**mlp.py**

**import** **numpy** **as** **np**

**import** **tensorflow** **as** **tf**  
**from** **sklearn.metrics** **import** roc\_auc\_score, accuracy\_score  
s = tf.InteractiveSession()

## Defining various initialization parameters for 784-512-256-10 MLP model  
num\_classes = y\_train.shape[1]  
num\_features = X\_train.shape[1]  
num\_output = y\_train.shape[1]  
num\_layers\_0 = 512  
num\_layers\_1 = 256  
starter\_learning\_rate = 0.001  
regularizer\_rate = 0.1

# Placeholders for the input data  
input\_X = tf.placeholder('float32',shape =(**None**,num\_features),name="input\_X")  
input\_y = tf.placeholder('float32',shape = (**None**,num\_classes),name='input\_Y')

## for dropout layer  
keep\_prob = tf.placeholder(tf.float32)

## Weights initialized by random normal function with std\_dev = 1/sqrt(number of input features)  
weights\_0 = tf.Variable(tf.random\_normal([num\_features,num\_layers\_0], stddev=(1/tf.sqrt(float(num\_features)))))  
bias\_0 = tf.Variable(tf.random\_normal([num\_layers\_0]))weights\_1 = tf.Variable(tf.random\_normal([num\_layers\_0,num\_layers\_1], stddev=(1/tf.sqrt(float(num\_layers\_0)))))  
bias\_1 = tf.Variable(tf.random\_normal([num\_layers\_1]))weights\_2 = tf.Variable(tf.random\_normal([num\_layers\_1,num\_output], stddev=(1/tf.sqrt(float(num\_layers\_1)))))  
bias\_2 = tf.Variable(tf.random\_normal([num\_output]))

## Initializing weigths and biases  
hidden\_output\_0 = tf.nn.relu(tf.matmul(input\_X,weights\_0)+bias\_0)  
hidden\_output\_0\_0 = tf.nn.dropout(hidden\_output\_0, keep\_prob)hidden\_output\_1 = tf.nn.relu(tf.matmul(hidden\_output\_0\_0,weights\_1)+bias\_1)  
hidden\_output\_1\_1 = tf.nn.dropout(hidden\_output\_1, keep\_prob)predicted\_y = tf.sigmoid(tf.matmul(hidden\_output\_1\_1,weights\_2) + bias\_2)

## Defining the loss function  
loss = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits\_v2(logits=predicted\_y,labels=input\_y)) \  
 + regularizer\_rate\*(tf.reduce\_sum(tf.square(bias\_0)) + tf.reduce\_sum(tf.square(bias\_1)))

## Variable learning rate  
learning\_rate = tf.train.exponential\_decay(starter\_learning\_rate, 0, 5, 0.85, staircase=**True**)

## Adam optimzer for finding the right weight  
optimizer = tf.train.AdamOptimizer(learning\_rate).minimize(loss,var\_list=[weights\_0,weights\_1,weights\_2,  
 bias\_0,bias\_1,bias\_2])

## Metrics definition  
correct\_prediction = tf.equal(tf.argmax(y\_train,1), tf.argmax(predicted\_y,1))  
accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

*## Training parameters*  
batch\_size = 128  
epochs=14  
dropout\_prob = 0.6training\_accuracy = []  
training\_loss = []  
testing\_accuracy = []s.run(tf.global\_variables\_initializer())  
**for** epoch **in** range(epochs):   
 arr = np.arange(X\_train.shape[0])  
 np.random.shuffle(arr)  
 **for** index **in** range(0,X\_train.shape[0],batch\_size):  
 s.run(optimizer, {input\_X: X\_train[arr[index:index+batch\_size]],  
 input\_y: y\_train[arr[index:index+batch\_size]],  
 keep\_prob:dropout\_prob})  
 training\_accuracy.append(s.run(accuracy, feed\_dict= {input\_X:X\_train,   
 input\_y: y\_train,keep\_prob:1}))  
 training\_loss.append(s.run(loss, {input\_X: X\_train,   
 input\_y: y\_train,keep\_prob:1}))  
   
 *## Evaluation of model*  
 testing\_accuracy.append(accuracy\_score(y\_test.argmax(1),   
 s.run(predicted\_y, {input\_X: X\_test,keep\_prob:1}).argmax(1)))  
 print("Epoch:**{0}**, Train loss: **{1:.2f}** Train acc: **{2:.3f}**, Test acc:**{3:.3f}**".format(epoch,  
 training\_loss[epoch],  
 training\_accuracy[epoch],  
 testing\_accuracy[epoch]))

*## Plotting chart of training and testing accuracy as a function of iterations*  
iterations = list(range(epochs))  
plt.plot(iterations, training\_accuracy, label='Train')  
plt.plot(iterations, testing\_accuracy, label='Test')  
plt.ylabel('Accuracy')  
plt.xlabel('iterations')  
plt.show()  
print("Train Accuracy: **{0:.2f}**".format(training\_accuracy[-1]))  
print("Test Accuracy:**{0:.2f}**".format(testing\_accuracy[-1]))