CS 596 Machine Learning

The goal of this homework assignment is to apply the feedforward Neural Network (FNN) to solve hand written digit recognition utilizing the MNIST dataset.

The MNIST dataset is divided as 60,000 entries of x\_train, y\_train for training dataset, and 10,000 entries of x\_test, y\_test for the testing dataset. In particular, the training dataset is divided further to use 50,000 images and labels for the trained model, and the other 10,000 images for evaluation.

Method:

1. using number of hidden layers.

train\_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross\_entropy)

for \_ in range(1000):

  batch = mnist.train.next\_batch(100)

  train\_step.run(feed\_dict={x: batch[0], y\_: batch[1]})

a) Using gradient Descent Optimizer,

b) Utilizing a 3 layer convoluted neural network, I got a much better accuracy rate of 0.9784.

ALPHA rate = 0.001

#First Convolutional Layer: will have 32 features for each 5x5 patch. (-,-,# of input channel, # output channel)

W\_conv1 = weight\_variable([5, 5, 1, 32])

b\_conv1 = bias\_variable([32])

//2nd and 3rd dimension corresponding to image width and height, and 4th dimension corresponding to the # of color channel.

x\_image = tf.reshape(x, [-1, 28, 28, 1])

h\_conv1 = tf.nn.relu(conv2d(x\_image, W\_conv1) + b\_conv1)

h\_pool1 = max\_pool\_2x2(h\_conv1)

#Second Convolutional layer: will have 64 features for each 5x5 patch.

W\_conv2 = weight\_variable([5, 5, 32, 64])

b\_conv2 = bias\_variable([64])

h\_conv2 = tf.nn.relu(conv2d(h\_pool1, W\_conv2) + b\_conv2)

h\_pool2 = max\_pool\_2x2(h\_conv2)

#Densely Connected Layer: the image size has been reduced to 7x7, the number of neurons are set o 1024 to process the entire image. The tensor from the pooling layer has been reshaped into vectors, multiply by a weight matrix, add a bias, and apply a ReLU.

W\_fc1 = weight\_variable([7 \* 7 \* 64, 1024])

b\_fc1 = bias\_variable([1024])

h\_pool2\_flat = tf.reshape(h\_pool2, [-1, 7\*7\*64])

h\_fc1 = tf.nn.relu(tf.matmul(h\_pool2\_flat, W\_fc1) + b\_fc1)

Dropout

keep\_prob = tf.placeholder(tf.float32)

h\_fc1\_drop = tf.nn.dropout(h\_fc1, keep\_prob)

// to reduce overfitting, we will apply the dropout before the readout layer

Readout Layer

W\_fc2 = weight\_variable([1024, 10])

b\_fc2 = bias\_variable([10])

As a result we get a output of “y\_conv” which is:

y\_conv = tf.matmul(h\_fc1\_drop, W\_fc2) + b\_fc2

Figure 1. Example of the output when ALPHA = 0.001, Activation Function = tf.nn.relu

======================================================================

step 0, training accuracy 0.11306

step 100, training accuracy 0.95316

step 200, training accuracy 0.96986

step 300, training accuracy 0.9724

step 400, training accuracy 0.9798

======================================================================

test accuracy 0.9784

Confusion matrix size = 10

[[9.680e+02 1.000e+00 1.000e+00 0.000e+00 0.000e+00 3.000e+00 2.000e+00 1.000e+00 4.000e+00 0.000e+00]

[0.000e+00 1.131e+03 1.000e+00 2.000e+00 0.000e+00 0.000e+00 0.000e+00 1.000e+00 0.000e+00 0.000e+00]

[1.000e+00 5.000e+00 1.014e+03 3.000e+00 0.000e+00 0.000e+00 0.000e+00 6.000e+00 3.000e+00 0.000e+00]

[0.000e+00 0.000e+00 2.000e+00 9.880e+02 0.000e+00 1.200e+01 0.000e+00 6.000e+00 2.000e+00 0.000e+00]

[0.000e+00 0.000e+00 1.000e+00 0.000e+00 9.720e+02 0.000e+00 5.000e+00 0.000e+00 1.000e+00 3.000e+00]

[0.000e+00 0.000e+00 0.000e+00 3.000e+00 0.000e+00 8.880e+02 0.000e+00 1.000e+00 0.000e+00 0.000e+00]

[3.000e+00 3.000e+00 0.000e+00 0.000e+00 2.000e+00 3.700e+01 9.110e+02 0.000e+00 2.000e+00 0.000e+00]

