RUBIK'S CODE Freedom. Wisdom. Excellence.

Introduction to TensorFlow – With Python Example

FEBRUARY 5, 2018 — 5 COMMENTS

Code that accompanies this article can be downloaded **here**.

Last week I presented to you my side-project – **Simple Neural Network in C#**. Now, as I mentioned in that article, the solution presented there is light years away from the optimal solution. More math and matrix multiplication should be done in order for this solution to come anywhere close to anything that can be professionally used. Lucky for us, smart people at Google created a library that does just that – TensorFlow. This is a widely popular opensource library that excels at numerical computing, which is as you figured out so far, essential for our neural network calculations. It has a massive set of application interfaces for most major languages used in deep learning field in general.

So, how TensorFlow works? Well, for starters their whole solution is revolving around tensors, primitive unit in TensorFlow. TensorFlow uses a tensor data structure to represent all data. In math, tensors are geometric objects that describe linear relations between other geometric objects. In TesnsorFlow they are multi-dimensional array or data, ie. matrixes. Ok, it's not as simple as that, but this is whole tensor concept goes deeper in linear algebra that I'd like to go to right now. Anyhow, we can observe tensors as n-dimensional arrays using which matrix operations are done easily and effectively. For example, in the code below, we defined two constant tensors and add one value to another:

```python
import tensorflow as tf

const1 = tf.constant([[1,2,3], [1,2,3]]);
const2 = tf.constant([[3,4,5], [3,4,5]]);

result = tf.add(const1, const2);

with tf.Session() as sess:
  output = sess.run(result)
  print(output)
```

tf_example.py hosted with ❤ by GitHub view raw

The constants, as you already figured out, are values that don't change. However, TensorFlow has rich API, which is well **documented** and using it we can define other types of data, like variables:

```python
import tensorflow as tf

var1 = tf.Variable([[1, 2], [1, 2]], name="variable1")
var2 = tf.Variable([[3, 4], [3, 4]], name="variable2")

result = tf.matmul(var1, var2)

with tf.Session() as sess:
  output = sess.run(result)
  print(output)
```

tf_variable_example.py hosted with ❤ by **GitHub** **view raw**

Apart from tensors, TensorFlow uses data flow graphs. Nodes in the graph represent mathematical operations, while edges represent the tensors communicated between them.

Installation and Setup

TensorFlow provides APIs for a wide range of languages, like Python, C++, Java, Go, Haskell and R (in a form of a third-party library). Also, it supports different types of operating systems. In this article, we are going to use Python on Windows 10 so only installation process on this platform will be covered. TensorFlow supports only **Python 3.5 and 3.6**, so make sure that you one of those versions installed on your system. For other operating systems and languages you can check **official installation guide**. Another thing we need to know is hardware configuration of our system. There are two options for installing TensorFlow:

- TensorFlow with CPU support only.
- TensorFlow with GPU support.

If your system has an NVIDIA® GPU then you can install TensorFlow with GPU support. Of course, GPU version is faster, but CPU is easier to install and to configure.

If you are using **Anaconda** installing TensorFlow can be done following these steps:

1. Create a conda environment "tensorflow" by running the command:

```
conda create -n tensorflow pip python=3.5
```

2. Activate created environment by issuing the command:

   ```
   activate tensorflow
   ```

3. Invoke the command to install TensorFlow inside your environment. For the CPU version run this command:

   ```
   pip install --ignore-installed --upgrade tensorflow
   ```

   For GPU version run the command:

   ```
   pip install --ignore-installed --upgrade tensorflow-gpu
   ```

Of course, you can install TensorFlow using "native pip", too. For the CPU version run:

```
pip3 install --upgrade tensorflow
```

For GPU TensorFlow version run the command:

```
pip3 install --upgrade tensorflow-gpu
```

Cool, now we have our TensorFlow installed. Let's run through the problem we are going to solve.

