## HU Extension Assignment 08 E63 Big Data Analytics

### Handed out: 03/24/2017 Due by 9:30AM EST on Saturday, 04/01/2017

You are welcome to implement TensorFlow problems in this problem set in any of supported languages.

All my comments and results in blue color.

**Problem 1.** Install newest release of TensorFlow 1.0.1 on the operating system of your choice. Use installation instructions on <https://www.tensorflow.org> site and instructions in the attached files: install-mac-docket.md, install-mac-native.md, install-ubuntu.md and install-win-native.md. Unless you know what you are doing do not install TensorFlow for GPU. Install TensorFlow for CPU. Use attach Jupyter notebook: 0\_test\_install.ipynb to demonstrate that TensorFlow is properly installed. **(10%)**

Installed TensorFlow on a Ubuntu machine using instructions at install.ubuntu.md

Once installed, I open Jupyter web browser end-point, upload 0\_test\_install.ipynb and run it. The below screenshot shows python notebook execution’s result:

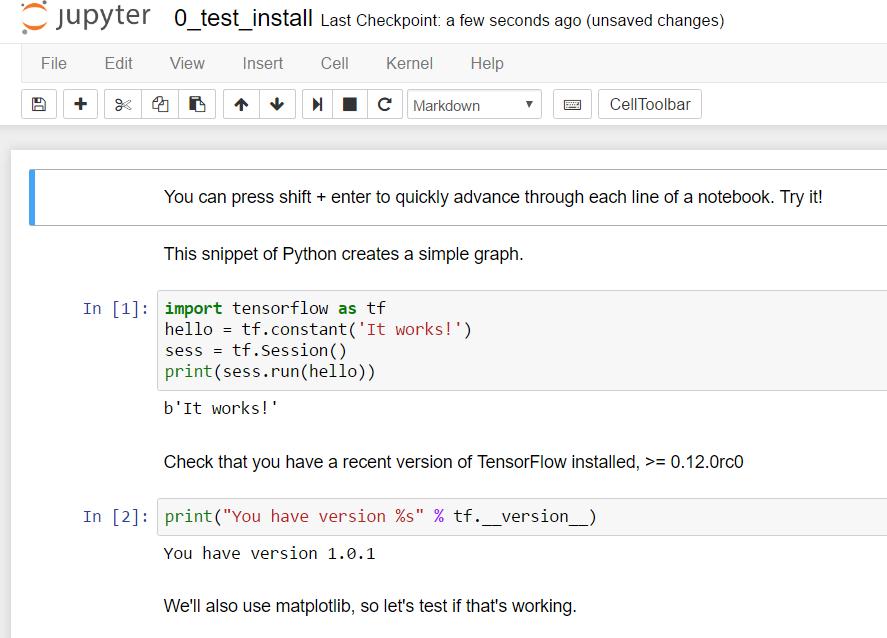


Figure 1 - Details of the provided Jupyter notebook

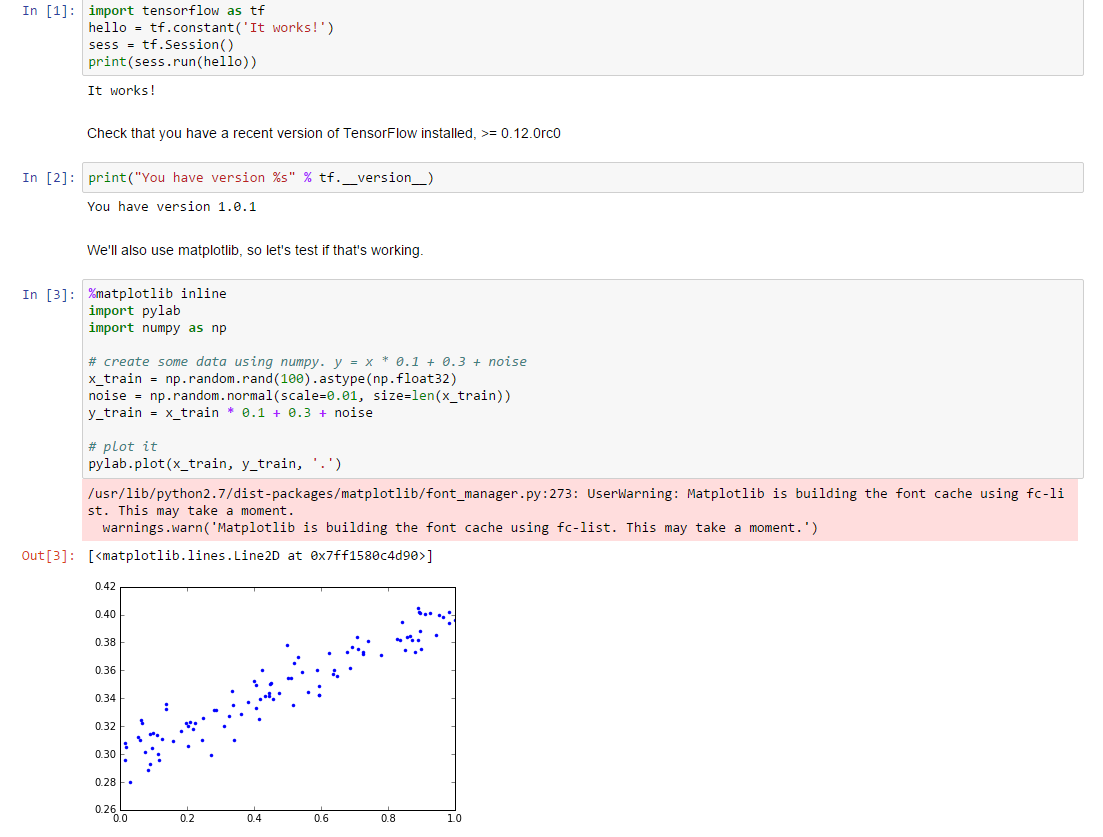


Figure 2 - Notebook complete execution

**Problem 2.** Construct a simple neural network (a network of logistic units) which will implement (X1 XOR X2) AND X3 function. Choose weights (-s) of all dendritic inputs and bias inputs. Demonstrate that your network works by presenting the truth table. Present your network by a simple graph. You can produce the graph in any way convenient including pan and paper**. (15%)**

First step is to decompose the implementation in AND OR and NOT logical gates, as reflected below:

(X1 XOR X2) AND X3 = ((X1 AND NOT X2) OR (NOT X1 AND X2)) AND X3

(A1(2) OR A2(2)) AND X3 = A1(3) AND A2(3) = H0(X)

This expression is described in he below neural network graph:



Figure 3 - Neural network grahp

