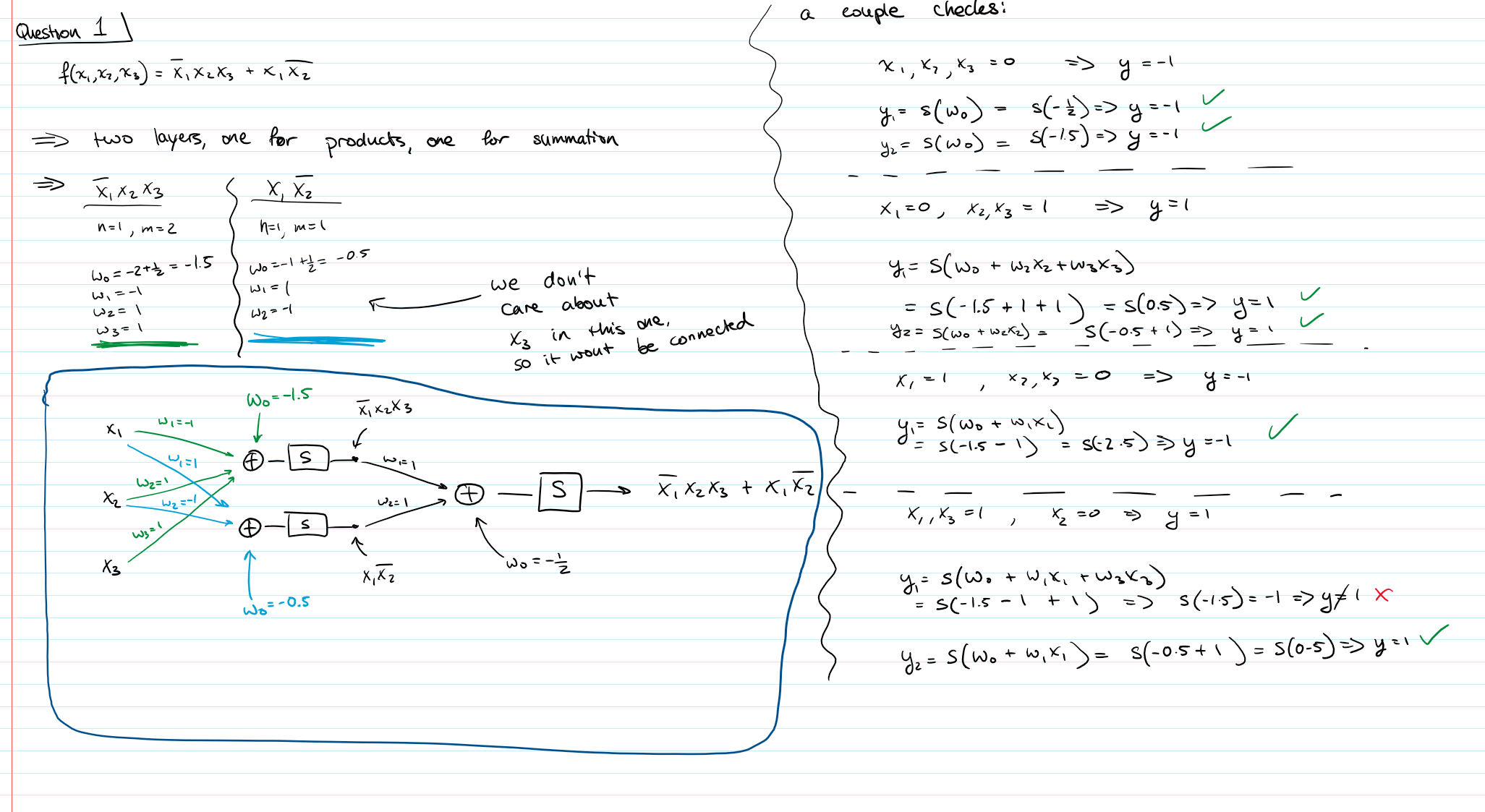
