tf.estimator Quickstart

TensorFlow's high-level machine learning API (tf.estimator) makes it easy to configure, train, and evaluate a variety of machine learning models. In this tutorial, you'll use tf.estimator to construct a neural network (https://en.wikipedia.org/wiki/Artificial_neural_network) classifier and train it on the Iris data set (https://en.wikipedia.org/wiki/Iris_flower_data_set) to predict flower species based on sepal/petal geometry. You'll write code to perform the following five steps:

1. Load CSVs containing Iris training/test data into a TensorFlow `Dataset`
2. Construct a neural network classifier (https://www.tensorflow.org/api_docs/python/tf/estimator/DNNClassifier?hl=zh-cn)
3. Train the model using the training data
4. Evaluate the accuracy of the model
5. Classify new samples

NOTE: Remember to install TensorFlow on your machine (https://www.tensorflow.org/install/index?hl=zh-cn) before getting started with this tutorial.

Complete Neural Network Source Code

Here is the full code for the neural network classifier:

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import os
from six.moves.urllib.request import urlopen

import numpy as np
import tensorflow as tf

# Data sets
IRIS_TRAINING = "iris_training.csv"
IRIS_TRAINING_URL = "http://download.tensorflow.org/data/iris_training.csv"

IRIS_TEST = "iris_test.csv"
IRIS_TEST_URL = "http://download.tensorflow.org/data/iris_test.csv"

def main():
  # If the training and test sets aren't stored locally, download them.
  if not os.path.exists(IRIS_TRAINING):
    raw = urlopen(IRIS_TRAINING_URL).read()
    with open(IRIS_TRAINING, "wb") as f:
      f.write(raw)

  if not os.path.exists(IRIS_TEST):
    raw = urlopen(IRIS_TEST_URL).read()
    with open(IRIS_TEST, "wb") as f:
      f.write(raw)

  # Load datasets.
  training_set = tf.contrib.learn.datasets.base.load_csv_with_header(
      filename=IRIS_TRAINING,
      target_dtype=np.int,
      features_dtype=np.float32)
  test_set = tf.contrib.learn.datasets.base.load_csv_with_header(
      filename=IRIS_TEST,
      target_dtype=np.int,
      features_dtype=np.float32)

  # Specify that all features have real-value data
  feature_columns = [tf.feature_column.numeric_column("x", shape=[4])]

  # Build 3 layer DNN with 10, 20, 10 units respectively.
  classifier = tf.estimator.DNNClassifier(feature_columns=feature_columns,
```

```
                                        hidden_units=[10, 20, 10],
                                        n_classes=3,
                                        model_dir="/tmp/iris_model")
    # Define the training inputs
    train_input_fn = tf.estimator.inputs.numpy_input_fn(
        x={"x": np.array(training_set.data)},
        y=np.array(training_set.target),
        num_epochs=None,
        shuffle=True)

    # Train model.
    classifier.train(input_fn=train_input_fn, steps=2000)

    # Define the test inputs
    test_input_fn = tf.estimator.inputs.numpy_input_fn(
        x={"x": np.array(test_set.data)},
        y=np.array(test_set.target),
        num_epochs=1,
        shuffle=False)

    # Evaluate accuracy.
    accuracy_score = classifier.evaluate(input_fn=test_input_fn)["accuracy"]

    print("\nTest Accuracy: {0:f}\n".format(accuracy_score))

    # Classify two new flower samples.
    new_samples = np.array(
        [[6.4, 3.2, 4.5, 1.5],
         [5.8, 3.1, 5.0, 1.7]], dtype=np.float32)
    predict_input_fn = tf.estimator.inputs.numpy_input_fn(
        x={"x": new_samples},
        num_epochs=1,
        shuffle=False)

    predictions = list(classifier.predict(input_fn=predict_input_fn))
    predicted_classes = [p["classes"] for p in predictions]

    print(
        "New Samples, Class Predictions:    {}\n"
        .format(predicted_classes))

