CS 559 Neural Networks HW #2 Report

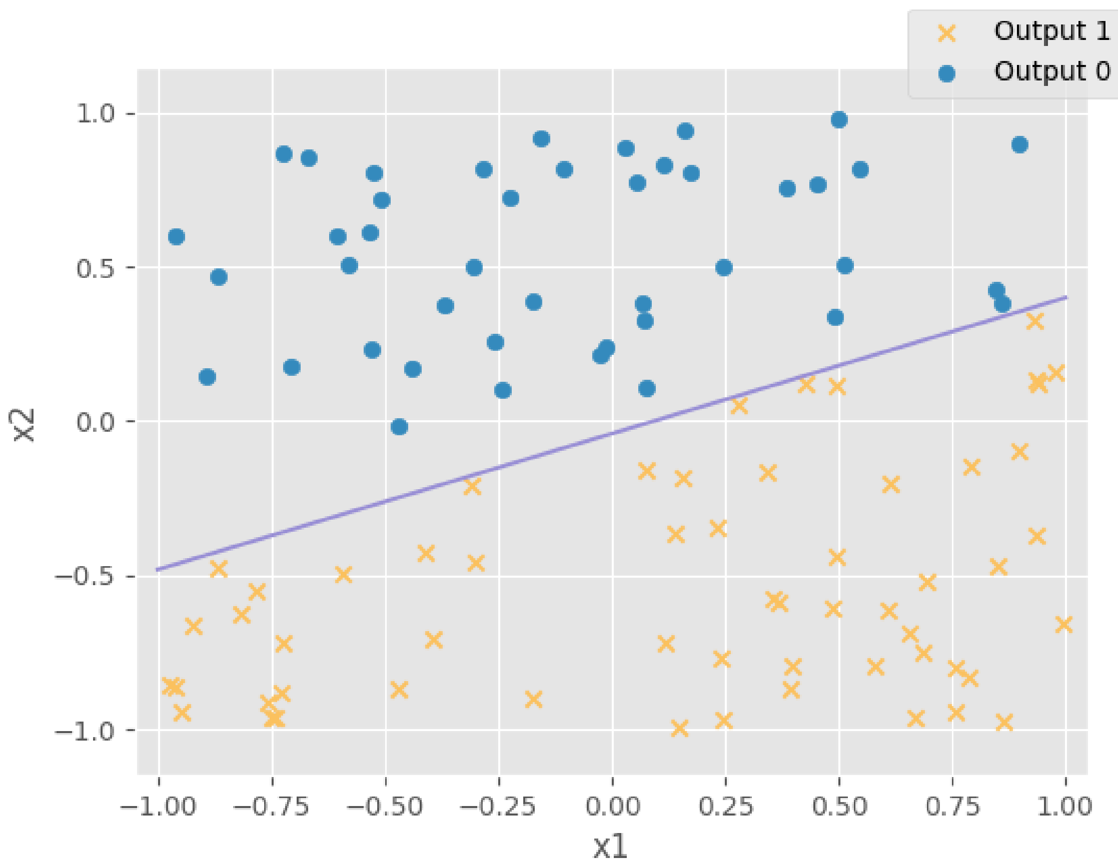
Author: Kai Bonsol

Index: I. Description, II. Results, III. Code

I. Description

I wrote a computer program that runs the perceptron training algorithm with the step activation function.

This program randomly acquires a true weights vector for n = 100 as $\mathrm{w\_true} =$ $[-0.0415\ 0.4406\ -0.9998]$. Then defines a dataset S to contain n vectors x_i randomly sampled from $[-1, 1]^2$. S0 is defined as the subset of S which satisfies $[1\ x1\ x2]\ [w0\ w1\ w2]^T < 0$ and S1 is defined as the subset of S which satisfies $[1\ x1\ x2]\ [w0\ w1\ w2]^T \geq 0$. Here is a plot of S0, S1, and the separating line defined by $x * w^T = 0$.

