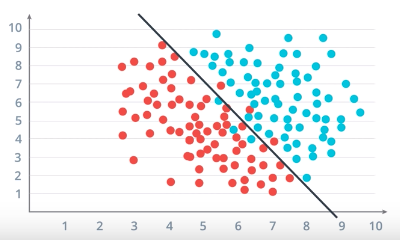
**Perceptron**

A perceptron is like a single processing unit which can be used to form a linear classification/decision boundary between different classes.



**Figure: Linear classification using perceptron**

A perceptron is responsible to find a line of the form:

WX + b = 0

where

* **X**is dimension size vector which forms the variables (eg. X = [x1, x2]),
* **W**is the coefficient vector corresponding to each X (eg. W = [w1, w2]),
* **b**is a scalar bias value added to the line equation.

So, the equation for a 2-dimensional data point would be:

WX + b = 0

where,   
W = {w1, w2}  
X = {x1, x2}  
=> **(w1 \* x1) + (w2 \* x2) + b = 0**

In a binary classifier, a data point is classified as either positive(blue)/negative(red) data point.

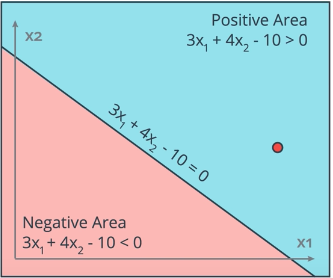
* Positive(Blue) data point if WX + b ≥ 0
* Negative(Red) data point if WX + b < 0

To find the value of WX + b, we simply replace the X values in the equation with the data point coordinates.

We use something called as Activation Function (here we can use ***Step Function***) to give the Blue points a label 1 and Red points a label 0.

def step\_function(X):  
 if (WX + b) >= 0:  
 return 1 # Positive label  
 else:  
 return 0 # Negative label

The question here is to know how to find the line of the equation i.e. **the weights (W) and the Bias (b)** such that the blue and red points are separated in such a way that maximum Red points are below the line and maximum Blue are above.



**Figure: An example scenario**

Red point is in the Positive (Blue) Region

In the above image the Negative point (Red) is in the Positive Region for our given line

3x1 + 4x2-10 = 0

Our objective is to somehow shift the line such that the red point comes below the new updated line, i.e., we need to bring the line closer to the Red point.

A neat trick for this is to do the following, suppose the red point has the coordinates (4,5):

1. For negative point in Positive region (i.e. WX + b ≥ 0):

* Take the line coefficients and subtract the misclassified point coordinate values from it.
* Also, subtract 1 from the bias value.

2. For positive point in Negative region (i.e. WX + b < 0):

* This way we shift the line towards the misclassified data point.
* For example, here we’ll do:

Old\_W: 3 4 -10

Coords: -4 -5 -1 <- subtracting 1 from Bias as well  
 ------------------  
New\_W = -1 -1 -11

But, there is a slight issue with this approach. This sudden change in coefficient values could lead to misclassifying many other points in the data set.

To avoid this issue, we update the weights slowly by introducing **learning rate.**

Learning rate ensures that the weights are updated slowly and hence there is a gradual shift of the line towards the desired region.

We multiple the learning rate with each of the coordinate values as well as the bias value.

In our previous example if learning rate = 0.1:

Old\_W: 3.0 4.0 -10.0  
Coords \* learning rate: -0.4 -0.5 -0.1  
 -------------------------  
New\_W = 2.6 3.5 -9.9

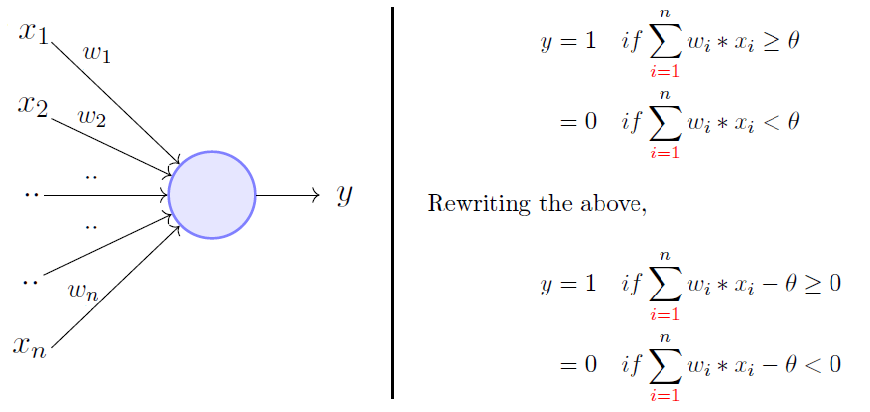
So, now our new line equation will be: (2.6 \* x1) + (3.5 \* x2) - 9.9 = 0

We do this process repeatedly to improve the weights for all the misclassified data points. The number of times we do this is known as the number of **epochs.**

