# CS 559 Homework 2

Huiyang Zhao
655490960

**Codes and figures generated are attached after comments.**

## Output of the program

```
------N = 100------
Optimal Weights: w0=[0.09565754], w1=[0.37954205], w2=[-0.55956994]
Initial weights: w0=[0.8907792], w1=[0.0298676], w2=[-0.13195889]
------Learning Rate η = 1------
Final weights: w0'=[7.8907792] w1'=[31.56439056] w2'=[-47.4250532]
Epoch Number:  7
------Learning Rate η = 10------
Final weights: w0'=[80.8907792] w1'=[313.71697874] w2'=[-461.0402344]
Epoch Number:  8
------Learning Rate η = 0.1------
Final weights: w0'=[0.8907792] w1'=[3.29441302] w2'=[-4.74087376]
Epoch Number:  8
------N = 1000------
Actual Weights: w0=[0.09565754], w1=[0.37954205], w2=[-0.55956994]
Initial weights: w0=[0.13981093], w1=[0.47700226], w2=[-0.841543]
------Learning Rate η = 1------
Final weights: w0'=[55.13981093] w1'=[216.9731698] w2'=[-319.18216933]
Epoch Number:  50
------Learning Rate η = 10------
Final weights: w0'=[560.13981093] w1'=[2193.0941528] w2'=[-3217.44465119]
Epoch Number:  26
------Learning Rate η = 0.1------
Final weights: w0'=[5.73981093] w1'=[22.75078849] w2'=[-33.39199751]
Epoch Number:  16
```

## Results and Comments

(h) vii. Write down the final weights you obtain in your report. How does these weights compare to the “optimal” weights [w0, w1, w2]?

**Answer.**

Final weights and ‘Optimal’ weights are outlined above.

It looks like that the final weights are not even ‘close’ to the optimal one. However, final weights w0' / w1' / w2' is so close to ‘optimal’ weights w0 / w1 / w2 that the line w0' + w1'x1 + w2'x2 = 0 can separate points in S1 and S0 correctly.

(l) Comment on how the changes in η effect the number of epochs needed until convergence.

**Answer.**

For N=100, when η = 1, the training converges faster than other situations.

On the other hand, for N=1000, when η = 0.1, it converges faster than when η = 1 or 10.

That means:

In different question, there are different optimal η.

η close to the optimal one will result in fast convergence, which requires a smaller number of epochs, while η far from the optimal one will result in slow convergence with larger number of epochs.

(m) Comment on whether we would get the exact same results (in terms of the effects of η on training performance) if we had started with different w0, w1, w2, S, w0', w1', w2'.

**Answer**

The fact is that with different w0, w1, w2, S, w0', w1', w2' we won't get the exact same results.
Suppose we start with w0', w1', w2' that are exactly 1 epoch from w0, w1, w2 on S when η = 10, then it might be tens of steps when η = 1, and hundreds of steps when η = 0.1
Other the other hand, if initial weights are pretty close to w0, w1, w2 on S, when η = 0.1 it will take much less time to converge then when η = 10.
We can conclude that both initial weights, optimal weights and the dataset S itself have great influence on convergence and the result.

(n) Do the same experiments with n=1000 samples. Comment on the differences compared to n=100.

