ECE 521 Assignment 2

# Work Breakdown

|  |  |
| --- | --- |
| Group Member Name | Contribution Percentage |
| Jixong Deng | 33% |
| Jeffrey Kirman | 33% |
| Connor Smith | 33% |

# 1.2 – Effect of hyperparameters

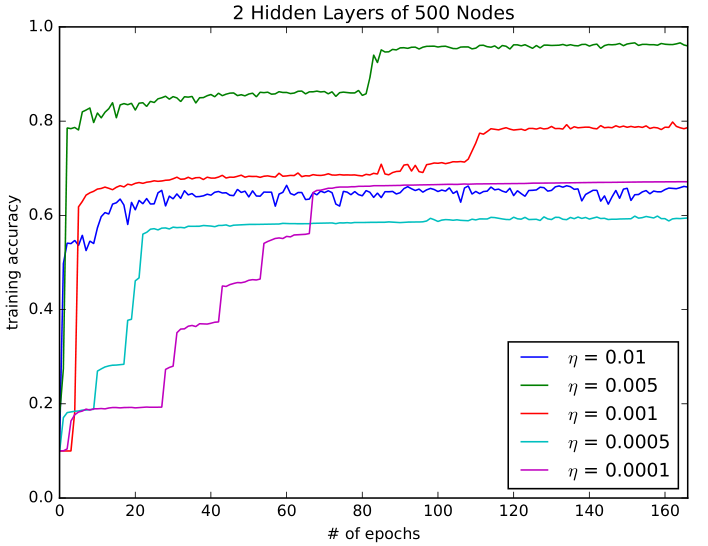
## Part 1 – Number of hidden units

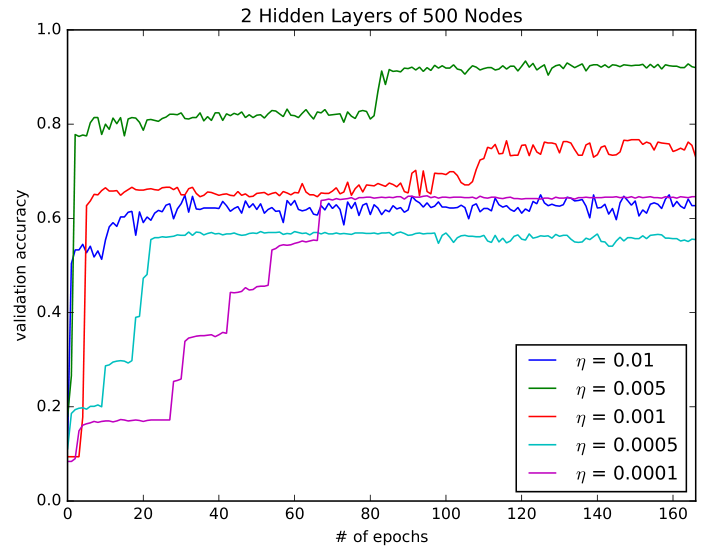
The code used to see the difference in performance for different number of hidden units can be seen in **hyperparameters.py** specifically in the function *number\_of\_hidden\_units()*. The plots showing the training error and accuracies vs iterations (per epoch) can be seen in Appendix A. The following table summarizes the results of tuning the hyperparameters:

|  |  |  |  |
| --- | --- | --- | --- |
| **# of hidden nodes** | **100** | **500** | **1000** |
| **Val. error (1 – acc)** |  |  |  |
| **Test error (1 – acc)** |  |  |  |

In summary, altering the number of hidden layers has little effect on the validation error for a single layer neural network for this dataset, but lower amount of nodes reduces computation time significantly, making it the better choice.

