PROJECT 2 - REPORT

NAME: DEEPAK GALA

AGGIE ID: 800584049

COURSE: DEEP LEARNING (EE 590 SELECTED TOPICS)

Note: The results presented in the report are using smaller epoch value. Although much higher value of epoch is preferred for better accuracy, a smaller epoch was used for demonstration purpose and faster processing.

PART A – NETWORK A

Epoch = 50

Accuracy = 71.97 %

Task 1: Displaying digits

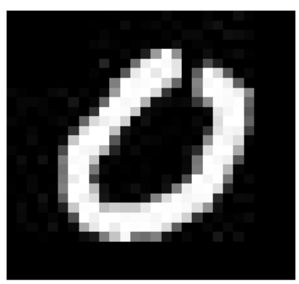


Image of Digit 0

To display one image of each digit from 0-9, I started creating a minibatch and searched for each digit in it. I had to change the minibatch size as the digits were repeated. I realized that in order to get all digits I need to access a minibatch of first 30 images.

mini_x, mini_y = mnist.train.next_batch(30)

I then recorded their positions in this minibatch and reshaped them.

Eg.

num=mini_x[7]

num0=num.reshape(28,28)

Each image was then saved using

scipy.misc.imsave('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A1 Output/0.jpg', num0)

Task 2: Visualizing weights

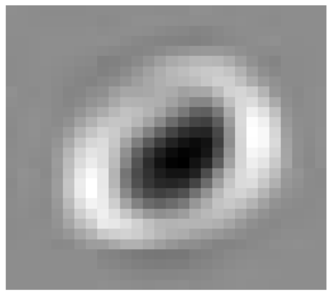


Image of weight W_0

The following part of code plots and saves the images of the first 10 weights.

```
for wi in range (9):
    weights=sess.run(W)
    im=weights[:,wi]
    wim=im.reshape(28,28)
    filename = "C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A1 Output/W_%d.jpg"%wi
    scipy.misc.imsave(filename, wim)
```

Accuracy was then calculated and recorded in a txt file using the following part of code

```
with open('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A1 Output/Accuracy.txt', 'w') as f:
    print('Accuracy = ',accuracy, file=f)

f.close()
```

The weights images were very close to the actual digits.

PART A – NETWORK B

Epoch = 50

Accuracy = 87.84 %

Same procedure as PART A – NETWORK A was followed to display images and visualize weights. The only difference was that this time there was 2 hidden layers and so the first 10 images of the weights did not make sense what they mean as they were nowhere close to the actual digits.

Here is the image of weight W0 for the first hidden layer

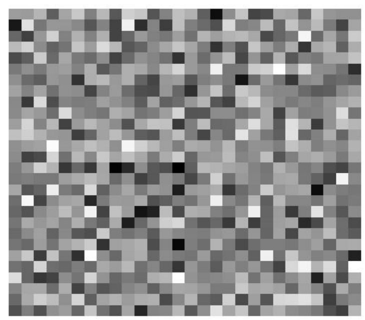
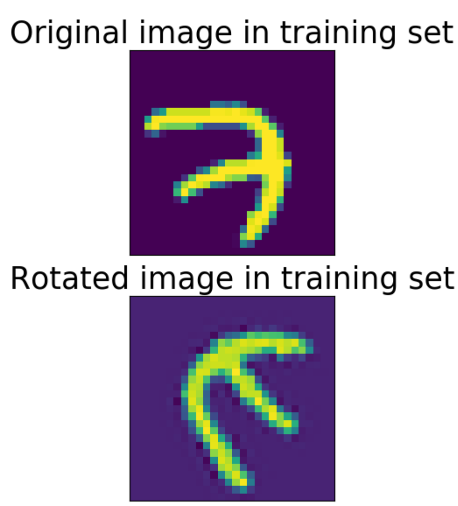


Image of weight W_0 of first hidden layer

PART B – NETWORK A

Epoch = 10

Accuracy = 54.03 %



In this part, the images were rotated before they were used to train, validate and test the neural network. This was achieved by implementing a function *rotateImage( ) as follows.*

```
def rotateImage(im):
#    print(im.shape)
    im_size=round(np.size(im)/784)
    for j in range(im_size):
        img=im[j]
        img=img.reshape(28,28)
        rot_image=rotate(img, random.randint(0,360),reshape=False)
#        print(rot_image.shape)
        rot_image=rot_image.reshape(784)
        im[j]=rot_image
#    print(im.shape)
    return im
```

Confusion Matrix

True Values

	0	1	2	3	4	5	6	7	8	9
0	802	0	72	4	12	1	26	61	2	0
1	0	1089	3	3	2	1	1	0	35	1
2	140	9	304	255	37	33	72	60	102	20
3	23	63	114	549	8	29	34	2	173	15
4	21	65	28	11	381	8	95	161	37	175
5	21	45	143	218	61	120	34	55	150	45
6	71	29	73	19	60	2	399	110	60	135
7	40	14	17	5	109	7	100	667	21	48
8	0	69	57	128	40	24	58	12	576	10
9	15	46	18	0	173	2	104	118	17	516

Estimated Values

Also, a confusion matrix was created for the test data using the following code:

```
y_test = sess.run(argMax_y, feed_dict={x: mnist.test.images, y: mnist.test.labels})
```

```
output_test = sess.run(argMax_output, feed_dict={x: mnist.test.images, y: mnist.test.labels})
C_matrix = np.zeros((10,10))
count=len(y_test)
for i in range(0,count):
    true_value = y_test[i]
    est_value = output_test[i]
    C_matrix[true_value,est_value] += 1



plt.figure(figsize = (15,15))
rlabels = ['  0  ','  1  ','  2  ','  3  ','  4  ','  5  ','  6  ','  7  ','  8  ','  9  ']
clabels = ['0','1','2','3','4','5','6','7','8','9']
ytable = plt.table(cellText=np.int_(C_matrix), loc='center', rowLabels=rlabels,colLabels=clabels)
ytable.set_fontsize(14)

table_props = ytable.properties()
table_cells = table_props['child_artists']
for cell in table_cells:
    cell.set_height(0.09)
    cell.set_width(0.09)

#   plt.rcParams['axes.labelweight'] = 'bold'
plt.xticks([], [])
plt.yticks([], [])
plt.ylabel('True Values', fontsize=25)
plt.xlabel('Estimated Values', fontsize=25)
plt.title('Confusion Matrix', fontsize=45)

#   ax = plt.axes()
```

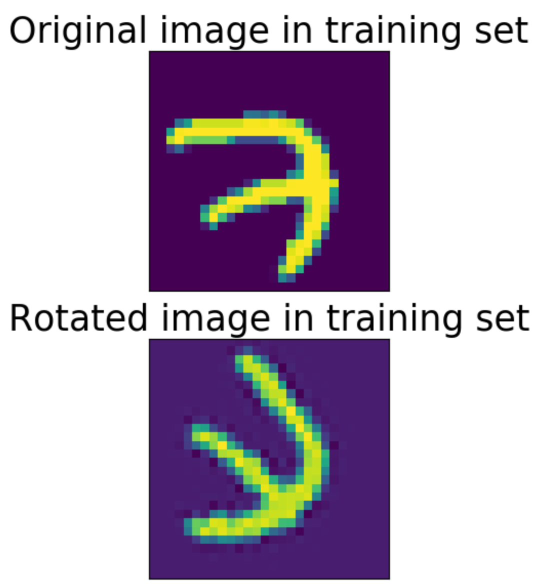
```
#       ax.xaxis.set_ticks_position('none')
#         tb = plt.gca()
#         tb.set_xticks([])
#         tb.set_yticks([])

    plt.show()
    plt.savefig('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part B1 Output/Confusion_Matrix.jpg')
```

PART B – NETWORK B

Epoch = 10

Accuracy = 77.40 %



Same procedure as PART B – NETWORK A was followed to rotate the image before training, validation and testing of the neural network B. Confusion matrix was also created for the neural network B with two hidden layers.

