towards
data science

Perceptron: The Artificial Neuron (An Essential Upgrade To The McCulloch-Pitts Neuron)

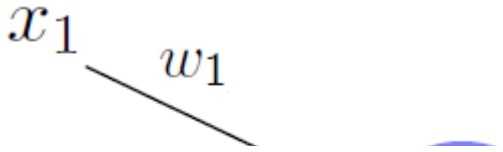
Akshay L Chandra Aug 11, 2018 · 7 min read

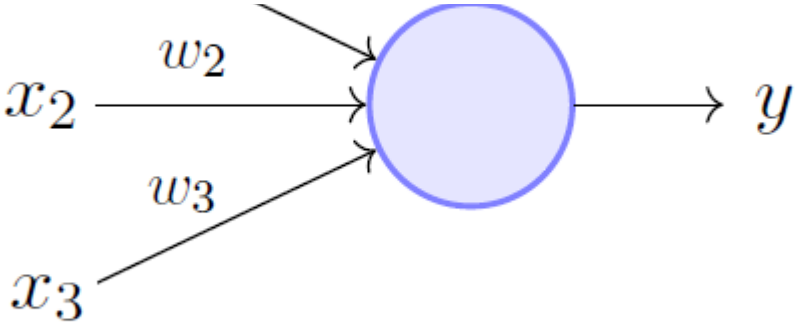
The most fundamental unit of a deep neural network is called an *artificial neuron*, which takes an input, processes it, passes it through an activation function like the Sigmoid, return the activated output. In this post, we are only going to talk about the *perceptron* model proposed before the 'activation' part came into the picture.

Frank Rosenblatt, an American psychologist, proposed the *classical perceptron* model in 1958. Further refined and carefully analyzed by Minsky and Papert (1969) — their model is referred to as the *perceptron* model. This is a follow-up post to my previous post on McCulloch-Pitts neuron, I suggest you at least quickly skim through it to better appreciate the Minsky-Papert contributions.

Citation Note: The concept, the content, and the structure of this article were inspired by the awesome lectures and the material offered by Prof. ***Mitesh M. Khapra*** *on NPTEL's Deep Learning course. Check it out!*

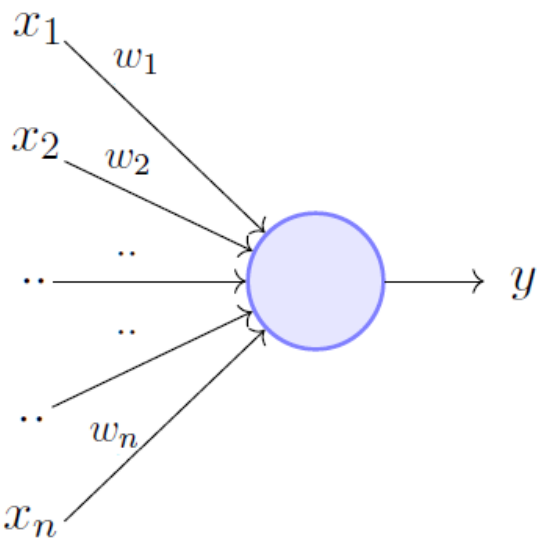
Perceptron





Perceptron Model (Minsky-Papert in 1969)

The *perceptron* model, proposed by Minsky-Papert, is a more general computational model than McCulloch-Pitts neuron. It overcomes some of the limitations of the M-P neuron by introducing the concept of numerical weights (a measure of importance) for inputs, and a mechanism for learning those weights. Inputs are no longer limited to boolean values like in the case of an M-P neuron, it supports real inputs as well which makes it more useful and generalized.



