CS 559: Neural Networks

Homework 2

Author:
Gaetano Coppoletta
Email:
gcoppo2@uic.edu

September 14, 2022

1 Introduction

This exercise consists in the usage of the perceptron training algorithm written in python. The step function is used as activation function. In the first part random sets of points and random optimal weights are generated. Then those points are classified following the following formulas:

Let $S_1 \subset S$ denote the collection of all $x = [x_1, x_2] \in S$ satisfying $[1x_1x_2][w_0w_1, w_2]^T >= 0$
Let $S_0 \subset S$ denote the collection of all $x = [x_1, x_2] \in S$ satisfying $[1x_1x_2][w_0w_1, w_2]^T < 0$

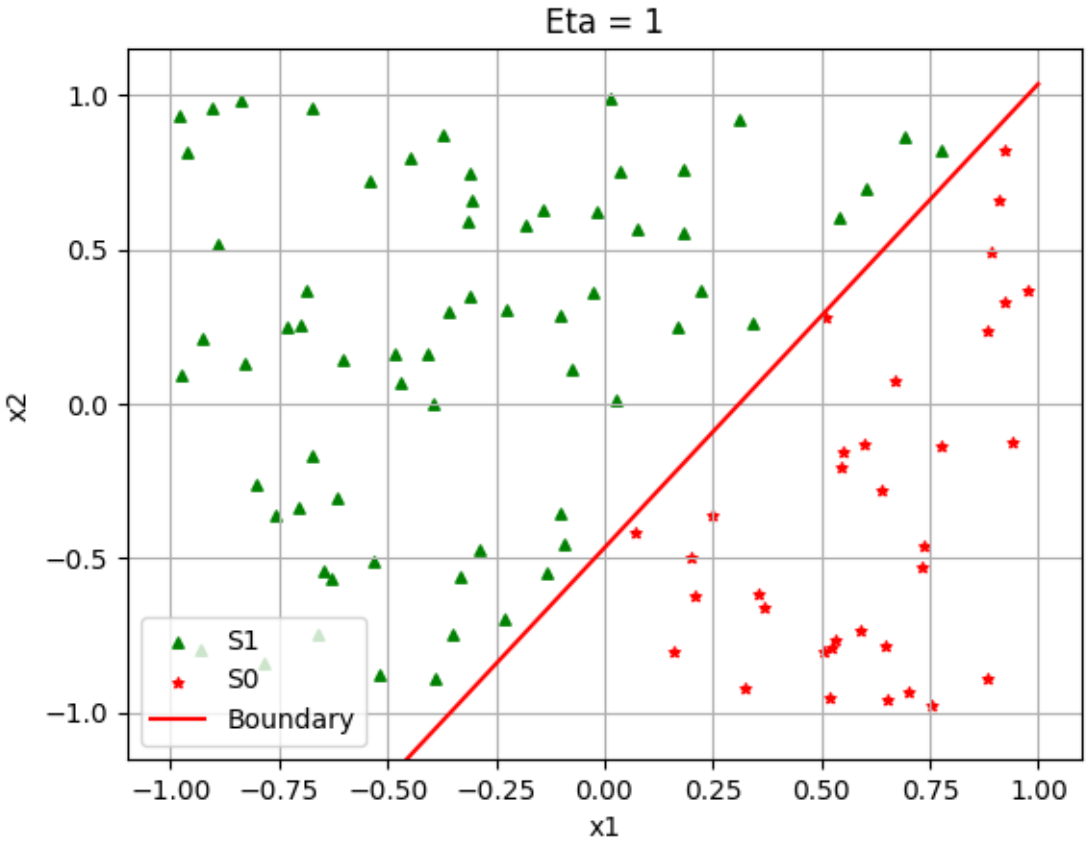
In the second part of the exercise new random weights are generated and the PTA runs in order to find weigh

2 Perceptron training algorithm

2.1 g

In the image below is showed the output of the first part of the exercise, where n=100 and η=1. In this part the code generates the two classes $S_1$and S_2.
The optimal weights are:

$w0 = 0.18010708921885632$
$w1 = -0.5793727394378525$
$w2 = 0.38555163014767446$



2.2 h

Now I use the perceptron training algorithm to find the weights that can separate the two classes defined in section g. I use as training parameter η=1, then I select the initial weights $w0^{'}, w1^{'}$ and $w2^{'}$ indipendently and uniformly at [-1,1]. In particular the values are:

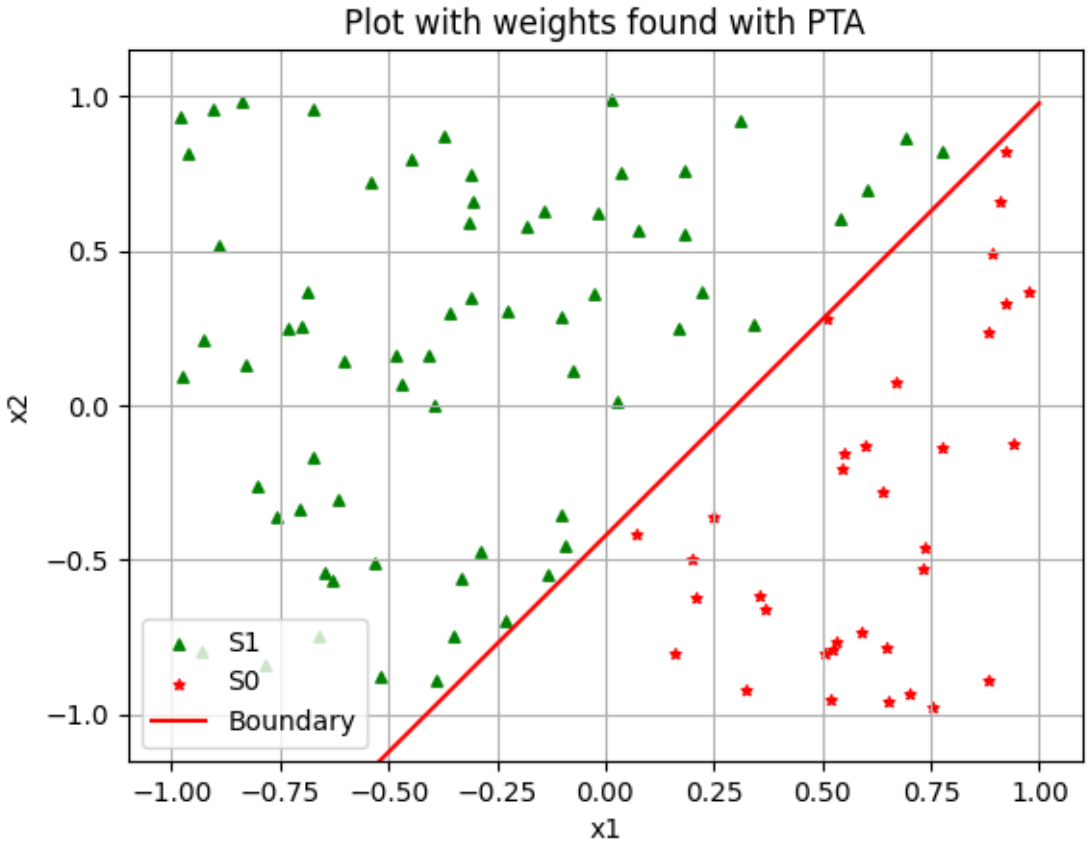
$w0^{'} = 0.918050817566082$
$w1^{'} = -0.8968130567872412$
$w2^{'} = -0.08731621603042572$

With those weights I have 10 misclassification during the first epoch. The PTA converges after 4 epoch, in particular the weights obtained are:

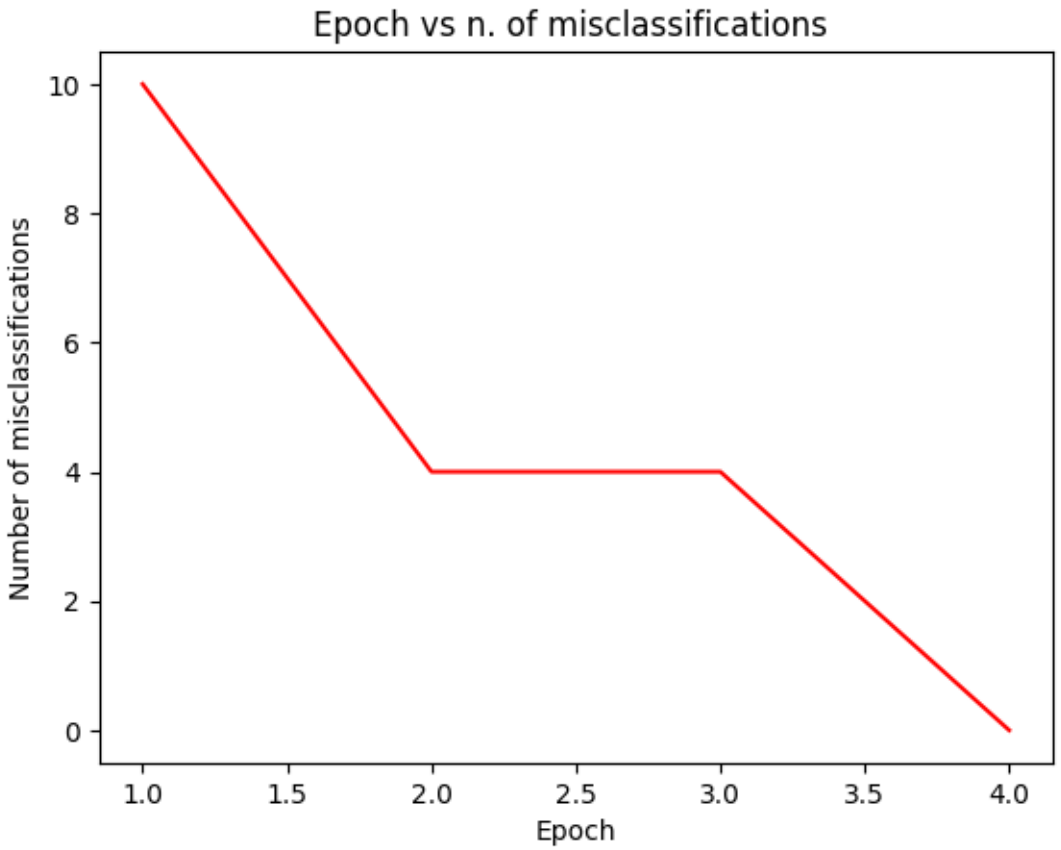
$w0^{'} = 0.918050817566082$
$w1^{'} = -3.0392376746115852$
$w2^{'} = 2.1714251443877686$

The weights obtained are different from the optimal one (Subsection g). This happens because

the problem of finding weights has not an unique solution. How can we know if the values found are in the solution domain? Simply plotting again the points and the boundary we can easily see that the classification is correct.