Confusion Matrix

True Values \ Estimated Values	0	1	2	3	4	5	6	7	8	9
0	505	0	152	16	47	63	109	13	22	53
1	0	1110	5	1	3	0	1	0	15	0
2	3	1	810	31	75	14	11	49	34	4
3	2	8	38	793	2	38	22	12	89	6
4	0	4	53	1	763	23	13	55	18	52
5	1	4	11	45	16	666	47	29	33	40
6	6	8	25	19	17	21	723	53	25	61
7	2	16	56	7	57	1	13	844	3	29
8	0	4	24	32	33	27	16	1	826	11
9	2	9	9	7	113	28	51	17	10	763

Estimated Values

PART C – NETWORK A

Epoch = 10

Accuracy = 79.97 %

Original image in training set

Scaled image in training set

In this part, the images were scaled before they were used to train, validate and test the neural network. This was achieved by implementing a function ScaleImage ( ) as follows

```
def ScaleImage(im):
#    print(im.shape)
#    all_scaled_image=[]
    im_size=round(np.size(im)/784)
    for j in range(im_size):
        img=im[j]
        img=img.reshape(28,28)
        scale_amount=np.random.uniform(low=0.5, high=1.0)
#        print(scale_amount)
        scaled_image=zoom(img,scale_amount)
        scaled_image_size=[round(math.sqrt(scaled_image.size)),round(math.sqrt(scaled_image.size))]
        size = (28,28)
        background =np.zeros(size)
        offset = [round((size[0] - scaled_image_size[0]) / 2), round((size[1] - scaled_image_size[1]) / 2)]
#            offset_y = (size[1] - scaled_image_size[1]) / 2
        background[offset[0]:offset[0] + scaled_image_size[0], offset[1]:offset[1] +
scaled_image_size[1]]=scaled_image
        scaled_padded_image =background
        scaled_padded_image=scaled_padded_image.reshape(784)
        im[j]=scaled_padded_image
#    print(im.shape)
    return im
```

Confusion Matrix

True Values \ Estimated Values	0	1	2	3	4	5	6	7	8	9
0	247	152	9	23	237	57	123	22	13	97
1	3	966	2	8	5	99	4	18	30	0
2	3	427	274	8	85	68	23	53	28	63
3	0	497	18	255	18	116	17	49	26	14
4	0	265	1	5	533	107	11	0	26	34
5	7	356	2	60	42	294	33	7	76	15
6	4	285	4	0	237	100	263	0	42	23
7	1	487	4	20	142	50	0	294	18	12
8	5	481	0	20	52	110	21	17	240	28
9	2	405	3	24	233	124	2	5	28	183

Also, a confusion matrix was created as mentioned in PART B – NETWORK A for the testing data.

PART C – NETWORK B

Epoch = 50

Accuracy = 86.32 %

Original image in training set

Scaled image in training set

Same procedure as PART B – NETWORK A was followed to scale the image before training, validation and testing of the neural network B.

Confusion matrix was also created for the neural network B with two hidden layers.

Confusion Matrix

True Values / Estimated Values

	0	1	2	3	4	5	6	7	8	9
0	304	144	4	5	179	39	139	35	8	123
1	7	972	1	3	2	106	4	28	12	0
2	7	382	330	11	59	48	30	80	17	68
3	1	497	14	266	14	115	12	43	24	24
4	1	268	2	2	595	54	22	1	6	31
5	4	325	1	37	56	390	33	6	25	15
6	2	279	7	2	235	86	322	1	4	20
7	0	482	9	14	146	46	6	292	10	23
8	0	446	2	12	55	121	39	9	247	43
9	5	415	0	6	206	70	5	6	15	281

Estimated Values

Conclusion:

The weights were visualized for a network with no hidden layer (network A) and two hidden layers (network B). It was clear that the weights in the network A was looking like the corresponding digits whereas in case of network B, the weights did not make any sense.

In part B, both neural network A and network B were tested for the rotated images. Where as in part C they were tested for scaled images.

Confusion matrix was calculated using the test data images.

All the images including the confusion matrix image were saved and the accuracy was recorded in a .txt file. The accuracy for each case is tabulated as follows.

	Network A	Network B
Original images (epoch = 50)	71.97 %	87.84 %
Rotated images (epoch = 10)	54.03 %	77.40 %
Scaled images (epoch = 10)	79.97 %	86.32 %

From the above results, it is clear that network B performs better than network A due to the presence of hidden layers.

