Basic Perceptron

Build a neuron with Bias

Try to add bias in the programme and perform your programme in any binary classification data

Understand the given code and explain your code after adding bias. Try to show data representation code and explain how bias helps to neuron.

1. **Introduction Of basic Neuron**

   When we say Neural Networks, we mean artificial Neural Networks (ANN). The idea of ANN is based on biological neural networks like the brain.

   The basic structure of a neural network is the neuron. A neuron in biology consists of three major parts: the soma (cell body), the dendrites, and the axon.

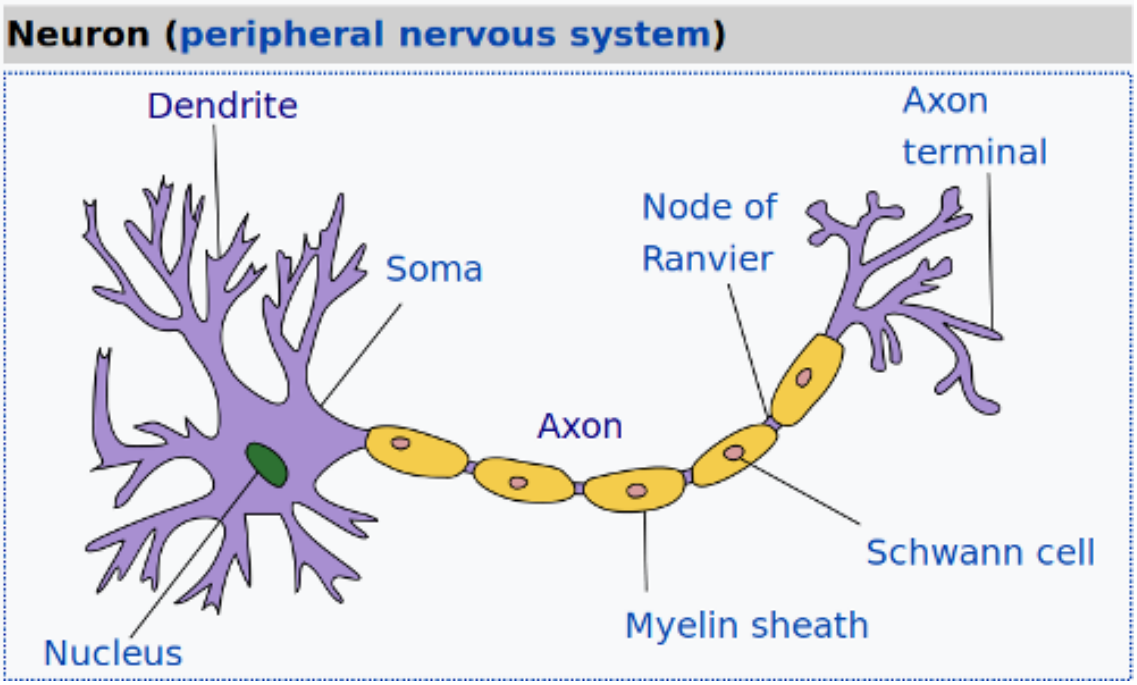

Figure 1: Biological Neuron

The dendrites branch of from the soma in a tree-like way and getting thinner with every branch. They receive signals (impulses) from other neurons at synapses. The axon - there is always only one - also leaves the soma and usually tend to extend for longer distances than the dentrites. The axon is used for sending the output of the neuron to other neurons or better to the synapsis of other neurons.

