrics": [
    {
      "type": "vcpu_usage"
    },
    {
      "type": "memory_info"
    }
  ]
}
```

**JSON objects and fields in the settings file**

- **"neuron_runtimes"** - array of objects specifying which Neuron Runtimes to monitor and what metric groups are enabled for each runtime
  - "address" - address of this Neuron Runtime
  - "metrics" - array of objects specifying which metric groups to capture for this Neuron Runtime
    * "type" - type of metric group
- **"period"** - this field applies to metric group objects and sets the amount of time between two updates for that metric group
  - if can be specified as part of the root and/or neuron_runtime objects where it applies to all their children, and/or as part of a metric group object
  - if there’s no period specified, a default value of 5 seconds will be used
- **"system_metrics"** - array of objects specifying which system level metric groups are enabled

**Neuron Runtime-level metric groups**

- **neuroncore_counters** - NeuronCore related metrics
- **memory_used** - data on the amount of memory used by the Neuron Runtime
- **vcpu_usage** - Neuron Runtime vCPU utilization data
- **inference_stats** - Neuron Runtime-wide inference stats, including error count and latency
- **hw_counters** - counters for correctable and uncorrectable memory ecc events
System-wide metric groups

- vcpu_usage - system-wide vCPU usage
- memory_info - system-wide memory usage

Execution model

neuron-monitor waits for one or more metric groups to be up for update, then collects the corresponding data, consolidates it into a report which is streamed to stdout as a JSON and goes back to waiting.

The JSON output format

Whenever the report gets updated, a complete JSON is written to stdout. This is its structure:

```json
{
   "neuron_runtime_data": [
      {
         "neuron_runtime_index": 0,
         "error": "",
         "report": {
            "neuroncore_counters": {
               [[...]
            },
            "inference_stats": {
               [[...]
            },
            "memory_used": {
               [[...]
            },
            "hw_counters": {
               [[...]
            },
            "neuron_runtime_vcpu_usage": {
               [[...]
            }
         }
      }
   ],
   "system_data": {
      "vcpu_usage": {
         [[...]
      },
      "memory_info": {
         [[...]
      }
   }
}
```
• "neuron_runtime_data" is an array containing one entry per each Neuron Runtime specified in the settings file
  - "neuron_runtime_index" is the zero-based index of this Neuron Runtime in the configuration file
  - "error" specifies any error that occurred when collecting data from this Neuron Runtime
  - "report" will contain the results for the Neuron Runtime-level metric groups; their formats are described below

• "system_data" is similar to "neuron_runtime_data"'s "report" but only contains system-level metric groups (not associated to any Neuron Runtime)

There is also instance information added to the root object regardless of the configuration:

```
"instance_data": {
  "instance_name": "My_Instance",
  "instance_id": "i-0011223344556677a",
  "instance_type": "inf1.xlarge",
  "instance_availability_zone": "us-west-2b",
  "instance_availability_zone_id": "usw2-az2",
  "instance_region": "us-west-2",
  "ami_id": "ami-0011223344556677b",
  "subnet_id": "subnet-112233ee",
  "error": ""
}
```

Depending on when the instance was launched, the following fields might not be available:

• instance_availability_zone_id: available only for instances launched in 2020-08-24 and later
• instance_region: available only for instances launched on 2020-08-24 and later
• instance_name: available only if instance_region is set and aws-cli tools are installed The error will contain an error string if getting one of the fields, except those mentioned above, resulted in error.

Each metric group requested in the settings file will get an entry in the resulting output. The general format for such an entry is:

```
"metric_group": {
  "period": 1.015, // Actual captured period, in seconds
  "error": "", // Error, if any occurred, otherwise an empty string
  [...] // Metric group specific data
}
```
Neuron Runtime level metric groups

neuroncore_conters

```
"neuroncore_conters": {  
  "period": 1.000113182,  
  "neuroncores_in_use": {  
    "0": {  
      "neuroncore_utilization": 42.01,  
      "loaded_models": [  
        "my_model:my_subgraph1"  
      ]  
    },  
    "1": {  
      "neuroncore_utilization": 42.02,  
      "loaded_models": [  
        "my_model:my_subgraph2"  
      ]  
    },  
    "2": {  
      "neuroncore_utilization": 42.03,  
      "loaded_models": [  
        "my_model:my_subgraph3"  
      ]  
    },  
    "3": {  
      "neuroncore_utilization": 42.04,  
      "loaded_models": [  
        "my_model:my_subgraph4"  
      ]  
    }  
  }  
}  
```

- "neuroncores_in_use" is an object containing data for all the NeuronCores that were active when the data was captured, indexed by NeuronCore index: "neuroncore_index": { neuroncore_data }
  - "neuroncore_utilization" - NeuronCore utilization, in percent, during the captured period
  - "loaded_models" - array containing strings formatted as "model_name:subgraph_name" which represent what models and subgraphs are loaded and associated with this NeuronCore

- "error" - string containing any error that occurred when collecting the data

inference_stats

```
"inference_stats": {  
  "period": 1.030613214,  
  "error_summary": {  
    "generic": 0,  
    "numerical": 0,  
    "transient": 0,  
    "runtime": 0,  
    "hardware": 0  
  }  
}  
```

(continues on next page)
"inference_summary": {
  "completed": 123,
  "completed_with_err": 0,
  "completed_with_num_err": 0,
  "timed_out": 0,
  "incorrect_input": 0,
  "failed_to_queue": 0
},
"latency_stats": {
  "total_latency": {
    "p0": 0.01100001,
    "p1": 0.01100002,
    "p25": 0.01100004,
    "p50": 0.01100008,
    "p75": 0.01100010,
    "p99": 0.01100012,
    "p100": 0.01100013
  },
  "device_latency": {
    "p0": 0.01000001,
    "p1": 0.01000002,
    "p25": 0.01000004,
    "p50": 0.01000008,
    "p75": 0.01000010,
    "p99": 0.01000012,
    "p100": 0.01000013
  }
},
"error": ""

• "error_summary" is an object containing the error counts for the captured period indexed by their type
  - "generic" - generic inference errors
  - "numeric" - NAN inference errors
  - "transient" - recoverable errors, such as ECC corrections
  - "runtime" - Neuron Runtime errors
  - "hardware" - hardware errors such as uncorrectable ECC issues

• "inference_summary" is an object containing all inference outcome counts for the captured period indexed by their type
  - "completed" - inferences completed successfully
  - "completed_with_err" - inferences that ended in an error other than numeric
  - "completed_with_num_err" - inferences that ended in a numeric error
  - "timed_out" - inferences that took longer than the Neuron Runtime configured timeout value
  - "incorrect_input" - inferences that failed to start due to incorrect input being provided
  - "failed_to_queue" - inference requests that were rejected due to Neuron Runtime not being able to queue them

• "latency_stats" contains two objects containing latency percentiles, in seconds, for the data captured for inferences executed during the captured period. If there are no inferences being executed during this time, the two objects will be null (i.e. "total_latency": null)
- "total_latency" - percentiles, in seconds, representing latency for an inference as measured by the Neuron Runtime
- "device_latency" - percentiles, in seconds, representing time spent by an inference exclusively on the Neuron device
- "error" - string containing any error that occurred when collecting the data

memory_used