[0.000e+00 5.000e+00 8.000e+00 0.000e+00 1.000e+00 0.000e+00 0.000e+00 1.007e+03 2.000e+00 5.000e+00]

[1.000e+00 1.000e+00 6.000e+00 4.000e+00 4.000e+00 1.900e+01 1.000e+00 3.000e+00 9.320e+02 3.000e+00]

[3.000e+00 6.000e+00 1.000e+00 4.000e+00 6.000e+00 8.000e+00 0.000e+00 7.000e+00 1.000e+00 9.730e+02]]

accuracy\_rate 0.9784

recall\_array [0.9877551 0.99647577 0.98255814 0.97821782 0.9898167 0.9955157 0.95093946 0.97957198 0.95687885 0.96432111]

precision\_list [0.99180328 0.98177083 0.98065764 0.98406375 0.98680203 0.91830403

0.99129489 0.97577519 0.98416051 0.98882114]

2) Learning Rate

ALPHA = 0.1 is fast, the model failed to converge.

ALPHA = 0.001 is the rate I decide for the final model

ALPHA = 0.01, accuracy rate is 0.9801

Figure 2. Example of the output when ALPHA = 0.01, Activation Function = tf.nn.relu

==============================================================================

step 0, training accuracy 0.10828

step 100, training accuracy 0.94688

step 200, training accuracy 0.97012

step 300, training accuracy 0.97322

step 400, training accuracy 0.97926

test accuracy 0.9801

==============================================================================

Confusion matrix size =10

[[9.730e+02 0.000e+00 0.000e+00 0.000e+00 0.000e+00 3.000e+00 1.000e+00 1.000e+00 2.000e+00 0.000e+00]

[0.000e+00 1.129e+03 1.000e+00 2.000e+00 0.000e+00 2.000e+00 0.000e+00 1.000e+00 0.000e+00 0.000e+00]

[2.000e+00 5.000e+00 1.010e+03 3.000e+00 1.000e+00 0.000e+00 2.000e+00 7.000e+00 2.000e+00 0.000e+00]

[1.000e+00 0.000e+00 1.000e+00 9.760e+02 0.000e+00 2.200e+01 0.000e+00 8.000e+00 1.000e+00 1.000e+00]

[0.000e+00 0.000e+00 1.000e+00 0.000e+00 9.650e+02 0.000e+00 4.000e+00 1.000e+00 3.000e+00 8.000e+00]

[1.000e+00 0.000e+00 0.000e+00 1.000e+00 0.000e+00 8.870e+02 1.000e+001.000e+00 1.000e+00 0.000e+00]

[5.000e+00 3.000e+00 0.000e+00 0.000e+00 1.000e+00 1.400e+01 9.320e+02 0.000e+00 3.000e+00 0.000e+00]

[0.000e+00 3.000e+00 1.100e+01 0.000e+00 0.000e+00 1.000e+00 0.000e+00 1.003e+03 2.000e+00 8.000e+00]

[6.000e+00 0.000e+00 1.000e+00 4.000e+00 3.000e+00 6.000e+00 0.000e+00 1.000e+00 9.490e+02 4.000e+00]

[4.000e+00 7.000e+00 2.000e+00 0.000e+00 5.000e+00 5.000e+00 0.000e+00 7.000e+00 2.000e+00 9.770e+02]]

accuracy\_rate 0.9801

recall\_array: [0.99285714 0.99471366 0.97868217 0.96633663 0.98268839 0.9943946 0.97286013 0.97568093 0.97433265 0.96828543]

precision\_list: [0.98084677 0.98430689 0.98344693 0.98985801 0.98974359 0.94361702 0.99148936 0.97378641 0.98341969 0.97895792]

3) Activation Function

h\_fc1 = tf.nn.relu(tf.matmul(h\_pool2\_flat, W\_fc1) + b\_fc1). (See Figure 1 & 2)

I also tried

tf.nn.tanh, and result was not good, so the only activation function gave the best result is the tf.nn.relu activation function.

Question 1. During model fitting for each set, we train a FNN model on the training samples (50,000) images, and apply the learn model over the validation set (10,000 images).

train\_accuracy = accuracy.eval(feed\_dict={x: x\_validate, y\_: vBatch\_y, keep\_prob: 1.0})

Where vBatch\_y = returnLabel(y\_validate), x\_validate and y\_validate is the 10,000 subset of the training dataset.

Question 2. Apply the model over the test samples (10,000 images ). For the 3 layer convoluted neural network, we fit the model with predicted values for ALPHA = 0.001 and ALPHA = 0.01. The confusion matrix, accuracy, per-class precision, and per-class recall are being calculated inside the main\_ha5.py.