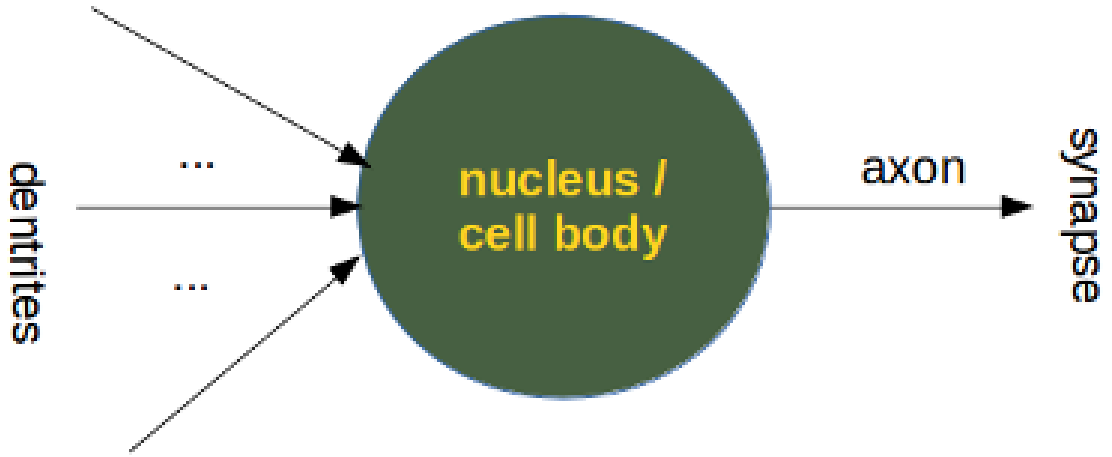

Figure 2: Graphical representation of biological neuron

A perceptron of artificial neural networks is simulating a biological neuron.

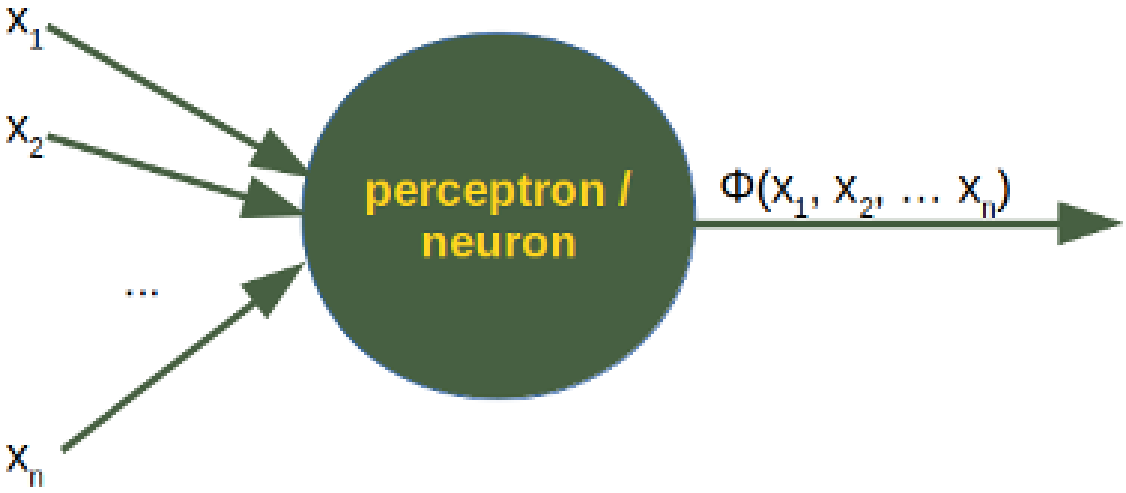

Figure 3: Mathematical process of simple neuron

When a signal comes in, it gets multiplied by a weight value that is assigned to this particular input. That is, if a neuron has three inputs, then it has three weights that can be adjusted individually. The weights usually get adjusted during the learn phase.

After this the modified input signals are summed up. It is also possible to add additionally a so-called bias b to this sum. The bias is a value which can also be adjusted during the learn phase.

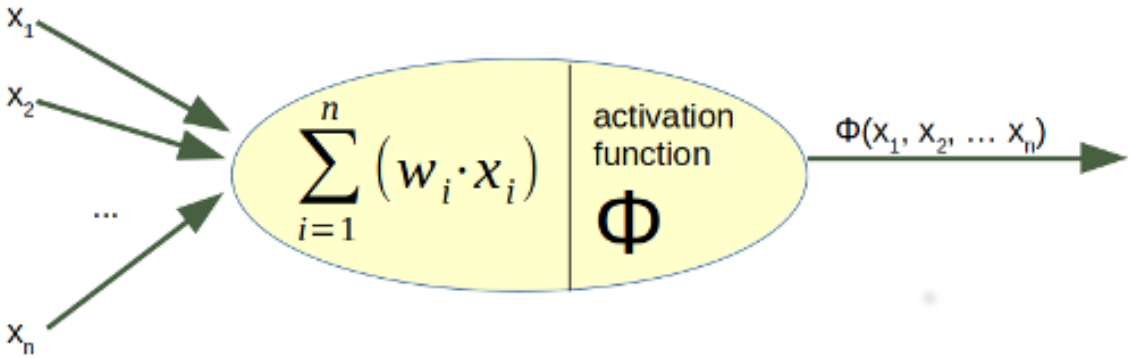

Figure 4: Perceptron

Finally, the actual output has to be determined. For this purpose an activation or step function ϕ is applied to weighted sum of the input values.