CODES

PART A – NETWORK A

```
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 29 11:30:46 2017

@author: Deepak

Part A - Network A
"""

import sys
sys.path.append('../../')
sys.path.append('../')
import numpy as np
import tensorflow as tf
import time, shutil, os
from fdl_examples.datatools import input_data
import matplotlib.pyplot as plt
import scipy
from scipy import misc

# read in MNIST data --------------------------------------------------
mnist = input_data.read_data_sets("../../data/", one_hot=True)

# run network ----------------------------------------------------------

# Parameters
learning_rate = 0.01
training_epochs = 50 # NOTE: you'll want to eventually change this
batch_size = 100
display_step = 1


def inference(x,W,b):
    output = tf.nn.softmax(tf.matmul(x, W) + b)

    w_hist = tf.summary.histogram("weights", W)
    b_hist = tf.summary.histogram("biases", b)
    y_hist = tf.summary.histogram("output", output)

    return output

def loss(output, y):
    dot_product = y * tf.log(output)
```

```
    # Reduction along axis 0 collapses each column into a single
    # value, whereas reduction along axis 1 collapses each row
    # into a single value. In general, reduction along axis i
    # collapses the ith dimension of a tensor to size 1.
    xentropy = -tf.reduce_sum(dot_product, axis=1)

    loss = tf.reduce_mean(xentropy)

    return loss

def training(cost, global_step):

    tf.summary.scalar("cost", cost)
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    train_op = optimizer.minimize(cost, global_step=global_step)

    return train_op


def evaluate(output, y):
    correct_prediction = tf.equal(tf.argmax(output, 1), tf.argmax(y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

    tf.summary.scalar("validation error", (1.0 - accuracy))

    return accuracy

if __name__ == '__main__':
#    if os.path.exists("logistic_logs/"):
#        shutil.rmtree("logistic_logs/")

    with tf.Graph().as_default():

        x = tf.placeholder("float", [None, 784]) # mnist data image of shape 28*28=784
        y = tf.placeholder("float", [None, 10]) # 0-9 digits recognition => 10 classes

        init = tf.constant_initializer(value=0)
        W = tf.get_variable("W", [784, 10],
                     initializer=init)
        b = tf.get_variable("b", [10],
                     initializer=init)

        output = inference(x,W,b)

        cost = loss(output, y)

        global_step = tf.Variable(0, name='global_step', trainable=False)
```

```
    train_op = training(cost, global_step)

    eval_op = evaluate(output, y)

    summary_op = tf.summary.merge_all()

    saver = tf.train.Saver()

    sess = tf.Session()

#     summary_writer = tf.summary.FileWriter("logistic_logs/",
#                                             graph_def=sess.graph_def)


    init_op = tf.global_variables_initializer()

    sess.run(init_op)


    # PLOTTING EACH DIGIT
    #x[7]=0, x[6]=1, x[13]=2, x[1]=3, x[2]=4, x[27]=5, x[26]=6, x[25]=7, x[9]=8, x[8]=9
    mini_x, mini_y = mnist.train.next_batch(30)

    num=mini_x[7]
    num0=num.reshape(28,28)

    num=mini_x[6]
    num1=num.reshape(28,28)

    num=mini_x[13]
    num2=num.reshape(28,28)

    num=mini_x[1]
    num3=num.reshape(28,28)

    num=mini_x[2]
    num4=num.reshape(28,28)

    num=mini_x[27]
    num5=num.reshape(28,28)

    num=mini_x[26]
    num6=num.reshape(28,28)

    num=mini_x[25]
    num7=num.reshape(28,28)
```

```
        num=mini_x[9]
        num8=num.reshape(28,28)

        num=mini_x[8]
        num9=num.reshape(28,28)

#       plt.imshow(num1)
#
        scipy.misc.imsave('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A1 Output/0.jpg',
num0)
        scipy.misc.imsave('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A1 Output/1.jpg',
num1)
        scipy.misc.imsave('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A1 Output/2.jpg',
num2)
        scipy.misc.imsave('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A1 Output/3.jpg',
num3)
        scipy.misc.imsave('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A1 Output/4.jpg',
num4)
        scipy.misc.imsave('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A1 Output/5.jpg',
num5)
        scipy.misc.imsave('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A1 Output/6.jpg',
num6)
        scipy.misc.imsave('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A1 Output/7.jpg',
num7)
        scipy.misc.imsave('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A1 Output/8.jpg',
num8)
        scipy.misc.imsave('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A1 Output/9.jpg',
num9)

#        scipy.misc.imsave('outfile.jpg', image_array)

        # Training cycle
        for epoch in range(training_epochs):

            avg_cost = 0.
            total_batch = int(mnist.train.num_examples/batch_size)
            # Loop over all batches
            for i in range(batch_size):
                minibatch_x, minibatch_y = mnist.train.next_batch(batch_size)

            # Fit training using batch data
            sess.run(train_op, feed_dict={x: minibatch_x, y: minibatch_y})
            # Compute average loss
            avg_cost += sess.run(cost, feed_dict={x: minibatch_x, y: minibatch_y})/total_batch
        # Display logs per epoch step
            if epoch % display_step == 0:
                print("Epoch:", '%04d' % (epoch+1), "cost =", "{:.9f}".format(avg_cost))
```

```
            accuracy = sess.run(eval_op, feed_dict={x: mnist.validation.images, y: mnist.validation.labels})

            print("Validation Error:", (1 - accuracy))

            summary_str = sess.run(summary_op, feed_dict={x: minibatch_x, y: minibatch_y})
            #summary_writer.add_summary(summary_str, sess.run(global_step))

            #saver.save(sess, "logistic_logs/model-checkpoint", global_step=global_step)


        print("Optimization Finished!")

        accuracy = sess.run(eval_op, feed_dict={x: mnist.test.images, y: mnist.test.labels})

        #PLOTTING FIRST 10 WEIGHTS
        for wi in range (9):
          weights=sess.run(W)
          im=weights[:,wi]
          wim=im.reshape(28,28)
          filename = "C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A1 Output/W_%d.jpg"%wi
          scipy.misc.imsave(filename, wim)


        print("Test Accuracy:", accuracy)

        with open('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A1 Output/Accuracy.txt', 'w')
as f:
          print('Accuracy = ',accuracy, file=f)

        f.close()
```

PART A – NETWORK B

```
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 29 11:30:46 2017

@author: Deepak

Part A - Network B
"""


import sys
sys.path.append('../../')
sys.path.append('../')
from fdl_examples.datatools import input_data
mnist = input_data.read_data_sets("../../data/", one_hot=True)
import tensorflow as tf
import time, shutil, os
import scipy