This neural network produces the below truth table:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **x1** | **x2** | **X3** | **a1(2)** | **a2(2)** | **a3(2)** | **a1(3)** | **a2(3)** | **h0(x)** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **0** |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | **0** |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | **0** |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | **0** |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | **1** |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | **0** |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | **1** |
| 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | **0** |

**Problem 3.** Calculate the first 30 Fibonacci numbers using TensorFlow API and recurrence relationship:

,

with seed values: .

This is the complete maths for the Fibbonaci sequence:

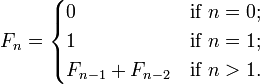


Figure 4 Fibonacci Sequence maths

To help you out, we are attaching Jupyter notebook entitled: 1\_warm\_up.ipynb which contains Fibonacci calculation based on the products of 2x2 matrixes of the form. Export TensorFlow graph crated in this problem to a directory of your choice. Read and display that graph using TensorBoard tool. When starting the TensorBoard it appears that you need to specify the full directory path to the log directory, like:

C:\> tensorboard –-logdir=E:/code/output

**(25%)**

According to this, this can be implemented recursively in Tensorflow with Python according to the below code:

|  |
| --- |
| import tensorflow as tf  #2 first values of the sequence 0 and 1. They constitute the seed values F(0) and F(1)  f = [tf.constant(0),tf.constant(1)]  #for positions 2 to 29 each number is calculated by adding the 2 previous in the sequence.  for i in range(2,30):  fib\_i = f[i-1] + f[i-2]  f.append(fib\_i)    with tf.Session() as sess:  result = sess.run(f)  print result |