if __name__ == "__main__":
    main()
```

The following sections walk through the code in detail.

Load the Iris CSV data to TensorFlow

The Iris data set (https://en.wikipedia.org/wiki/Iris_flower_data_set) contains 150 rows of data, comprising 50 samples from each of three related Iris species: *Iris setosa*, *Iris virginica*, and *Iris versicolor*.



From left to right, *Iris setosa* (https://commons.wikimedia.org/w/index.php?curid=170298) **(by Radomil** (https://commons.wikimedia.org/wiki/User:Radomil)**, CC BY-SA 3.0), *Iris versicolor*** (https://commons.wikimedia.org/w/index.php?curid=248095) **(by Dlanglois** (https://commons.wikimedia.org/wiki/User:Dlanglois)**, CC BY-SA 3.0), and *Iris virginica*** (https://www.flickr.com/photos/33397993@N05/3352169862) **(by Frank Mayfield** (https://www.flickr.com/photos/33397993@N05)**, CC BY-SA 2.0).**

Each row contains the following data for each flower sample: sepal (https://en.wikipedia.org/wiki/Sepal) length, sepal width, petal (https://en.wikipedia.org/wiki/Petal) length, petal width, and flower species. Flower species are represented as integers, with 0 denoting *Iris setosa*, 1 denoting *Iris versicolor*, and 2 denoting *Iris virginica*.

Sepal Length	Sepal Width	Petal Length	Petal Width	Species
5.1	3.5	1.4	0.2	0
4.9	3.0	1.4	0.2	0
4.7	3.2	1.3	0.2	0
…	…	…	…	…
7.0	3.2	4.7	1.4	1
6.4	3.2	4.5	1.5	1
6.9	3.1	4.9	1.5	1
…	…	…	…	…
6.5	3.0	5.2	2.0	2
6.2	3.4	5.4	2.3	2
5.9	3.0	5.1	1.8	2

For this tutorial, the Iris data has been randomized and split into two separate CSVs:

- A training set of 120 samples (iris_training.csv (http://download.tensorflow.org/data/iris_training.csv?hl=zh-cn))
- A test set of 30 samples (iris_test.csv (http://download.tensorflow.org/data/iris_test.csv?hl=zh-cn)).

To get started, first import all the necessary modules, and define where to download and store the dataset:

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import os
from six.moves.urllib.request import urlopen

import tensorflow as tf
import numpy as np

IRIS_TRAINING = "iris_training.csv"
IRIS_TRAINING_URL = "http://download.tensorflow.org/data/iris_training.csv"

IRIS_TEST = "iris_test.csv"
IRIS_TEST_URL = "http://download.tensorflow.org/data/iris_test.csv"
```

Then, if the training and test sets aren't already stored locally, download them.

```
if not os.path.exists(IRIS_TRAINING):
  raw = urlopen(IRIS_TRAINING_URL).read()
  with open(IRIS_TRAINING,'wb') as f:
    f.write(raw)

if not os.path.exists(IRIS_TEST):
  raw = urlopen(IRIS_TEST_URL).read()
  with open(IRIS_TEST,'wb') as f:
    f.write(raw)
```

Next, load the training and test sets into `Datasets` using the `load_csv_with_header()` (https://www.github.com/tensorflow/tensorflow/blob/r1.4/tensorflow/contrib/learn/python/learn/datasets/base.py) method in `learn.datasets.base`. The `load_csv_with_header()` method takes three required arguments:

- **`filename`**, which takes the filepath to the CSV file
- **`target_dtype`**, which takes the numpy datatype (http://docs.scipy.org/doc/numpy/user/basics.types.html) of the dataset's target value.

- **features_dtype**, which takes the numpy datatype (http://docs.scipy.org/doc/numpy/user/basics.types.html) of the dataset's feature values.

Here, the target (the value you're training the model to predict) is flower species, which is an integer from 0–2, so the appropriate **numpy** datatype is **np.int**:

```
# Load datasets.
training_set = tf.contrib.learn.datasets.base.load_csv_with_header(
    filename=IRIS_TRAINING,
    target_dtype=np.int,
    features_dtype=np.float32)
test_set = tf.contrib.learn.datasets.base.load_csv_with_header(
    filename=IRIS_TEST,
    target_dtype=np.int,
    features_dtype=np.float32)
```

Datasets in tf.contrib.learn are named tuples (https://docs.python.org/2/library/collections.html#collections.namedtuple); you can access feature data and target values via the **data** and **target** fields. Here, **training_set.data** and **training_set.target** contain the feature data and target values for the training set, respectively, and **test_set.data** and **test_set.target** contain feature data and target values for the test set.

Later on, in "Fit the DNNClassifier to the Iris Training Data," (#fit_dnnclassifier) you'll use **training_set.data** and **training_set.target** to train your model, and in "Evaluate Model Accuracy," (#evaluate_accuracy) you'll use **test_set.data** and **test_set.target**. But first, you'll construct your model in the next section.

Construct a Deep Neural Network Classifier

tf.estimator offers a variety of predefined models, called **Estimators**, which you can use "out of the box" to run training and evaluation operations on your data. Here, you'll configure a Deep Neural Network Classifier model to fit the Iris data. Using tf.estimator, you can instantiate your **tf.estimator.DNNClassifier** (https://www.tensorflow.org/api_docs/python/tf/estimator/DNNClassifier?hl=zh-cn) with just a couple lines of code:

```
# Specify that all features have real-value data
feature_columns = [tf.feature_column.numeric_column("x", shape=[4])]

# Build 3 layer DNN with 10, 20, 10 units respectively.
classifier = tf.estimator.DNNClassifier(feature_columns=feature_columns,
                                        hidden_units=[10, 20, 10],
                                        n_classes=3,
                                        model_dir="/tmp/iris_model")
```

The code above first defines the model's feature columns, which specify the data type for the features in the data set. All the feature data is continuous, so **tf.feature_column.numeric_column** is the appropriate function to use to construct the feature columns. There are four features in the data set (sepal width, sepal height, petal width, and petal height), so accordingly **shape** must be set to **[4]** to hold all the data.