# Architecture
n_hidden_1 = 256
n_hidden_2 = 256

# Parameters
learning_rate = 0.01
training_epochs = 50 # NOTE: you'll want to eventually change this
batch_size = 100
display_step = 1

#def layer(input, weight_shape, bias_shape):
def layer(input, W, b):
  return tf.nn.relu(tf.matmul(input, W) + b)

#def inference(x,W1,b1,W2,b2):
def inference(x):
  with tf.variable_scope("hidden_1"):
    hidden_1 = layer(x, W1,b1)

  with tf.variable_scope("hidden_2"):
    hidden_2 = layer(hidden_1,W2,b2)

  with tf.variable_scope("output"):
    #output = layer(hidden_2, [n_hidden_2, 10], [10])
    output = layer(hidden_2, W3,b3)
```

```
    return output

def loss(output, y):
    xentropy = tf.nn.softmax_cross_entropy_with_logits(logits=output, labels=y)
    loss = tf.reduce_mean(xentropy)
    return loss

def training(cost, global_step):
    tf.summary.scalar("cost", cost)
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    train_op = optimizer.minimize(cost, global_step=global_step)
    return train_op


def evaluate(output, y):
    correct_prediction = tf.equal(tf.argmax(output, 1), tf.argmax(y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    tf.summary.scalar("validation", accuracy)
    return accuracy

if __name__ == '__main__':

    #if os.path.exists("mlp_logs/"):
    #    shutil.rmtree("mlp_logs/")

    with tf.Graph().as_default():

        with tf.variable_scope("mlp_model"):

            x = tf.placeholder("float", [None, 784]) # mnist data image of shape 28*28=784
            y = tf.placeholder("float", [None, 10]) # 0-9 digits recognition => 10 classes

            weight_init1 = tf.random_normal_initializer(stddev=(2.0/784)**0.5)
            weight_init2 = tf.random_normal_initializer(stddev=(2.0/n_hidden_1)**0.5)
            weight_init3 = tf.random_normal_initializer(stddev=(2.0/n_hidden_2)**0.5)
            bias_init = tf.constant_initializer(value=0.)

            W1 = tf.get_variable("W1", [784, n_hidden_1],
                        initializer=weight_init1)
            b1 = tf.get_variable("b1", [n_hidden_1],
                        initializer=bias_init)
            W2 = tf.get_variable("W2", [n_hidden_1, n_hidden_2],
                        initializer=weight_init2)
            b2 = tf.get_variable("b2", [n_hidden_2],
                        initializer=bias_init)
            W3 = tf.get_variable("W3", [n_hidden_2, 10],
                        initializer=weight_init3)
            b3 = tf.get_variable("b3", [10],
```

```
                 initializer=bias_init)

        #output = inference(x,W1,b1,W2,b2)
        output = inference(x)

        cost = loss(output, y)

        global_step = tf.Variable(0, name='global_step', trainable=False)

        train_op = training(cost, global_step)

        eval_op = evaluate(output, y)

        summary_op = tf.summary.merge_all()

        saver = tf.train.Saver()

        sess = tf.Session()

#        summary_writer = tf.summary.FileWriter("mlp_logs/",
#                                    graph_def=sess.graph_def)


        init_op = tf.global_variables_initializer()

        sess.run(init_op)

        # saver.restore(sess, "mlp_logs/model-checkpoint-66000")
        # PLOTTING EACH DIGIT
        #x[7]=0, x[6]=1, x[13]=2, x[1]=3, x[2]=4, x[27]=5, x[26]=6, x[25]=7, x[9]=8, x[8]=9
        mini_x, mini_y = mnist.train.next_batch(30)

        num=mini_x[7]
        num0=num.reshape(28,28)

        num=mini_x[6]
        num1=num.reshape(28,28)

        num=mini_x[13]
        num2=num.reshape(28,28)

        num=mini_x[1]
        num3=num.reshape(28,28)

        num=mini_x[2]
        num4=num.reshape(28,28)

        num=mini_x[27]
```

```
        num5=num.reshape(28,28)

        num=mini_x[26]
        num6=num.reshape(28,28)

        num=mini_x[25]
        num7=num.reshape(28,28)

        num=mini_x[9]
        num8=num.reshape(28,28)

        num=mini_x[8]
        num9=num.reshape(28,28)

#       plt.imshow(num1)
#

        scipy.misc.imsave('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A2 Output/0.jpg',
num0)
        scipy.misc.imsave('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A2 Output/1.jpg',
num1)
        scipy.misc.imsave('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A2 Output/2.jpg',
num2)
        scipy.misc.imsave('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A2 Output/3.jpg',
num3)
        scipy.misc.imsave('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A2 Output/4.jpg',
num4)
        scipy.misc.imsave('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A2 Output/5.jpg',
num5)
        scipy.misc.imsave('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A2 Output/6.jpg',
num6)
        scipy.misc.imsave('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A2 Output/7.jpg',
num7)
        scipy.misc.imsave('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A2 Output/8.jpg',
num8)
        scipy.misc.imsave('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A2 Output/9.jpg',
num9)

#       scipy.misc.imsave('outfile.jpg', image_array)


        # Training cycle
        for epoch in range(training_epochs):

            avg_cost = 0.
            total_batch = int(mnist.train.num_examples/batch_size)
            # Loop over all batches
            for i in range(total_batch):
```

```
            minibatch_x, minibatch_y = mnist.train.next_batch(batch_size)
            # Fit training using batch data
            sess.run(train_op, feed_dict={x: minibatch_x, y: minibatch_y})
            # Compute average loss
            avg_cost += sess.run(cost, feed_dict={x: minibatch_x, y: minibatch_y})/total_batch
          # Display logs per epoch step
          if epoch % display_step == 0:
            print("Epoch:", '%04d' % (epoch+1), "cost =", "{:.9f}".format(avg_cost))

            accuracy = sess.run(eval_op, feed_dict={x: mnist.validation.images, y:
mnist.validation.labels})

            print("Validation Error:", (1 - accuracy))

            summary_str = sess.run(summary_op, feed_dict={x: minibatch_x, y: minibatch_y})
#               summary_writer.add_summary(summary_str, sess.run(global_step))

#               saver.save(sess, "mlp_logs/model-checkpoint", global_step=global_step)


        print("Optimization Finished!")


        accuracy = sess.run(eval_op, feed_dict={x: mnist.test.images, y: mnist.test.labels})

            #PLOTTING THE FIRST 10 WEIGHTS OF FIRST HIDDEN LAYER
        for wi in range (9):
          weights=sess.run(W1)
          im=weights[:,wi]
          wim=im.reshape(28,28)
          filename = "C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A2
Output/W1_%d.jpg"%wi
          scipy.misc.imsave(filename, wim)


        print("Test Accuracy:", accuracy)

      with open('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part A2 Output/Accuracy.txt', 'w')
as f:
        print('Accuracy = ',accuracy, file=f)

      f.close()
```

