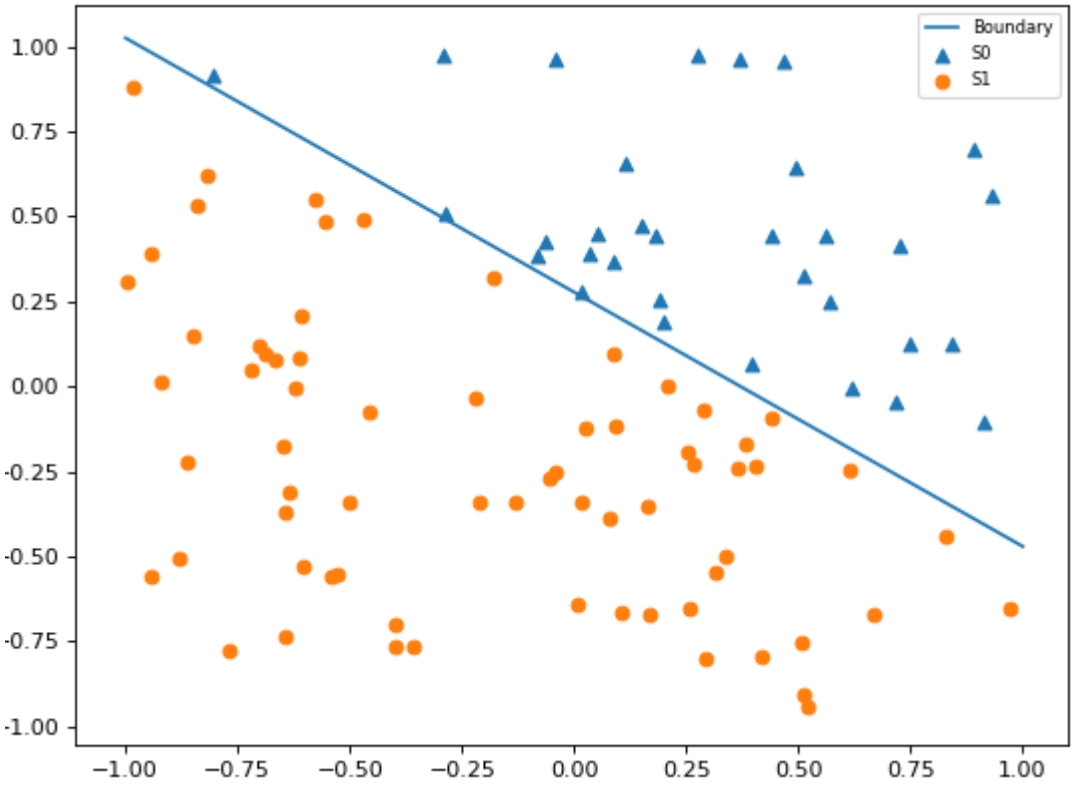
Q3) Following are the answers for the report

e) initial weights for w0 w1 w2 :

Initial weights [-0.25 0.67518076 0.90285714]

i)



j) (ii) Initial weights (w0 , w1 , w2) :

[-0.88227746 0.18202592 -0.53098944]

(iii) No of misclassifications:

Missclassification = 17

(iv) new set of weights

[0.11772254 -1.57955371 -2.90913254]

(v) Missclassification = 11

(vi) Epoch's until convergence

epoch 1

[0.11772254 -1.57955371 -2.90913254]

Missclassification = 11

epoch 2

[1.11772254 -1.90840051 -3.4217573]

Missclassification = 8

epoch 3

[1.11772254 -2.86833541 -3.44031099]

