towards
data science

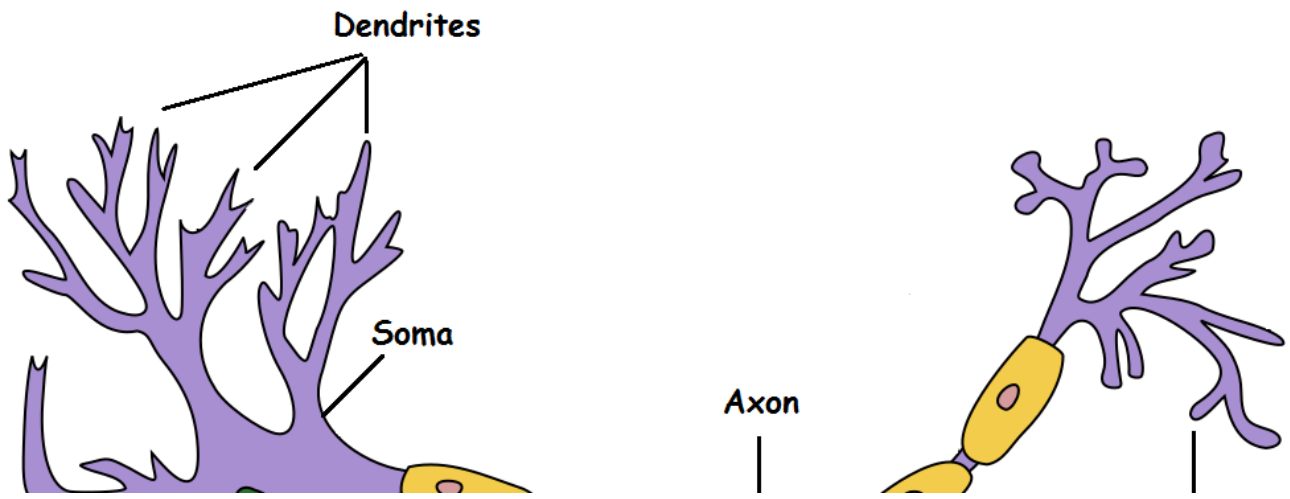
McCulloch-Pitts Neuron — Mankind's First Mathematical Model Of A Biological Neuron

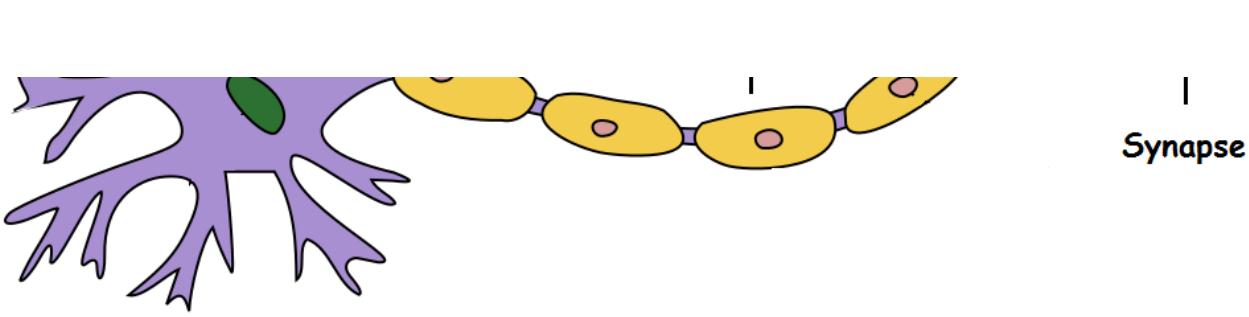
Akshay L Chandra Jul 24, 2018 · 10 min read

It is very well known that the most fundamental unit of deep neural networks is called an *artificial neuron/perceptron*. But the very first step towards the *perceptron* we use today was taken in 1943 by McCulloch and Pitts, by mimicking the functionality of a biological neuron.

