**Machine Learning – HW 4**

Atieh Kashani

In this assignmentthe Forward Neural Network has been applied on the MINST dataset for hand written recognition. The database has a training set of 60000 digit images and a test of 10000 images.

**Question 1:** After loading the MINST dataset, transforming each image from 28 X 28 to a 784 pixel vector, and normalizing the inputs from gray scale of 0-255 to values between 0-1, the 60000 training images and labels has been splittedinto twosubsets, 50000 images for training the FNN model and 10000 images for evaluation. I have implemented the following code to split the training data set in to two subsets:

nTrain=50000  
indices = np.random.permutation(x\_train.shape[0])  
  
  
training\_idx, validation\_idx = indices[:nTrain], indices[nTrain:]  
  
trainX = x\_train[training\_idx] *# training samples*trainY = y\_train[training\_idx] *# labels of training samples*validationX = x\_train[validation\_idx] *# validation samples*validationY = y\_train[validation\_idx] *# labels of validation samples*

To implement the neural network algorithm I have used the provided Tensorflow implementation and file “main\_ha4.py “ contains my codes including the steps for setting the model hyper-parameters, preparing the Tensorflow graph, and Run the graph.

The total number of image pixels is 784 and the number of classes is 10 (one class per digit).

I have implemented the two hidden layer network in the “inference” method in “ func\_two\_layer\_ft.py” file :

*# Layer 1 - First hidden layer***with** tf.variable\_scope(**'Layer1'**):  
 *# Define the variables* weights = tf.get\_variable(  
 name=**'weights'**,  
 shape=[image\_pixels, hidden\_units\_first],  
 initializer=tf.truncated\_normal\_initializer(  
 stddev=1.0 / np.sqrt(float(image\_pixels))),  
 regularizer=tf.contrib.layers.l2\_regularizer(reg\_constant)  
 )  
  
 biases = tf.Variable(tf.zeros([hidden\_units\_first]), name=**'biases'**)  
  
 *# Define the layer's output* hidden1 = tf.nn.relu(tf.matmul(images, weights) + biases)  
  
*#Layer 2 - Second hidden layer***with** tf.variable\_scope(**'Layer2'**):  
 *# Define the variables* weights = tf.get\_variable(  
 name=**'weights'**,  
 shape=[hidden\_units\_first, hidden\_units\_second],  
 initializer=tf.truncated\_normal\_initializer(  
 stddev=1.0 / np.sqrt(float(hidden\_units\_first))),  
 regularizer=tf.contrib.layers.l2\_regularizer(reg\_constant)  
 )  
  
 biases = tf.Variable(tf.zeros([hidden\_units\_second]), name=**'biases'**)  
  
 *# Define the layer's output* hidden2 = tf.nn.relu(tf.matmul(hidden1, weights) + biases)  
  
  
*# Layer 3***with** tf.variable\_scope(**'Layer3'**):  
 *# Define variables* weights = tf.get\_variable(**'weights'**, [hidden\_units\_second, classes],  
 initializer=tf.truncated\_normal\_initializer(  
 stddev=1.0 / np.sqrt(float(hidden\_units\_second))),  
 regularizer=tf.contrib.layers.l2\_regularizer(reg\_constant))  
  
 biases = tf.Variable(tf.zeros([classes]), name=**'biases'**)  
  
 *# Define the layer's output* logits = tf.matmul(hidden2, weights) + biases  
  
 *# Define summery-operation for 'logits'-variable* tf.summary.histogram(**'logits'**, logits)

I have tried different sets of hyper-parameters ( number of hidden layers 1 and 2 with different number of neural units for each layer ( 200, 400, 500 ) ( 50, 100 , 300) , different learning rate ( 0.01, 0.001, 0.05 ) and different activation and output functions, and train the FNN model on the training samples ( 50000 images ) and apply the model over the validation set ( 10000 images). Following are the results of the models that achieve the top accuracy (The first model has the highest accuracy):

