NAIRR Pilot National Artificial Intelligence Research Resource Pilot

Introduction to TensorFlow

Mahidhar Tatineni

April 2, 2023

AI Workshop Denver, CO April 2-3, 2025

Overview

- **Overview of TensorFlow**
  - Introduction
  - Features
- **Applications of TensorFlow**
- **Typical training setup for deep learning**
  - Introduction to Tensors
  - Loading and preprocessing data
  - Building a model
  - Compiling a model and setting up a training loop
- **Accessing TensorFlow on ACCESS and NAIRR resources**
  - ACCESS/NAIRR resources – e.g. Delta AI, Expanse, Bridges-2, Jetstream2
  - Cloud resources – direct or via Cloudbank – AWS, AZURE, Google
  - Conda installs, Containers
- **Hands on examples**
  - Containerized setup
    - Jupyter example on Expanse using customized container
    - Batch job example on Delta/Delta AI using NGC container
  - Scaling up using horovod, tfdist – example on Expanse

Overview

- **Overview of TensorFlow**
  - Introduction
  - Features
- **Applications of TensorFlow**
- **Typical training setup for deep learning**
  - Introduction to Tensors
  - Loading and preprocessing data
  - Building a model
  - Compiling a model and setting up a training loop
- **Accessing TensorFlow on ACCESS and NAIRR resources**
  - ACCESS/NAIRR resources – e.g. Delta AI, Expanse, Bridges-2, Jetstream2
  - Cloud resources – direct or via Cloudbank – AWS, AZURE, Google
  - Conda installs, Containers
- **Hands on examples**
  - Containerized setup
    - Jupyter example on Expanse using customized container
    - Batch job example on Delta/Delta AI using NGC container
  - Scaling up using horovod, tfdist – example on Expanse

TensorFlow – Introduction

- Open-source end-to-end machine learning platform originally developed by Google Brain team
- Flexible architecture that allows machine learning algorithms to be described as a graph of connected operations.
- The framework backend can be adapted to many different hardware architectures => can be trained and executed on GPUs, CPUs, and TPUs across various platforms without rewriting code, ranging from portable devices to desktops to high-end servers.
- Tensor data input to framework. Primary approach is to build a computational graph that defines the dataflow for training.

Today's tutorial is based on the TensorFlow guide and tutorials. We are limited in time so we are looking at basics and will do simple hands-on training examples. Following links have a lot more detail. Once we know how to run things interactively using Jupyter and via batch scripts, attendees can try out most of the material below on one of the machines:
https://www.tensorflow.org/guide
https://www.tensorflow.org/tutorials

TensorFlow – Features

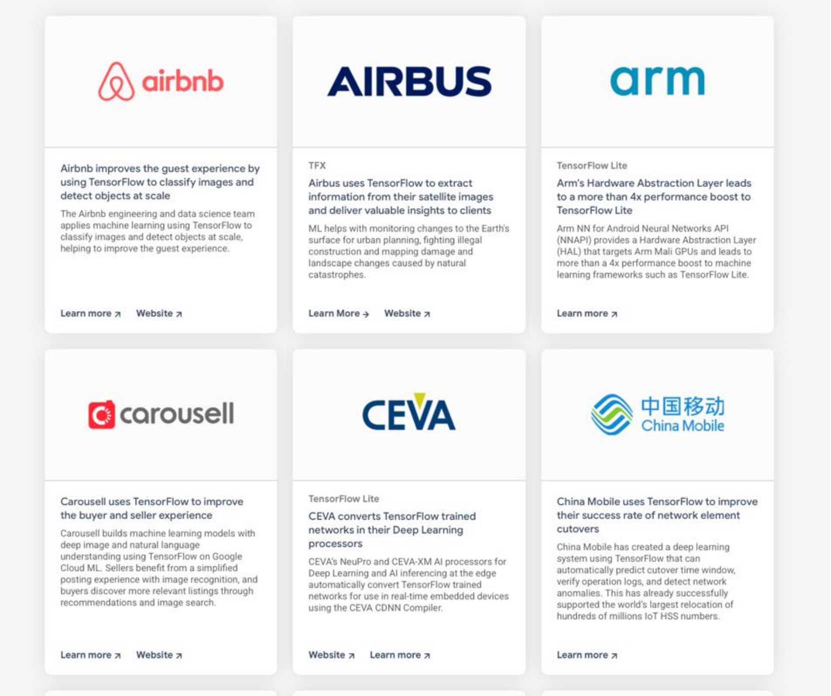
- Multidimensional-array based numeric computation
- Automatic differentiation
- Model construction, training, and export
- High level Keras API in python
    - Originally a standalone package, integrated into TensorFlow now, now back outside and also supports PyTorch as backend
    - Covers typical machine learning workflow components - data processing, hyperparameter tuning, and deployment.
    - Enables fast experimentation, can use CPUs, GPUs, TPUs or other specialized hardware
- Tensor data input to framework. Primary approach is to build a computational graph that defines the dataflow for training.
- Can also use eager execution model
- TensorBoard unified visualization framework (TensorFlow, Keras) allows monitoring, visualization of computational graphs, and debugging

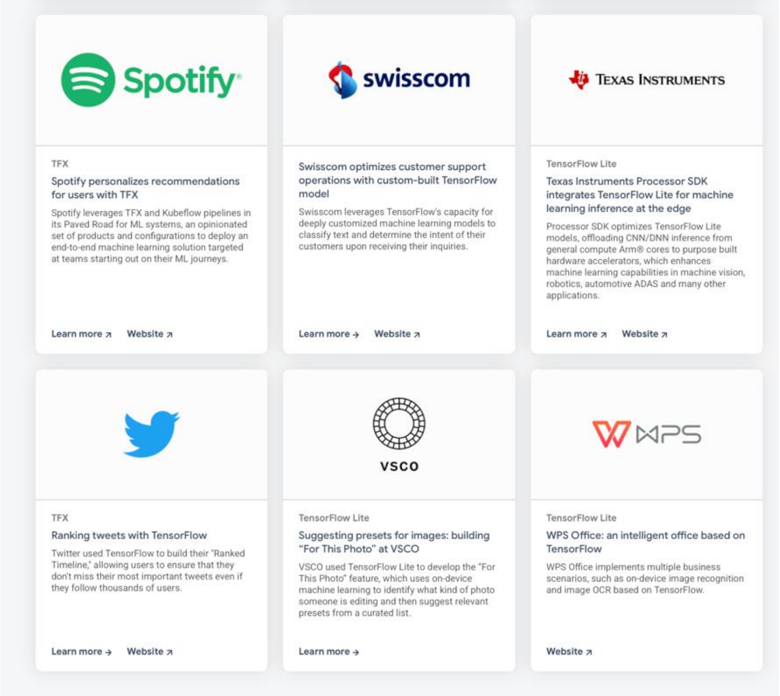
Overview

- **Overview of TensorFlow**
    - Introduction
    - Features
- **Applications of TensorFlow**
- **Typical training setup for deep learning**
    - Introduction to Tensors
    - Loading and preprocessing data
    - Building a model
    - Compiling a model and setting up a training loop
- **Accessing TensorFlow on ACCESS and NAIRR resources**
    - ACCESS/NAIRR resources – e.g. Delta AI, Expanse, Bridges-2, Jetstream2
    - Cloud resources – direct or via Cloudbank – AWS, AZURE, Google
    - Conda installs, Containers
- **Hands on examples**
    - Containerized setup
        - Jupyter example on Expanse using customized container
        - Batch job example on Delta/Delta AI using NGC container
    - Scaling up using horovod, tfdist – example on Expanse

TensorFlow Applications -

- Several application areas - Image processing and video detection, Time series algorithms, Modeling. Domains include health care, security, nuclear fusion, social sciences, climate science.
- Several commercial case studies on TensorFlow site: https://www.tensorflow.org/about/case-studies





