## HU Extension Assignment 09 E63 Big Data Analytics

### Handed out: 04/01/2017 Due by 9:30AM EST on Saturday, 04/08/2017

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

**Problem 1.** 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.

By modifying the script linear\_regression.py to consider three explanatory values we finally get the below script that as a result prints MSE for the test data (15% of records, the other 85% was used to train the model.).

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  import tensorflow as tf  import xlrd  DATA\_FILE = 'Reduced\_Car\_Data.xlsx'  # Step 1: read in data from the .xls file  book = xlrd.open\_workbook(DATA\_FILE, encoding\_override="utf-8")  sheet = book.sheet\_by\_index(0)  #Train data vector  train\_data = np.asarray([sheet.row\_values(i) for i in range(1, 86)])  n\_train\_samples = len(train\_data)  #test data vector  test\_data = np.asarray([sheet.row\_values(i) for i in range(86, 101)])  # Step 2: create placeholders for input X1, X2, X3(displacement,horsepower and weight of fire) and label Y (gasoline consumption)  X1 = tf.placeholder(tf.float32, name='displacement')  X2 = tf.placeholder(tf.float32, name='horsepower')  X3 = tf.placeholder(tf.float32, name='weight')  Y = tf.placeholder(tf.float32, name='consumption')  # Step 3: create weight and bias, initialized to 0  u = tf.Variable(0.0, name='weights\_displacement')  v = tf.Variable(0.0, name='weights-horsepower')  w = tf.Variable(0.0, name='weights\_weight')  b = tf.Variable(0.0, name='bias')  # Step 4: build model to predict Y  Y\_predicted = X1 \* u + X2 \* v + X3 \* w + b  # Step 5: use the square error as the loss function  loss = tf.square(Y - Y\_predicted, name='loss')  # Step 6: using gradient descent with learning rate of 0.00000001 to minimize loss  optimizer = tf.train.GradientDescentOptimizer(learning\_rate=0.00000001).minimize(loss)  with tf.Session() as sess:  # Step 7: initialize the necessary variables, in this case, w and b  sess.run(tf.global\_variables\_initializer())  writer = tf.summary.FileWriter('./linear\_reg', sess.graph)  # Step 8: train the model  for i in range(100): # train the model 100 times  total\_loss = 0  for x1, x2, x3, y in train\_data:  # Session runs train\_op and fetch values of loss  \_, l = sess.run([optimizer, loss], feed\_dict={X1: x1, X2: x2, X3: x3, Y:y})  total\_loss += l  #print 'Epoch {0}: {1}'.format(i, total\_loss/n\_train\_samples)  # close the writer when you're done using it  writer.close()  # Step 9: output the values of u,v w and b  u\_value, v\_value, w\_value, b\_value = sess.run([u, v, w, b])    #Step 10: Plot predicted value versus actual test data  X1\_TEST, X2\_TEST, X3\_TEST, Y\_TEST = test\_data.T[0], test\_data.T[1], test\_data.T[2], test\_data.T[3]  Y\_MODEL = [X1\_TEST \* u\_value + X2\_TEST \* v\_value + X3\_TEST \* w\_value + b\_value]  #Step 11: Accuracy measure calculation  mse\_test = ((Y\_TEST-Y\_MODEL) \*\* 2).mean()  print "Mean Square Error (MSE): %f" %mse\_test |

The result prints MSE:

|  |
| --- |
| Mean Square Error (MSE): 78.207180 |

**The above problem was problem 5 of the previously released problem set 8. Even if you have submitted your solution for this problem as a part of your solution for problem set 8, please submit it again. You can leave the text of your solution unchanged or modify it. That is up to you.**

**Problem 2.** Consider the attached file linear\_regression.py and the attached data file fire\_theft.xls.

I execute this script from Jupyter notebook, just with some minor changes (epoch = 500 and learning rate =0.000001), aiming to make results compatible with quad an cubic fits.