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

Biological Neurons: An Overly Simplified Illustration





A Biological Neuron — Wikipedia

Dendrite: Receives signals from other neurons

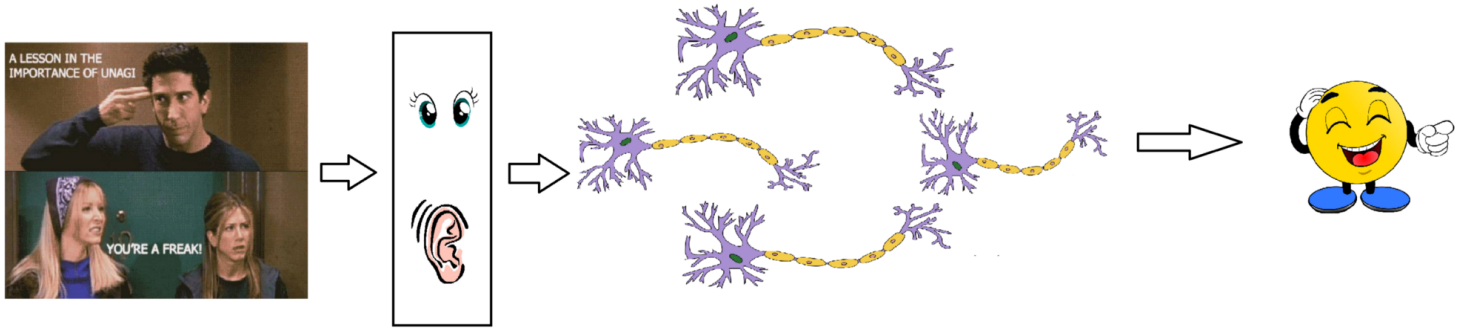
Soma: Processes the information

Axon: Transmits the output of this neuron

Synapse: Point of connection to other neurons

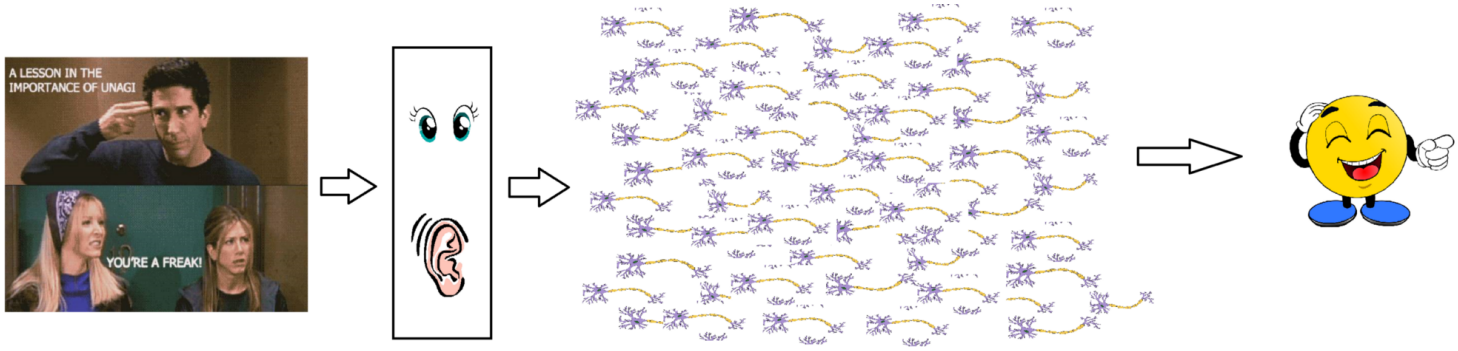
Basically, a neuron takes an input signal (dendrite), processes it like the CPU (soma), passes the output through a cable like structure to other connected neurons (axon to synapse to other neuron's dendrite). Now, this might be biologically inaccurate as there is a lot more going on out there but on a higher level, this is what is going on with a neuron in our brain — takes an input, processes it, throws out an output.

Our sense organs interact with the outer world and send the visual and sound information to the neurons. Let's say you are watching Friends. Now the information your brain receives is taken in by the "laugh or not" set of neurons that will help you make a decision on whether to laugh or not. Each neuron gets fired/activated only when its respective criteria (more on this later) is met like shown below.



Not real.

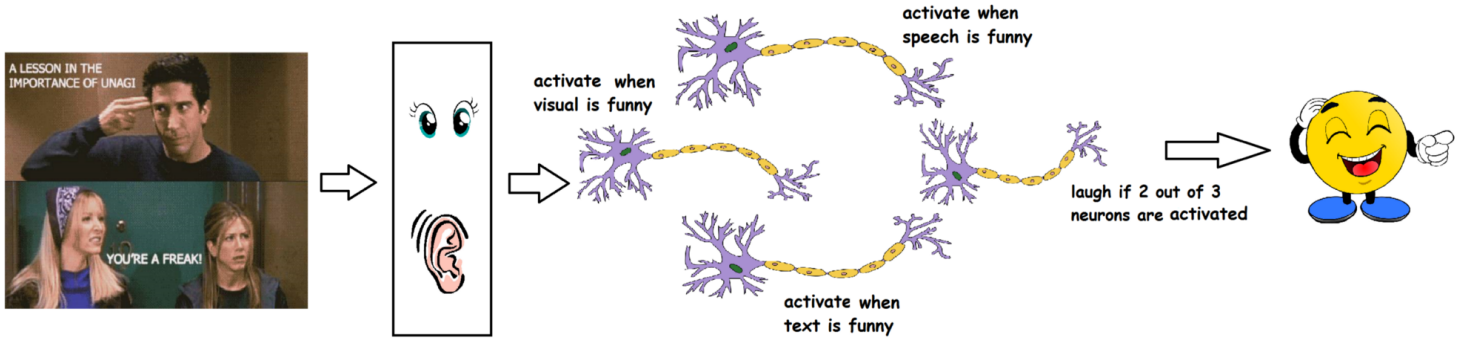
Of course, this is not entirely true. In reality, it is not just a couple of neurons which would do the decision making. There is a massively parallel interconnected network of 10^{11} neurons (100 billion) in our brain and their connections are not as simple as I showed you above. It might look something like this:



Still not real but closer.

Now the sense organs pass the information to the first/lowest layer of neurons to process it. And the output of the processes is passed on to the next layers in a hierarchical manner, some of the neurons will fire and some won't and this process goes on until it results in a final response — in this case, laughter.

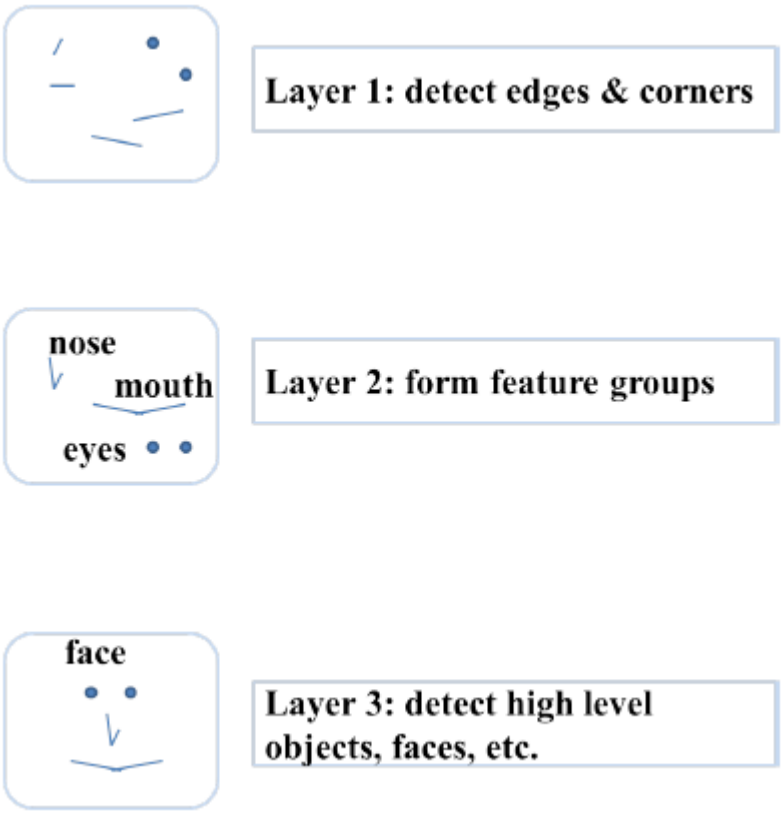
This massively parallel network also ensures that there is a division of work. Each neuron only fires when its intended criteria is met i.e., a neuron may perform a certain role to a certain stimulus, as shown below.