Iris Data Set Classification Problem

Iris Data Set, along with the **MNIST dataset**, is probably one of the best-known datasets to be found in the pattern recognition literature. It is sort of “Hello World” example for machine learning classification problems. It was first introduced by Ronald Fisher back in 1936. He was British statistician and botanist and he used this example in this paper *The use of multiple measurements in taxonomic problems,* which is often referenced to this day. The dataset contains 3 classes of 50 instances each. Each class refers to one type of iris plant: *Iris setosa*, *Iris virginica,* and *Iris versicolor*. First class is linearly separable from the other two, but the latter two are not linearly separable from each other. Each record has five attributes:

- Sepal length in cm
- Sepal width in cm
- Petal length in cm
- Petal width in cm
- Class (*Iris setosa*, *Iris virginica, Iris versicolor*)

The goal of the neural network, we are going to create is to predict the class of the Iris flower based on other attributes. Meaning it needs to create a **model**, which is going to describe a relationship between attribute values and the class.

TensorFlow Workflow

Most of the TensorFlow codes follow this workflow:

- Import the dataset
- Extend dataset with additional columns to describe the data
- Select the type of model
- Training
- Evaluate accuracy of the model
- Predict results using the model

If you followed my previous **blog posts**, one could notice that training and evaluating processes are important parts of developing any Artificial Neural Network. These processes are usually done on two datasets, one for training and other for testing the accuracy of the trained network. Often, we get just one set of data, that we need to split into two separate datasets and that use one for training and other for testing. The ratio is usually 80% to 20%. This time this is already done for us. You can download training set and test set with code that accompanies this article from **here**.

Code

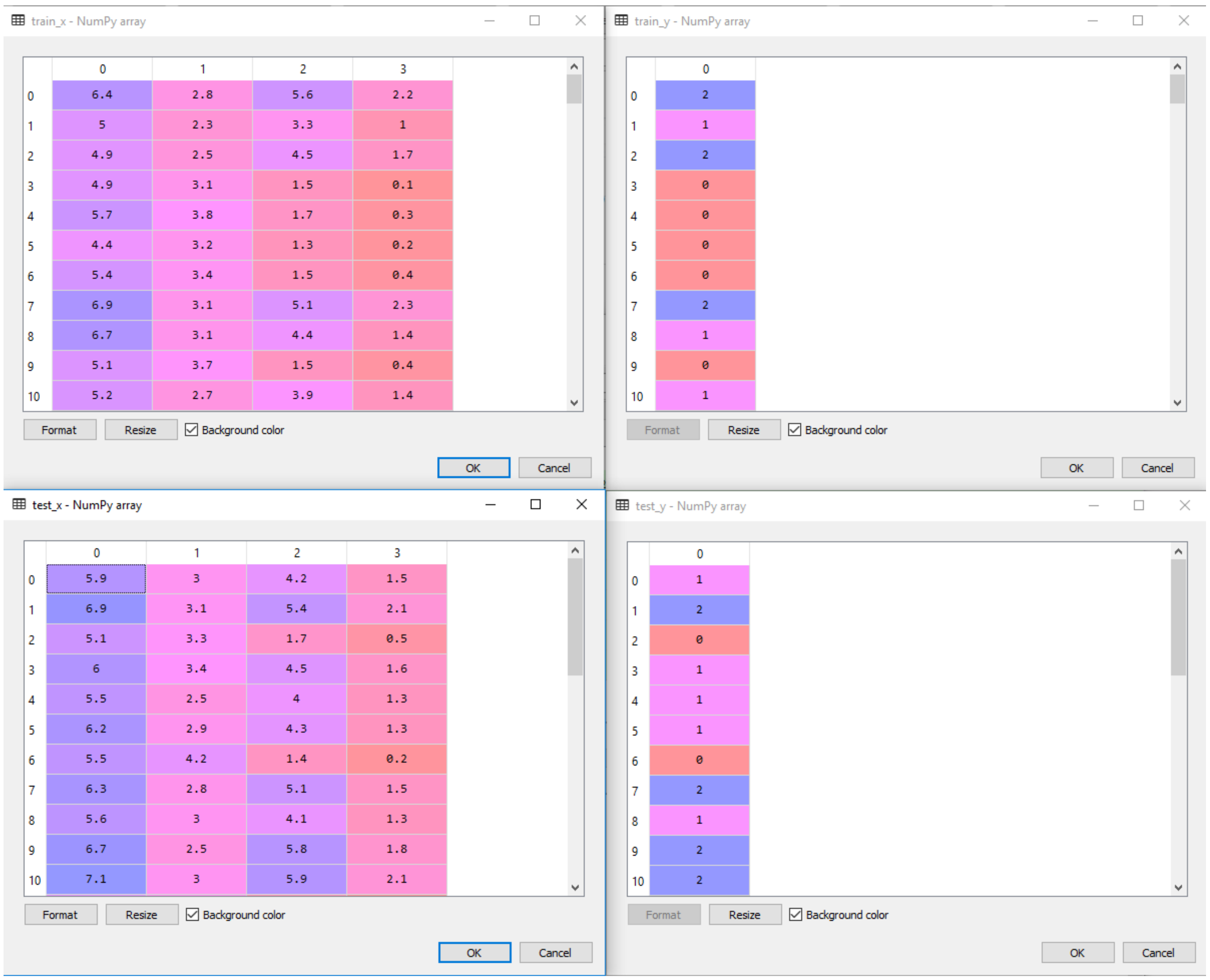
Before we continue, I need to mention that I use **Spyder IDE** for development so I will explain the whole process using this environment. Let's dive in!