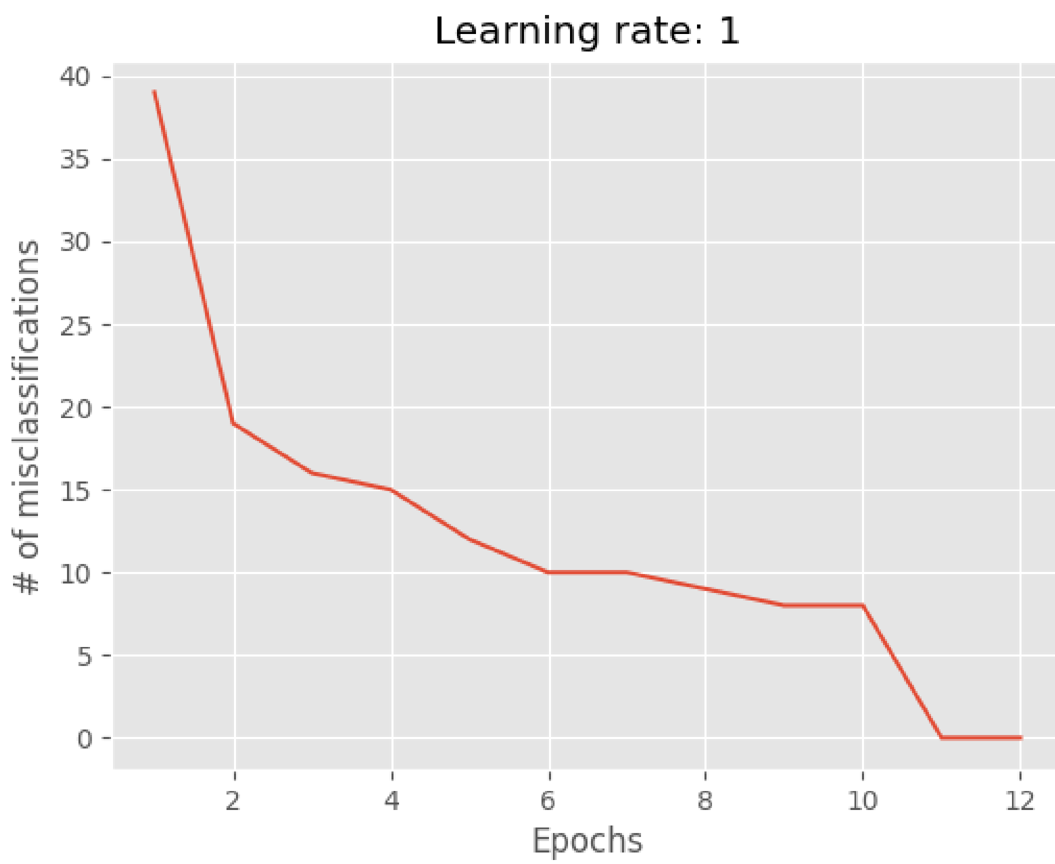


II. Results

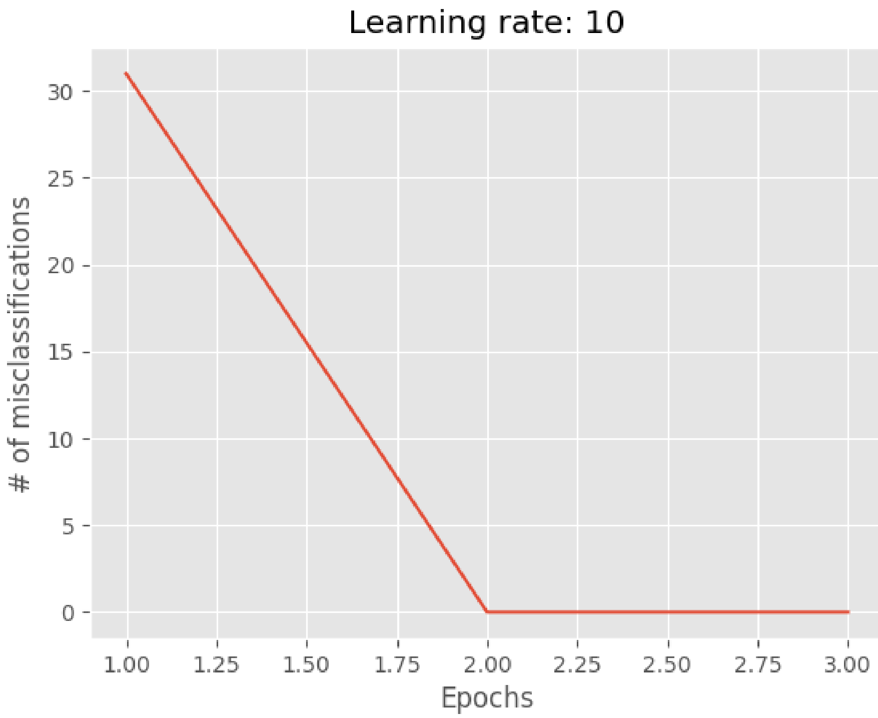
To begin the perceptron training algorithm, a weights vector is randomly initialized as $w' = [w0'\ w1'\ w2'] = [0.2831\ \text{-}0.2200\ \text{-}0.0280]$. The number of misclassifications using w' was 43, after one epoch using η = 1, we obtain w'', which yields 39 misclassifications. Continuing this process until convergence, which I defined as the updated vector being equal to the last within a 1e-03 threshold, we obtain the "optimal" weights $w\_opt = [0.0486\ 0.2973\ \text{-}0.9536]$ which are relatively close to w_true.