PART B – NETWORK A

```
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 29 17:21:35 2017

@author: Deepak

Part B - Netwok A
"""


import sys
sys.path.append('../../')
sys.path.append('../')
import numpy as np
import tensorflow as tf
import time, shutil, os
from fdl_examples.datatools import input_data
import matplotlib.pyplot as plt
import scipy
from scipy.ndimage.interpolation import rotate
import random
from sklearn.metrics import confusion_matrix
import numpy

# read in MNIST data --------------------------------------------------
mnist = input_data.read_data_sets("../../data/", one_hot=True)


# run network -----------------------------------------------------------

# Parameters
learning_rate = 0.01
training_epochs = 10 # NOTE: you'll want to eventually change this
batch_size = 100
display_step = 1


def inference(x,W,b):
   output = tf.nn.softmax(tf.matmul(x, W) + b)

   w_hist = tf.summary.histogram("weights", W)
   b_hist = tf.summary.histogram("biases", b)
   y_hist = tf.summary.histogram("output", output)

   return output
```

```
def loss(output, y):
    dot_product = y * tf.log(output)

    # Reduction along axis 0 collapses each column into a single
    # value, whereas reduction along axis 1 collapses each row
    # into a single value. In general, reduction along axis i
    # collapses the ith dimension of a tensor to size 1.
    xentropy = -tf.reduce_sum(dot_product, axis=1)

    loss = tf.reduce_mean(xentropy)

    return loss

def training(cost, global_step):

    tf.summary.scalar("cost", cost)
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    train_op = optimizer.minimize(cost, global_step=global_step)

    return train_op


def evaluate(output, y):
    correct_prediction = tf.equal(tf.argmax(output, 1), tf.argmax(y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

    tf.summary.scalar("validation error", (1.0 - accuracy))

    return accuracy

def rotateImage(im):
#    print(im.shape)
    im_size=round(np.size(im)/784)
    for j in range(im_size):
        img=im[j]
        img=img.reshape(28,28)
        rot_image=rotate(img, random.randint(0,360),reshape=False)
#        print(rot_image.shape)
        rot_image=rot_image.reshape(784)
        im[j]=rot_image
#    print(im.shape)
    return im


if __name__ == '__main__':
#    if os.path.exists("logistic_logs/"):
#        shutil.rmtree("logistic_logs/")
```

```
with tf.Graph().as_default():

    x = tf.placeholder("float", [None, 784]) # mnist data image of shape 28*28=784
    y = tf.placeholder("float", [None, 10]) # 0-9 digits recognition => 10 classes

    init = tf.constant_initializer(value=0)
    W = tf.get_variable("W", [784, 10],
                initializer=init)
    b = tf.get_variable("b", [10],
                initializer=init)

    output = inference(x,W,b)

    cost = loss(output, y)

    global_step = tf.Variable(0, name='global_step', trainable=False)

    train_op = training(cost, global_step)

    eval_op = evaluate(output, y)

    summary_op = tf.summary.merge_all()

    saver = tf.train.Saver()

    sess = tf.Session()

#     summary_writer = tf.summary.FileWriter("logistic_logs/",
#                                graph_def=sess.graph_def)


    init_op = tf.global_variables_initializer()

    sess.run(init_op)


    argMax_y=tf.argmax(y,1)
    argMax_output=tf.argmax(output,1)


#     print(output)
#     matrix = np.zeros((10,10))
#     y=mnist.test.labels
#     outputs=mnist
#     for indx in range(10):
#         row = y[indx]
#         column = output[indx]
#         matrix[row,column] += 1
```

```
#       print(matrix)

#       mini_x, mini_y = mnist.train.next_batch(30)

#       num=mini_x[7]
#       num0=num.reshape(28,28)
#       rot_num0=scipy.misc.imrotate(num0, random.randint(0,360), interp='bilinear')
#       plt.imshow(rot_num0)

      total_batch = int(mnist.train.num_examples/batch_size)
      z_x1,z_y=mnist.train.next_batch(batch_size*total_batch)
          # Displaying original image
      num=z_x1[0]
      num0=num.reshape(28,28)
      plt.subplot(211)
      plt.title('Original image in training set', fontsize=25)
      plt.xticks([], [])
      plt.yticks([], [])
      plt.imshow(num0)

      z_x=rotateImage(z_x1)


      # Training cycle
      for epoch in range(training_epochs):

        avg_cost = 0.
        total_batch = int(mnist.train.num_examples/batch_size)
        z_x,z_y=mnist.train.next_batch(batch_size*total_batch)

        # Loop over all batches
        for i in range(total_batch):
#             minibatch_x, minibatch_y = mnist.train.next_batch(batch_size)

#             minibatch_x1=minibatch_x
#             minibatch_x=rotateImage(minibatch_x)
            minibatch_x=z_x[i*100:(i+1)*100,0:784]
            minibatch_y=z_y[i*100:(i+1)*100,0:784]


            # Fit training using batch data
            sess.run(train_op, feed_dict={x: minibatch_x, y: minibatch_y})
            # Compute average loss
            avg_cost += sess.run(cost, feed_dict={x: minibatch_x, y: minibatch_y})/total_batch
        # Display logs per epoch step
        if epoch % display_step == 0:
            print("Epoch:", '%04d' % (epoch+1), "cost =", "{:.9f}".format(avg_cost))
```

```
            accuracy = sess.run(eval_op, feed_dict={x: rotateImage(mnist.validation.images), y: 
mnist.validation.labels})

            print("Validation Error:", (1 - accuracy))


            summary_str = sess.run(summary_op, feed_dict={x: minibatch_x, y: minibatch_y})
            #summary_writer.add_summary(summary_str, sess.run(global_step))

            #saver.save(sess, "logistic_logs/model-checkpoint", global_step=global_step)
      num1=z_x[0]
      num1=num.reshape(28,28)

      plt.subplot(212)
      plt.imshow(num1)
      plt.title('Rotated image in training set', fontsize=25)
      plt.xticks([], [])
      plt.yticks([], [])
      plt.savefig('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part B1 
Output/Original_Rotated.jpg')



      print("Optimization Finished!")

      accuracy = sess.run(eval_op, feed_dict={x: rotateImage(mnist.test.images), y: mnist.test.labels})

      print("Test Accuracy:", accuracy)
      with open('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part B1 Output/Accuracy.txt', 'w') 
as f:
         print('Accuracy = ',accuracy, file=f)

      f.close()


      # CONFUSION MATRIX GENERATION

      y_test = sess.run(argMax_y, feed_dict={x: mnist.test.images, y: mnist.test.labels})
      output_test = sess.run(argMax_output, feed_dict={x: mnist.test.images, y: mnist.test.labels})
      C_matrix = np.zeros((10,10))
      count=len(y_test)
      for i in range(0,count):
         true_value = y_test[i]
         est_value = output_test[i]
         C_matrix[true_value,est_value] += 1
```

```
    plt.figure(figsize = (15,15))
    rlabels = ['  0  ','  1  ','  2  ','  3  ','  4  ','  5  ','  6  ','  7  ','  8  ','  9  ']
    clabels = ['0','1','2','3','4','5','6','7','8','9']
    ytable = plt.table(cellText=np.int_(C_matrix), loc='center', rowLabels=rlabels,colLabels=clabels)
    ytable.set_fontsize(14)

    table_props = ytable.properties()
    table_cells = table_props['child_artists']
    for cell in table_cells:
        cell.set_height(0.09)
        cell.set_width(0.09)

#       plt.rcParams['axes.labelweight'] = 'bold'
    plt.xticks([], [])
    plt.yticks([], [])
    plt.ylabel('True Values', fontsize=25)
    plt.xlabel('Estimated Values', fontsize=25)
    plt.title('Confusion Matrix', fontsize=45)

#       ax = plt.axes()
#       ax.xaxis.set_ticks_position('none')
#           tb = plt.gca()
#           tb.set_xticks([])
#           tb.set_yticks([])

    plt.show()
    plt.savefig('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part B1
Output/Confusion_Matrix.jpg')
```

