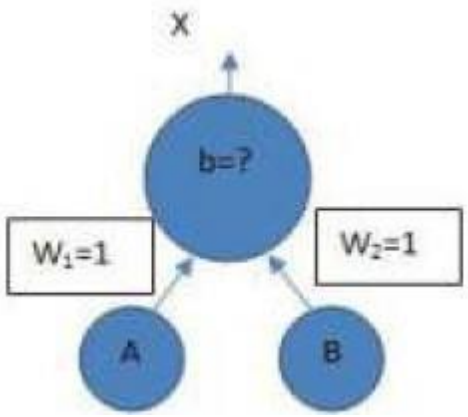
Assignment No: 02

Aim: Write a program to find the Boolean function to implement following single layer perceptron. Assume all activation functions to be the threshold function which is 1 for all input values greater than zero and 0, otherwise



Objectives:

1. To learn single layer perceptron.
2. To learn Boolean logic implementation using perceptron.

Software Requirements:

Ubuntu 18.04

Hardware Requirements:

Pentium IV system with latest configuration

Theory:

The most common Boolean functions are AND, OR, NOT. The Boolean logic AND only returns 1 if both inputs are 1 else 0, Boolean logic OR returns 1 for all inputs with 1, and will only return 0 if both input is 0 and lastly logic NOT returns the invert of the input, if the input is 0 it returns 1, if the input is 1 it returns 0. To make it clear the image below shows the truth table for the basic Boolean Function.

A	B	Z
0	0	0
0	1	0
1	0	0
1	1	1

AND

A	B	Z
0	0	0
0	1	1
1	0	1
1	1	1

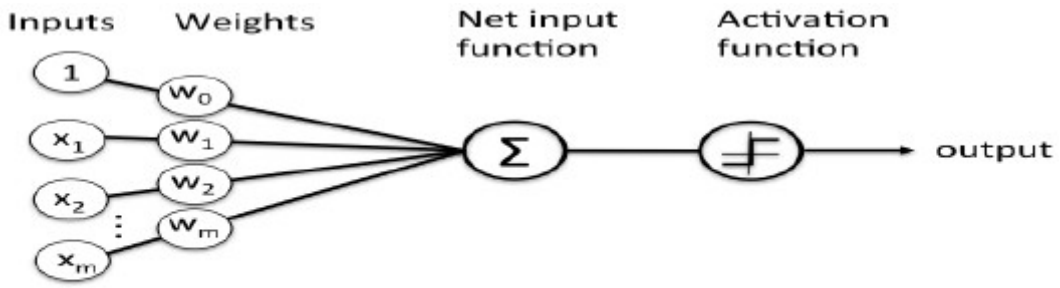
OR

A	Z
0	1
1	0

NOT

The columns A and B are the inputs and column Z is the output. So, for the inputs A = 0, B = 0 the output is Z = 0.

Perceptron



Schematic of Rosenblatt's perceptron.

A perceptron is the basic part of a neural network. A perceptron represents a single neuron on a human's brain, it is composed of the dataset (Xm) , the weights (Wm) and an activation function, that will then produce an output and a bias. The datasets (inputs) are converted into an ndarray which is then matrix multiplied to another ndarray that holds the weights. Summing up all matrix multipy and adding a bias will create the net input function, the output would then passed into an activation function that would determine if the neuron needs to fire an output or not.

Most common activation function used for classification used is a sigmoid function, which is a great function for classification (Although sigmoid is not the leading activation function for middle layers of neural networks [ehem ReLU / Leaky ReLU] it still is widely used for final classifications.)

The perceptron is simply separating the input into 2 categories, those that cause a fire, and those that don't. It does this by looking at (in the 2-dimensional case):

w1I1 + w2I2 < t

If the LHS is < t, it doesn't fire, otherwise it fires. That is, it is drawing the line:

w1I1 + w2I2 = t

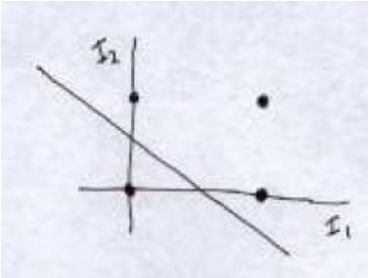
and looking at where the input point lies. Points on one side of the line fall into 1 category, points on the other side fall into the other category. And because the weights and thresholds can be anything, this is just *any line* across the 2 dimensional input space.

So what the perceptron is doing is simply drawing a line across the 2-d input space. Inputs to one side of the line are classified into one category, inputs on the other side are classified into another.

e.g. the OR perceptron, w1=1, w2=1, t=0.5, draws the line:

I1 + I2 = 0.5

across the input space, thus separating the points (0,1),(1,0),(1,1) from the point (0,0):



As you might imagine, not every set of points can be divided by a line like this. Those that can be, are called *linearly separable*.

In 2 input dimensions, we draw a 1 dimensional line. In n dimensions, we are drawing the (n-1) dimensional *hyperplane*:

w1I1 + .. + wnIn = t

Perceptron Learning Algorithm

Algorithm: Perceptron Learning Algorithm

```
P ← inputs with label 1;
N ← inputs with label 0;
Initialize w randomly;
while !convergence do
    Pick random x ∈ P ∪ N ;
    if x ∈ P and w.x < 0 then
        w = w + x ;
    end
    if x ∈ N and w.x ≥ 0 then
        w = w − x ;
    end
end
//the algorithm converges when all the
 inputs are classified correctly
```

We initialize **w** with some random vector. We then iterate over all the examples in the data, (*P* U *N*) both positive and negative examples. Now if an input **x** belongs to *P*, ideally what should the dot product **w.x** be? It would be greater than or equal to 0 because that's the only thing what our perceptron wants at the end of the day so let's give it that. And if **x** belongs to *N*, the dot product MUST be less than 0. So if you look at the if conditions in the while loop:

```
while !convergence do
    Pick random x ∈ P ∪ N ;
    if x ∈ P   and   w.x < 0 then
        w = w + x ;
    end
    if x ∈ N   and   w.x ≥ 0 then
        w = w − x ;
    end
end
```

Case 1: When **x** belongs to *P* and its dot product **w.x** < 0

Case 2: When **x** belongs to *N* and its dot product **w.x** ≥ 0

Only for these cases, we are updating our randomly initialized **w**. Otherwise, we don't touch **w** at all because Case 1 and Case 2 are violating the very rule of a perceptron. So we are adding **x** to **w** (ahem vector addition ahem) in Case 1 and subtracting **x** from **w** in Case 2.

Conclusion

Thus we learned implementation of single layer perceptron for AND , OR and NOT Boolean function.