AMRITA
VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

21AI637 Deep Learning 3-0-2-4 (S2 MTech AI)
21AM645 Deep Learning 3-0-2-4 (S2 M.Tech CS(AI/ML)

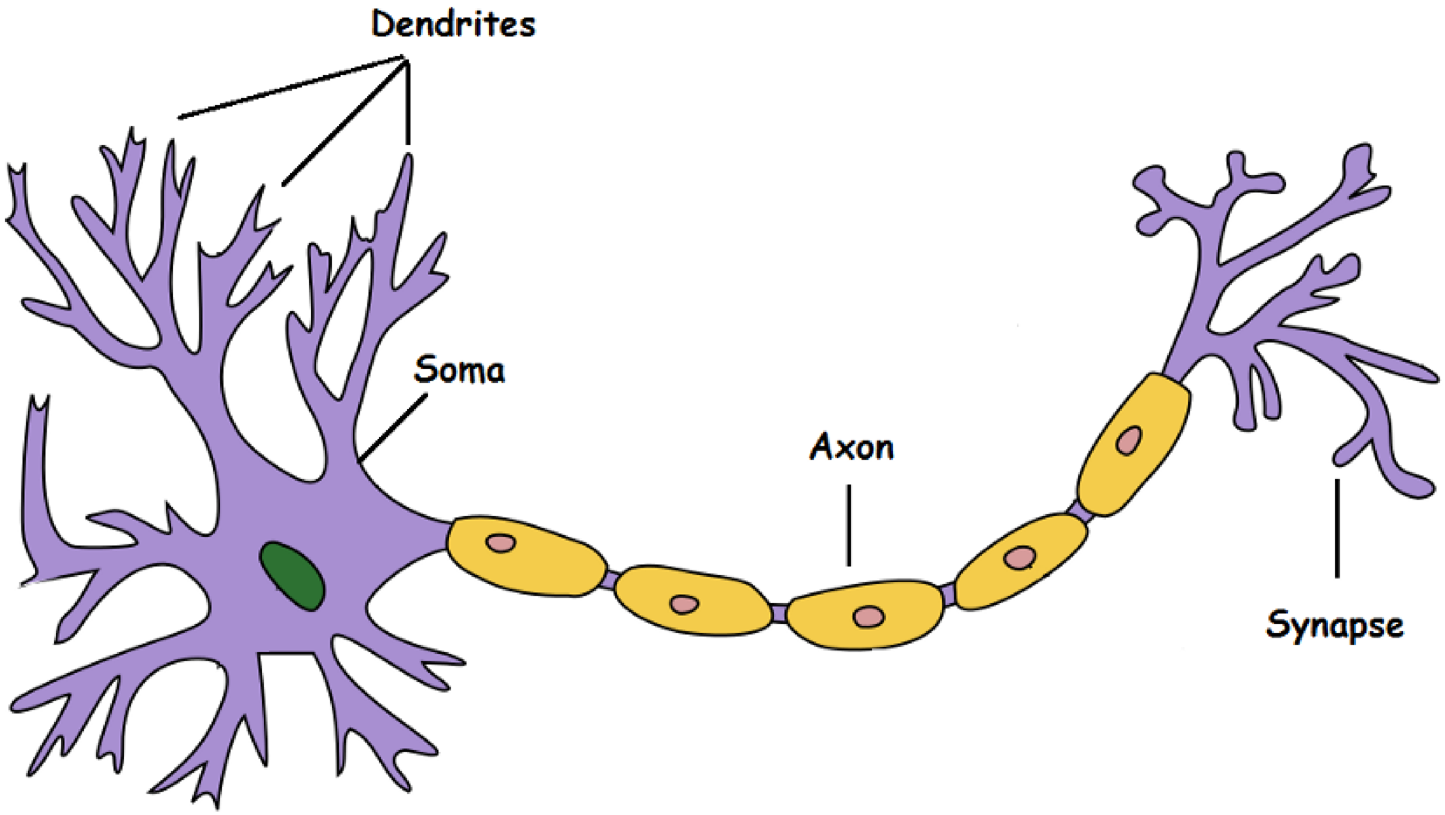
MP Neuron Perceptron

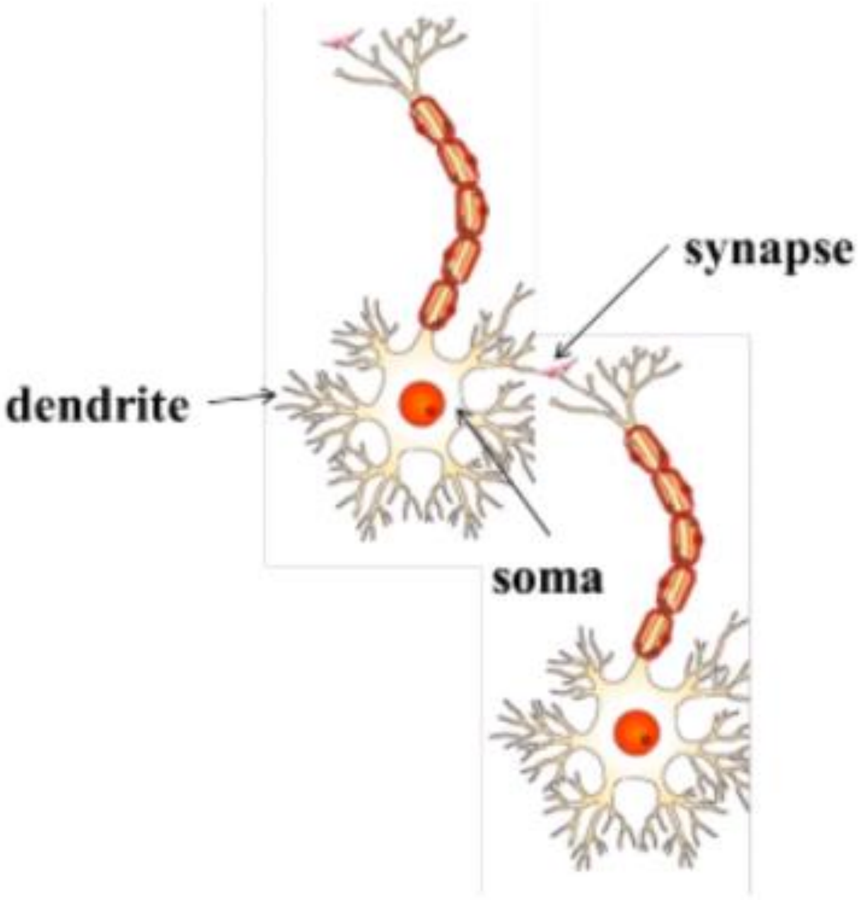
Courtesy :: NPTel lecture series from IIT Madras – Mitesh Kapra,
Coursera : Andrew NG, Deep Learning
fast.ai, Dive into Deep Learning

Takes an input, processes it, throws out an output.

Biological Neuron

Dendrite: Receives signals from other neurons
Soma: Processes the information
Axon: Transmits the output of this neuron
Synapse: Point of connection to other neurons





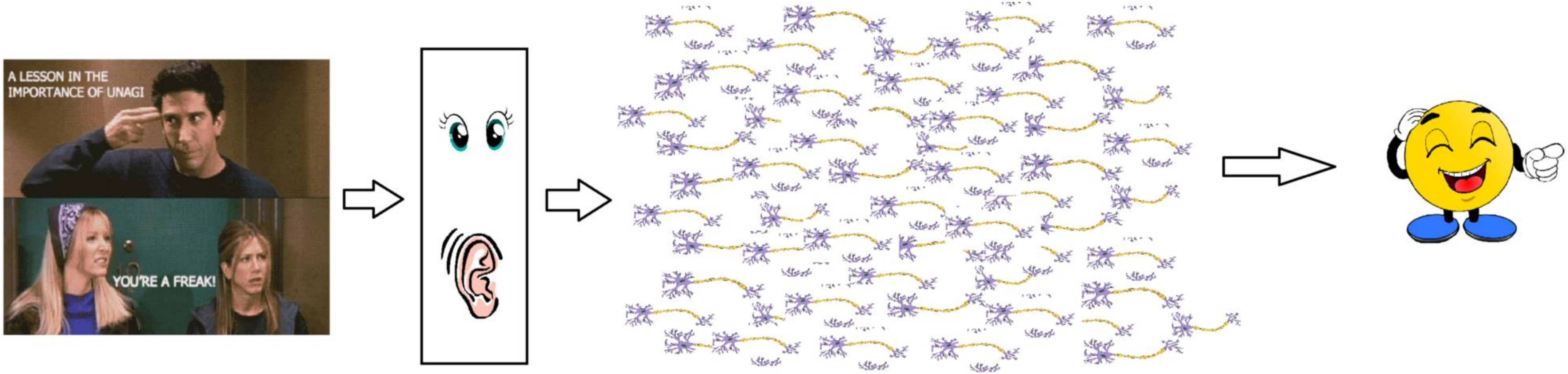
Courtesy: Towardsdatascience.com

Biological Neuron

Each neuron gets fired/activated only when its respective criteria is met

Our sense organs interact with the outer world and send the visual and sound information to the neurons

. Let's say you are watching a video clip. 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.