The first thing we need to do is to import the dataset and to parse it. For this, we are going to use another Python library – *Pandas*. This is another open source library that provides easy to use data structures and data analysis tools for the Python.

```
# Import `tensorflow` and `pandas`
import tensorflow as tf
import pandas as pd

COLUMN_NAMES = [
        'SepalLength',
        'SepalWidth',
        'PetalLength',
        'PetalWidth',
        'Species'
        ]

# Import training dataset
training_dataset = pd.read_csv('iris_training.csv', names=COLUMN_NAMES, header=0)
train_x = training_dataset.iloc[:, 0:4]
train_y = training_dataset.iloc[:, 4]

# Import testing dataset
test_dataset = pd.read_csv('iris_test.csv', names=COLUMN_NAMES, header=0)
test_x = test_dataset.iloc[:, 0:4]
test_y = test_dataset.iloc[:, 4]
```

import_data.py hosted with ❤ by **GitHub** **view raw**

As you can see, first we used *read_csv* function to import the dataset into local variables, and then we separated inputs *(train_x, test_x)* and expected outputs *(train_y, test_y)* creating four separate matrixes. Here is how they look like:



train_x - NumPy array

	0	1	2	3
0	6.4	2.8	5.6	2.2
1	5	2.3	3.3	1
2	4.9	2.5	4.5	1.7
3	4.9	3.1	1.5	0.1
4	5.7	3.8	1.7	0.3
5	4.4	3.2	1.3	0.2
6	5.4	3.4	1.5	0.4
7	6.9	3.1	5.1	2.3
8	6.7	3.1	4.4	1.4
9	5.1	3.7	1.5	0.4
10	5.2	2.7	3.9	1.4

train_y - NumPy array

	0
0	2
1	1
2	2
3	0
4	0
5	0
6	0
7	2
8	1
9	0
10	1

test_x - NumPy array

	0	1	2	3
0	5.9	3	4.2	1.5
1	6.9	3.1	5.4	2.1
2	5.1	3.3	1.7	0.5
3	6	3.4	4.5	1.6
4	5.5	2.5	4	1.3
5	6.2	2.9	4.3	1.3
6	5.5	4.2	1.4	0.2
7	6.3	2.8	5.1	1.5
8	5.6	3	4.1	1.3
9	6.7	2.5	5.8	1.8
10	7.1	3	5.9	2.1

test_y - NumPy array

	0
0	1
1	2
2	0
3	1
4	1
5	1
6	0
7	2
8	1
9	2
10	2

Great! We prepared data that is going to be used for training and for testing. Now, we need to define feature columns, that are going to help our Neural Network.

```python
# Setup feature columns
columns_feat = [
    tf.feature_column.numeric_column(key='SepalLength'),
    tf.feature_column.numeric_column(key='SepalWidth'),
    tf.feature_column.numeric_column(key='PetalLength'),
    tf.feature_column.numeric_column(key='PetalWidth')
]
```