PART B – NETWORK B

```
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 29 19:49:16 2017

@author: Deepak

Part B - Network A
"""

import sys
sys.path.append('../../')
sys.path.append('../')
from fdl_examples.datatools import input_data
mnist = input_data.read_data_sets("../../data/", one_hot=True)
from scipy.ndimage import rotate
import random
import tensorflow as tf
import time, shutil, os
import matplotlib.pyplot as plt
import numpy as np

# Architecture
n_hidden_1 = 256
n_hidden_2 = 256

# Parameters
learning_rate = 0.01
training_epochs = 10 # NOTE: you'll want to eventually change this
batch_size = 100
display_step = 1

#def layer(input, weight_shape, bias_shape):
def layer(input, W, b):
    return tf.nn.relu(tf.matmul(input, W) + b)

#def inference(x,W1,b1,W2,b2):
def inference(x):
    with tf.variable_scope("hidden_1"):
        hidden_1 = layer(x, W1,b1)

    with tf.variable_scope("hidden_2"):
        hidden_2 = layer(hidden_1,W2,b2)

    with tf.variable_scope("output"):
        #output = layer(hidden_2, [n_hidden_2, 10], [10])
```

```
        output = layer(hidden_2, W3,b3)

    return output

def loss(output, y):
    xentropy = tf.nn.softmax_cross_entropy_with_logits(logits=output, labels=y)
    loss = tf.reduce_mean(xentropy)
    return loss

def training(cost, global_step):
    tf.summary.scalar("cost", cost)
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    train_op = optimizer.minimize(cost, global_step=global_step)
    return train_op


def evaluate(output, y):
    correct_prediction = tf.equal(tf.argmax(output, 1), tf.argmax(y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    tf.summary.scalar("validation", accuracy)
    return accuracy

def rotateImage(im):
#    print(im.shape)
    im_size=round(np.size(im)/784)
    for j in range(im_size):
        img=im[j]
        img=img.reshape(28,28)
        rot_image=rotate(img, random.randint(0,360),reshape=False)
#        print(rot_image.shape)
        rot_image=rot_image.reshape(784)
        im[j]=rot_image
#    print(im.shape)
    return im


if __name__ == '__main__':

#    if os.path.exists("mlp_logs/"):
#        shutil.rmtree("mlp_logs/")

    with tf.Graph().as_default():

        with tf.variable_scope("mlp_model"):

            x = tf.placeholder("float", [None, 784]) # mnist data image of shape 28*28=784
            y = tf.placeholder("float", [None, 10]) # 0-9 digits recognition => 10 classes
```

```
weight_init1 = tf.random_normal_initializer(stddev=(2.0/784)**0.5)
weight_init2 = tf.random_normal_initializer(stddev=(2.0/n_hidden_1)**0.5)
weight_init3 = tf.random_normal_initializer(stddev=(2.0/n_hidden_2)**0.5)
bias_init = tf.constant_initializer(value=0.)

W1 = tf.get_variable("W1", [784, n_hidden_1],
            initializer=weight_init1)
b1 = tf.get_variable("b1", [n_hidden_1],
            initializer=bias_init)
W2 = tf.get_variable("W2", [n_hidden_1, n_hidden_2],
            initializer=weight_init2)
b2 = tf.get_variable("b2", [n_hidden_2],
            initializer=bias_init)
W3 = tf.get_variable("W3", [n_hidden_2, 10],
            initializer=weight_init3)
b3 = tf.get_variable("b3", [10],
            initializer=bias_init)

#output = inference(x,W1,b1,W2,b2)
output = inference(x)

cost = loss(output, y)

global_step = tf.Variable(0, name='global_step', trainable=False)

train_op = training(cost, global_step)

eval_op = evaluate(output, y)

summary_op = tf.summary.merge_all()

saver = tf.train.Saver()

sess = tf.Session()

summary_writer = tf.summary.FileWriter("mlp_logs/",
                    graph_def=sess.graph_def)


init_op = tf.global_variables_initializer()

sess.run(init_op)

argMax_y=tf.argmax(y,1)
argMax_output=tf.argmax(output,1)

# saver.restore(sess, "mlp_logs/model-checkpoint-66000")
```

```
        total_batch = int(mnist.train.num_examples/batch_size)
        z_x1,z_y=mnist.train.next_batch(batch_size*total_batch)
            # Displaying original image
        num=z_x1[0]
        num0=num.reshape(28,28)
        plt.subplot(211)
        plt.title('Original image in training set', fontsize=25)
        plt.xticks([], [])
        plt.yticks([], [])
        plt.imshow(num0)

        z_x=rotateImage(z_x1)


        # Training cycle
        for epoch in range(training_epochs):

          avg_cost = 0.
          total_batch = int(mnist.train.num_examples/batch_size)
          z_x,z_y=mnist.train.next_batch(batch_size*total_batch)

          # Loop over all batches
          for i in range(total_batch):
#              minibatch_x, minibatch_y = mnist.train.next_batch(batch_size)

##              minibatch_x1=minibatch_x
#              minibatch_x=rotateImage(minibatch_x)
             minibatch_x=z_x[i*100:(i+1)*100,0:784]
             minibatch_y=z_y[i*100:(i+1)*100,0:784]

             # Fit training using batch data
             sess.run(train_op, feed_dict={x: minibatch_x, y: minibatch_y})
             # Compute average loss
             avg_cost += sess.run(cost, feed_dict={x: minibatch_x, y: minibatch_y})/total_batch
          # Display logs per epoch step
          if epoch % display_step == 0:
             print("Epoch:", '%04d' % (epoch+1), "cost =", "{:.9f}".format(avg_cost))

             accuracy = sess.run(eval_op, feed_dict={x: rotateImage(mnist.validation.images), y:
mnist.validation.labels})

             print("Validation Error:", (1 - accuracy))

             summary_str = sess.run(summary_op, feed_dict={x: minibatch_x, y: minibatch_y})
#              summary_writer.add_summary(summary_str, sess.run(global_step))

#              saver.save(sess, "mlp_logs/model-checkpoint", global_step=global_step)
```

```
        num1=z_x[0]
        num1=num.reshape(28,28)

        plt.subplot(212)
        plt.imshow(num1)
        plt.title('Rotated image in training set', fontsize=25)
        plt.xticks([], [])
        plt.yticks([], [])
        plt.savefig('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part B2
Output/Original_Rotated.jpg')


        print("Optimization Finished!")


        accuracy = sess.run(eval_op, feed_dict={x: rotateImage(mnist.test.images), y: mnist.test.labels})

        print("Test Accuracy:", accuracy)
        with open('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part B2 Output/Accuracy.txt', 'w')
as f:
            print('Accuracy = ',accuracy, file=f)

        f.close()


    # CONFUSION MATRIX GENERATION

    y_test = sess.run(argMax_y, feed_dict={x: rotateImage(mnist.test.images), y: mnist.test.labels})
    output_test = sess.run(argMax_output, feed_dict={x: rotateImage(mnist.test.images), y:
mnist.test.labels})
    C_matrix = np.zeros((10,10))
    count=len(y_test)
    for i in range(0,count):
        true_value = y_test[i]
        est_value = output_test[i]
        C_matrix[true_value,est_value] += 1


    plt.figure(figsize = (15,15))
    rlabels = ['  0  ','  1  ','  2  ','  3  ','  4  ','  5  ','  6  ','  7  ','  8  ','  9  ']
    clabels = ['0','1','2','3','4','5','6','7','8','9']
    ytable = plt.table(cellText=np.int_(C_matrix), loc='center', rowLabels=rlabels,colLabels=clabels)
    ytable.set_fontsize(14)

    table_props = ytable.properties()
    table_cells = table_props['child_artists']
    for cell in table_cells:
```

```
        cell.set_height(0.09)
        cell.set_width(0.09)

#     plt.rcParams['axes.labelweight'] = 'bold'
      plt.xticks([], [])
      plt.yticks([], [])
      plt.ylabel('True Values', fontsize=25)
      plt.xlabel('Estimated Values', fontsize=25)
      plt.title('Confusion Matrix', fontsize=45)

#     ax = plt.axes()
#     ax.xaxis.set_ticks_position('none')
#         tb = plt.gca()
#         tb.set_xticks([])
#         tb.set_yticks([])

      plt.show()
      plt.savefig('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part B2
Output/Confusion_Matrix.jpg')
```