**Answer:**

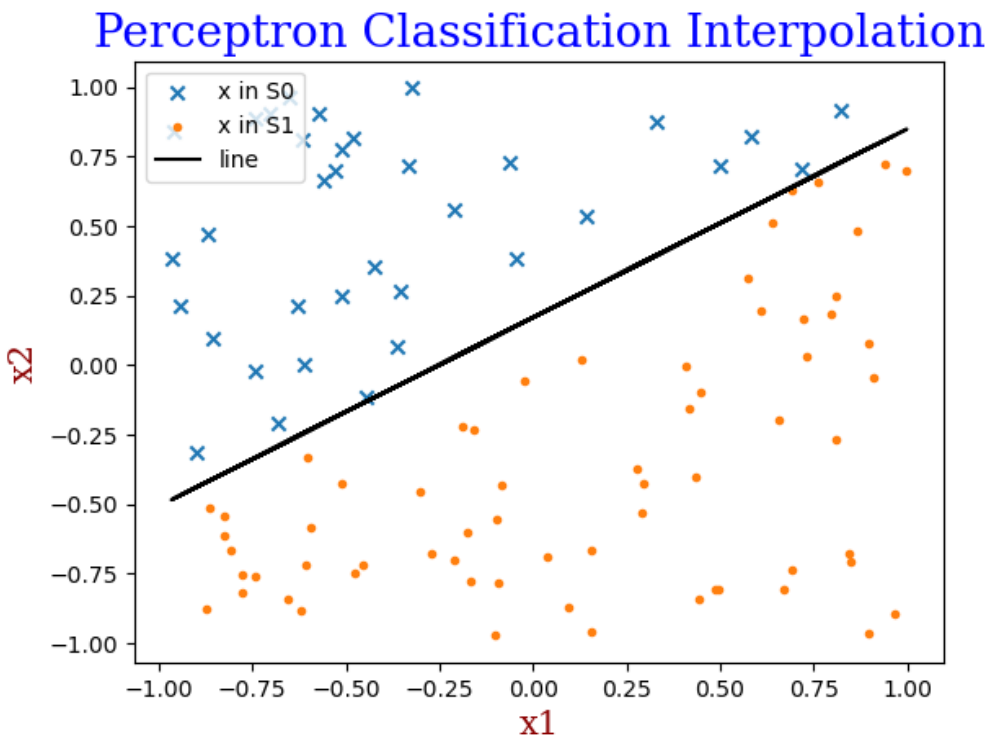
It is pretty clear that for n=1000, which can be described as an increase size of input dataset, the time required to converge is also increased significantly.
On the other hand, a larger input dataset will make the convergence more accurate and closer to the optimal separating line.
Also, a small dataset might end up with an overfit function with weights calculated by the network. Overfit means the network will work very good with the training set, but might fail in the training of test set.