Division of work

It is believed that neurons are arranged in a hierarchical fashion (however, many credible alternatives with experimental support are proposed by the scientists) and each

layer has its own role and responsibility. To detect a face, the brain could be relying on the entire network and not on a single layer.



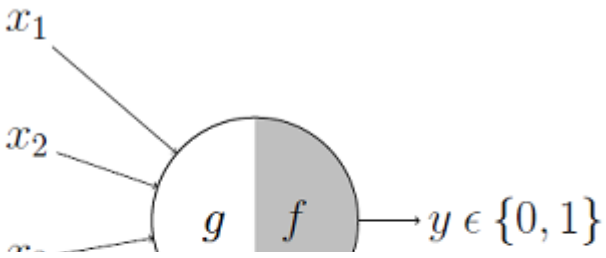
Sample illustration of hierarchical processing. **Credits:** Mitesh M. Khapra's lecture slides

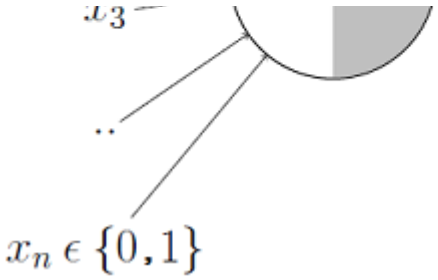
Now that we have established how a biological neuron works, lets look at what McCulloch and Pitts had to offer.

Note: My understanding of how the brain works is very very very limited. The above illustrations are overly simplified.

McCulloch-Pitts Neuron

The first computational model of a neuron was proposed by Warren MuCulloch (neuroscientist) and Walter Pitts (logician) in 1943.





This is where it all began..

It may be divided into 2 parts. The first part, ***g*** takes an input (ahem dendrite ahem), performs an aggregation and based on the aggregated value the second part, ***f*** makes a decision.

Lets suppose that I want to predict my own decision, whether to watch a random football game or not on TV. The inputs are all boolean i.e., {0,1} and my output variable is also boolean {0: Will watch it, 1: Won't watch it}.

- So, ***x_1*** could be *isPremierLeagueOn* (I like Premier League more)
- ***x_2*** could be *isItAFriendlyGame* (I tend to care less about the friendlies)
- ***x_3*** could be *isNotHome* (Can't watch it when I'm running errands. Can I?)
- ***x_4*** could be *isManUnitedPlaying* (I am a big Man United fan. GGMU!) and so on.

These inputs can either be *excitatory* or *inhibitory*. Inhibitory inputs are those that have maximum effect on the decision making irrespective of other inputs i.e., if ***x_3*** is 1 (not home) then my output will always be 0 i.e., the neuron will never fire, so ***x_3*** is an inhibitory input. Excitatory inputs are NOT the ones that will make the neuron fire on their own but they might fire it when combined together. Formally, this is what is going on:

$$g(x_1, x_2, x_3, ..., x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

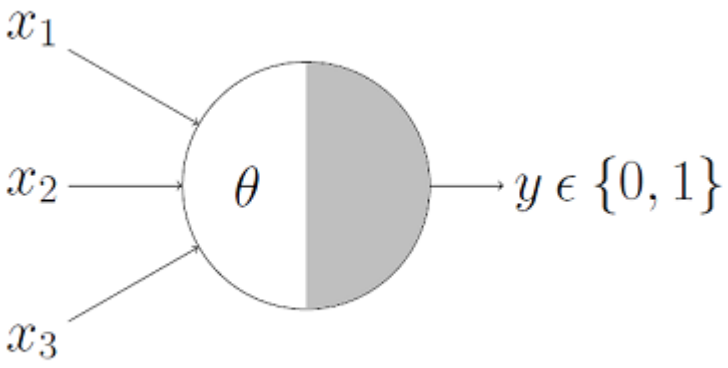
$$\begin{aligned} y = f(g(\mathbf{x})) &= 1 \quad if \quad g(\mathbf{x}) \geq \theta \\ &= 0 \quad if \quad g(\mathbf{x}) < \theta \end{aligned}$$

We can see that ***g(x)*** is just doing a sum of the inputs — a simple aggregation. And ***theta*** here is called thresholding parameter. For example, if I always watch the game when the sum turns out to be 2 or more, the ***theta*** is 2 here. This is called the Thresholding Logic.

Boolean Functions Using M-P Neuron

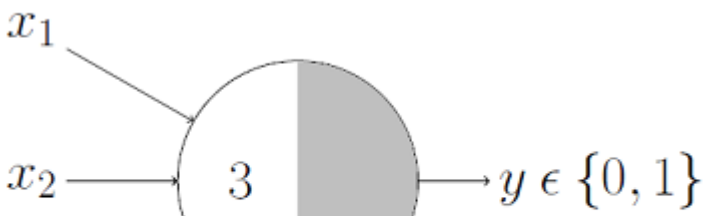
So far we have seen how the M-P neuron works. Now lets look at how this very neuron can be used to represent a few boolean functions. Mind you that our inputs are all boolean and the output is also boolean so essentially, the neuron is just trying to learn a boolean function. A lot of boolean decision problems can be cast into this, based on appropriate input variables— like whether to continue reading this post, whether to watch Friends after reading this post etc. can be represented by the M-P neuron.

