**CS559 - Neural Networks**

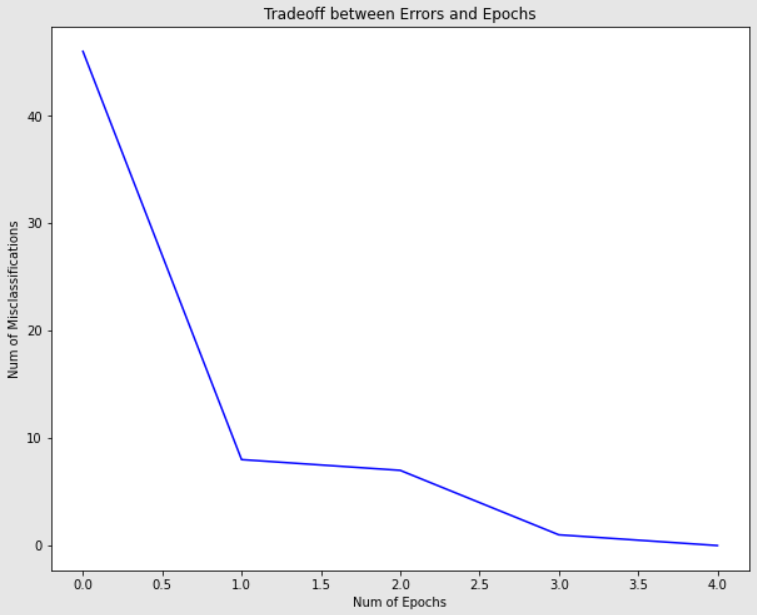
HW3 - Final Report

**(f) The graph for this step is given below.**

We observe that the training phase of the network terminates with 0 errors. The number of misclassifications decreases exponentially as the epochs progress. We have 0% error because our network has been trained on the training examples itself and hence it overfits those training examples and gives 100% accuracy on only those examples. The weights that the network learns cannot make an acceptable accuracy in it’s predictions for unseen examples and therefore the accuracy for test data set is lower than the training accuracy at higher epochs. Furthermore, we observe that the number of misclassifications in the training set comes down to 0% error after about 3 epochs.

In the first iteration, the network was trained only on 50 training samples and this learning was evaluated on 10,000 test samples. Hence, the network was naive towards those 50 training samples and also had the opportunity to fine tune its weights and learn more but couldn't. Therefore, the test error rate is high and not 0%.

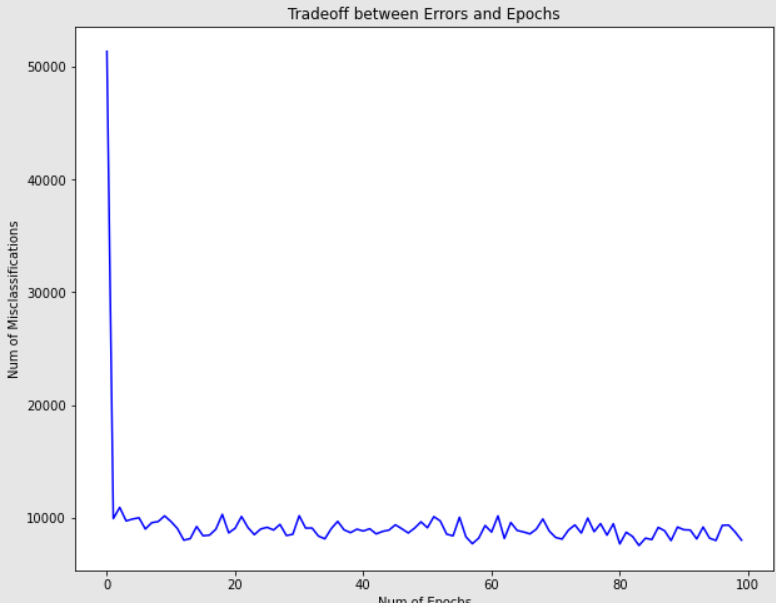
However at 0th epoch, the percentage of errors obtained in test data is 43.86% which is comparatively much lower than error obtained in the training data because we have updated the weights and the network now better understands how to process the inputs after being trained for 5 epochs.



**(g) We observe that the number of misclassifications in the training set comes down to 0% after about 31 epochs. The number of epochs is high because of a higher number of training examples in the iteration.**

This time as well, the network got only 1000 input training samples and after that it was evaluated on 10,000 test samples. Hence, the network was biased towards those 1000 training samples and also had the opportunity to fine tune its weights and learn more but couldn't. Therefore, the test error rate is high and not 0%.

The percentage of errors in this test data validation iteration is 17.39% (lower than the previous step)but not 0% eventually because we have updated the weights accordingly and the network better understands how to process the inputs after being trained for 31 epochs.

