CSEE 5590 - Special Topics  
 Deep Learning – Lab 1

**Author:**   
Kalyan Kilaru  
Class Id: 22

**Configuration:**  
IDE: pycharm  
python: version 3  
  
**Objective:** The task is to use TesnsorFlow for implementing Logistic Regression and using TensorBoard to plot it.   
 **Introduction:**  
The main objective of this lab assignment is to get familiar with the TensorFlow and TensorBoard. After doing a little research online for examples that are related to Logistic Regression for Python 3 was that Pycharm was used initially. But for TensorBoard, Anaconda was determined to be the best software to run it.

**Approach:**

Data set from examples is selected and logistic regression model was trained on this dataset.

**Datasets:**

The data set used was the input\_data set from the Tensorflow example tutorials.

**Evaluation & Discussion:**

The following code was used to run:

import tensorflow as tf

# Import MNIST data

from tensorflow.examples.tutorials.mnist import input\_data

mnist = input\_data.read\_data\_sets("/tmp/data/", one\_hot=True)

# Parameters

learning\_rate = 0.01

training\_epochs = 25

b\_size = 100

display\_step = 1

# tf Graph Input

x = tf.placeholder(tf.float32, [None, 785])

y = tf.placeholder(tf.float32, [None, 10])

W = tf.Variable(tf.zeros([784, 10]))

b = tf.Variable(tf.zeros([10]))

# Model

predict = tf.nn.softmax(tf.matmul(x, W) + b) # Softmax

# Minimize error

cost = tf.reduce\_mean(-tf.reduce\_sum(y\*tf.log(pred), reduction\_indices=1))

# Gradient Descent

optimizer = tf.train.GradientDescentOptimizer(learning\_rate).minimize(cost)

# Initialize variables (i.e. assign their default value)

init = tf.global\_variables\_initializer()

# Start training

with tf.Session() as sess:

# initialization

sess.run(init)

writer = tf.summary.FileWriter('./graphs/linear\_reg', sess.graph)

# Train the model

for item in range(training\_epochs):

a\_cost = 0.

batch = int(mnist.train.num\_examples/batch\_size)

# Loop all batchs

for i in range(batch):

xi, yi = mnist.train.next\_batch( batch\_size)

# optimization (backprop) and cost op (to get loss value)

\_, c = sess.run([optimizer, cost], feed\_dict={x: xi,

y: yi})

# Average loss computation

avg\_cost += c / batc

if (epoch+1) % display\_step == 0:

print("Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(avg\_cost))

writer.close()