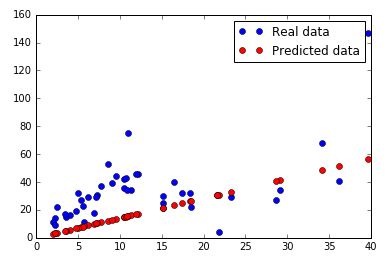


Figure 1 - Fires vs thefts simple linear regression fit

Compare results of this original solution to a solution with new feature quadratic in X (the number of fires), and then with a solution cubic in X. For all three solutions, plot the diagram of predicted values vs. original target values. You can have three different diagrams or you can present all those curves and data on one diagram

As mentioned in Prof. Djordjevic’s lecture, intuitively the best fir seems to be not linear but also quadratic. So, I proceed with adapting the script provided to act as quadratic

The script will be same as previous one with some slight modifications to use quadratic or cubic fit functions.

|  |
| --- |
| # Step 3: create weight and bias, initialized to 0  w = tf.Variable(0.0, name='weights\_1')  u = tf.Variable(0.0, name='weights\_2')  b = tf.Variable(0.0, name='bias')  # Step 4: build model to predict Y  Y\_predicted = X \* X \* w + X \* u + b  # Step 9: output the values of w and b  w\_value, u\_value, b\_value = sess.run([w, u, b])  plt.plot(X, X \* X \* w\_value + X \* u\_value + b\_value, 'ro', label='Predicted data') |

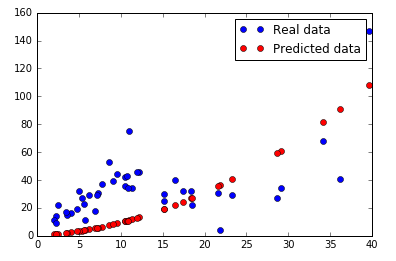


Figure 2 - Fires vs thefts quadratic fit

Similar for the cubic approach:

|  |
| --- |
| # Step 3: create weight and bias, initialized to 0  w = tf.Variable(0.0, name='weights\_1')  u = tf.Variable(0.0, name='weights\_2')  b = tf.Variable(0.0, name='bias')  # Step 4: build model to predict Y  Y\_predicted = X \* X \* w + X \* u + b  # Step 9: output the values of w and b  w\_value, u\_value, b\_value = sess.run([w, u, b])  plt.plot(X, X \* X \* w\_value + X \* u\_value + b\_value, 'ro', label='Predicted data') |

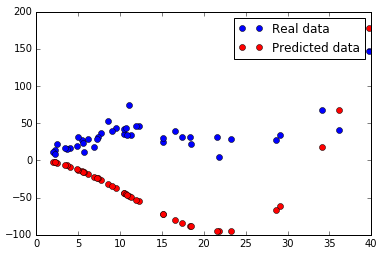


Figure 3 - Fires vs thefts cubic fit

There is an issue with the cubic function if using same learning rate as in linear quad functions, cubic doesn’t work. I have tried different epoch and learning rate values but I haven’t found any valid combination for the three of them. Therefore, finally, I have used same epoch number 500, but a smaller learning rate (0.000000001 versus 0.0000001)

Either way is fine. Perform these calculations using the same set of parameters. Present TensorBoard Graphs for all three solutions and point to any differences between them.

And their corresponding graphs as shown in TensorBoard:

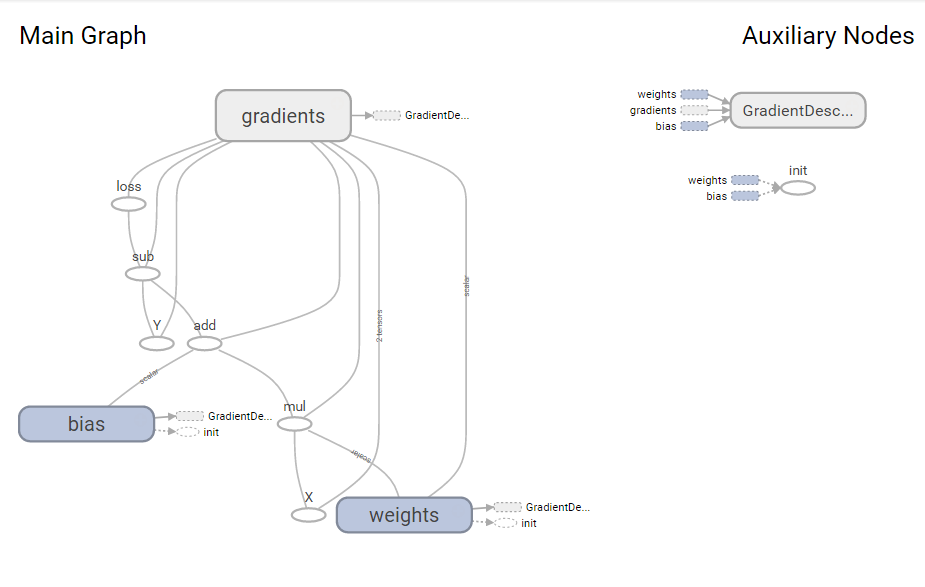


Figure 4 - Simple linear regression graph

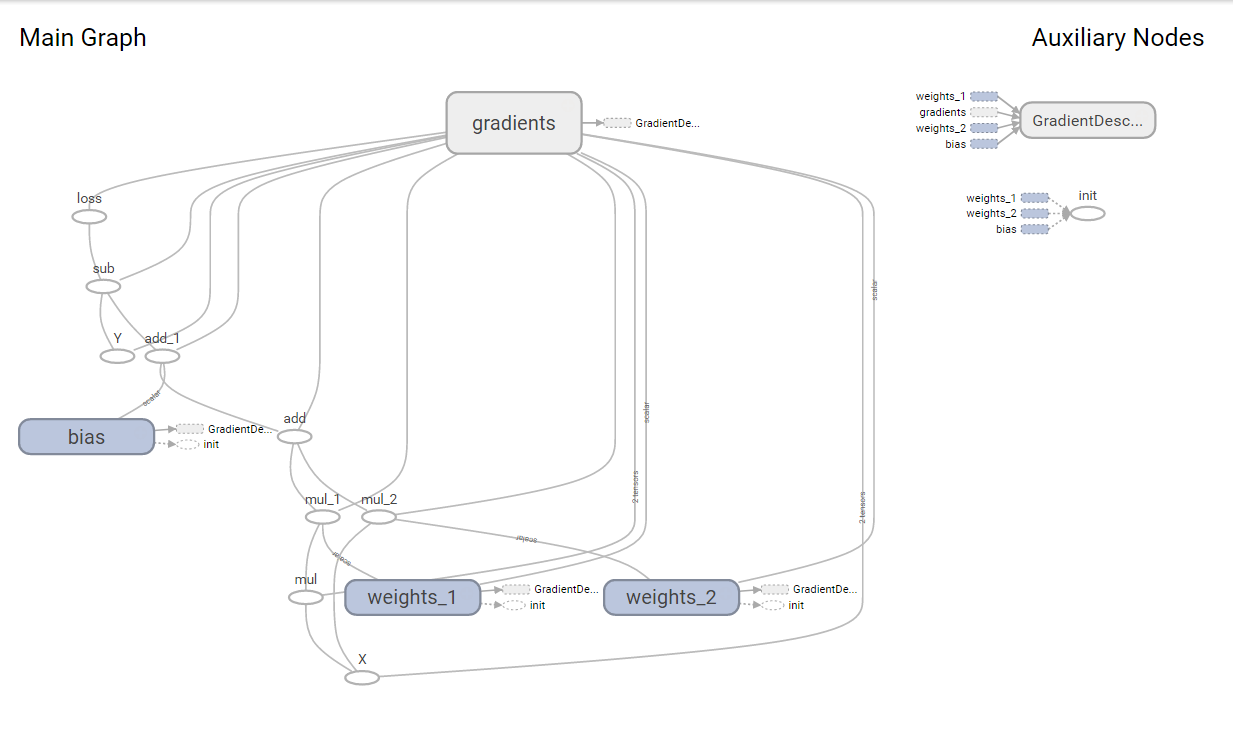


Figure 5 – Quadratic function regression graph

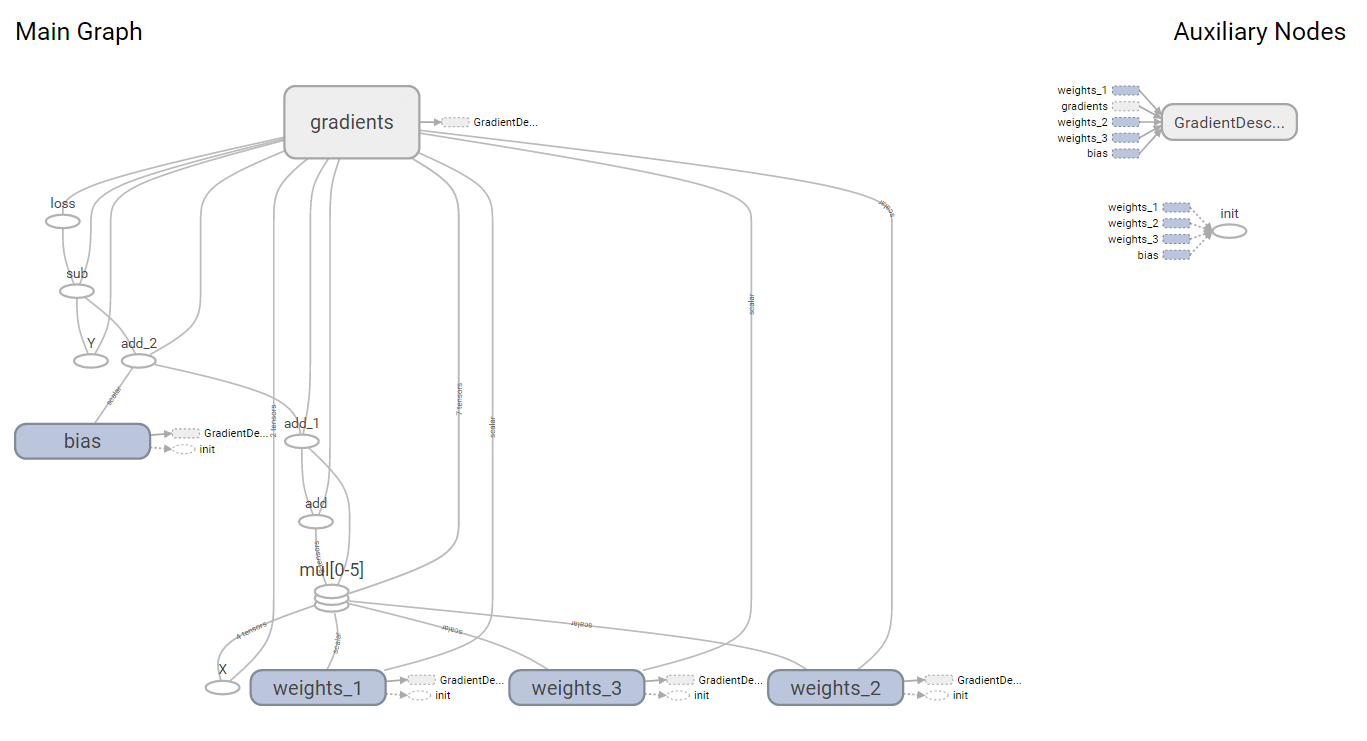


Figure 6 - Cubic function regression graph

The main difference is the number of weights, since we have 1, 2 and 3 weights depending on the fit function, what can be clearly appreciated on the TensorBoard graph. The other 2 main components, gradients and loss, are connected with the weights, since this weights are modified by the gradient to minimize loss’ values. Bias (intercept) remains the same when referring to representation with TensorBoard, just the weights change since number increases with functions’ complexity.

**Problem 3**. Consider the attached file logistic\_regression\_mnist.py. We have stated the results of that program in class but left many details unexplained. Search through TensorFlow API documentation and the Internet and describe for us what is the meaning and purpose of functions used in step 5 and step 6.

**Step 5:** States that the loss function, that the algorithm tries to minimize when later training the model, is based on defined by the mean of the so called entropy. This entropy refers to the value of the probability error in discrete classification, what is the case that occupies us, since our dependent variable is a discrete value. Additionally this measure assumes our dependent variable values are mutually exclusive, that is the case as well, since a number cannot be, for instance, 0 and 1 at the same time.

**Step 6:** Uses gradient descent as the algorithm to minimize loss function defined at Step5. Gradient descent is the function used to minimize the loss function, while the learning rate determines the size of the steps we take to reach a minimum, considering these steps follow the downhill direction of the surface created by the loss function till reaching a valley.

Demonstrate that you can run the code successfully.