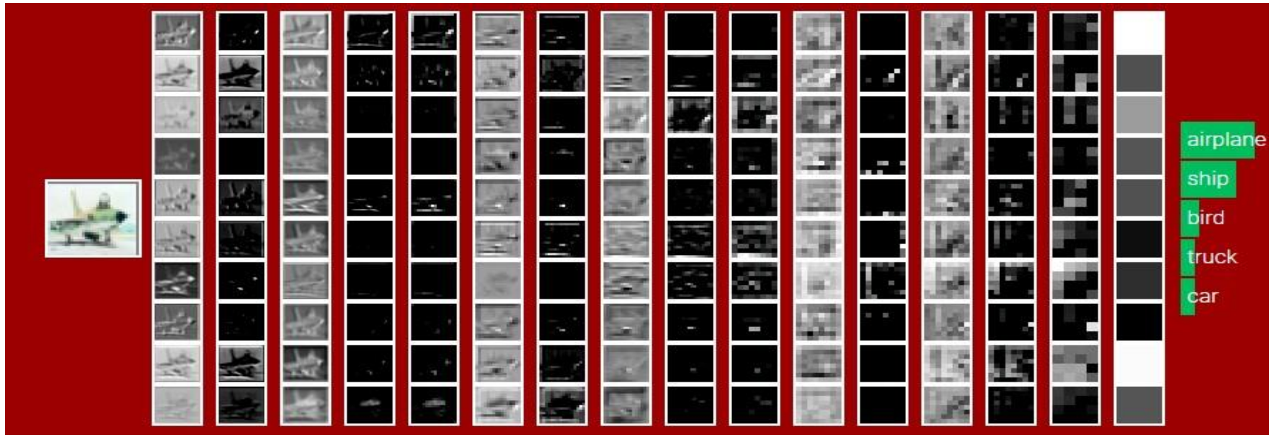


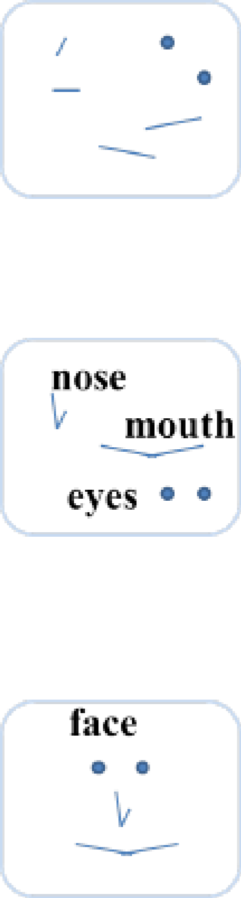
There is a massively parallel interconnected network of 10^{11} neurons (100 billion) in our brain and their connections are not simple

Courtesy: Towardsdatascience.com

Neurons are arranged in a hierarchical fashion

Neurons are arranged in a hierarchical fashion 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.





Layer 1: detect edges & corners

Layer 2: form feature groups

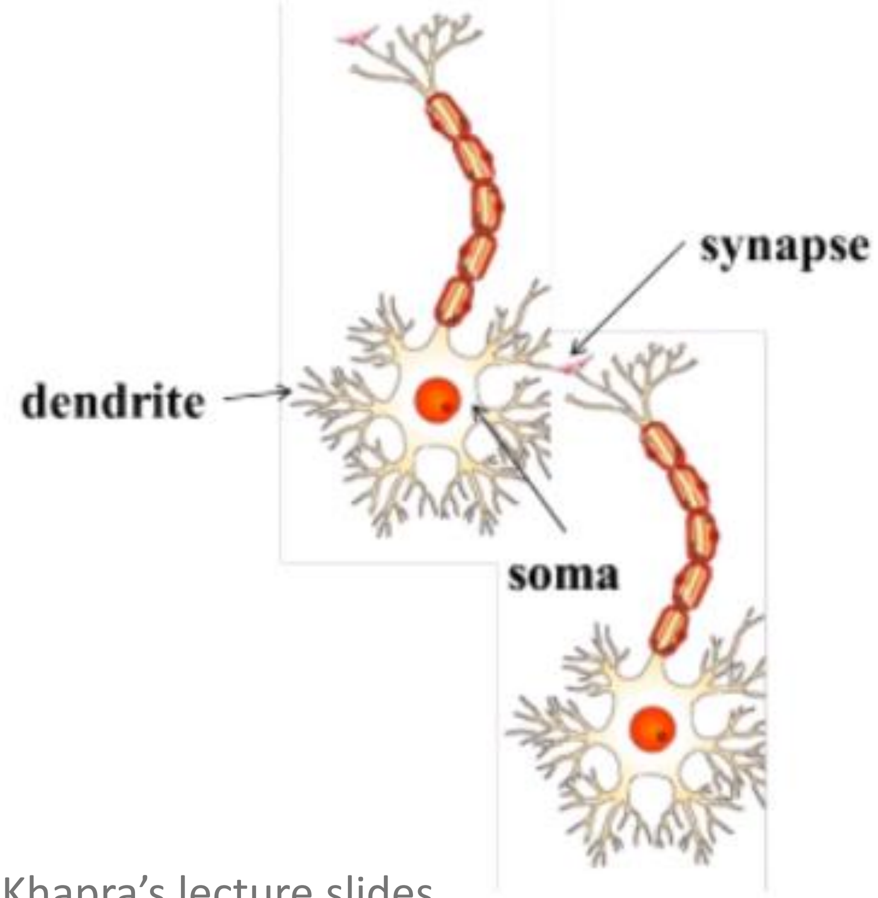
Layer 3: detect high level objects, faces, etc.

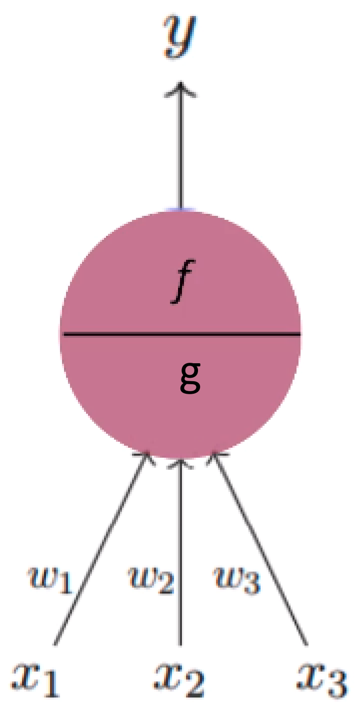
Courtesy :Sample illustration of hierarchical processing. Credits: Mitesh M. Khapra's lecture slides

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.

Recall Biological Neuron

The most fundamental unit of deep neural networks is called an *artificial neuron/perceptron*





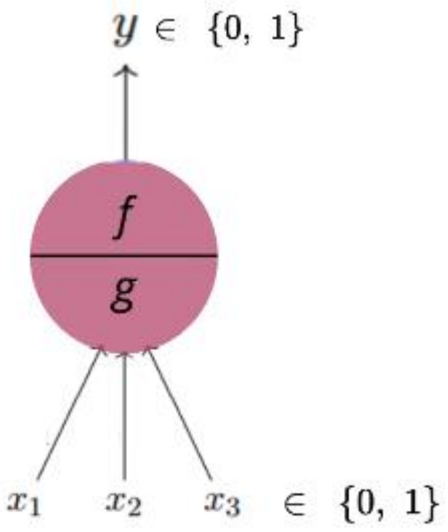
Courtesy :: Mitesh M. Khapra's lecture slides

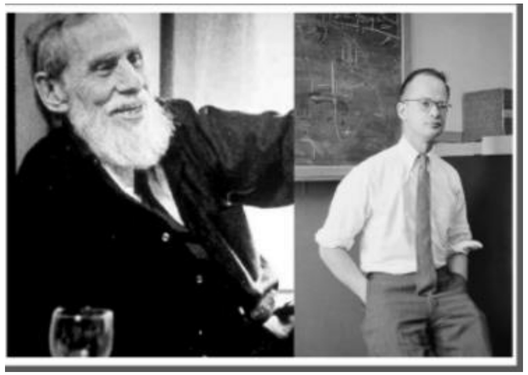
Artificial Neuron

“threshold logic” to mimic the thought process

The fundamental Building block of Deep Learning

Artificial Neural Networks





McCulloch-Pitts Neuron

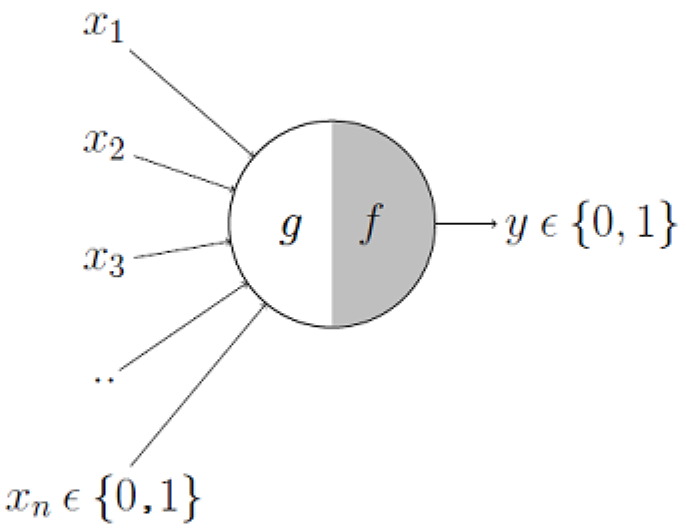
The first computational model of a neuron was proposed by Warren MuCulloch (neuroscientist) and Walter Pitts (logician) in 1943

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.

Excitatory or inhibitory inputs.



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

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

These inputs can either be *excitatory* or *inhibitory*.