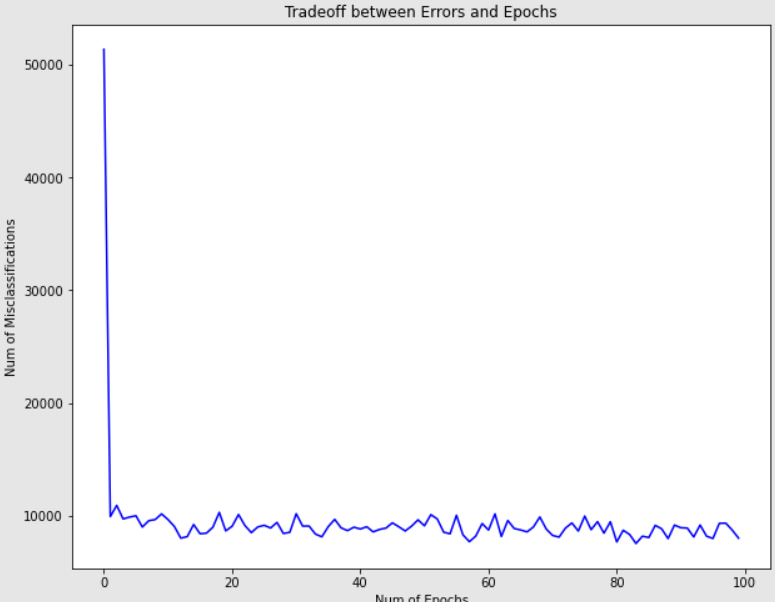


The network has now been trained on 1000 input training samples (far more than 50 samples as in (f)) and also the PTA algorithm has run for comparatively more number of epochs. Now, the network has better estimation of weights and has also approached convergence far earlier as compared to (f). It is also more likely that the network has seen all the 0 - 9 digit labeled samples which was much less likely in the case of (f). The error rate in the test set validation decreases as the number of training examples in the training set increases as the network has learned more efficiently on a larger set of training samples. Therefore, there is a lower test set error rate of 16.73% in the case of n = 1000 as compared to an error of 43.86% in the case where n = 50.

Even though the error rate seems to approach 0, it can never be 0 because we want a more generalized model which is not overfitted on small subset training samples and which is capable of handling unseen test samples and does not do a bad job in classifying the test samples.

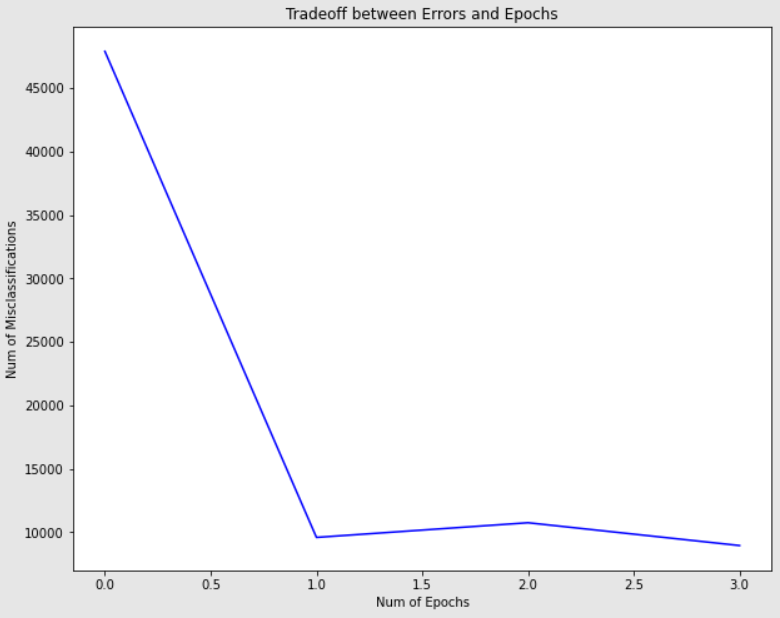
**(h) The number of epochs to be done has been limited to 100. After the first few epochs, we observe that the error rate does not fall anymore and we just keep oscillating between 7,000 to 12,000 for the training data.**

For the testing data, the percentage of misclassified data is 14.77% which is more or less similar to n = 1000. This means that the network has learned a sufficient amount of knowledge from just 1000 samples and performs only a little bit better when trained over all 60,000 training samples.