This is the result of successfully running the code:

|  |
| --- |
| Successfully downloaded train-images-idx3-ubyte.gz 9912422 bytes.  Extracting ./mnist/train-images-idx3-ubyte.gz  Successfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.  Extracting ./mnist/train-labels-idx1-ubyte.gz  Successfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.  Extracting ./mnist/t10k-images-idx3-ubyte.gz  Successfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.  Extracting ./mnist/t10k-labels-idx1-ubyte.gz  Average loss epoch 0: 1.28781130403  Average loss epoch 1: 0.732461173634  Average loss epoch 2: 0.600318741965  Average loss epoch 3: 0.536900505736  Average loss epoch 4: 0.497908431125  Average loss epoch 5: 0.471158397156  Average loss epoch 6: 0.451184895419  Average loss epoch 7: 0.436043063035  Average loss epoch 8: 0.423492477351  Average loss epoch 9: 0.413101743394  Average loss epoch 10: 0.404410636856  Average loss epoch 11: 0.397051841021  Average loss epoch 12: 0.390328963598  Average loss epoch 13: 0.384493462689  Average loss epoch 14: 0.379063774343  Average loss epoch 15: 0.374666112251  Average loss epoch 16: 0.370301068246  Average loss epoch 17: 0.366608850745  Average loss epoch 18: 0.36317632363  Average loss epoch 19: 0.359884819715  Average loss epoch 20: 0.356741146365  Average loss epoch 21: 0.35361897011  Average loss epoch 22: 0.351067657897  Average loss epoch 23: 0.34885715299  Average loss epoch 24: 0.346559218663  Average loss epoch 25: 0.344361989613  Average loss epoch 26: 0.342270499899  Average loss epoch 27: 0.340135002748  Average loss epoch 28: 0.338477533424  Average loss epoch 29: 0.336555942059  Total time: 18.6766180992 seconds  Optimization Finished!  Accuracy 0.912 |

Fetch for us the TensorBoard Graph.

Below is TensorBoard graph result

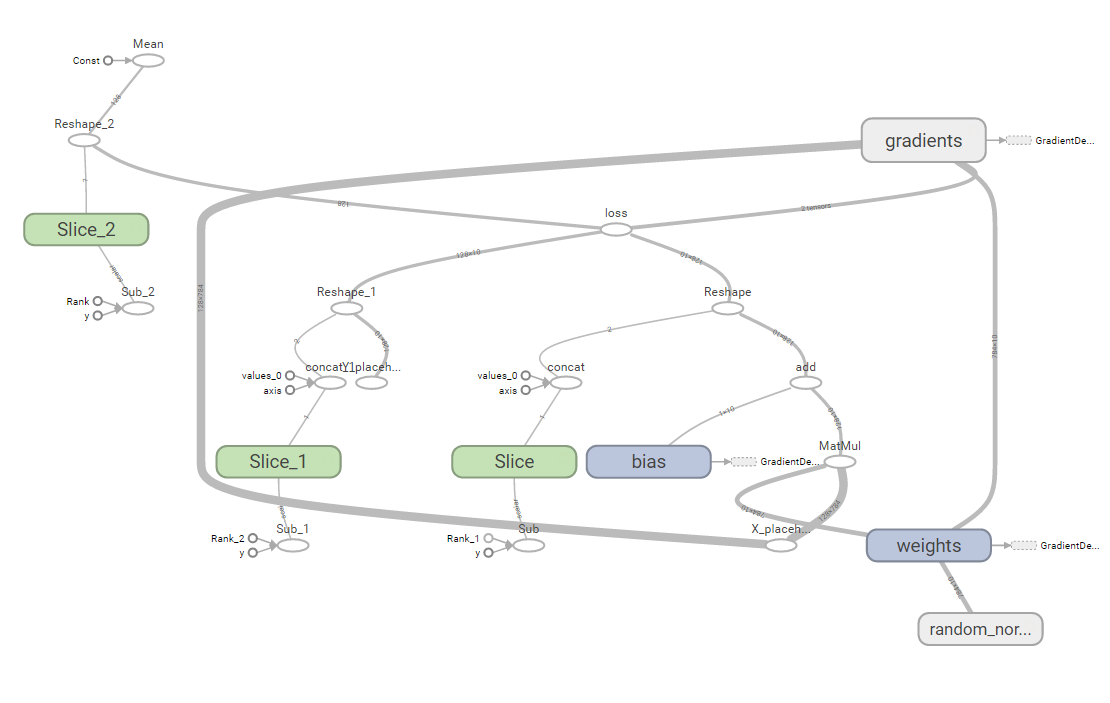


Figure 7 - TensorFlow graph for logistic\_regression\_mnist.py script

Vary parameter batch\_size through values: 8, 64, 128, 256 and report and plot changes in the execution time and accuracy. Keep other parameters the same as in the original program.