M-P Neuron: A Concise Representation



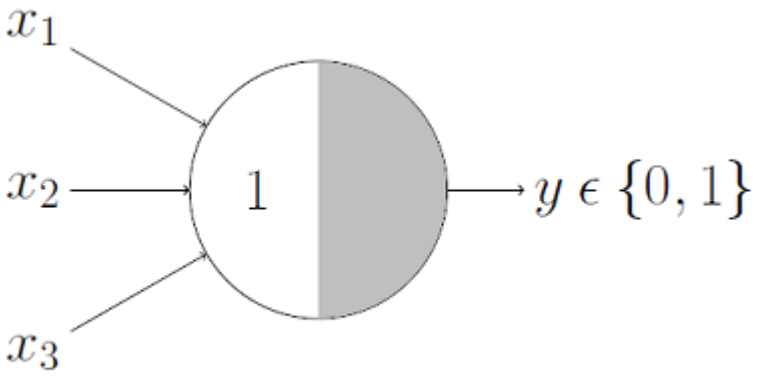
This representation just denotes that, for the boolean inputs ***x_1***, ***x_2*** and ***x_3*** if the ***g(x)*** i.e., **sum ≥ theta**, the neuron will fire otherwise, it won't.

AND Function



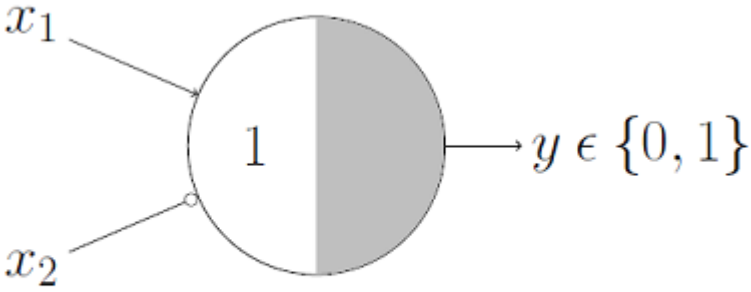
An AND function neuron would only fire when ALL the inputs are ON i.e., $g(\mathbf{x}) \geq 3$ here.

OR Function



I believe this is self explanatory as we know that an OR function neuron would fire if ANY of the inputs is ON i.e., $g(\mathbf{x}) \geq 1$ here.

A Function With An Inhibitory Input



$$x_1\ AND\ !x_2^*$$

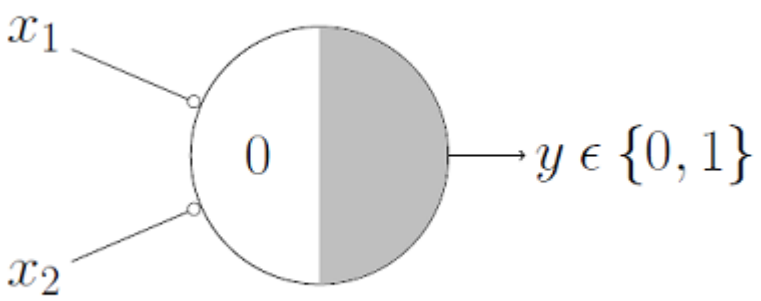
Now this might look like a tricky one but it's really not. Here, we have an inhibitory input i.e., ***x_2*** so whenever ***x_2*** is 1, the output will be 0. Keeping that in mind, we know that ***x_1 AND !x_2*** would output 1 only when ***x_1*** is 1 and ***x_2*** is 0 so it is obvious that the threshold parameter should be 1.

Lets verify that, the $g(\mathbf{x})$ i.e., ***x_1*** + ***x_2*** would be ≥ 1 in only 3 cases:

Case 1: when ***x_1*** is 1 and ***x_2*** is 0
Case 2: when ***x_1*** is 1 and ***x_2*** is 1
Case 3: when ***x_1*** is 0 and ***x_2*** is 1

But in both Case 2 and Case 3, we know that the output will be 0 because ***x_2*** is 1 in both of them, thanks to the inhibition. And we also know that ***x_1 AND !x_2*** would output 1 for Case 1 (above) so our thresholding parameter holds good for the given function.

NOR Function



For a NOR neuron to fire, we want ALL the inputs to be 0 so the thresholding parameter should also be 0 and we take them all as inhibitory input.

NOT Function

For a NOT neuron, 1 outputs 0 and 0 outputs 1. So we take the input as an inhibitory input and set the thresholding parameter to 0. It works!

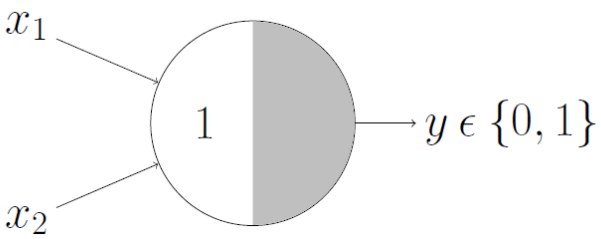
Can any boolean function be represented using the M-P neuron? Before you answer that, lets understand what M-P neuron is doing geometrically.

Geometric Interpretation Of M-P Neuron

This is the best part of the post according to me. Lets start with the OR function.