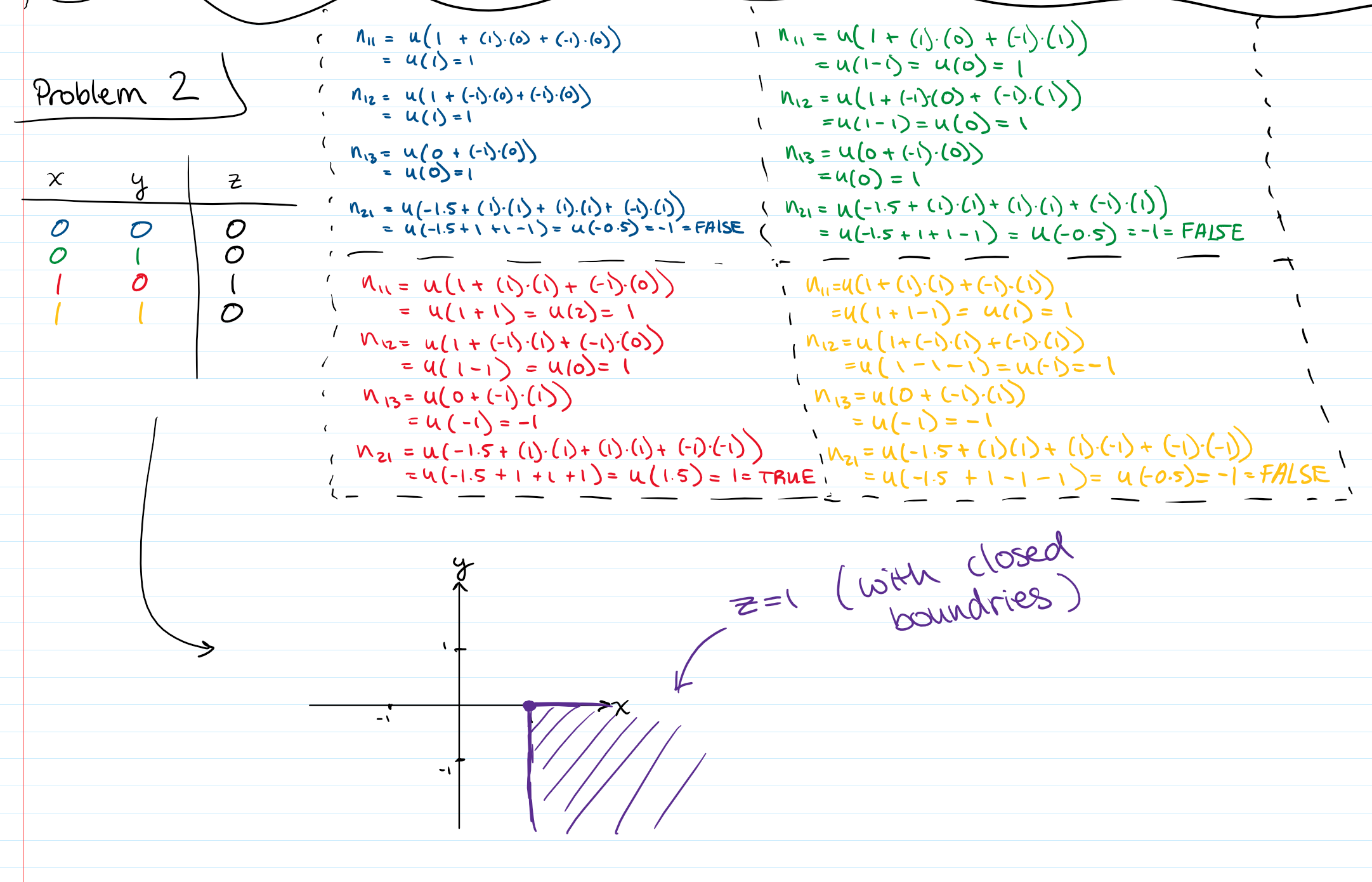
Atanas Delevski

ECE 559 Homework #1 Report

09/28/2020

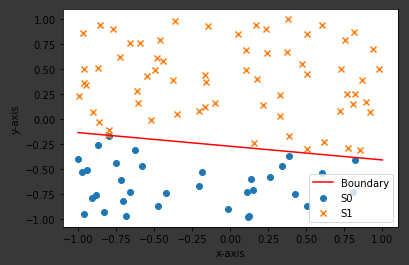




***Problem 3***

**Part E)**  W0, W1, W2 = 0.22508806, 0.11330638, 0.8312127

**Part I)** Graph Below.

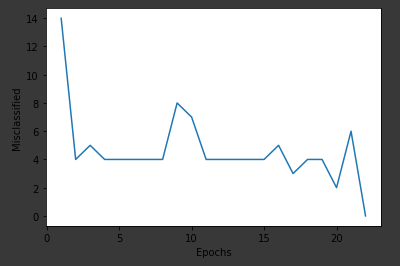


**Part J)**

**ii)** W0, W1, W2 = 0.0707831, -0.21998457, -0.02801867

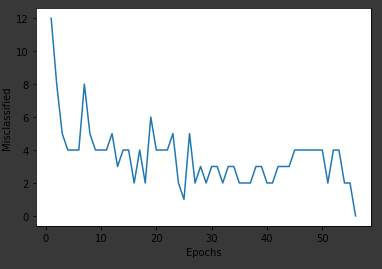
**vii)** The final weights: W0, W1, W2 = 2.0707831, 1.07574481, 7.49824272

These weights were achieved after 22 epochs, and have the same ratio as my “optimal weights”, but they are about 10 times bigger.



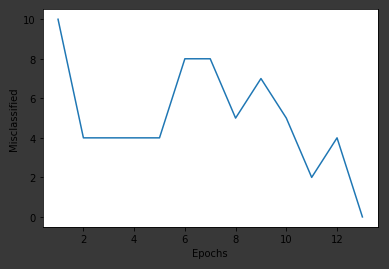
**Park K)**

**Part L)** Learning Rate (Eta) = 10



Increasing the learning rate to 10 increased the amount of epochs to 56.

**Part M)** Learning Rate (Eta) = 0.01



Decreasing the learning rate to 0.1 decreased the amount of epochs to 13.

**Part N)**

The bigger we set the learning rate, the more unstable the learning became, and the more it had to backtrack in the opposite direction after learning wrongly. Hence, the bigger learning rates meant longer learning time and more epochs. Conversely, decreasing the learning rate made the learning more stable and therefore the convergence was faster and with fewer epochs. This is probably due to the fact that the algorithm was not wasting any time backtracking as it did with a high learning rate.

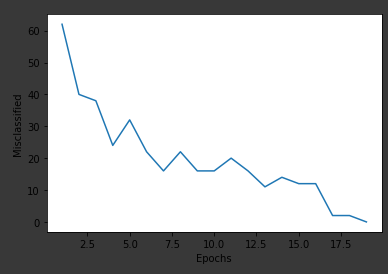
**Part O)**

Is this question asking whether changing the weights in between changing the learning rate would have changed the outcomes? Or is it asking whether changing the weights in general to different weights from the start of the experiment and running everything the same way would change the outcome? If it is the former, then the answer is YES, changing the weights in between the different learning rate changes would completely change the outcome as a different set of weights and vectors might produce a totally different difficulty of convergence. If it is the latter, then the answer is NO. Having a different set of weights and vectors all together will not change the fact that a learning rate of 10 is too high and would therefore be unstable.

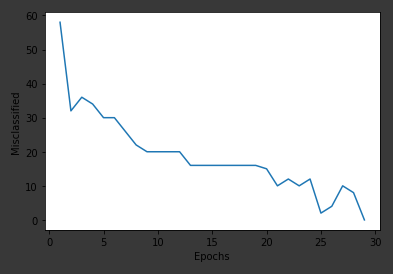
**PART P ON NEXT PAGE**

**Part P) n = 1000**

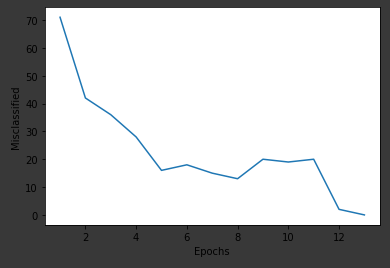
**i) Learning Rate (Eta) = 1 (19 epochs vs 22 for n = 100)**

****

**ii) Learning Rate (Eta) = 10 (29 epochs vs 56 for n = 100)**

****

**iii) Learning Rate (Eta) = 0.1 (13 epochs vs 13 for n = 100)**

****

Increasing the number of vectors (i.e. n) to 1000 increased the performance of the algorithm when it came to the learning rates of 1 and 10, by lowering the number of epochs required. But for the learning rate of 0.1, which performed the best out of all the learning rates, it actually did not change the number of epochs needed.

**CODE:**

# Author: Atanas Delevski

# Date: 09/28/2020

# Title: ECE 559 Homework 1 Question 3

from matplotlib import pyplot as plt

import numpy as np

np.random.seed(1) # for consistency

########### FUNCTIONS #############

def get\_weights\_and\_bias():

w0 = (2/4) \* np.random.random\_sample(1) - (1/4)

w1 = 2 \* np.random.random\_sample(1) - 1

w2 = 2 \* np.random.random\_sample(1) - 1

return w0, w1, w2

def get\_vectors\_s(n):

S = 2 \* np.random.random\_sample((n,2)) - 1

return S

####################################

########## INITIAL WORK ##############

S = get\_vectors\_s(100)

w0i, w1i, w2i = get\_weights\_and\_bias()

weights\_i = np.array((w0i, w1i, w2i))

# print(weights\_i)

S0 = np.empty(shape=(0,2),dtype='object')

S1 = np.empty(shape=(0,2),dtype='object')

labels = np.empty(shape=(0, 1),dtype='int')

for vector in S:

if np.dot([1, vector[0], vector[1]], weights\_i) < 0:

S0 = np.vstack((S0, vector))

labels = np.vstack((labels, 0))

elif np.dot([1, vector[0], vector[1]], weights\_i) >= 0:

S1 = np.vstack((S1, vector))

labels = np.vstack((labels, 1))

plt.scatter(([vector[0] for vector in S0]), ([vector[1] for vector in S0]), marker='o', label='S0')

plt.scatter(([vector[0] for vector in S1]), ([vector[1] for vector in S1]), marker='x', label='S1')

x = np.linspace(-1,1)

y = (-w0i-w1i\*x)/w2i

plt.plot(x, y, '-r', label='Boundary')

plt.xlabel('x-axis')

plt.ylabel('y-axis')

plt.legend()

plt.show()

w0, w1, w2 = get\_weights\_and\_bias()

weights = np.array((w0, w1, w2))

# print(weights)

#########################################

######## Perceptron Class ##################

class Perceptron(object):

def \_\_init\_\_(self, num\_of\_inputs, weights, learning\_rate):

self.num\_of\_inputs = num\_of\_inputs

self.learning\_rate = learning\_rate

self.weights = weights

self.epochs = 0

self.misclassified = 0

self.count = 0

self.misclassified\_list = []

def predict(self, inputs):

result = np.dot([1, inputs[0], inputs[1]], self.weights)

if result >= 0:

prediction = np.array([1])

else:

prediction = np.array([0])

return prediction

def train(self, training\_inputs, labels):

self.misclassified = 0

for label, inputs in zip(labels, training\_inputs):

self.count += 1

prediction = self.predict(inputs)

print(f'Input: {inputs}, Label: {label}, Prediction: {prediction}, Count: {self.count}')

if prediction > label:

self.weights[0] -= self.learning\_rate \* 1

self.weights[1] -= self.learning\_rate \* inputs[0]

self.weights[2] -= self.learning\_rate \* inputs[1]

self.misclassified += 1

elif prediction < label:

self.weights[0] += self.learning\_rate \* 1

self.weights[1] += self.learning\_rate \* inputs[0]

self.weights[2] += self.learning\_rate \* inputs[1]

self.misclassified += 1

else:

print('Prediction = Label')

print('Misclassified: ', self.misclassified)

self.misclassified\_list.append(self.misclassified)

self.epochs += 1

print(f'Weights: {self.weights}, Epochs: {self.epochs}')

#######################################################

################ USAGE ###############################

perceptron = Perceptron(2, weights, 0.1)

perceptron.train(S, labels)

print(perceptron.misclassified)

while (perceptron.misclassified != 0):

perceptron.train(S, labels)

print(perceptron.misclassified\_list)

epochs = np.arange(1, perceptron.epochs+1)

plt.plot(epochs, perceptron.misclassified\_list)

plt.xlabel('Epochs')

plt.ylabel('Misclassified')

plt.show()

#####################################################