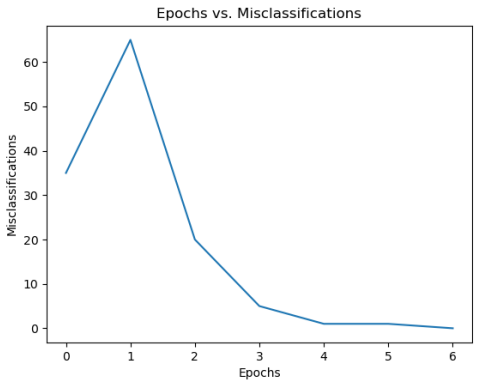
```python
# CS559 Neural Network
# Huiyang Zhao
# UIN 655490960

import numpy as np
import matplotlib.pyplot as plt


# step activation function u(·)
def step(x):
    if x >= 0:
        return 1
    else:
        return 0


font1 = {'family': 'serif', 'color': 'blue', 'size': 20}
font2 = {'family': 'serif', 'color': 'darkred', 'size': 15}

# (a) Pick (your code should pick it) w0 uniformly at random on [-1/4,1/4].
# (b) Pick w1 uniformly at random on [-1, 1].
# (c) Pick w2 uniformly at random on [-1, 1].
w0 = np.random.uniform(-0.25, 0.25, 1)
w1 = np.random.uniform(-1, 1, 1)
w2 = np.random.uniform(-1, 1, 1)

# (d) Pick n = 100 vectors x1,...,xn independently and uniformly at
# random on [-1, 1]^2, call the collection of these vectors S.
# (e) Let S1 ⊂ S denote the collection of all x = [x1 x2] ∈ S
# satisfying [1 x1 x2][w0 w1, w2]^T ≥ 0.
# (f) Let S0 ⊂ S denote the collection of all x = [x1 x2] ∈ S
# satisfying [1 x1 x2][w0 w1, w2]^T < 0.
S, S1, S0 = [], [], []
x1S1, x2S1 = [], []
x1S0, x2S0 = [], []
x1line, x2line = [], []
Label = []

# (n) Do the same experiments with n = 1000 samples.
# N = 100
# N = 1000
Sizes = [100, 1000]

for N in Sizes:
    print('------N = {}------'.format(N))
    for i in range(N):
        x_i = np.random.uniform(-1, 1, 2)
        S.append(x_i)

        output = step(w0 + w1 * x_i[0] + w2 * x_i[1])
        Label.append(output)

        x1_val = x_i[0]
        x1line.append(x1_val)
        x2_val = (-w0 - w1 * x_i[0]) / w2
        x2line.append(x2_val)
```

```
        if output == 1:
            S1.append(x_i)
            x1S1.append(x_i[0])
            x2S1.append(x_i[1])
        elif output == 0:
            S0.append(x_i)
            x1S0.append(x_i[0])
            x2S0.append(x_i[1])

    # (g) In one plot, show the line w0 + w1x1 + w2x2 = 0, with x1 being the
    # "x-axis" and x2 being the "y-axis."
    # In the same plot, show all the points in S1 and all the points in S0.
    # Use different symbols for S0 and S1.
    plt.figure()
    plt.title("Perceptron Classification Interpolation", fontdict=font1)
    plt.xlabel("x1", fontdict=font2)
    plt.ylabel("x2", fontdict=font2)
    plt.scatter(x1S0, x2S0, marker='x')
    plt.scatter(x1S1, x2S1, marker='.')
    plt.plot(x1line, x2line, 'k-')
    plt.legend(["x in S0", "x in S1", "line"], loc="upper left")
    plt.show()
    # plt.draw()
    print('Optimal Weights: w0={}, w1={}, w2={}'.format(w0, w1, w2))

    # (h) Use the perceptron training algorithm to find the weights that can
    # separate the two classes S0 and S1
    # (j) Repeat the same experiment with η = 10. Do not change w0, w1, w2,
    # S, w00, w01, w02. As in the case η = 1, draw a graph that shows the
    # epoch number vs the number of misclassifications.
    # (k) Repeat the same experiment with η = 0.1. Do not change w0, w1, w2,
    # S, w00, w01, w02. As in the case η = 1, draw a graph that shows the
    # epoch number vs the number of misclassifications.

    # i. Use the training parameter η = 1.
    learning_rates = [1, 10, 0.1]

    w0_init = np.random.uniform(-1, 1, 1)
    w1_init = np.random.uniform(-1, 1, 1)
    w2_init = np.random.uniform(-1, 1, 1)

    print('Initial weights: w0={}, w1={}, w2={}'.format(w0_init, w1_init,
w2_init))

    for lr in learning_rates:
        print('------Learning Rate η = {}------'.format(lr))
        # ii. Pick w0prime, w1prime, w2prime independently and uniformly at
        # random on [-1, 1]. Write them in your report.
        # print('Initial weights: w0={}, w1={}, w2={}'.format(w0_init,
w1_init, w2_init))
        w0_p = w0_init
        w1_p = w1_init
        w2_p = w2_init

        flag = True
        num_epoch = 0
```