```
"memory_used": {
  "period": 1.030366715,
  "neuron_runtime_used_bytes": {
    "host": 1000000,
    "neuron_device": 2000000
  },
  "loaded_models": [
    {
      "name": "my_model",
      "uuid": "aaaaaaaaaaabbbbbbbbbbb000000000000000000000000",
      "model_id": 10234,
      "is_running": true,
      "memory_used_bytes": {
        "host": 250000,
        "neuron_device": 500000
      },
      "subgraphs": {
        "sg00": {
          "memory_used_bytes": {
            "host": 250000,
            "neuron_device": 500000
          },
          "neuroncore_index": 2,
          "neuron_device_index": 0
        }
      }
    },
    [...]
  ],
  "error": ""
},
```

- "runtime_memory" summarizes the amount of memory used by the Neuron Runtime at the time of capture
  - "neuron_runtime_used_bytes" - current amount of memory used by the Neuron Runtime
  - all memory usage objects contain these two fields:
    * "host" - host DRAM usage in bytes
    * "neuron_device" - Neuron device DRAM usage in bytes
- "loaded_models" - array containing objects representing loaded models
  - "name" - name of the model
  - "uuid" - unique id for the model
  - "model_id" - Neuron Runtime-assigned ID for this model
  - "is_running" - true if this model is currently started, false otherwise
- "memory_used_bytes" - total memory usage for the model
- "subgraphs" - object containing all the subgraph for the model indexed by their name:
  "subgraph_name": { subgraph_data }
  * "memory_used_bytes" - memory usage for this subgraph
  * "neuroncore_index" - NeuronCore index with which the subgraph is associated
  * "neuron_device_index" - Neuron device index on which the subgraph is loaded
- "error" - string containing any error that occurred when collecting the data

**hw_counters**

```json
"hw_counters": {
  "period": 1.030359284,
  "neuron_devices": [
    {,
      "neuron_device_index": 0,
      "mem_ecc_corrected": 0,
      "mem_ecc_uncorrected": 0,
      "sram_ecc_uncorrected": 0
    },
    "error": ""
  ],
},
```

- "neuron_devices" - array containing ECC data for all Neuron devices controlled by this Neuron Runtime for the captured period
  - "neuron_device_index" - Neuron device index
  - "mem_ecc_corrected" - number of corrected ECC events in the Neuron device’s DRAM
  - "mem_ecc_uncorrected" - number of uncorrected ECC events in the Neuron device’s DRAM
  - "sram_ecc_uncorrected" - number of uncorrected ECC events in the Neuron device’s SRAM
- "error" - string containing any error that occurred when collecting the data

**neuron_runtime_vcpu_usage**

```json
"neuron_runtime_vcpu_usage": {
  "period": 1.030604818,
  "vcpu_usage": {,
    "user": 42.01,
    "system": 12.34
  },
  "error": ""
}
```

- "vcpu_usage" - object showing vCPU usage in percentages for the Neuron Runtime during the captured period
  - "user" - percentage of time spent in user code by this Neuron Runtime
  - "system" - percentage of time spent in kernel code by this Neuron Runtime
- "error" - string containing any error that occurred when collecting the data
System level metric groups

vcpu_usage

```
"vcpu_usage": {
  "period": 0.999974868,
  "average_usage": {
    "user": 32.77,
    "nice": 0,
    "system": 22.87,
    "idle": 39.36,
    "io_wait": 0,
    "irq": 0,
    "soft_irq": 0
  },
  "usage_data": {
    "0": {
      "user": 34.41,
      "nice": 0,
      "system": 27.96,
      "idle": 37.63,
      "io_wait": 0,
      "irq": 0,
      "soft_irq": 0
    },
    "1": {
      "user": 56.84,
      "nice": 0,
      "system": 28.42,
      "idle": 14.74,
      "io_wait": 0,
      "irq": 0,
      "soft_irq": 0
    },
    [...]
  },
  "context_switch_count": 123456,
  "error": ""
}
```

- each vCPU usage object contains the following fields:
  - "user" - percentage of time spent in user code
  - "nice" - percentage of time spent executing niced user code
  - "system" - percentage of time spent executing kernel code
  - "idle" - percentage of time spent idle
  - "io_wait" - percentage of time spent waiting for IO operations
  - "irq" - percentage of time spent servicing hardware interrupts
  - "soft_irq" - percentage of time spent servicing software interrupts
- "average_usage" - contains the average usage across all vCPUs during the captured period
- "usage_data" - contains per vCPU usage during the captured period
- "context_switch_count" - contains the number of vCPU context switches during the captured period
• "error" - string containing any error that occurred when collecting the data

memory_info

```
"memory_info": {  
    "period": 5.346411129,  
    "memory_total_bytes": 49345835008,  
    "memory_used_bytes": 16042344448,  
    "swap_total_bytes": 0,  
    "swap_used_bytes": 0,  
    "error": ""
}
```

• "memory_total_bytes" - total size of the host memory, in bytes
• "memory_used_bytes" - amount of host memory in use, in bytes
• "swap_total_bytes" - total size of the host swap file, in bytes
• "swap_used_bytes" - amount of swap memory in use, in bytes

Companion scripts

neuron-monitor is installed with two example Python companion script: neuron-monitor-cloudwatch.py and neuron-monitor-prometheus.py.

neuron-monitor-cloudwatch.py

It requires Python3 and the boto3 Python module. It is installed to: /opt/aws/neuron/bin/neuron-monitor-cloudwatch.py.

Using neuron-monitor-cloudwatch.py

```
neuron-monitor | neuron-monitor-cloudwatch.py --namespace <namespace> --region <region>
```

For example:

```
neuron-monitor | neuron-monitor-cloudwatch.py --namespace neuron_monitor_test --region us-west-2
```

neuron-monitor-prometheus.py

It requires Python3 and the Prometheus client Python module. It is installed to: /opt/aws/neuron/bin/neuron-monitor-prometheus.py.
Using neuron-monitor-prometheus.py

```
neuron-monitor | neuron-monitor-prometheus.py --port <port>
```

For example:

```
neuron-monitor | neuron-monitor-prometheus.py --port 8008
```

The default value for `--port` is 8000.

If your data visualization framework is Grafana, we provided a Grafana dashboard which integrates with Prometheus and this script.

**Neuron Tools 1.x Basic Tools**

**Identifying Neuron Devices**

To identify number of Neuron Devices in a given instance use the `neuron-ls` command.

