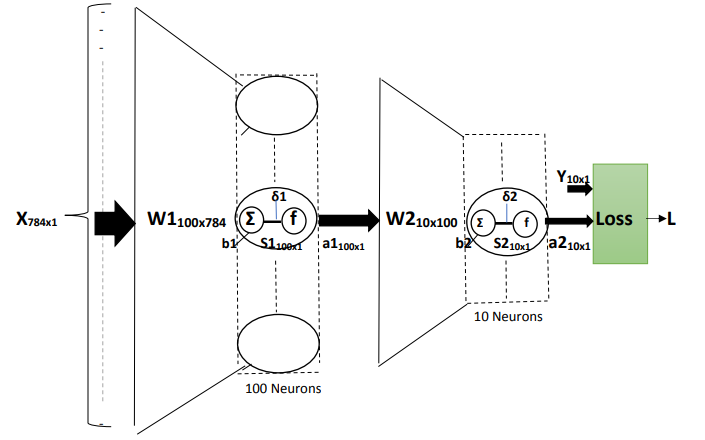
Goncalo Mendes

DEEP LEARNING

Assignment 3

For this assignment Implement a matrix based Dense Neural Network for character recognition on the MNIST dataset, with an output that could vary in 10 different outputs, the digits from 0 to 9. Below is the architecture that guided us through the mathematical derivations and then to adjust the matrix dimensions to make it work across the forward pass and the back propagation.