2. **A simple neural network implementing AND function:**
You could imagine that you have two attributes describing am eddible object like a fruit for example: sweetness and sourness

We could describe this by points in a two-dimensional space. The x axis for the sweetness and the y axis for the sourness. Imagine now that we have two fruits as points in this space, i.e. an orange at position (3.5, 1.8) and a lemon at (1.1, 3.9).

We could define dividing lines to define the points which are more lemon-like and which are more orange-like. The following program calculates and renders a bunch of lines. The red ones are completely unusable for this purpose, because they are not separating the classes. Yet, it is obvious that even the green ones are not all useful.

Listing 1: Plotting lines Python script!

```
import numpy as np
import matplotlib.pyplot as plt


def create_distance_function(a, b, c):
    """ 0 = ax + by + c """
    def distance(x, y):
        """ returns tuple (d, pos)
            d is the distance
            If pos == -1 point is below the line,
            0 on the line and +1 if above the line
        """
        nom = a * x + b * y + c
        if nom == 0:
            pos = 0
        elif (nom<0 and b<0) or (nom>0 and b>0):
            pos = -1
```

```
        else:
            pos = 1
        return (np.absolute(nom) / np.sqrt( a ** 2 + b ** 2), pos)
    return distance


points = [ (3.5, 1.8), (1.1, 3.9) ]

fig, ax = plt.subplots()
ax.set_xlabel("sweetness")
ax.set_ylabel("sourness")
ax.set_xlim([-1, 6])
ax.set_ylim([-1, 8])
X = np.arange(-0.5, 5, 0.1)


colors = ["r", ""] # for the samples

size = 10
for (index, (x, y)) in enumerate(points):
    if index== 0:
        ax.plot(x, y, "o",
                color="darkorange",
                markersize=size)
    else:
        ax.plot(x, y, "oy",
                markersize=size)

step = 0.05
for x in np.arange(0, 1+step, step):
    slope = np.tan(np.arccos(x))
    dist4line1 = create_distance_function(slope, -1, 0)
    #print("x: ", x, "slope: ", slope)
    Y = slope * X

    results = []
    for point in points:
        results.append(dist4line1(*point))
    #print(slope, results)
    if (results[0][1] != results[1][1]):
        ax.plot(X, Y, "g-")
    else:
        ax.plot(X, Y, "r-")


plt.show()
```