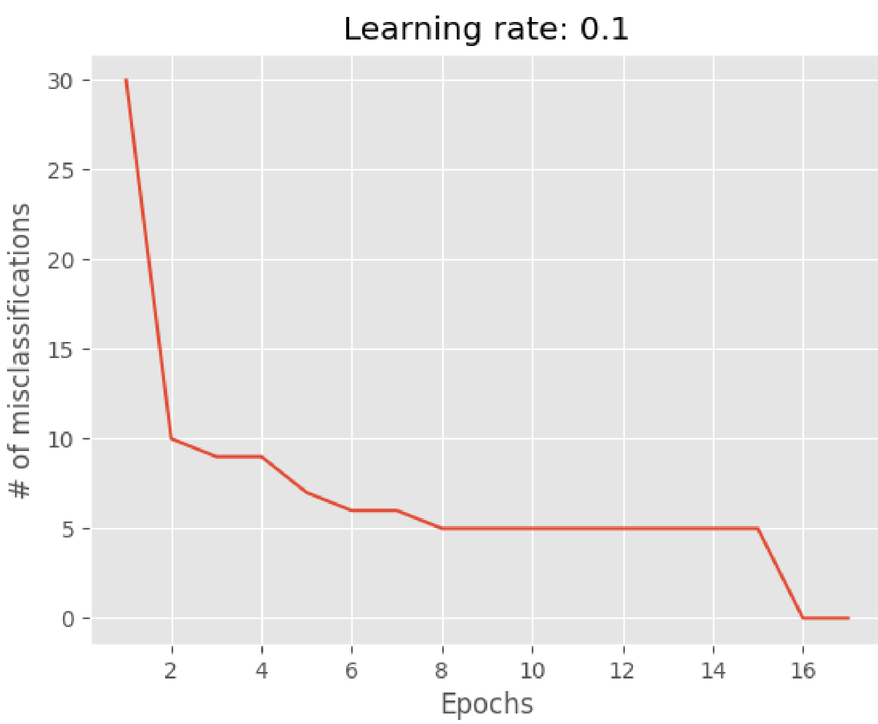
We can calculate $abs(w\_opt - w\_true) = [0.0901\ 0.1433\ 0.0462]$ as an idea of how close the optimal is to the true weights.

Analyzing the effect of different learning rates, we first graph the # of epochs against the # of misclassifications for a learning rate η = 1:



Here are the results of the same initial w being ran with learning rates η = 10 and η = 0.1:



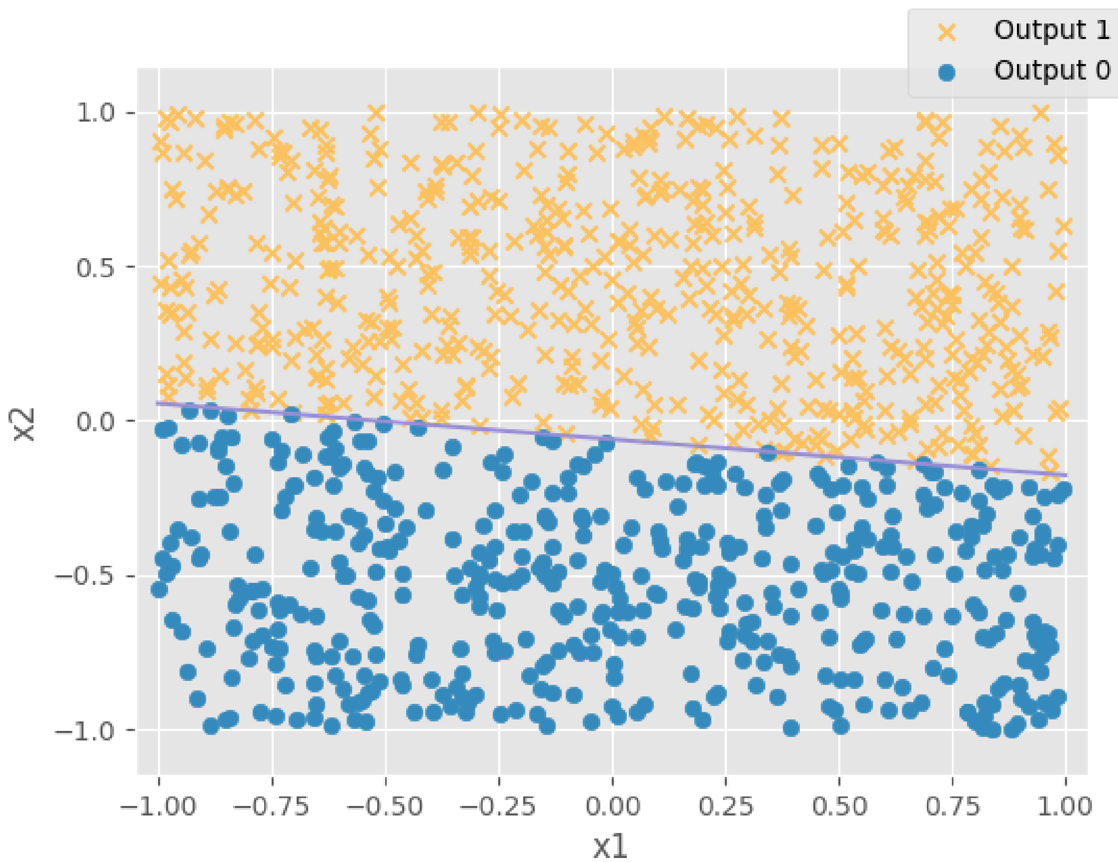


Answer to (l), (m)

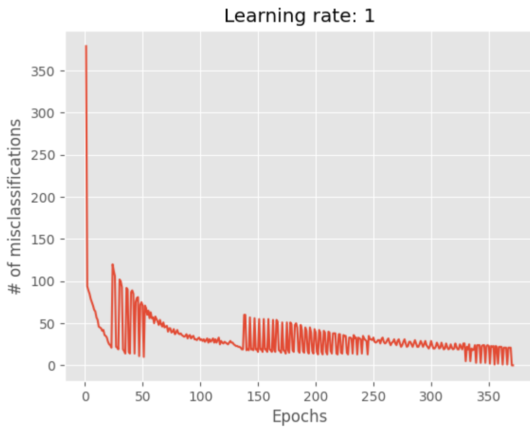
With S0, S1, and initial w, we can note that higher learning rates seem to find an optimal weight vector much quicker than lower learning rates. $\eta = 1$ yields 0 misclassifications at 12 epochs, $\eta = 0.1$ yields 0 misclassifications at 16 epochs whereas $\eta = 10$ yields 0 misclassifications at only 2 epochs. Note this result is conditioned on S0, S1, and the initial w. For instance, it is in the realm of possibility that the initial w perfectly separately S0 and S1 and there is no need for many epochs given any learning rate. It could also be such that higher learning rates for particular datasets and initial w converge slower than lower learning rates, i.e. an initial w that is already relatively close to w_true.