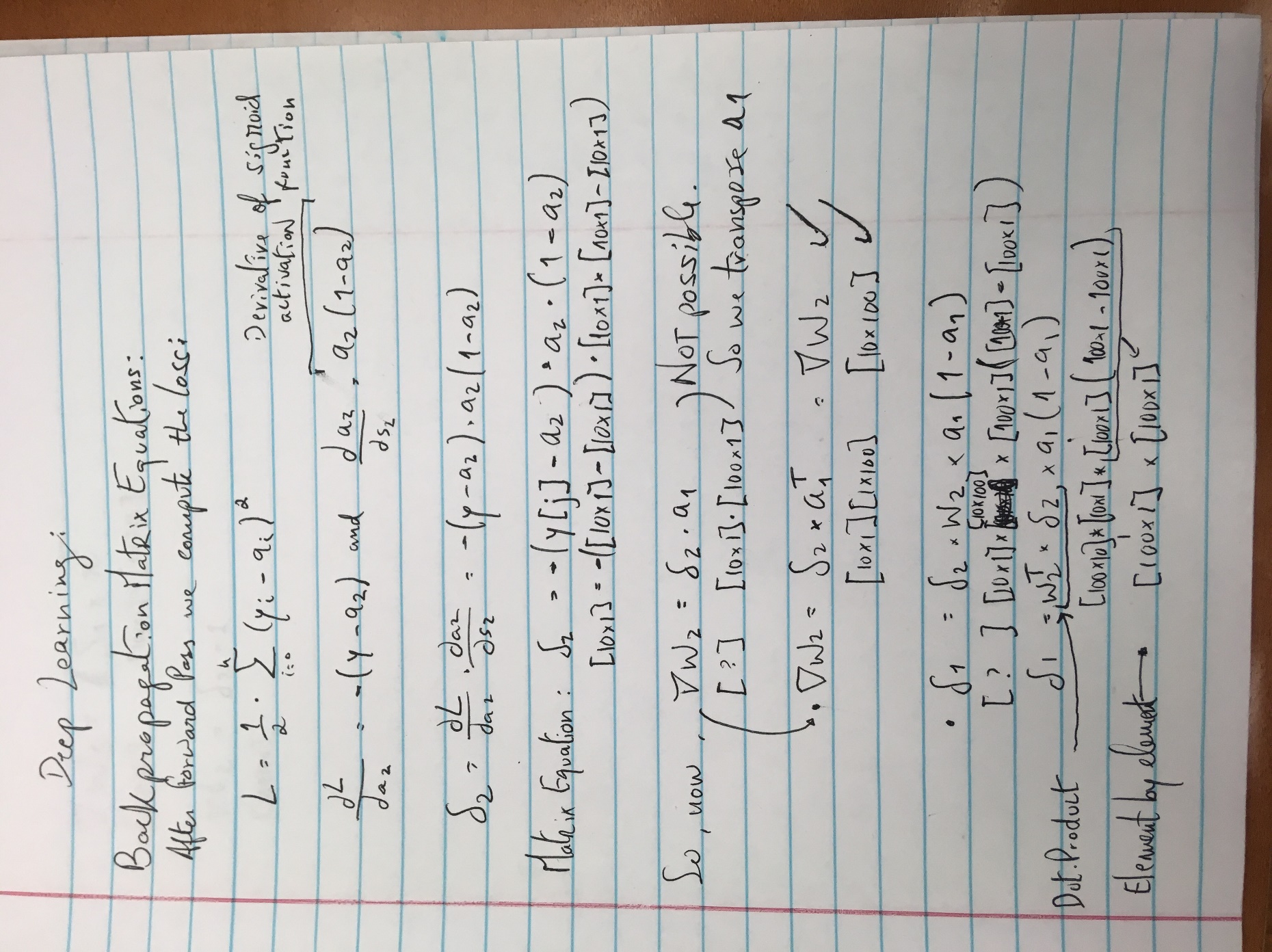


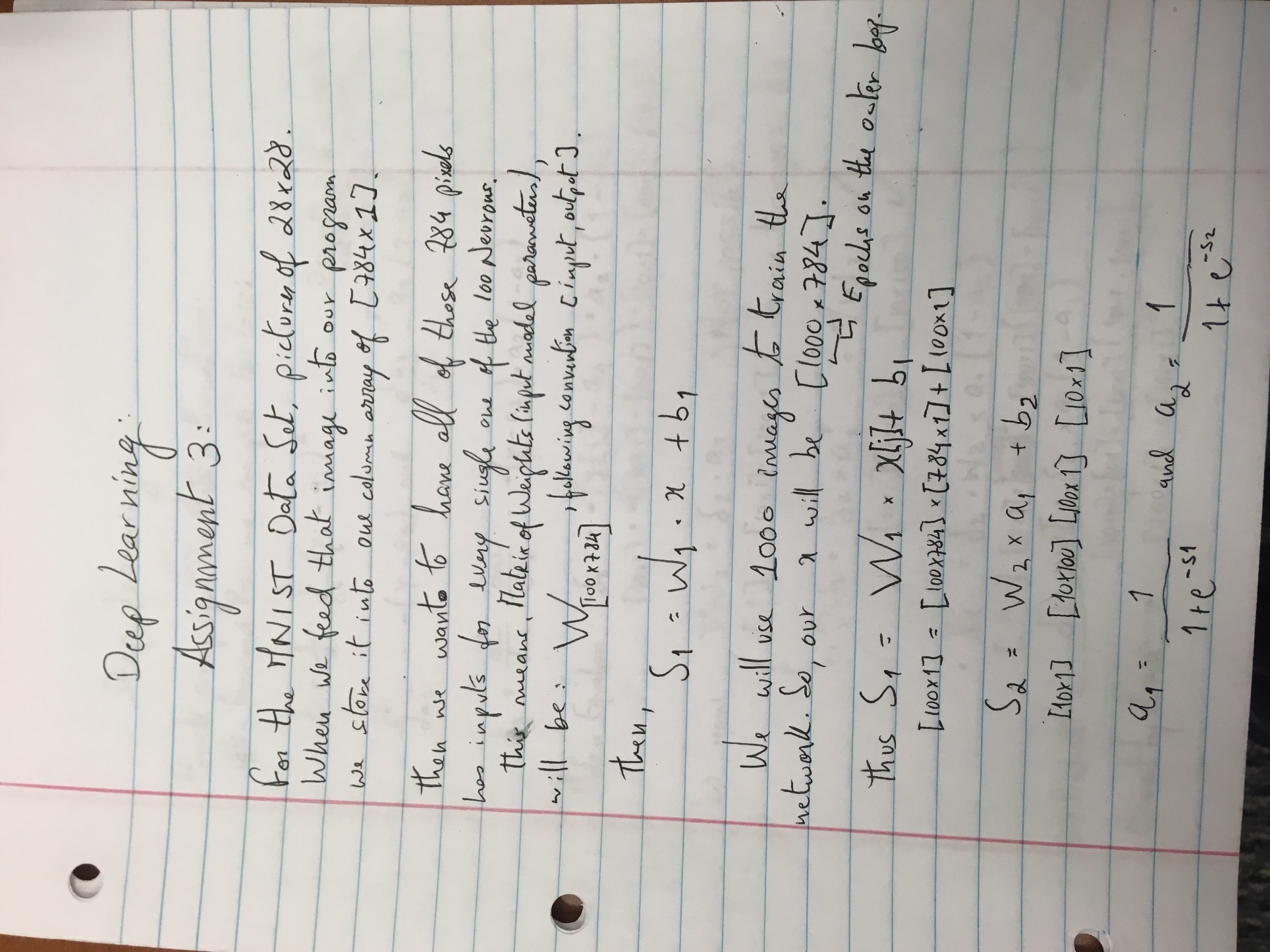
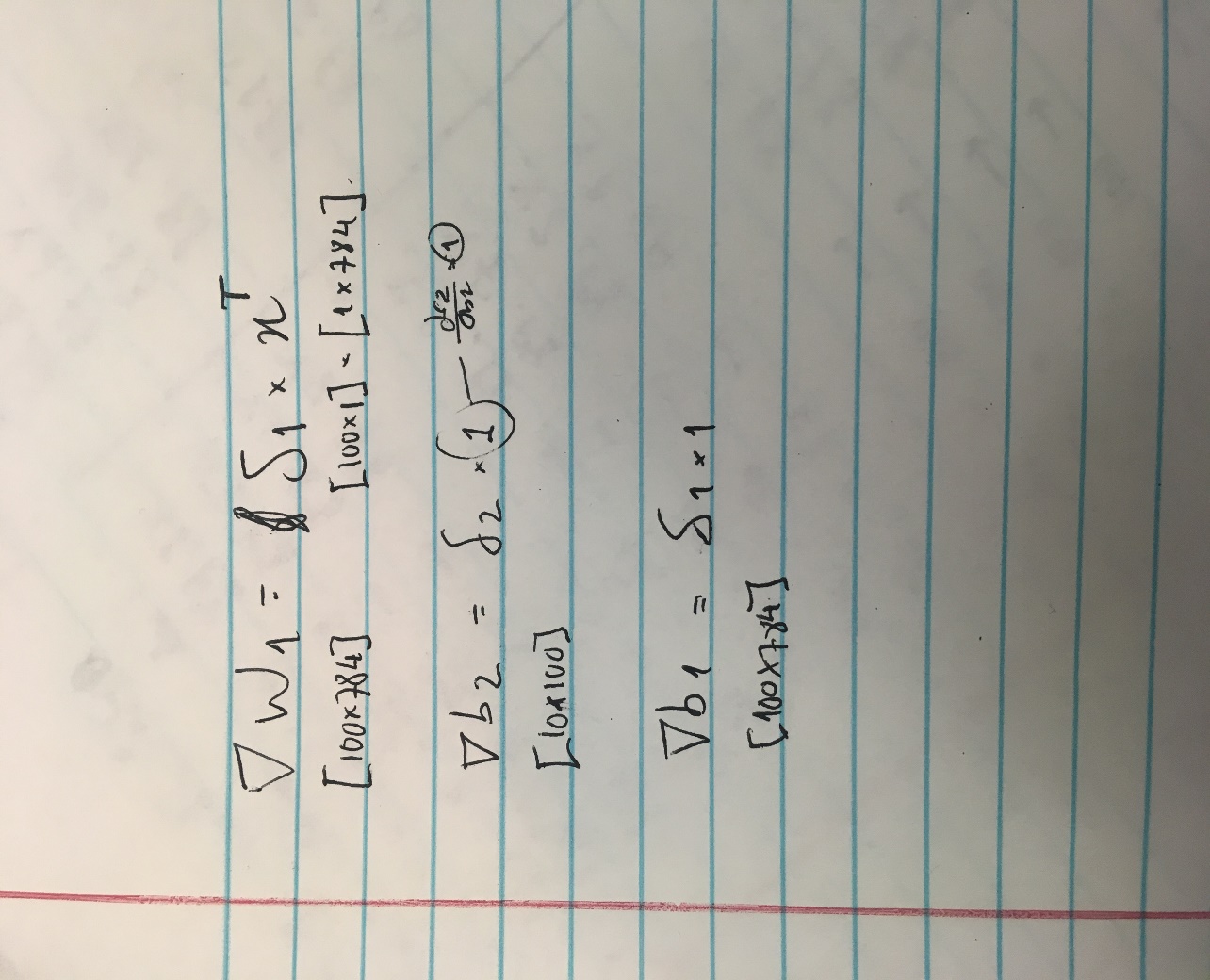
The pictures from the MNIST Data set were of 28x28 pixels, so when we get it, we store it a one-dimension array of 784x1. Then we want to have those pixels as the inputs into our neural network, going into every single one of the neurons of the first hidden layer. Which means each pixel is an input and will have the number of weights associated with it into each of the neurons.

So the matrix of Weights W will be of dimension 100x784 in case the layer has 100 neurons. And we are looping through this process for 1000 training images.

a)

Below is the mathematical derivations for the matrix equations for the forward pass and computing s1, a1, s2 and a2



Afterwards we coded the forward pass and backpropagation. For this processes we implemented two different methods. One was using the Stochastic Gradient Descent which consists in updating the weights every element of the training data. So every 100 epochs for example, if we have 1000 training images, it is making updates 100,000 times. The source code for this method is below.

import os

import sys

import cv2

import numpy as np

from sklearn.utils import shuffle

import matplotlib.pyplot as plt

train = np.empty((1000,28,28),dtype='float64')

trainY = np.zeros((1000,10,1))

test = np.empty((10000,28,28),dtype='float64')

testY = np.zeros((10000,10,1))

# Load in the images

i = 0

for filename in os.listdir('C:/Users/Windows User/Desktop/UB Spring 2019/Deep Learning/Assignment 3/Data/Training1000/'):

y = int(filename[0])

trainY[i,y] = 1.0

train[i] = cv2.imread('C:/Users/Windows User/Desktop/UB Spring 2019/Deep Learning/Assignment 3/Data/Training1000/{0}'.format(filename),0)/255.0 #

#for color, use 1

i = i + 1

i = 0 # read test data

for filename in os.listdir('C:/Users/Windows User/Desktop/UB Spring 2019/Deep Learning/Assignment 3/Data/Test10000'):

y = int(filename[0])

testY[i,y] = 1.0

test[i] = cv2.imread('C:/Users/Windows User/Desktop/UB Spring 2019/Deep Learning/Assignment 3/Data/Test10000/{0}'.format(filename),0)/255.0

i = i + 1

trainX = train.reshape(train.shape[0],train.shape[1]\*train.shape[2],1)

testX = test.reshape(test.shape[0],test.shape[1]\*test.shape[2],1)

numNeuronsLayer1 = 100

numNeuronsLayer2 = 10

numEpochs = 150

loss\_store = np.ndarray((numEpochs,1))

x\_axis = np.ndarray((numEpochs,1))

#---------------------NN------------------------

w1 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer1,784))

b1 = np.random.uniform(low=-1,high=1,size=(numNeuronsLayer1,1))

w2 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer2,numNeuronsLayer1))

b2 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer2,1))

learningRate = 0.1;

for n in range(0,numEpochs):

loss = 0

trainX,trainY = shuffle(trainX, trainY) # shuffle data for stochastic behavior

for i in range(trainX.shape[0]): # will go through one image per iteration.

# do forward pass

# your equations for the forward pass

s1 = np.dot(w1,trainX[i]) + b1

a1 = 1/(1+np.exp(-1\*s1)) # np.exp operates on the array

s2 = np.dot(w2,a1) + b2

a2 = 1/(1+np.exp(-1\*s2))

# do backprop and compute the gradients \* also works instead

delta2 = np.multiply(-np.multiply(trainY[i]-a2,a2),(1-a2))

delta1 = np.multiply(np.multiply(np.dot(np.transpose(w2),delta2),a1), (1-a1))

gradw2 = np.dot(delta2,np.transpose(a1))

gradw1 = np.dot(delta1,np.transpose(trainX[i]))

gradb1 = np.multiply(delta1,1)

gradb2 = np.multiply(delta2,1)

#loss += (0.5 \* ((a2-trainY[i])\*(a2-trainY[i]))).sum()

loss += (0.5 \* np.multiply((a2-trainY[i]),(a2-trainY[i]))).sum()

# your equations for computing the deltas and the gradients

# adjust the weights

w2 = w2 - learningRate \* gradw2

b2 = b2 - learningRate \* gradb2

w1 = w1 - learningRate \* gradw1

b1 = b1 - learningRate \* gradb1

loss\_store[n,0] = loss;

x\_axis[n,0] = n;

print("epoch = " + str(n) + " loss = " + (str(loss)))

print("done training , starting testing..")