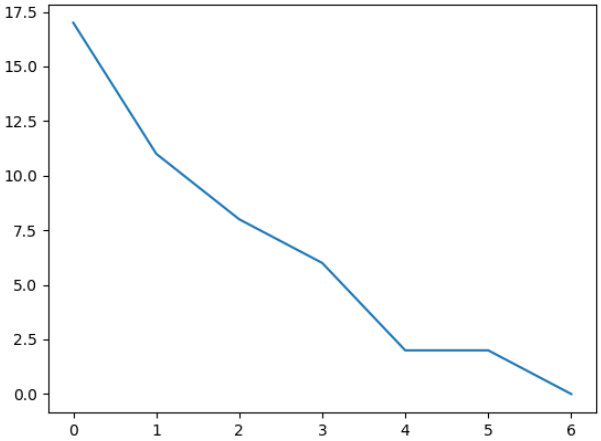
Missclassification = 6

epoch 4
[1.11772254 -2.54973466 -4.16942067]
Missclassification = 2
epoch 5
[1.11772254 -3.24533341 -3.79518748]
Missclassification = 2
epoch 6
[1.11772254 -2.91033605 -4.21959937]
Missclassification = 0
epoch 7
[1.11772254 -2.91033605 -4.21959937]
Missclassification = 19
epoch 1
[9.11772254 -17.5122963 -29.91636394]
Missclassification = 6
epoch 2
[9.11772254 -18.0938839 -35.07347968]
Missclassification = 2
epoch 3
[9.11772254 -25.04987138 -31.33114777]
Missclassification = 8
epoch 4
[9.11772254 -25.58546723 -38.65041578]
Missclassification = 0
epoch 5
[9.11772254 -25.58546723 -38.65041578]
Missclassification = 14
epoch 1
[0.11772254 -0.29104965 -0.40756788]
Missclassification = 0
epoch 2
[0.11772254 -0.29104965 -0.40756788]

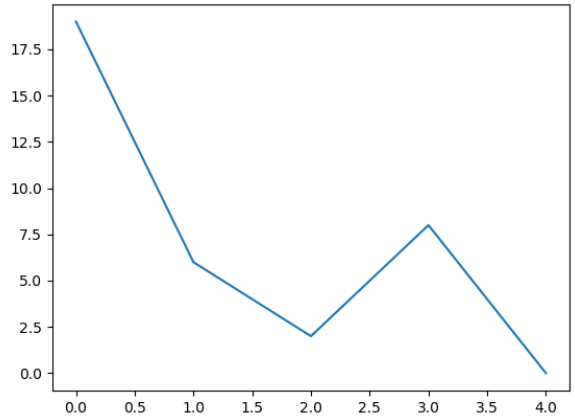
(vii) Optimal weights after last epoch
[0.11772254 -0.29104965 -0.40756788]

They are not the same as the initial weights but as there can be multiple converging points because of inequality can be satisfied by different set of weights, but the line that we got would be similar to the initial graph.

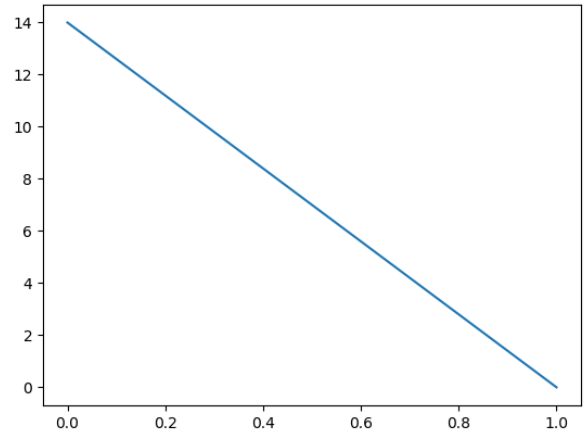
k) The graph epoch number vs the misclassification, Learning rate = 1