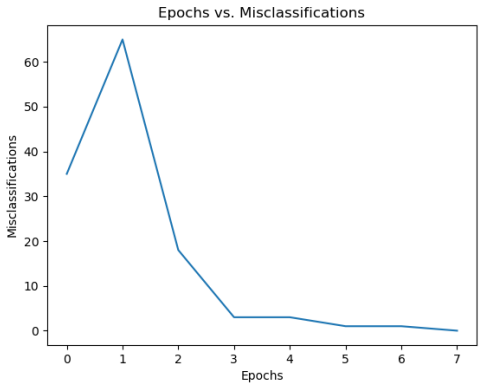
```
        array_misclassified = []

        while flag:
            # iii.Record the number of misclassifications.
            num_misclassified = 0
            # iv.After one epoch of the perceptron training algorithm, you
            # will find a new set of weights
            w0_pp = w0_p
            w1_pp = w1_p
            w2_pp = w2_p
            # print('Current weights: w0={} w1={} w2={}'.format(w0_p, w1_p,
w2_p))
            # vi.Do another epoch of the perceptron training algorithm,
            # find a new set of weights, record the number of
            # misclassifications, and so on, until convergence.
            for i in range(N):
                output = step(w0_p + w1_p * S[i][0] + w2_p * S[i][1])
                # if output == 1 and Label[i] == 0:
                #     num_misclassified += 1
                #     w0_pp = w0_pp - lr * 1
                #     w1_pp = w1_pp - lr * S[i][0]
                #     w2_pp = w2_pp - lr * S[i][1]
                # elif output == 0 and Label[i] == 1:
                #     num_misclassified += 1
                #     w0_pp = w0_pp + lr * 1
                #     w1_pp = w1_pp + lr * S[i][0]
                #     w2_pp = w2_pp + lr * S[i][1]
                if output != Label[i]:
                    num_misclassified += 1
                    w0_pp = w0_pp + lr * 1 * (Label[i] - output)
                    w1_pp = w1_pp + lr * S[i][0] * (Label[i] - output)
                    w2_pp = w2_pp + lr * S[i][1] * (Label[i] - output)

            num_epoch += 1
            w0_p = w0_pp
            w1_p = w1_pp
            w2_p = w2_pp
            # v.Record the number of misclassifications
            array_misclassified.append(num_misclassified)
            if num_misclassified == 0:
                flag = False

        # vii.Write down the final weights you obtain in your report. How
        # does these weights compare to the “optimal” weights[w0, w1, w2]?
        print('Final weights: w0\'={} w1\'={} w\'={}'.format(w0_p, w1_p,
w2_p))
        print('Epoch Number: ', num_epoch)

        # (i) Regarding the previous step, draw a graph that shows the epoch
        # number vs the number of misclassifications.
        array_epoch = list(range(num_epoch))
        plt.figure()
        plt.title('Epochs vs. Misclassifications')
        plt.xlabel('Epochs')
        plt.ylabel('Misclassifications')
        plt.plot(array_epoch, array_misclassified)
        plt.show()
```