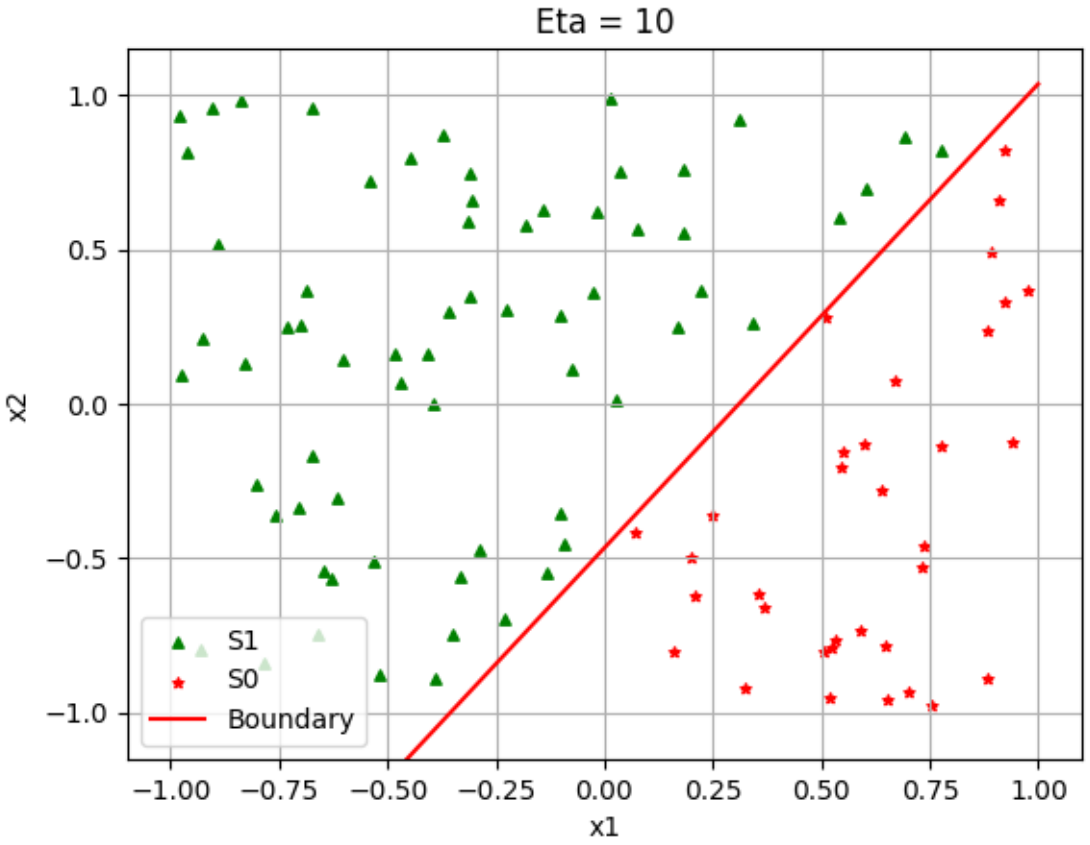


2.3 i

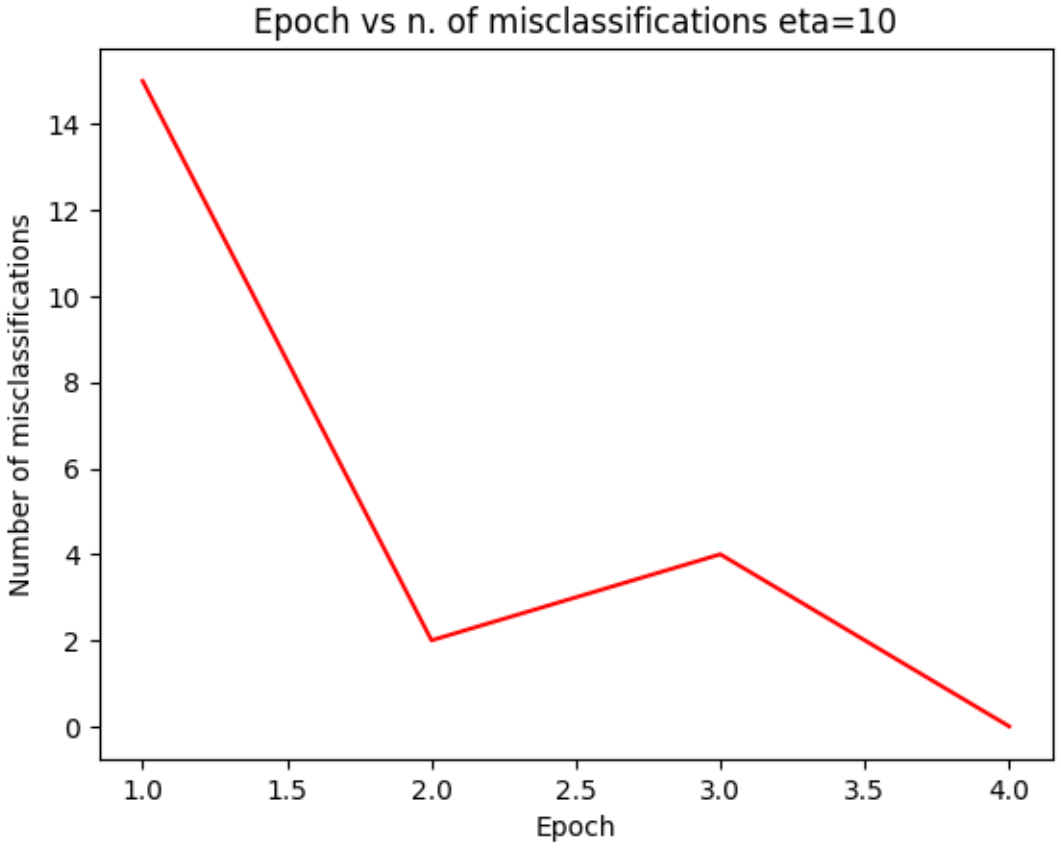


2.4 j

Now the same algorithm runs with η=10 instead of 1. The other parameters are maintained as before. The following picture shows what is obtained with this value.