Also, confusion matrix, average accuracy, per-class Precision and Recall have be calculated:

* Number of hidden layers: 1, Number of neural units for each hidden layer: 400, Learning rate:0.05 , Activation function: tanh\_opt

Step 0, training accuracy 0.1475

Step 100, training accuracy 0.7925

Step 200, training accuracy 0.835

Step 300, training accuracy 0.825

Step 400, training accuracy 0.8725

Step 500, training accuracy 0.8425

Step 600, training accuracy 0.885

Step 700, training accuracy 0.895

Step 800, training accuracy 0.805

Step 900, training accuracy 0.85

Saved checkpoint

Step 1000, training accuracy 0.83

Step 1100, training accuracy 0.8475

Step 1200, training accuracy 0.8375

Step 1300, training accuracy 0.89

Step 1400, training accuracy 0.8225

Step 1500, training accuracy 0.805

Step 1600, training accuracy 0.865

Step 1700, training accuracy 0.865

Step 1800, training accuracy 0.865

Step 1900, training accuracy 0.8425

Saved checkpoint

Validation accuracy 0.8538

Total time: 43.81s

confusion\_matrix

[[ 925 0 4 3 1 16 16 3 9 1]

[ 0 1103 1 3 2 3 1 2 9 0]

[ 16 42 801 22 24 1 62 16 21 1]

[ 5 22 23 861 5 45 7 20 22 9]

[ 1 18 4 0 896 2 25 2 5 55]

[ 23 56 10 55 30 643 35 11 19 9]

[ 13 21 31 0 6 11 886 1 3 0]

[ 7 40 13 0 13 6 0 936 1 32]

[ 3 66 17 47 7 28 9 3 763 33]

[ 7 24 11 19 83 5 2 90 13 724]]

accuracy

0.8538

recall\_array

[0.94580777 0.98131673 0.79622266 0.84494603 0.88888889 0.72166105

0.91152263 0.89312977 0.7817623 0.7402863 ]

precision\_array

[0.925 0.79238506 0.87540984 0.85247525 0.83973758 0.84605263

0.84947267 0.86346863 0.88208092 0.83796296]

* Number of hidden layers: 2 , Number of neural units for each hidden layer: 300, 100 , Learning rate:0.05 , Activation function: tanh\_opt

Step 0, training accuracy 0.0525

Step 100, training accuracy 0.77

Step 200, training accuracy 0.79

Step 300, training accuracy 0.7775

Step 400, training accuracy 0.7875

Step 500, training accuracy 0.7775

Step 600, training accuracy 0.775

Step 700, training accuracy 0.7825

Step 800, training accuracy 0.815

Step 900, training accuracy 0.76

Saved checkpoint

Step 1000, training accuracy 0.7925

Step 1100, training accuracy 0.7925

Step 1200, training accuracy 0.79

Step 1300, training accuracy 0.79

Step 1400, training accuracy 0.8025

Step 1500, training accuracy 0.795

Step 1600, training accuracy 0.765

Step 1700, training accuracy 0.7875

Step 1800, training accuracy 0.77

Step 1900, training accuracy 0.755

Saved checkpoint

Validation accuracy 0.7869

Total time: 51.58s

confusion\_matrix

[[ 941 1 8 13 2 7 9 2 1 0]

[ 0 1076 9 6 0 0 0 2 9 3]

[ 30 51 795 57 30 0 60 22 1 1]

[ 6 29 35 810 3 34 2 27 29 8]

[ 6 14 6 0 761 0 29 4 3 135]

[ 52 19 37 113 21 440 40 10 109 41]

[ 21 16 26 0 12 2 868 0 23 0]

[ 6 54 3 3 7 0 0 1005 0 48]

[ 19 86 11 69 24 78 36 3 613 25]

[ 11 22 6 9 228 6 2 133 6 560]]

accuracy

0.7869

recall\_array

[0.95630081 0.97375566 0.75931232 0.82400814 0.79436326 0.49886621

0.89669421 0.89253996 0.63589212 0.56968464]

precision\_array

[0.86172161 0.78654971 0.84935897 0.75 0.69944853 0.77601411

0.82982792 0.83195364 0.7720403 0.68209501]