l) Learning rate = 10

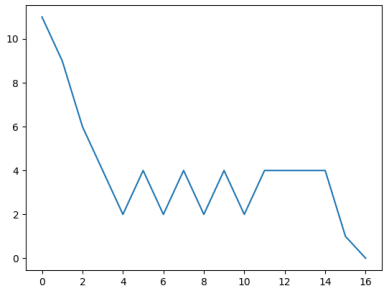


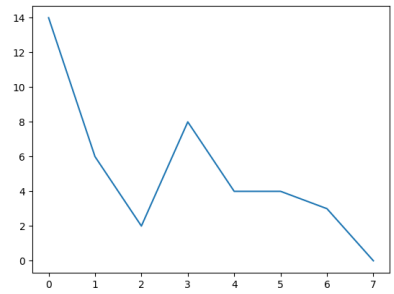
m) Learning rate = 0.1

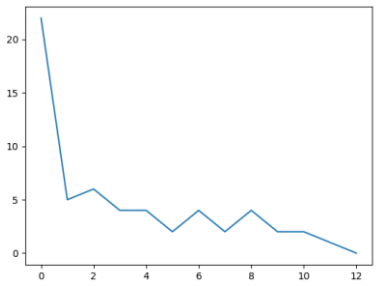


n) If the epoch is set to too high like in the case rate = 10 sometime it changes weights too abruptly and could also leading to increase in misclassification as it changes too drastically. When the learning rate is too low like 0.1 it takes time to converge and sometimes can converge too early if it's near a local maxima (near an optimal set of weights).

o) It might not be the case, if we change on different set of weights and do the experiment again, as following



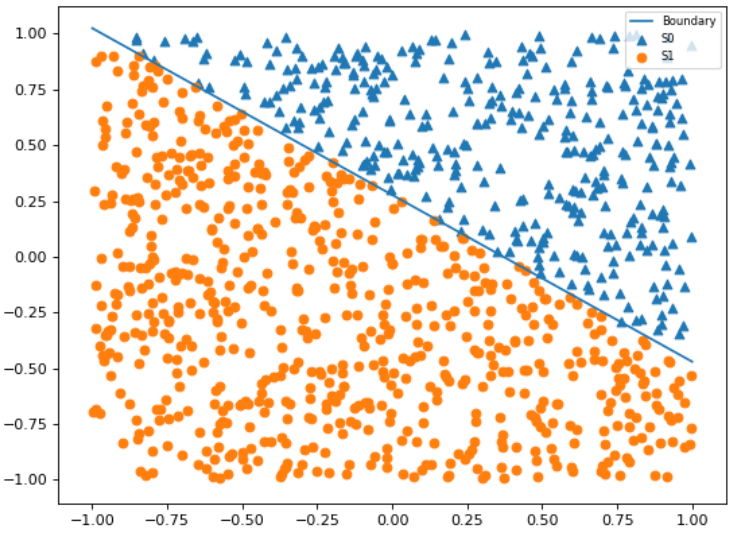


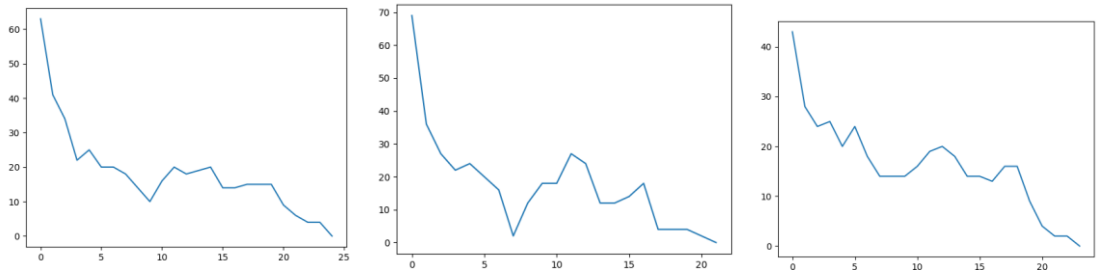


We got different graphs but the idea is similar as rate is too high the change is abrupt, if it too low then the change happens slowly. Although we can't say which would lead to faster convergence but the change of weights follows a similar fashion.

p) n (no of samples) = 1000

Following are the graph obtained with n =1000





n = 1 n = 10 n =0.1

So with n increased 1000 it takes more time to converge as the weight vector has to change with each of the sample, so it takes more no of epochs. So as the sample size increases our simple perceptron training algorithm will take more time to converge as the weight is updated with each sample.