Overview

- **Overview of TensorFlow**
  - Introduction
  - Features
- **Applications of TensorFlow**
- **Typical training setup for deep learning**
  - Introduction to Tensors
  - Loading and preprocessing data
  - Building a model
  - Compiling a model and setting up a training loop
- **Accessing TensorFlow on ACCESS and NAIRR resources**
  - ACCESS/NAIRR resources – e.g. Delta AI, Expanse, Bridges-2, Jetstream2
  - Cloud resources – direct or via Cloudbank – AWS, AZURE, Google
  - Conda installs, Containers
- **Hands on examples**
  - Containerized setup
    - Jupyter example on Expanse using customized container
    - Batch job example on Delta/Delta AI using NGC container
  - Scaling up using horovod, tfdist – example on Expanse

Introduction to Tensors

- Tensors are multi-dimensional arrays with a uniform type (called a dtype). You can see all supported dtypes at https://www.tensorflow.org/api_docs/python/tf/dtypes.
- Tensors shapes:
  - Shape: The length (number of elements) of each of the axes of a tensor.
  - Rank: Number of tensor axes. A scalar has rank 0, a vector has rank 1, a matrix is rank 2.
  - Axis or Dimension: A particular dimension of a tensor.
  - Size: The total number of items in the tensor, the product of the shape vector's elements.
  - Can have variable numbers of elements along some axis - Ragged Tensors
- TensorFlow follows standard Python indexing rules
  - indexes start at 0
  - negative indices count backwards from the end
  - colons, :, are used for slices: start:stop:step
  - Higher rank tensors are indexed by passing multiple indices
- Tensor shapes can be manipulated
  - tf.reshape function can be used to reshape tensors
  - The data layout in memory stays the same and the new tensor is created with the requested shape
  - Broadcasting happens automatically; Can use tf.broadcast_to function too.

Basic tensor creation examples (from TensorFlow guide)

```
import tensorflow as tf
import numpy as np

rank_0_tensor = tf.constant(4)
print(rank_0_tensor)


rank_2_tensor = tf.constant([[1, 2], [3, 4], [5, 6]],
dtype=tf.float16)
print(rank_2_tensor)


np.array(rank_2_tensor)
rank_2_tensor.numpy()
```

```
>>> rank_0_tensor = tf.constant(4)
>>> print(rank_0_tensor)
tf.Tensor(4, shape=(), dtype=int32)

>>> rank_2_tensor = tf.constant([[1, 2], [3, 4], [5, 6]],
dtype=tf.float16)
>>> print(rank_2_tensor)
tf.Tensor(
[[1. 2.]
 [3. 4.]
 [5. 6.]], shape=(3, 2), dtype=float16)

>>> np.array(rank_2_tensor)
array([[1., 2.],
      [3., 4.],
      [5., 6.]], dtype=float16)
```

Basic tensor math (from TensorFlow guide)

```
a = tf.constant([[1, 2], [3, 4]])
b = tf.ones([2,2], dtype=tf.int32)
print(tf.add(a, b), "\n")

print(tf.multiply(a, b), "\n")

print(tf.matmul(a, b), "\n")
```

```
>>> a = tf.constant([[1, 2],
...                  [3, 4]])
>>> b = tf.ones([2,2], dtype=tf.int32)
>>> print(tf.add(a, b), "\n")
tf.Tensor(
[[2 3]
 [4 5]], shape=(2, 2), dtype=int32)

>>> print(tf.multiply(a, b), "\n")
tf.Tensor(
[[1 2]
 [3 4]], shape=(2, 2), dtype=int32)

>>> print(tf.matmul(a, b), "\n")
tf.Tensor(
[[3 3]
 [7 7]], shape=(2, 2), dtype=int32)
```

Reshaping Tensors

```
x = tf.constant([[1], [2], [3]])
print(x.shape)

reshaped = tf.reshape(x, [1, 3])
print(x.shape)
print(reshaped.shape)

rank_3_tensor = tf.constant([ [[0, 1, 2, 3, 4], [5, 6, 7, 8, 9]], [[10, 11, 12, 13, 14], [15, 16, 17, 18, 19]], [[20, 21, 22, 23, 24], [25, 26, 27, 28, 29]],])
print(tf.reshape(rank_3_tensor, [-1]))
```

```
>>> x = tf.constant([[1], [2], [3]])
2025-03-25 06:19:25.334425: E external/local_xla/xla/stream_executor/cuda/cuda_driver.cc:274] failed call to cuInit: CUDA_ERROR_NO_DEVICE: no CUDA-capable device is detected
>>> print(x.shape)
(3, 1)

>>> reshaped = tf.reshape(x, [1, 3])
>>> print(x.shape)
(3, 1)
>>> print(reshaped.shape)
(1, 3)

>>> print(tf.reshape(rank_3_tensor, [-1]))
tf.Tensor(
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
 22 23 24 25 26 27 28 29], shape=(30,), dtype=int32)
```

TensorFlow Introduction – Loading and preprocessing data

- Keras has high level utilities to read and preprocess images from directories
  - tf.keras.utils.image_dataset_from_directory

```
train_ds = tf.keras.utils.image_dataset_from_directory( data_dir, validation_split=0.2, subset="training", seed=123, image_size=(img_height, img_width), batch_size=batch_size)
```

  - tf.keras.layers.Rescaling

- tf.data API enables build of complex input pipelines
  - Specify data source – from memory or from directories
  - Split data into training/validation sets
  - Optimize for performance – shuffled batched data
  - https://www.tensorflow.org/guide/data_performance

- Catalog of TensorFlow datasets already available. Set up as tf.data.Datasets and provide high performance input pipelines

```
(train_dataset, val_dataset, test_dataset), metadata = tensorflow_datasets.load( 'tf_flowers', split=['train[:80%]', 'train[80%:90%]', 'train[90%:]'], with_info=True, as_supervised=True, )
```

TensorFlow Introduction – Building models

- Fundamental abstraction is a layer – takes tensor inputs, does computations on them, and provides output for next layer
- Models group layers together – used in a training loop
- Defining models
  - Sequential API – linear stack of layer

    ```
    model = keras.Sequential(name="my_sequential")
    model.add(layers.Dense(2, activation="relu", name="layer1"))
    model.add(layers.Dense(3, activation="relu", name="layer2"))
    model.add(layers.Dense(4, name="layer3"))
    ```
  - Functional API - non-linear topology, shared layers, multiple inputs/outputs
  - Model subclassing – fully customizable layers, custom python code for model architecture
- Calculate predictions
  - For example: `predictions = model(x_train[:1]).numpy()`
- Loss function definition
  - For example: `loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)`

TensorFlow Introduction – Compiling models and training

- Compile model – choose an optimizer, use loss function
  `model.compile(optimizer='adam', loss=loss_fn, metrics=['accuracy’])`

- Train model. For example, run for 10 epochs:
  `model.fit(x_train, y_train, epochs=10)`

- Evaluate
  `model.evaluate(x_test, y_test, verbose=2)`

- More detailed example:
  https://www.tensorflow.org/tutorials/quickstart/advanced

Saving TensorFlow Model - Checkpoints

- Checkpoints capture the exact value of all parameters (tf.Variable objects) used by a model
- Only useful when source code that will use the saved parameter values is available
- Example from https://www.tensorflow.org/guide/checkpoint
- Create checkpoint objects
  - tf.train.Checkpoint object to manually create a checkpoint
  - tf.train.CheckpointManager to manage multiple checkpoint

```
opt = tf.keras.optimizers.Adam(0.1)
dataset = toy_dataset()
iterator = iter(dataset)
ckpt = tf.train.Checkpoint(step=tf.Variable(1), optimizer=opt, net=net, iterator=iterator)
manager = tf.train.CheckpointManager(ckpt, './tf_ckpts', max_to_keep=3)
```

- Checkpoint: save_path = manager.save()
- Restore: ckpt.restore(manager.latest_checkpoint)

