Miscellaneous Topics

December 21, 2017

1 Deep Nets with TF Abstractions

Let's explore a few of the various abstractions that TensorFlow offers. You can check out the tf.contrib documentation for more options.

2 The Data

To compare these various abstractions we'll use a dataset easily available from the SciKit Learn library. The data is comprised of the results of a chemical analysis of wines grown in the same region in Italy by three different cultivators. There are thirteen different measurements taken for different constituents found in the three types of wine. We will use the various TF Abstractions to classify the wine to one of the 3 possible labels.

First let's show you how to get the data:

```
In [3]: from sklearn.datasets import load_wine
        wine_data = load_wine()
        print(type(wine_data))
```

```
<class 'sklearn.utils.Bunch'>
```

The data is a sklearn.utils.Bunch object, which is very similar to a dictionary.

```
In [4]: wine_data.keys()
```

```
Out[4]: dict_keys(['target', 'feature_names', 'data', 'target_names', 'DESCR'])
```

You can get a full description with **print(wine_data.DESCR)**. For now, let's go ahead and grab the features and the labels for the data.

```
In [5]: feat_data = wine_data['data']
        labels = wine_data['target']
```

2.0.1 Train Test Split

As with any machine learning model, you should do some sort of test train split so you can evaluate your model's performance. Because this particular dataset is small, we'll just do a simple 70/30 train test split and we won't have any holdout data set.

Again, we'll use SciKit-Learn here for convienence:

```
In [6]: from sklearn.model_selection import train_test_split

        X_train, X_test, y_train, y_test = train_test_split(feat_data,
                                                            labels,
                                                            test_size=0.3,
                                                           random_state=101)
```

2.0.2 Scale the Data

With Neural Network models, its important to scale the data, again we can do this easily with SciKit Learn (I promise we'll get to TensorFlow soon!)

```
In [7]: from sklearn.preprocessing import MinMaxScaler
        scaler = MinMaxScaler()
```

Keep in mind we only fit the scaler to the training data, we don't want to assume we'll have knowledge of future test data.

```
In [8]: scaled_x_train = scaler.fit_transform(X_train)
        scaled_x_test = scaler.transform(X_test)
```

3 Abstractions

With our data set up, its now time to explore some TensorFlow abstractions! Let's start with the Estimator API, its one the abstractions featured in the official documentation tutorials.

3.1 Estimator API

We first start by importing both tensorflow and the estimator API.

```
In [9]: import tensorflow as tf
        from tensorflow import estimator
```

The estimator API can perform both Deep Neural Network Classification and Regression, as well as straight Linear Classification and Linear Regression. You can

```
In [ ]: estimator.DNNClassifier
        estimator.DNNRegressor
        estimator.
```

```
In [261]: X_train.shape
```

```
Out[261]: (124, 13)
```

In [262]:
```
feat_cols = [tf.feature_column.numeric_column("x", shape=[13])]
```

In [263]:
```
deep_model = estimator.DNNClassifier(
        hidden_units=[13,13,13],
        feature_columns=feat_cols,
        n_classes=3,
        optimizer=tf.train.GradientDescentOptimizer(
            learning_rate=0.01))
```

```
INFO:tensorflow:Using default config.
WARNING:tensorflow:Using temporary folder as model directory: C:\Users\Marcial\AppData\Local\Tem
INFO:tensorflow:Using config: {'_keep_checkpoint_max': 5, '_save_checkpoints_steps': None, '_sav
```

In [264]:
```
input_fn = estimator.inputs.numpy_input_fn(
        x={'x':scaled_x_train},
        y=y_train,
        shuffle=True,
        batch_size=10,
        num_epochs=5)
```

In [265]:
```
deep_model.train(input_fn=input_fn,steps=500)
```

```
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Saving checkpoints for 1 into C:\Users\Marcial\AppData\Local\Temp\tmpn5i9jsyx\mo
INFO:tensorflow:step = 1, loss = 10.1293
INFO:tensorflow:Saving checkpoints for 62 into C:\Users\Marcial\AppData\Local\Temp\tmpn5i9jsyx\m
INFO:tensorflow:Loss for final step: 1.79451.
```

