INSTALLATION AND WORKING OF DEEP LEARNING LIBRARY (TensorFlow) ON ARC

OVERVIEW

TensorFlow framework exposes high level interfaces for deep learning architecture specification, model training, tuning, and validation.

A Python virtual environment can be set-up on Arc to install and run TensorFlow. An example of using TensorFlow with MNIST dataset in the Python virtual environment on Arc is discussed further in the sections below.

INSTALLING AND RUNNING TENSORFLOW EXAMPLE ON ARC

1. **Login to Arc using your user credentials

```
localhost $ ssh -X -p 22 username@arc.utsa.edu
```

****Note(Mac Users)**: - Mac users will have to download and install XQuartz for launching GUI-based applications on remote Linux systems.

****Note(Windows Users)**: -Windows users will have to download and install Xming/Mobaxterm for launching GUI-based applications on remote Linux systems.

2. Switch to compute node for an hour using the following command

```
[username@login001]$ srun -p compute1 -N 1 -n 1 -t \
01:00:00 --pty bash
```

3. Create and activate a Python Virtual Environment, enter the following commands sequentially

```
[username@c001]$ pip install virtualenv
[username@c001]$ virtualenv mypython
[username@c001]$ source mypython/bin/activate
```

Note: To deactivate the environment, enter command "**deactivate mypython**".

4. Install tensorflow and keras libraries and test the installation by printing the tensor created

```
(mypython)[username@c001]$ pip install --upgrade \
tensorflow==2.0
(mypython)[username@c001]$ pip install --upgrade \
keras==2.3.0
(mypython)[username@c001]$ python3
Python 3.8.8 (default, Apr 13 2021, 19:58:26)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for
more information.
>>> import tensorflow as tf
>>> hello = tf.constant("Hello, Tensorflow!")
>>> print(hello)
tf.Tensor(b'Hello, Tensorflow!', shape=(),
dtype=string)
```

RUNNING TENSORFLOW EXAMPLE ON ARC

Checkpoint features can help in saving model progress during training. The model can resume training where it left off and avoid starting from scratch if something happens during the training. This mode is designed to solve the MNIST handwritten digit classification problem. The training dataset is included in the Keras package and can be loaded by calling mnist.load_data() function.

5. Load the CUDA toolkit (NVIDIA CUDA provides development environment for high performance computing) and cudnn (the NVIDIA CUDA Deep Learning Neural Network library) libraries.

```
(mypython)[username@c001]$ ml cuda/toolkit/11.3
(mypython)[username@c001]$ ml cuda/cudnn/8.2.1.32
```

6. Here is an example code showing how to implement checkpointing and restart in Tensorflow applications (`program_name.py`). In this example, the checkpoint file name is "`mymodel.h5`". This can be changed to another *.h5 file.

```
import tensorflow as tf
from tensorflow.keras.callbacks import ModelCheckpoint
import os.path
from os import path

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

filename = "mymodel.h5"

# check if checkpoint file exists. if does, load the model and
skip building the model
if (path.isfile(filename)):
```

```
    print("Resuming")
    model = tf.keras.models.load_model(filename)
else:
    print('Build the model from scratch')

    model = tf.keras.models.Sequential([
        tf.keras.layers.Flatten(input_shape=(28, 28)),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(10, activation='softmax')
    ])

    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

checkpoint = ModelCheckpoint(filename, monitor='loss',
verbose=1, save_best_only=True, mode='min')

model.fit(x_train, y_train, epochs=5, batch_size = 1000,
validation_split = 0.1, callbacks=[checkpoint])

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

Listing 1. Checkpoint and Restart code example (program_name.py)

5. The Deep Learning Model can be run either in batch mode using a Slurm batch job-script or interactively on a compute node.

Running the model in existing Interactive-Mode: You can run the code shown in Listing 1 by using the following set of commands in an interactive session (you can use the `srun` command on Arc to start an interactive session):

```
(mypython)[username@c001]$ time python3 \
program_name.py
```

When the model is trained the first time, it will build the model from scratch as there is no checkpoint file yet. A snippet of the output on the command line looks like the following:

```
Build the model from scratch
Epoch 1/5
54/54 [==============================] - 1s 8ms/step -
loss: 0.9088 - accuracy: 0.7429 - val_loss: 0.3207 -
val_accuracy: 0.9152

Epoch 00001: loss improved from inf to 0.90881, saving
model to mymodel.h5
Epoch 2/5
54/54 [==============================] - 0s 4ms/step -
loss: 0.3699 - accuracy: 0.8943 - val_loss: 0.2330 -
val_accuracy: 0.9382

Epoch 00002: loss improved from 0.90881 to 0.36992,
saving model to mymodel.h5
Epoch 3/5
54/54 [==============================] - 0s 4ms/step -
loss: 0.2943 - accuracy: 0.9151 - val_loss: 0.1956 -
val_accuracy: 0.9475

Epoch 00003: loss improved from 0.36992 to 0.29429,
saving model to mymodel.h5
Epoch 4/5
54/54 [==============================] - 0s 3ms/step -
loss: 0.2541 - accuracy: 0.9268 - val_loss: 0.1725 -
val_accuracy: 0.9540

Epoch 00004: loss improved from 0.29429 to 0.25414,
saving model to mymodel.h5
Epoch 5/5
```

```
54/54 [==============================] - 0s 4ms/step -
loss: 0.2259 - accuracy: 0.9348 - val_loss: 0.1538 -
val_accuracy: 0.9605

Epoch 00005: loss improved from 0.25414 to 0.22594,
saving model to mymodel.h5
313/313 - 0s - loss: 0.1814 - accuracy: 0.9500

real 0m5.511s
user 0m25.224s
sys  0m20.343s
```

When the model is executed again in the same directory, The model is loaded from the checkpoint file and continues the training from where it was left off. The output looks like the following:

```
Resuming
Epoch 1/5
54/54 [==============================] - 1s 7ms/step -
loss: 0.2028 - accuracy: 0.9419 - val_loss: 0.1402 -
val_accuracy: 0.9632

Epoch 00001: loss improved from inf to 0.20277, saving
model to mymodel.h5
Epoch 2/5
54/54 [==============================] - 0s 4ms/step -
loss: 0.1863 - accuracy: 0.9467 - val_loss: 0.1299 -
val_accuracy: 0.9652

Epoch 00002: loss improved from 0.20277 to 0.18626,
saving model to mymodel.h5
Epoch 3/5
```

```
54/54 [==============================] - 0s 4ms/step -
loss: 0.1705 - accuracy: 0.9514 - val_loss: 0.1210 -
val_accuracy: 0.9687

Epoch 00003: loss improved from 0.18626 to 0.17048,
saving model to mymodel.h5
Epoch 4/5
54/54 [==============================] - 0s 4ms/step -
loss: 0.1590 - accuracy: 0.9536 - val_loss: 0.1141 -
val_accuracy: 0.9702

Epoch 00004: loss improved from 0.17048 to 0.15904,
saving model to mymodel.h5
Epoch 5/5
54/54 [==============================] - 0s 4ms/step -
loss: 0.1482 - accuracy: 0.9571 - val_loss: 0.1059 -
val_accuracy: 0.9713

Epoch 00005: loss improved from 0.15904 to 0.14821,
saving model to mymodel.h5
313/313 - 0s - loss: 0.1252 - accuracy: 0.9640

real	0m5.039s
user	0m27.404s
sys	0m19.905s
```

Running the model in Batch-Mode: Interactive jobs can only be executed until a particular time frame. In order to run your job for more than that timeframe, you need to submit your model training as a batch job to the cluster. A sample Slurm batch job-script to run the python program of deep learning model in batch mode is shown in Listing 2. This script should be run from a login node.

