Running Tensorflow and Keras on HPCC[1]

Mengying Sun 05/26/2017

1 Terminology

- Neural Network: a network that is trained to build a mapping between the given input and the output, consisting of one or multiple layers.
- Deep network/ Deep learning: neural network with multiple hidden layers, usually more than 4.
- Python: one of the most popular programing languages. There are two versions, Python 2 and Python 3. Tensorflow uses the version 3 to enable GPU acceleration on HPCC.
- Tensorflow (TF): The most popular open-source deep network library, originally developed by Google.
- Keras: a high-level neural networks API that simplifies the layer definition and modal construction in Tensorflow (it uses Tensorflow as its backend).
- CUDA: a parallel computing platform and programming model invented by NVIDIA. CUDA is required to use GPU on HPCC.
- cuDNN: The NVIDIA CUDA Deep Neural Network library.

2 Install Keras on HPCC

Since Tensorflow is the backend, we first import it [2].

Currently, it is better to run your code on node `dev-intel16-k80`. You may encounter errors on other development nodes.

```
1 ssh dev-intel16-k80
2 source /opt/software/tensorflow/0.12/usetfgpu12
```

The command line be like
`(python) [your_msuid@dev-intel16-k80 ~]$`

[1] The High Performance Computer Center in Michigan State University

[2] The latest Tensorflow version is 1.1.0. Here we use old version 0.12 to demo.

This provides you some of the basic modules like CUDA and cuDNN as well as Python 3.6.0. You don't need to learn how to use CUDA and cuDNN, TF will take care of them.

Next, create your own virtual environment, so that you can manage Python package and other libraries by yourself. Run:

```
cd ~
virtualenv myPy
```

("myPy" can be change to anything you like.)

Next, activate your own virtualenv by

```
source myPy/bin/activate
```

The previous Python 3.6.0 is substituted by the Python in your virtualenv. The command line becomes

`(myPy) [your_msuid@dev-intel16-k80 ~]$`

Now we can install Keras. But remember, Keras rely on Tensoflow, since you `source` your virtualenv, the previous TF is no longer available. So we execute `pip install` as follows:

```
pip install tensorflow
pip install tensorflow-gpu
pip install keras
pip install h5py
pip install numpy
pip install pandas
pip install ...
```

Make sure you set TF as your backend. Because Keras supports both TF and another library "Theano" as the backend, configuration is required prior to code execution. For that,

```
cd ~/.keras
ls
nano keras.json
```

Check whether "backend" is "tensorflow", if not, change it.

3 Miscellaneous Issues

3.1 Re-enter HPCC

Commands you need to execute when you exit HPCC and login again:

```
source /opt/software/tensorflow/0.12/usetfgpu12
source myPy/bin/activate
```

Sourcing TF is essential because it provides the runtime environment (CUDA, cuDNN, PATH configuration etc.) for tensorflow-gpu. You just need to replace its python part because there is no `keras` in TF Python 3.6.0.

3.2 Run Python Script

After you load everything, just run

```
cd path_to_your_code
python your_code.py
```

3.3 Qsub

A .qsub example:

```
#!/bin/bash -login

#PBS -N hpcc_keras_demo
#PBS -l nodes=1:ppn=2:gpus=2:shared
#PBS -l feature='gpgpu:intel16'
#PBS -l walltime=04:00:00
#PBS -l mem=20gb
#PBS -j oe
#PBS -m bea
#PBS -o /mnt/home/sunmeng2/proj01/

# source tensorflow gpu modules and activate virtual environment
source /opt/software/tensorflow/0.12/usetfgpu12
source /mnt/home/sunmeng2/myPy/bin/activate

# change to directory where your code is located, call your execute
python /mnt/home/sunmeng2/proj01/demo.py
```