Source Code (Python):

```
import numpy as np
import random
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure

class Perceptron:
    def __init__(self,n):
        self.W = self.getRandomWeights(-1,1)
        self.S = self.getS(n)
        self.S0,self.S1 = self.getTrainingPoint(self.S,self.W)
        self.S1_list = [list(a) for a in self.S1]
        self.S0_list = [list(a) for a in self.S0]

    def getS(self,n):
        S  = []                          #collection of input vectors
        for i in range(n):
            x1 = random.uniform(-1,1)    #x-axis
            x2 = random.uniform(-1,1)    #y-axis
            X = np.array([x1,x2])
            S.append(X)
        return S

    def getTrainingPoint(self,S,W):
        S0 = []
        S1 = []
        for X in S:
            if ((([1] + list(X) ) @ W.T) >= 0):    # Checking X @ W.T
                S0.append(X)
            else:
                S1.append(X)
        return S0, S1

    def getRandomWeights(self,a,b):
        w0 = random.uniform(-1/4,-1/4)
        w1 = random.uniform(a,b)
        w2 = random.uniform(a,b)
        W = np.array([w0,w1,w2])         # weight vector Ω
        return W

    # for plotting the graphs
    def graph(self,W):
        w0,w1,w2 = W
        x = np.array(range(-1,2))
        figure(figsize=(8,6), dpi=80, facecolor='w', edgecolor='k')
        plt.plot(x, -((w1*x + w0)/w2),label='Boundary')
```

```
        S0 = self.S0
        S1 = self.S1
        xs =[S0[i][0] for i in range(len(self.S0))]
        ys= [S0[i][1] for i in range(len(S0))]
        plt.scatter(xs,ys,marker="^",label='S0')
        xs =[S1[i][0] for i in range(len(S1))]
        ys= [S1[i][1] for i in range(len(S1))]
        plt.scatter(xs,ys,marker="o",label='S1')
        plt.legend(prop={'size':7.5})
        plt.show()

    def stepFunc(self,x):
        if (x >= 0):
            return 1
        else:
            return 0

    def boolClass(self,X):
        if list(X) in self.S1_list:
            return(1)    #positive class
        elif list(X) in self.S0_list:
            return(0)    #negative class

    # Perceptron training algoritm
    def weightUpdate(self,W1,S,rate):
        mis = 0
        for X_ele in S:
            X = np.array([1] + list(X_ele))  # X input vector
            y = self.stepFunc(W1.T @ X)
            d = self.boolClass(X_ele)              #desired input
            if (y == 1 and d == 0):
                W1 = W1 - ( rate * X)
                mis += 1
            elif (y == 0 and d == 1):
                W1 = W1 + ( rate * X)
                mis += 1
        return((W1,mis))

    # changing no of epochs
    def PTA(self,W1,S,rate):
        mis = -1
        epoch = 0
        misList = []
        while mis != 0:
            W1,mis = self.weightUpdate(W1,S,rate)
            epoch = epoch + 1
            print("Missclassification = %d" % mis)
            print("epoch %d" % epoch)
            print(W1)
            misList.append(mis)
        return (epoch, misList,W1)

    def graphEpochList(self,epoch,misList):
        plt.plot(np.array(range(epoch)),misList)
        plt.show()

if __name__ == "__main__":
    ob = Perceptron(n=1000)
    ob.graph(ob.W)
    print(ob.W)
    W1 = ob.getRandomWeights(-1,1)
    print(W1)
    epoch, mislist, W1_upd = ob.PTA(W1, ob.S, rate=1)
    ob.graphEpochList(epoch, mislist)  # plotting the epoch and misclassifications
    epoch, mislist, _ = ob.PTA(W1, ob.S, rate=10)
    ob.graphEpochList(epoch, mislist)  # plotting the epoch and misclassifications
    epoch,mislist,_ = ob.PTA(W1,ob.S,rate=0.1)
    ob.graphEpochList(epoch,mislist)   #plotting the epoch vs missclassifcation
```