accuracyCount = 0

#plot the fitted line

area = 10

colors =['black']

plt.scatter(x\_axis, loss\_store, s=area, c=colors, alpha=0.5, linewidths=8)

plt.title('LOSS / NUM EPOCHS')

plt.xlabel('NUM EPOCHS')

plt.ylabel('LOSS')

yfitted = loss\_store

line,= plt.plot(x\_axis, yfitted, '--', linewidth=2) #line plot

line.set\_color('red')

plt.show()

count = 0

for i in range(testY.shape[0]):

# do forward pass

s1 = np.dot(w1,testX[i]) + b1

a1 = 1/(1+np.exp(-1\*s1)) # np.exp operates on the array

s2 = np.dot(w2,a1) + b2

a2 = 1/(1+np.exp(-1\*s2))

count = count+1

print("Counter = " + str(count)) # determine index of maximum output value

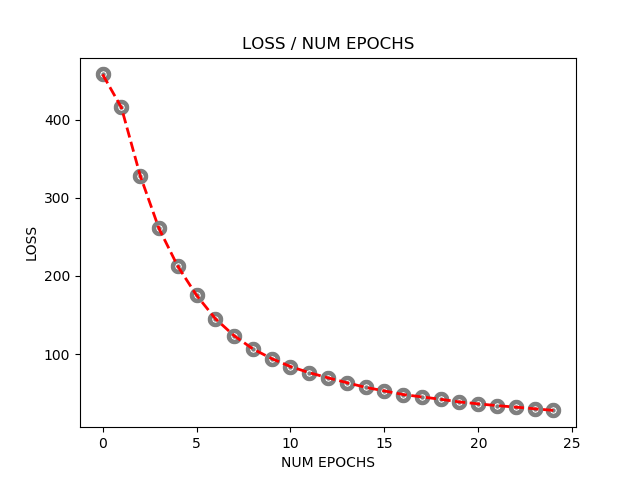
a2index = a2.argmax(axis = 0)

