DECADES TensorFlow Examples

Here we will walk through a simple Keras examples (from the official Keras examples (https://github.com/keras-team/keras/tree/master/examples)) in its entirety for the TensorFlow flow through DECADES.

We will go through a convolutional neural net example. Before reading this guide, we hope that you have familiarity with CNNs and have gone over:

1. Intro to DECADES programming document (intro.ipynb)
2. Intro to programming DECADES through TensorFlow (DECADES_TensorFlow.ipynb)

Location

This example can be found in the docker at /decades/applications/tensorflow/examples/.

Convolutional Neural Network

The example code is written in dec_mnist_cnn.py. We first import the necessary TensorFlow libraries:

```
'''
Trains a simple convnet on the MNIST dataset.

Gets to 99.25% test accuracy after 12 epochs
(there is still a lot of margin for parameter tu
ning).
16 seconds per epoch on a GRID K520 GPU.
'''

from __future__ import print_function

# Helper libraries
import os
import sys
sys.path.append('../')

# Tensorflow and tf.keras
import tensorflow as tf
from tensorflow import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.optimizers import Adadelta
from keras import backend as K
```

We then need to import the DECADES TensorFlow library, "DEC_TensorFlow" so that we can feed the computation graph to the DECADES framework and employ our tools to perform an analysis.

```
# DECADES libraries
import DEC_TensorFlow as dtf
```

We then create a function that makes it easier to dump and keep track of program traces. The function takes in a graph of all operations in the TensorFlow program (which is simply `tf.get_default_graph().get_operations()`) as well as the name of the file to where the DECADES TensorFlow C++ code is to be written. Within the function we employ the DECADES framework with `dtf.run()` . We also print out the important TensorFlow functions with `dtf.print_trace()` . This `dump_trace()` function is optional; you can simply pass in a C++ filename and `tf.get_default_graph()` to dtf.run().

```
def dump_trace(ops, filename):

    filedir = "output/"

    if (not os.path.exists(filedir)):
        os.mkdir(filedir)

    call_trace = open(filedir + filename + ".txt
", "w")

    for op in ops:
        call_trace.write(str(op.name) + "\n")

    call_trace.close()

    args_trace_name = filedir + filename + "_arg
s.txt"
    with open(args_trace_name, 'w') as args_trac
e:
        sys.stdout = args_trace
        dtf.print_trace(tf.get_default_graph())
        dtf.run(filedir + filename + ".cpp", tf.
get_default_graph())
        sys.stdout = sys.__stdout__
```

Now we can write out our neural net training code. We first set our training parameters (batch size, number of classes, number of epochs, etc.) and load in data from the MNIST dataset. We also need to perform some data reshaping based on how the image data is formatted.

```
batch_size = 128
num_classes = 10
epochs = 1 #12 originally

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.loa
d_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0],
1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1,
img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0],
img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img
_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, nu
m_classes)
y_test = keras.utils.to_categorical(y_test, num_
classes)
```

We can now utilize the Keras API functions to build our neural net.

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax
'))
```

We then compile the model, perform training with `model.fit()` , and perform inference with `model.evaluate()` .

```
model.compile(loss=keras.losses.categorical_cros
sentropy,
              optimizer=Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))

score = model.evaluate(x_test, y_test, verbose=0
)
```

And we can print out our accuracies.

```
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Lastly, we utilize our `dump_trace()` function by passing in 1) the operations graph, `tf.get_default_graph().get_operations()` and 2) the filename that we obtain from our current directory `os.path.splitext(os.path.basename(__file__))[0]` .

```
# For DECADES: dump TF function trace
dump_trace(tf.get_default_graph().get_operations
(), os.path.splitext(os.path.basename(__file__))
[0])
```

Now that you have finished writing the code for this example, you can run the application:

```
python dec_mnist_cnn.py
```

Once the application has finished running, you should have generated a C++ file, `dec_mnist_cnn.cpp` .