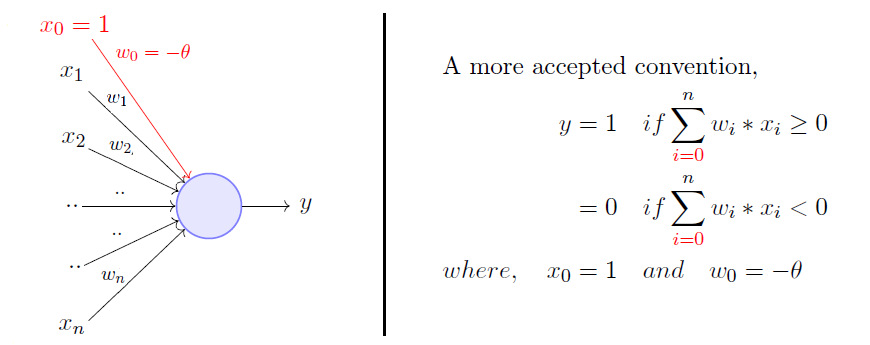
So our **Perceptron Algorithm** would be as follows *(α = learning rate)*:

1. Start with random weights (W) and bias (b)  
2. For every misclassified point (p1, p2, …, pn):  
 2.1 if Red point (i.e. prediction = 1 WX+b ≥ 0 but should be < 0)  
 For i <- 1 to n  
 update wi <- wi - α \* xi  
 update b <- b - α 2.2 if Blue point (i.e. prediction = 0 WX+b <0 but should be >=0)  
 For i <- 1 to n  
 update wi <- wi + α \* xi  
 update b <- b + α  
3. end

# **Perceptron in logic gate**



Rewriting the threshold as shown above and making it a constant input with a variable weight, we would end up with something like the following:



## 

## AND Gate

Initialization: w1=w2=1, b=-1

The question is, what are the weights and bias for the OR perceptron?

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
| |  | | --- | |  | | x1 | x2 | w1 | w2 | WX | b | WX+b | Output | Comments |
|  | 0 | 0 | 1 | 1 | 0 | -1 | -1 | 0 | This is valid |
|  | 0 | 1 | 1 | 1 | 1 | -1 | 0 | 0 | This is valid |
|  | 1 | 0 | 1 | 1 | 1 | -1 | 0 | 0 | This is valid |
|  | 1 | 1 | 1 | 1 | 2 | -1 | 1 | 1 | This is valid |

Thus the w1=w2=1, b=-1

Assignment: Solve it for three input AND Gate

## OR Gate

Initialization: w1=w2=1, b=-1

The question is, what are the weights and bias for the OR perceptron?

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x1 | x2 | w1 | w2 | WX | b | WX-b | Output | Comments |
| |  | | --- | |  | | 0 | 0 | 1 | 1 | 0 | -1 | -1 | 0 | Row 1 is valid |
|  | 0 | 1 | 1 | 1 | 1 | -1 | 0 | 0 | Row 2 is not valid |
|  | 0 | 1 | 1 | 2 | 2 | -1 | 1 | 1 | Row 2 is valid |
|  | 1 | 0 | 1 | 2 | 1 | -1 | 0 | 0 | Row 3 is not valid |
|  | 1 | 0 | 2 | 2 | 2 | -1 | 1 | 1 | Row 3 is valid |
|  | 1 | 1 | 2 | 2 | 4 | -1 | 3 | 1 | Row 4 is valid |
|  |  |  |  |  |  |  |  |  |  |
|  | x1 | x2 | w1 | w2 | WX | b | WX-b | Output | Comments |
|  | 0 | 0 | **2** | **2** | 0 | -1 | -1 | 0 | Row 1 is valid |
|  | 0 | 1 | **2** | **2** | 2 | -1 | 1 | 1 | Row 2 is valid |
|  | 1 | 0 | **2** | **2** | 2 | -1 | 1 | 1 | Row 3 is valid |
|  | 1 | 1 | **2** | **2** | 4 | -1 | 3 | 1 | Row 4 is valid |

Thus, the w1=w2=2, b=-1

Assignment: Solve it for three input OR Gate

**NOT Gate**

Initialization: w1=1, b=-1

The question is, what are the weights and bias for the NOT perceptron?

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | |  | | x1 | w1 | WX | b | WX-b | Ouput | Comments |  |  |
|  | 0 | 1 | 0 | -1 | -1 | 0 | R1 Invalid |  |  |
|  | 0 | 1 | 0 | 1 | 1 | 1 | R1 Valid |  |  |
|  | 1 | 1 | 1 | 1 | 2 | 1 | R2 Invalid |  |  |
|  | 1 | -1 | -1 | 1 | 0 | 0 | R2 Valid |  |  |
|  | Thus |  |  |  |  |  |  |  |  |
|  | x1 | w1 | WX | b | WX-b | Ouput | Comments |  |  |
|  | 0 | **-1** | 0 | **1** | 1 | 1 | R1 Valid |  |  |
|  | 1 | **-1** | -1 | **1** | 0 | 0 | R2 Valid |  |  |

From the table, w1=-1, b=1

**NOR**

Initialization: w1=w2=1, b=-1

The question is, what are the weights and bias for the NOR perceptron?