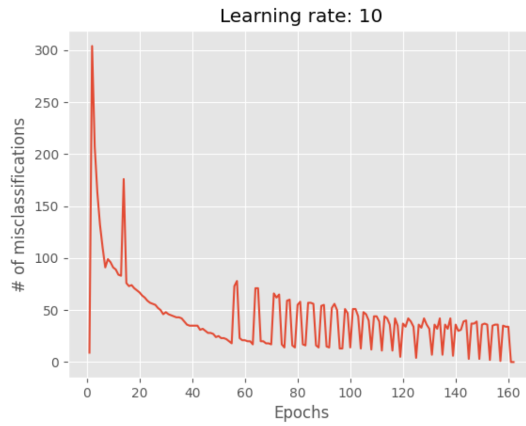
This experiment was repeated for n = 1000 samples, the following graphs are obtained:

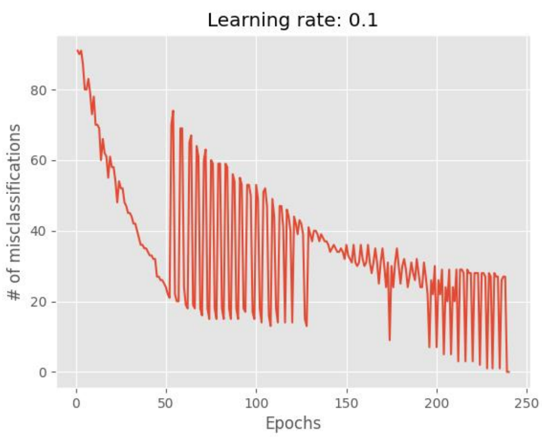
S0 and S1 and separating line:



Different learning rates- epochs vs # of misclassifications:







Here we can see the # of misclassifications tends to oscillate with each epoch, and we also see evidence that the initial w, S0 and S1 play a role in how the learning rate effects the time of convergence. Interestingly, a learning rate of 0.1 and a learning rate of 10 both outperform a learning rate of 1 in terms of convergence time.

The true weights for the n = 1000 experiment was $\mathrm{w\_true} = [0.0522 \; 0.0991 \; 0.8524]$

With a learning rate of 1, the perceptron training algorithm yields $\mathrm{w\_opt} = [0.0610 \; 0.1132 \; 0.9917]$. We can calculate $\mathrm{abs(w\_true - w\_opt)} = [0.0088 \; 0.0141 \; 0.1393]$ which has a smaller L2 norm than the abs(w_true – w_opt) for n = 100; suggesting that PTA performs better with larger datasets.