```
$ neuron-ls
```

```
+--------+--------+--------+-----------+--------------+---------+---------+---------+
| NEURON | NEURON | NEURON | CONNECTED | PCI | RUNTIME | RUNTIME | RUNTIME |
| DEVICE | CORES | MEMORY | DEVICES | BDF | ADDRESS | PID | VERSION |
+--------+--------+--------+-----------+--------------+---------+---------+---------+
| 0 | 4 | 8 GB | 1 | 0000:00:1c.0 | NA | 12410 | NA |
| 1 | 4 | 8 GB | 2, 0 | 0000:00:1d.0 | NA | 12410 | NA |
| 2 | 4 | 8 GB | 3, 1 | 0000:00:1e.0 | NA | 12410 | NA |
| 3 | 4 | 8 GB | 2 | 0000:00:1f.0 | NA | 12410 | NA |
+--------+--------+--------+-----------+--------------+---------+---------+---------+

The above output is taken from an Inf1.6xlarge instance.

- **NEURON DEVICE** -> Logical ID assigned to the NeuronDevice. This id can be used when configuring multiple runtimes to use different NeuronDevices.
- **NEURON CORES** -> Number of NeuronCores present in the NeuronDevice.
- **NEURON MEMORY** -> Amount DRAM memory in NeuronDevice.
- **CONNECTED DEVICES** -> Shows other NeuronDevices connected to this NeuronDevice.
- **PCI BDF** -> PCI Bus Device Function (BDF) ID of the device.
- **RUNTIME ADDRESS** -> Shows address of runtime process using this NeuronDevice.
- **RUNTIME PID** -> Shows process id of runtime process using this NeuronDevice.
- **RUNTIME VERSION** -> Shows version of runtime process using this NeuronDevice.
NeuronCore Groups

Multiple NeuronCores (NC) can be combined to form a NeuronCore Group (NCG). Neuron framework layer will automatically create a default NeuronCore Group. To view list of available NCGs the following command can be used.

```bash
$ neuron-cli list-ncg
Device count 4 NC count 16
Found 4 NCG's
+--------+----------+--------------------+----------------+
| NCG ID | NC COUNT | DEVICE START INDEX | NC START INDEX |
+--------+----------+--------------------+----------------+
| 1 | 2 | 0 | 0 |
| 2 | 4 | 0 | 2 |
| 3 | 3 | 1 | 2 |
| 4 | 1 | 2 | 1 |
+--------+----------+--------------------+----------------+
```

The above examples shows there are 4 NCGs created on the system with the following grouping NCG ID 1: Device0:(Core0, Core1) NCG ID 2: Device0:(Core2, Core3), Device1:(Core0, Core1) NCG ID 3: Device1:(Core2, Core3), Device2:(Core0) NCG ID 2: Device1:(Core1)

Listing Models

Multiple models can be loaded into a single NCG, but only one can be in READY state at any given moment. Inference can be performed only on models in the READY state.

The `neuron-cli list-model` command should be used to view all the models.

```bash
$ neuron-cli list-model
+----------------------------------------------+----------+--------------+------------+
| UUID | MODEL ID | MODEL STATUS | NEURON DEVICE | NC | NC | NAME |
+----------------------------------------------+----------+--------------+------------+
|     |          |              |             |     |     |      |
+----------------------------------------------+----------+--------------+------------+
| 63c43dd60b0411eaa9160288cac7f65c30 | 10011 | STANDBY | 1 | 0 | 1 | test0_1_concat_multi |
| 63c43dd60b0411eaa9160288cac7f65c30330808637f | 10010 | STANDBY | 0 | 0 | 1 | test0_1_concat_multi |
| 63c43dd60b0411eaa9160288cac7f65ce078 | 10009 | READY | 1 | 1 | 1 | test0_1_concat_multi |
| 63c43dd60b0411eaa9160288cac7f65ca05f | 10008 | READY | 1 | 0 | 1 | test0_1_concat_multi |
| 6a9726 | 10007 | READY | 0 | 2 | 2 | onv_h1_2tpb_cpu_2tpb |
| 529c31da0b0411eaa95730288cac7f65cb03b0afc627f | 10006 | READY | 0 | 0 | 0 | t-test0_5conv_h1_cpu |
+----------------------------------------------+----------+--------------+------------+
```

- UUID -> UUID generated for this model during compile time.
- MODEL ID -> Neuron Runtime identifier for this model.
• **MODEL STATUS -> READY** = The model is loaded on to the NeuronDevice and active on the NeuronCore. (Inference can be done only on models with READY state) **STANDBY** = The model is loaded on to the NeuronDevice but another model is currently active on the NeuronCore. (A model switch is needed to start inference)

**View Resource Usage**

Each model loaded consumes a different amount of memory (host and device), NeuronCore and CPU usage. The `neuron-top` command can be used to view the memory and NeuronCore usage.

```bash
$ neuron-top
neuron-top - 2020-02-12 23:03:15
NN Models: 2 total, 2 running
Number of VNCs tracked: 16
0000:00:1c.0 Utilizations: Neuron core0 0.00%, Neuron core1 0.00%, Neuron core2 0.00%, Neuron core3 0.00%,
0000:00:1e.0 Utilizations: Neuron core0 0.00%, Neuron core1 0.00%, Neuron core2 0.00%, Neuron core3 0.00%,

<table>
<thead>
<tr>
<th>Model ID</th>
<th>Model Name</th>
<th>UUID</th>
<th>Node ID</th>
<th>Subgraph</th>
<th>Exec. Unit</th>
<th>Host Mem</th>
<th>Device Mem</th>
<th>Neuron Core %</th>
</tr>
</thead>
<tbody>
<tr>
<td>10018</td>
<td></td>
<td>d12cf238420d1e8a8e270a0b835c0a32</td>
<td>3</td>
<td>0</td>
<td>0000:00:1e.0:0</td>
<td>33554816</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>10017</td>
<td></td>
<td>d12cf238420d1e8a8e270a0b835c0a32</td>
<td>3</td>
<td>0</td>
<td>0000:00:1c.0:0</td>
<td>33554816</td>
<td>0.00</td>
<td></td>
</tr>
</tbody>
</table>
```

In the above output:

- **Model ID** -> Unique Identifier for models loaded in the NeuronDevice
- **Model Name** -> Neuron Compiler Version-compiler work directory/User defined model name
- **Node ID** -> For Internal use only
- **UUID** -> Unique Id assigned by the Neuron Compiler for a Model
- **Exec. Unit** -> BDF of Neuron Device followed by the Neuron Core ID, b:d:f.NC
- **Host Mem** -> Host memory consumed by the Model in bytes
- **Device Mem** -> NeuronDevice memory consumed by the Model in bytes
- **Neuron Core %** -> Utilization % of the neuron core at sample time. If there are no active inferences this value will be 0.