feature_columns.py hosted with ❤ by **GitHub** **view raw**

We now need to choose model we are going to use. In our problem, we are trying to predict a class of Iris Flower based on the attributes data. That is why we are going to choose one of the estimators from the TensorFlow API. An object of the Estimator class encapsulates the logic that builds a TensorFlow graph and runs a TensorFlow session. For this purpose, we are going to use *DNNClassifier.* We are going to add two hidden layers with ten neurons in each.

```python
# Build Neural Network - Classifier
classifier = tf.estimator.DNNClassifier(
    feature_columns=columns_feat,
    # Two hidden layers of 10 nodes each.
    hidden_units=[10, 10],
    # The model is classifying 3 classes
    n_classes=3)
```

building_classifier.py hosted with ❤ by **GitHub** **view raw**

After that, we will train our neural network with the data we picked from the training dataset. Firstly, we will define training function. This function needs to supply neural network with data from the training set by extending it and creating multiple batches. Training works best if the training examples are in random order. That is why the *shuffle* function has been called. To sum it up, *train_function* creates batches of data using passed training dataset, by randomly picking data from it and supplying it back to *train* method of *DNNClassifier.*

```python
# Define train function
def train_function(inputs, outputs, batch_size):
    dataset = tf.data.Dataset.from_tensor_slices((dict(inputs), outputs))
    dataset = dataset.shuffle(1000).repeat().batch(batch_size)
    return dataset.make_one_shot_iterator().get_next()

# Train the Model.
classifier.train(
    input_fn=lambda:train_function(train_x, train_y, 100),
    steps=1000)
```

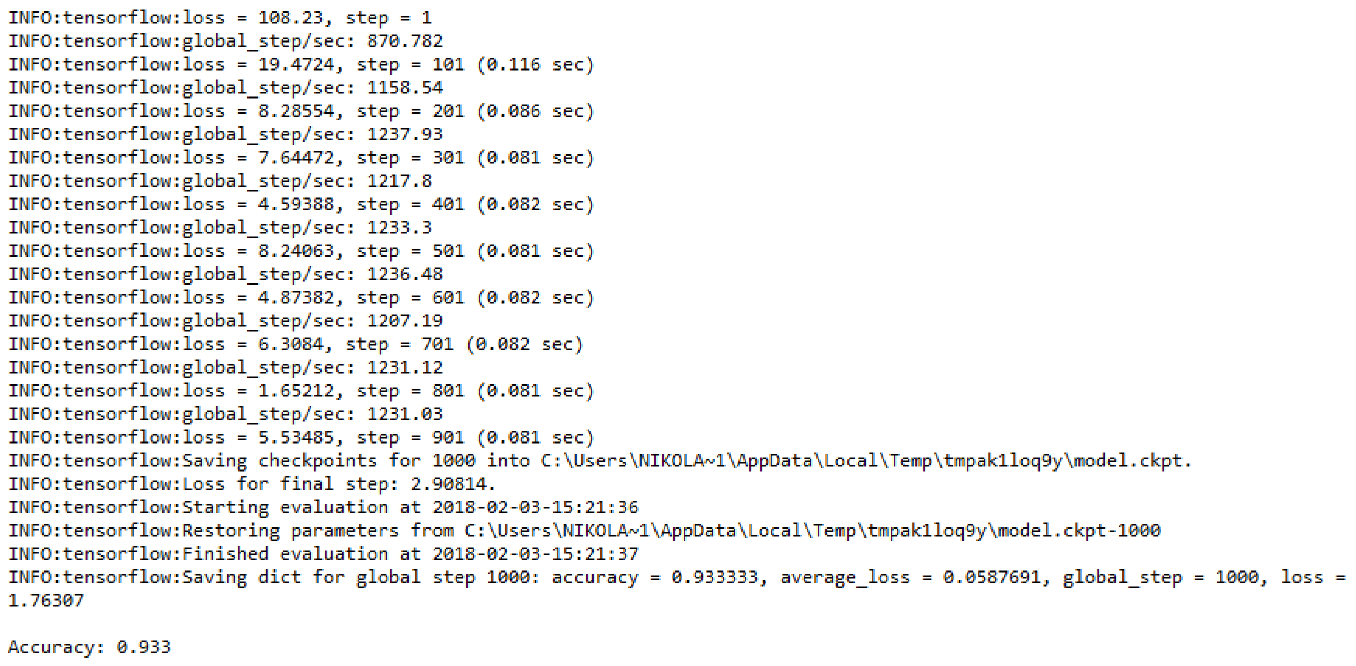
training.py hosted with ❤ by **GitHub** **view raw**