$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i \geq \theta$$

$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i < \theta$$

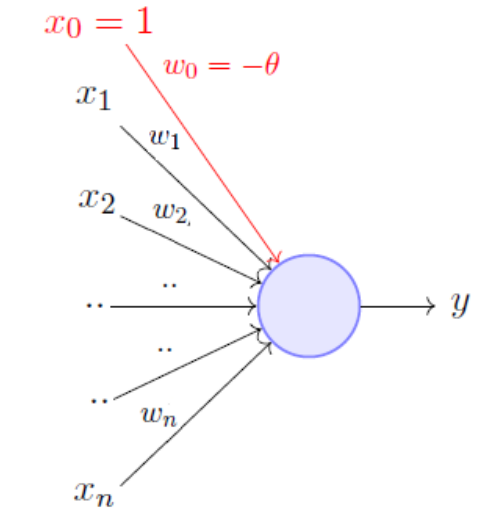
Rewriting the above,

$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i - \theta \geq 0$$

$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i - \theta < 0$$

Now, this is very similar to an M-P neuron but we take a weighted sum of the inputs and set the output as one only when the sum is more than an arbitrary threshold (***theta***). However, according to the convention, instead of hand coding the thresholding parameter ***thetha***, we add it as one of the inputs, with the weight -***theta*** like shown

below, which makes it learn-able (more on this in my next post — *Perceptron Learning Algorithm*).



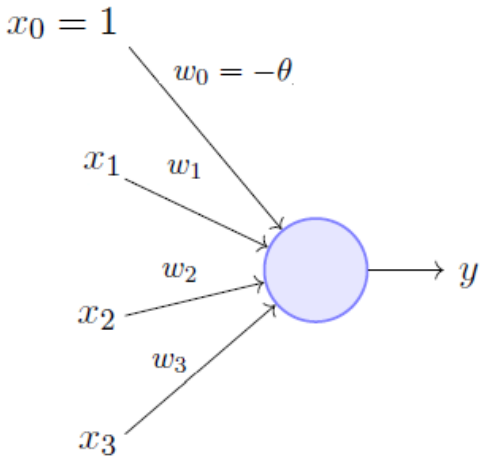
A more accepted convention,

$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$

$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$

$$where, \quad x_0 = 1 \quad and \quad w_0 = -\theta$$

Consider the task of predicting whether I would watch a random game of football on TV or not (the same example from my M-P neuron post) using the behavioral data available. And let's assume my decision is solely dependent on 3 binary inputs (binary for simplicity).



$$x_1 = isPremierLeagueOn$$
$$x_2 = isManUnitedPlaying$$
$$x_3 = isFriendlyGame$$

Here, ***w_0*** is called the bias because it represents the prior (prejudice). A football freak may have a very low threshold and may watch any football game irrespective of the league, club or importance of the game [***theta = 0***]. On the other hand, a selective viewer like me may only watch a football game that is a premier league game, featuring

Man United game and is not friendly [***theta = 2***]. The point is, the **weights** and the **bias** will depend on the data (my viewing history in this case).

Based on the data, if needed the model may have to give a lot of importance (high weight) to the *isManUnitedPlaying* input and penalize the weights of other inputs.

Perceptron vs McCulloch-Pitts Neuron

What kind of functions can be implemented using a *perceptron*? How different is it from McCulloch-Pitts neurons?

McCulloch Pitts Neuron
(assuming no inhibitory inputs)

$$y = 1 \quad if \sum_{i=0}^{n} x_i \geq 0$$

$$= 0 \quad if \sum_{i=0}^{n} x_i < 0$$

Perceptron

$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$

$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$

From the equations, it is clear that even a *perceptron* separates the input space into two halves, positive and negative. All the inputs that produce an output 1 lie on one side (positive half space) and all the inputs that produce an output 0 lie on the other side (negative half space).

In other words, a single *perceptron* can only be used to implement **linearly separable** functions, just like the M-P neuron. Then what is the difference? Why do we claim that the *perceptron* is an updated version of an M-P neuron? Here, the weights, including the threshold can be **learned** and the inputs can be **real** values.

Boolean Functions Using Perceptron

OR Function — Can Do!

Just revisiting the good old OR function the *perceptron* way.