# Graphs for N = 100

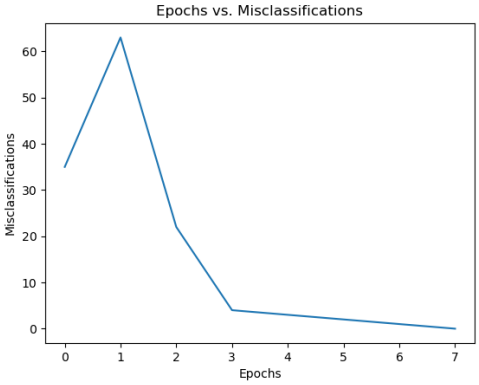



(a) $\eta = 1$

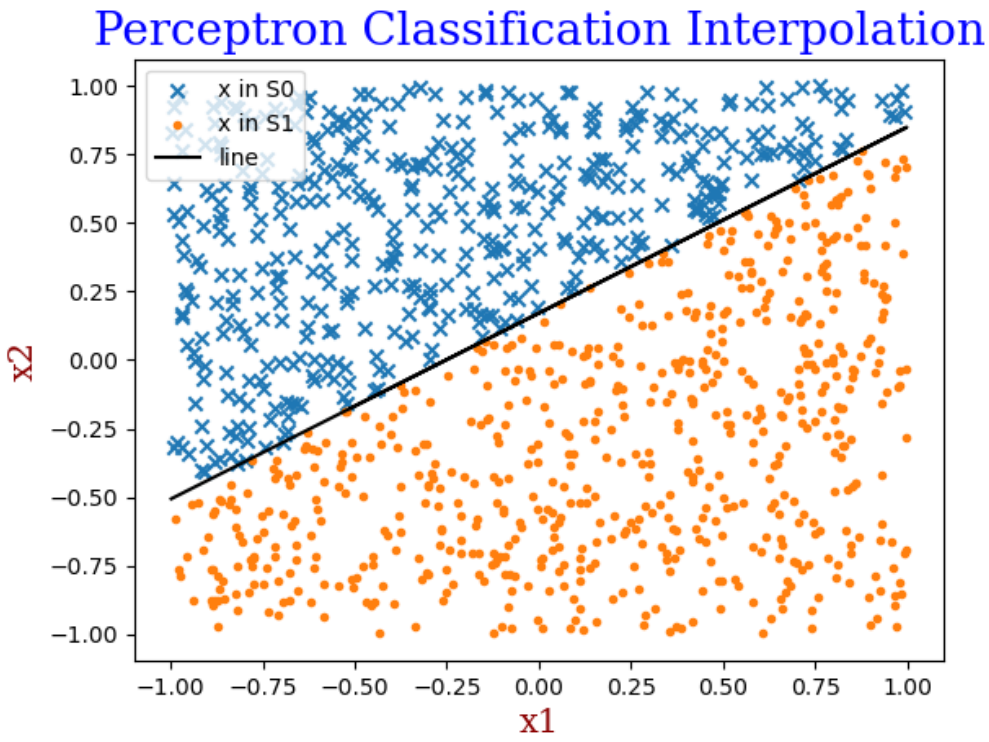


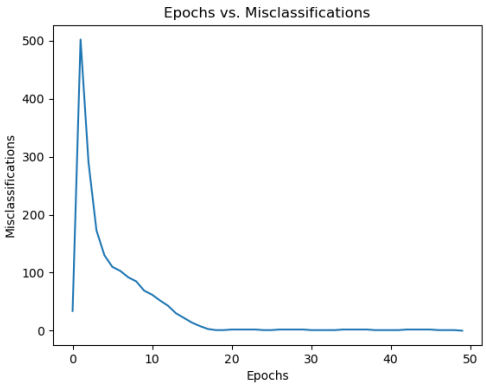

(b) $\eta = 10$



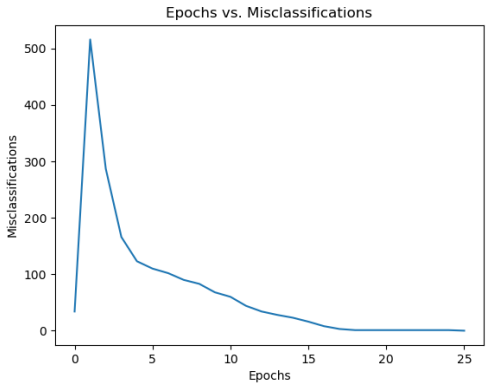

(c) $\eta = 0.1$

# Graphs for N = 1000

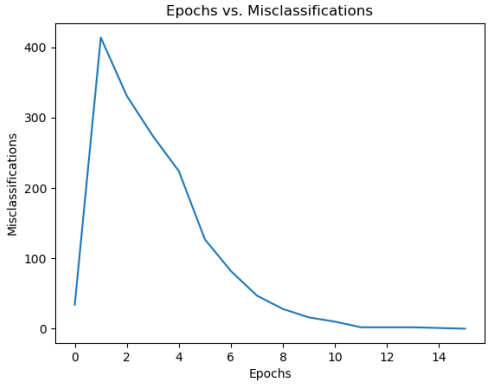



(a) $\eta = 1$




(b) $\eta = 10$




(c) $\eta = 0.1$