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | |  | | x1 | x2 | w1 | w2 | WX | b | WX-b | Output | Comments |
|  | 0 | 0 | 1 | 1 | 0 | -1 | -1 | 0 | R1 Invalid |
|  | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | R1 Valid |
|  | 0 | 1 | 1 | 1 | 1 | 1 | 2 | 1 | R2 Invalid |
|  | 0 | 1 | 1 | -1 | -1 | 1 | 0 | 0 | R2 Valid |
|  | 1 | 0 | 1 | -1 | 1 | 1 | 2 | 1 | R3 Invalid |
|  | 1 | 0 | -1 | -1 | -1 | 1 | 0 | 0 | R3 Valid |
|  | 1 | 1 | -1 | -1 | -2 | 1 | -1 | 0 | R4 Valid |
|  | Thus |  |  |  |  |  |  |  |  |
|  | x1 | x2 | w1 | w2 | WX | b | WX-b | Output | Comments |
|  | 0 | 0 | **-1** | **-1** | 0 | **1** | 1 | 1 | R1 Valid |
|  | 0 | 1 | **-1** | **-1** | -1 | **1** | 0 | 0 | R2 Valid |
|  | 1 | 0 | **-1** | **-1** | -1 | **1** | 0 | 0 | R3 Valid |
|  | 1 | 1 | **-1** | **-1** | -2 | **1** | -1 | 0 | R4 Valid |

**NAND**

Initialization: w1=w2=1, b=-1

The question is, what are the weights and bias for the NAND perceptron?

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| NAND |  |  |  |  |  |  |  |  |  |
| |  | | --- | |  | | x1 | x2 | w1 | w2 | WX | b | WX-b | Output | Comments |
|  | 0 | 0 | 1 | 1 | 0 | -1 | -1 | 0 | R1 Invalid |
|  | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | R1 Valid |
|  | 0 | 1 | 1 | 1 | 1 | 1 | 2 | 1 | R2 Valid |
|  | 1 | 0 | 1 | 1 | 1 | 1 | 2 | 1 | R3 Valid |
|  | 1 | 1 | 1 | 1 | 2 | 1 | 3 | 1 | R4 Invalid |
|  | 1 | 1 | -1 | -1 | -2 | 1 | -1 | 0 | R4 Valid |
|  |  |  |  |  |  |  |  |  |  |
|  | x1 | x2 | w1 | w2 | WX | b | WX-b | Output | Comments |
|  | 0 | 0 | -1 | -1 | 0 | **1** | 1 | 1 | R1 Valid |
|  | 0 | 1 | -1 | -1 | -1 | **1** | 0 | 0 | R2 Invalid |
|  | 1 | 0 | -1 | -1 | -1 | **1** | 0 | 0 | R3 Invalid |
|  | 1 | 1 | -1 | -1 | -2 | **1** | -1 | 0 | R4 Valid |
|  |  |  |  |  |  |  |  |  |  |
|  | x1 | x2 | w1 | w2 | WX | b | WX-b | Output | Comments |
|  | 0 | 0 | -1 | -1 | 0 | **2** | 2 | 1 | R1 Valid |
|  | 0 | 1 | -1 | -1 | -1 | **2** | 1 | 1 | R2 Valid |
|  | 1 | 0 | -1 | -1 | -1 | **2** | 1 | 1 | R3 Valid |
|  | 1 | 1 | -1 | -1 | -2 | **2** | 0 | 0 | R4 Valid |

Thus, the w1=w2=-1, b=2

**X-OR**

The boolean representation of an XOR gate is;

We first simplify the boolean expression

Thus, the XOR gate consists of

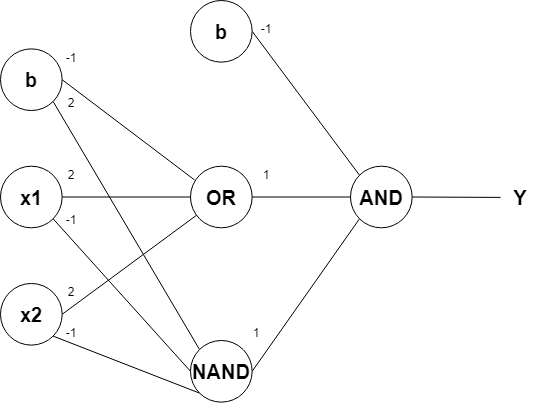
* an OR gate
* a NAND gate
* an AND gate for

Assignment

Find the value of W1, W2 and b

Hints

* OR (2x1+2x2–1)
* NAND (-x1-x2+2)
* AND (x1+x2–1)



Python exercise:

<https://towardsdatascience.com/perceptrons-logical-functions-and-the-xor-problem-37ca5025790a>