Saving TensorFlow Model – SavedModel Format

- A SavedModel contains a complete TensorFlow program
  - trained parameters (tf.Variable)
  - computation

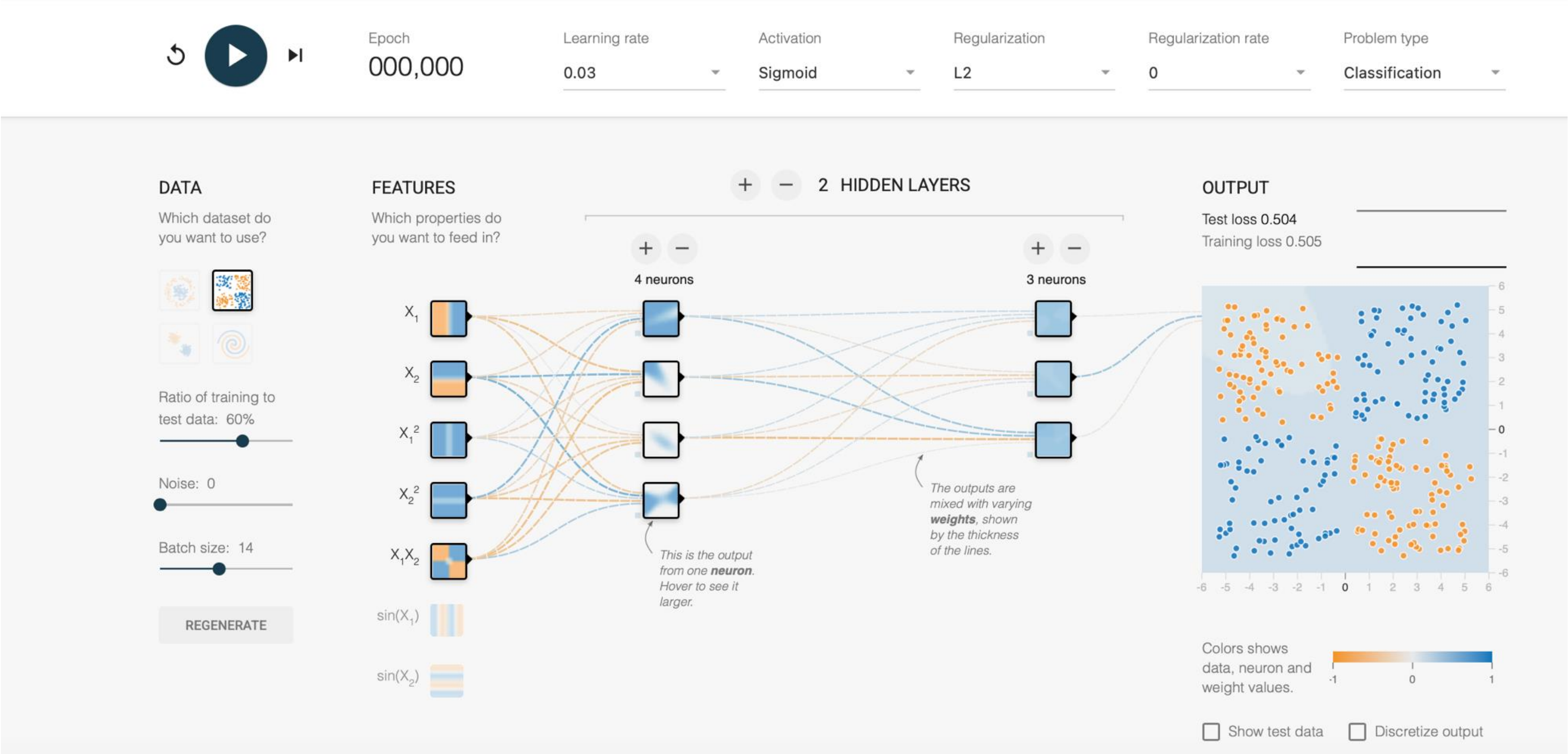
  ⇒ does not require the original model building code to run

- SavedModel API
  - Save: tf.saved_model.save(model, path_to_dir)
  - Load: model = tf.saved_model.load(path_to_dir)

- See full example at: https://www.tensorflow.org/guide/saved_model

TensorFlow – Explore Components!

- TensorFlow playground site is a nice place to explore components.
  https://playground.tensorflow.org



Scaling up TensorFlow for multi-node runs

- Two different approaches: 1) tf.distribute 2) horovod (MPI based)

- tf.distribute.Strategy API
  - distribute training across multiple GPUs, multiple machines, or TPUs

- tf.distribute.MirroredStrategy
  - All the model's variables copied to each processor
  - Combine the gradients from all processors using all-reduce
  - Update values on all processors
  - Synchronous training using multiple GPUs on single node

- tf.distribute.MultiWorkerMirroredStrategy
  - Similar to tf.distribute.MirroredStrategy
  - Synchronous training on many GPUs on multiple workers

- Slurm compatible
  - tf.distribute.cluster_resolver.SlurmClusterResolver(jobs=None, port_base=8888, gpus_per_node=None, gpus_per_task=None, tasks_per_node=None, auto_set_gpu=True, rpc_layer='grpc')

Scaling up TensorFlow for multi-node runs - Horovod

- Horovod is a distributed deep learning training framework for TensorFlow, Keras, PyTorch, and Apache MXNet

- Developed at Uber to take a single-GPU training script and scale it to train across many GPUs in parallel leveraging message passing interface (MPI) libraries. HPC sites already have optimal MPI installs so it's a good fit. Can also leverage NCCL.

- Typical modifications needed in your code
  - hvd.init() – initialization (MPI_INIT)
  - Code to pin to individual GPUs
  - Scale the learning rate by the number of workers.
  - Use hvd.DistributedOptimizer - averages gradients using **allreduce** or **allgather** and then applies those averaged gradients.
  - Broadcast initial variables
  - Save checkpoints only on worker 0

- Can leverage the MPI optimizations for the collective operations being used. Can use horovodrun or mpirun to launch the jobs. We will use mpirun in the hands-on example.

TensorBoard framework: Visualize computational graphs, debug tool

- Browser based tool
  - We cannot run a browser directly on most HPC machines
  - Use port forwarding and run the browser on client