PART C – NETWORK A

```
# -*- coding: utf-8 -*-
"""
Created on Sat Sep 30 10:34:39 2017

Part C - Network A

@author: Deepak
"""



import sys
sys.path.append('../../')
sys.path.append('../')
import numpy as np
import tensorflow as tf
import time, shutil, os
from fdl_examples.datatools import input_data
mnist = input_data.read_data_sets("../../data/", one_hot=True)
import matplotlib.pyplot as plt
import scipy
import math
from scipy.ndimage.interpolation import zoom


# read in MNIST data -----------------------------------------------
mnist = input_data.read_data_sets("../../data/", one_hot=True)


# run network ---------------------------------------------------------

# Parameters
learning_rate = 0.01
training_epochs = 10 # NOTE: you'll want to eventually change this
batch_size = 100
display_step = 1


def inference(x,W,b):
    output = tf.nn.softmax(tf.matmul(x, W) + b)

    w_hist = tf.summary.histogram("weights", W)
    b_hist = tf.summary.histogram("biases", b)
    y_hist = tf.summary.histogram("output", output)
```

```
    return output

def loss(output, y):
    dot_product = y * tf.log(output)

    # Reduction along axis 0 collapses each column into a single
    # value, whereas reduction along axis 1 collapses each row
    # into a single value. In general, reduction along axis i
    # collapses the ith dimension of a tensor to size 1.
    xentropy = -tf.reduce_sum(dot_product, axis=1)

    loss = tf.reduce_mean(xentropy)

    return loss

def training(cost, global_step):

    tf.summary.scalar("cost", cost)
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    train_op = optimizer.minimize(cost, global_step=global_step)

    return train_op


def evaluate(output, y):
    correct_prediction = tf.equal(tf.argmax(output, 1), tf.argmax(y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

    tf.summary.scalar("validation error", (1.0 - accuracy))

    return accuracy

def ScaleImage(im):
#    print(im.shape)
#    all_scaled_image=[]
    im_size=round(np.size(im)/784)
    for j in range(im_size):
        img=im[j]
        img=img.reshape(28,28)
        scale_amount=np.random.uniform(low=0.5, high=1.0)
#        print(scale_amount)
        scaled_image=zoom(img,scale_amount)
        scaled_image_size=[round(math.sqrt(scaled_image.size)),round(math.sqrt(scaled_image.size))]
        size = (28,28)
        background =np.zeros(size)
        offset = [round((size[0] - scaled_image_size[0]) / 2), round((size[1] - scaled_image_size[1]) / 2)]
#            offset_y = (size[1] - scaled_image_size[1]) / 2
```

```
        background[offset[0]:offset[0] + scaled_image_size[0], offset[1]:offset[1] +
scaled_image_size[1]]=scaled_image
        scaled_padded_image =background
        scaled_padded_image=scaled_padded_image.reshape(784)
        im[j]=scaled_padded_image
#    print(im.shape)
    return im



if __name__ == '__main__':
#    if os.path.exists("logistic_logs/"):
#        shutil.rmtree("logistic_logs/")

    with tf.Graph().as_default():

        x = tf.placeholder("float", [None, 784]) # mnist data image of shape 28*28=784
        y = tf.placeholder("float", [None, 10]) # 0-9 digits recognition => 10 classes

        init = tf.constant_initializer(value=0)
        W = tf.get_variable("W", [784, 10],
                         initializer=init)
        b = tf.get_variable("b", [10],
                         initializer=init)

        output = inference(x,W,b)

        cost = loss(output, y)

        global_step = tf.Variable(0, name='global_step', trainable=False)

        train_op = training(cost, global_step)

        eval_op = evaluate(output, y)

        summary_op = tf.summary.merge_all()

        saver = tf.train.Saver()

        sess = tf.Session()

#        summary_writer = tf.summary.FileWriter("logistic_logs/",
#                                            graph_def=sess.graph_def)


        init_op = tf.global_variables_initializer()

        sess.run(init_op)
```

```
        argMax_y=tf.argmax(y,1)
        argMax_output=tf.argmax(output,1)

        total_batch = int(mnist.train.num_examples/batch_size)
        z_x1,z_y=mnist.train.next_batch(batch_size*total_batch)
            # Displaying original image
        num=z_x1[5]
        num0=num.reshape(28,28)
        plt.subplot(211)
        plt.title('Original image in training set', fontsize=25)
        plt.xticks([], [])
        plt.yticks([], [])
        plt.imshow(num0)

        z_x=ScaleImage(z_x1)


        # Training cycle
        for epoch in range(training_epochs):

          avg_cost = 0.
          # Loop over all batches
          for i in range(total_batch):
#             minibatch_x, minibatch_y = mnist.train.next_batch(batch_size)
            minibatch_x=z_x[i*100:(i+1)*100,0:784]
            minibatch_y=z_y[i*100:(i+1)*100,0:784]

#             minibatch_x1=minibatch_x

#             minibatch_x1=ScaleImage(minibatch_x1)
            # Fit training using batch data
            sess.run(train_op, feed_dict={x: minibatch_x, y: minibatch_y})
            # Compute average loss
            avg_cost += sess.run(cost, feed_dict={x: minibatch_x, y: minibatch_y})/total_batch
          # Display logs per epoch step
          if epoch % display_step == 0:
            print("Epoch:", '%04d' % (epoch+1), "cost =", "{:.9f}".format(avg_cost))

            accuracy = sess.run(eval_op, feed_dict={x: ScaleImage(mnist.validation.images), y:
mnist.validation.labels})

            print("Validation Error:", (1 - accuracy))

            summary_str = sess.run(summary_op, feed_dict={x: minibatch_x, y: minibatch_y})
            #summary_writer.add_summary(summary_str, sess.run(global_step))

            #saver.save(sess, "logistic_logs/model-checkpoint", global_step=global_step)
```

```
    #Displaying scaled image
    num1=z_x[5]
    num1=num.reshape(28,28)

    plt.subplot(212)
    plt.imshow(num1)
    plt.title('Scaled image in training set', fontsize=25)
    plt.xticks([], [])
    plt.yticks([], [])
    plt.savefig('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part C1
Output/Original_Scaled.jpg')




    print("Optimization Finished!")

    accuracy = sess.run(eval_op, feed_dict={x: ScaleImage(mnist.test.images), y: mnist.test.labels})

    print("Test Accuracy:", accuracy)
    with open('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part C1 Output/Accuracy.txt', 'w')
as f:
      print('Accuracy = ',accuracy, file=f)

    f.close()


    # CONFUSION MATRIX GENERATION

    y_test = sess.run(argMax_y, feed_dict={x: ScaleImage(mnist.test.images), y: mnist.test.labels})
    output_test = sess.run(argMax_output, feed_dict={x: ScaleImage(mnist.test.images), y:
mnist.test.labels})
    C_matrix = np.zeros((10,10))
    count=len(y_test)
    for i in range(0,count):
      true_value = y_test[i]
      est_value = output_test[i]
      C_matrix[true_value,est_value] += 1


    plt.figure(figsize = (15,15))
    rlabels = ['  0  ','  1  ','  2  ','  3  ','  4  ','  5  ','  6  ','  7  ','  8  ','  9  ']
    clabels = ['0','1','2','3','4','5','6','7','8','9']
    ytable = plt.table(cellText=np.int_(C_matrix), loc='center', rowLabels=rlabels,colLabels=clabels)
    ytable.set_fontsize(14)
```

```
    table_props = ytable.properties()
    table_cells = table_props['child_artists']
    for cell in table_cells:
        cell.set_height(0.09)
        cell.set_width(0.09)

#       plt.rcParams['axes.labelweight'] = 'bold'
    plt.xticks([], [])
    plt.yticks([], [])
    plt.ylabel('True Values', fontsize=25)
    plt.xlabel('Estimated Values', fontsize=25)
    plt.title('Confusion Matrix', fontsize=45)

#       ax = plt.axes()
#       ax.xaxis.set_ticks_position('none')
#           tb = plt.gca()
#           tb.set_xticks([])
#           tb.set_yticks([])

    plt.show()
    plt.savefig('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part C1
Output/Confusion_Matrix.jpg')

#           py.iplot(table, filename='Confusion_Matrix')
```

