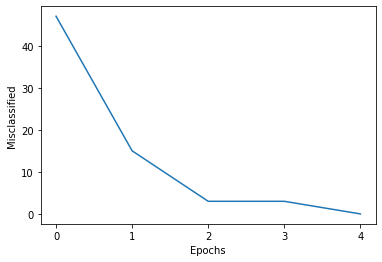
Atanas Delevski

ECE 559 Homework #2 Report

10/05/2020

**Part F)** n = 50, η = 1, ε = 0.001

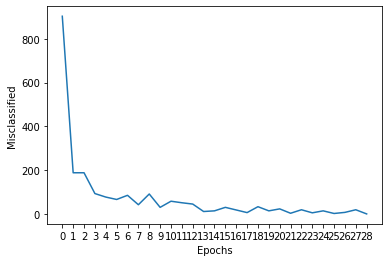


The percentage of misclassified test samples = (4494)/(10000) = 44.94%

So, obviously the number of misclassified samples is different between the training and testing. The training continues until there are 0 errors left. The huge number of misclassifications after testing is due to the fact that we only trained with 50 samples, and so our Perceptron is not fully trained.

(Next parts on next pages)

**Part G)** n = 1000, η = 1, ε = 0.001



The percentage of misclassified test samples = (1783)/(10000) = 17.83%

Since we increased the number of samples from 50 to 1000, the Perceptron was able to train for much longer and on more samples, and so it was better-able to converge. In other words, since it had more samples to train on, it did a better job of “learning”. There is still a discrepancy, since the training errors converge to 0, but the testing results in 1783 errors. However, this result is way smaller compared to the 4494 errors we had when n = 50.

**Part H)** n = 60000, η = 1, ε = 0.03

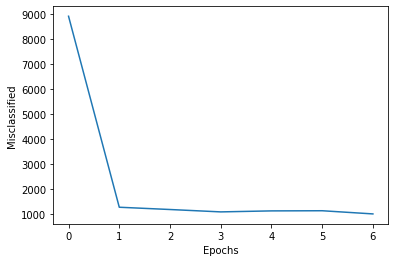
My algorithm had a very hard time converging when I set n = 60000. If I set my epsilon to anything lower than 0.15, then it would just keep training and training until infinity because the errors never get that low. When there are so many samples, the algorithm is probably confusing itself and is trying to learn too much and so it is overfitting and therefore cannot generalize well enough to perform with very little errors on the test samples.

**Part I)**

My algorithm was not able to converge when n = 60000, so I did this part with n = 10000

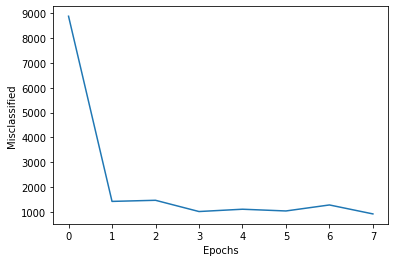
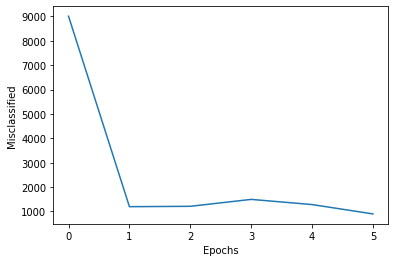
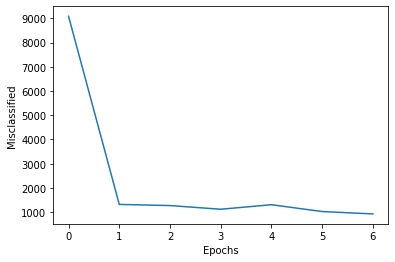
I picked my ε = 0.1 (Anything lower and it also would not converge)

**0-th Time)** (With original weights)



0-th) The percentage of misclassified test samples = (1355)/(10000) = 13.55%

**1st Time)**  **2nd Time)**  **3rd Time)**



1st) The percentage of misclassified test samples = (1307)/(10000) = 13.07%

2nd) The percentage of misclassified test samples = (1216)/(10000) = 12.16%

3rd) The percentage of misclassified test samples = (1263)/(10000) = 12.63%

As you can see, changing the weights can have an effect on the percentage of test errors, but it is nothing significant. This means that our algorithm is very consistent, and different starting points do not account for huge differences in performance.

Code:

from mnist import MNIST

from matplotlib import pyplot as plt

import numpy as np

np.random.seed(1)

mnist = MNIST('dataset/MNIST')

train\_set, train\_labels = mnist.load\_training() #60000 samples

test\_set, test\_labels = mnist.load\_testing() #10000 samples

eta = 1

epsilon = 0

n = 1000

W = 2 \* np.random.rand(10,784) - 1

class Multi\_Class\_Perceptron(object):

def \_\_init\_\_(self, num\_of\_inputs, num\_of\_classes, weights):

self.num\_of\_inputs = num\_of\_inputs

self.num\_of\_classes = num\_of\_classes

self.weights = weights

self.epochs = 0

self.misclassified\_list = []

def find\_errors(self, train\_set, train\_labels, n):

errors = 0

for i in range(n):

v = np.dot(self.weights, train\_set[i])

max, = np.where(v == v.max())

if max[0] != train\_labels[i]:

errors += 1

return errors

def update\_weights(self, train\_set, train\_labels, eta, epsilon, n):

for i in range(n):

d = np.zeros((10, 1))

d[train\_labels[i]] = 1

temp = W @ np.asarray(train\_set[i])

result = [a\_i - b\_i for a\_i, b\_i in zip(d, np.heaviside(temp, 1))]

semi = result @ np.asarray(train\_set[i]).reshape((1, 784))

final = eta \* semi

self.weights += final

def train(self, train\_set, train\_labels, eta, epsilon, n):

errors = self.find\_errors(train\_set, train\_labels, n)

self.misclassified\_list.append(errors)

self.update\_weights(train\_set, train\_labels, eta, epsilon, n)

self.epochs += 1

errors = self.find\_errors(train\_set, train\_labels, n)

self.misclassified\_list.append(errors)

while (errors/n > epsilon):

self.update\_weights(train\_set, train\_labels, eta, epsilon, n)

self.epochs += 1

errors = self.find\_errors(train\_set, train\_labels, n)

self.misclassified\_list.append(errors)

print(f'The final errors are: {errors}, which took {self.epochs} epochs for which the errors were {self.misclassified\_list} (Starting at Epoch 0)')

def test(self, test\_set, test\_labels):

errors = 0

for i in range(9999):

v\_prime = np.dot(self.weights, test\_set[i])

prime\_max, = np.where(v\_prime == v\_prime.max())

if prime\_max[0] != test\_labels[i]:

errors += 1

return errors

perceptron = Multi\_Class\_Perceptron(784, 10, W)

perceptron.find\_errors(train\_set, train\_labels, n)

perceptron.train(train\_set, train\_labels, eta, epsilon, n)

epochs = np.arange(0, perceptron.epochs+1)

plt.plot(epochs, perceptron.misclassified\_list)

plt.xlabel('Epochs')

plt.xticks(np.arange(0, perceptron.epochs+1, step=1))

plt.ylabel('Misclassified')

plt.show()

test\_errors = perceptron.test(test\_set, test\_labels)

print(f'The percentage of misclassified test samples = ({test\_errors})/(10000) = {round((test\_errors/10000)\*100, 2)}%')