Tensorflow with Keras

Sauman Das

September 2021

1 Introduction

Last week we went over a basic Pytorch example. This week, we will try to get more into TensorFlow and Keras, a popular deep learning framework made by Google. This handout will mainly consist of general info about the library, but we will be doing a live coding demo to learn how to properly use the library.

2 Important Documentation

2.1 Data Loading

One of the most useful tools that TensorFlow provides is their Data Generators. You can use these generators directly while training your models. Generators are not only convenient, but also a requirement at times. For example, if we have 100,000 RGB images each of size 224x224x3, we will not be able to store them all in an array because we will run out of RAM. Even if you do not run out of RAM, your model may train very slowly. With a generator, we can generate a batch of images/data and then recycle them when yielding the subsequent batch.

TensorFlow allows us to build custom generators when the data is not as clean as we would like it to be. But in general, when dealing with image data, we can use TensorFlow's *ImageDataGenerator*. This makes it really easy to apply data augmentation without writing many additional lines of code.

2.2 Building a Model

There are three primary ways to build a model using Keras. The simplest is using the Sequential API.

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(8, activation='relu'))
model.add(tf.keras.layers.Dense(4, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

The Sequential API is great for constructing simple models, however, it poses many limitations which is why we often use another method called the Functional API.

```
layer1 = tf.keras.layers.Dense(8, activation='relu')
layer2 = tf.keras.layers.Dense(4, activation='relu')(layer1)
layer3 = tf.keras.layers.Dense(1, activation='sigmoid')(layer2)
model = tf.keras.Model(inputs=layer1, outputs=layer3)
```

Finally, the third method which is the most complex is by using model subclassing. You can read more about this method at this link: *Model Subclassing*

2.3 Training with *.fit*

This is the most simple way to train a TensorFlow model, though sometimes it may not work.

Simple Case:

```
history = model.fit(x, y, validation_data=(x_val, y_val), batch_size=8, epochs=3)
```

2.4 Training with *Gradient Tape*

With the convenient *.fit* method, it may seem unnecessary to use any other form of training. However, there are many cases where we want to train the model in a special way. When we call *.fit*, the data is automatically passed through the model with the given batch size and the entire model is updated. There are many scenarios where we would not like the gradients to flow through the entire model, or we would like to specify the way batches are selected instead of the default that is provided.

Basic Scenario:

```
with tf.GradientTape() as tape:
        # feed the images into the model
        logits = model(x)

        # Compute the loss
        loss = tf.nn.sigmoid_cross_entropy_with_logits(labels=y, logits=logits)

    grads = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

3 Real World Research

Check out how DeepMind used TensorFlow to publish breakthrough research through the AlphaFold project: *AlphaFold Github Link*