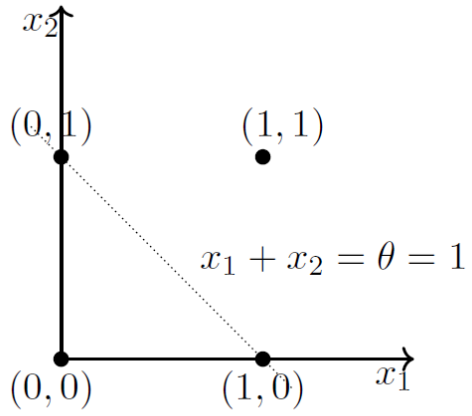
OR Function

We already discussed that the OR function's thresholding parameter ***theta*** is 1, for obvious reasons. The inputs are obviously boolean, so only 4 combinations are possible — (0,0), (0,1), (1,0) and (1,1). Now plotting them on a 2D graph and making use of the OR function's aggregation equation
i.e., ***x_1 + x_2 ≥ 1*** using which we can draw the decision boundary as shown in the graph below. Mind you again, this is not a real number graph.



OR function

$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 1$$

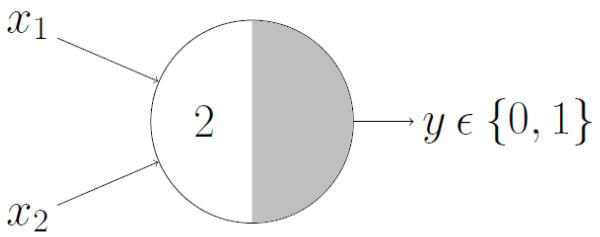


We just used the aggregation equation i.e., ***x_1 + x_2 =1*** to graphically show that all those inputs whose output when passed through the OR function M-P neuron lie ON or ABOVE that line and all the input points that lie BELOW that line are going to output 0.

Voila!! The M-P neuron just learnt a linear decision boundary! The M-P neuron is splitting the input sets into two classes — positive and negative. Positive ones (which output 1) are those that lie ON or ABOVE the decision boundary and negative ones (which output 0) are those that lie BELOW the decision boundary.

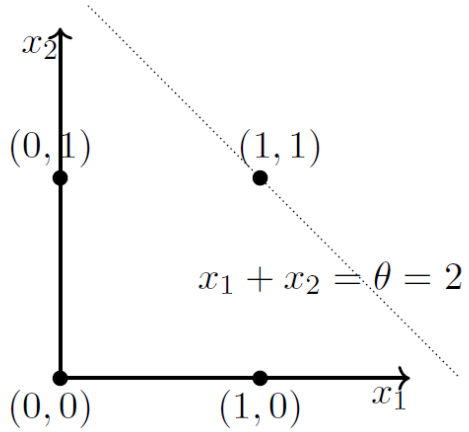
Lets convince ourselves that the M-P unit is doing the same for all the boolean functions by looking at more examples (if it is not already clear from the math).

AND Function



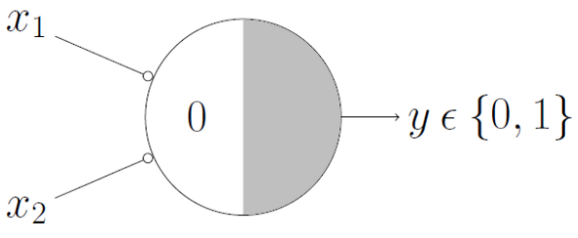
AND function

$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 2$$

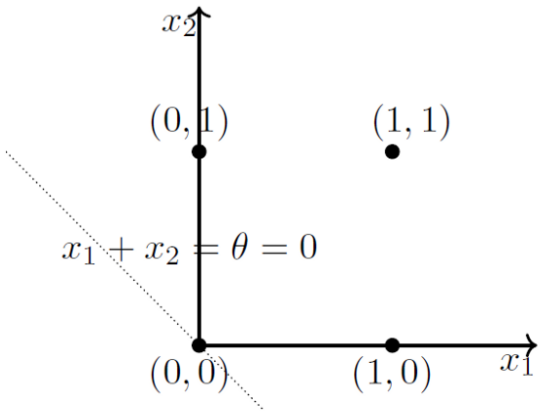


In this case, the decision boundary equation is ***x_1 + x_2 =2***. Here, all the input points that lie ON or ABOVE, just (1,1), output 1 when passed through the AND function M-P neuron. It fits! The decision boundary works!

Tautology



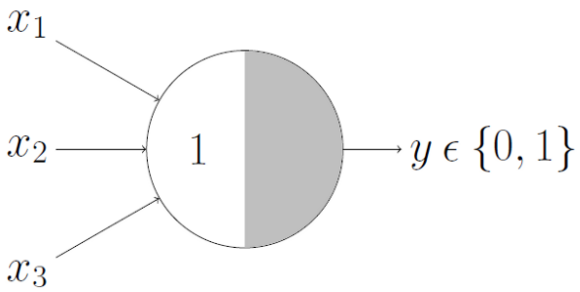
Tautology (always ON)



Too easy, right?

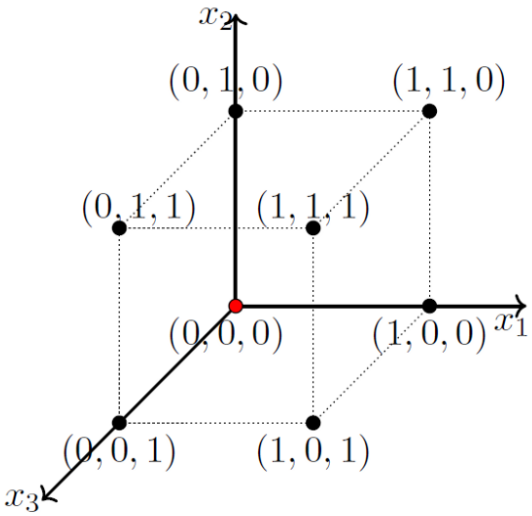
I think you get it by now but what if we have more than 2 inputs?