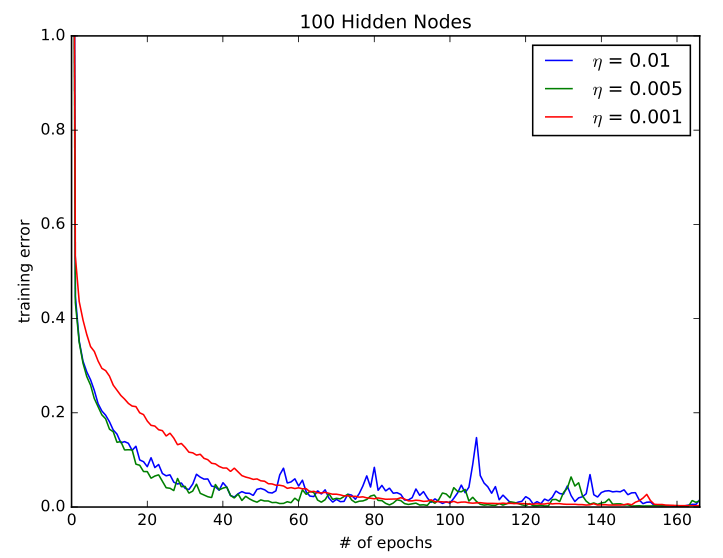
## Part 2 – Number of layers

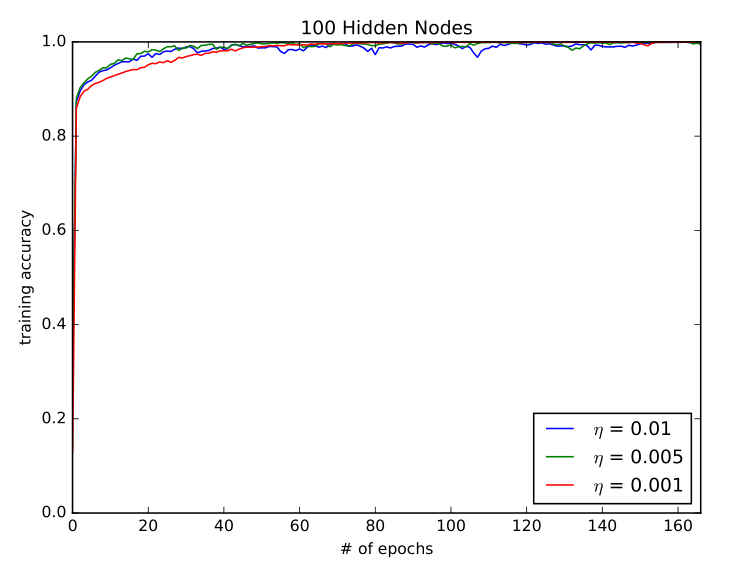


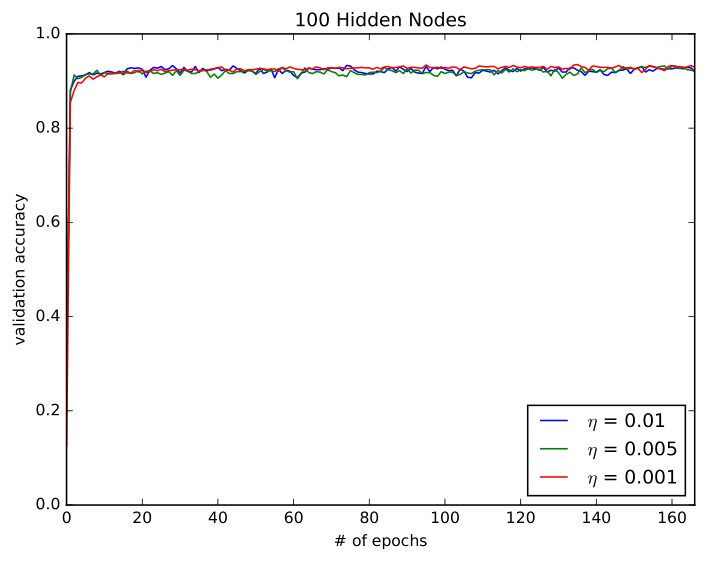