Inhibitory inputs are those that have maximum effect on the decision making irrespective of other inputs

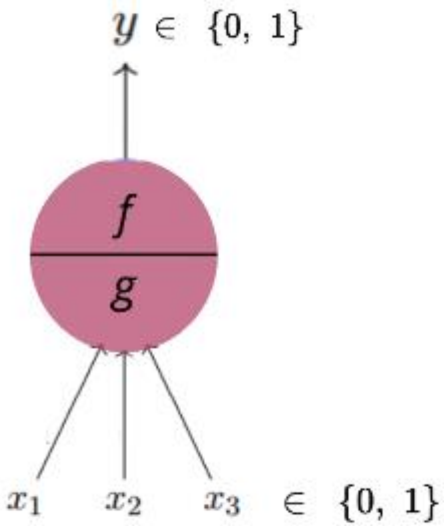
i.e., if ***x_3*** is 1 (not home) then my output will always be 0 i.e., the neuron will never fire, so ***x_3*** is an inhibitory input

Excitatory inputs are NOT the ones that will make the neuron fire on their own but they might fire it when combined together

The mathematical Model- McCulloch-Pitts Neuron (MP Neuron)

- McCulloch and Pitts proposed a highly simplified computational model of the neuron.
- ***g*** aggregates the inputs and the function ***f*** takes a decision based on this aggregation.
- The inputs can be excitatory or inhibitory

These inputs can either be *excitatory* or *inhibitory*



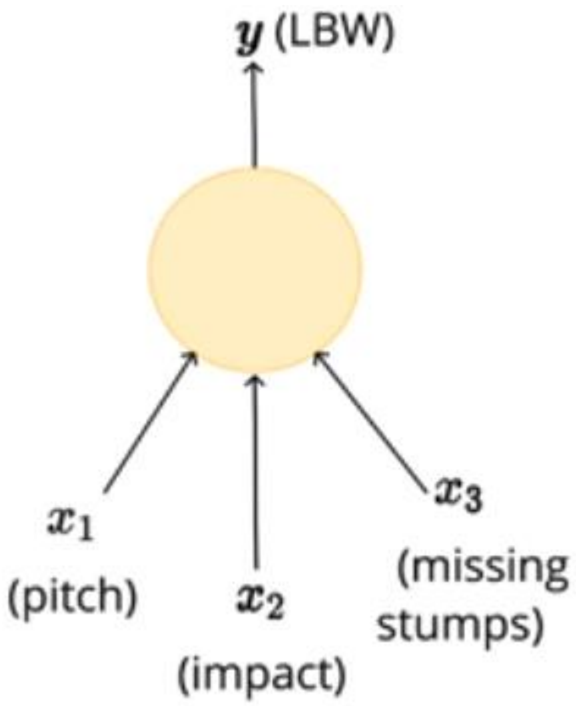
$y = 0$ if any x_i is inhibitory, else

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

$$y = f(g(x)) = 1 \text{ if } g(x) \geq b$$

$$= 0 \text{ if } g(x) < b$$

What task can be done with MP Neuron



Pitch in line	Impact	Missing stumps	Is it LBW? (y)
1	0	0	0
0	1	1	0
1	1	1	1
0	1	0	[illegible]

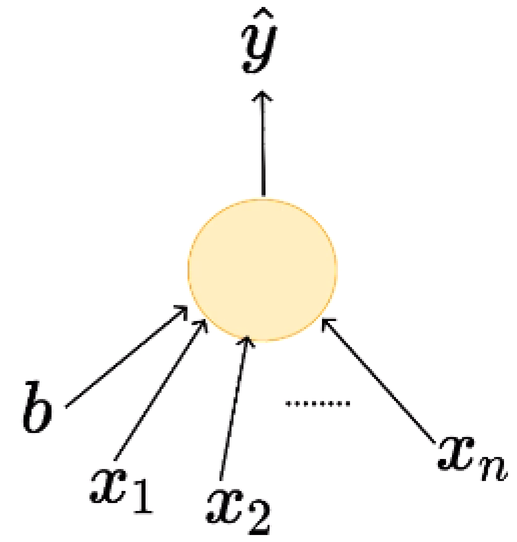
Courtesy :: Mitesh M. Khapra's lecture slides

We aim at minimizing the loss function

Loss Function

Launch (within 6 months)	0	1	1	0	0	1	0	1	1	0
Weight (<160g)	1	0	1	0	0	0	1	0	0	1
Screen size (<5.9 in)	1	0	1	0	1	0	1	0	1	0
dual sim	1	1	0	0	0	1	0	1	0	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1	0
NFC	0	1	1	0	1	0	1	1	1	0
Radio	1	0	0	1	1	1	0	0	0	0
Battery(>3500mAh)	0	0	0	1	0	1	0	1	0	0
Price > 20k	0	1	1	0	0	0	1	1	1	0
Like? (y)	1	0	**1**	**0**	1	1	**0**	**1**	0	0
prediction $\hat{y}$	1	0	**0**	**1**	1	1	**1**	**0**	0	0
loss	0	0	1	-1	0	0	-1	1	0	0

$$loss = \sum_i y_i - \hat{y}_i$$



$$\hat{y} = \sum_{i=1}^{n} x_i \geq b$$

$$\hat{y} = x_1 + x_2 \geq b$$

$$\boxed{loss = \sum_i (y_i - \hat{y}_i)^2}$$

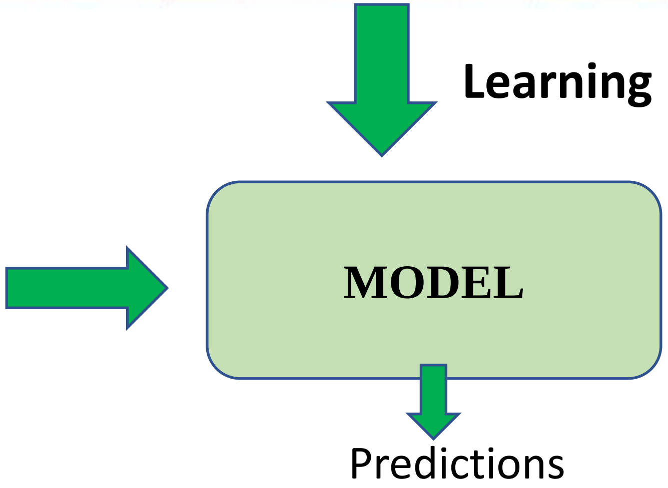
Courtesy :: Mitesh M. Khapra's lecture slides

Training data

Launch (within 6 months)	0	1	1	0	0	1	0	1	1	0
Weight (<160g)	1	0	1	0	0	0	1	0	0	1
Screen size (<5.9 in)	1	0	1	0	1	0	1	0	1	0
dual sim	1	1	0	0	0	1	0	1	0	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1	0
NFC	0	1	1	0	1	0	1	1	1	0
Radio	1	0	0	1	1	1	0	0	0	0
Battery(>3500mAh)	0	0	0	1	0	1	0	1	0	0
Price > 20k	0	1	1	0	0	0	1	1	1	0
Like? (y)	1	1	1	0	0	1	1	1	0	0
predicted	1	1	0	1	1	1	1	0	0	0

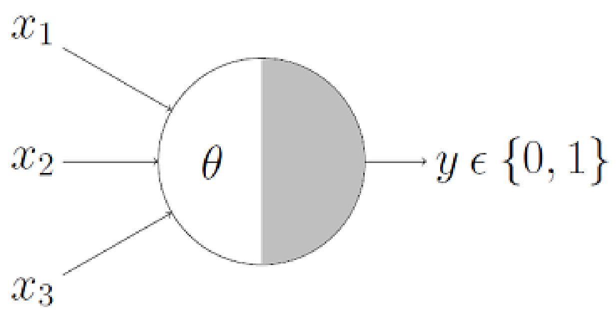
Test data

1	0	0	1
0	1	1	1
0	1	1	1
0	1	0	0
1	0	0	0
0	0	1	0
1	1	1	0
1	1	1	0
0	0	1	0
0	1	0	0
0	1	1	0



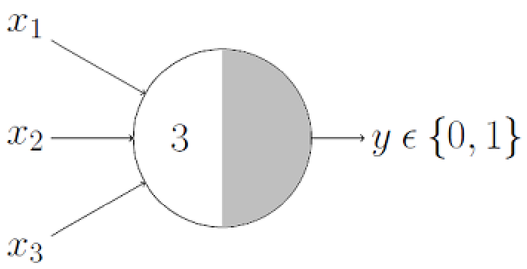
$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} = \frac{3}{4} = 75\%$$

Boolean Functions Using M-P Neuron



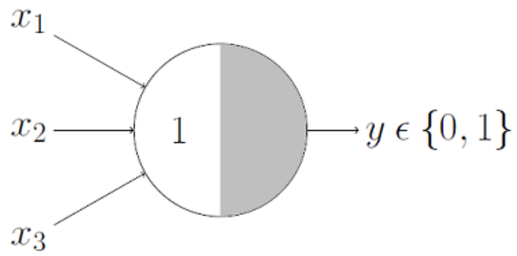
sum ≥ theta, the neuron will fire otherwise, it won't.

AND Function



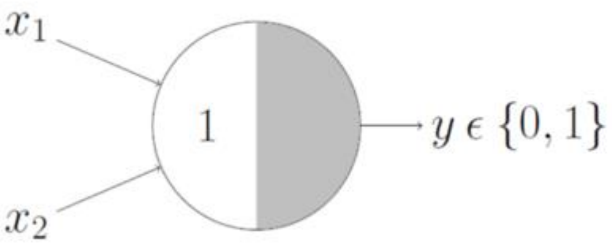
An AND function neuron would only fire when ALL the inputs are ON i.e., ***g*(x)** ≥ 3 here.

