Decades_Tensorflow_Pipeline

June 14, 2019

1 DECADES TensorFlow

Welcome to the DECADES TensorFlow path! Here we will detail how to utilize the DECADES framework for neural net applications written in TensorFlow.

1.1 Keras

Keras is a high-level API for building and training deep learning models in TensorFlow. This API allows you to easily write NN applications without having to worry about detailed TensorFlow backend functions. We support the entire Keras API, so you can utilize any Keras API function. Our main functions of interest are the commonly used functions that appear in many NN applications and largely contribute to total computation time. We list these functions below:

Utilities: * datasets * initializers * metrics * models * utils

Computation: * activations * elu * linear * relu * selu * sigmoid * softmax * tanh

Layers: * Activation * Add * AveragePooling2D * BatchNormalization * Conv2D * Dense * (GRU) * (GRUCell) * (LSTM) * (LSTMCell) * MaxPool2D * Multiply * multiply * PReLU * ReLU * (Reshape) * (RNN) * (SimpleRNN) * (SimpleRNNCell) * Softmax * Subtract * subtract * (ZeroPadding2D)

Losses

Optimizers

Please refer to the Keras API documentation for details on all Keras functions and their inputs.

1.2 Environment

You must first gather the necessary data for your application of interest. For example, this might simply entail running `prep.sh`.

In order to run TensorFlow programs on the docker, you need the DECADES TensorFlow environment running. Once you have obtained the data on the docker, perform the following (after deactivating any provided conda environments, if necessary):

```
In [ ]: conda activate tf
```

1.3 Writing Neural Net Applications with DECADES

It is easy to use the DECADES framework when writing NN applications. You first write your application using Keras functions (you can find many examples here), like you normally would.

Once you have written your application, you need to make two simple additions to invoke the DECADES TensorFlow library.

First, import the DECADES TensorFlow library:

```
In [ ]: import DEC_TensorFlow as dtf
```

This will incorporate the DECADES TensorFlow library so we can use its functions to generate a C++ translation of your NN application. We generate a C++ file so that we can feed it to our analysis tools that are built for C++ programs. The generated C++ will capture all of the important functions (highlighted above) and express them in C++ syntax so we can invoke DECADES accelerators and measure the performance and power of your application.

To generate this C++ from running your application, you will need to add this function call at the very bottom of your `main()` function or your code. You can find examples of how to to run tensorflow applications with Decades pipeline here

```
In [ ]: dtf.run(filename, tf.get_default_graph())
```

where `filename` is the name you would like for your C++ file. It must end with `.cpp`.

If you would like to use our evaluation tools, you must have `filename` match the name of your Python application. For example, if you're writing `app.py`, then `filename` should be `app`.

To go through an example, take a look here.

1.3.1 Batching with DECADES

Most NN applications use batching both for training and inference. For the DECADES tools to best capture it, we suggest to define the batch size statically in the first layer of the model, as shown below:

```
In [ ]: # example from the mnist_mlp.py Keras app
        model.Sequential()
        model.add(Dense(512, activation='relu', input_shape=(784,), batch_size=128))
```

The statically defined batch size can slow down considerably the DECADES simulation in the case of networks with convolutional layers. In those cases we recommend you to avoid the statically defined batch size, at least during the development and testing phases.