**What's New**

**Neuron Tools 1.x Release Notes**

This documents lists the release notes for AWS Neuron tools. Neuron tools are used for debugging, profiling and gathering inferentia system information.
Table of Contents

- Known Issues and Limitations
  - [1.7.25.0]
  - [1.7.20.0]
  - [1.7.10.0]
  - [1.7.4.0]
  - [1.6.1.0]
  - [1.5.6.0]
  - [1.4.12.0]
  - [1.4.8.0]
  - [1.4.2.0]
  - [1.3.1.0]
  - [1.2.7.0]
  - [1.1.228.0]
  - [1.0.11054.0]
  - [1.0.10616.0]
  - [1.0.10272.0]
  - [1.0.10182.0]
  - [1.0.9700.0]
  - [1.0.9043.0]
  - [1.0.8550.0]
  - [1.0.8131.0]
  - [1.0.6554.0]
  - [1.0.5832.0]
  - [1.0.5165.0]
  - [1.0.4587.0]
  - [1.0.4250.0]
  - [1.0.3657.0]
Known Issues and Limitations

- neuron-top has a visible screen stutter as the number of loaded models increases above 40. This is only a visual issue with no impact on performance. The issue is caused by the re-rendering the UI on screen refresh. We will fix this in a future release.

[1.7.25.0]
Date: 08/30/2021

Resolved Issues

- Minor updates.

[1.7.20.0]
Date: 08/12/2021

Resolved Issues

- Minor updates.

[1.7.10.0]
Date: 07/26/2021

Resolved Issues

- Bug fixes and minor enhancements.

[1.7.4.0]
Date: 07/02/2021

Resolved Issues

- Bug fixes and minor enhancements.
[1.6.1.0]

Date: 05/28/2021

Resolved Issues

• Bug fixes and minor enhancements.

[1.5.6.0]

Date: 05/01/2021

Major New Features

• Neuron profiler improved to support an upcoming change to neff generation.

Resolved Issues

• Bug fixes and minor enhancements.

[1.4.12.0]

Date: 03/04/2020

Resolved Issues

• Bug fixes and minor enhancements.

[1.4.8.0]

Date: 02/24/2020

Resolved Issues

• Fix for CVE-2021-3177.
[1.4.2.0]

Date: 01/30/2020

Major New Features

• Adds neuron-htop as a new front-end over neuron-monitor. New tool is an improved interface over neuron-top with an intention of eventually replacing the older version of neuron-top in the future. Try it out and provide any feedback/issus on GitHub.

Improvements

• Extended support for uncompressed neffs to all tools.

Resolved Issues

• Runtime memory usage error was not clearing in neuron-monitor.

[1.3.1.0]

Date: 12/23/2020

Improvements

• Minor internal enhancement to neuron-monitor to help track system resources used by neuron-monitor.

[1.2.7.0]

Date: 11/17/2020

Major New Features

• neuron-monitor now provides system-wide memory usage statistics. Many JSON field names have been updated. We’ve added a new sample script which exports most of neuron-monitor’s metrics to a Prometheus monitoring server. Additionally, we also provided a sample Grafana dashboard - in JSON format - which can be imported to a Grafana instance via its web interface. This dashboard can then present the metric data made available to Prometheus by neuron-monitor. More details on how to use neuron-monitor with this new feature can be found in the Neuron Monitor User Guide.

• Neuron tools updated the NeuronCore utilization metric to include all inf1 compute engines and DMAs. The new metric definition is more comprehensive and provides a better representation of execution efficiency.
Resolved Issues

- Fixed a memory leak in neuron-monitor when attempting to connect to the GRPC address of a Neuron Runtime which is not running.

[1.1.228.0]

Date: 10/22/2020

Major New Features

- n/a

Improvements

- All the tools now use nd0:nc0 to identify NeuronDevice and NeuronCore instead of bdf.
- neuron-cli list-model now shows NCG Id for each loaded model.
- neuron-top columns are reordered to show usage details first.
- neuron-top shows weights in human readable format(MB, GB).

Resolved Issues

- neuron-top now correctly shows NC usage if multiple models are loaded onto the same NC.

[1.0.11054.0]

Date: 09/22/2020

Major New Features

Beta release of neuron-monitor for streaming metric information about inference execution from your inf1. We provided a sample script for connecting neuron-monitor output directly into CloudWatch. Usage of the new tool is a simple one-liner:

```
neuron-monitor | neuron-monitor-cloudwatch.py --namespace neuron_monitor_test --region us-west-2
```

More details on how to use neuron-monitor can be found in the Neuron Monitor User Guide.
Improvements

- neuron-ls now shows connected devices as a list. This information can be used when creating a neuron core group.

Resolved Issues

- n/a

1.0.10616.0

Date: 08/19/2020

Major New Features

- n/a

Improvements

- Various minor improvements.

Resolved Issues

- n/a

1.0.10272.0

Date: 08/08/2020

Major New Features

- n/a

Improvements

- Various minor improvements.
Resolved Issues

• n/a

[1.0.10182.0]

Date: 08/05/2020

Major New Features

• n/a

Improvements

• Various minor improvements.

Resolved Issues

• n/a

[1.0.9700.0]

Date: 07/16/2020

Major New Features

• n/a

Improvements

• neuron-ls now supports JSON output format through a new command line option –json-output.

Resolved Issues

• n/a
Summary

- Enhancements to neuron-cli to improve loading of large models
- Fix aws-neuron-runtime-base uninstall to cleanup all the relevant files
- Migrated neuron-discovery service to use IMDSv2 to query instance type

Major New Features

- Added new commandline options to **neuron-cli** to improve the performance on loading large models

  --ncg-id <value>

  Legal values for ncg-id:
  - “-1”: runtime will create the NCG (default)
  - “0”: NCG will be created by neuron-cli
  - “>=1”: Model will be loaded to the NCG id specified

  During model load, neuron-cli parses the NEFF file for parameters needed to create an NCG. The runtime will parse the same NEFF file a second time during the load. Allowing the runtime to create the NCG reduces load time by skipping the redundant parse in neuron-cli.

  --enable-direct-file-load

  By default, neuron-cli loads models into its own memory and streams the model to the Neuron Runtime using GRPC. When the `--enable-direct-file-load` flag is passed, the load operation will skip the copy and only pass the filepath of the model to the Neuron Runtime. This saves time and memory during model loads.

Resolved Issues

- None

[1.0.8550.0]

Date: 5/15/2020
Summary

• Point fix for installation and startup errors of neuron-discovery service in the aws-neuron-runtime-base package.

Please update to aws-neuron-runtime-base package version 1.0.7173 or newer:

```bash
# Ubuntu 18 or 16:
sudo apt-get update
sudo apt-get install aws-neuron-runtime-base

# Amazon Linux, Centos, RHEL
sudo yum update
sudo yum install aws-neuron-runtime-base
```

Major New Features

• None

Resolved Issues

• Installation of aws-neuron-runtime-base version 1.0.7044 fails to successfully move service files into the service folder. Release of aws-neuron-runtime-base version 1.0.7173 fixes this installation issue.

• Added a dependency on the networking service in the neuron-discovery service to avoid potential for discovery to start before networking. If networking starts first, neuron-discovery will fail to start.