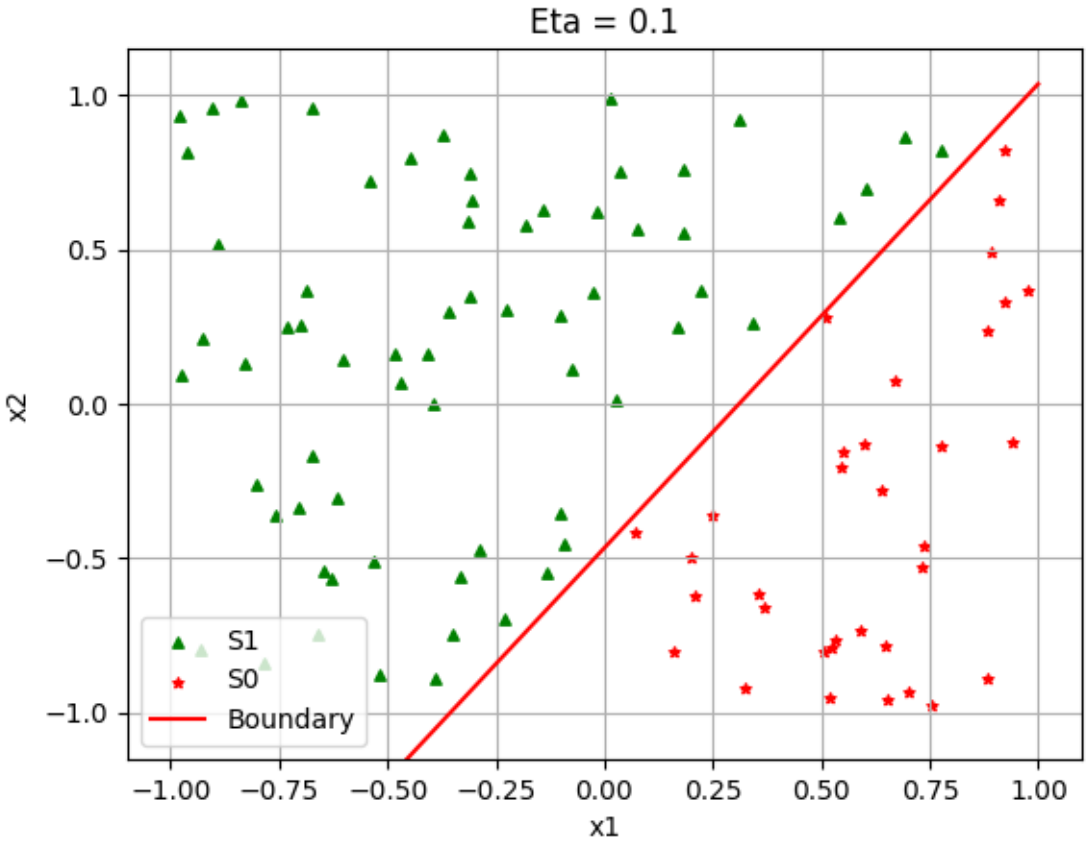


As we can see from the chart below, the algorithm converges in 4 epochs as before.