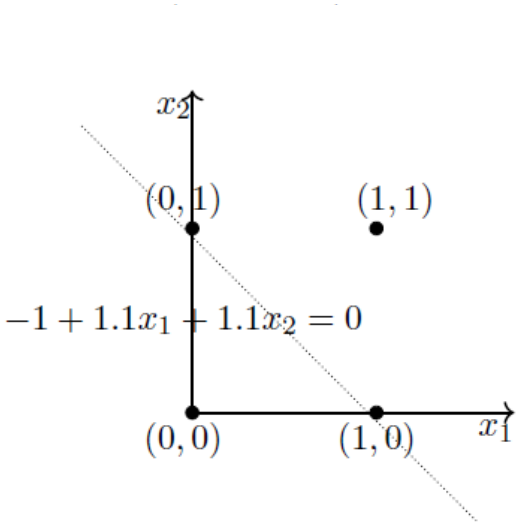
x_1	x_2	OR

One possible solution is

$w_0 = -1,\ w_1 = 1.1,\ w_2 = 1.1$

0	0	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$
1	0	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
0	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
1	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$

$$
\begin{aligned}
w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 &\implies w_0 < 0 \\
w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 &\implies w_2 > -w_0 \\
w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 &\implies w_1 > -w_0 \\
w_0 + w_1 \cdot 1 + w_2 \cdot 1 \geq 0 &\implies w_1 + w_2 > -w_0
\end{aligned}
$$



Try solving the equations on your own.

The above 'possible solution' was obtained by solving the linear system of equations on the left. It is clear that the solution separates the input space into two spaces, negative and positive half spaces. I encourage you to try it out for AND and other boolean function.

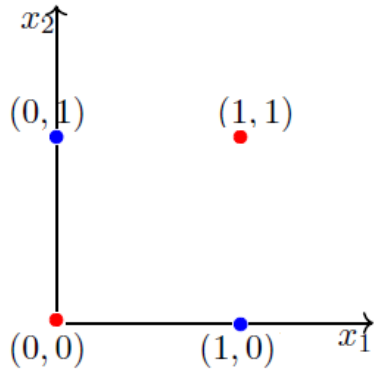
Now if you actually try and solve the linear equations above, you will realize that there can be multiple solutions. But which solution is the best? To more formally define the 'best' solution, we need to understand errors and error surfaces, which we will do in my next post on *Perceptron Learning Algorithm*.

XOR Function — Can't Do!

Now let's look at a non-linear boolean function i.e., you cannot draw a line to separate positive inputs from the negative ones.

x_1	x_2	XOR	
0	0	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$
1	0	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
0	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
1	1	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$

$$
\begin{aligned}
w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 &\implies w_0 < 0 \\
w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 &\implies w_2 > -w_0 \\
w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 &\implies w_1 > -w_0 \\
w_0 + w_1 \cdot 1 + w_2 \cdot 1 \geq 0 &\implies w_1 + w_2 < -w_0
\end{aligned}
$$

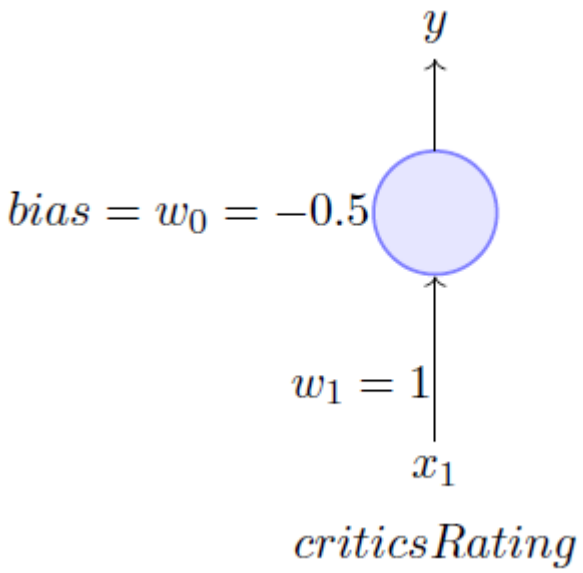


Notice that the fourth equation contradicts the second and the third equation. Point is, there are no *perceptron* solutions for non-linearly separated data. So the key take away is that a **single** *perceptron* cannot learn to separate the data that are non-linear in nature.