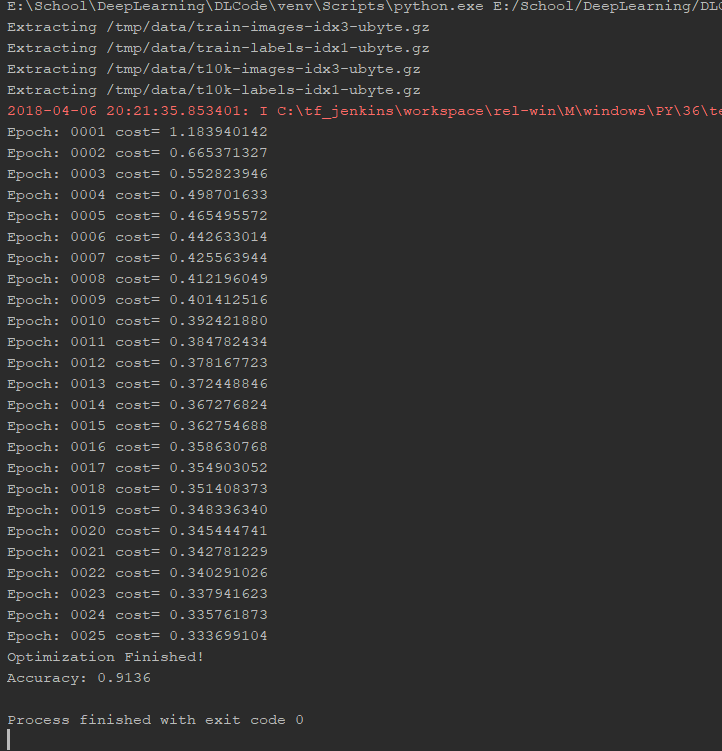
# Testing model

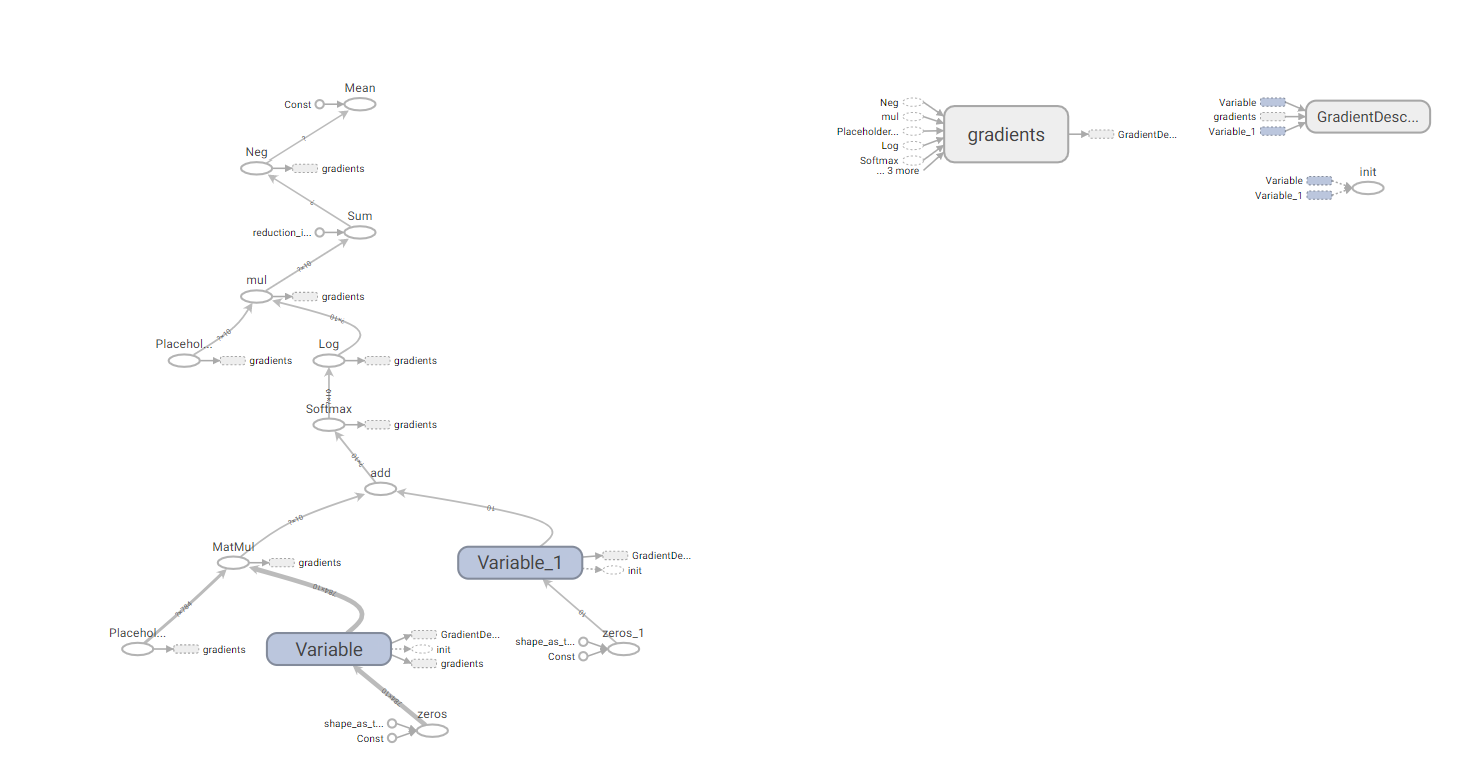
correct\_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))

# Calculate accuracy

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

The code ran in PyCharm and the screehots are attached below.



Tensorboard: 

**Conclusion:**

Tensorflow is very efficient in handling very large datsets. The graph from tensorboard is displayed and model accuracy is increased using gradient descent.