**CS559 - Neural Networks**

HW2 - Final Report

1. **Perceptron Training Algorithm - Computer program**

**h(vii): The initial, optimal and final weights have been recorded as below for the Training Algorithm execution.**

Initial Weights:

W0 : -0.22892157456646872

W1 : 0.4493305767595295

W2 : 0.36204553208662693

Optimal Weights:

W0 : 0.22882066259795253

W1 : 0.6455712374212401

W2 : 0.4184463886303911

Final Weights via PTA:

W0’ : 1.2817270849172564

W1’ : 3.620266560419754

W2’ : 2.3005280236499397

W0(Optimal-final): 1.0529064223193039

W1(Optimal-final): 2.974695322998514

W2(Optimal-final): 1.8820816350195486

From the above weights, we see that the Training Algorithm has not calculated the weights optimally as was expected. However, When we draw the decision boundary using these values, the resultant line is able to classify the points into S0 and S1 sets correctly. This means that the decision boundary obtained by the PTA is close to the optimal value.

The perceptron converges to the optimal values depending on our choice of the Hyperparameters i.e initial weights, the amount of input data as well as the learning rate eta. Tuning these hyperparameters will lead the algorithm to converge towards the optimal boundary.

**Tradeoff between η(eta) and number of Epochs:**

*Below results have been taken after running the PTA 1000 times.*

N = 100 and eta = 1

Epochs needed until convergence: 5

N = 100 and eta = 10

Epochs needed until convergence: 20

N = 100 and eta = 0.1

Epochs needed until convergence: 35

N = 1000 and eta = 1

Epochs needed until convergence: 198

N = 1000 and eta = 10

Epochs needed until convergence: 248

N = 1000 and eta = 0.1

Epochs needed until convergence: 265

**h(l)** : The value of the learning rate (eta) determines how slowly or rapidly the training algorithm will converge. Smaller learning rates will require more training epochs as in each epoch a smaller changes are made to the weights. Similarly, larger learning rates correspond to comparatively larger changes in the weights with each epoch and could lead to a faster convergence.

**h(m)**: The results of training an algorithm is also dependent on the initial weights in addition to the number of input examples and the learning rate. Say if we start with the initial weights that are exactly equal to the optimal weights required for classification. In this case, only 1 epoch will be needed for the algorithm to converge. On the other hand, if the initial weights are extremely far from the optimal weights, it will need many epochs to converge. So, the initial weights used before we begin the training does play a role in our final results and training time. However, as long as we dont change the other hyperparameters like eta and Number of inputs, the difference is not significant.

**h(n):** As we increase the number of inputs, the time taken to train the algorithm and reach a convergence increases proportionally. However, a larger input set improves the algorithm’s learning accuracy and helps in eliminating problems like high variance and underfitting of the network.

Code:

import random

import numpy as np

from string import Template

import matplotlib.pyplot as plt

#Step Activation Function

*def* step\_activation(*x*):

return 1 if *x*>=0 else 0

#Function for initializing the Weights and input vectors uniformly and randomly

*def* initialize(*n*):

w0 = random.uniform(-1/4,1/4)

w1 = random.uniform(-1,1)

w2 = random.uniform(-1,1)

Weights = [w0,w1,w2]

S = np.random.uniform(-1.0,1.0,(*n*,2))

print('Initial Weights: ' + str(Weights))

return Weights,S

#Function to plot the points and decision boundary

*def* initial\_graph(*Weights*, *S*):

plt.clf()

plt.rcParams.update({'figure.figsize':(10,8), 'figure.dpi':100})

S0 = []

S1 = []

#Classifying the points based on the line equation w0 + w1x1 + w2x2 = 0

for pt in *S*:

x1, x2 = pt

if (*Weights*[1]\*x1 + *Weights*[2]\*x2 + *Weights*[0]) >=0:

S0.append(pt)

else:

S1.append(pt)

x = list(map(*lambda* *pt* : *pt*[0], S0))

y = list(map(*lambda* *pt* : *pt*[1], S0))

plt.scatter(x, y, *alpha*=0.5, *label*="$S\_0$", *marker*="v", *edgecolors*='none', *color* = 'red')

x = list(map(*lambda* *pt* : *pt*[0], S1))

y = list(map(*lambda* *pt* : *pt*[1], S1))

plt.scatter(x, y, *alpha*=0.5, *label*="$S\_1$",*marker*= '\*', *edgecolors*='none' , *color* = 'blue')

x = np.linspace(-1, 1, 100)

y = (-*Weights*[0] - *Weights*[1]\*x) / *Weights*[2]

plt.plot(x, y, *label*="Decision Boundary")

plt.xlabel('x1')

plt.ylabel('x2')

plt.suptitle('Graphical Representation of Perceptron Classification (N = ' + str(N) + ')')

plt.legend()

plt.show()