> ***The XOR Affair***
>
> *In the book published by Minsky and Papert in 1969, the authors implied that, since a single artificial neuron is incapable of implementing some functions such as the XOR logical function, larger networks also have similar limitations, and therefore should be dropped. Later research on three-layered perceptrons showed how to implement such functions, therefore saving the technique from obliteration.*
>
> — Wikipedia

(Optional) Motivation For Sigmoid Neurons

As I mentioned earlier, the artificial neurons we use today are slightly different from the *perceptron* we looked at, the difference is the activation function. here. Some might say that the thresholding logic used by a *perceptron* is very harsh. For example, if you look at a problem of deciding if I will be watching a movie or not, based only on one real-valued input (***x_1*** = *criticsRating*) and if the threshold we set is 0.5 (***w_0*** = -0.5) and ***w_1*** = 1 then our setup would look like this:

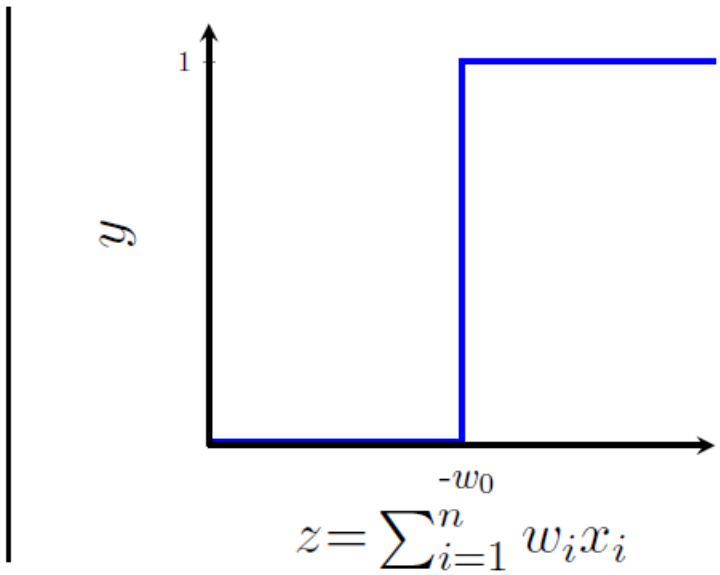


What would be the decision for a movie with *criticsRating* = 0.51? *Yes!*
What would be the decision for a movie with *criticsRating* = 0.49? *No!*

Some might say that its harsh that we would watch a movie with a rating of 0.51 but not the one with a rating of 0.49 and this is where Sigmoid comes into the picture. Now convince yourself that this harsh thresholding is not attributed to just one specific problem we chose here, it could happen with any or every problem we deal with. It is a characteristic of the *perceptron* function itself which behaves like a step function.

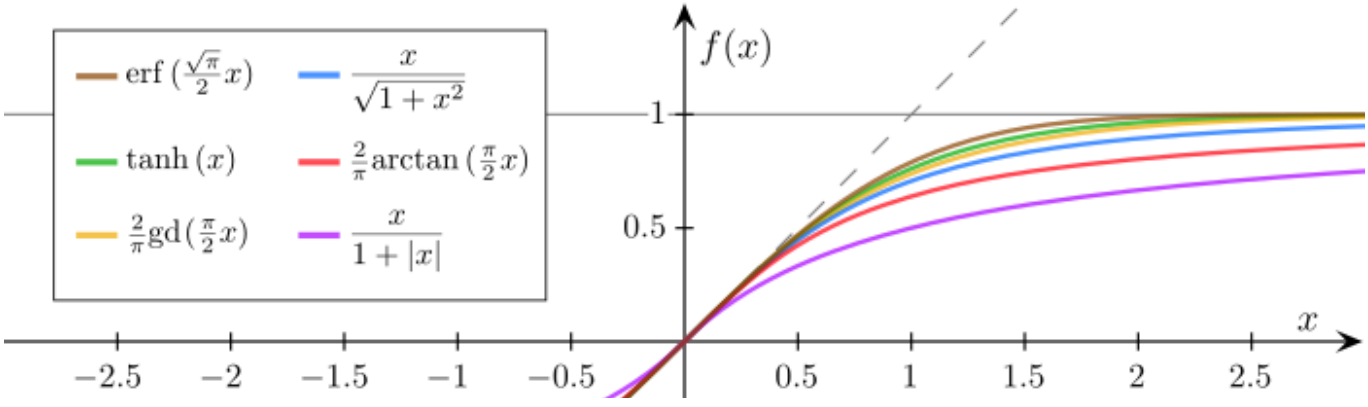
$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$