And the result:

|  |
| --- |
| [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229] |

To save Tensorflow graph to a new folder named problem\_3, add:

|  |
| --- |
| tf.summary.FileWriter ("problem\_3", sess.graph) |

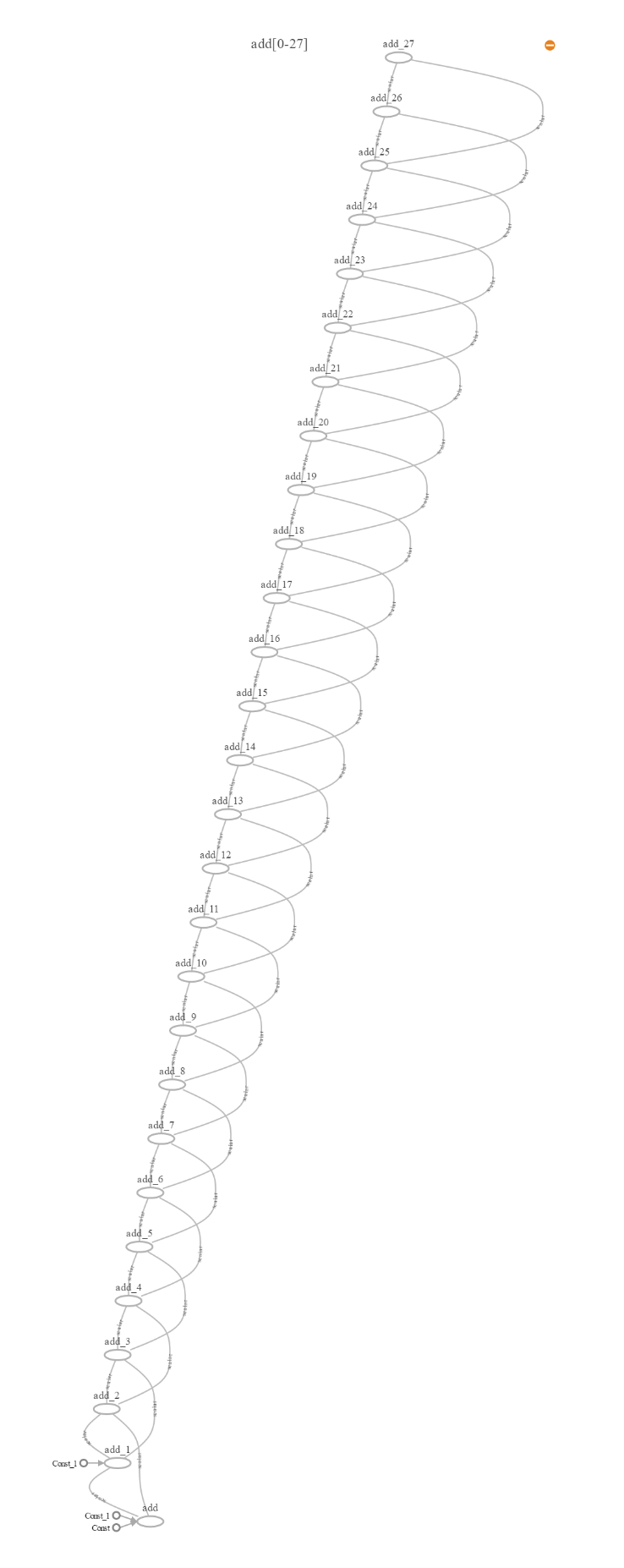


Figure 5Fibonacci recursive calculation graph

**Problem 4.** Please examine attached Jupyter notebook 2\_linear\_regression.ipynb. As you are running its cells, the notebook will complain about non-existent API calls. This notebook was written in an earlier version of TensorFlow API and some calls changed their names. Fix all code by replacing older calls with calls in TF 1.0.1.

All errors are related to deprecated method that raise “no existing module” error. It is needed to substitute this methods with the updated ones corresponding to TF 1.0.0 version. Once fixed code runs smoothly.

This is the complete script once errors are fixed:

|  |
| --- |
| # coding: utf-8  # This notebook contains code that demonstrates linear regression. Uncomment lines to learn more.  # In[1]:  # Import tensorflow and other libraries.  from \_\_future\_\_ import absolute\_import  from \_\_future\_\_ import division  from \_\_future\_\_ import print\_function  import tensorflow as tf  import numpy as np  import math  get\_ipython().magic(u'matplotlib inline')  import pylab  # In[2]:  sess = None  def resetSession():  tf.reset\_default\_graph()  global sess  if sess is not None: sess.close()  sess = tf.InteractiveSession()  # In[4]:  resetSession()  # Create input data using NumPy. y = x \* 0.1 + 0.3 + noise  x\_train = np.random.rand(100).astype(np.float32)  noise = np.random.normal(scale=0.01, size=len(x\_train))  y\_train = x\_train \* 0.1 + 0.3 + noise  # Uncomment the following line to plot our input data.  # pylab.plot(x\_train, y\_train, '.')  # In[5]:  # Create some fake evaluation data  x\_eval = np.random.rand(len(x\_train)).astype(np.float32)  noise = np.random.normal(scale=0.01, size=len(x\_train))  y\_eval = x\_eval \* 0.1 + 0.3 + noise  # In[6]:  # Build inference graph.  # Create Variables W and b that compute y\_data = W \* x\_data + b  W = tf.Variable(tf.random\_normal([1]), name='weights')  b = tf.Variable(tf.random\_normal([1]), name='bias')  # Uncomment the following lines to see what W and b are.  # print(W)  # print(b)  # Create a placeholder we'll use later to feed x's into the graph for training and eval.  # shape=[None] means we can put in any number of examples.  # This is used for minibatch training, and to evaluate a lot of examples at once.  x = tf.placeholder(shape=[None], dtype=tf.float32, name='x')  # Uncomment this line to see what x is  # print(x)  # This is the same as tf.add(tf.mul(W, x), b), but looks nicer  y = W \* x + b  # At this point, we have:  # \* x\_train: x input features  # \* y\_train: observed y for each x that we will train on  # \* x\_eval, y\_eval: Same as above, but a smaller set that we will not train on, and instead evaluate our effectiveness.  # In[9]:  # Write the graph so we can look at it in TensorBoard  # https://www.tensorflow.org/versions/r0.12/how\_tos/summaries\_and\_tensorboard/index.html  sw = tf.summary.FileWriter('summaries/', graph=tf.get\_default\_graph())  # In[10]:  # Create a placeholder we'll use later to feed the correct y value into the graph  y\_label = tf.placeholder(shape=[None], dtype=tf.float32, name='y\_label')  # print (y\_label)  # In[11]:  # Build training graph.  loss = tf.reduce\_mean(tf.square(y - y\_label)) # Create an operation that calculates loss.  optimizer = tf.train.GradientDescentOptimizer(0.5) # Create an optimizer.  train = optimizer.minimize(loss) # Create an operation that minimizes loss.  # Uncomment the following 3 lines to see what 'loss', 'optimizer' and 'train' are.  # print("loss:", loss)  # print("optimizer:", optimizer)  # print("train:", train)  # In[13]:  # Create an operation to initialize all the variables.  init = tf.global\_variables\_initializer()  print(init)  sess.run(init)  # In[14]:  # Uncomment the following line to see the initial W and b values.  # print(sess.run([W, b]))  # In[15]:  # Uncomment these lines to test that we can compute a y from an x (without having trained anything).  # x must be a vector, hence [3] not just 3.  # x\_in = [3]  # sess.run(y, feed\_dict={x: x\_in})  # In[16]:  # Calculate loss on the evaluation data before training  def eval\_loss():  return sess.run(loss, feed\_dict={x: x\_eval, y\_label: y\_eval})  eval\_loss()  # In[22]:  # Track of how loss changes, so we can visualize it in TensorBoard  tf.summary.scalar('loss', loss)  summary\_op = tf.summary.merge\_all()  # In[23]:  # Perform training.  for step in range(201):  # Run the training op; feed the training data into the graph  summary\_str, \_ = sess.run([summary\_op, train], feed\_dict={x: x\_train, y\_label: y\_train})  sw.add\_summary(summary\_str, step)  # Uncomment the following two lines to watch training happen real time.  # if step % 20 == 0:  # print(step, sess.run([W, b]))  # In[24]:  # Uncomment the following lines to plot the predicted values  # pylab.plot(x\_train, y\_train, '.', label="target")  # pylab.plot(x\_train, sess.run(y, feed\_dict={x: x\_train, y\_label: y\_train}), label="predicted")  # pylab.legend()  # In[25]:  # Check accuracy on eval data after training  eval\_loss()  # Demonstrate saving and restoring a model  # In[26]:  def predict(x\_in): return sess.run(y, feed\_dict={x: [x\_in]})  # In[34]:  # Save the model  saver = tf.train.Saver()  saver.save(sess, 'my\_checkpoint.ckpt')  # In[28]:  # Current prediction  predict(3)  # In[29]:  # Reset the model by running the init op again  sess.run(init)  # In[30]:  # Prediction after variables reinitialized  predict(3)  # In[40]:  saver.restore(sess, './my\_checkpoint.ckpt')  # In[41]:  # Predictions after variables restored  predict(3) |

Uncomment all optional (print) lines. Provide a copy of this notebook with all intermediate results and the image of TensorFlow graph as captured by the TensorBoard.