* Number of hidden layers: 2, Number of neural units for each hidden layer: 400, 100, Learning rate:0.01 , Activation function: relu

Step 0, training accuracy 0.055

Step 100, training accuracy 0.6075

Step 200, training accuracy 0.705

Step 300, training accuracy 0.6975

Step 400, training accuracy 0.725

Step 500, training accuracy 0.72

Step 600, training accuracy 0.73

Step 700, training accuracy 0.7525

Step 800, training accuracy 0.7575

Step 900, training accuracy 0.7525

Saved checkpoint

Step 1000, training accuracy 0.7725

Step 1100, training accuracy 0.8

Step 1200, training accuracy 0.7425

Step 1300, training accuracy 0.77

Step 1400, training accuracy 0.7775

Step 1500, training accuracy 0.775

Step 1600, training accuracy 0.7825

Step 1700, training accuracy 0.76

Step 1800, training accuracy 0.775

Step 1900, training accuracy 0.815

Saved checkpoint

Validation accuracy 0.7922

Total time: 51.08s

confusion\_matrix

[[ 952 0 2 5 3 6 10 2 7 1]

[ 0 1121 8 8 0 0 1 1 9 1]

[ 20 45 751 45 24 1 61 12 16 2]

[ 13 33 44 812 1 30 8 26 55 5]

[ 3 9 2 0 775 0 29 2 12 140]

[ 52 18 23 184 24 418 47 17 75 30]

[ 16 8 39 2 16 4 915 0 5 0]

[ 10 49 7 0 4 0 0 925 2 34]

[ 15 85 36 108 21 19 20 8 632 35]

[ 11 17 5 15 185 6 1 114 9 621]]

accuracy

0.7922

recall\_array

[0.96356275 0.97563098 0.76867963 0.79065239 0.7973251 0.47072072

0.91044776 0.8971872 0.64555669 0.63109756]

precision\_array

[0.87179487 0.80938628 0.81897492 0.68871925 0.7359924 0.86363636

0.83791209 0.83559169 0.76885645 0.7146145 ]

**Question 2**: After selecting the hyper-parameters based on the results of applying the trained FNN model over on the validation set, I have applied the model that achieved the top accuracy using those hyper-parameters on the testing samples (10000 images) and followings are the results:

Test accuracy 0.8594

Total time: 41.87s

confusion\_matrix

[[ 925 0 8 4 1 6 8 2 7 0]

[ 0 1116 2 4 2 5 4 3 12 1]

[ 16 51 813 19 19 2 46 24 16 4]

[ 9 24 38 822 5 27 6 22 27 9]

[ 2 20 5 0 811 1 16 3 1 66]

[ 39 45 8 76 24 641 35 18 23 14]

[ 14 21 25 0 6 12 909 1 3 0]

[ 2 37 11 1 13 6 0 959 1 31]

[ 10 66 14 60 13 21 15 1 740 32]

[ 12 14 12 24 87 4 3 79 7 777]]