OR Function With 3 Inputs



OR function

$$x_1 + x_2 + x_3 = \sum_{i=1}^{3} x_i \geq 1$$

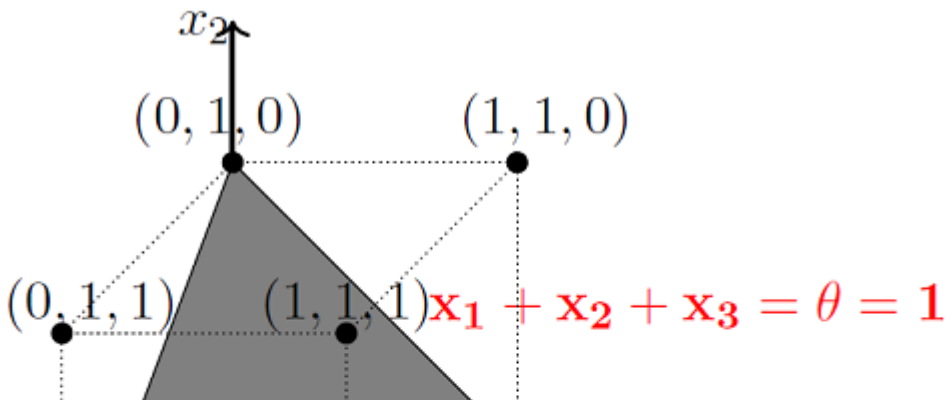


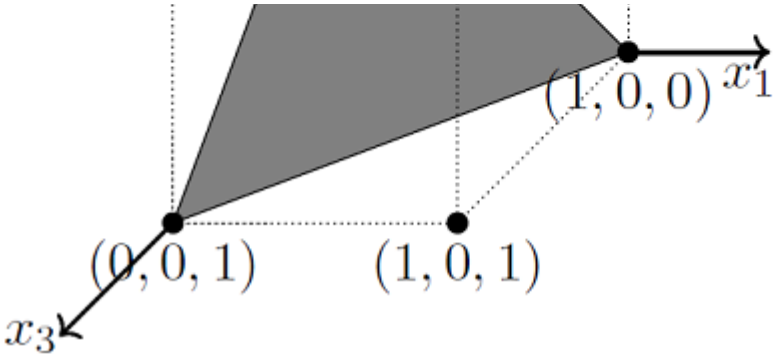
Lets just generalize this by looking at a 3 input OR function M-P unit. In this case, the possible inputs are 8 points — (0,0,0), (0,0,1), (0,1,0), (1,0,0), (1,0,1),… you got the point(s). We can map these on a 3D graph and this time we draw a decision boundary in 3 dimensions.

> *"Is it a bird? Is it a plane?"*

Yes, it is a PLANE!

The plane that satisfies the decision boundary equation ***x_1 + x_2 + x_3 = 1*** is shown below:





Take your time and convince yourself by looking at the above plot that all the points that lie ON or ABOVE that plane (positive half space) will result in output 1 when passed through the OR function M-P unit and all the points that lie BELOW that plane (negative half space) will result in output 0.

Just by hand coding a thresholding parameter, M-P neuron is able to conveniently represent the boolean functions which are linearly separable.

Linear separability (for boolean functions): There exists a line (plane) such that all inputs which produce a 1 lie on one side of the line (plane) and all inputs which produce a 0 lie on other side of the line (plane).

Limitations Of M-P Neuron

- What about non-boolean (say, real) inputs?
- Do we always need to hand code the threshold?
- Are all inputs equal? What if we want to assign more importance to some inputs?
- What about functions which are not linearly separable? Say XOR function.

I hope it is now clear why we are not using the M-P neuron today. Overcoming the limitations of the M-P neuron, Frank Rosenblatt, an American psychologist, proposed the classical perception model, the mighty *artificial neuron*, in 1958. It is more generalized computational model than the McCulloch-Pitts neuron where weights and thresholds can be learnt over time.

More on *perceptron* and how it learns the weights and thresholds etc. in my future posts.

Conclusion

In this article, we briefly looked at biological neurons. We then established the concept of MuCulloch-Pitts neuron, the first ever mathematical model of a biological neuron. We represented a bunch of boolean functions using the M-P neuron. We also tried to get a geometric intuition of what is going on with the model, using 3D plots. In the end, we also established a motivation for a more generalized model, the one and only *artificial neuron/perceptron* model.

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

Deep Learning | Artificial Neuron | Biological Neuron | Muculloch Pitts Neuron | Perceptron

towards
data science

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

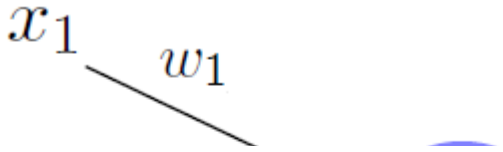
Akshay L Chandra Aug 11, 2018 · 7 min read

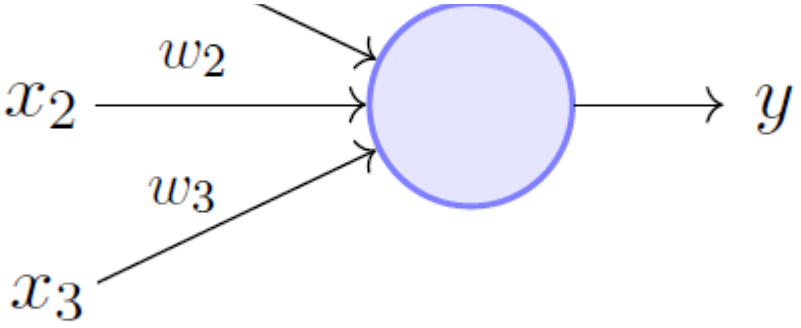
The most fundamental unit of a deep neural network is called an *artificial neuron*, which takes an input, processes it, passes it through an activation function like the Sigmoid, return the activated output. In this post, we are only going to talk about the *perceptron* model proposed before the 'activation'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