Jupyter notebook file has been uploaded to Canvas for your review. File name is **e63\_Assign08\_NN\_TensorFlow\_Problem\_4.ipynb.**

Tensorboard is initiated with the required command from a system prompt:

|  |
| --- |
| tensorboard --logdir ./ --host 0.0.0.0 --port 7777 |

By opening the browser pointing to the provided URL we get access to TensorBoard’s UI

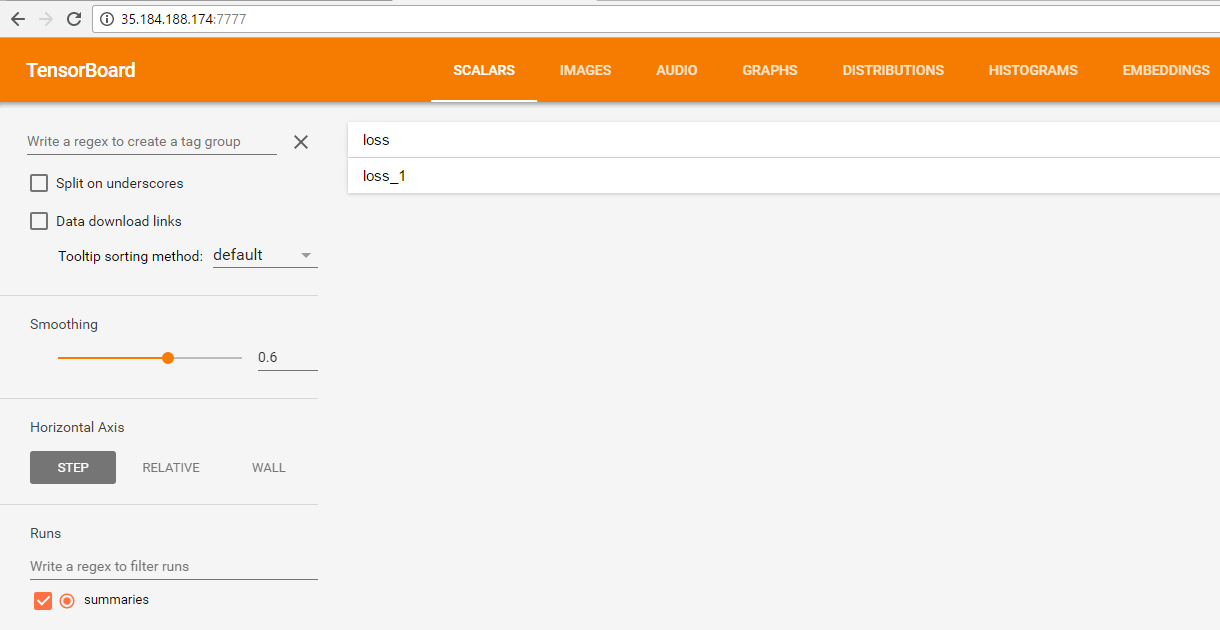


Figure 6 - Detail of TensorBoard's UI

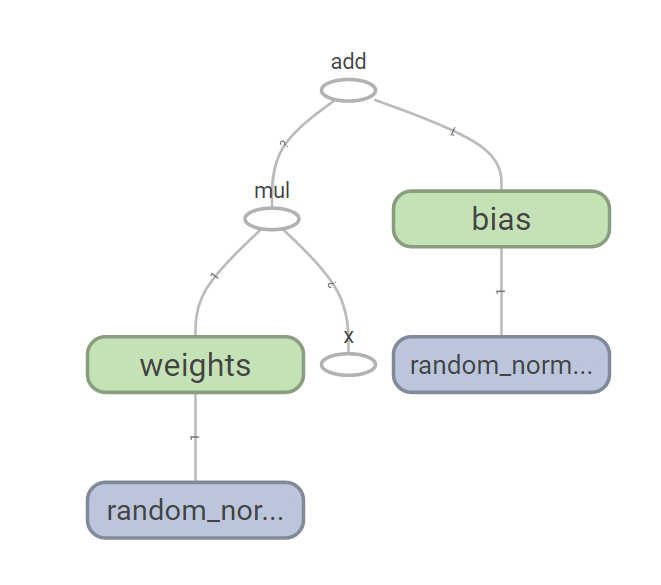


Figure 7 TensorFlow graph for the linear regression script

**(20%)**

**Problem 5.** Please considered attached Excel file called Reduced\_Car\_Data.xlsx. This is the data set we used previously except that we have now removed several descriptive variables and left only: Displacement, Horsepower and Weight. Please build a regression model using TensorFlow that will predict the gasoline consumption (MPG - Miles Per Gallon) of cars based on three remaining variables. Please extract a percentage of data to serve as a training set and a percentage to serve as the test set. Please report on the accuracy of your model.

**(30%)**

I divided the original file into 2 files, 15% of the records for test, 85% for train and saved them as CSV to be consumed from TensorFlow.

This is the code that creates, tests and train the model:

|  |
| --- |
| import tensorflow as tf  import numpy as np  tf.logging.set\_verbosity(tf.logging.ERROR)  #train records  train\_data = np.loadtxt(open("../problem\_5/car\_train.csv"), delimiter=",")  x\_data = train\_data[:,0:3] # first 3 columns are features  y\_data = train\_data[:,3] # last column is the petrol consumption  #test records  test\_data = np.loadtxt(open("../problem\_5/car\_test.csv"), delimiter=",")  x\_test = test\_data[:,0:3] # same columns as in train data  y\_test = test\_data[:,3]  #features  feature\_columns=[tf.contrib.layers.real\_valued\_column('', dimension=3)]  # train and evaluate a linear regression model  #add features to the model  m = tf.contrib.learn.LinearRegressor(feature\_columns)  #first trains the model  m.fit(x\_data, y\_data, steps=200)  #afterwards tests the model  test = m.evaluate(x\_test, y\_test, steps=1)  #prediction for the test data  prediction = m.predict(x\_test)  #compare actual test data with prediction  j=0  sum2ErrAcc = 0  for i in prediction:  sum2Err = (y\_test[j]- i)\*\*2  print str(i) + "-->" + str(y\_test[j]) + " || Sqr Error:" + str(sum2Err)  sum2ErrAcc= sum2ErrAcc + sum2Err  j = j +1  #MSE calculated manually  print "MSE:" + str(sum2ErrAcc/j)  #MSE error directly provided by TensorFlow  print test |

The result shows MSE provided by TensorFlow (loss) and same error calculated manually (actual value 🡪predicted value). Note both values are the same as supposed to be:

|  |
| --- |
| 19.7964-->18.0 || Sqr Error:3.22694356955  18.0981-->15.0 || Sqr Error:9.59806225761  17.555-->18.0 || Sqr Error:0.198048494654  18.6682-->16.0 || Sqr Error:7.11915646266  19.256-->17.0 || Sqr Error:5.08955555275  19.7055-->15.0 || Sqr Error:22.1417628473  17.3688-->14.0 || Sqr Error:11.348551097  18.0553-->14.0 || Sqr Error:16.4457036514  18.1474-->14.0 || Sqr Error:17.2011316246  16.4183-->15.0 || Sqr Error:2.0116686509  28.8258-->14.0 || Sqr Error:219.805361929  24.1941-->16.0 || Sqr Error:67.1436717623  22.8923-->17.0 || Sqr Error:34.7191277581  21.5942-->19.0 || Sqr Error:6.73010194864  19.1997-->20.0 || Sqr Error:0.640532947818  MSE:28.2279587036  {'loss': 28.227959, 'global\_step': 200} |

Please, describe every step of your work and present all intermediate and final results in a Word document. Please, copy past text version of all essential command and snippets of results into the Word document with explanations of the purpose of those commands. We cannot retype text that is in JPG images. Please, always submit a separate copy of the original, working scripts and/or class files you used. Sometimes we need to run your code and retyping is too costly. Please include in your MS Word document only relevant portions of the console output or output files. Sometime either console output or the result file is too long and including it into the MS Word document makes that document too hard to read. PLEASE DO NOT EMBED files into your MS Word document. For issues and comments visit the class Discussion Board.