OR Function



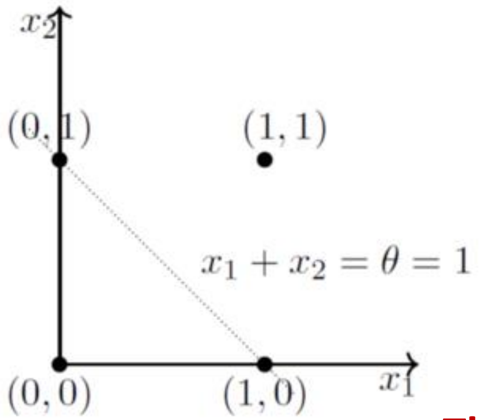
neuron would fire if ANY of the inputs is ON i.e., ***g*(x)** ≥ 1 here.

Geometric Interpretation OR function



OR function

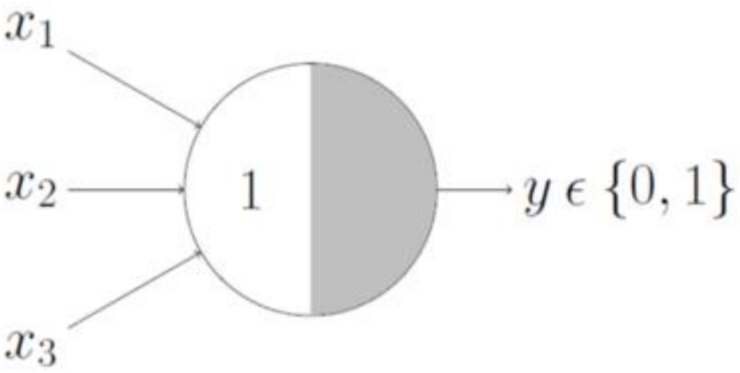
$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 1$$



***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.

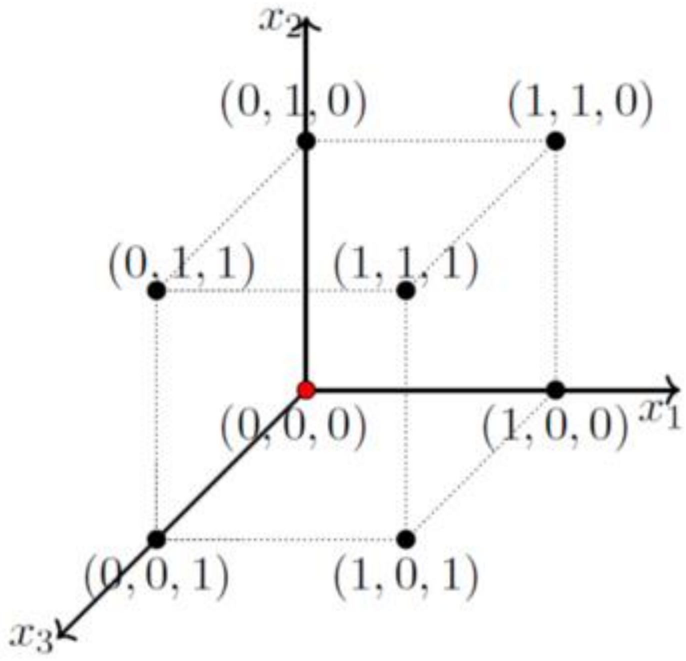
The M-P neuron just learnt a linear decision boundary!

OR Function With 3 Inputs

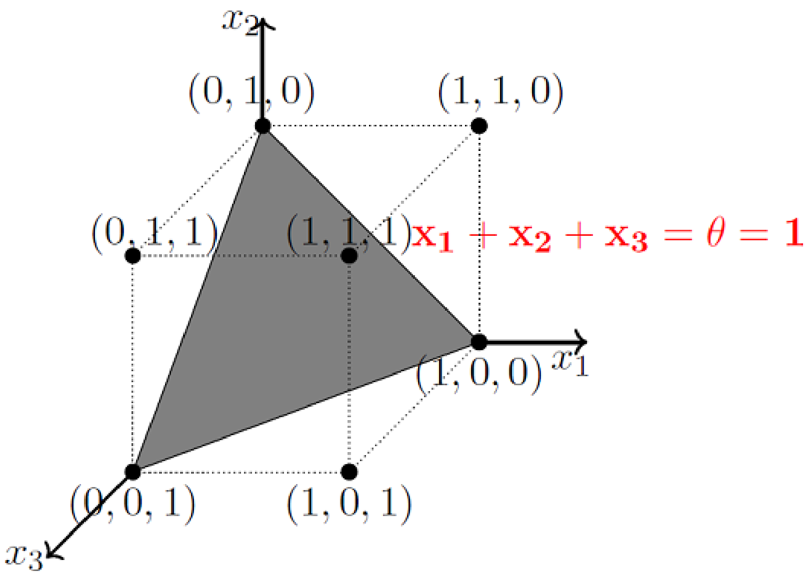


OR function

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



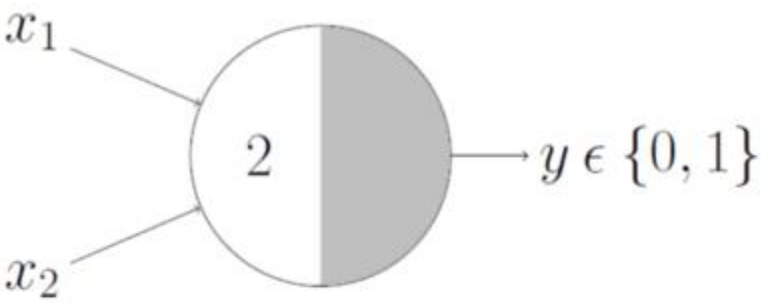
The plane that satisfies the decision boundary equation ***x_1 + x_2 + x_3 =* 1** is shown below:



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.

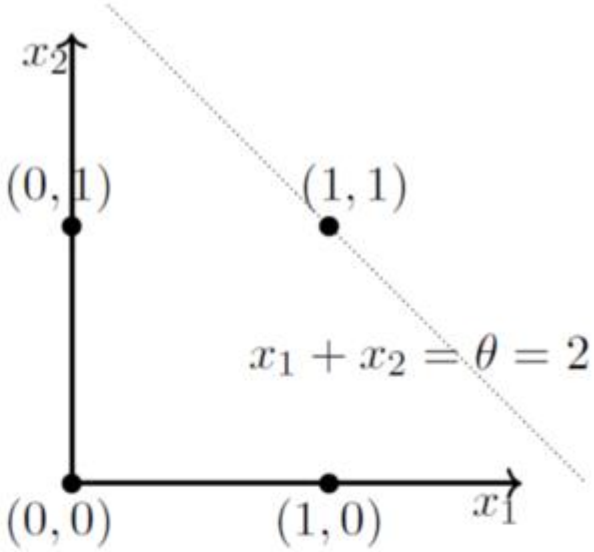
Boolean Functions Using M-P Neuron

AND Function



AND function

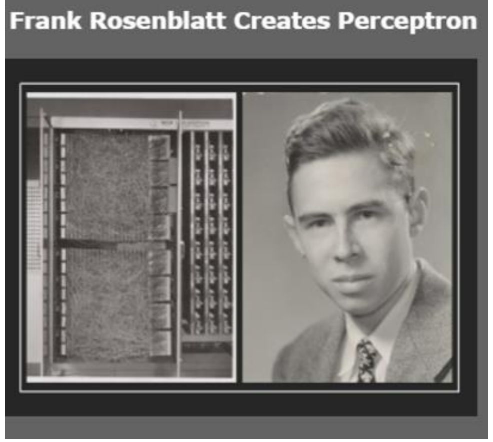
$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 2$$



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!

Limitations Of M-P Neuron

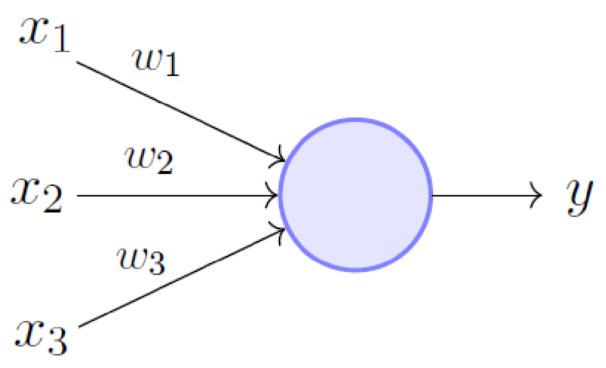
- What about non-boolean (say, real) inputs?
- Do we always need to hand code the threshold?
- Are all inputs equal? What if we want to assign more importance to some inputs?
- What about functions which are not linearly separable? Say XOR function.