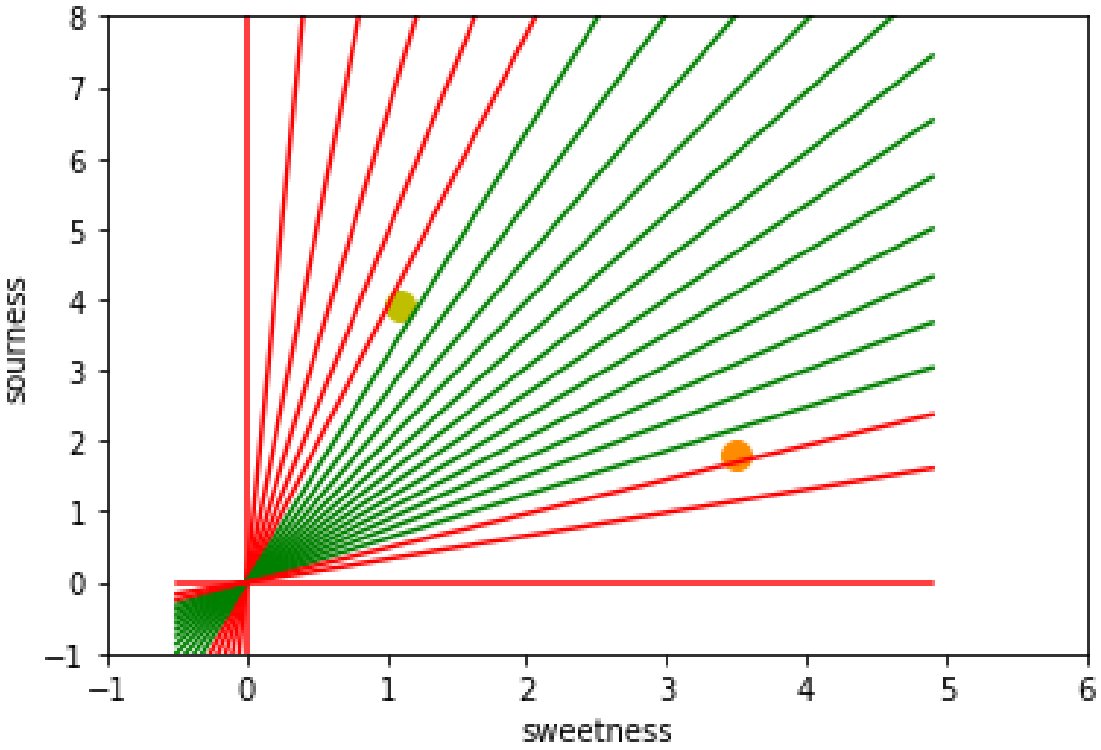

Figure 5: Neural Network With out Bias

In the following program, we train a neural network to classify two clusters in a 2-dimensional space. We show this in the following diagram with the two classes class1 and class2. We will create those points randomly with the help of a line, the points of class2 will be above the line and the points of class1 will be below the line.

Listing 2: Perceptron Python script!

```
import numpy as np

class Perceptron:

    def __init__(self, input_length, weights=None):
        if weights is None:
            self.weights = np.ones(input_length) * 0.5
        else:
            self.weights = weights

    @staticmethod
    def unit_step_function(x):
        if x > 0.5:
            return 1
        return 0
```

```
    def __call__(self, in_data):
        weighted_input = self.weights * in_data
        weighted_sum = weighted_input.sum()
        return Perceptron.unit_step_function(weighted_sum)

p = Perceptron(2, np.array([0.5, 0.5]))

data_in = np.empty((2,))
for in1 in range(2):
    for in2 in range(2):
        data_in = (in1, in2)
        data_out = p(data_in)
        print(data_in, data_out)
```