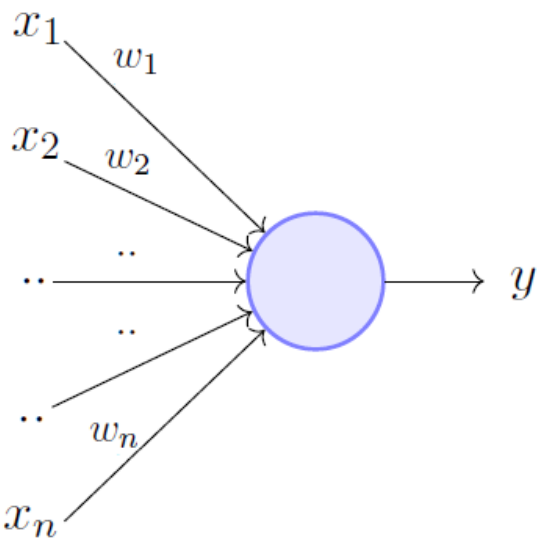
Perceptron





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.



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

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

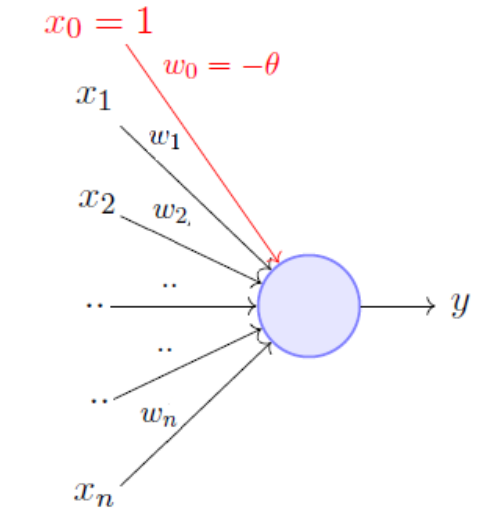
Rewriting the above,

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

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

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*).



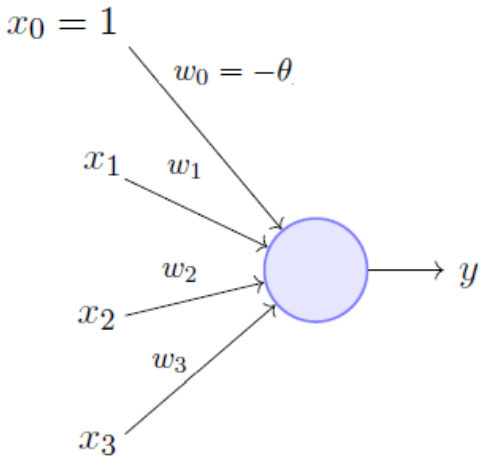
A more accepted convention,

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

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

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

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.

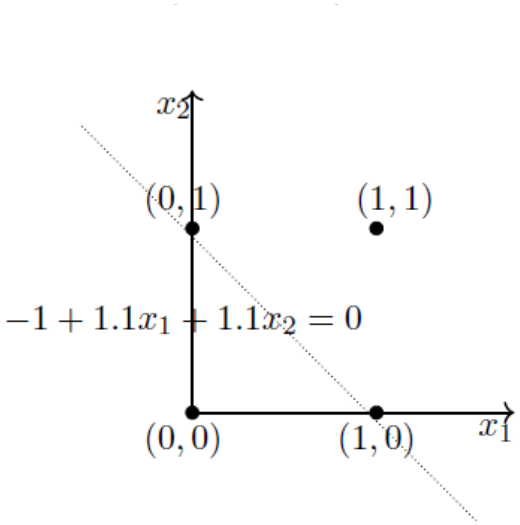
x_1	x_2	OR

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.

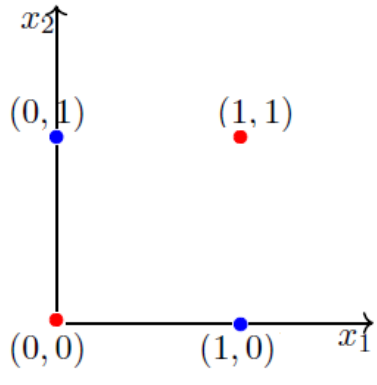
Now if you actually try and solve the linear equations above, you will realize that there can be multiple solutions. But which solution is the best? To more formally define the 'best' solution, we need to understand errors and error surfaces, which we will do in my next post on *Perceptron Learning Algorithm*.

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

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

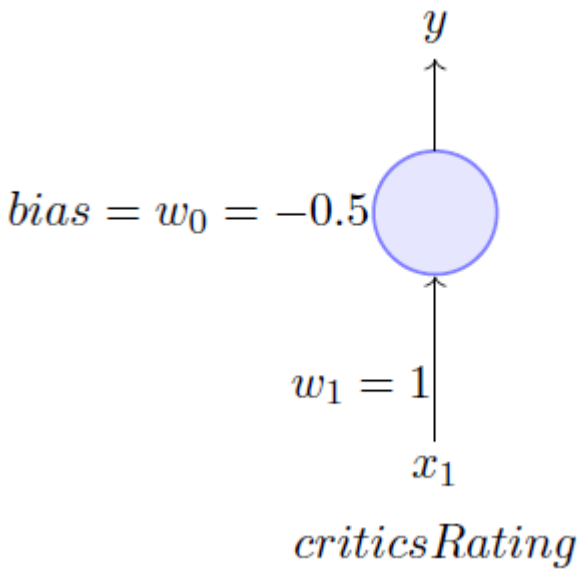


Notice that the fourth equation contradicts the second and the third equation. Point is, there are no *perceptron* solutions for non-linearly separated data. So the key take away is that a **single** *perceptron* cannot learn to separate the data that are non-linear in nature.