4 Demo in Keras

We use a small subset of UK biobank height data as an example. Our response is human height (500 training, 500 testing), I randomly select 20 SNPs as variables.

```
from __future__ import print_function
import numpy as np
import pandas as pd     # read csv as data frame
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten    # type of layers
from keras import regularizers      # not used yet
from keras.models import load_model
from keras import optimizers
from keras.callbacks import CSVLogger

PATH_ROOT = '/mnt/research/quantgen/projects/sunmeng2/proj01/output/random_subset/'

XTRN_location = PATH_ROOT + 'XTRN_demo.csv'
yTRN_location = PATH_ROOT + 'yTRN_demo.csv'
XTST_location = PATH_ROOT + 'XTST_demo.csv'
yTST_location = PATH_ROOT + 'yTST_demo.csv'

x_train = pd.read_csv(XTRN_location).as_matrix()
y_train = pd.read_csv(yTRN_location).as_matrix()
x_test = pd.read_csv(XTST_location).as_matrix()
y_test = pd.read_csv(yTST_location).as_matrix()

# Computation graph
model = Sequential()
model.add(Dense(256, input_dim=20, kernel_initializer='normal', activation='relu'))
model.add(Dense(16, kernel_initializer='normal'))
model.add(Dense(1, kernel_initializer='normal'))

# Compile model
model.compile(loss='mean_squared_error', optimizer='adam')

# Train model
out_file = './result_log.log'
csv_logger = CSVLogger(out_file, separator=',', append=False)


```

```
# Fit Model
model.fit(x_train, y_train,
                verbose=1,
                validation_data=(x_test, y_test),
                batch_size=64,
                epochs=10,
                callbacks=[csv_logger])

# prediction
y_pred = model.predict(x_test, batch_size=64, verbose=0)
y_pred = y_pred.reshape((500,))
y_test = y_test.reshape((500,))

corr_coef = np.corrcoef(y_test, y_pred)[0, 1]
print("Correlation y_pred & y_test: ", corr_coef)
```

5 Advance Features in Keras

5.1 CSVLogger

CSVLogger is a callback that save monitored metrics. After you define and compile your model, run

```
from keras import callbacks

...

# CSV logger
csv_path = os.path.join(ROOT_PATH, 'result_output/' + model_name + '_log.log')

csv_logger = callbacks.CSVLogger(csv_path, append=True)
model.fit(X_train, Y_train,
                batch_size=batch_size,
                epochs=nb_epoch,
                verbose=1,
                validation_data=(X_test, Y_test),
                callbacks=[csv_logger])
```

Sample output:

```
epoch,loss,val_loss
0,38.2108598633,42.2379761353
1,38.1071882935,42.235447937
2,38.0264015503,42.2281964111
3,37.9269591675,42.2269703979
4,37.7690524292,42.2314937134
5,37.5990875854,42.2710250854
6,37.3220716553,42.343774231
7,37.0408745422,42.3654646912
8,36.6247359314,42.4591444092
9,37.3142503357,42.7219475708
```

5.2 Save Model and Weights to Disk

Simply run:

```
model_path = 'your_model_path.h5'
model.save(model_path)
```

Do not forget the ".h5" in your path. This function saves your model and weights to disk so that you can continue training your model next time or reproducing your result.

To load your model, `load_model` class is needed.

```
from keras.models import load_model

...

model_path = 'your_model_path.h5'
model = load_model(model_path, custom_objects={})
```

5.3 Early Stopping

Early stopping is a callback that allows you to save your model when the loss or accuracy reach a plateau.

```
early_stopper = callbacks.EarlyStopping(monitor='val_loss', min_delta=0, ↩
    patience=5, verbose=0)
model.fit(X_train, Y_train,
            batch_size=batch_size,
            epochs=nb_epoch,
```

```
            verbose=1,
            validation_data=(X_test, Y_test),
            callbacks=[csv_logger, early_stopper])
```

5.4 Show Model Structure

To show your model structure (output dimension, number of parameters and so on), use

```
print(model.summary())
```

Sample output:

```

Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 256)               5376
_________________________________________________________________
dense_2 (Dense)              (None, 16)                4112
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 17
=================================================================
Total params: 9,505
Trainable params: 9,505
Non-trainable params: 0
```

6 Tips

- It's better to create your neural network learning model in a python **class**, and then import it in your main code.
- Be care of your input data dimension.
- Parameters and settings that you can tune:
  - Learning rate
  - Optimizer
  - Loss function
  - Batch size
  - Epoch number

  - Regularization
  - Model weights initializer
  - Data enhancement in preprocessing
  - Layer hidden unit number
  - Network structure

- Do not include everything in your code before running; first build a sketch that works, and then add functions to it one by one.
- Keras has its limitations. Use Tensorflow instead in complicated models.

7 Reference

- HPCC wiki - Tensorflow: `https://wiki.hpcc.msu.edu/display/hpccdocs/TensorFlow`
- Tensorflow: `https://www.tensorflow.org/`
- Keras: `https://keras.io/`