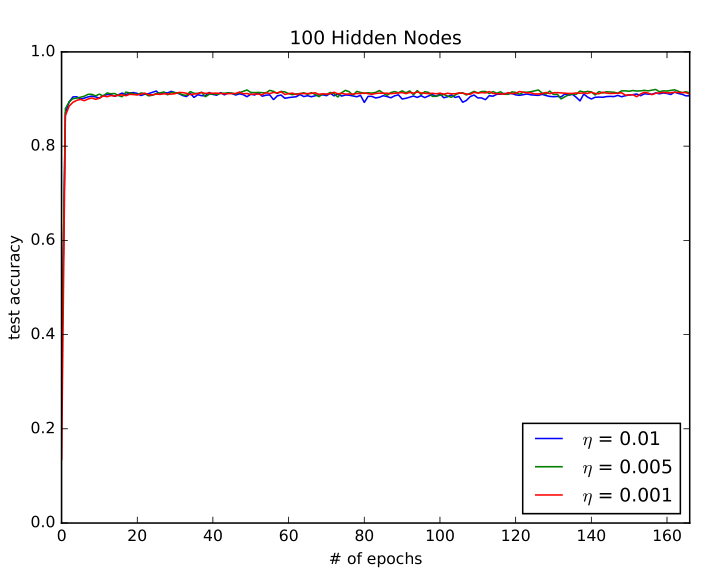
# Appendix A – Graphs

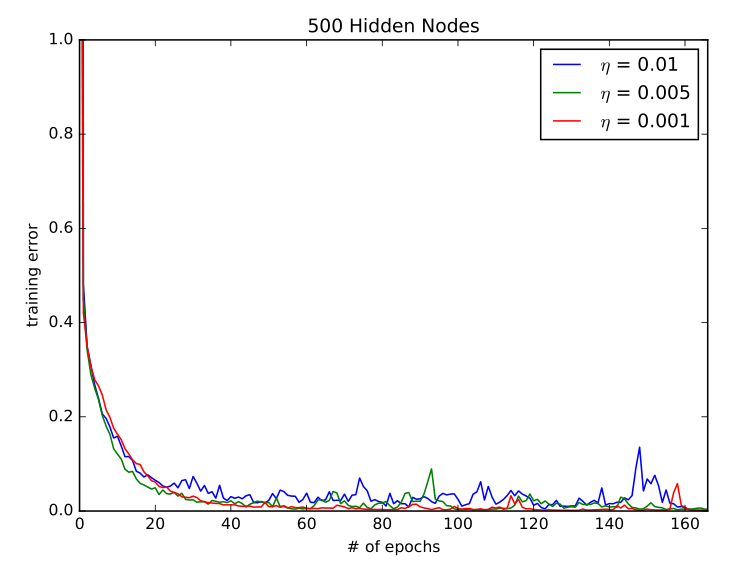
## 1.2.1:

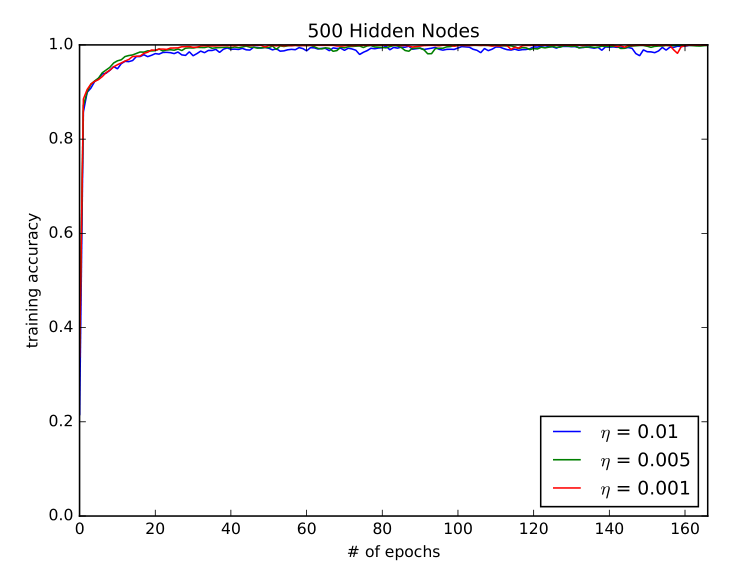


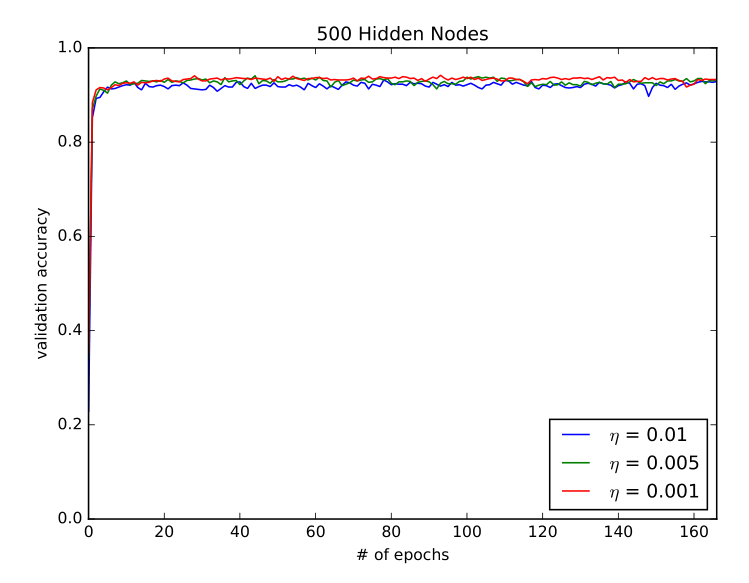


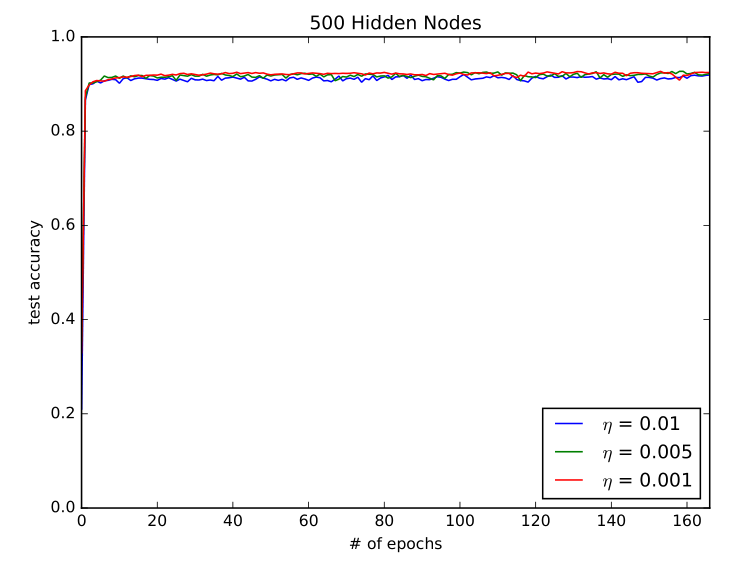


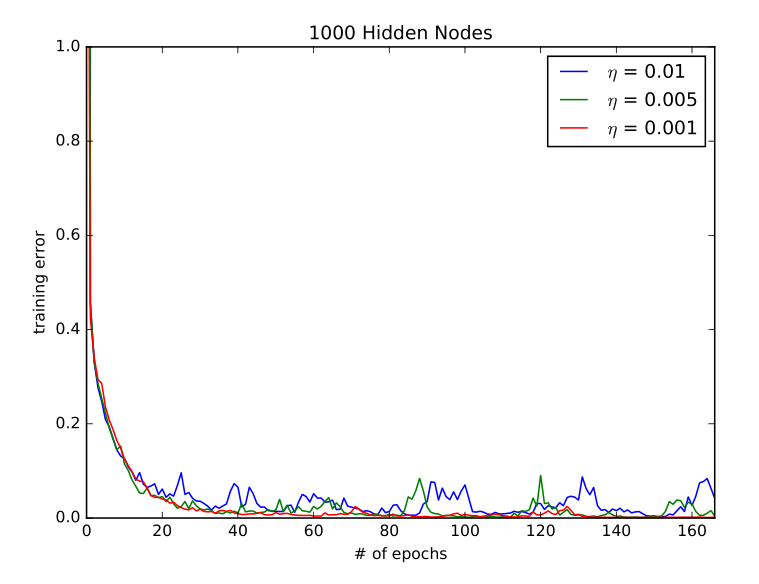


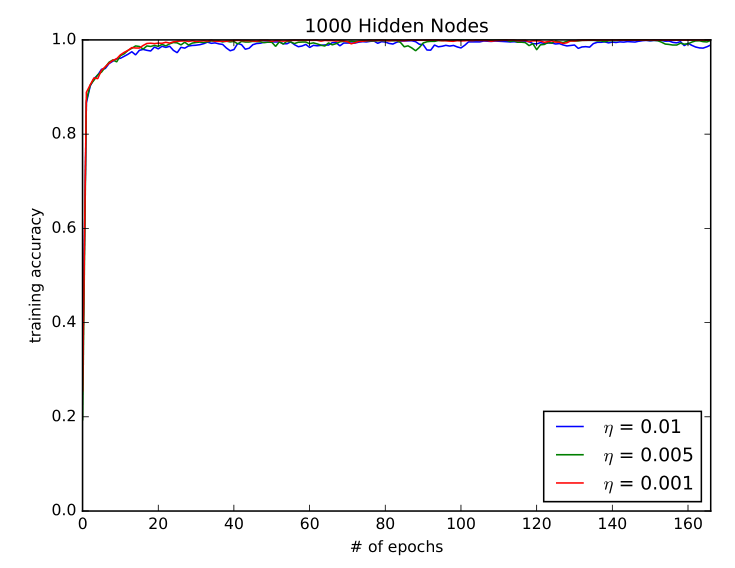


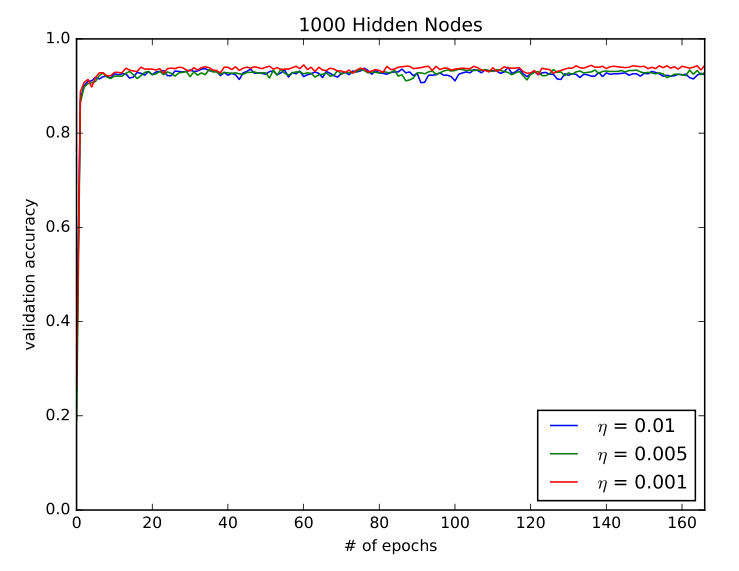


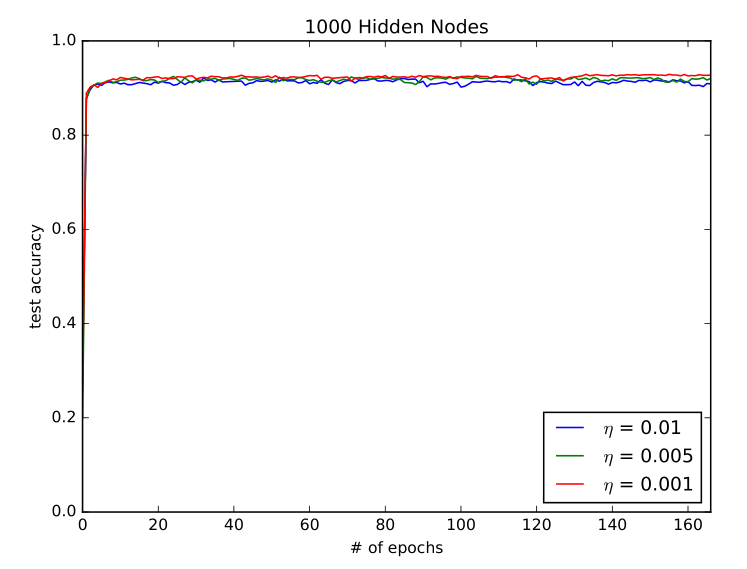












# Appendix B – Python code

## hyperparameters.py

|  |
| --- |
| import numpy as np  import tensorflow as tf  def load\_notMNIST():  with np.load("notMNIST.npz") as data:  Data, Target = data["images"], data["labels"]  np.random.seed(521)  randIndx = np.arange(len(Data))  np.random.shuffle(randIndx)  Data = Data[randIndx]/255  Target = Target[randIndx]  trainData, trainTarget = Data[:15000], Target[:15000]  t = np.zeros((trainTarget.shape[0], 10))  t[np.arange(trainTarget.shape[0]), trainTarget] = 1  trainTarget = t  validData, validTarget = Data[15000:16000], Target[15000:16000]  t = np.zeros((validTarget.shape[0], 10))  t[np.arange(validTarget.shape[0]), validTarget] = 1  validTarget = t  testData, testTarget = Data[16000:], Target[16000:]  t = np.zeros((testTarget.shape[0], 10))  t[np.arange(testTarget.shape[0]), testTarget] = 1  testTarget = t  return (trainData.reshape(trainData.shape[0], -1), trainTarget, validData.reshape(validData.shape[0], -1), validTarget, testData.reshape(testData.shape[0], -1), testTarget)  def create\_new\_layer(input\_tensor, num\_hidden\_units):  '''  @param input\_tensor - outputs of the previous layer in the neural network, without the bias term.  @param num\_hidden\_units - number of hidden units to use for this new layer  '''  # Create the new layer weight matrix using Xavier initialization  input\_dim = int(input\_tensor.shape[-1])  initializer = tf.contrib.layers.xavier\_initializer()  W\_shape = [input\_dim, num\_hidden\_units]  W = tf.get\_variable("W", initializer=initializer(W\_shape), dtype=tf.float32)  # todo: zero initializer?  