[1.0.8131.0]

Date: 5/11/2020

Summary

Major New Features

• All tools now support use of an environment variable (NEURON_RTD_ADDRESS) to specify the runtime address or by explicitly specifying the address with the -a flag. Not specifying an address will continue to rely on default address set during installation.

• When run as root, neuron-ls output will now include runtime details (address, pid, and version).

```bash
$ sudo neuron-1s
+--------------+---------+--------+----------+-----------+------+------+-------------
 | PCI BDF     | LOGICAL | NEURON | MEMORY | MEMORY | EAST | WEST | RUNTIME | RUNTIME | RUNTIME |
 | ADDRESS     | PID     | VERSION| ADDRESS | ADDRESS | ADDRESS | ADDRESS | ADDRESS | ADDRESS | ADDRESS |
+--------------+---------+--------+----------+-----------+------+------+-------------
| 0000:00:1c.0 | 0 | 4 | 4096 MB | 4096 MB | 1 | 0 | unix:/run/ 
| neuron.sock | 8871 | 1.0.x.x |
| 0000:00:1d.0 | 1 | 4 | 4096 MB | 4096 MB | 1 | 1 | unix:/run/ 
| neuron.sock | 8871 | 1.0.x.x |
```
Resolved Issues

- Backwards compatibility of neuron-top with older versions of Neuron Runtime is now restored.

Known Issues and Limitations

- neuron-top has a visible screen stutter as the number of loaded models increases above 40. This is only a visual issue with no impact on performance. The issue is caused by the re rendering the UI on screen refresh. We will fix this in a future release.

[1.0.6554.0]

Date: 3/26/2020

Summary

Fixed the issue where neuron-top was negatively impacting inference throughput.

Major New Features

N/A

Resolved Issues

- neuron-top no longer has a measurable impact on inference throughput regardless of instance size.
  - This version of neuron-top requires Neuron Runtime version 1.0.6222.0 or newer. Backwards compatibility will be fixed in the next release.
- neuron-top now correctly shows when a model is unloaded.
Known Issues and Limitations

- neuron-top has a visible screen stutter as the number of loaded models increases above 40. This is only a visual issue with no impact on performance. The issue is caused by the re rendering the UI on screen refresh. We will fix this in a future release.

[1.0.5832.0]
Date: 2/27/2020

Summary

Improved neuron-cli output to display device placement information about each model.

Major New Features

N/A

Resolved Issues

N/A

Known Issues and Limitations

- neuron-top consumes one vCPU to monitor hardware resources, which might affect performance of the system on inf1.xlarge. Using a larger instance size will not have the same limitation. In a future release we will improve this for smaller instance sizes.

[1.0.5165.0]
Date: 1/27/2020

Summary

Improved neuron-top load time, especially when a large amount of models are loaded.

Major New Features

N/A
Resolved Issues

N/A

Known Issues and Limitations

- neuron-top consumes one vCPU to monitor hardware resources, which might affect performance of the system on inf1.xlarge. Using a larger instance size will not have the same limitation. In a future release we will improve this for smaller instance sizes.

Other Notes

[1.0.4587.0]

Date: 12/20/2019

Summary

Minor bug fixes to neuron-top and neuron-ls.

Major New Features

Resolved Issues

- neuron-top: now shows model name and uuid to help distinguish which model is consuming resources. Previously only showed model id.
- neuron-ls: lists device memory size correctly in MB

Known Issues and Limitations

Other Notes

[1.0.4250.0]

Date: 12/1/2019

Summary

Major New Features

Resolved Issues

- neuron-top may take longer to start and refresh when numerous models are loaded
- neuron-top may crash when trying to calculate the utilization of the devices
Known Issues and Limitations

Other Notes

[1.0.3657.0]

Date: 11/25/2019

Major New Features

N/A, this is the first release.

Resolved Issues

N/A, this is the first release.

Known Issues and Limits

- neuron-top may take longer to start and refresh when numerous models are loaded.
  - Workaround: Unload the models not in use before using neuron-top
- neuron-top may crash when trying to calculate the utilization of the devices.

Other Notes


## 17.1 Current Release - Neuron 1.16.3 (01/05/2022)

### 17.1.1 Release included packages

List of Neuron packages included in Neuron release version 1.16.3:

<table>
<thead>
<tr>
<th>Package</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>driver</td>
<td>aws-neuron-dkms-2.2.6.0</td>
</tr>
<tr>
<td>runtime-lib</td>
<td>libnrt.so (version 2.2.18.0)</td>
</tr>
<tr>
<td>k8-plugin</td>
<td>aws-neuron-k8-plugin-1.7.3.0</td>
</tr>
<tr>
<td>k8-scheduler</td>
<td>aws-neuron-k8-scheduler-1.7.3.0</td>
</tr>
<tr>
<td>tools</td>
<td>aws-neuron-tools-2.0.327.0</td>
</tr>
<tr>
<td>compiler</td>
<td>neuron-cc-1.8.5.0</td>
</tr>
<tr>
<td>neuronperf</td>
<td>neuronperf-1.0.85.0</td>
</tr>
<tr>
<td>pytorch</td>
<td>torch-neuron-1.5.1.2.0.536.0</td>
</tr>
<tr>
<td>pytorch</td>
<td>torch-neuron-1.7.1.2.0.536.0</td>
</tr>
<tr>
<td>pytorch</td>
<td>torch-neuron-1.8.1.2.0.536.0</td>
</tr>
<tr>
<td>pytorch</td>
<td>torch-neuron-1.9.1.2.0.536.0</td>
</tr>
<tr>
<td>tensorflow</td>
<td>tensorflow-neuron-1.15.5.2.0.4.0</td>
</tr>
<tr>
<td>tensorflow</td>
<td>tensorflow-neuron-2.1.4.2.0.4.0</td>
</tr>
<tr>
<td>tensorflow</td>
<td>tensorflow-neuron-2.2.3.2.0.4.0</td>
</tr>
<tr>
<td>tensorflow</td>
<td>tensorflow-neuron-2.3.4.2.0.4.0</td>
</tr>
<tr>
<td>tensorflow</td>
<td>tensorflow-neuron-2.4.3.2.0.4.0</td>
</tr>
<tr>
<td>tensorflow</td>
<td>tensorflow-neuron-2.5.1.2.0.4.0</td>
</tr>
<tr>
<td>tensorflow-server</td>
<td>tensorflow-model-server-neuron-1.15.0.2.0.4.0</td>
</tr>
<tr>
<td>tensorflow-server</td>
<td>tensorflow-model-server-neuron-2.1.4.2.0.4.0</td>
</tr>
<tr>
<td>tensorflow-server</td>
<td>tensorflow-model-server-neuron-2.2.3.2.0.4.0</td>
</tr>
<tr>
<td>tensorflow-server</td>
<td>tensorflow-model-server-neuron-2.3.4.2.0.4.0</td>
</tr>
<tr>
<td>tensorflow-server</td>
<td>tensorflow-model-server-neuron-2.4.3.2.0.4.0</td>
</tr>
<tr>
<td>tensorflow-server</td>
<td>tensorflow-model-server-neuron-2.5.2.2.0.4.0</td>
</tr>
</tbody>
</table>
17.1.2 Release supported frameworks