Then, the code creates a **DNNClassifier** model using the following arguments:

- **feature_columns=feature_columns**. The set of feature columns defined above.
- **hidden_units=[10, 20, 10]**. Three hidden layers (http://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw), containing 10, 20, and 10 neurons, respectively.
- **n_classes=3**. Three target classes, representing the three Iris species.
- **model_dir=/tmp/iris_model**. The directory in which TensorFlow will save checkpoint data and TensorBoard summaries during model training.

Describe the training input pipeline

The **tf.estimator** API uses input functions, which create the TensorFlow operations that generate data for the model. We can use **tf.estimator.inputs.numpy_input_fn** to produce the input pipeline:

```
# Define the training inputs
train_input_fn = tf.estimator.inputs.numpy_input_fn(
```

```
    x={"x": np.array(training_set.data)},
    y=np.array(training_set.target),
    num_epochs=None,
    shuffle=True)
```

Fit the DNNClassifier to the Iris Training Data

Now that you've configured your DNN **`classifier`** model, you can fit it to the Iris training data using the **`train`** (https://www.tensorflow.org/api_docs/python/tf/estimator/Estimator?hl=zh-cn#train) method. Pass **`train_input_fn`** as the **`input_fn`**, and the number of steps to train (here, 2000):

```
# Train model.
classifier.train(input_fn=train_input_fn, steps=2000)
```

The state of the model is preserved in the **`classifier`**, which means you can train iteratively if you like. For example, the above is equivalent to the following:

```
classifier.train(input_fn=train_input_fn, steps=1000)
classifier.train(input_fn=train_input_fn, steps=1000)
```

However, if you're looking to track the model while it trains, you'll likely want to instead use a TensorFlow **`SessionRunHook`** (https://www.tensorflow.org/api_docs/python/tf/train/SessionRunHook?hl=zh-cn) to perform logging operations.

Evaluate Model Accuracy

You've trained your **`DNNClassifier`** model on the Iris training data; now, you can check its accuracy on the Iris test data using the **`evaluate`** (https://www.tensorflow.org/api_docs/python/tf/estimator/Estimator?hl=zh-cn#evaluate) method. Like **`train`**, **`evaluate`** takes an input function that builds its input pipeline. **`evaluate`** returns a **`dict`**s with the evaluation results. The following code passes the Iris test data—**`test_set.data`** and **`test_set.target`**—to **`evaluate`** and prints the **`accuracy`** from the results:

```
# Define the test inputs
test_input_fn = tf.estimator.inputs.numpy_input_fn(
    x={"x": np.array(test_set.data)},
    y=np.array(test_set.target),
    num_epochs=1,
    shuffle=False)

# Evaluate accuracy.
accuracy_score = classifier.evaluate(input_fn=test_input_fn)["accuracy"]

print("\nTest Accuracy: {0:f}\n".format(accuracy_score))
```

Note: The **`num_epochs=1`** argument to **`numpy_input_fn`** is important here. **`test_input_fn`** will iterate over the data once, and then raise **`OutOfRangeError`**. This error signals the classifier to stop evaluating, so it will evaluate over the input once.

When you run the full script, it will print something close to:

```
Test Accuracy: 0.966667
```

Your accuracy result may vary a bit, but should be higher than 90%. Not bad for a relatively small data set!

Classify New Samples

Use the estimator's **`predict()`** method to classify new samples. For example, say you have these two new flower samples:

Sepal Length	Sepal Width	Petal Length	Petal Width
6.4	3.2	4.5	1.5
5.8	3.1	5.0	1.7

You can predict their species using the **`predict()`** method. **`predict`** returns a generator of dicts, which can easily be converted to a list. The following code retrieves and prints the class predictions:

```
# Classify two new flower samples.
new_samples = np.array(
    [[6.4, 3.2, 4.5, 1.5],
     [5.8, 3.1, 5.0, 1.7]], dtype=np.float32)
predict_input_fn = tf.estimator.inputs.numpy_input_fn(
    x={"x": new_samples},
    num_epochs=1,
    shuffle=False)

predictions = list(classifier.predict(input_fn=predict_input_fn))
predicted_classes = [p["classes"] for p in predictions]

print(
    "New Samples, Class Predictions:    {}\n"
    .format(predicted_classes))
```

Your results should look as follows:

```
New Samples, Class Predictions:    [1 2]
```

The model thus predicts that the first sample is *Iris versicolor*, and the second sample is *Iris virginica*.

Additional Resources

- To learn more about using tf.estimator to create linear models, see Large-scale Linear Models with TensorFlow (https://www.tensorflow.org/tutorials/linear?hl=zh-cn).
- To build your own Estimator using tf.estimator APIs, check out Creating Estimators in tf.estimator (https://www.tensorflow.org/extend/estimators?hl=zh-cn).
- To experiment with neural network modeling and visualization in the browser, check out Deep Playground (http://playground.tensorflow.org/?hl=zh-cn).
- For more advanced tutorials on neural networks, see Convolutional Neural Networks (https://www.tensorflow.org/tutorials/deep_cnn?hl=zh-cn) and Recurrent Neural Networks (https://www.tensorflow.org/tutorials/recurrent?hl=zh-cn).

上次更新日期：十一月 2, 2017