b = tf.get\_variable("b", shape=[1, num\_hidden\_units], dtype=tf.float32)  # MatMul the extended input tensor by the new weight matrix and add the biases  output\_tensor = tf.matmul(input\_tensor, W) + b  # Return this operation  return output\_tensor  def number\_of\_hidden\_units():  # Constants  B = 500  iters = 5000  learning\_rates = [0.01, 0.005, 0.001]  hidden\_units = [100,500,1000]  output\_data = [[],[],[]]    # Load data  (trainData, trainTarget, validData, validTarget,  testData, testTarget) = load\_notMNIST()    # Precalculations  num\_iters\_per\_epoch = len(trainData)//B # number of iterations we have to do for one epoch  print("Num epochs = ",iters/num\_iters\_per\_epoch)  inds = np.arange(trainData.shape[0])    # Set place-holders & variables  X = tf.placeholder(tf.float32, shape=(None, trainData.shape[-1]), name='X')  Y = tf.placeholder(tf.float32, shape=(None, 10), name='Y')  learning\_rate = tf.placeholder(tf.float32, name='learning-rate')    for h in range(0, len(hidden\_units)):  for lr in range(len(learning\_rates)):  # Build graph  with tf.variable\_scope("layer1\_"+str(hidden\_units[h])+"\_"+str(lr), reuse=tf.AUTO\_REUSE):  s\_1 = create\_new\_layer(X, hidden\_units[h])  x\_1 = tf.nn.relu(s\_1)  with tf.variable\_scope("layer2\_"+str(hidden\_units[h])+"\_"+str(lr), reuse=tf.AUTO\_REUSE):  s\_2 = create\_new\_layer(x\_1, 10)  x\_2 = tf.nn.softmax(s\_2)    # Calculate loss & accuracy  loss = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(logits=s\_2, labels=Y))  accuracy = tf.reduce\_mean(tf.cast(tf.equal(tf.argmax(x\_2, 1), tf.argmax(Y, 1)), tf.float32))    print("Number of hidden units", hidden\_units[h])    with tf.Session() as sess:  with tf.variable\_scope("default", reuse=tf.AUTO\_REUSE):  optimizer = tf.train.AdamOptimizer(learning\_rate).minimize(loss)  coord = tf.train.Coordinator()  threads = tf.train.start\_queue\_runners(sess=sess, coord=coord)  sess.run(tf.global\_variables\_initializer())  sess.run(tf.local\_variables\_initializer())  print("Learning rate = ",learning\_rates[lr])  temp\_output = []  for i in range(iters):  if (i % num\_iters\_per\_epoch == 0):  np.random.shuffle(inds)  sess.run([optimizer], feed\_dict={learning\_rate: learning\_rates[lr],  X: trainData[inds[B\*(i%num\_iters\_per\_epoch):B\*((i+1)%num\_iters\_per\_epoch)]],  Y: trainTarget[inds[B\*(i%num\_iters\_per\_epoch):B\*((i+1)%num\_iters\_per\_epoch)]]})  if (i % num\_iters\_per\_epoch == 0):  t\_loss, t\_acc = sess.run([loss, accuracy], feed\_dict={X: trainData, Y: trainTarget})  v\_loss, v\_acc = sess.run([loss, accuracy], feed\_dict={X: validData, Y: validTarget})  test\_loss, test\_acc = sess.run([loss, accuracy], feed\_dict={X: testData, Y: testTarget})  print("Epoch: {}, Training Loss: {}, Accuracies: [{}, {}, {}]".format(i//num\_iters\_per\_epoch, t\_loss, t\_acc, v\_acc, test\_acc))  temp\_output.append([t\_loss, t\_acc, v\_acc, test\_acc])  output\_data[h].append(temp\_output)    np.save('Q1-2-1.npy', output\_data)  return output\_data  def number\_of\_layers():  # Constants  B = 250  iters = 5000  learning\_rates = [0.01, 0.005, 0.001, 0.0005, 0.0001]  hidden\_units = [500]  output\_data = [[]]    # Load data  (trainData, trainTarget, validData, validTarget,  testData, testTarget) = load\_notMNIST()    # Precalculations  num\_iters\_per\_epoch = len(trainData)//B # number of iterations we have to do for one epoch  print("Num epochs = ",iters/num\_iters\_per\_epoch)  inds = np.arange(trainData.shape[0])    # Set place-holders & variables  X = tf.placeholder(tf.float32, shape=(None, trainData.shape[-1]), name='X')  Y = tf.placeholder(tf.float32, shape=(None, 10), name='Y')  learning\_rate = tf.placeholder(tf.float32, name='learning-rate')    for h in range(0, len(hidden\_units)):  for lr in range(len(learning\_rates)):  # Build graph  with tf.variable\_scope("layer1\_"+str(hidden\_units[h])+"\_"+str(lr), reuse=tf.AUTO\_REUSE):  s\_1 = create\_new\_layer(X, hidden\_units[h])  x\_1 = tf.nn.relu(s\_1)  with tf.variable\_scope("layer2\_"+str(hidden\_units[h])+"\_"+str(lr), reuse=tf.AUTO\_REUSE):  