accuracy

0.8513

recall\_array

[0.96253902 0.97127937 0.8049505 0.83114257 0.87675676 0.69447454

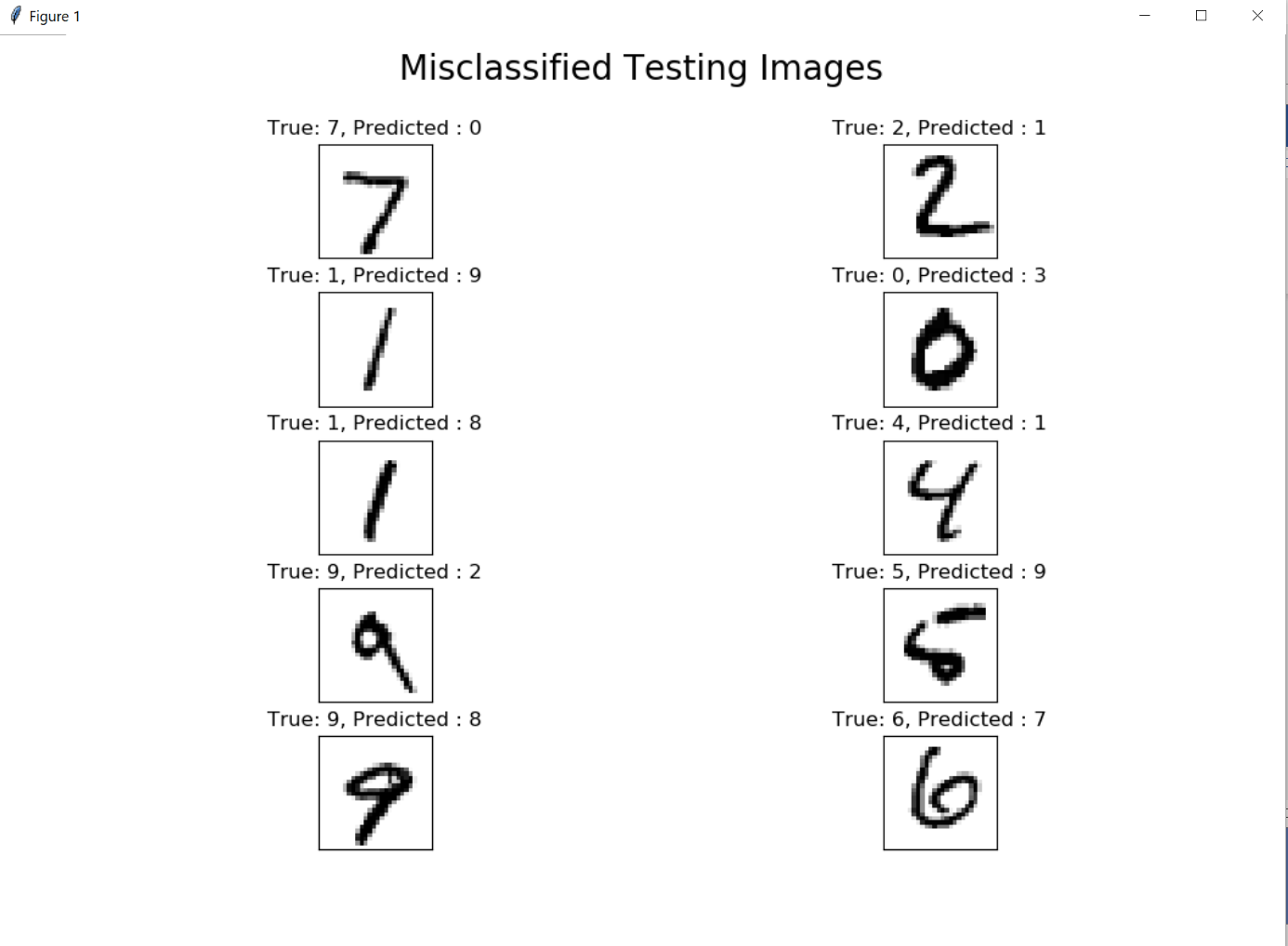
0.9172553 0.90386428 0.76131687 0.76251227]

precision\_array

[0.898931 0.80057389 0.86858974 0.81386139 0.82670744 0.88413793

0.87236084 0.86241007 0.88410992 0.83190578]

* To visualize and compare the true label and the model results, I have implemented the related code in the “main\_ha4.py” file that identifies the misclassified image by comparing the true and predicted labels and plot ten misclassified testing images**.** Following plot displays ten testing images for which this model made error and failed to classify them correctly:



Some of the reasons of these failure cases are: because of the noise in the samples the model cannot learn the labels and predict them accurately. Another factor that can affect the model is the batch size. Large batch size can reduce the accuracy of the model.