List of frameworks included in Neuron release version 1.16.3:

<table>
<thead>
<tr>
<th>Framework</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>pytorch</td>
<td>pytorch-1.5.1</td>
</tr>
<tr>
<td>pytorch</td>
<td>pytorch-1.7.1</td>
</tr>
<tr>
<td>pytorch</td>
<td>pytorch-1.8.1</td>
</tr>
<tr>
<td>pytorch</td>
<td>pytorch-1.9.1</td>
</tr>
<tr>
<td>tensorflow</td>
<td>tensorflow-1.15.5</td>
</tr>
<tr>
<td>tensorflow</td>
<td>tensorflow-2.1.4</td>
</tr>
<tr>
<td>tensorflow</td>
<td>tensorflow-2.2.3</td>
</tr>
<tr>
<td>tensorflow</td>
<td>tensorflow-2.3.4</td>
</tr>
<tr>
<td>tensorflow</td>
<td>tensorflow-2.4.3</td>
</tr>
<tr>
<td>tensorflow</td>
<td>tensorflow-2.5.1</td>
</tr>
<tr>
<td>mxnet</td>
<td>mxnet-1.5.1</td>
</tr>
<tr>
<td>mxnet</td>
<td>mxnet-1.8.0</td>
</tr>
</tbody>
</table>

See SDK Maintenance Policy for more information.

17.1.3 Dependency Software Supported Versions

<table>
<thead>
<tr>
<th>Software</th>
<th>Supported</th>
</tr>
</thead>
</table>
| Python   | • Python 3.6  
          | • Python 3.7  |

17.2 Deep Learning AMI (DLAMI) Versions Matrix

The DLAMI version can be determined by examining the AMI name of the EC2 instance from EC2 console or examining the file README in the default home directory.

Table 1: latest DLAMI versions

<table>
<thead>
<tr>
<th>DLAMI version</th>
<th>Default Neuron version</th>
<th>How to upgrade to latest Neuron</th>
</tr>
</thead>
<tbody>
<tr>
<td>v46.0</td>
<td>v1.14.0 (May 2021)</td>
<td>see Neuron Pip Packages within DLAMI Conda Environments FAQ</td>
</tr>
<tr>
<td>v43.0</td>
<td>v1.12.2 (Mar 2021)</td>
<td>see Neuron Pip Packages within DLAMI Conda Environments FAQ</td>
</tr>
<tr>
<td>v42.2</td>
<td>v1.12.2 (Mar 2021)</td>
<td>see Neuron Pip Packages within DLAMI Conda Environments FAQ</td>
</tr>
<tr>
<td>v41.0</td>
<td>v1.12.1 (Feb 2021)</td>
<td>see Neuron Pip Packages within DLAMI Conda Environments FAQ</td>
</tr>
<tr>
<td>v40.0</td>
<td>v1.11.0 (Dec 2020)</td>
<td>see Neuron Pip Packages within DLAMI Conda Environments FAQ</td>
</tr>
<tr>
<td>v39.0</td>
<td>v1.11.0 (Dec 2020)</td>
<td>see Neuron Pip Packages within DLAMI Conda Environments FAQ</td>
</tr>
<tr>
<td>v38.0</td>
<td>v1.10.0 (Nov 2020)</td>
<td>see Neuron Pip Packages within DLAMI Conda Environments FAQ</td>
</tr>
<tr>
<td>v37.0</td>
<td>v1.10.0 (Nov 2020)</td>
<td>see Neuron Pip Packages within DLAMI Conda Environments FAQ</td>
</tr>
</tbody>
</table>

See Additional Setup Resources for more information.
17.3 Previous Releases Content

pre-release-content
The AWS Neuron feature roadmap provides visibility onto what we are working on in terms of functional and performance in the near future. We hope this will help you better plan how to use Neuron with your products. We’d love to get our customers feedback as well, to help us ensure we are working on the most important requests.
19.1 Security disclosures

If you think you’ve found a potential security issue, please do not post it in the Issues. Instead, please follow the instructions here (https://aws.amazon.com/security/vulnerability-reporting/) or email AWS security directly (mailto: aws-security@amazon.com).

19.2 SDK Maintenance Policy

19.2.1 Overview

This document outlines the maintenance policy for AWS Neuron Software Development Kit (SDK) and its underlying dependencies. AWS regularly provides the Neuron SDK with updates that may contain support for new or updated APIs, new features, enhancements, bug fixes, security patches, or documentation updates. Updates may also address changes with dependencies, language runtimes, and operating systems. Neuron SDK releases are available as Conda (up to Neuron 1.13.0) and Pip Packages that can be installed within Amazon Machine Images (AMIs).

We recommend users to stay up-to-date with SDK releases to keep up with the latest features, security updates, and underlying dependencies. Continued use of an unsupported SDK version is not recommended and is done at the user’s discretion.
19.2.2 Neuron SDK

AWS Neuron is the SDK for AWS Inferentia, the custom designed machine learning chips enabling high-performance deep learning inference applications on EC2 Inf1 instances. Neuron includes a deep learning compiler, runtime and tools that are natively integrated into TensorFlow, PyTorch and MXNet. With Neuron, you can develop, profile, and deploy high-performance inference applications on top of EC2 Inf1 instances.

The Neuron SDK release versions are in the form of X.Y.Z where X represents the major version and Y represent the minor version. Increasing the major version of an SDK indicates that this SDK underwent significant and substantial changes, and some of those changes may not maintain the same programming model. Increasing the minor version of an SDK indicates that this SDK underwent addition of new features, support of new dependency software versions, end-of-support of certain dependency software, enhancement and/or bugfixes. Applications may need to be updated in order for them to work with the newest SDK version. It is important to update major versions carefully and in accordance with the upgrade guidelines provided by AWS.

19.2.3 Dependency Software

Neuron SDK has underlying dependencies, such as language runtimes, operating systems, or third party libraries and machine learning frameworks. These dependencies are typically tied to the language community or the vendor who owns that particular component. The following terms are used to classify underlying dependencies:

- Operating system (OS): Examples include Amazon Linux AMI, Amazon Linux 2.
- Language runtime: Examples include Python.
- Third party library / framework: Examples include PyTorch, TensorFlow, MXNet and ONNX.

Each community or vendor maintains their own versioning policy and publishes their own end-of-support schedule for their product.

19.2.4 Neuron SDK version life-cycle

The life-cycle for Neuron SDK version consists of 3 phases, which are outlined below.

- **Supported (Phase 1)**
  During this phase, AWS will provide critical bugfixes and security patches. Usually AWS will support each Neuron SDK version for at least 12 months, but AWS reserves the right to stop supporting an SDK version before the 12 months period.