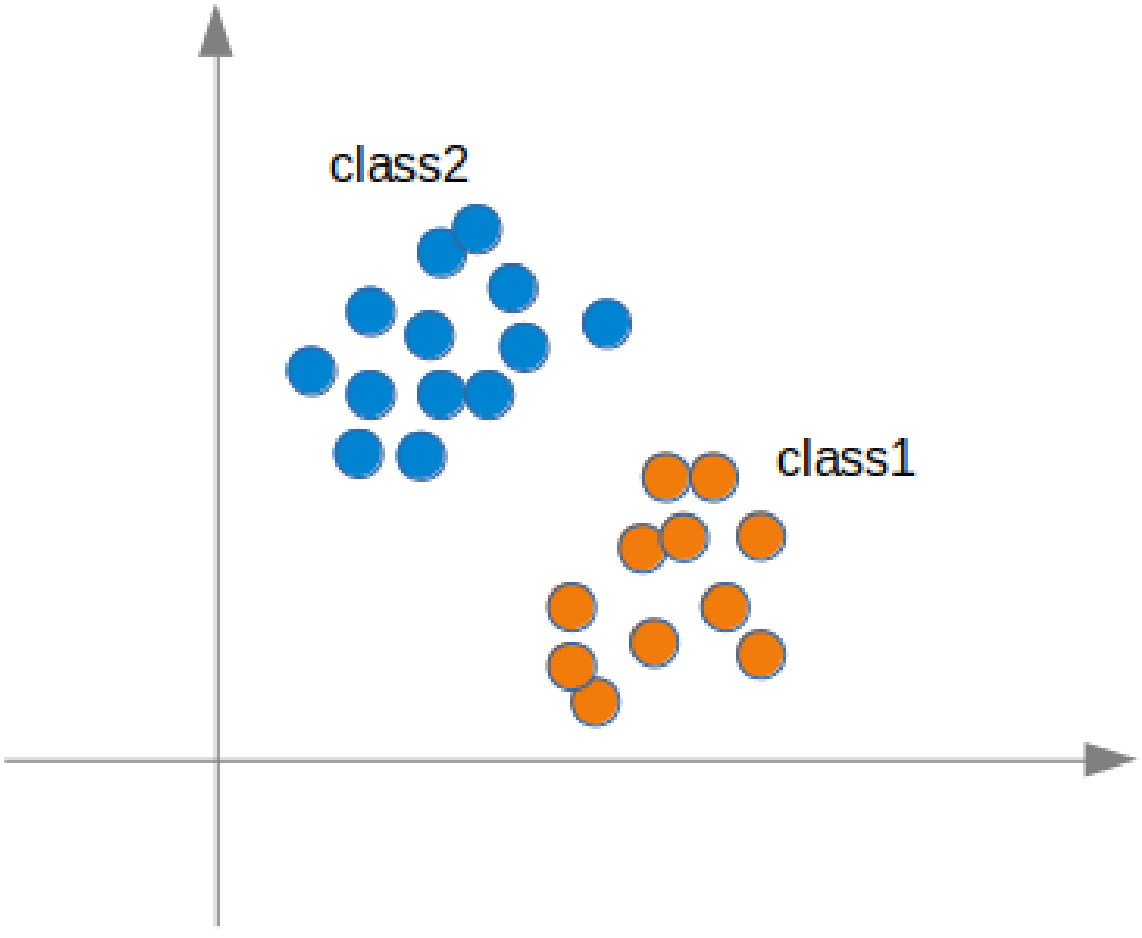

Figure 6: Data divided in classes

We will see that the neural network will find a line that separates the two classes. This line should not be mistaken for the line, which we used to create the points.

This line is called a decision boundary.

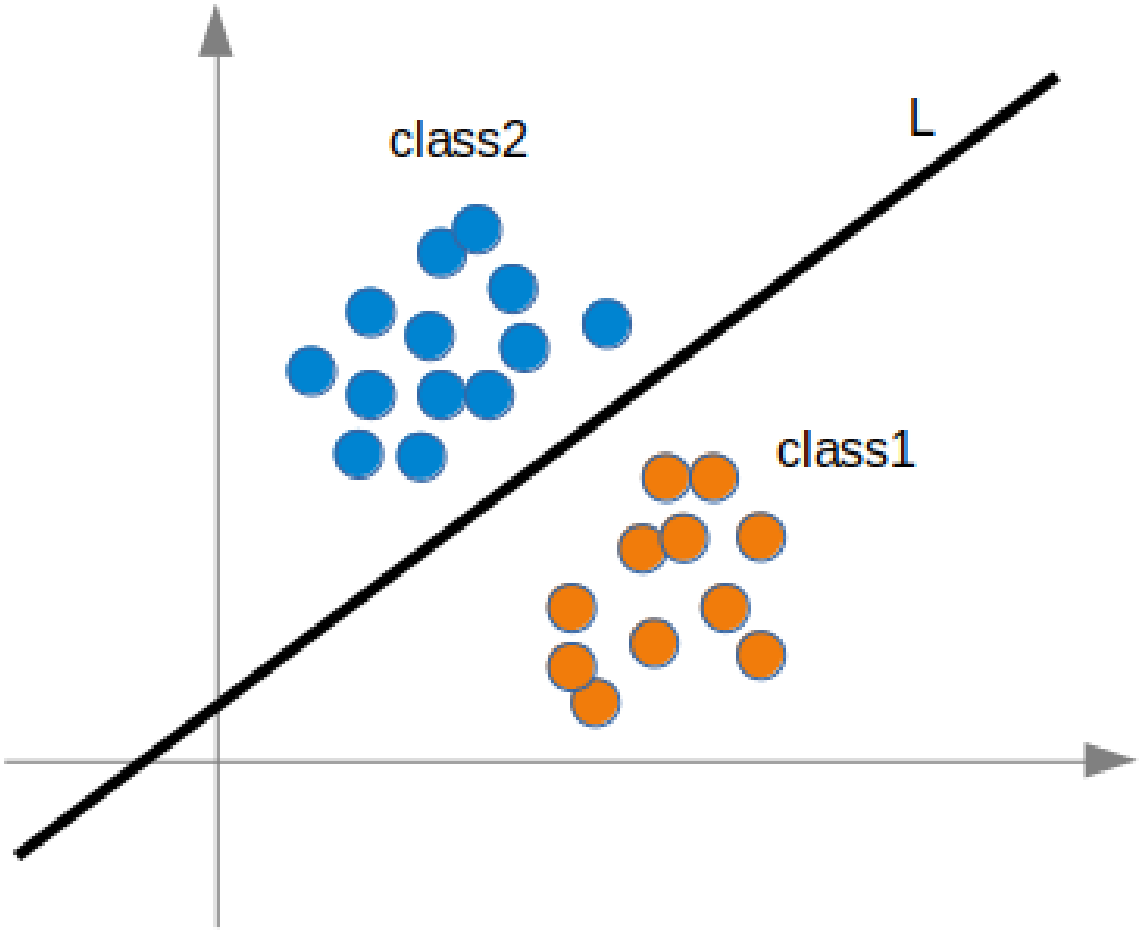

Figure 7: Line identification through Neural Network