s\_2 = create\_new\_layer(x\_1, hidden\_units[h])  x\_2 = tf.nn.softmax(s\_2)  with tf.variable\_scope("layer3\_"+str(hidden\_units[h])+"\_"+str(lr), reuse=tf.AUTO\_REUSE):  s\_3 = create\_new\_layer(x\_2, 10)  x\_3 = tf.nn.softmax(s\_3)    # Calculate loss & accuracy  loss = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(logits=s\_3, labels=Y))  accuracy = tf.reduce\_mean(tf.cast(tf.equal(tf.argmax(x\_3, 1), tf.argmax(Y, 1)), tf.float32))    print("Number of hidden layers: 2, Number of hidden units", hidden\_units[h])    with tf.Session() as sess:  with tf.variable\_scope("default", reuse=tf.AUTO\_REUSE):  optimizer = tf.train.AdamOptimizer(learning\_rate).minimize(loss)  coord = tf.train.Coordinator()  threads = tf.train.start\_queue\_runners(sess=sess, coord=coord)  sess.run(tf.global\_variables\_initializer())  sess.run(tf.local\_variables\_initializer())  print("Learning rate = ",learning\_rates[lr])  temp\_output = []  for i in range(iters):  if (i % num\_iters\_per\_epoch == 0):  np.random.shuffle(inds)  sess.run([optimizer], feed\_dict={learning\_rate: learning\_rates[lr],  X: trainData[inds[B\*(i%num\_iters\_per\_epoch):B\*((i+1)%num\_iters\_per\_epoch)]],  Y: trainTarget[inds[B\*(i%num\_iters\_per\_epoch):B\*((i+1)%num\_iters\_per\_epoch)]]})  if (i % num\_iters\_per\_epoch == 0):  t\_loss, t\_acc = sess.run([loss, accuracy], feed\_dict={X: trainData, Y: trainTarget})  v\_loss, v\_acc = sess.run([loss, accuracy], feed\_dict={X: validData, Y: validTarget})  test\_loss, test\_acc = sess.run([loss, accuracy], feed\_dict={X: testData, Y: testTarget})  print("Epoch: {}, Training Loss: {}, Accuracies: [{}, {}, {}]".format(i//num\_iters\_per\_epoch, t\_loss, t\_acc, v\_acc, test\_acc))  temp\_output.append([t\_loss, t\_acc, v\_acc, test\_acc])  output\_data[h].append(temp\_output)    np.save('Q1-2-2.npy', output\_data)  return output\_data    #output = number\_of\_hidden\_units()  output = number\_of\_layers() |