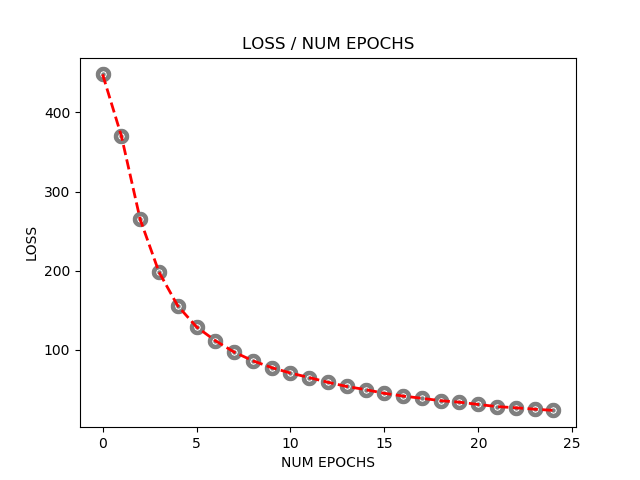
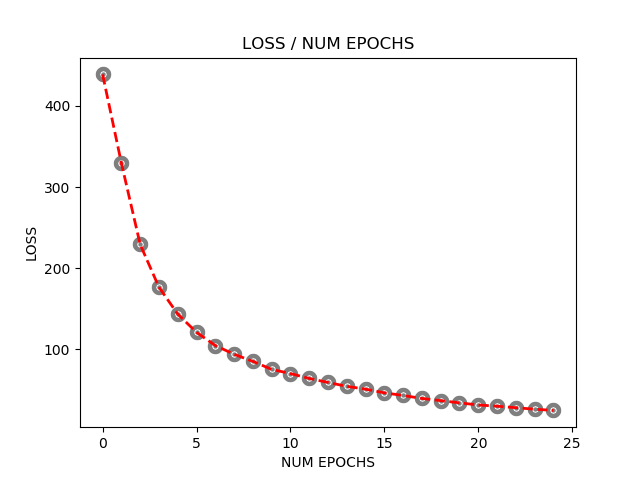
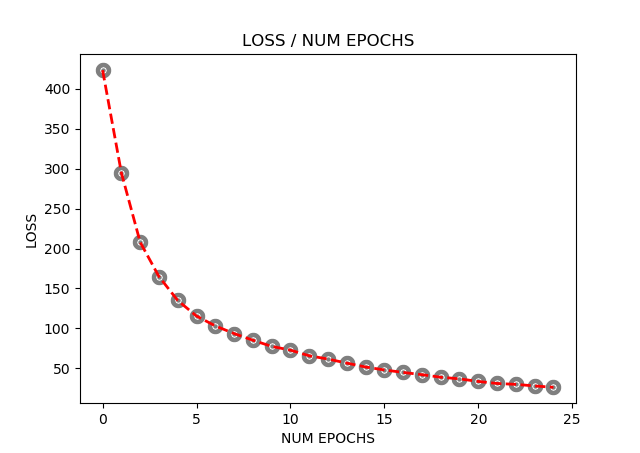
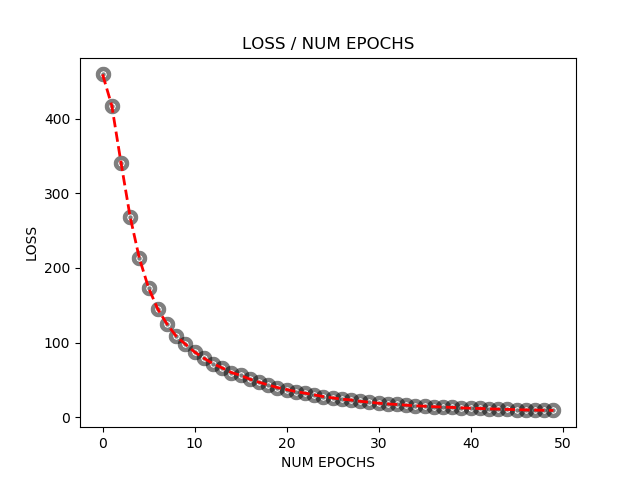
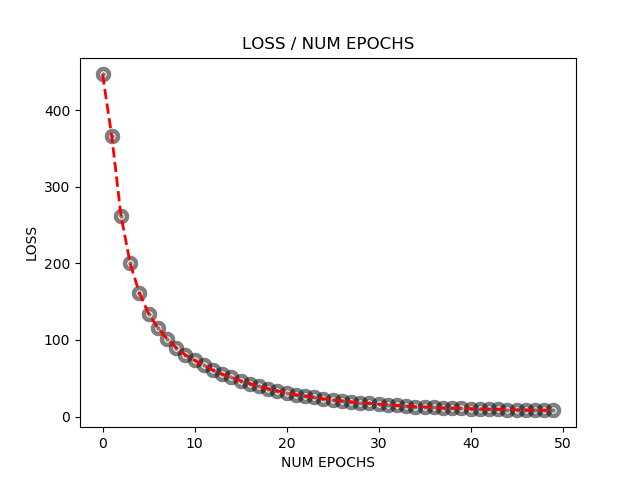
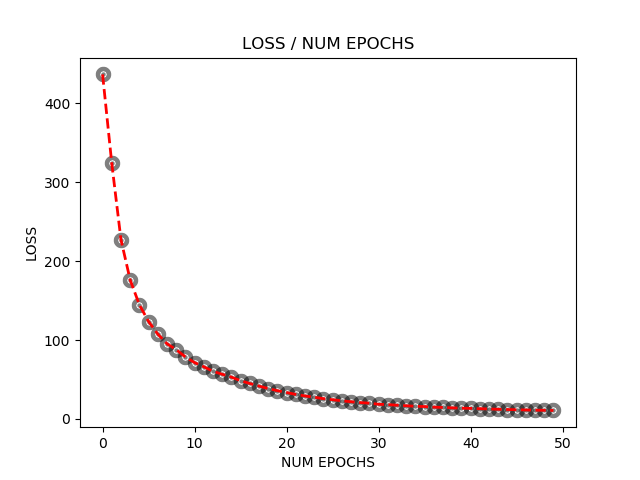
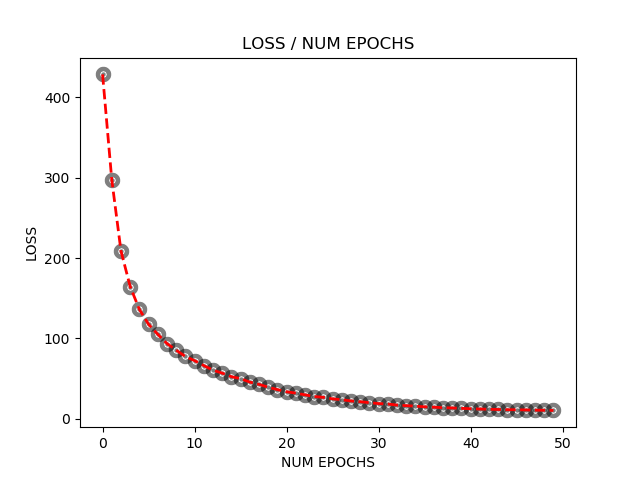
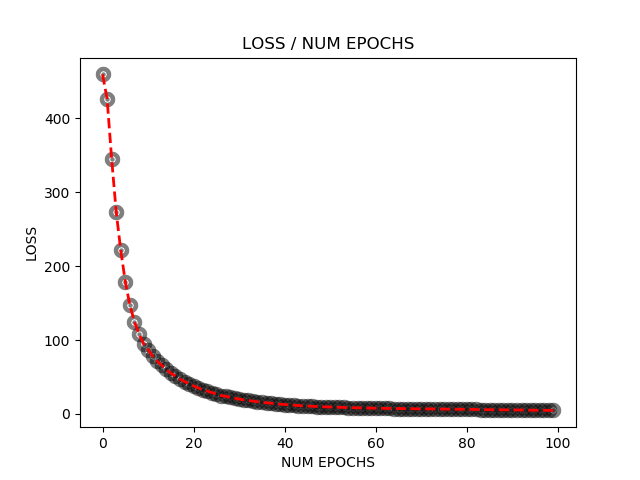
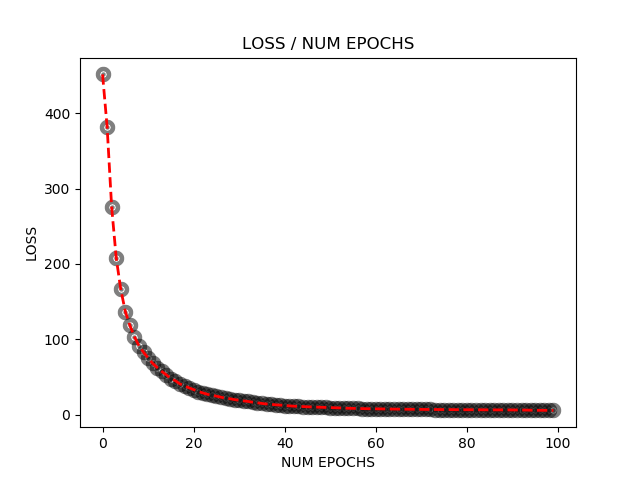
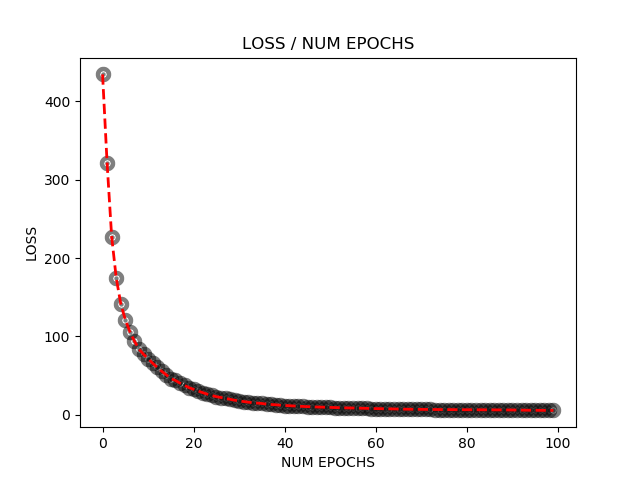
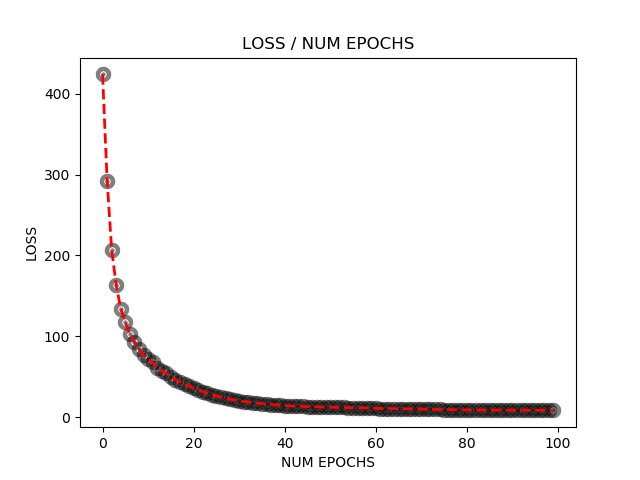
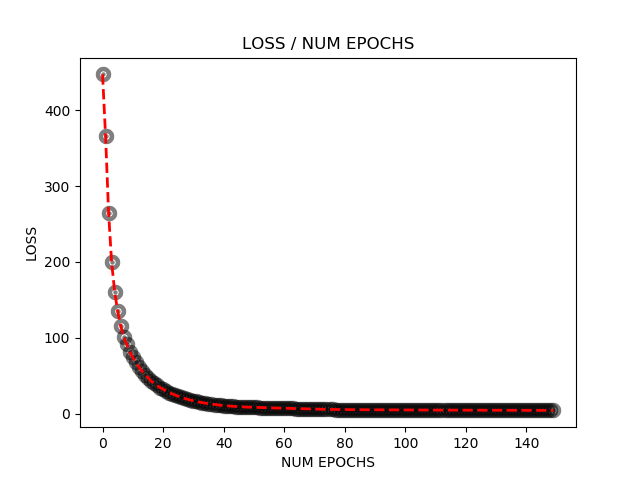
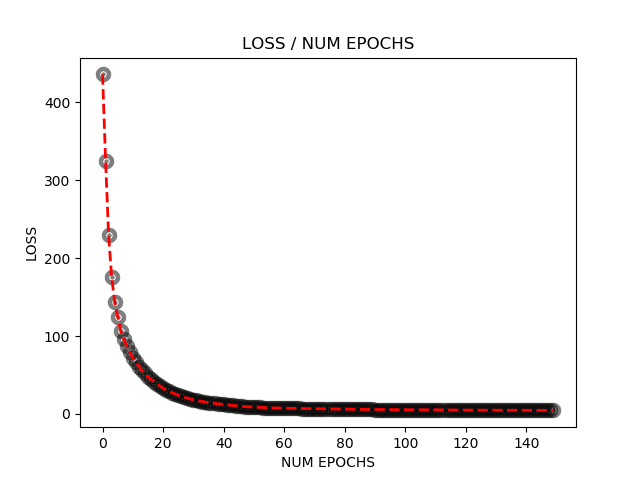
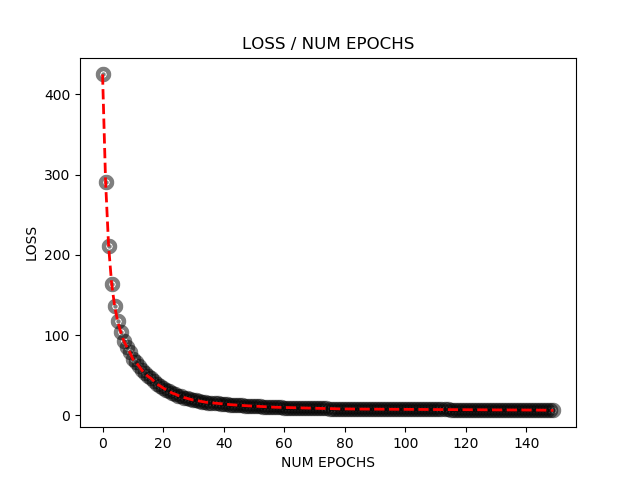
if (testY[i,a2index] == 1):

accuracyCount = accuracyCount + 1

print("Accuracy count = " + str(accuracyCount/10000.0))

For this method, we plotted the development of the loss over the number of epochs. We considered the cases of 25, 50, 100, 150 neurons of the hidden layer and for each of these cases, plotted for 25, 50, 100, 150 epochs as shown below.





The second method we implemented was the Mini-Batch, which consisted of coding a second loop that is ruled by a variable we call batch, whose size determines how many training elements’ gradients we accumulate before updating the weights. For this case, the size of our batch was 10 which meant, every 10 images we go through, we increment the results of the gradients and only every 10 we update the weights which means that for 1000 training images across 100 epochs we are only updating the weights 10,000 times. We graphed Mini-Batch results for the loss over the number of epochs as shown. This was also for the same cases of Neurons, and on each for all the cases of number of epochs.