*def* Perceptron\_Training():

epoch = 0

missclassifications\_for\_epoch = []

global w0\_, w1\_, w2\_, w0, w1, w2

#Calculating the desired and actual outputs of the input vectors and comparing them.

while True:

misclassifications = 0

for point in X:

x1, x2 = point

desired\_output = step\_activation(w0 + w1\*x1 + w2\*x2)

actual\_output = step\_activation(w0\_ + w1\_\*x1 + w2\_\*x2)

#If there is a mismatch in the outputs, then the weights are updated.

if actual\_output != desired\_output:

w0\_ = w0\_ + eta \* 1 \* (desired\_output - actual\_output)

w1\_ = w1\_ + eta \* x1 \* (desired\_output - actual\_output)

w2\_ = w2\_ + eta \* x2 \* (desired\_output - actual\_output)

misclassifications = misclassifications + 1

#If all the points have been classified correctly, then PTA is stopped.

if misclassifications == 0:

missclassifications\_for\_epoch.append(0)

break

missclassifications\_for\_epoch.append(misclassifications)

epoch = epoch + 1

print('Epochs taken: ' + str(epoch))

return missclassifications\_for\_epoch

#Function to plot the graph of number of misclassifications against the epoch

*def* plot\_missclassifications(*missclassifications\_for\_epoch*):

plt.clf()

number\_of\_epochs = len(*missclassifications\_for\_epoch*)

x = list(range(number\_of\_epochs))

y = *missclassifications\_for\_epoch*

plt.plot(x, y, *label*="Misclassifications per epoch")

plt.suptitle('Misclassifications per epoch (\u03B7 = ' + str(eta) + ' and N = ' + str(N) + ')')

plt.xlabel('Epoch Number')

plt.ylabel('Number of Misclassifications')

plt.legend()

plt.show()

#Function to print the optimal and final weights

*def* Weight\_Comparison():

print('Optimal Weights: ' + str([w0, w1, w2])+'\n')

print('Final Weights: ' + str([w0\_, w1\_, w2\_]) + '\n')

print('W0(Optimal-final): ' + str(abs(abs(w0) - abs(w0\_))))

print('W1(Optimal-final): ' + str(abs(abs(w1) - abs(w1\_))))

print('W2(Optimal-final): ' + str(abs(abs(w2) - abs(w2\_))))

*def* execute():

global N, w0, w1, w2, w0\_, w1\_, w2\_, X, eta

rows = []

N = 0

eta = 0

#Picking the initial weights w0', w1', w2'

w0\_ = random.uniform(-1, 1)

w1\_ = random.uniform(-1, 1)

w2\_ = random.uniform(-1, 1)

initial\_weights = [w0\_, w1\_, w2\_].copy()

row = []

# setting N = 100 and training the perceptron for eta values 1, 10, 0.1

N = 100

Weights, X = initialize(N)

initial\_graph(Weights, X)

eta = 1

missclassifications\_for\_epoch = Perceptron\_Training()

plot\_missclassifications(missclassifications\_for\_epoch)

row.append(len(missclassifications\_for\_epoch))

Weight\_Comparison()

w0\_, w1\_, w2\_ = initial\_weights

eta = 10

missclassifications\_for\_epoch = Perceptron\_Training()

plot\_missclassifications(missclassifications\_for\_epoch)

row.append(len(missclassifications\_for\_epoch))

w0\_, w1\_, w2\_ = initial\_weights

eta = 0.1

missclassifications\_for\_epoch = Perceptron\_Training()

plot\_missclassifications(missclassifications\_for\_epoch)

row.append(len(missclassifications\_for\_epoch))

# setting N = 1000 and training the perceptron for eta values 1, 10, 0.1

N = 1000

Weights, X = initialize(N)

initial\_graph(Weights, X)

eta = 1

missclassifications\_for\_epoch = Perceptron\_Training()

plot\_missclassifications(missclassifications\_for\_epoch)

row.append(len(missclassifications\_for\_epoch))

w0\_, w1\_, w2\_ = initial\_weights

eta = 10

missclassifications\_for\_epoch = Perceptron\_Training()

plot\_missclassifications(missclassifications\_for\_epoch)

row.append(len(missclassifications\_for\_epoch))

w0\_, w1\_, w2\_ = initial\_weights

eta = 0.1

missclassifications\_for\_epoch = Perceptron\_Training()

plot\_missclassifications(missclassifications\_for\_epoch)

row.append(len(missclassifications\_for\_epoch))

rows.append(row)

for x in range(1):

execute()

Plots:















