Name: Ayush Chanchal

Sap: 500097569

Batch: B6

EXPERIMENT: 07

**Perceptron for the and function:**

In this we will program a Neural Network in Python which implements the logical "And" function. It is defined for two inputs in the following way:

| **Input1** | **Input2** | **Output** |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

As we know that a neural network with one perceptron and two input values can be interpreted as a decision boundary, i.e. straight line dividing two classes. The two classes we want to classify in our example look like this:

import matplotlib.pyplot as plt

import numpy as np

fig, ax = plt.subplots()

xmin, xmax = -0.2, 1.4

X = np.arange(xmin, xmax, 0.1)

ax.scatter(0, 0, *color*="r")

ax.scatter(0, 1, *color*="r")

ax.scatter(1, 0, *color*="r")

ax.scatter(1, 1, *color*="g")

ax.set\_xlim([xmin, xmax])

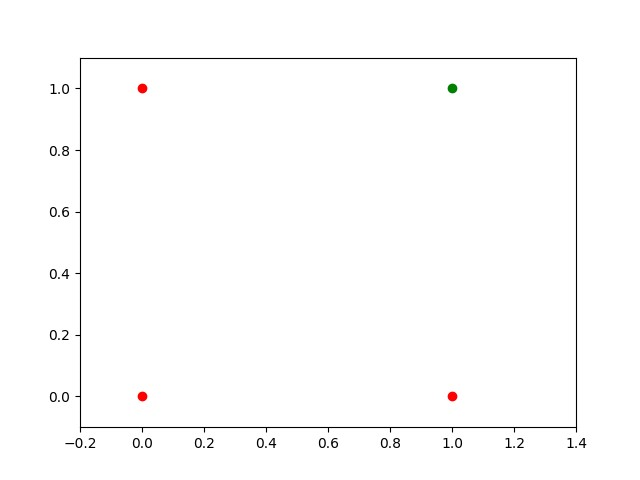
ax.set\_ylim([-0.1, 1.1])

m = -1

#ax.plot(X, m \* X + 1.2, label="decision boundary")

plt.plot()

plt.show()



We found out that such a primitive neural network is only capable of creating straight lines going through the origin.

So dividing line like this:

import matplotlib.pyplot as plt

import numpy as np

fig, ax = plt.subplots()

xmin, xmax = -0.2, 1.4

X = np.arange(xmin, xmax, 0.1)

ax.set\_xlim([xmin, xmax])

ax.set\_ylim([-0.1, 1.1])

m = -1

for m in np.arange(0, 6, 0.1):

    ax.plot(X, m \* X )

ax.scatter(0, 0, *color*="r")

ax.scatter(0, 1, *color*="r")

ax.scatter(1, 0, *color*="r")

ax.scatter(1, 1, *color*="g")

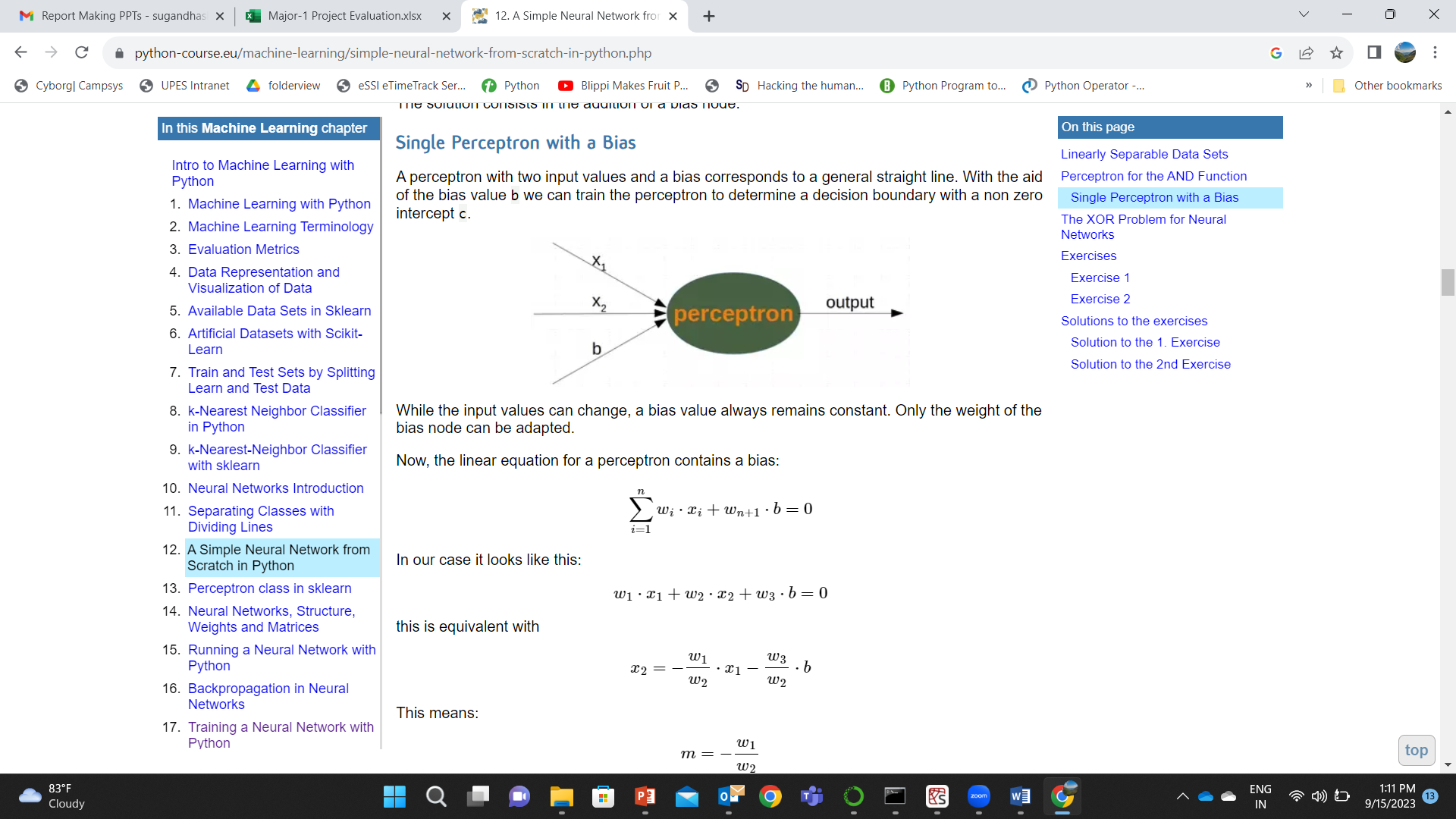
plt.plot()

plt.show()

### 

### **Single Perceptron with a Bias**

A perceptron with two input values and a bias corresponds to a general straight line. With the aid of the bias value b we can train the perceptron to determine a decision boundary with a non zero intercept c.



import numpy as np

from collections import Counter

class Perceptron:

    def \_\_init\_\_(*self*,

*weights*,

*bias*=1,

*learning\_rate*=0.3):

        """

        'weights' can be a numpy array, list or a tuple with the

        actual values of the weights. The number of input values

        is indirectly defined by the length of 'weights'

        """

*self*.weights = np.array(*weights*)

*self*.bias = *bias*

*self*.learning\_rate = *learning\_rate*

    @staticmethod

    def unit\_step\_function(*x*):

        if  *x* <= 0:

            return 0

        else:

            return 1

    def \_\_call\_\_(*self*, *in\_data*):

*in\_data* = np.concatenate( (*in\_data*, [*self*.bias]) )

        result = *self*.weights @ *in\_data*

        return Perceptron.unit\_step\_function(result)

    def adjust(*self*,

*target\_result*,

*in\_data*):

        if type(*in\_data*) != np.ndarray:

*in\_data* = np.array(*in\_data*)  #

        calculated\_result = *self*(*in\_data*)

        error = *target\_result* - calculated\_result

        if error != 0:

*in\_data* = np.concatenate( (*in\_data*, [*self*.bias]) )

            correction = error \* *in\_data* \* *self*.learning\_rate

*self*.weights += correction

    def evaluate(*self*, *data*, *labels*):

        evaluation = Counter()

        for sample, label in zip(*data*, *labels*):

            result = *self*(sample) # predict

            if result == label:

                evaluation["correct"] += 1

            else:

                evaluation["wrong"] += 1

        return evaluation

import numpy as np

from perceptron import \*

def labelled\_samples(*n*):

    for \_ in range(*n*):

        s = np.random.randint(0, 2, (2,))

        yield (s, 1) if s[0] == 1 and s[1] == 1 else (s, 0)

p = Perceptron(*weights*=[0.3, 0.3, 0.3],

*learning\_rate*=0.2)

for in\_data, label in labelled\_samples(30):

    p.adjust(label,

             in\_data)

test\_data, test\_labels = list(zip(\*labelled\_samples(30)))

evaluation = p.evaluate(test\_data, test\_labels)

print(evaluation)

mport matplotlib.pyplot as plt

import numpy as np

fig, ax = plt.subplots()

xmin, xmax = -0.2, 1.4

X = np.arange(xmin, xmax, 0.1)

ax.scatter(0, 0, *color*="r")

ax.scatter(0, 1, *color*="r")

ax.scatter(1, 0, *color*="r")

ax.scatter(1, 1, *color*="g")

ax.set\_xlim([xmin, xmax])

ax.set\_ylim([-0.1, 1.1])

m = -p.weights[0] / p.weights[1]

c = -p.weights[2] / p.weights[1]

print(m, c)

ax.plot(X, m \* X + c )

plt.plot()

plt.show()

Output:

Counter({'correct': 30})

-0.33333333333333326 1.0000000000000002