Source code and simulations are below.

import os

import sys

import cv2

import numpy as np

from sklearn.utils import shuffle

import matplotlib.pyplot as plt

train = np.empty((1000,28,28),dtype='float64')

trainY = np.zeros((1000,10,1))

test = np.empty((10000,28,28),dtype='float64')

testY = np.zeros((10000,10,1))

# Load in the images

i = 0

for filename in os.listdir('C:/Users/Windows User/Desktop/UB Spring 2019/Deep Learning/Assignment 3/Data/Training1000'):

y = int(filename[0])

trainY[i,y] = 1.0

train[i] = cv2.imread('C:/Users/Windows User/Desktop/UB Spring 2019/Deep Learning/Assignment 3/Data/Training1000/{0}'.format(filename),0)/255.0 #

#for color, use 1

i = i + 1

i = 0 # read test data

for filename in os.listdir('C:/Users/Windows User/Desktop/UB Spring 2019/Deep Learning/Assignment 3/Data/Test10000'):

y = int(filename[0])

testY[i,y] = 1.0

test[i] = cv2.imread('C:/Users\Windows User/Desktop/UB Spring 2019/Deep Learning/Assignment 3/Data/Test10000/{0}'.format(filename),0)/255.0

i = i + 1

trainX = train.reshape(train.shape[0],train.shape[1]\*train.shape[2],1)

testX = test.reshape(test.shape[0],test.shape[1]\*test.shape[2],1)

numNeuronsLayer1 = 150

numNeuronsLayer2 = 10

numEpochs = 150

loss\_store = np.ndarray((numEpochs,1))

x\_axis = np.ndarray((numEpochs,1))

#---------------------NN------------------------

w1 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer1,784))

b1 = np.random.uniform(low=-1,high=1,size=(numNeuronsLayer1,1))

w2 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer2,numNeuronsLayer1))

b2 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer2,1))

gradw1 = 0;

gradw2 = 0;

gradb1 = 0;

gradb2 = 0;

learningRate = 0.1;

batch\_size = 10;

for n in range(0,numEpochs):

loss = 0

trainX,trainY = shuffle(trainX, trainY) # shuffle data for stochastic behavior

ind= trainX.shape[0]//10

for i in range(1,trainX.shape[0]//10): # will go through one image per iteration.

for k in range(1,batch\_size+1):

# Forward Pass

# Equations for computing a1,s1,a2,s2 - outputs of the sum (s) and activation fuction (a).

s1 = np.dot(w1,trainX[i\*k]) + b1

a1 = 1/(1+np.exp(-1\*s1)) # np.exp operates on the array

s2 = np.dot(w2,a1) + b2

a2 = 1/(1+np.exp(-1\*s2))

# Backpropagation

# equations for computing the deltas and the gradients

loss += (0.5 \* np.multiply((a2-trainY[k\*i]),(a2-trainY[k\*i]))).sum()

#loss += (0.5 \* ((a2-trainY[i])\*(a2-trainY[i]))).sum()

delta2 = np.multiply(-np.multiply(trainY[i\*k]-a2,a2),(1-a2))

delta1 = np.multiply(np.multiply(np.dot(np.transpose(w2),delta2),a1), (1-a1))

gradw2 += np.dot(delta2,np.transpose(a1))

gradw1 += np.dot(delta1,np.transpose(trainX[i\*k]))

gradb1 += np.multiply(delta1,1)

gradb2 += np.multiply(delta2,1)

# adjust the weights

w2 = w2 - learningRate \* (gradw2/10)

b2 = b2 - learningRate \* (gradb2/10)

w1 = w1 - learningRate \* (gradw1/10)

b1 = b1 - learningRate \* (gradb1/10)

gradw1 = 0;

gradw2 = 0;

gradb1 = 0;

gradb2 = 0;

loss\_store[n,0] = loss;

x\_axis[n,0] = n;

print("epoch = " + str(n) + " loss = " + (str(loss)))

print("done training , starting testing..")

accuracyCount = 0

#plot the fitted line

area = 10

colors =['black']

plt.scatter(x\_axis, loss\_store, s=area, c=colors, alpha=0.5, linewidths=8)

plt.title('LOSS / NUM EPOCHS')

plt.xlabel('NUM EPOCHS')

plt.ylabel('LOSS')

yfitted = loss\_store

line,= plt.plot(x\_axis, yfitted, '--', linewidth=2) #line plot

line.set\_color('red')

plt.show()

for i in range(testY.shape[0]):

# do forward pass

s1 = np.dot(w1,testX[i]) + b1

a1 = 1/(1+np.exp(-1\*s1)) # np.exp operates on the array

s2 = np.dot(w2,a1) + b2

a2 = 1/(1+np.exp(-1\*s2))