III. Code:

```python
# Author: Kai Bonsol
# CS 559 Neural Networks HW #2

import numpy as np
import matplotlib.pyplot as plt

# 1. Write a computer program that runs the perceptron training
#    algorithm with the step activation function u(.). Implement
#    the following steps and report your results.

np.random.seed(1)
def experiment(N):
    # (a)
    w0_true = np.random.uniform(-0.25, 0.25, 1)

    # (b)
    w1_true = np.random.uniform(-1, 1, 1)
    # (c)
    w2_true = np.random.uniform(-1, 1, 1)

    w_true = np.array([w0_true, w1_true, w2_true])

    # (d)
    n = N
    S = []
    for i in range(n):
        S.append(np.random.uniform(-1, 1, 2))
    S = np.asarray(S)

    # (e)
    S1 = []
    for x in S:
        if np.matmul(np.concatenate(([1], x)), w_true) >= 0:
            S1.append(x)
    S1 = np.asarray(S1)

    # (f)
    S0 = []
    for x in S:
        if np.matmul(np.concatenate(([1], x)), w_true) < 0:
            S0.append(x)
    S0 = np.asarray(S0)

    # (g)
    plt.style.use('ggplot')

    fig1, ax1 = plt.subplots()

    ax1.set_xlabel("x1")
    ax1.set_ylabel("x2")

    S1X = S1[:,0]
    S1Y = S1[:,1]
    S0X = S0[:,0]
    S0Y = S0[:,1]
```

```python
ax1.scatter(S1X, S1Y, color='C4', marker='x', label='Output 1')
ax1.scatter(S0X, S0Y, color='C8', marker='o', label='Output 0')

X = np.linspace(-1, 1)
Y = -(w0_true+w1_true*X) / w2_true

ax1.plot(X, Y, color='C9')
ax1.set_xlim([-1.05, 1.05])
ax1.set_ylim([-1.15, 1.15])

ax1.legend(loc='upper right', bbox_to_anchor=(1.05, 1.10))

fig1.savefig("hw2_true.png")
#fig1.show()

# (h): use PTA to find w0, w1, w2.

# h(i)
learning_rate = 1

# h(ii)
w0 = np.random.uniform(-1, 1, 1)
w1 = np.random.uniform(-1, 1, 1)
w2 = np.random.uniform(-1, 1, 1)

print('w0:', w0)
print('w1:', w1)
print('w2:', w2)
print()

w = np.array([w0, w1, w2])

# h(iii)
def report_misclassification(w, S0, S1):
    num_misclassifications = 0
    for x in S1:
        if np.matmul(np.concatenate(([1], x)), w) < 0:
            num_misclassifications += 1
    for x in S0:
        if np.matmul(np.concatenate(([1], x)), w) >= 0:
            num_misclassifications += 1
    return num_misclassifications

print("# misclassifications for w0', w1', w2':",
      report_misclassification(w, S0, S1))

def step(num):
    if num >= 0:
        return 1
    return 0

def run_epoch(w, learning_rate, S0, S1):
    for x in S0:
        xi = np.concatenate(([1], x))
        w = w.flatten()
        w = w + (learning_rate * (-1 * step(np.matmul(xi, w))) * xi)
```

```
        for x in S1:
            xi = np.concatenate(([1], x))
            w = w.flatten()
            w = np.add(w, learning_rate * xi * (1 - step(np.matmul(xi, w))))
        return w

    # h(iv)
    w = run_epoch(w, learning_rate, S0, S1)

    # h(v)
    print("# misclassifications for w0'', w1'', w2'':",
          report_misclassification(w, S0, S1))

    # h(vi)
    epoch = 1
    epoch_list = [1]
    misclass_list = [report_misclassification(w, S0, S1)]
    def perceptron_training_algorithm(epoch, epoch_list, misclass_list,
learning_rate, S0, S1, w):
        while True:
            epoch += 1
            epoch_list.append(epoch)
            new_w = run_epoch(w, learning_rate, S0, S1)
            misclass_list.append(report_misclassification(new_w, S0, S1))
            if np.allclose(new_w, w, rtol=1e-03):
                w = new_w
                break
            w = new_w

        w = w / np.linalg.norm(w)
        return w

    w = perceptron_training_algorithm(epoch, epoch_list, misclass_list,
learning_rate, S0, S1, w)

    # h(vii)
    print('PTA Acquired Weights:', w)
    print('True weights:', w_true)

    # (i)
    def graph_epoch_misclassification(epoch_list, misclass_list,
learning_rate, n):
        fig2, ax2 = plt.subplots()
        ax2.plot(epoch_list, misclass_list)
        ax2.set_xlabel("Epochs")
        ax2.set_ylabel("# of misclassifications")
        ax2.set_title("Learning rate: " + str(learning_rate))
        #fig2.show()
        fig2.savefig("hw2_" + str(int(learning_rate)) + "_N" + str(n) +
".png")

    graph_epoch_misclassification(epoch_list, misclass_list, learning_rate,
n)

    # (j)
    epoch = 0
    epoch_list = []
```

```
    misclass_list = []
    w = np.array([w0, w1, w2])
    learning_rate = 10
    w = perceptron_training_algorithm(epoch, epoch_list, misclass_list,
learning_rate, S0, S1, w)
    graph_epoch_misclassification(epoch_list, misclass_list, learning_rate,
n)

    # (k)
    epoch = 0
    epoch_list = []
    misclass_list = []
    w = np.array([w0, w1, w2])
    learning_rate = 0.1
    w = perceptron_training_algorithm(epoch, epoch_list, misclass_list,
learning_rate, S0, S1, w)
    graph_epoch_misclassification(epoch_list, misclass_list, learning_rate,
n)

# (l), (m) in report

experiment(100)
# (n)
experiment(1000)
```