> ***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:

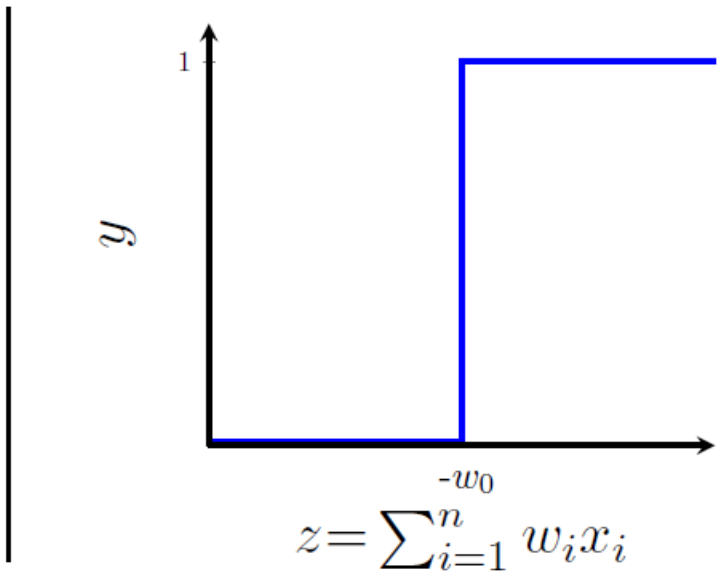


What would be the decision for a movie with *criticsRating* = 0.51? *Yes!*
What would be the decision for a movie with *criticsRating* = 0.49? *No!*

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.

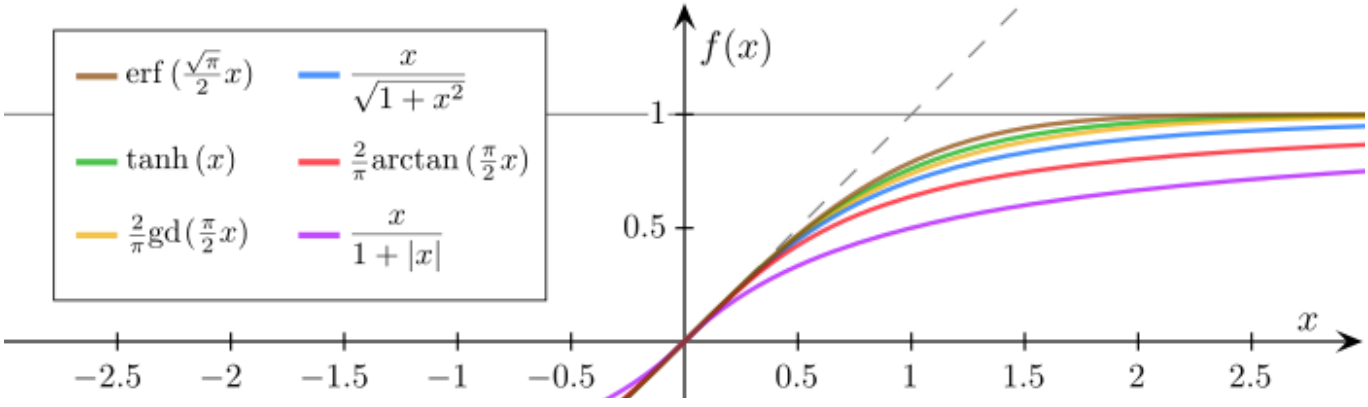
$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$

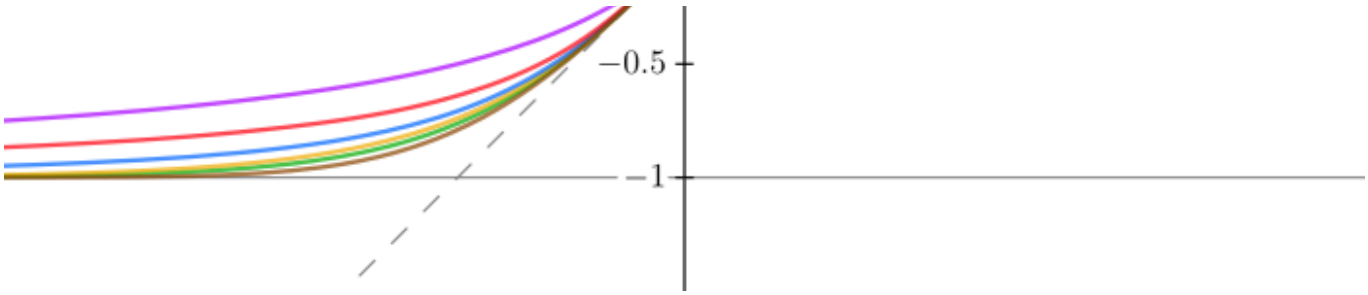
$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$



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:

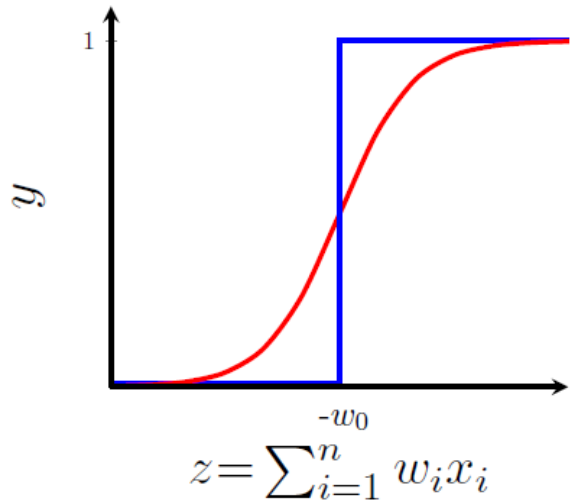




- Wikipedia

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