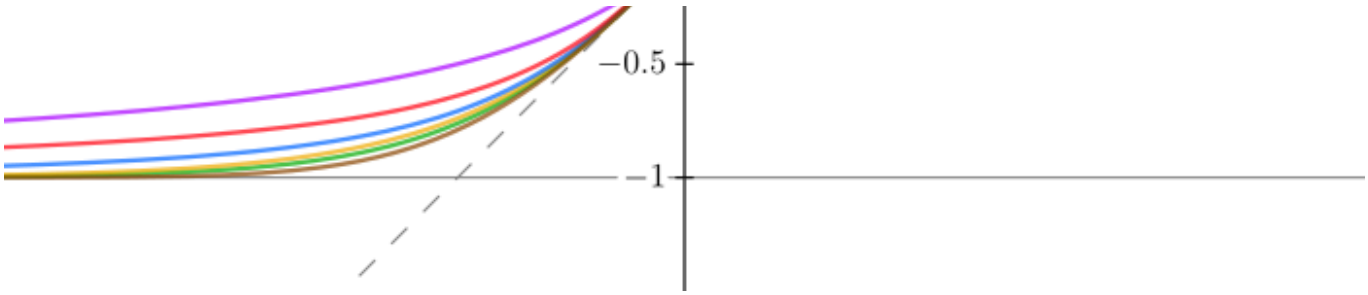
$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$



There will be this sudden change in the decision (from 0 to 1) when z value crosses the threshold (***-w_0***). For most real-world applications we would expect a smoother decision function which gradually changes from 0 to 1.

Introducing sigmoid neurons where the output function is much smoother than the step function seems like a logical and obvious thing to do. Mind you that a sigmoid function is a mathematical function with a characteristic "S"-shaped curve, also called the **sigmoid** curve. There are many functions that can do the job for you, some are shown below:

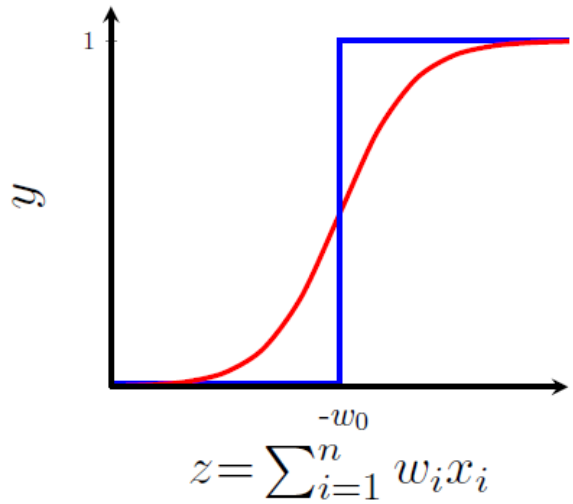




- Wikipedia

One of the simplest one to work with is the logistic function.

$$y = \frac{1}{1 + e^{-(w_0 + \sum_{i=1}^{n} w_i x_i)}}$$



Quick Question: What happens to y when z is infinite? Or when it is -infinite?

We no longer see a sharp transition around the ***w_0***. Also, the output is no longer binary but a real value between 0 and 1 which can be interpreted as a probability. So instead of yes/no decision, we get the probability of yes. The output here is **smooth**, **continuous** and **differentiable** and just how any learning algorithm likes it. To verify this yourself, please look through the **backpropagation** concept in Deep Learning.

Conclusion

In this post, we looked at a *perceptron*, the fundamental unit of deep neural networks. We also showed with examples how a *perceptron*, in contrast with the McCulloch-Pitts neuron, is more generalized and overcomes a few of the pertaining limitations at the time. We briefly established the motivation for Sigmoid neurons as well.

In my next post, we will closely look at the famous *Perceptron Learning Algorithm* and try and get an intuition of why it works, without getting into any of the complex proofs,

along with an implementation of the algorithm in Python from scratch.

Thank you for reading the article.
Live and let live!
A

Photo by Clint Adair on Unsplash

Thanks to Wendy Wong.

Artificial Neural Network | Perceptron | Neural Networks | Deep Learning | Neurons