  **Note:** AWS will address new features or Dependency Software updates by publishing a new version with an increment in the Neuron SDK minor version.

- **End-of-Support Announcement (Phase 2)**
  AWS will announce the End-of-Support phase at least 3 months before a specific Neuron SDK version enters End-of-Support phase. During this period, the SDK will continue to be supported.

- **End-of-Support (Phase 3)**
  When a Neuron SDK version reaches end-of support, it will no longer receive critical bugfixes and security patches. Previously published Neuron SDK versions will continue to be available via Conda (up to Neuron 1.13.0) or Pip packages. Use of an SDK version which has reached end-of-support is done at the user’s discretion. We recommend users to upgrade to the latest Neuron SDK version.
19.2.5 Dependency Software version life-cycle

The life-cycle for Dependency Software version consists of 4 phases, but there may not be a Phase 3 (Maintenance) period in some cases. The phases are outlined below.

- **Supported (Phase 1)**
  
  During this phase, Dependency Software version is supported. AWS will provide regular updates, bug fixes and/or security patches to the Dependency Software version, AWS will address those updates and bug fixes by including them in a new Neuron SDK version with an increment in the Neuron SDK minor version. There is no minimum support period for a Dependency Software version.

- **Maintenance and/or End-of-Support Announcement (Phase 2)**
  
  AWS will announce the Maintenance phase or the End-of-Support phase of Dependency Software version.

  Since each community or vendor maintains their own versioning policy and publishes their own end-of-support schedule for their product, there is no minimum duration to do the announcement before Dependency Software version enters Maintenance phase or End-of-Support phase and in some cases the announcement can happen at the same time when the Dependency Software version enters Maintenance phase or End-of-Support phase.

  During this period, the Dependency Software version will continue to be supported.

- **Maintenance (Phase 3)**
  
  During the maintenance phase, AWS limits Dependency Software version to address critical bug fixes and security issues only. There is no minimum Maintenance period.

  This phase is optional and AWS will reserve the right to skip it for specific Dependency Software products.

- **End-of-Support (Phase 4)**
  
  When a Dependency Software version reaches end-of-support, it will no longer receive updates or releases. Previously published releases will continue to be available via Conda (up to Neuron 1.13.0) or Pip packages.

  Use of an SDK which has reached end-of-support is done at the user’s discretion. We recommend users to upgrade to the new major version.

  When a Dependency Software version reaches end-of-support, it will no longer receive critical bugfixes and security patches. Previously published Dependency Software versions will continue to be available via Neuron SDK Conda (up to Neuron 1.13.0) or Pip packages.

  Use of a Dependency Software version which has reached end-of-support is done at the user’s discretion. We recommend users to upgrade to the latest Neuron SDK version that include the latest Dependency Software versions.

---

**Note:** AWS reserves the right to stop support for an underlying dependency without a maintenance phase.

19.2.6 Communication

Maintenance and End-Of-Support announcements are communicated as follows:

- Neuron SDK documentation.

To see the list of available Neuron SDK versions and supported Dependency Software versions see Release Details and What’s New in latest Neuron version.
19.3 Software maintenance mode

- Neuron Runtime 1.x (neuron-rtd) enters maintenance mode
- Neuron support for Apache MXNet 1.5 enters maintenance mode
- neuron-cli enters maintenance mode

19.3.1 Neuron Runtime 1.x (neuron-rtd) enters maintenance mode

10/27/2021 - Starting with Neuron 1.16.0 release, Neuron Runtime 1.x (neuron-rtd) is entering maintenance mode and replaced with Neuron Runtime 2.x, a shared library named libnrt.so. Future releases of Neuron Runtime 1.x (neuron-rtd) will address critical bug fixes and security issues only. Previous releases of Neuron Runtime 1.x (neuron-rtd) will continue to be available via rpm and deb packages.

For more information please see:
- Introducing Neuron Runtime 2.x (libnrt.so)
- Setup Guide
- SDK Maintenance Policy

19.3.2 Neuron support for Apache MXNet 1.5 enters maintenance mode

10/27/2021 - Starting Neuron release 1.16.0, Neuron support for MXNet 1.5 is entering maintenance mode. Future releases of Neuron supporting MXNet 1.5 will address critical bug fixes and security issues only. Previous releases of Apache MXNet 1.5 will continue to be available via pip packages.

Current users of Neuron MXNet 1.5 can migrate their applications to Neuron MXNet 1.8, for more information about Neuron MXNet support and how to upgrade to latest Neuron MXNet 1.8, please see visit Neuron Apache MXNet (Incubating).

19.3.3 neuron-cli enters maintenance mode

10/27/2021 - Starting Neuron release 1.16.0, with the introduction of Neuron Runtime 2.x, neuron-cli is entering maintenance mode. neuron-cli functionality will be available only if Neuron Runtime 1.x (neuron-rtd) is being used by the application. If the application is using Neuron Runtime 2.x shared library (libnrt.so), neuron-cli functionality will not be available.

If you have used neuron-cli in previous releases, and you are migrating to newer Neuron releases where applications require Neuron Runtime 2.x shared library, please see the below Frequently Asked questions (FAQ). Future releases of neuron-cli will address critical bug fixes and security issues only. Previous releases of neuron-cli will continue to be available via rpm and deb packages.
Frequently Asked questions (FAQ)

Is there another tool that provide the same functionality as `neuron-cli list-model`?


Is there another tool that provide the same functionality as `neuron-cli create-ncg`, `neuron-cli destroy-ncg`, and `neuron-cli list-ncg`?

No, these functionalities are no longer needed with Neuron Runtime 2.x. NeuronCore Groups (NCG) is deprecated and NEURONCORE_GROUP_SIZES environment variable is in the process of being deprecated. Please start using NEURON_RT_VISIBLE_CORES instead. See Neuron Runtime Configuration and Migrate your application to Neuron Runtime 2.x (libnrt.so) for more information.

Is there another tool that provide the same functionality as `neuron-cli reset`?

No, this functionality is no longer needed with Neuron Runtime 2.x. Before introducing libnrt.so, in certain cases after an application crashed models had to be unloaded manually by calling neuron-cli reset.

With libnrt.so, applications runs in the context of the libnrt.so shared library and when an application exits the Neuron driver will free all resources associated with the application.

For more information please see:

- Introducing Neuron Runtime 2.x (libnrt.so)
- Neuron Tools
- Setup Guide
- SDK Maintenance Policy

19.4 Software deprecation

- End of support for NeuronCore Groups (NCG)
- Announcing end of support for NEURONCORE_GROUP_SIZES
- End of support for Neuron Conda packages in Deep Learning AMI starting Neuron 1.14.0
- End of support for Ubuntu 16 starting Neuron 1.14.0
- End of support for classic TensorBoard-Neuron starting Neuron 1.13.0 and introducing Neuron Plugin for TensorBoard
- End of support for Python 3.5
- End of support for ONNX
- End of support for PyTorch 1.3
19.4.1 End of support for NeuronCore Groups (NCG)

10/27/2021 - Before the introduction of Neuron Runtime 2.x, NeuronCore Group (NCG) has been used by Neuron Runtime 1.x to define an execution group of one or more NeuronCores where models can be loaded and executed. It also provided separation between processes.