**(i) On the basis of the above test results,a threshold(epsilon) of 0.15 has been chosen because it converges faster and the results are still comparable to a lower threshold value.**

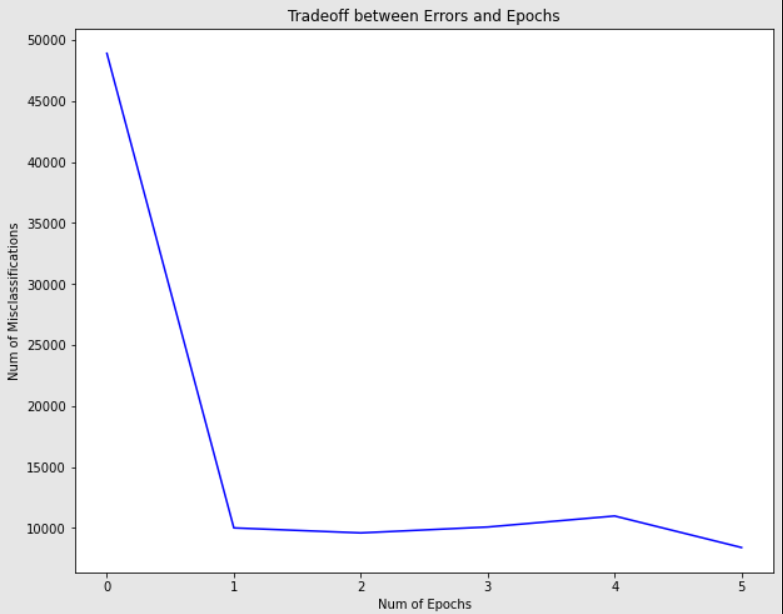
#### **First Time**



Number of errors in test data: 1677

Percentage of test errors: 16.77

#### **Second Time**

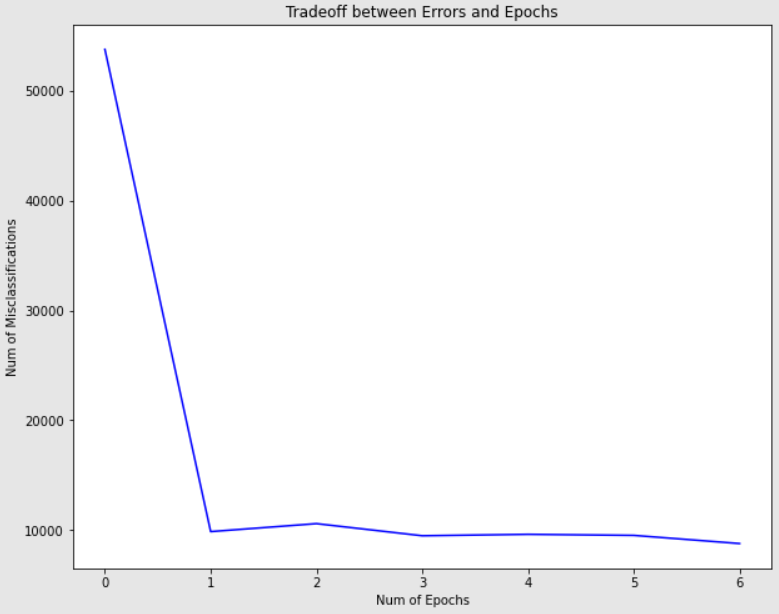


#### Number of errors in test data: 1396

#### Percentage of test errors: 13.96

#### 

#### **Third Time**



Number of errors in test data: 1685

Percentage of test errors: 16.85

**We see that the tradeoff between Errors and Epochs is almost similar for the 3 iterations. The percentage of error varies in the range of 13-17% which is comparable to the results where only 1000 training samples were used.**

Code:

import os

import sys

import path

import gzip

import struct

import numpy as np

import matplotlib.pyplot as plt

#Reading Minst source data as NumPy arrays

def read\_idx(filename):

with open(filename, 'rb') as f:

\_, \_, dimensions = struct.unpack('>HBB', file.read(4))

shape = tuple(struct.unpack('>I', file.read(4))[0] for dim in range(dimensions))

return np.frombuffer(file.read(), dtype=np.uint8).reshape(shape)

#Reading Minst source data as NumPy arrays

def read\_idx\_gz(filename):

with gzip.open(filename, 'rb') as file:

\_, \_, dimensions = struct.unpack('>HBB', file.read(4))

shape = tuple(struct.unpack('>I', file.read(4))[0] for dim in range(dimensions))

return np.frombuffer(file.read(), dtype=np.uint8).reshape(shape)

#The training and test data files need to be in the same directory as the python file for the code to execute correctly

absolutepath = os.path.abspath('train-images-idx3-ubyte.gz')

fileDirectory = os.path.dirname(absolutepath)

train\_data = os.path.join(fileDirectory, os.path.abspath('train-images-idx3-ubyte.gz'))

train\_label= os.path.join(fileDirectory, os.path.abspath('train-labels-idx1-ubyte.gz'))

test\_data = os.path.join(fileDirectory, os.path.abspath('t10k-images-idx3-ubyte.gz'))

test\_label = os.path.join(fileDirectory, os.path.abspath('t10k-labels-idx1-ubyte.gz'))