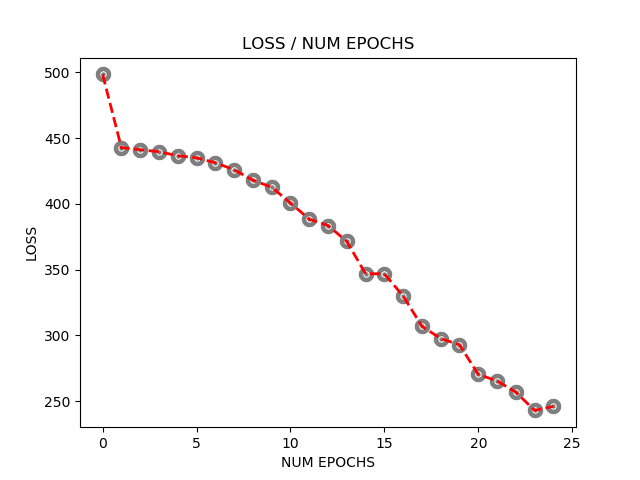
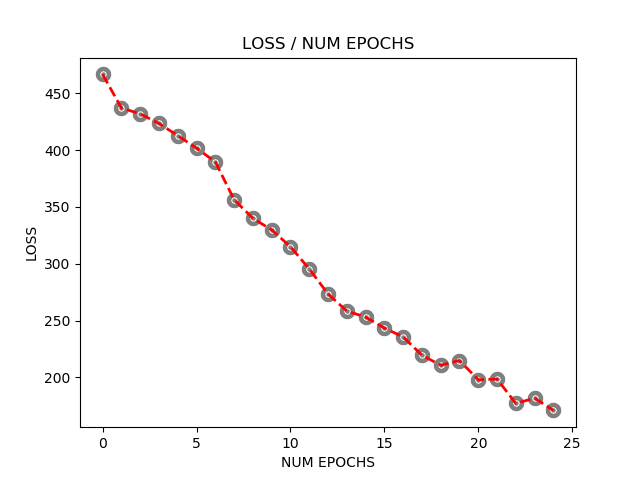
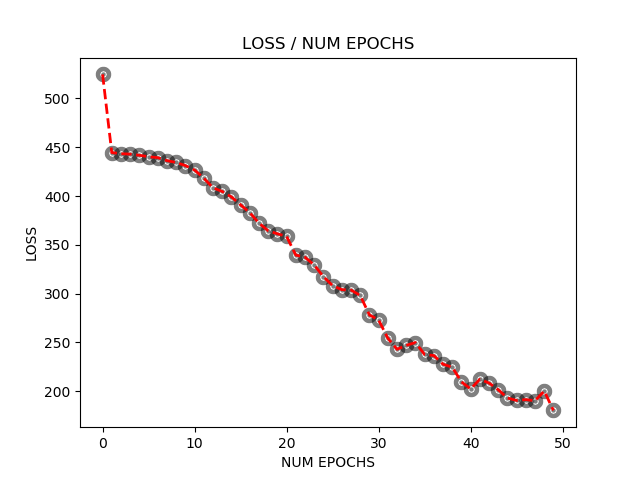
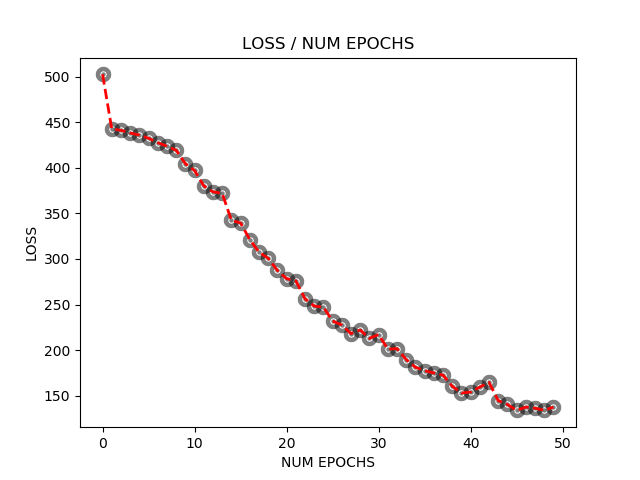
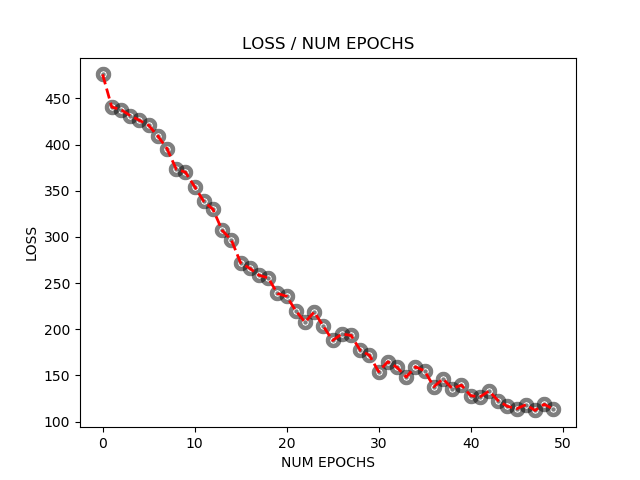
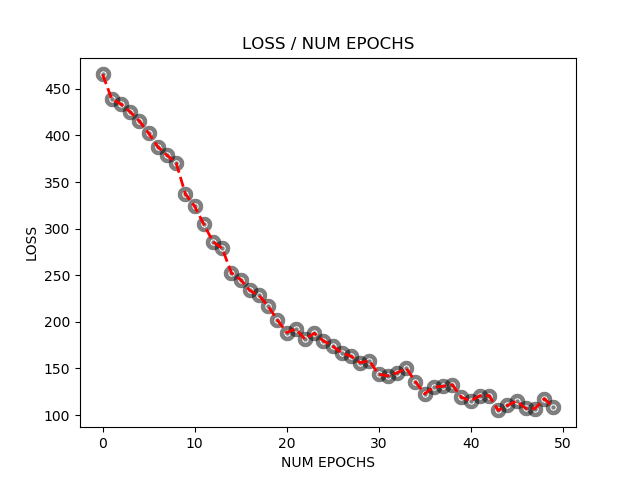
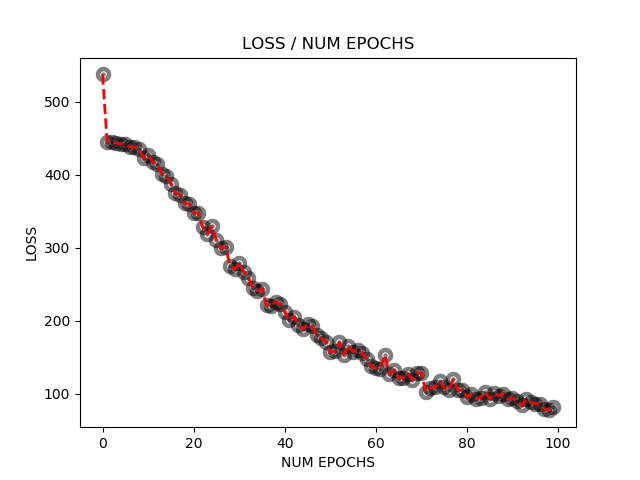
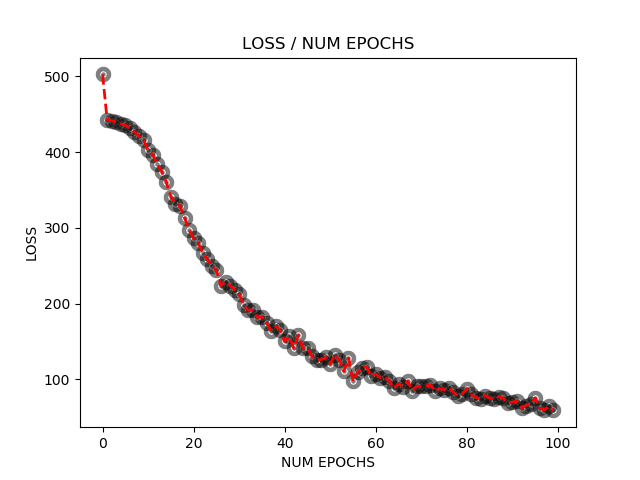
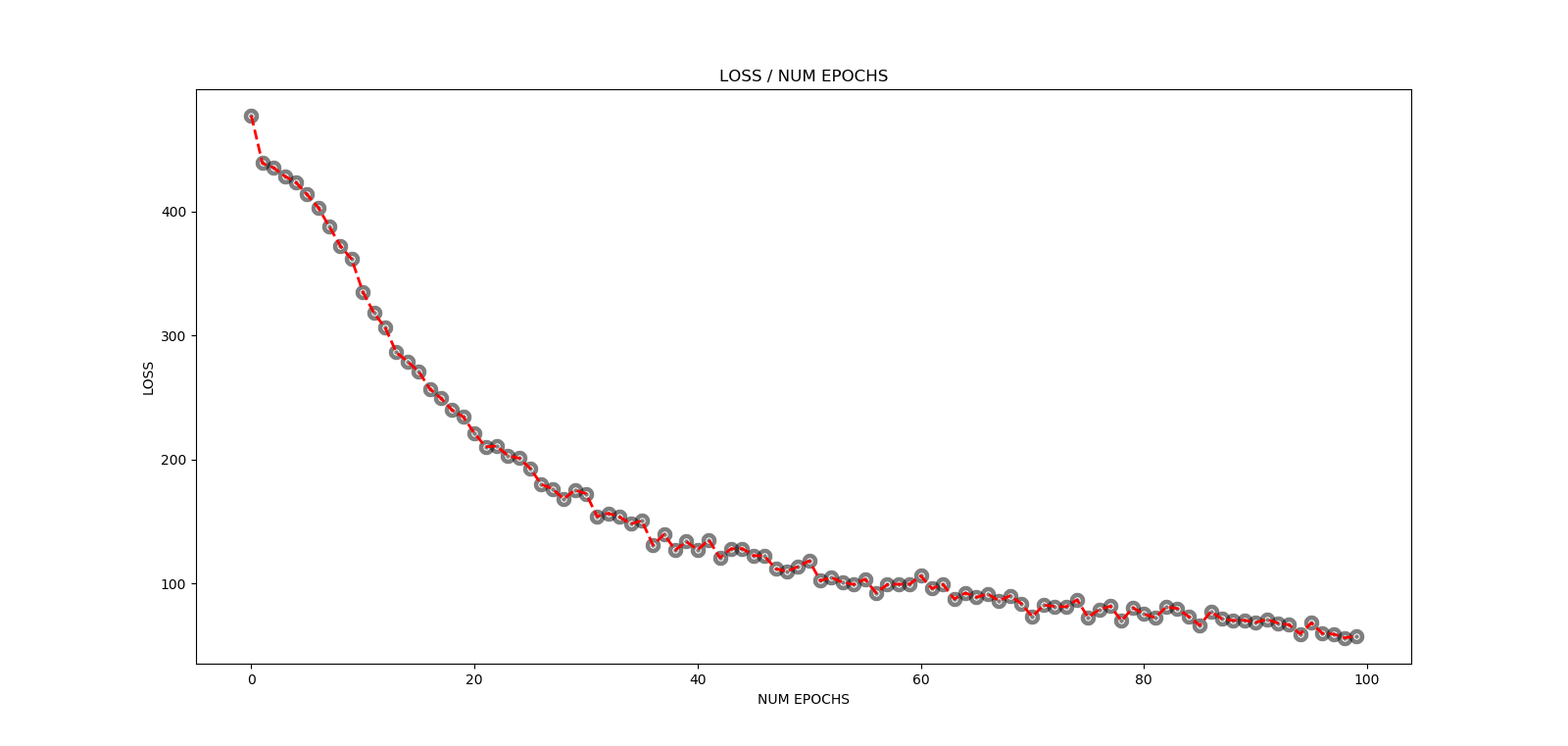
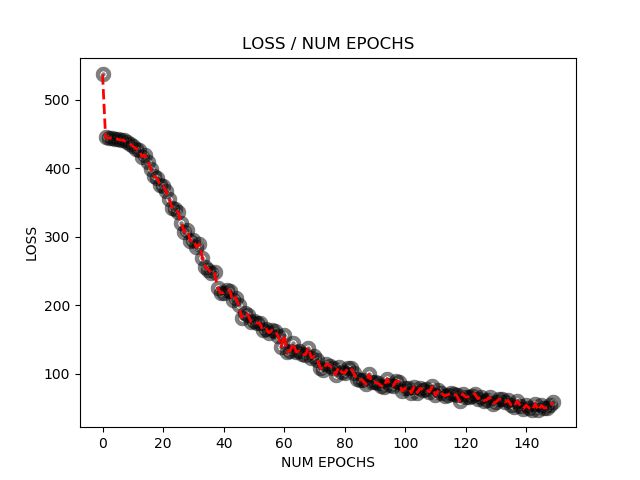
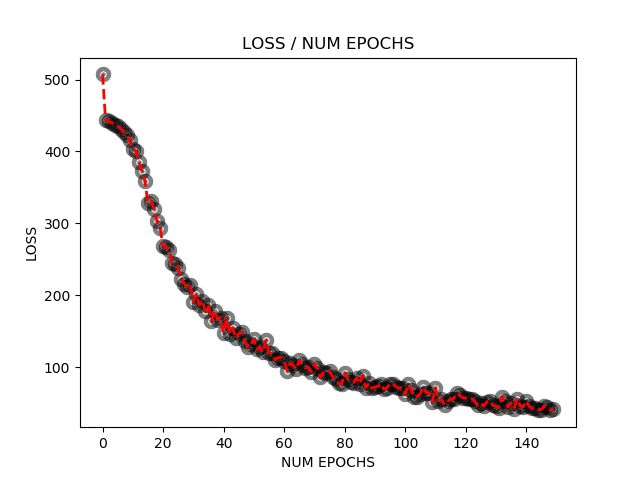
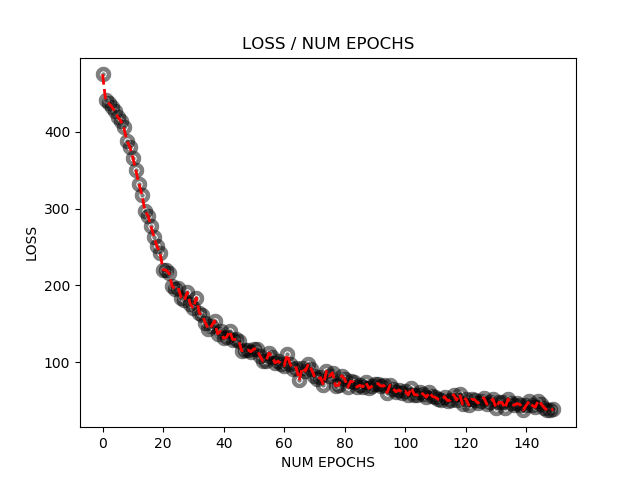
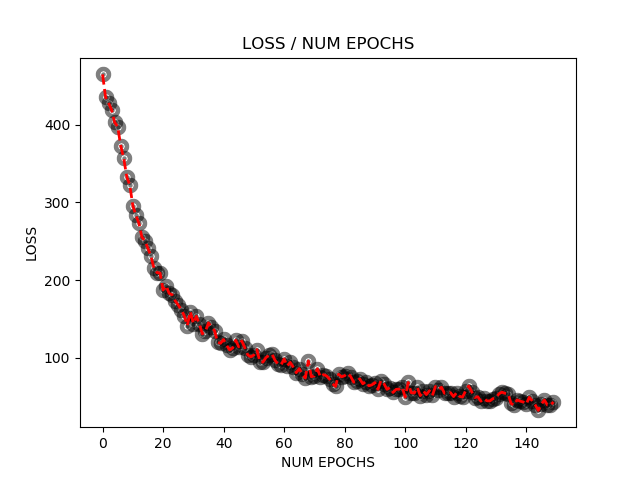
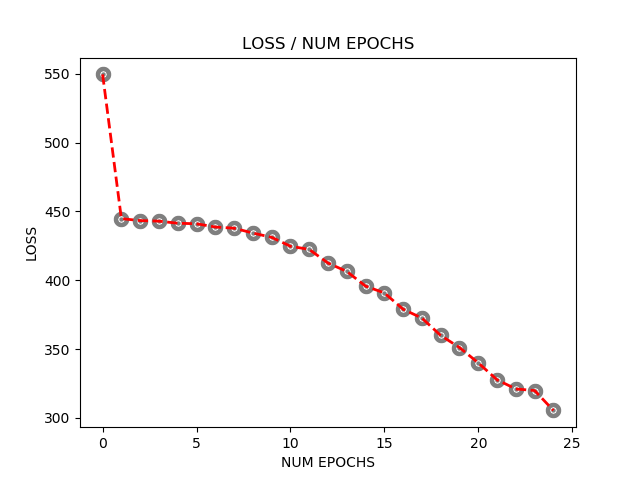
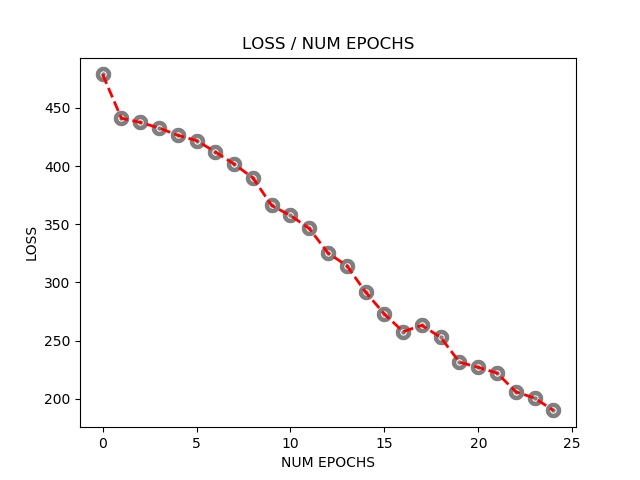
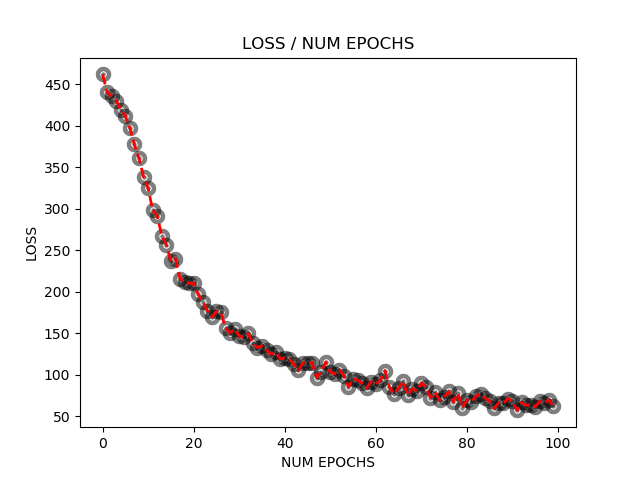
# determine index of maximum output value