Finally, we call *evaluate* function that will evaluate our neural network and give us back accuracy of the network.

```python
# Define evaluation function
def evaluation_function(attributes, classes, batch_size):
    attributes=dict(attributes)
    if classes is None:
        inputs = attributes
    else:
        inputs = (attributes, classes)
    dataset = tf.data.Dataset.from_tensor_slices(inputs)
    assert batch_size is not None, "batch_size must not be None"
    dataset = dataset.batch(batch_size)
    return dataset.make_one_shot_iterator().get_next()

# Evaluate the model.
eval_result = classifier.evaluate(
    input_fn=lambda:evaluation_function(test_x, test_y, 100))
```

evaluation.py hosted with ❤ by **GitHub** **view raw**

When we run this code I've got these results:



```
INFO:tensorflow:loss = 108.23, step = 1
INFO:tensorflow:global_step/sec: 870.782
INFO:tensorflow:loss = 19.4724, step = 101 (0.116 sec)
INFO:tensorflow:global_step/sec: 1158.54
INFO:tensorflow:loss = 8.28554, step = 201 (0.086 sec)
INFO:tensorflow:global_step/sec: 1237.93
INFO:tensorflow:loss = 7.64472, step = 301 (0.081 sec)
INFO:tensorflow:global_step/sec: 1217.8
INFO:tensorflow:loss = 4.59388, step = 401 (0.082 sec)
INFO:tensorflow:global_step/sec: 1233.3
INFO:tensorflow:loss = 8.24063, step = 501 (0.081 sec)
INFO:tensorflow:global_step/sec: 1236.48
INFO:tensorflow:loss = 4.87382, step = 601 (0.082 sec)
INFO:tensorflow:global_step/sec: 1207.19
INFO:tensorflow:loss = 6.3084, step = 701 (0.082 sec)
INFO:tensorflow:global_step/sec: 1231.12
INFO:tensorflow:loss = 1.65212, step = 801 (0.081 sec)
INFO:tensorflow:global_step/sec: 1231.03
INFO:tensorflow:loss = 5.53485, step = 901 (0.081 sec)
INFO:tensorflow:Saving checkpoints for 1000 into C:\Users\NIKOLA~1\AppData\Local\Temp\tmpak1loq9y\model.ckpt.
INFO:tensorflow:Loss for final step: 2.90814.
INFO:tensorflow:Starting evaluation at 2018-02-03-15:21:36
INFO:tensorflow:Restoring parameters from C:\Users\NIKOLA~1\AppData\Local\Temp\tmpak1loq9y\model.ckpt-1000
INFO:tensorflow:Finished evaluation at 2018-02-03-15:21:37
INFO:tensorflow:Saving dict for global step 1000: accuracy = 0.933333, average_loss = 0.0587691, global_step = 1000, loss =
1.76307

Accuracy: 0.933
```

So, I got the accuracy of 0.93 for my neural network, which is pretty good. After this, we can call our classifier using single data and get predictions for it.

Conclusion

Neural networks have been around for a long time and almost all important concepts were introduced back to 1970s or 1980s. The problem that was stopping the whole field to take off was that back then we had no powerful computers and GPUs to run these kinds of processes. Now, not only we can do that, but Google made Neural Networks popular by making this great tool – TensorFlow publically available. Today we have other higher-level APIs that simplify implementation of neural networks even further. Some of them run on top of the TensorFlow, like Keras. In **next article**, you can see how to implement neural network using this high-level API.

Thanks for reading!

This article is a part of Artificial Neural Networks Series, which you can check out **here**.

Read more posts from the author at **Rubik's Code**.
