# The perceptron trick:

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

![](https://miro.medium.com/v2/resize:fit:500/1\*WFKqno6dAKNO6Kms3wqlcA.png)

Linear Differentiating Boundary b/w Red & Blue data points

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.

## Here comes the use of Perceptron Trick!

![](https://miro.medium.com/v2/resize:fit:416/1\*Eu0hZT4erUBa4ygUkrBrsQ.png)

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 `(3 \* x1) + (4 \* x2) — 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.

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

- Take the line coefficients and add the misclassified point coordinate values into it.

- Also, add 1 to the bias value.

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.\*\*