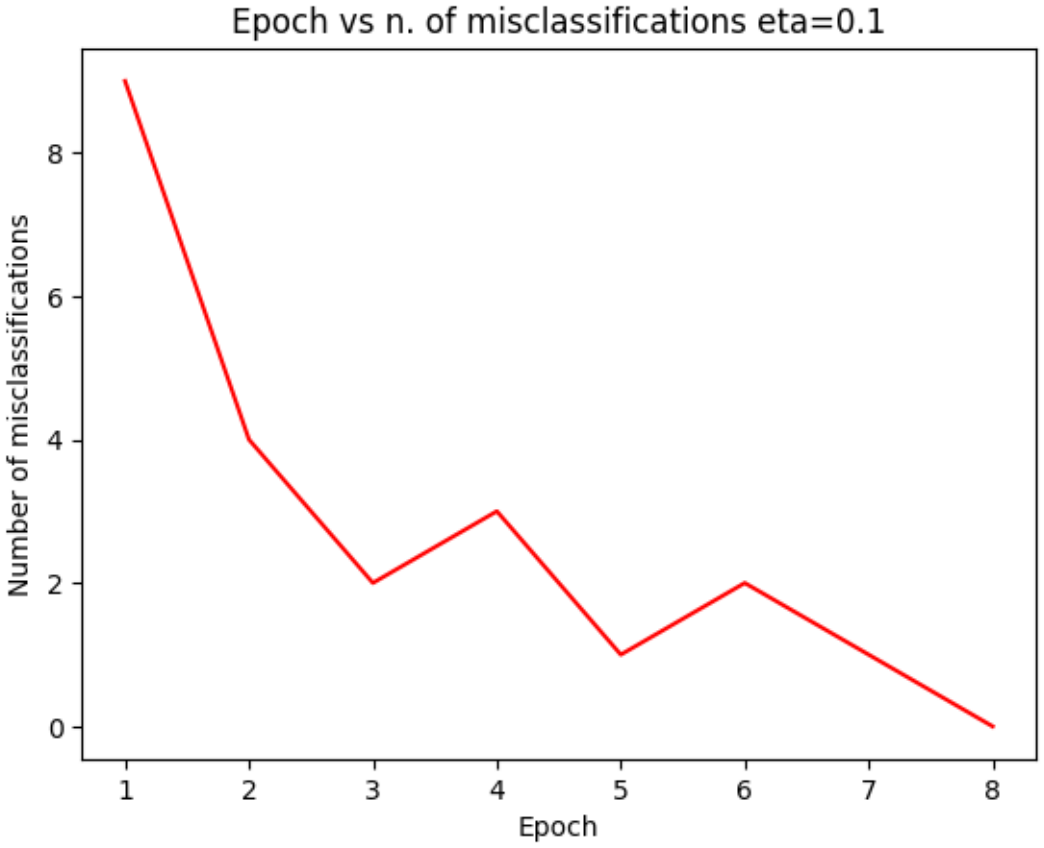


2.5 k

I change η to 0.1 instead of 1, the other parameters are maintained as before. The chart below shows the result obtained.



In this case, the algorithm converges in 8 epochs.



2.6 l

Changing the learning rate parameter has an impact on the time to converge. A small η means that the effect of the past inputs are more pronounced and the weights fluctuates less in magnitude. We can see that a smaller η (0.1) leads to a bigger number of epochs needed to the algorithm to converge. This is not a general rule, in fact,the time to converge depends also on the set of points and on the set of initial weights (random). A large learning rate parameter means a faster adaptations to changes in the underlying classes. In this case we have large fluctuations in magnitude. It may also take longer to converge.
Whatever the η is, if the classes are linearly separable, then the algorithm will converge for any $\eta > 0$.

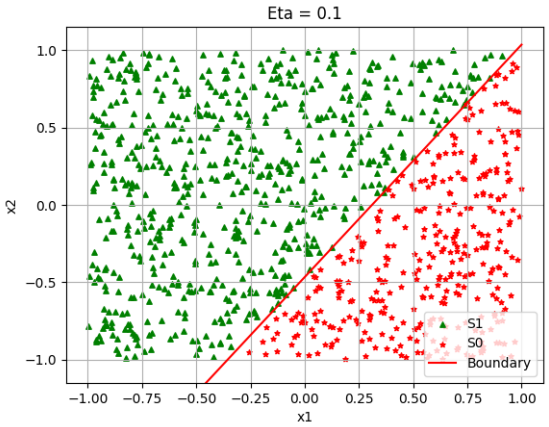
2.7 m

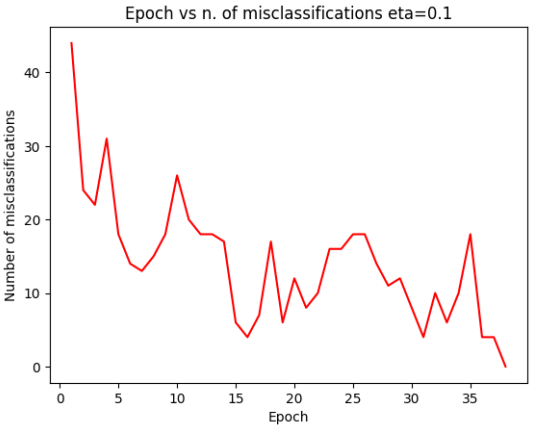
If we start with different $w_0, w_1, w_2, S, w_0^{'}, w_1^{'}, w_2^{'}$ we would not get the same results because of the randomness with which the desired output, the set of points and the starting weights for the PTA are generated. Talking about η we don't know how it will impact the new set of values. We don't know in advance how η will impact the training performance because it depends on the