PART C – NETWORK B

```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct  2 21:11:42 2017

@author: Deepak

Part C - Network B
"""

import sys
sys.path.append('../../')
sys.path.append('../')
import numpy as np
import tensorflow as tf
import time, shutil, os
from fdl_examples.datatools import input_data
mnist = input_data.read_data_sets("../../data/", one_hot=True)
import matplotlib.pyplot as plt
import scipy
import math
from scipy.ndimage.interpolation import zoom

# Architecture
n_hidden_1 = 256
n_hidden_2 = 256

# Parameters
learning_rate = 0.01
training_epochs = 5 # NOTE: you'll want to eventually change this
batch_size = 100
display_step = 1

#def layer(input, weight_shape, bias_shape):
def layer(input, W, b):
    return tf.nn.relu(tf.matmul(input, W) + b)

#def inference(x,W1,b1,W2,b2):
def inference(x):
    with tf.variable_scope("hidden_1"):
        hidden_1 = layer(x, W1,b1)

    with tf.variable_scope("hidden_2"):
        hidden_2 = layer(hidden_1,W2,b2)
```

```
    with tf.variable_scope("output"):
        #output = layer(hidden_2, [n_hidden_2, 10], [10])
        output = layer(hidden_2, W3,b3)

    return output

def loss(output, y):
    xentropy = tf.nn.softmax_cross_entropy_with_logits(logits=output, labels=y)
    loss = tf.reduce_mean(xentropy)
    return loss

def training(cost, global_step):
    tf.summary.scalar("cost", cost)
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    train_op = optimizer.minimize(cost, global_step=global_step)
    return train_op


def evaluate(output, y):
    correct_prediction = tf.equal(tf.argmax(output, 1), tf.argmax(y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    tf.summary.scalar("validation", accuracy)
    return accuracy

def ScaleImage(im):
#    print(im.shape)
#    all_scaled_image=[]
    im_size=round(np.size(im)/784)
    for j in range(im_size):
        img=im[j]
        img=img.reshape(28,28)
        scale_amount=np.random.uniform(low=0.5, high=1.0)
#        print(scale_amount)
        scaled_image=zoom(img,scale_amount)
        scaled_image_size=[round(math.sqrt(scaled_image.size)),round(math.sqrt(scaled_image.size))]
        size = (28,28)
        background =np.zeros(size)
        offset = [round((size[0] - scaled_image_size[0]) / 2), round((size[1] - scaled_image_size[1]) / 2)]
#            offset_y = (size[1] - scaled_image_size[1]) / 2
        background[offset[0]:offset[0] + scaled_image_size[0], offset[1]:offset[1] +
scaled_image_size[1]]=scaled_image
        scaled_padded_image =background
        scaled_padded_image=scaled_padded_image.reshape(784)
        im[j]=scaled_padded_image
#    print(im.shape)
    return im

if __name__ == '__main__':
```

```
#    if os.path.exists("mlp_logs/"):
#        shutil.rmtree("mlp_logs/")

    with tf.Graph().as_default():

        with tf.variable_scope("mlp_model"):

            x = tf.placeholder("float", [None, 784]) # mnist data image of shape 28*28=784
            y = tf.placeholder("float", [None, 10]) # 0-9 digits recognition => 10 classes

            weight_init1 = tf.random_normal_initializer(stddev=(2.0/784)**0.5)
            weight_init2 = tf.random_normal_initializer(stddev=(2.0/n_hidden_1)**0.5)
            weight_init3 = tf.random_normal_initializer(stddev=(2.0/n_hidden_2)**0.5)
            bias_init = tf.constant_initializer(value=0.)

            W1 = tf.get_variable("W1", [784, n_hidden_1],
                        initializer=weight_init1)
            b1 = tf.get_variable("b1", [n_hidden_1],
                        initializer=bias_init)
            W2 = tf.get_variable("W2", [n_hidden_1, n_hidden_2],
                        initializer=weight_init2)
            b2 = tf.get_variable("b2", [n_hidden_2],
                        initializer=bias_init)
            W3 = tf.get_variable("W3", [n_hidden_2, 10],
                        initializer=weight_init3)
            b3 = tf.get_variable("b3", [10],
                        initializer=bias_init)

            #output = inference(x,W1,b1,W2,b2)
            output = inference(x)

            cost = loss(output, y)

            global_step = tf.Variable(0, name='global_step', trainable=False)

            train_op = training(cost, global_step)

            eval_op = evaluate(output, y)

            summary_op = tf.summary.merge_all()

            saver = tf.train.Saver()

            sess = tf.Session()

#            summary_writer = tf.summary.FileWriter("mlp_logs/",
#                                                   graph_def=sess.graph_def)
```

```
init_op = tf.global_variables_initializer()

sess.run(init_op)

argMax_y=tf.argmax(y,1)
argMax_output=tf.argmax(output,1)

# saver.restore(sess, "mlp_logs/model-checkpoint-66000")
total_batch = int(mnist.train.num_examples/batch_size)
z_x1,z_y=mnist.train.next_batch(batch_size*total_batch)
    # Displaying original image
num=z_x1[10]
num0=num.reshape(28,28)
plt.subplot(211)
plt.title('Original image in training set', fontsize=25)
plt.xticks([], [])
plt.yticks([], [])
plt.imshow(num0)

z_x=ScaleImage(z_x1)
# Training cycle
for epoch in range(training_epochs):

  avg_cost = 0.

  # Loop over all batches
  for i in range(total_batch):
#         minibatch_x, minibatch_y = mnist.train.next_batch(batch_size)
    minibatch_x=z_x[i*100:(i+1)*100,0:784]
    minibatch_y=z_y[i*100:(i+1)*100,0:784]
#         num=minibatch_x[0]
#         num0=num.reshape(28,28)
#         plt.imshow(num0)

#         minibatch_x1=minibatch_x

#         minibatch_x=ScaleImage(minibatch_x)

    # Fit training using batch data
    sess.run(train_op, feed_dict={x: minibatch_x, y: minibatch_y})
    # Compute average loss
    avg_cost += sess.run(cost, feed_dict={x: minibatch_x, y: minibatch_y})/total_batch
  # Display logs per epoch step
  if epoch % display_step == 0:
    print("Epoch:", '%04d' % (epoch+1), "cost =", "{:.9f}".format(avg_cost))
```

```
                accuracy = sess.run(eval_op, feed_dict={x: ScaleImage(mnist.validation.images), y:
mnist.validation.labels})

                print("Validation Error:", (1 - accuracy))

#                 summary_str = sess.run(summary_op, feed_dict={x: minibatch_x, y: minibatch_y})
#                 summary_writer.add_summary(summary_str, sess.run(global_step))

 #                saver.save(sess, "mlp_logs/model-checkpoint", global_step=global_step)


        num1=z_x[10]
        num1=num.reshape(28,28)

        plt.subplot(212)
        plt.imshow(num1)
        plt.title('Scaled image in training set', fontsize=25)
        plt.xticks([], [])
        plt.yticks([], [])
        plt.savefig('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part C2
Output/Original_Scaled.jpg')



        print("Optimization Finished!")


        accuracy = sess.run(eval_op, feed_dict={x: ScaleImage(mnist.test.images), y: mnist.test.labels})

        print("Test Accuracy:", accuracy)

        with open('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part C2 Output/Accuracy.txt', 'w')
as f:
            print('Accuracy = ',accuracy, file=f)

        f.close()


        # CONFUSION MATRIX GENERATION

        y_test = sess.run(argMax_y, feed_dict={x: ScaleImage(mnist.test.images), y: mnist.test.labels})
        output_test = sess.run(argMax_output, feed_dict={x: ScaleImage(mnist.test.images), y:
mnist.test.labels})
        C_matrix = np.zeros((10,10))
        count=len(y_test)
        for i in range(0,count):
            true_value = y_test[i]
            est_value = output_test[i]
```

```
            C_matrix[true_value,est_value] += 1


        plt.figure(figsize = (15,15))
        rlabels = ['  0  ','  1  ','  2  ','  3  ','  4  ','  5  ','  6  ','  7  ','  8  ','  9  ']
        clabels = ['0','1','2','3','4','5','6','7','8','9']
        ytable = plt.table(cellText=np.int_(C_matrix), loc='center', rowLabels=rlabels,colLabels=clabels)
        ytable.set_fontsize(14)

        table_props = ytable.properties()
        table_cells = table_props['child_artists']
        for cell in table_cells:
            cell.set_height(0.09)
            cell.set_width(0.09)

#       plt.rcParams['axes.labelweight'] = 'bold'
        plt.xticks([], [])
        plt.yticks([], [])
        plt.ylabel('True Values', fontsize=25)
        plt.xlabel('Estimated Values', fontsize=25)
        plt.title('Confusion Matrix', fontsize=45)

#       ax = plt.axes()
#       ax.xaxis.set_ticks_position('none')
#           tb = plt.gca()
#           tb.set_xticks([])
#           tb.set_yticks([])

        plt.show()
        plt.savefig('C:/Users/Deepak/Dropbox/Deep Learning/Project 2/Part C2
Output/Confusion_Matrix.jpg')

#           py.iplot(table, filename='Confusion_Matrix')
```