```
#!/bin/bash
```

```
#SBATCH -J program_name
#SBATCH -o program_name.txt
#SBATCH -p normal
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -t 00:10:00

ml cuda/toolkit/11.3
ml cuda/cudnn/8.2.1.32

source mypython/bin/activate

time python3 program_name.py
```

Listing 2 : Batch Job Script for checkpoint and restart example (job_script1.slurm)

If you are currently on a compute node and would like to switch back to the login node then please enter the exit command as follows:

```
(mypython)[username@c001]$ exit
```

The job-script shown in Listing 2 can be submitted as follows:

```
[username@login001]$ sbatch job_script1.slurm
```

The output from the Slurm batch-job shown in Listing 2 can be checked by opening the output file as follows:

```
[username@login001]$ cat program_name.txt
```

A snippet from the output file is shown here:

When the model is trained the first time, it will build the model from scratch as there is no checkpoint file yet. A snippet of the output on the command line looks like the following:

```
Build the model from scratch
Epoch 1/5
54/54 [==============================] - 1s 8ms/step -
loss: 0.9088 - accuracy: 0.7429 - val_loss: 0.3207 -
val_accuracy: 0.9152

Epoch 00001: loss improved from inf to 0.90881, saving
model to mymodel.h5
Epoch 2/5
54/54 [==============================] - 0s 4ms/step -
loss: 0.3699 - accuracy: 0.8943 - val_loss: 0.2330 -
val_accuracy: 0.9382

Epoch 00002: loss improved from 0.90881 to 0.36992,
saving model to mymodel.h5
Epoch 3/5
54/54 [==============================] - 0s 4ms/step -
loss: 0.2943 - accuracy: 0.9151 - val_loss: 0.1956 -
val_accuracy: 0.9475

Epoch 00003: loss improved from 0.36992 to 0.29429,
saving model to mymodel.h5
Epoch 4/5
54/54 [==============================] - 0s 3ms/step -
loss: 0.2541 - accuracy: 0.9268 - val_loss: 0.1725 -
val_accuracy: 0.9540

Epoch 00004: loss improved from 0.29429 to 0.25414,
saving model to mymodel.h5
Epoch 5/5
54/54 [==============================] - 0s 4ms/step -
loss: 0.2259 - accuracy: 0.9348 - val_loss: 0.1538 -
val_accuracy: 0.9605
```

```
Epoch 00005: loss improved from 0.25414 to 0.22594,
saving model to mymodel.h5
313/313 - 0s - loss: 0.1814 - accuracy: 0.9500

real	0m5.511s
user	0m25.224s
sys	0m20.343s
```

When the model is executed again in the same directory, The model is loaded from the checkpoint file and continues the training from where it was left off. The output looks like the following:

```
Resuming
Epoch 1/5
54/54 [==============================] - 1s 7ms/step -
loss: 0.2028 - accuracy: 0.9419 - val_loss: 0.1402 -
val_accuracy: 0.9632

Epoch 00001: loss improved from inf to 0.20277, saving
model to mymodel.h5
Epoch 2/5
54/54 [==============================] - 0s 4ms/step -
loss: 0.1863 - accuracy: 0.9467 - val_loss: 0.1299 -
val_accuracy: 0.9652

Epoch 00002: loss improved from 0.20277 to 0.18626,
saving model to mymodel.h5
Epoch 3/5
54/54 [==============================] - 0s 4ms/step -
loss: 0.1705 - accuracy: 0.9514 - val_loss: 0.1210 -
val_accuracy: 0.9687

Epoch 00003: loss improved from 0.18626 to 0.17048,
saving model to mymodel.h5
```

```
Epoch 4/5
54/54 [==============================] - 0s 4ms/step -
loss: 0.1590 - accuracy: 0.9536 - val_loss: 0.1141 -
val_accuracy: 0.9702

Epoch 00004: loss improved from 0.17048 to 0.15904,
saving model to mymodel.h5
Epoch 5/5
54/54 [==============================] - 0s 4ms/step -
loss: 0.1482 - accuracy: 0.9571 - val_loss: 0.1059 -
val_accuracy: 0.9713

Epoch 00005: loss improved from 0.15904 to 0.14821,
saving model to mymodel.h5
313/313 - 0s - loss: 0.1252 - accuracy: 0.9640

real 0m5.039s
user 0m27.404s
sys  0m19.905s
```

REFERENCES

1. https://hpcsupport.utsa.edu/foswiki/bin/view/Main/TensorFlow
2. https://uoa-eresearch.github.io/eresearch-cookbook/recipe/2014/11/26/python-virtual-env/
3. https://portal.tacc.utexas.edu/software/tensorflow