We run the script 4 time with the different batch\_size values and create to arrays that contain batch size and accuracy values. Then we plot these to arrays, so we can look at model’s accuracy depending on the batch size value

|  |
| --- |
| Execution Time🡪 [88.28086519241333, 22.751082181930542, 18.608703136444092, 16.622669219970703]  Batch Size 🡪[8, 64, 128, 256]  Accuray 🡪[0.92620000000000002, 0.91659999999999997, 0.91159999999999997, 0.90329999999999999] |

Now we plot the result:

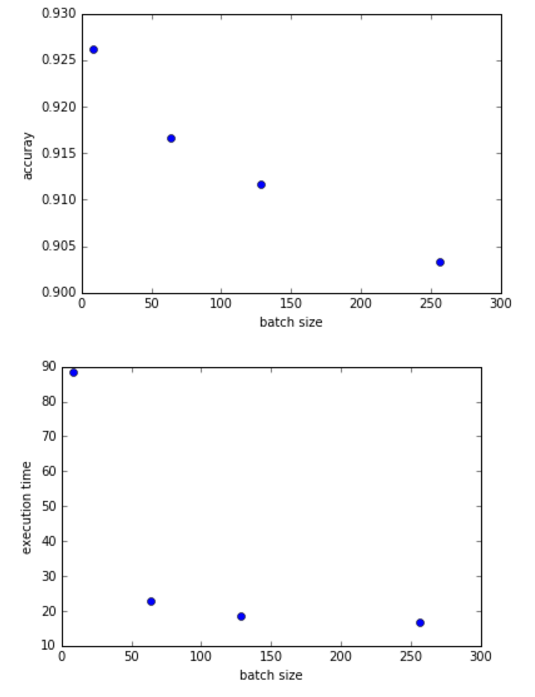


Figure 8 - Plot for different batch sizes

Similarly, vary parameter learning\_rate through values 0.001, 0.005, 0.01, 0.02 and 0.05. Report and plot changes in the execution time and accuracy

Same for the required different learning rates

|  |
| --- |
| Execution Time 🡪[16.53122115135193, 16.4654598236084, 17.512901067733765, 17.307039976119995, 16.571649074554443]  Learning Rate 🡪[0.001, 0.005, 0.01, 0.02, 0.05]  Accuracy 🡪[0.85589999999999999, 0.89290000000000003, 0.90339999999999998, 0.91139999999999999, 0.91820000000000002] |

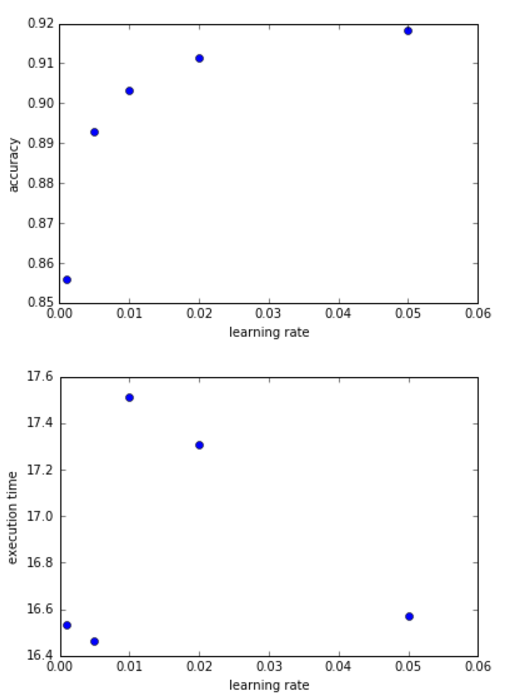


Figure 9 - Plot for different learning rates

.

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.