# Perceptron learning rule

Today we are going to see what is the perceptron learning rule and how to implement an algorithm that trains a perceptron to do a binary classification.

However, firstly we need to state that this perceptron learning rule holds for those perceptrons that have the following activation function:

Activation(**w**·**x**) = +1 if **w**·**x** ≥ 0

-1 if **w**·**x** < 0

Let’s remind ourself that **w** is the weight vector (w0, w1, …, wn) and **x** is the feature vector (1, x1,…, xn).

The first 1 in the feature vector **x** is fundamental as |**w**| must be equal to | **x** | and so we need to add a 1 at the beginning since **w** contains w0.

The perceptron learning algorithm is the following and uses the perceptron learning rule so that to update the weight vector **w**. Indeed, the learning part when using perceptrons is to find the vector **w** so that it produces a line which perfectly separates the two classifications.

Repeat until changed? == true:

changed? <- false

For all datapoints (**x**n, yn) in the dataset:

If Activation(**w**·**x**n) = yn then:

Skip

Else:

**w** <- **w** + αyn**x**n //perceptron learning rule

changed? <- true

Algorithm explanation:

This algorithm consists of passing through the whole data set many times until convergence. Convergence will occur when after passing through the whole data set the weight vector has not been changed. If the weight vector has not been changed then this means that the perceptron has guessed correctly all outputs and so it has learnt the binary classification.

Indeed, when the perceptron guesses a datapoint correctly then intuitively we do not need to alter the weight vector.

Contrarily, when the perceptron guesses wrong then we need to change the weight vector by using the perceptron learning rule which says that the new weight vector is equal to the old weight vector + (learning rate times the correct output times the feature vector).

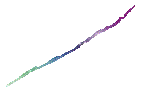
It is guaranteed that with a small learning rate then the algorithm will converge.

## Intuition behind the perceptron learning rule

Firstly, we need to understand why the weight vector **w** is perpedicular to the function **w·x** = 0.

This is trivial to see because every vector that lies on the line **w·x** = 0 will be a vector **v** such that **w**·v = 0 and so by definition of the dot product **w** must be perpendicular to v.

Thus the weight vector is perpedicular to the line (hyperplane) **w·x** = 0 as shown by the following representation:



Now, we update the weight vector **w** only when the perceptron guesses wrong for a particular data point.

Thus, two occurrences of errors can occur in a binary setting:

* Activation(**w**·**x**n) = +1 because **w**·**x**n ≥ 0 but yn = -1 and so **w**·**x**n should be < 0
* Activation(**w**·**x**n) = -1 because **w**·**x**n < 0 but yn = +1 and so **w**·**x**n should be ≥ 0

We know that the dot product **x**·**y** geometrically is equal to |**x**| × |y| × cos(Θ**xy**)

Given that the length of vectors is always positive then if the dot product is positive it means that the angle between the two vector is between 0° and 90° while if it is negative then it is between 90° and 180°

## First case

In the first case, the dot product **w**·**x**n ≥ 0 but it should be **w**·**x**n < 0. Therefore, currently the angle between **w** and **x**n is between 0° and 90° but it should be larger between 90° and 180°. Thus, we need to make the angle Θ**wxn**larger.

This is what it looks like:



We need to make it larger and so we need to make the weight vector **w** go distant from the feature vector **xn.**

We can do this by subtracting to **w x**n weighted by a constant. In other words, the constant we are talking about is the learning rate­­­­­­­­­­



As we can see after the update, the angle between **w** and **x**n is larger and in this case greater than 90° and so now **w**·**x**n would output the correct result -1.

The same thing holds for the other second case but in that case we need to make the angle between **w** and **x**n smaller. Consequently, we need to make **w** closer to **x**n and so we will sum α**xn** to **w**.

This is what the perceptron learning rule does and why the real value yn is in the formula as if yn= -1 then we need to subtract otherwise we need to sum.

## GOFAI vs Contemporary AI

In the past, in the AI field it was used to reasoning and learn in a symbolic way and so through symbols by using high-order logics. Nowadays, instead AI focuses on learning through the adjustment of vectors as we can see in the perceptron learning algorithm. Therefore, Hinton suggested that learning is just a sequence of states containing vector spaces.