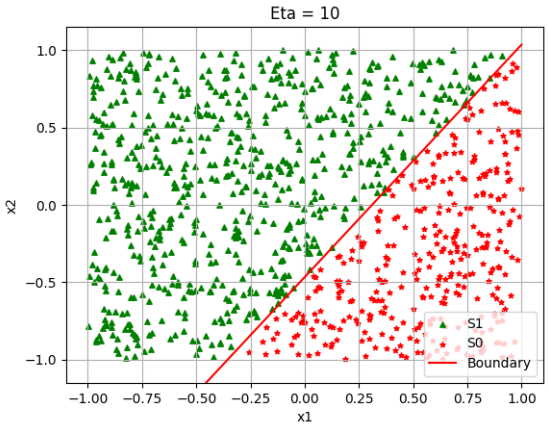
set of points, on the desired output and on the initial weights of the PTA. So the randomness of these values prevents us to know which learning rate parameter will be the best.

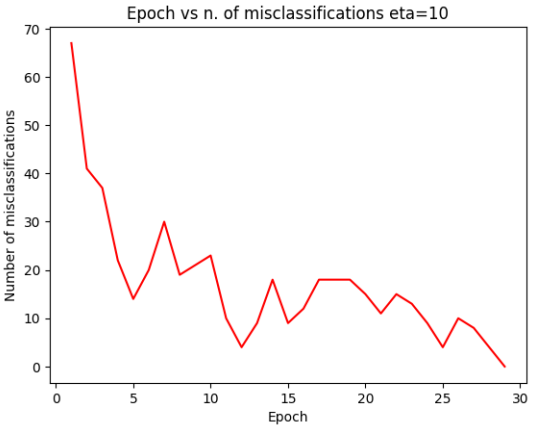
2.8 n

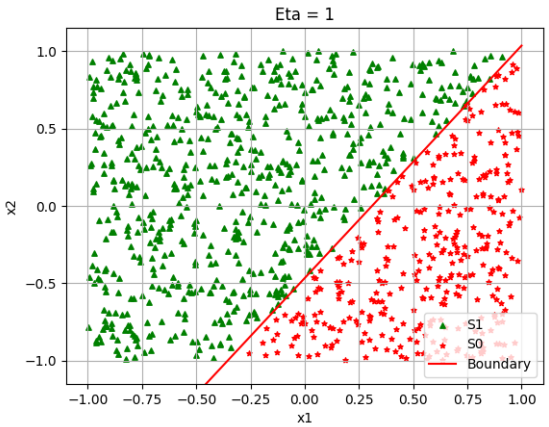
In the images below the results obtained with n=1000 instead of 100 are shown. Since more points need to be classified the algorithm is slower and converges in more time. With n =100 we have seen that the algorithm converges fast, in the example illustrated above we have 8 epochs in the worst case. With n=1000, instead, we notice that the PTA converges in more than 30 epochs, this seems to be related to the bigger number of points.