a2index = a2.argmax(axis = 0)

if (testY[i,a2index] == 1):

accuracyCount = accuracyCount + 1

print("Accuracy count = " + str(accuracyCount/10000.0))



Finally, we implemented two different activation functions, the RELU and the Tanh. For each of them we derived the math to obtain the derivatives and then we coded into our neural network, which will have an effect both on the forward pass and on the backpropagation.

Below is the code and the simulation of the RELU Function for the case of 50 neurons on 100 epochs.

After trying we also changed the learning rate to a lower value in order to decrease the amount the gradients are allowed to change at a time. The RELU function’s behavior is more aggressive in changing, so it has to be modelled slower.

import os

import sys

import cv2

import numpy as np

from sklearn.utils import shuffle

import matplotlib.pyplot as plt

train = np.empty((1000,28,28),dtype='float64')

trainY = np.zeros((1000,10,1))

test = np.empty((10000,28,28),dtype='float64')

testY = np.zeros((10000,10,1))

# Load in the images

i = 0

for filename in os.listdir('C:/Users/Windows User/Desktop/UB Spring 2019/Deep Learning/Assignment 3/Data/Training1000/'):

y = int(filename[0])

trainY[i,y] = 1.0

train[i] = cv2.imread('C:/Users/Windows User/Desktop/UB Spring 2019/Deep Learning/Assignment 3/Data/Training1000/{0}'.format(filename),0)/255.0 #

#for color, use 1

i = i + 1

i = 0 # read test data

for filename in os.listdir('C:/Users/Windows User/Desktop/UB Spring 2019/Deep Learning/Assignment 3/Data/Test10000'):

y = int(filename[0])

testY[i,y] = 1.0

test[i] = cv2.imread('C:/Users/Windows User/Desktop/UB Spring 2019/Deep Learning/Assignment 3/Data/Test10000/{0}'.format(filename),0)/255.0

i = i + 1

trainX = train.reshape(train.shape[0],train.shape[1]\*train.shape[2],1)

testX = test.reshape(test.shape[0],test.shape[1]\*test.shape[2],1)

numNeuronsLayer1 = 50

numNeuronsLayer2 = 10

numEpochs = 100

loss\_store = np.ndarray((numEpochs,1))

x\_axis = np.ndarray((numEpochs,1))

#---------------------NN------------------------

w1 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer1,784))

b1 = np.random.uniform(low=-1,high=1,size=(numNeuronsLayer1,1))

w2 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer2,numNeuronsLayer1))

b2 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer2,1))

learningRate = 0.005; # RELU Activation Function can increase much faster, so learning rate on the gradients must be smaller.

for n in range(0,numEpochs):

loss = 0

trainX,trainY = shuffle(trainX, trainY) # shuffle data for stochastic behavior

for i in range(trainX.shape[0]): # will go through one image per iteration.