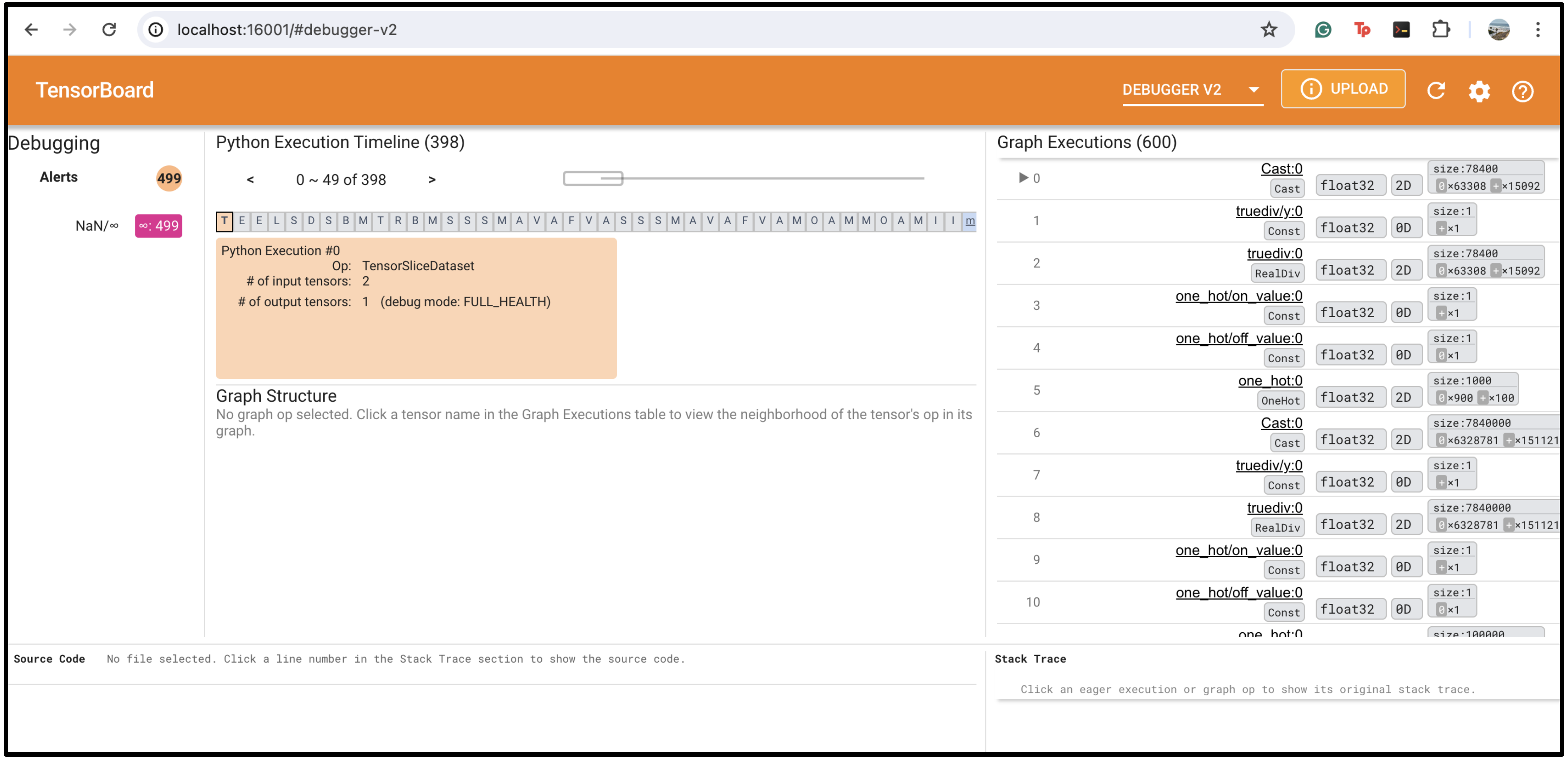
- Multistep process
  - Get an interactive node on the HPC system
  - Run your TensorFlow example and Tensorboard in an interactive shell
  - Port forward (from the compute node to your local machine/laptop where you run the browser)
  - For example, on Expanse:

    ssh -L 16001:localhost:6006 username@exp-XX-YY.expanse.sdsc.edu
  - Laptop/desktop client browser then opens: http://localhost:16001

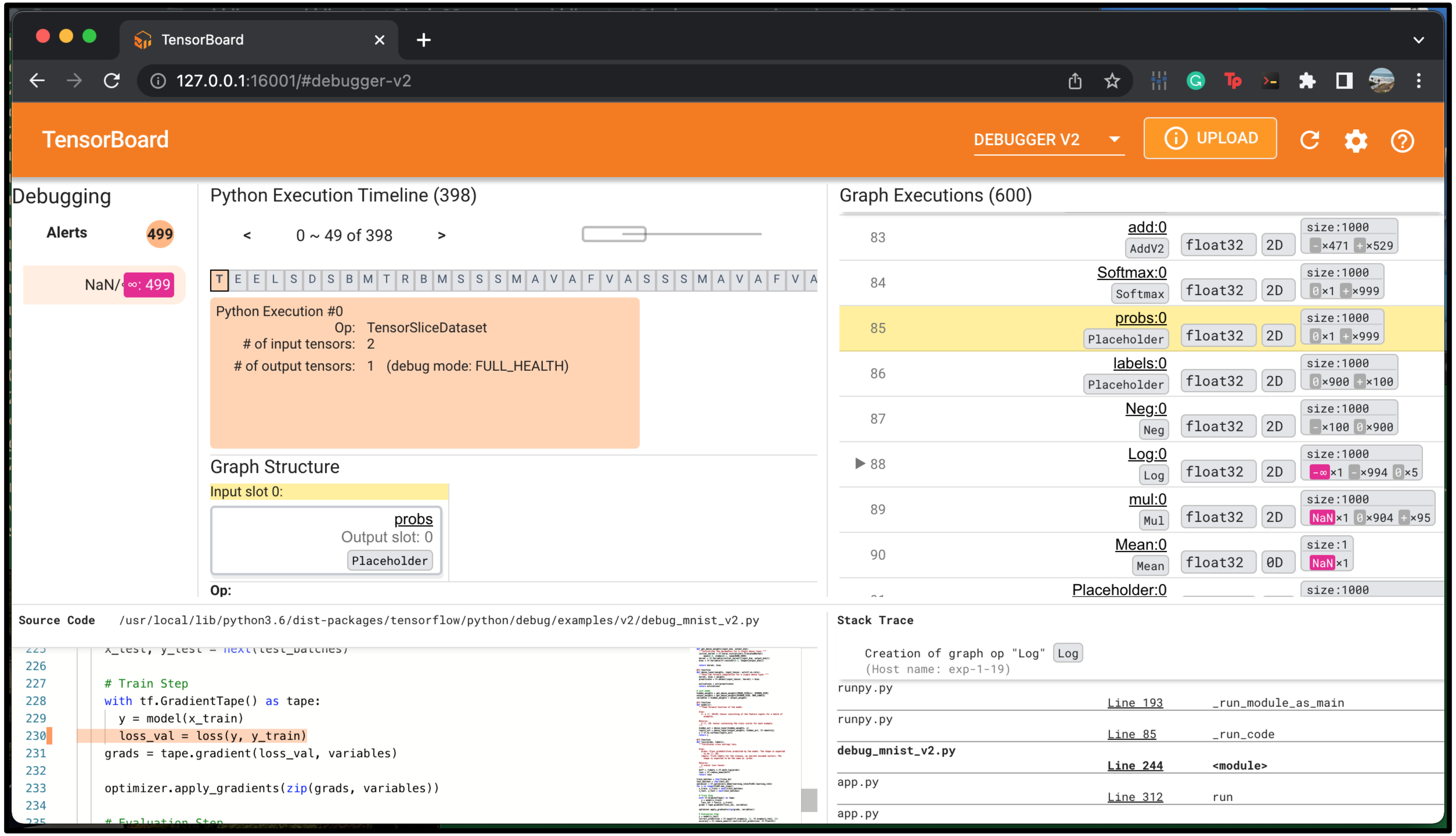
- Some snapshots in the next few slides illustrating the debug process. Example from:

  https://www.tensorflow.org/tensorboard/debugger_v2

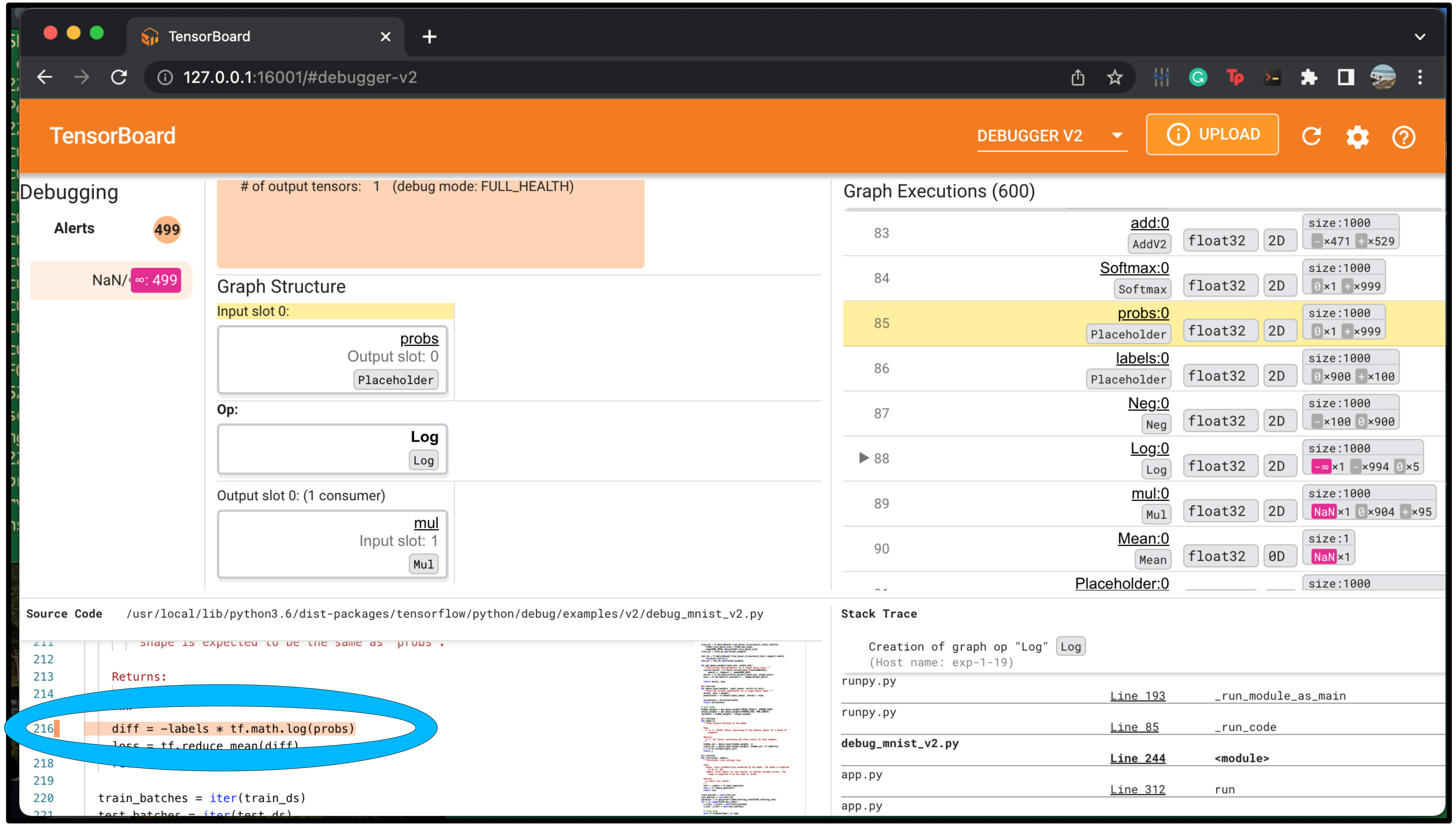
TensorBoard for debugging problems



TensorBoard for debugging problems



TensorBoard for debugging problems



Overview

- **Overview of TensorFlow**
  - Introduction
  - Features
- **Applications of TensorFlow**
- **Typical training setup for deep learning**
  - Introduction to Tensors
  - Loading and preprocessing data
  - Building a model
  - Compiling a model and setting up a training loop
- **Accessing TensorFlow on ACCESS and NAIRR resources**
  - ACCESS/NAIRR resources – e.g. Delta AI, Expanse, Bridges-2, Jetstream2
  - Cloud resources – direct or via Cloudbank – AWS, AZURE, Google
  - Conda installs, Containers
- **Hands on examples**
  - Containerized setup
    - Jupyter example on Expanse using customized container
    - Batch job example on Delta/Delta AI using NGC container
  - Scaling up using horovod, tfdist – example on Expanse

TensorFlow on ACCESS and NAIRR resources

- All ACCESS resources support TensorFlow either via conda installs or containers. A variety of hardware options available
  - https://allocations.access-ci.org/resources

- NAIRR resources include options from ACCESS managed resources and also several cloud vendors.
  - GPU resources
  - Custom processors like Cerebras, Gaudi, Gaudi2
  - Details at:
  https://nairrpilot.org/opportunities/allocations

TensorFlow Usage

- Two major approaches
  - Containerized installs
  - Conda based installs

- Containerized installs
  - Most HPC sites will have a mechanism to run containers – Singularity, Apptainer, Shifter, Docker. Can run on Slurm clusters. Containers are easy to use in Kubernetes clusters (e.g. like on PNRP nautilus cluster)
  - Vendors like NVIDIA, AMD, Intel provide performant containers for PyTorch and TensorFlow that are optimized for their hardware. For example, via NGC (NVIDIA), Infinity Hub (AMD). Can layer additional packages on the base containers
  - HPC sites will also have custom containers that are adapted to the driver versions and hardware at their sites. Again, good starting points to layer in additional packages.

- Conda based installs
  - https://www.anaconda.com/docs/tools/working-with-conda/applications/tensorflow
  - https://anaconda.org/conda-forge/tensorflow
  - Note of caution – Make sure the version you choose is compatible with the driver version on your system. For example, if your system has driver version supporting up to CUDA 12.6, don't install a newer version.

Overview

- **Overview of TensorFlow**
  - Introduction
  - Features
- **Applications of TensorFlow**
- **Typical training setup for deep learning**
  - Introduction to Tensors
  - Loading and preprocessing data
  - Building a model
  - Compiling a model and setting up a training loop
- **Accessing TensorFlow on ACCESS and NAIRR resources**
  - ACCESS/NAIRR resources – e.g. Delta AI, Expanse, Bridges-2, Jetstream2
  - Cloud resources – direct or via Cloudbank – AWS, AZURE, Google
  - Conda installs, Containers
- **Hands on examples**
  - Containerized setup
    - Jupyter example on Expanse using customized container
    - Batch job example on Delta/Delta AI using NGC container
  - Scaling up using horovod, tfdist – example on Expanse

Hands On Examples

Commands, instructions, and scripts in github repo for workshop

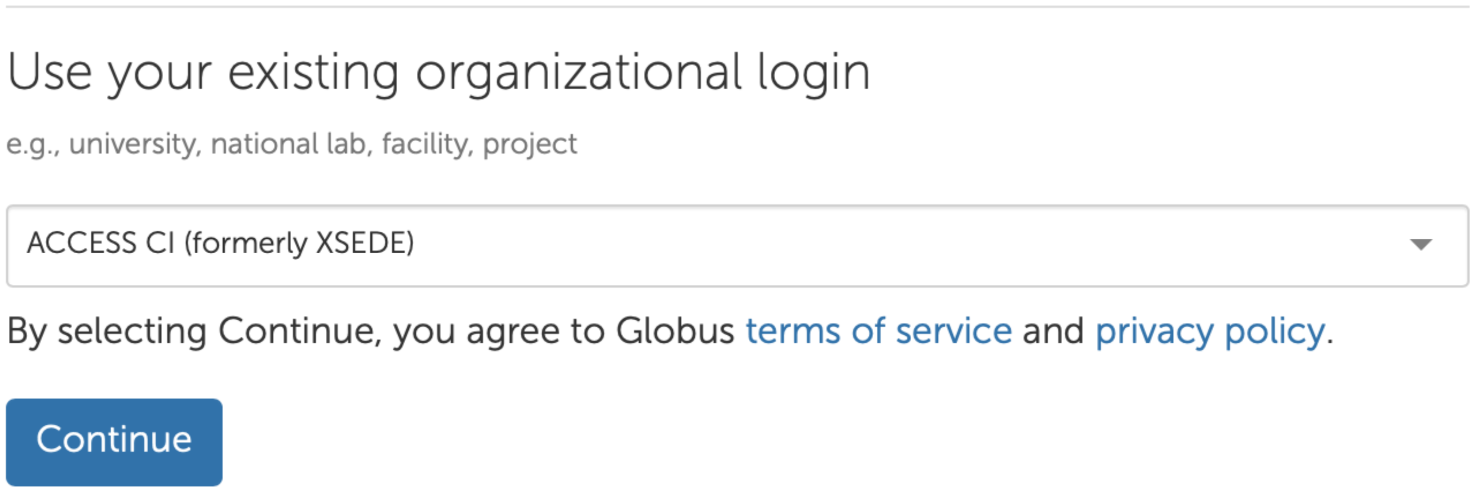
Snapshots with process details and results follow for offline information

Hands On Examples – Runs using Expanse Portal

- Expanse portal

https://portal.expanse.sdsc.edu

- Make sure you choose ACCESS CI as the organization, and then you can login with ACCESS credentials. Note: On the page after you login, do not link identies, just pick the first access ci based option as that is already linked.