Training\_Data = read\_idx\_gz(train\_data)

Training\_Labels = read\_idx\_gz(train\_label)

Testing\_Data = read\_idx\_gz(test\_data)

Testing\_Labels = read\_idx\_gz(test\_label)

#Function to calculate step activation

def stepActivation(x):

i = 0

output = np.empty([10,1])

for inp in x:

if inp >= 0:

output[i] = 1.0

else:

output[i] = 0.0

i += 1

return output

#Function to compare the Perceptron outputs with the actual output and count the misses in correct predictions.

def training\_errors(Training\_Data,epoch, errors,n):

for i in range(n):

xi = Training\_Data[i]

xi.resize(784, 1)

v = np.matmul(w,xi)

predicted\_output = v.argmax(axis=0)

actual\_output = Training\_Labels[i]

if predicted\_output != actual\_output:

errors[epoch] += 1

return errors[epoch]

#Function to update the weights of the network based on the diff and learning rate

def updating\_Weights(w):

for i in range(n):

xi = Training\_Data[i]

xi.resize(784, 1)

output = stepActivation(np.matmul(w,xi))

y = np.array(output)

labels = np.zeros((1,10)).T

labels[Training\_Labels[i]] = 1

diff = labels - y

xi\_transpose = np.transpose(xi)

Delta\_W = learning\_rate \* np.matmul(diff, xi\_transpose)

w += Delta\_W

#Function to learn the weights but based on a threshold of 100 on epochs

def learning\_weights(Training\_Data,w, epochs, threshold, learning\_rate,n):

while epochs<100:

errors.append(0)

errors[epochs] = training\_errors(Training\_Data,epochs, errors,n)

updating\_Weights(w)

epochs += 1

if errors[epochs-1]/n <= threshold:

break

#Function to plot missclassification vs epoch tradeoff for each iteration

def plot\_Epoch\_Missclassification\_Graph(errors):

fig, ax = plt.subplots(figsize=(10,8))

plt.plot(range(len(errors)), errors, c = 'blue')

plt.ylabel('Num of Misclassifications')

plt.xlabel('Num of Epochs')

plt.title('Tradeoff between Errors and Epochs')

plt.show()

def calculate\_Errors():

test\_errors = 0

for i in range(len(Testing\_Data)):

xi = Testing\_Data[i]

xi.resize(784, 1)

v = np.matmul(w ,xi)

predicted\_output = v.argmax(axis=0)

actual\_output = Testing\_Labels[i]

if predicted\_output != actual\_output:

test\_errors += 1

print("Number of errors in test data: ", test\_errors)

print("Percentage of test errors: ", test\_errors\*100/len(Testing\_Data))

def main():

global w,n,epoch,threshold,learning\_rate,errors

#f

w = np.random.uniform(-1, 1, size=(10,784))

n = 50

epoch = 0

threshold = 0.0

learning\_rate = 1.0

errors = []

learning\_weights(Training\_Data,w, epoch, threshold, learning\_rate,n)

plot\_Epoch\_Missclassification\_Graph(errors)

calculate\_Errors()

#2

w = np.random.uniform(-1, 1, size=(10,784))

n = 1000

epoch = 0

threshold = 0.0

learning\_rate = 1.0

errors = []

learning\_weights(Training\_Data,w, epoch, threshold, learning\_rate,n)

plot\_Epoch\_Missclassification\_Graph(errors)

calculate\_Errors()

#3

w = np.random.uniform(-1, 1, size=(10,784))

n = 60000

epoch = 0

threshold = 0.0

learning\_rate = 1.0

errors = []

learning\_weights(Training\_Data,w, epoch, threshold, learning\_rate,n)

plot\_Epoch\_Missclassification\_Graph(errors)

calculate\_Errors()

#4

w = np.random.uniform(-1, 1, size=(10,784))

n = 60000

epoch = 0

threshold = 0.15

learning\_rate = 1.0

errors = []

learning\_weights(Training\_Data,w, epoch, threshold, learning\_rate,n)

plot\_Epoch\_Missclassification\_Graph(errors)

calculate\_Errors()

#5

w = np.random.uniform(-1, 1, size=(10,784))

n = 60000

epoch = 0

threshold = 0.15

learning\_rate = 1.0

errors = []

learning\_weights(Training\_Data,w, epoch, threshold, learning\_rate,n)

plot\_Epoch\_Missclassification\_Graph(errors)

calculate\_Errors()

#6

w = np.random.uniform(-1, 1, size=(10,784))

n = 60000

epoch = 0

threshold = 0.15

learning\_rate = 1.0

errors = []

learning\_weights(Training\_Data,w, epoch, threshold, learning\_rate,n)

plot\_Epoch\_Missclassification\_Graph(errors)

calculate\_Errors()

for i in range(1):

main()