Out[265]:
```
<tensorflow.python.estimator.canned.dnn.DNNClassifier at 0x13fc2625780>
```

In [266]:
```
input_fn_eval = estimator.inputs.numpy_input_fn(
        x={'x':scaled_x_test},
        shuffle=False)
```

In [267]:
```
preds = list(deep_model.predict(input_fn=input_fn_eval))
```

```
INFO:tensorflow:Restoring parameters from C:\Users\Marcial\AppData\Local\Temp\tmpn5i9jsyx\model.
```

In [268]:
```
predictions = [p['class_ids'][0] for p in preds]
```

In [269]:
```
from sklearn.metrics import confusion_matrix,classification_report
```

In [270]:
```
print(classification_report(y_test,predictions))
```

```
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        19
```

```
          1       1.00      0.91      0.95        22
          2       0.87      1.00      0.93        13

avg / total       0.97      0.96      0.96        54
```

---

4 TensorFlow Keras

4.0.1 Create the Model

```
In [728]: from tensorflow.contrib.keras import models

In [729]: dnn_keras_model = models.Sequential()
```

4.0.2 Add Layers to the model

```
In [730]: from tensorflow.contrib.keras import layers

In [731]: dnn_keras_model.add(
              layers.Dense(units=13,input_dim=13,activation='relu'))

In [732]: dnn_keras_model.add(layers.Dense(units=13,activation='relu'))
          dnn_keras_model.add(layers.Dense(units=13,activation='relu'))

In [733]: dnn_keras_model.add(layers.Dense(units=3,activation='softmax'))
```

4.0.3 Compile the Model

```
In [734]: from tensorflow.contrib.keras import losses,optimizers,metrics

In [735]: # explore these
          # losses.

In [736]: #optimizers.

In [744]: losses.sparse_categorical_crossentropy

Out[744]: <function tensorflow.contrib.keras.python.keras.losses.sparse_categorical_crossentropy

In [738]: dnn_keras_model.compile(optimizer='adam',
                          loss='sparse_categorical_crossentropy',
                          metrics=['accuracy'])
```

4.0.4 Train Model

```
In [741]: dnn_keras_model.fit(scaled_x_train,y_train,epochs=50)
```

```
Epoch 1/50
124/124 [==============================] - 0s - loss: 1.1481 - acc: 0.3226
Epoch 2/50
124/124 [==============================] - 0s - loss: 1.1310 - acc: 0.3226
Epoch 3/50
124/124 [==============================] - 0s - loss: 1.1141 - acc: 0.3226
Epoch 4/50
124/124 [==============================] - 0s - loss: 1.1000 - acc: 0.3226
Epoch 5/50
124/124 [==============================] - 0s - loss: 1.0866 - acc: 0.3226
Epoch 6/50
124/124 [==============================] - 0s - loss: 1.0746 - acc: 0.3790
Epoch 7/50
124/124 [==============================] - 0s - loss: 1.0627 - acc: 0.4597
Epoch 8/50
124/124 [==============================] - 0s - loss: 1.0521 - acc: 0.4758
Epoch 9/50
124/124 [==============================] - 0s - loss: 1.0422 - acc: 0.5323
Epoch 10/50
124/124 [==============================] - 0s - loss: 1.0327 - acc: 0.5565
Epoch 11/50
124/124 [==============================] - 0s - loss: 1.0236 - acc: 0.5968
Epoch 12/50
124/124 [==============================] - 0s - loss: 1.0141 - acc: 0.6210
Epoch 13/50
124/124 [==============================] - 0s - loss: 1.0041 - acc: 0.6532
Epoch 14/50
124/124 [==============================] - 0s - loss: 0.9944 - acc: 0.6613
Epoch 15/50
124/124 [==============================] - 0s - loss: 0.9845 - acc: 0.6532
Epoch 16/50
124/124 [==============================] - 0s - loss: 0.9744 - acc: 0.6532
Epoch 17/50
124/124 [==============================] - 0s - loss: 0.9636 - acc: 0.6532
Epoch 18/50
124/124 [==============================] - 0s - loss: 0.9531 - acc: 0.6613
Epoch 19/50
124/124 [==============================] - 0s - loss: 0.9415 - acc: 0.6532
Epoch 20/50
124/124 [==============================] - 0s - loss: 0.9297 - acc: 0.6532
Epoch 21/50
124/124 [==============================] - 0s - loss: 0.9175 - acc: 0.6532
Epoch 22/50
124/124 [==============================] - 0s - loss: 0.9052 - acc: 0.6613
Epoch 23/50
```

```
124/124 [==============================] - 0s - loss: 0.8916 - acc: 0.6613
Epoch 24/50
124/124 [==============================] - 0s - loss: 0.8781 - acc: 0.6694
Epoch 25/50
124/124 [==============================] - 0s - loss: 0.8639 - acc: 0.6694
Epoch 26/50
124/124 [==============================] - 0s - loss: 0.8487 - acc: 0.6694
Epoch 27/50
124/124 [==============================] - 0s - loss: 0.8334 - acc: 0.6935
Epoch 28/50
124/124 [==============================] - 0s - loss: 0.8164 - acc: 0.7177
Epoch 29/50
124/124 [==============================] - 0s - loss: 0.7997 - acc: 0.7500
Epoch 30/50
124/124 [==============================] - 0s - loss: 0.7825 - acc: 0.7581
Epoch 31/50
124/124 [==============================] - 0s - loss: 0.7644 - acc: 0.7823
Epoch 32/50
124/124 [==============================] - 0s - loss: 0.7461 - acc: 0.7984
Epoch 33/50
124/124 [==============================] - 0s - loss: 0.7268 - acc: 0.8387
Epoch 34/50
124/124 [==============================] - 0s - loss: 0.7076 - acc: 0.8548
Epoch 35/50
124/124 [==============================] - 0s - loss: 0.6868 - acc: 0.8871
Epoch 36/50
124/124 [==============================] - 0s - loss: 0.6661 - acc: 0.8952
Epoch 37/50
124/124 [==============================] - 0s - loss: 0.6446 - acc: 0.8952
Epoch 38/50
124/124 [==============================] - 0s - loss: 0.6224 - acc: 0.9032
Epoch 39/50
124/124 [==============================] - 0s - loss: 0.6000 - acc: 0.9274
Epoch 40/50
124/124 [==============================] - 0s - loss: 0.5784 - acc: 0.9274
Epoch 41/50
124/124 [==============================] - 0s - loss: 0.5568 - acc: 0.9274
Epoch 42/50
124/124 [==============================] - 0s - loss: 0.5330 - acc: 0.9274
Epoch 43/50
124/124 [==============================] - 0s - loss: 0.5114 - acc: 0.9274
Epoch 44/50
124/124 [==============================] - 0s - loss: 0.4894 - acc: 0.9355
Epoch 45/50
124/124 [==============================] - 0s - loss: 0.4684 - acc: 0.9435
Epoch 46/50
124/124 [==============================] - ETA: 0s - loss: 0.5167 - acc: 0.875 - 0s - loss: 0.44
Epoch 47/50
```

```
124/124 [==============================] - ETA: 0s - loss: 0.3963 - acc: 1.000 - 0s - loss: 0.42
Epoch 48/50
124/124 [==============================] - 0s - loss: 0.4085 - acc: 0.9516
Epoch 49/50
124/124 [==============================] - 0s - loss: 0.3896 - acc: 0.9516
Epoch 50/50
124/124 [==============================] - 0s - loss: 0.3715 - acc: 0.9516
```

```
Out[741]: <tensorflow.contrib.keras.python.keras.callbacks.History at 0x13fd46aecf8>
```

```
In [742]: predictions = dnn_keras_model.predict_classes(scaled_x_test)
```

```
32/54 [================>...] - ETA: 0s
```

```
In [743]: print(classification_report(predictions,y_test))
```

```
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        19
          1       0.91      1.00      0.95        20
          2       1.00      0.87      0.93        15

avg / total       0.97      0.96      0.96        54
```

5 Layers API

https://www.tensorflow.org/tutorials/layers

5.1 Formating Data

```
In [1]: import pandas as pd
        from sklearn.datasets import load_wine
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import MinMaxScaler
```

```
In [2]: wine_data = load_wine()
        feat_data = wine_data['data']
        labels = wine_data['target']
```

```
In [23]: X_train, X_test, y_train, y_test = train_test_split(feat_data,
                                                             labels,
                                                             test_size=0.3,
                                                            random_state=101)
```

```
In [4]: scaler = MinMaxScaler()
        scaled_x_train = scaler.fit_transform(X_train)
        scaled_x_test = scaler.transform(X_test)
        # ONE HOT ENCODED
        onehot_y_train = pd.get_dummies(y_train).as_matrix()
        one_hot_y_test = pd.get_dummies(y_test).as_matrix()
```

5.1.1 Parameters

```
In [5]: num_feat = 13
        num_hidden1 = 13
        num_hidden2 = 13
        num_outputs = 3
        learning_rate = 0.01
```

```
In [6]: import tensorflow as tf
        from tensorflow.contrib.layers import fully_connected
```

5.1.2 Placeholder

```
In [7]: X = tf.placeholder(tf.float32,shape=[None,num_feat])
        y_true = tf.placeholder(tf.float32,shape=[None,3])
```

5.1.3 Activation Function

```
In [8]: actf = tf.nn.relu
```

5.1.4 Create Layers

```
In [9]: hidden1 = fully_connected(X,num_hidden1,activation_fn=actf)
```

```
In [10]: hidden2 = fully_connected(hidden1,num_hidden2,activation_fn=actf)
```

```
In [11]: output = fully_connected(hidden2,num_outputs)
```

5.1.5 Loss Function

```
In [12]: loss = tf.losses.softmax_cross_entropy(
             onehot_labels=y_true, logits=output)
```

5.1.6 Optimizer

```
In [13]: optimizer = tf.train.AdamOptimizer(learning_rate)
         train = optimizer.minimize(loss)
```

5.1.7 Init

```
In [14]: init = tf.global_variables_initializer()
```

```
In [21]: training_steps = 1000
         with tf.Session() as sess:
             sess.run(init)

             for i in range(training_steps):
                 sess.run(train,feed_dict={X:scaled_x_train,y_true:y_train})

             # Get Predictions
             logits = output.eval(feed_dict={X:scaled_x_test})

             preds = tf.argmax(logits,axis=1)

             results = preds.eval()
```

```
In [25]: from sklearn.metrics import confusion_matrix,classification_report
         print(classification_report(results,y_test))
```

```
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        19
          1       1.00      1.00      1.00        22
          2       1.00      1.00      1.00        13

avg / total       1.00      1.00      1.00        54
```

6 TensorBoard

```
In [1]: import tensorflow as tf
        with tf.name_scope("OPERATION_A"):
            a = tf.add(1,2,name="First_add")
            a1 = tf.add(100,200,name='a_add')
            a2 = tf.multiply(a,a1)


        with tf.name_scope("OPERATION_B"):
            b = tf.add(3,4,name='Second_add')
            b1 = tf.add(300,400,name='b_add')
            b2 = tf.multiply(b,b1)

        c = tf.multiply(a2,b2,name='final_result')

        with tf.Session() as sess:
```

```
    writer = tf.summary.FileWriter("./output",sess.graph)
    print(sess.run(c))
    writer.close()
```

```
4410000
```

```
In [2]: k = tf.placeholder(tf.float32)

        # Make a normal distribution, with a shifting mean
        mean_moving_normal = tf.random_normal(shape=[1000], mean=(5*k), stddev=1)
        # Record that distribution into a histogram summary
        tf.summary.histogram("normal/moving_mean", mean_moving_normal)

        # Setup a session and summary writer
        with tf.Session() as sess:
            writer = tf.summary.FileWriter("./tmp/histogram_example")

            summaries = tf.summary.merge_all()

            # Setup a loop and write the summaries to disk
            N = 400
            for step in range(N):

                k_val = step/float(N)
                summ = sess.run(summaries, feed_dict={k: k_val})
                writer.add_summary(summ, global_step=step)

            writer.close()
```