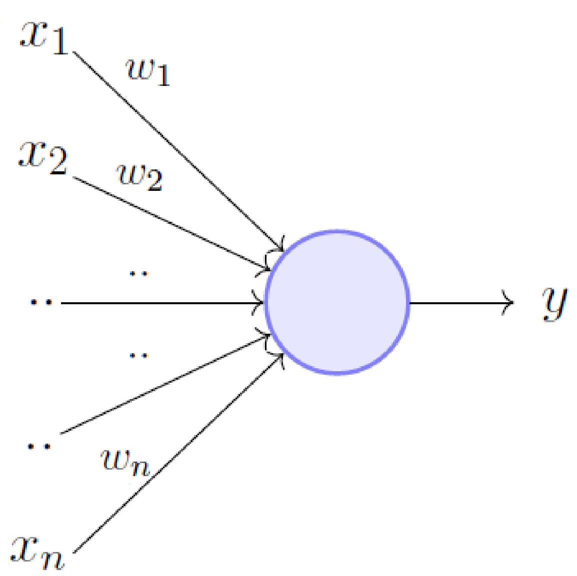
Frank Rosenblatt

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

The *perceptron* model



Perceptron Model (Minsky-Papert in 1969)



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

Rewriting the above,

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

It overcomes some of the limitations of the M-P neuron by introducing

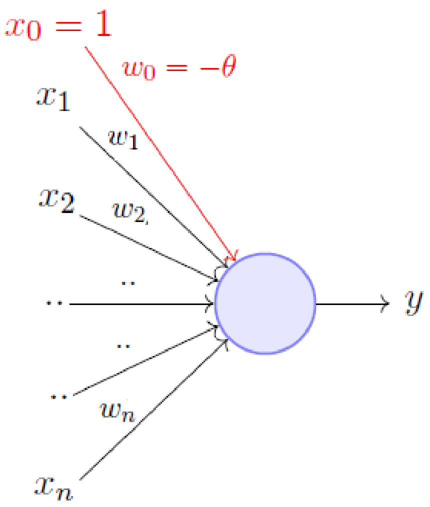
- the concept of numerical weights (a measure of importance) for inputs,
- a mechanism for learning those weights.
- Inputs are no longer limited to boolean values like in the case of an M-P neuron,
- it supports real inputs as well which makes it more useful and generalized.

The perceptron Model:
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 ***theta***, we add it as one of the inputs, with the weight -***theta*** like shown in next slide, which makes it learn-able

Theta add as one of the inputs- theta(θ) is

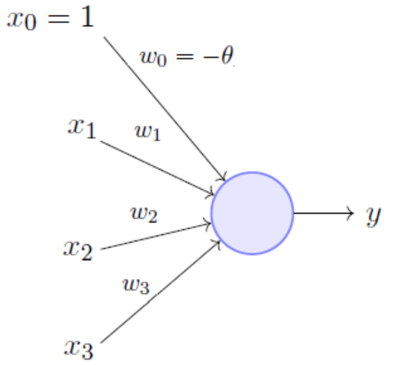
Football watching



A more accepted convention,

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

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



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

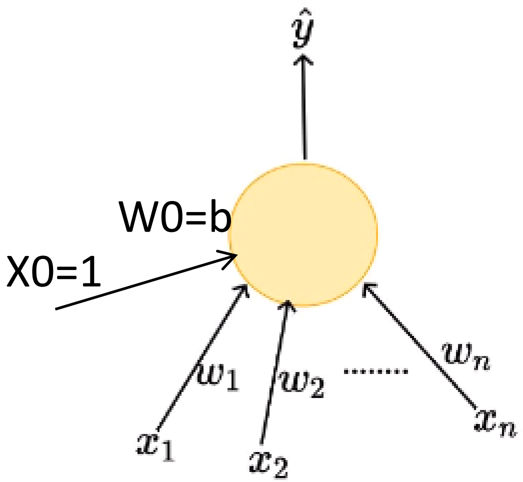
theta, is treated as **bias**, with the weight -***theta*** like shown above, which makes it learn-able

The **weights** and the **bias** will depend on the data (my viewing history in football watching).

- Here, **theta(θ)** 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 may only watch a football game that is a premier league game, featuring Man United game and is not friendly [***theta = 2***].

Perceptron- Frank Rosenblatt

Launch (within 6 months)	0	1	1	0	0	1	0	1	1
Weight	0.19	0.63	0.33	1	0.36	0.66	0	0.70	0.48
Screen size	0.64	0.87	0.67	0.88	0.7	0.91	0	1	0.47
dual sim	1	1	0	0	0	1	0	1	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1
NFC	0	1	1	0	1	0	1	1	1
Radio	1	0	0	1	1	1	0	0	0
Battery	0.36	0.51	0.36	1	0.34	0.67	0	0.57	0.43
Price	0.09	0.63	0.41	0.19	0.06	0	0.72	0.94	1
Like (y)	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**



$$\hat{y} = 1 \text{ if } \sum_{i=1}^{n} w_i x_i \geq b$$

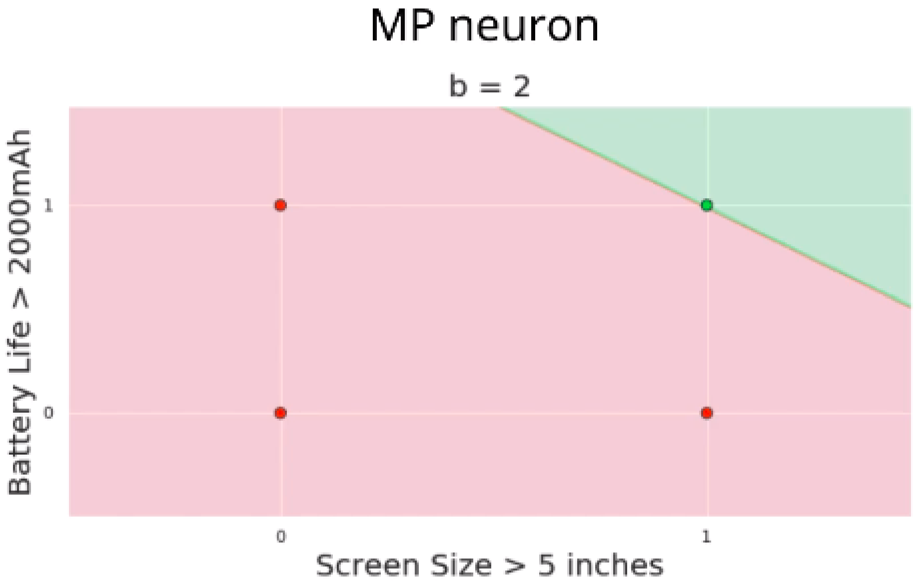
$$\hat{y} = 0 \text{ otherwise}$$

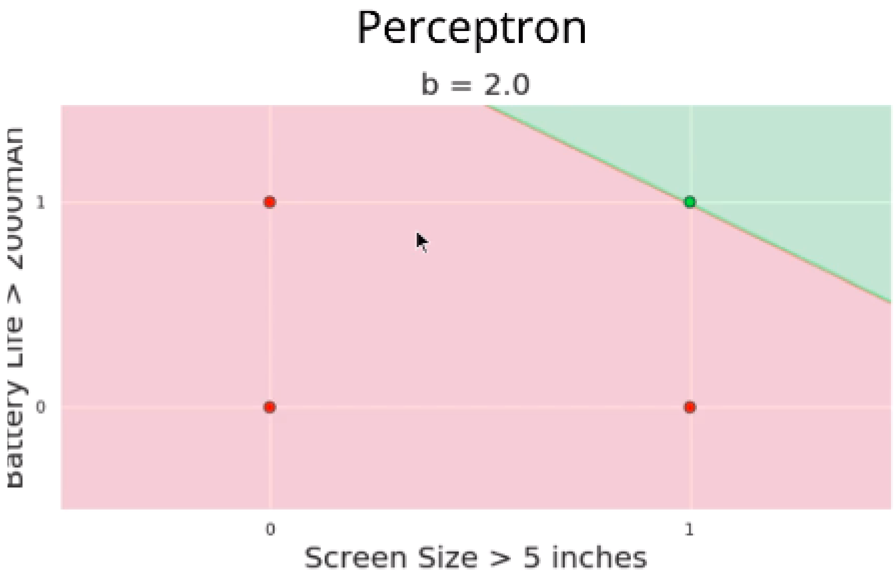
$$\mathbf{x}.\mathbf{w} = x_1.w_1 + x_2.w_2 + ...x_n.w_n \quad = \sum_{i=1}^{n} x_i.w_i$$

- **Real Inputs**
- **Boolean outputs**
- **Weight assigned for each input**
- **Linear**

$$\hat{y} = 1 \ (\text{if } \mathbf{w}.\mathbf{x} \geq 0)$$
$$\hat{y} = 0 \ (\text{otherwise})$$