# do forward pass

# your equations for the forward pass

s1 = np.dot(w1,trainX[i]) + b1

a1 = np.maximum(0,s1) # np.maximum hopefully operates on the array

s2 = np.dot(w2,a1) + b2

a2 = np.maximum(0,s2)

# do backprop and compute the gradients \* also works instead

dev\_relu = a2.copy()

dev\_relu[dev\_relu < 0]=0

dev\_relu[dev\_relu > 0]=1

delta2 = -np.multiply(trainY[i]-a2,dev\_relu)

dev\_relu1 = a1.copy()

dev\_relu1[dev\_relu1 < 0]=0

dev\_relu1[dev\_relu1 > 0]=1

delta1 = np.multiply(np.dot(np.transpose(w2),delta2), dev\_relu1)

gradw2 = np.dot(delta2,np.transpose(a1))

gradw1 = np.dot(delta1,np.transpose(trainX[i]))

gradb1 = np.multiply(delta1,1)

gradb2 = np.multiply(delta2,1)

#loss += (0.5 \* ((a2-trainY[i])\*(a2-trainY[i]))).sum()

loss += (0.5 \* np.multiply((a2-trainY[i]),(a2-trainY[i]))).sum()

# your equations for computing the deltas and the gradients

# adjust the weights

w2 = w2 - learningRate \* gradw2

b2 = b2 - learningRate \* gradb2

w1 = w1 - learningRate \* gradw1

b1 = b1 - learningRate \* gradb1

loss\_store[n,0] = loss;

x\_axis[n,0] = n;

print("epoch = " + str(n) + " loss = " + (str(loss)))

print("done training , starting testing..")

accuracyCount = 0

#plot the fitted line

area = 10

colors =['black']

plt.scatter(x\_axis, loss\_store, s=area, c=colors, alpha=0.5, linewidths=8)

plt.title('LOSS / NUM EPOCHS')

plt.xlabel('NUM EPOCHS')

plt.ylabel('LOSS')

yfitted = loss\_store

line,= plt.plot(x\_axis, yfitted, '--', linewidth=2) #line plot

line.set\_color('red')

plt.show()

count = 0

for i in range(testY.shape[0]):

# do forward pass

s1 = np.dot(w1,testX[i]) + b1

a1 = 1/(1+np.exp(-1\*s1)) # np.exp operates on the array

s2 = np.dot(w2,a1) + b2

a2 = 1/(1+np.exp(-1\*s2))

count = count+1

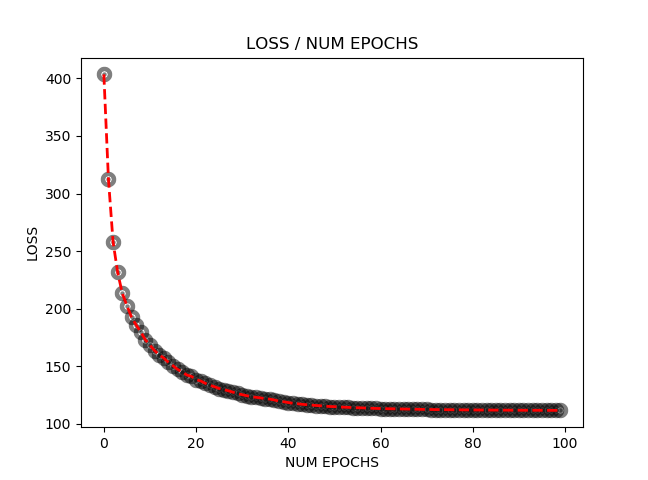
print("Counter = " + str(count)) # determine index of maximum output value

a2index = a2.argmax(axis = 0)

if (testY[i,a2index] == 1):

accuracyCount = accuracyCount + 1

print("Accuracy count = " + str(accuracyCount/10000.0))



For the Tanh activation function we also changed the code for forward pass and backpropagation and then computed and graphed the result for the same case.

The code seemed to be easy to change to apply the Tanh activation function but the output was not good, and it doesn’t seem to be working. In the process of fixing it.

import os

import sys

import cv2

import numpy as np

from sklearn.utils import shuffle

import matplotlib.pyplot as plt

train = np.empty((1000,28,28),dtype='float64')

trainY = np.zeros((1000,10,1))

test = np.empty((10000,28,28),dtype='float64')

testY = np.zeros((10000,10,1))

# Load in the images

i = 0

for filename in os.listdir('C:/Users/Windows User/Desktop/UB Spring 2019/Deep Learning/Assignment 3/Data/Training1000/'):

y = int(filename[0])

trainY[i,y] = 1.0

train[i] = cv2.imread('C:/Users/Windows User/Desktop/UB Spring 2019/Deep Learning/Assignment 3/Data/Training1000/{0}'.format(filename),0)/255.0 #

#for color, use 1

i = i + 1

i = 0 # read test data

for filename in os.listdir('C:/Users/Windows User/Desktop/UB Spring 2019/Deep Learning/Assignment 3/Data/Test10000'):

y = int(filename[0])

testY[i,y] = 1.0

test[i] = cv2.imread('C:/Users/Windows User/Desktop/UB Spring 2019/Deep Learning/Assignment 3/Data/Test10000/{0}'.format(filename),0)/255.0

i = i + 1

trainX = train.reshape(train.shape[0],train.shape[1]\*train.shape[2],1)

testX = test.reshape(test.shape[0],test.shape[1]\*test.shape[2],1)

numNeuronsLayer1 = 50

numNeuronsLayer2 = 10

numEpochs = 100

loss\_store = np.ndarray((numEpochs,1))

x\_axis = np.ndarray((numEpochs,1))

#---------------------NN------------------------

w1 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer1,784))

b1 = np.random.uniform(low=-1,high=1,size=(numNeuronsLayer1,1))

w2 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer2,numNeuronsLayer1))

b2 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer2,1))

learningRate = 0.1;

for n in range(0,numEpochs):

loss = 0

trainX,trainY = shuffle(trainX, trainY) # shuffle data for stochastic behavior

for i in range(trainX.shape[0]): # will go through one image per iteration.

# do forward pass

# your equations for the forward pass

s1 = np.dot(w1,trainX[i]) + b1

a1 = np.tanh(s1) # np.exp operates on the array

s2 = np.dot(w2,a1) + b2

a2 = np.tanh(s2)

# do backprop and compute the gradients \* also works instead

delta2 = -np.multiply(trainY[i]-a2,np.multiply(a2,a2))

delta1 = np.multiply(np.dot(np.transpose(w2),delta2),np.multiply(a1,a1))

gradw2 = np.dot(delta2,np.transpose(a1))

gradw1 = np.dot(delta1,np.transpose(trainX[i]))

gradb1 = np.multiply(delta1,1)

gradb2 = np.multiply(delta2,1)

#loss += (0.5 \* ((a2-trainY[i])\*(a2-trainY[i]))).sum()

loss += (0.5 \* np.multiply((a2-trainY[i]),(a2-trainY[i]))).sum()

# your equations for computing the deltas and the gradients

# adjust the weights

w2 = w2 - learningRate \* gradw2

b2 = b2 - learningRate \* gradb2

w1 = w1 - learningRate \* gradw1

b1 = b1 - learningRate \* gradb1

loss\_store[n,0] = loss;

x\_axis[n,0] = n;

print("epoch = " + str(n) + " loss = " + (str(loss)))

print("done training , starting testing..")

accuracyCount = 0

#plot the fitted line

area = 10

colors =['black']

plt.scatter(x\_axis, loss\_store, s=area, c=colors, alpha=0.5, linewidths=8)

plt.title('LOSS / NUM EPOCHS')

plt.xlabel('NUM EPOCHS')

plt.ylabel('LOSS')

yfitted = loss\_store

line,= plt.plot(x\_axis, yfitted, '--', linewidth=2) #line plot

line.set\_color('red')

plt.show()

count = 0

for i in range(testY.shape[0]):

# do forward pass

s1 = np.dot(w1,testX[i]) + b1

a1 = 1/(1+np.exp(-1\*s1)) # np.exp operates on the array

s2 = np.dot(w2,a1) + b2

a2 = 1/(1+np.exp(-1\*s2))

count = count+1

print("Counter = " + str(count)) # determine index of maximum output value

a2index = a2.argmax(axis = 0)

if (testY[i,a2index] == 1):

accuracyCount = accuracyCount + 1

print("Accuracy count = " + str(accuracyCount/10000.0))