Hands On Examples – Expanse portal

SDSC

The Expanse portal provides an integrated, and easy to use interface to access Expanse HPC resource.

With the portal, researchers can manage files (create, edit, move, upload, and download), view job templates for various applications, submit and monitor jobs, run interactive applications, and connect via SSH. The portal has no end-user installation requirements other than access to a modern up-to-date web browser

Pinned Apps A featured subset of all available apps

Active Jobs
System Installed App

Home Directory
System Installed App

Job Composer
System Installed App

expanse Shell Access
System Installed App

MATLAB
System Installed App

RSTUDIO
System Installed App

Allocation and Usage Information
System Installed App

Jupyter
System Installed App

Hands on examples – Expanse portal Jupyter notebook form

Jupyter Session

Account:

use300

Partition (Please choose the gpu, gpu-shared, or gpu-preempt as the partition if using gpus):

shared

Time limit (min):

30

Number of cores:

1

Memory required per node (GB):

2

GPUs (optional):

0

Singularity Image File Location: (Use your own or to include from existing container library at /cm/shared/apps/container e.g., /cm/shared/apps/containers/singularity/pytorch/pytorch-latest.sif)

/cm/shared/apps/containers/singularity/tensorflow/tensorflow-latest.sif

Environment modules to be loaded (E.g., to use latest version of system Anaconda3 include cpu,gcc,anaconda3):

singularitypro

Hands on examples – Expanse portal Jupyter status

Satellite Reverse Proxy Service

SDSC Expanse

Job State: Unknown

In Queue
    Job has not yet started.

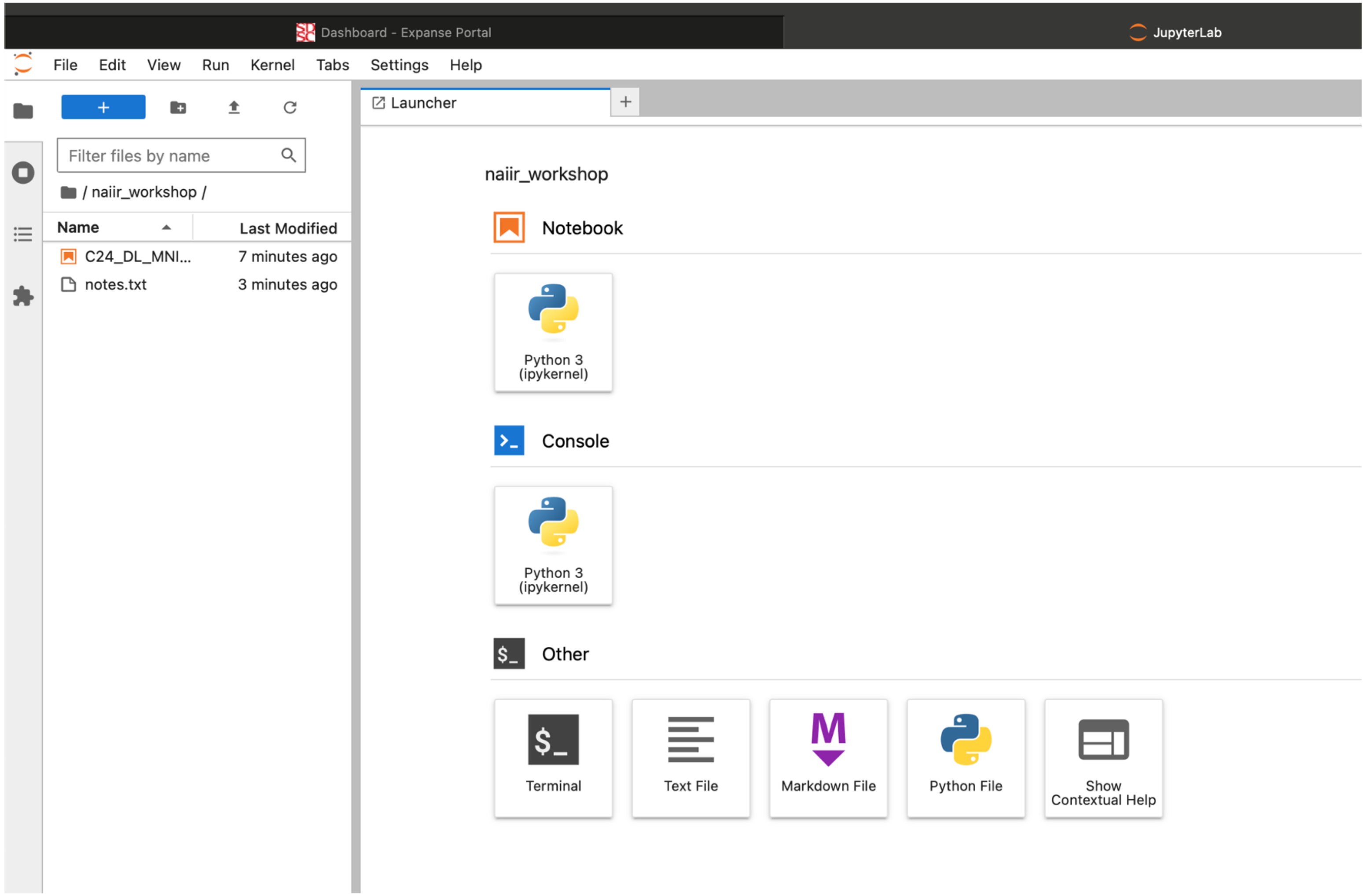
Running
    Job has started, but has not redeemed Satellite Token.

Mapped
    Job has redeemed Satellite Token, but no proxy entry exists yet.

Proxied
    Proxy entry created, ready to go!

Dead
    Job died or exited, no further progress will occur.

Hands on examples – Jupyter interface after job is mapped and proxied



Hands on examples – MNIST Example, Load Data

Reference: Paul Rodriguez talk at CIML/SDSC Summer Institutes

```
import done
```

[2]:

```
#Load MNIST data from Keras datasets
(X_train, Y_train), (X_test, Y_test) = tf.keras.datasets.mnist.load_data()

X_train=X_train[0:1000,]  #only need smaller subset to get good results
Y_train=Y_train[0:1000,]

# --------- Reshape input data, b/c Keras expects N-3D images (ie 4D matrix)
X_train = X_train[:,:,:,np.newaxis]
X_test  = X_test[:,:,:,np.newaxis]

#Scale 0 to 1  - or should we not scale
X_train = X_train/255.0
X_test  = X_test/255.0

# Convert 1-dimensional class arrays to 10-dimensional class matrices
Y_train = keras.utils.to_categorical(Y_train, 10)
Y_test  = keras.utils.to_categorical(Y_test,  10)

# ------------- End loading and preparing data --------------
print('X train shape:', X_train.shape)
print('X test shape:', X_test.shape)
```

```
X train shape: (1000, 28, 28, 1)
X test shape: (10000, 28, 28, 1)
```