Geometric Interpretation





$$\hat{y} = x_1 + x_2 \geq b$$

$$x_1 + x_2 - b = 0$$

$$x_2 = mx_1 + c,\ m = -1,\ c = b$$

Fixed Slope

$$w_1x_1 + w_2x_2 \geq b$$

Flexible Slope

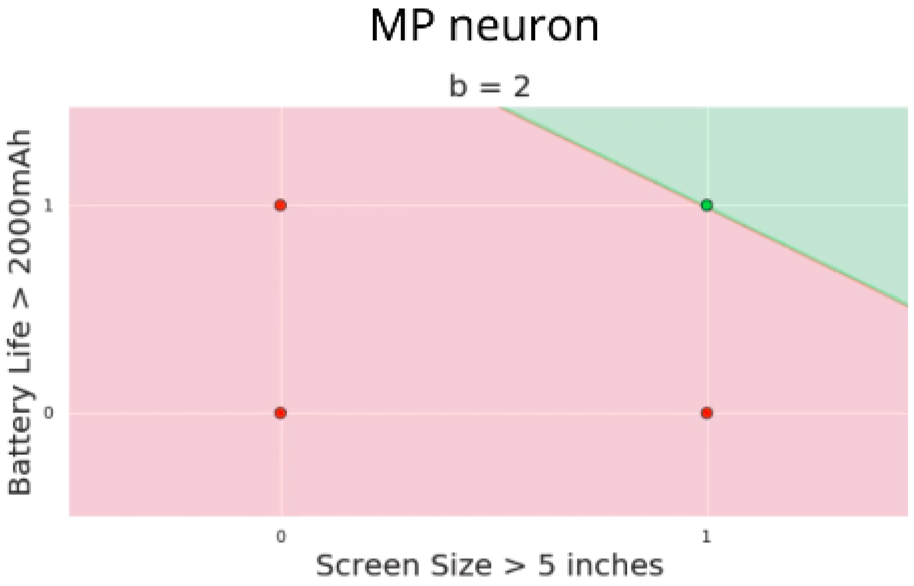
$$w_1x_1 + w_2x_2 - b = 0$$

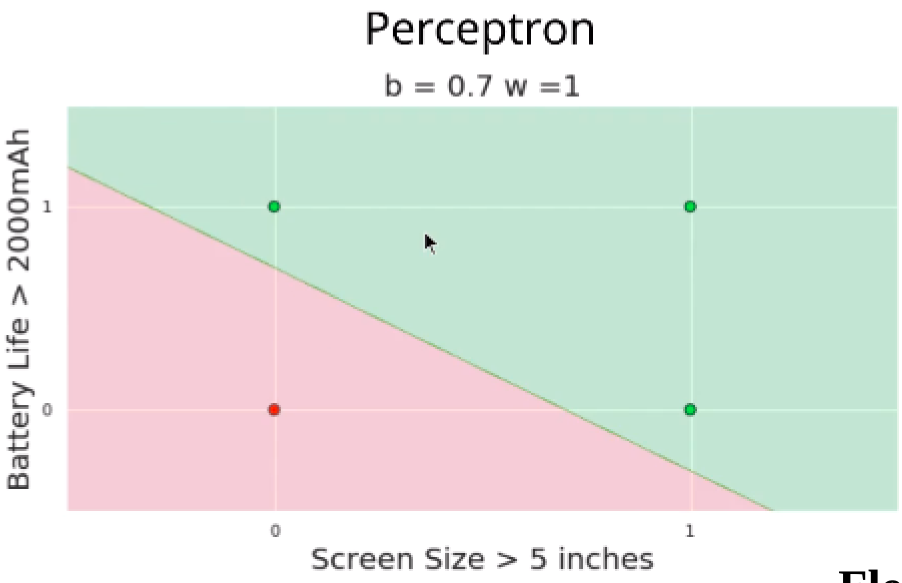
$$x_2 = -\frac{w_1}{w_2}x_1 + \frac{b}{w_2} = 0$$