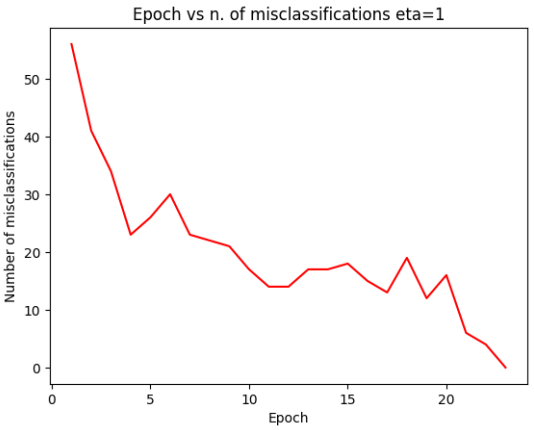












3 Code

```
1
2 import numpy as np
3 from matplotlib import pyplot as plt
```

```
4
5 if __name__ == '__main__':
6     w0= np.random.uniform(-1/4,1/4)
7     w1=np.random.uniform(-1,1)
8     w2=np.random.uniform(-1,1)
9     print("w0",w0)
10    print("w1",w1)
11    print("w2",w2)
12    n_points=100
13
14    S = np.random.uniform(-1, 1, size=(n_points, 3))
15    #S = | x1 | x2 | classification(0/1)
16
17    w = np.array([w0,w1,w2])
18    S1=np.zeros((n_points,2))
19    s1_cnt=0
20    s0_cnt=0
21    S0=np.zeros((n_points,2))
22    x=np.zeros(n_points)
23
24    for i in range(n_points):
25        X=np.array([1, S[i][0], S[i][1]])
26        #X*w
27        res = np.dot(X,w)
28        if(res>=0):
29            S1[s1_cnt][0]=S[i][0]
30            S1[s1_cnt][1]=S[i][1]
31            #save the desired output
32            S[i][2]=1
33            s1_cnt+=1
34        else:
35            S0[s0_cnt][0]=S[i][0]
36            S0[s0_cnt][1]=S[i][1]
37            #save the desired output
38            S[i][2]=0
39            s0_cnt+=1
40
41
42    plt.title("Eta = 1")
43
44    plt.xlabel("x1")
45    plt.ylabel("x2")
46    plt.ylim(-1.15,1.15)
47    plt.grid()
48    plt.scatter(S1[0:s1_cnt,0],S1[0:s1_cnt,1],color='green',marker='^',s=15)
49    plt.scatter(S0[0:s0_cnt, 0], S0[0:s0_cnt, 1], color='red', marker='*', s=15)
50
51    x = np.linspace(-1,1,1000)
52    y=(-w0-w1*x)/w2
53    plt.plot(x,y,'r')
54    plt.legend(["S1", "S0","Boundary"])
55    plt.show()
56
```

```
    #(h) PTA
    #eta
    n=1

    w0_= np.random.uniform(-1,1)
    w1_=np.random.uniform(-1,1)
    w2_=np.random.uniform(-1,1)
    w=np.array([w0_,w1_,w2_])

    print(w[0],w[1],w[2])

    em = np.zeros((1000,2),dtype=float)
    #| epoch | misclassification |
    index=0
    misclassification=-1
    em[index][0]=1

    while(misclassification!=0):
        #initialize the number of misclassification to 0
        misclassification=0
        em[index][1] = 0
        for i in range(0,n_points):
            X=np.array([1, S[i][0], S[i][1]])
            res = np.dot(X,w)
            if(res>=0 and S[i][2]==0):
                #update the weights
                #weights=weights- n*xi

                w = w - (n*X.transpose())
                em[index][1]+=1
                misclassification+=1
            elif(res<0 and S[i][2]==1):
                w = w + (n*X.transpose())
                em[index][1]+=1
                misclassification+=1
        #print("Misclassification ",em[index][1],"Epoch: ",em[index][0])
        em[index+1][0]=em[index][0]+1
        index+=1

    print(w[0],w[1],w[2])
    plt.title("Epoch vs n. of misclassifications eta=1")
    plt.xlabel("Epoch")
    plt.ylabel("Number of misclassifications")
    plt.plot(em[0:index,0],em[0:index,1],'r')
    plt.show()

    plt.title("Plot with weights found with PTA")

    plt.xlabel("x1")
    plt.ylabel("x2")
    plt.ylim(-1.15,1.15)
    plt.grid()
    plt.scatter(S1[0:s1_cnt,0],S1[0:s1_cnt,1],color='green',marker='^',s=15)
```

```
110     plt.scatter(S0[0:s0_cnt, 0], S0[0:s0_cnt, 1], color='red', marker='*', s=15)
111     x = np.linspace(-1,1,100)
112     y=(-w[0]-w[1]*x)/w[2]
113     plt.plot(x,y,'r')
114     plt.legend(["S1", "S0","Boundary"])
115     plt.show()
```