Hands on examples – MNIST Example Build Model

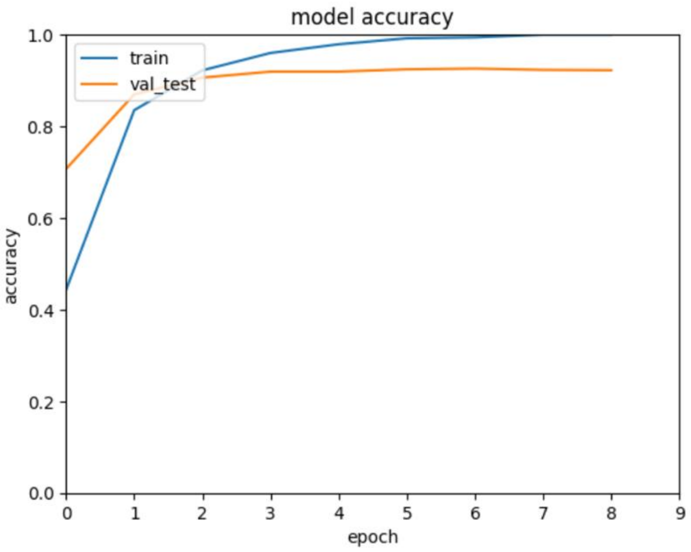
https://keras.io/api/optimizers/

[3]:
```
# --------------Set up Model ---------------------
def build_model(numfilters):
    mymodel = keras.models.Sequential()
    mymodel.add(keras.layers.Convolution2D(numfilters,      #<<<< ------ 1
                                  (3, 3),
                                  strides=1,
                                  data_format="channels_last",
                                  activation='relu',
                                  input_shape=(28,28,1)))
    mymodel.add(keras.layers.Convolution2D(numfilters,      #<<<< ------ 1
                                  (3, 3),
                                  strides=1,
                                  data_format="channels_last",
                                  activation='relu'))
    mymodel.add(keras.layers.MaxPooling2D(pool_size=(2,2),strides=2,data_format="channels_last")) #get Max over 2
    mymodel.add(keras.layers.Flatten())            #reorganize 2DxFilters output into 1D

    #---------------Now add final classification layers
    mymodel.add(keras.layers.Dense(32, activation='relu'))
    mymodel.add(keras.layers.Dense(10, activation='softmax'))

    # --------- Now configure model algorithm -----
    mymodel.compile(loss='categorical_crossentropy',
              optimizer=keras.optimizers.Adam(learning_rate=0.01),
              metrics=['accuracy'])

    return mymodel
```

Hands on examples – MNIST Example Accuracy

[6]:
```python
import matplotlib.pyplot as plt      #These provide matlab type of plotting functions
import matplotlib.image as mpimg
%matplotlib inline

# list all data in history and print out performance
print(fit_history.history.keys())
numtraining_epochs=len(fit_history.history['accuracy'])
# summarize history for accuracy
plt.figure()
plt.axis([0 ,numtraining_epochs, 0, 1])
plt.plot(fit_history.history['accuracy'])
plt.plot(fit_history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val_test'], loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```



Hands on example on NCSA Delta – Using NGC Container

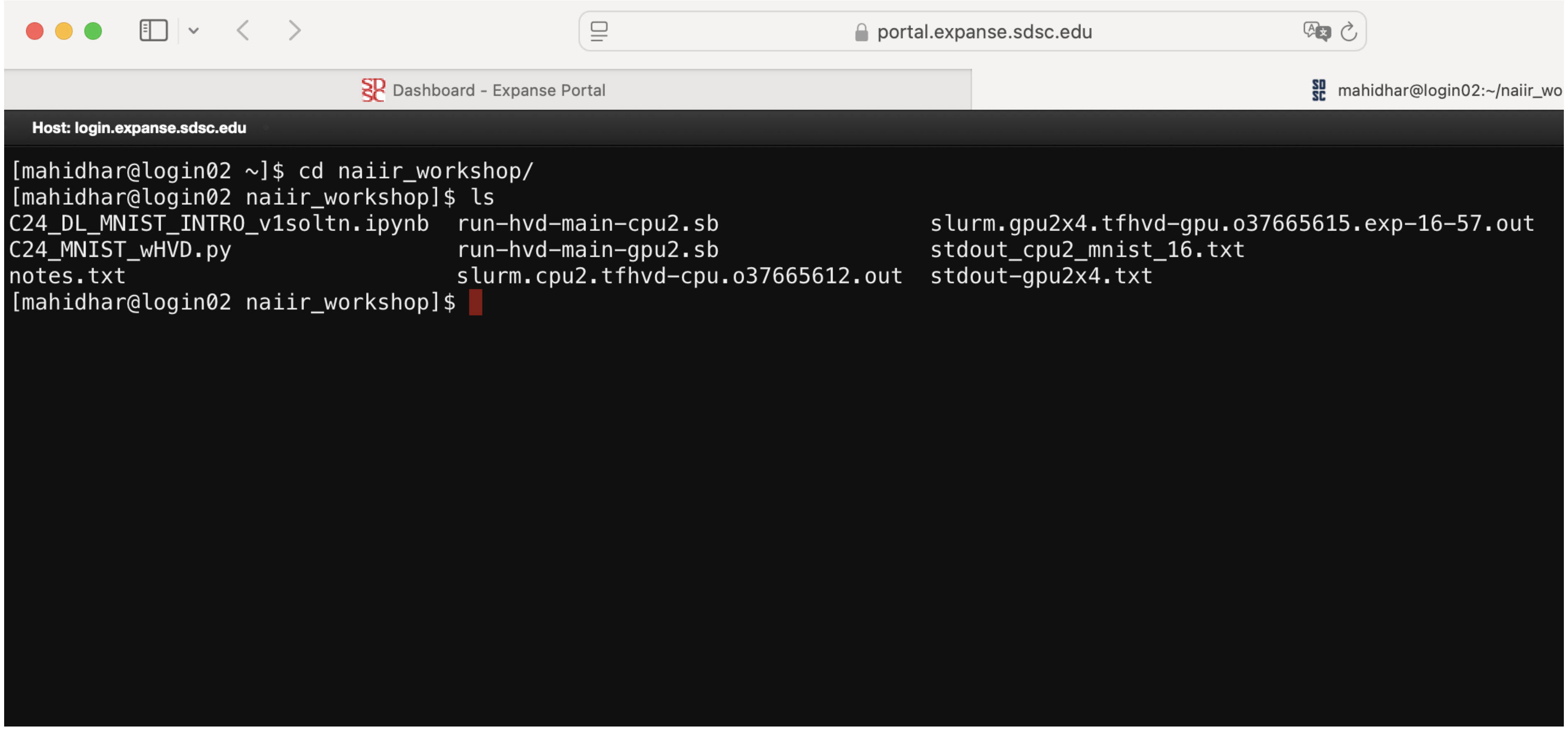
```
#!/bin/bash
#SBATCH --mem=64g
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=16
#SBATCH --partition=gpuA100x4-interactive
#SBATCH --account=beeh-delta-gpu
#SBATCH --job-name=tfNGC
### GPU options ###
#SBATCH --gpus-per-node=1
#SBATCH --gpus-per-task=1
#SBATCH --gpu-bind=verbose,per_task:1
#SBATCH -t 00:30:00

module reset # drop modules and explicitly load the ones needed
             # (good job metadata and reproducibility)
             # $WORK and $SCRATCH are now set
module list  # job documentation and metadata

echo "job is starting on `hostname`"

# run the container binary with arguments: python3 <program.py>
# --bind /projects/bbXX  # add to apptainer arguments to mount directory inside container
apptainer run --nv \
 /sw/external/NGC/tensorflow:22.06-tf2-py3 python3 \
 tensorflow_test.py
```

Hands on example on NCSA Delta – Using NGC Container

```
[mahidhar@dt-login04 nairr_workshop]$ sbatch delta-example.sb
Submitted batch job 8577459
[mahidhar@dt-login04 nairr_workshop]$ squeue -u $USER
       JOBID    PARTITION         NAME          USER ST       TIME  NODES   NODELIST(REASON) FEATURES
     8577459 gpuA100x4-in        tfNGC      mahidhar  R       0:01      1            gpua001 (null)
[mahidhar@dt-login04 nairr_workshop]$ tail slurm-8577459.out

Container image Copyright (c) 2022, NVIDIA CORPORATION & AFFILIATES. All rights reserved.
Copyright 2017-2022 The TensorFlow Authors.  All rights reserved.

Various files include modifications (c) NVIDIA CORPORATION & AFFILIATES.  All rights reserved.

This container image and its contents are governed by the NVIDIA Deep Learning Container License.
By pulling and using the container, you accept the terms and conditions of this license:
https://developer.nvidia.com/ngc/nvidia-deep-learning-container-license

[mahidhar@dt-login04 nairr_workshop]$ tail slurm-8577459.out
1875/1875 [==============================] - 2s 975us/step - loss: 0.3022 - accuracy: 0.9121
Epoch 2/5
1875/1875 [==============================] - 2s 969us/step - loss: 0.1461 - accuracy: 0.9555
Epoch 3/5
1875/1875 [==============================] - 2s 970us/step - loss: 0.1078 - accuracy: 0.9676
Epoch 4/5
1875/1875 [==============================] - 2s 972us/step - loss: 0.0882 - accuracy: 0.9730
Epoch 5/5
1875/1875 [==============================] - 2s 970us/step - loss: 0.0758 - accuracy: 0.9759
313/313 - 0s - loss: 0.0713 - accuracy: 0.9784 - 350ms/epoch - 1ms/step
[mahidhar@dt-login04 nairr_workshop]$
```

Hands on examples – Expanse multi-node runs

- Start by clicking on the “terminal” app in the portal



Hands on examples – submit script for multi-node runs using tfdidt

```
#!/usr/bin/env bash
#SBATCH --job-name=clusterresolver
#SBATCH --account=use300
#SBATCH --partition=gpu
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=40
#SBATCH --gpus=8
#SBATCH --mem=200G
#SBATCH --output=slurm.gpu2node.%x.o%j.out
#SBATCH --time=00:30:00


module load singularitypro/3.11
srun singularity exec --bind /scratch,/expanse /cm/shared/apps/containers/singularity/tensorflow/tensorflow-latest.sif python3 test.py
```

Hands on examples – Submit Expanse multi-node job using sbatch

- Submit using sbatch:

  sbatch --res=nairrworkshop run-clusterresolver-gpu2node.sb

```
[mahidhar@login01 naiir_workshop]$ sbatch --res=nairrworkshop run-clusterresolve
r-gpu2node.sb
Submitted batch job 37742419
[mahidhar@login01 naiir_workshop]$ squeue -u $USER
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REA
SON)
          37742419       gpu clusterr mahidhar PD       0:00      2 (Reservation
)
[mahidhar@login01 naiir_workshop]$
```

Hands on examples – submit script for multi-node runs

```
#!/usr/bin/env bash
#SBATCH --job-name=tfhvd-cpu
#SBATCH --account=gue998
#SBATCH --partition=compute
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8  #<<<<<------ change
this to 16 or 4 and obse
rve changes in training time (listed at end of
stdoutput file)
#SBATCH --cpus-per-task=1
#SBATCH --mem=243G
#SBATCH --time=00:15:00
#SBATCH --output=slurm.cpu2.%x.o%j.out
#SBATCH --time=00:30:00
#----------- set up modules ----------------------------
module reset
module load slurm
module load gcc/10.2.0        #compiler, unix
```

```
module load openmpi/4.1.3      #open mpi
module load singularitypro/3.11  #container
module list
#----------- set up some environmental settings --------
export OMPI_MCA_btl='self,vader'
export UCX_TLS='shm,rc,ud,dc'
export UCX_NET_DEVICES='mlx5_0:1'
export UCX_MAX_RNDV_RAILS=1
#might need to cd into the working directory
#cd /home/$USER/MNODETest
#------ execute the mpirun command to launch container
instances -----------
mpirun -n ${SLURM_NTASKS} singularity exec --bind
/expanse,/scratch /cm/s
hared/apps/containers/singularity/tensorflow/tensorflow_
22.08-tf2-py3.sif
 python3 ./C24_MNIST_wHVD.py >
stdout_cpu2_mnist_${SLURM_NTASKS}.txt
```

Hands on examples – Submit Expanse multi-node job using sbatch

- Submit using sbatch:

 sbatch run-hvd-main-cpu2.sb

```
[mahidhar@login02 naiir_workshop]$ ls -lt
total 141
-rw-r--r-- 1 mahidhar use300  1412 Mar 17 07:22 slurm.gpu2x4.tfhvd-gpu.o37665615.exp-16-57.out
-rw-r--r-- 1 mahidhar use300    34 Mar 17 07:22 stdout-gpu2x4.txt
-rw-r--r-- 1 mahidhar use300  1031 Mar 17 07:22 run-hvd-main-gpu2.sb
-rw-r--r-- 1 mahidhar use300  7169 Mar 17 07:22 stdout_cpu2_mnist_16.txt
-rw-r--r-- 1 mahidhar use300  3218 Mar 17 07:22 slurm.cpu2.tfhvd-cpu.o37665612.out
-rw-r--r-- 1 mahidhar use300  1200 Mar 17 07:19 run-hvd-main-cpu2.sb
-rw-r--r-- 1 mahidhar use300  7834 Mar 17 07:18 C24_MNIST_wHVD.py
-rw-r--r-- 1 mahidhar use300 78648 Mar 17 07:09 C24_DL_MNIST_INTRO_v1soltn.ipynb
-rw-r--r-- 1 mahidhar use300   256 Mar 17 07:01 notes.txt
[mahidhar@login02 naiir_workshop]$ sbatch run-hvd-main-cpu2.sb
Submitted batch job 37665677
[mahidhar@login02 naiir_workshop]$ squeue -u $USER
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
          37665677   compute tfhvd-cp mahidhar PD       0:00      2 (Priority)
[mahidhar@login02 naiir_workshop]$
```

Hands on examples – Submit Expanse multi-node job using sbatch

- Check output, for example:

cat slurm.cpu2.tfhvd-cpu.o37665677.out

```
[mahidhar@login02 naiir_workshop]$ cat slurm.cpu2.tfhvd-cpu.o37665677.out
Resetting modules to system default. Reseting $MODULEPATH back to system default. All extra directories will be removed from $MODULEPATH.

Currently Loaded Modules:
  1) shared             4) DefaultModules          7) ucx/1.10.1/dnpjjuc
  2) cpu/0.17.3b (c)    5) slurm/expanse/23.02.7   8) openmpi/4.1.3/oq3qvsv
  3) sdsc/1.0           6) gcc/10.2.0/npcyll4      9) singularitypro/3.11

  Where:
   c:  built natively for AMD Rome



WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0100s vs `on_train_batch_end` ti
43s). Check your callbacks.
WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0100s vs `on_train_batch_end` ti
43s). Check your callbacks.
WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0100s vs `on_train_batch_end` ti
42s). Check your callbacks.
WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0100s vs `on_train_batch_end` ti
44s). Check your callbacks.
WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0099s vs `on_train_batch_end` ti
44s). Check your callbacks.
WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0100s vs `on_train_batch_end` ti
43s). Check your callbacks.
WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0100s vs `on_train_batch_end` ti
44s). Check your callbacks.
```

References

- Tensorflow guide : https://www.tensorflow.org/guide
- Tensorflow case studies: https://www.tensorflow.org/about/case-studies
- NVIDIA GPU accelerated TensorFlow: https://catalog.ngc.nvidia.com/orgs/nvidia/containers/tensorflow
- TensorFlow tutorials: https://www.tensorflow.org/tutorials
- CIML Workshop – excellent resource for many hands-on examples https://github.com/ciml-org/ciml-summer-institute-2024