$$m = -\frac{w_1}{w_2},\ c = \frac{b}{w_2}$$

Courtesy :: Mitesh M. Khapra’s lecture slides

Geometric Interpretation





$$\hat{y} = x_1 + x_2 \geq b$$

$$x_1 + x_2 - b = 0$$

$$x_2 = mx_1 + c,\ m = -1,\ c = b$$

Fixed Slope

Flexible Slope

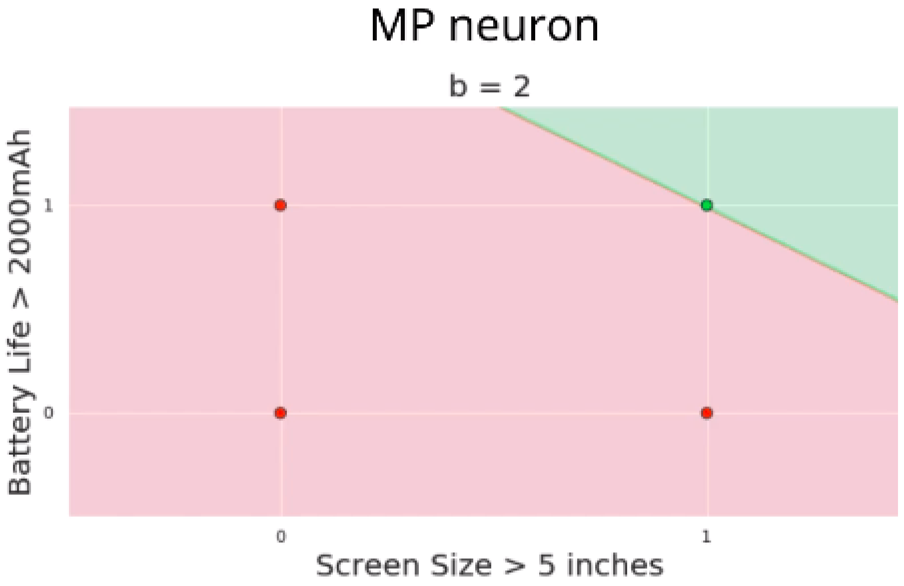
$$w_1x_1 + w_2x_2 \geq b$$

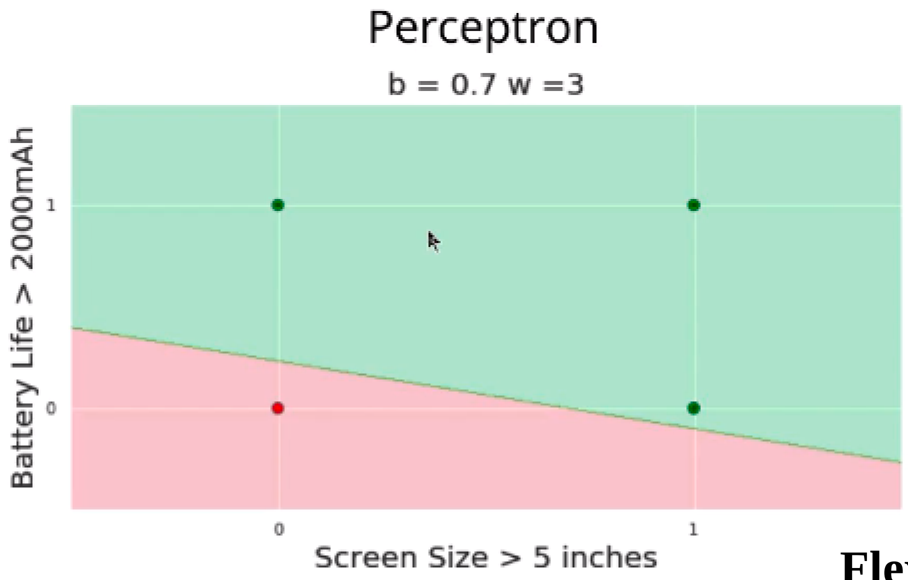
$$w_1x_1 + w_2x_2 - b = 0$$

$$x_2 = -\frac{w_1}{w_2}x_1 + \frac{b}{w_2} = 0$$

$$m = -\frac{w_1}{w_2},\ c = \frac{b}{w_2}$$

Geometric Interpretation





$$\hat{y} = x_1 + x_2 \geq b$$

$$x_1 + x_2 - b = 0$$

$$x_2 = mx_1 + c,\ m = -1,\ c = b$$

Fixed Slope

Flexible Slope

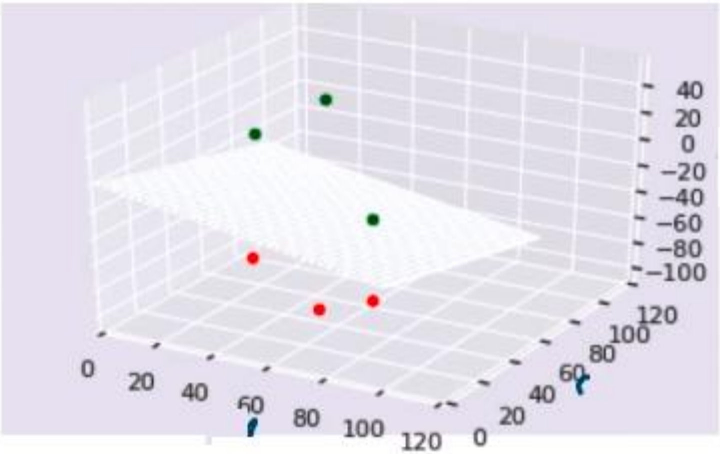
$$w_1 x_1 + w_2 x_2 \geq b$$

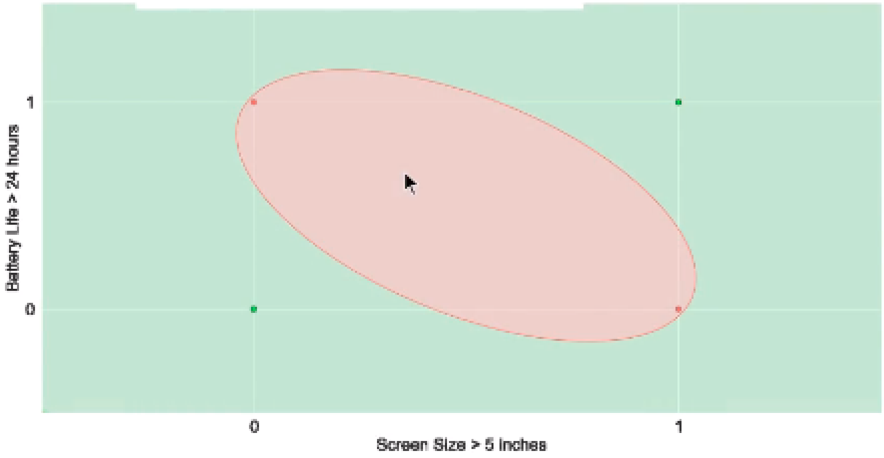
$$w_1 x_1 + w_2 x_2 - b = 0$$

$$x_2 = -\frac{w_1}{w_2} x_1 + \frac{b}{w_2} = 0$$

$$m = -\frac{w_1}{w_2},\ c = \frac{b}{w_2}$$

More dimensions





Boolean Functions Using Perceptron

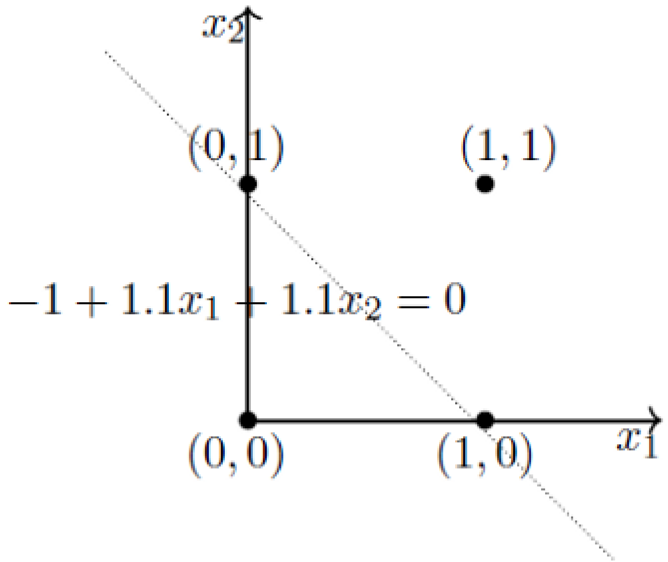
OR Function

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

$$w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0$$
$$w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \implies w_2 > -w_0$$
$$w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \implies w_1 > -w_0$$
$$w_0 + w_1 \cdot 1 + w_2 \cdot 1 \geq 0 \implies w_1 + w_2 > -w_0$$

One possible solution is

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



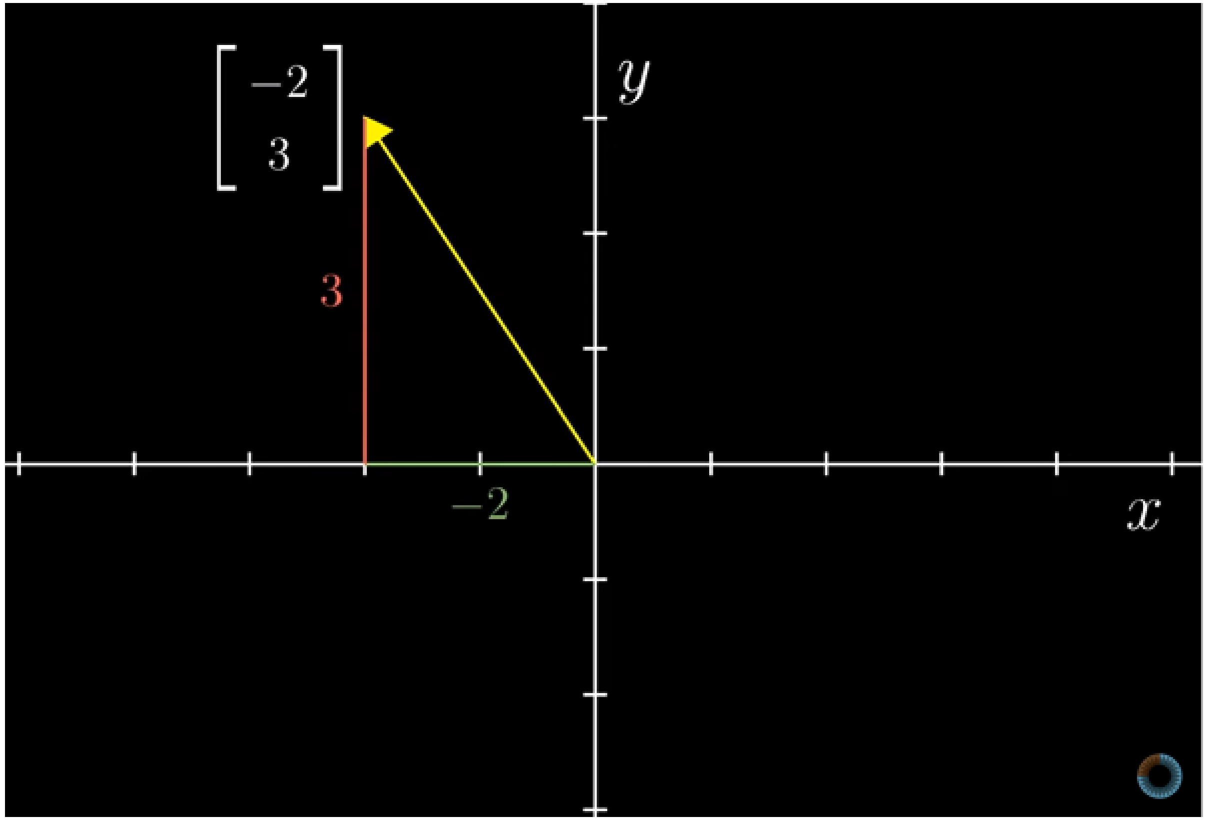
The above ‘possible solution’ was obtained by solving the linear system of equations on the left. It is clear that the solution separates the input space into two spaces, negative and positive half spaces

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, - and perceptron learning algorithm

Basics of Linear Algebra- Vector Visualization

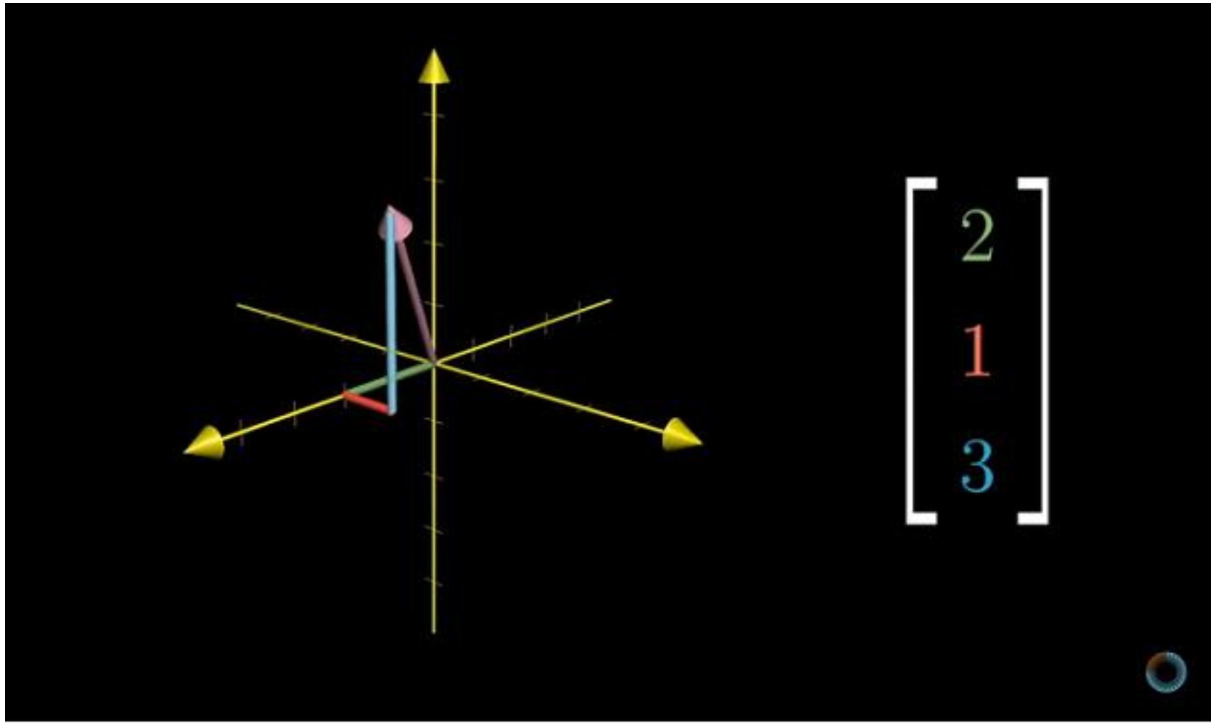
Vector Representations

A 2-dimensional vector can be represented on a 2D plane as follows:



Source: 3Blue1Brown's video on Vectors

3D Vector Representations



Dot Product of 2 vectors

Dot product :Imagine you have two vectors with size **n+1**, **w** and **x**, the dot product of these vectors (***w.x***) could be computed as follows:

$$\mathbf{w} = [w_0, w_1, w_2, ..., w_n]$$

$$\mathbf{x} = [1, x_1, x_2, ..., x_n]$$

$$\mathbf{w} \cdot \mathbf{x} = \mathbf{w}^{\mathrm{T}}\mathbf{x} = \sum_{i=0}^{n} w_i * x_i$$

The transpose is just to write it in a matrix multiplication form.

Here, **w** and **x** are just two lonely arrows in an **n+1 dimensional** space

Intuitively, their dot product quantifies how much one vector is going in the direction of the other.

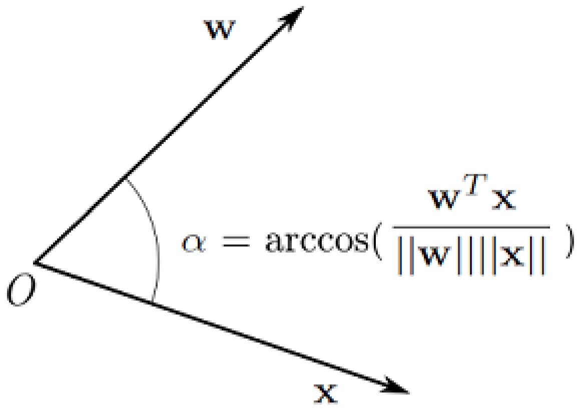
So technically, the perceptron was only computing a lame dot product (before checking if it's greater or lesser than 0).