With the introduction of Neuron Runtime 2.x, the strict separation of NeuronCores into groups is no longer needed and NeuronCore Groups (NCG) is deprecated. Neuron Runtime 2.x enables each process to own a set of NeuronCores, and within each process, Neuron Runtime 2.x supports loading and executing multiple models on separate, different or overlapping sets of NeuronCores.

Please note that NEURONCORE_GROUP_SIZES environment variable is in the process of being deprecated, and for a transition period NEURONCORE_GROUP_SIZES can be used to preserve the old NeuronCore Group behavior. The frameworks internally would convert NEURONCORE_GROUP_SIZES to use runtime’s new mode of mapping models to NeuronCores.

For more information see details about NEURON_RT_VISIBLE_CORES at Neuron Runtime Configuration and Migrate your application to Neuron Runtime 2.x (libnrt.so).

19.4.2 Announcing end of support for NEURONCORE_GROUP_SIZES

10/27/2021 - NEURONCORE_GROUP_SIZES environment variable is in the process of being deprecated, future Neuron releases may no longer support the NEURONCORE_GROUP_SIZES environment variable. Please start using NEURON_RT_VISIBLE_CORES instead.

See End of support for NeuronCore Groups (NCG), Neuron Runtime Configuration and Migrate your application to Neuron Runtime 2.x (libnrt.so) for more information.

19.4.3 End of support for Neuron Conda packages in Deep Learning AMI starting Neuron 1.14.0

05/28/2021 - Starting with Neuron SDK 1.14.0, we will no longer support conda packages to install Neuron SDK framework in DLAMI and we will no longer update conda packages used to install Neuron SDK framework (Neuron conda packages) with new versions.

Starting with Neuron SDK 1.14.0, pip packages (Neuron pip packages) will be used to install Neuron SDK framework in DLAMI conda environment. To upgrade Neuron SDK framework DLAMI users should use pip upgrade commands instead of conda update commands. Instructions are available in this blog and in Neuron SDK documentation (https://awsdocs-neuron.readthedocs-hosted.com/en/latest/neuron-intro/neuron-install-guide.html#deep-learning-ami-dlami).

Starting with Neuron SDK 1.14.0, run one of the following commands to upgrade to latest Neuron framework of your choice:

- To upgrade Neuron PyTorch:

```
source activate aws_neuron_pytorch_p36
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
pip install --upgrade torch-neuron neuron-cc[tensorflow] torchvision
```

- To upgrade Neuron TensorFlow:

```
source activate aws_neuron_tensorflow_p36
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
pip install --upgrade tensorflow-neuron tensorboard-neuron neuron-cc
```
To upgrade Neuron MXNet:

```bash
source activate aws_neuron_mxnet_p36
pip config set global.extra-index-url https://pip.repos.neuron.amazonaws.com
pip install --upgrade mxnet-neuron neuron-cc
```

For more information please check the blog.

### 19.4.4 End of support for Ubuntu 16 starting Neuron 1.14.0

05/01/2021 - Ubuntu 16.04 entered end of life phase officially in April 2021 (see https://ubuntu.com/about/release-cycle) and will not receive any public software or security updates. Starting with Neuron SDK 1.14.0, Ubuntu 16 is no longer supported for Neuron, users who are using Ubuntu 16 are requested to migrate to Ubuntu18 or Amazon Linux 2.

Customers who choose to upgrade libc on Ubuntu 16 to work with Neuron v1.13.0 (or higher versions) are highly discouraged from doing that since Ubuntu 16 will no longer receive public security updates.

### 19.4.5 End of support for classic TensorBoard-Neuron starting Neuron 1.13.0 and introducing Neuron Plugin for TensorBoard

05/01/2021 - Starting with Neuron SDK 1.13.0, we are introducing Neuron Plugin for TensorBoard and we will no longer support classic TensorBoard-Neuron. Users are required to migrate to Neuron Plugin for TensorBoard.

Starting with Neuron SDK 1.13.0, if you are using TensorFlow-Neuron within DLAMI Conda environment, attempting to run tensorboard with the existing version of TensorBoard will fail. Please update the TensorBoard version before installing the Neuron plugin by running `pip install TensorBoard --force-reinstall`, for installation instructions see Neuron Plugin for TensorBoard.

Users who are using Neuron SDK releases before 1.13.0, can find classic TensorBoard-Neuron documentation at Neuron 1.12.2 documentation.

For more information see Neuron Plugin for TensorBoard Release Notes and Neuron Plugin for TensorBoard.

### 19.4.6 End of support for Python 3.5

2/24/2021 - As Python 3.5 reached end-of-life in October 2020, and many packages including TorchVision and Transformers have stopped support for Python 3.5, we will begin to stop supporting Python 3.5 for frameworks, starting with PyTorch-Neuron version [1.1.7.0] in this release. You can continue to use older versions with Python 3.5.

### 19.4.7 End of support for ONNX

11/17/2020 - ONNX support is limited and from this version onwards we are not planning to add any additional capabilities to ONNX. We recommend running models in TensorFlow, PyTorch or MXNet for best performance and support.
19.4.8 End of support for PyTorch 1.3

7/16/2020 - Starting this release we are ending the support of PyTorch 1.3 and migrating to PyTorch 1.5.1, customers are advised to migrate to PyTorch 1.5.1.

19.5 Contact Us

For support please checkout the Github issues or Neuron AWS forums for an answer, if none of those resources have an answer to your question please open a ticket.

If you have an urgent need for a feature you can also contact us directly at aws-neuron-support@amazon.com.
INDEX

B

built-in function
torch.neuron.DataParallel(), 382
torch.neuron.DataParallel.disable_dynamic_batching(), 383
torch_neuron.trace(), 377

F

FLOAT32_TO_FLOAT16 (torch_neuron.Optimization attribute), 378

N

neuron-cc command line option
neuron-cc compile [parameters], 1196
neuron-cc list-operators [parameters], 1199
neuron-cc [options] <command> [parameters], 1196
neuron-cc compile [parameters]
neuron-cc command line option, 1196
neuron-cc list-operators [parameters]
neuron-cc command line option, 1199
neuron-cc [options] <command> [parameters]
neuron-cc command line option, 1196
neuron-ls command line option
neuron-ls [options], 1273
neuron-ls [options]
neuron-ls command line option, 1273
neuron-monitor command line option
neuron-monitor [parameters], 1261
neuron-monitor [parameters]
neuron-monitor command line option, 1261

T

torch.neuron.DataParallel()
built-in function, 382
torch.neuron.DataParallel.disable_dynamic_batching()
built-in function, 383
torch_neuron.Optimization (built-in class), 378
torch_neuron.trace()