The decision boundary line which a perceptron gives out that separates positive examples from the negative ones is really just **w . x** = 0.

Angle Between Two Vectors

$$\mathbf{w}^T\mathbf{x} = \|\mathbf{w}\|\|\mathbf{x}\|\, cos\alpha$$

$$cos\alpha = \frac{\mathbf{w}^T\mathbf{x}}{||\mathbf{w}||||\mathbf{x}||}$$



When we say that the cosine of the angle between **w** and **x** is 0,

arrow **w** is being perpendicular to arrow **x** in an n+1 dimensional space (in 2-dimensional space).

So when the dot product of two vectors is 0, they are perpendicular to each other.

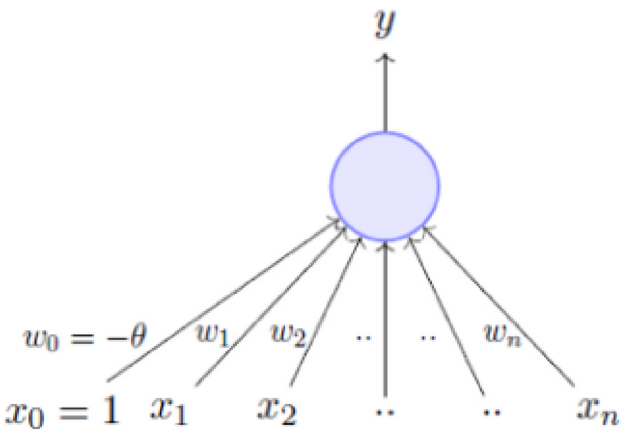
you can get the angle between two vectors, if only you knew the vectors, given you know how to calculate vector magnitudes and their dot products

$$cos\alpha = \frac{\mathbf{w}^T\mathbf{x}}{||\mathbf{w}||||\mathbf{x}||} \quad \Big| \quad cos\alpha \propto \mathbf{w}^T\mathbf{x}$$

$$\text{So if } \mathbf{w}^T\mathbf{x} > 0 \quad \Rightarrow cos\alpha > 0 \quad \Rightarrow \alpha < 90$$

$$\text{Similarly, if } \mathbf{w}^T\mathbf{x} < 0 \quad \Rightarrow cos\alpha < 0 \quad \Rightarrow \alpha > 90$$

Perceptron Learning Algorithm- setting up the problem



$$
\begin{aligned}
x_1 &= isActorDamon \\
x_2 &= isGenreThriller \\
x_3 &= isDirectorNolan \\
x_4 &= imdbRating(scaled\ to\ 0\ to\ 1) \\
\ldots &\quad \ldots \\
x_n &= criticsRating(scaled\ to\ 0\ to\ 1)
\end{aligned}
$$

We are going to use a perceptron to estimate if I will be watching a movie based on historical data with the above-mentioned inputs. The data has positive and negative examples, positive being the movies I watched i.e., 1. Based on the data, we are going to learn the weights using the perceptron learning algorithm. For visual simplicity, we will only assume two-dimensional input.

Learning Algorithm

Our goal is to find the **w** vector that can perfectly classify positive inputs and negative inputs in our data

Initialise w_1, w_2, b

Iterate over data:

$$\mathscr{L} = compute\_loss(x_i)$$

$$update(w_1, w_2, b, \mathscr{L})$$

till satisfied

Weight	Screen size	Like
0.19	0.64	1
0.63	0.87	1
0.33	0.67	0
1	0.88	0

$$\hat{y} = 1 \text{ (if } \mathbf{w}.\mathbf{x} \geq 0)$$
$$\hat{y} = 0 \text{ (otherwise)}$$

Perceptron Learning Algorithm

P- positive examples N – Negative examples

Algorithm: Perceptron Learning Algorithm

$P \leftarrow$ *inputs with label* 1;

$N \leftarrow$ *inputs with label* 0;

Initialize **w** randomly;

$$\hat{y} = 1 \ (\text{if } \textstyle\sum_{i=0}^{n} w_i x_i \geq 0)$$
$$\hat{y} = 0 \ (\text{otherwise})$$

$$\hat{y} = 1 \ (\text{if } \mathbf{w}.\mathbf{x} \geq 0)$$
$$\hat{y} = 0 \ (\text{otherwise})$$

$$\boldsymbol{w} = \begin{bmatrix} W_1 \\ W_2 \\ W_3 \\ W_0 \end{bmatrix} \qquad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ b \end{bmatrix}$$

Perceptron Learning Algorithm

Algorithm: Perceptron Learning Algorithm

$P \leftarrow$ *inputs with label* 1;

$N \leftarrow$ *inputs with label* 0;

Initialize **w** randomly;

while !*convergence* **do**

end

//the algorithm converges when all the inputs are classified correctly

$$\hat{y} = 1 \text{ (if } \sum_{i=0}^{n} w_i x_i \geq 0)$$
$$\hat{y} = 0 \text{ (otherwise)}$$

$$\hat{y} = 1 \text{ (if } \mathbf{w}.\mathbf{x} \geq 0)$$
$$\hat{y} = 0 \text{ (otherwise)}$$

$$\boldsymbol{w} = \begin{bmatrix} W_1 \\ W_2 \\ W_3 \\ W_0 \end{bmatrix} \qquad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ b \end{bmatrix}$$

Perceptron Learning Algorithm

Algorithm: Perceptron Learning Algorithm

$P \leftarrow$ *inputs with label* 1;
$N \leftarrow$ *inputs with label* 0;
Initialize **w** randomly;
while !*convergence* **do**
    Pick random $\mathbf{x} \in P \cup N$;
    if $\mathbf{x} \in P$ *and* $\sum_{i=0}^{n} w_i * x_i < 0$ **then**
        $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
    end
    if $\mathbf{x} \in N$ *and* $\sum_{i=0}^{n} w_i * x_i \geq 0$ **then**
        $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
    end
end
//the algorithm converges when all the inputs are classified correctly

(P U N) both positive and negative examples

$$\hat{y} = 1 \ (\text{if } \textstyle\sum_{i=0}^{n} w_i x_i \geq 0)$$
$$\hat{y} = 0 \ (\text{otherwise})$$

$$\hat{y} = 1 \ (\text{if } \mathbf{w}.\mathbf{x} \geq 0)$$
$$\hat{y} = 0 \ (\text{otherwise})$$

$$\boldsymbol{w} = \begin{bmatrix} W_1 \\ W_2 \\ W_3 \\ W_0 \end{bmatrix} \qquad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ b \end{bmatrix}$$

Case 1: When **x** belongs to P and its dot product **w.x** < 0
Case 2: When **x** belongs to N and its dot product **w.x** ≥ 0

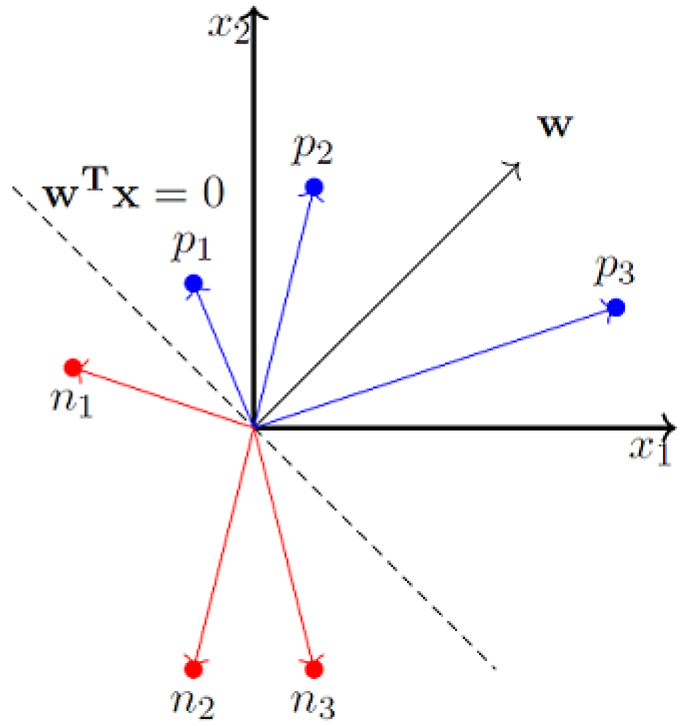
Intuition behind perceptron learning Algorithm

when **x** belongs to *P*, we want **w.x** > 0, basic perceptron rule. What we also mean by that is when **x** belongs to *P*, the angle between **w** and **x** should be less than 90 degrees. because the cosine of the angle is proportional to the dot product.

$$cos\alpha = \frac{\mathbf{w}^T\mathbf{x}}{||\mathbf{w}||||\mathbf{x}||} \quad \Bigg| \quad cos\alpha \propto \mathbf{w}^T\mathbf{x}$$

$$\text{So if } \mathbf{w}^T\mathbf{x} > 0 \quad \Rightarrow cos\alpha > 0 \quad \Rightarrow \alpha < 90$$

$$\text{Similarly, if } \mathbf{w}^T\mathbf{x} < 0 \quad \Rightarrow cos\alpha < 0 \quad \Rightarrow \alpha > 90$$



whatever the **w** vector may be, as long as it makes an angle less than 90 degrees with the positive example data vectors (**x** E *P*) and an angle more than 90 degrees with the negative example data vectors (**x** E *N*), we are cool. So ideally, it should look something like the figure on left←-:

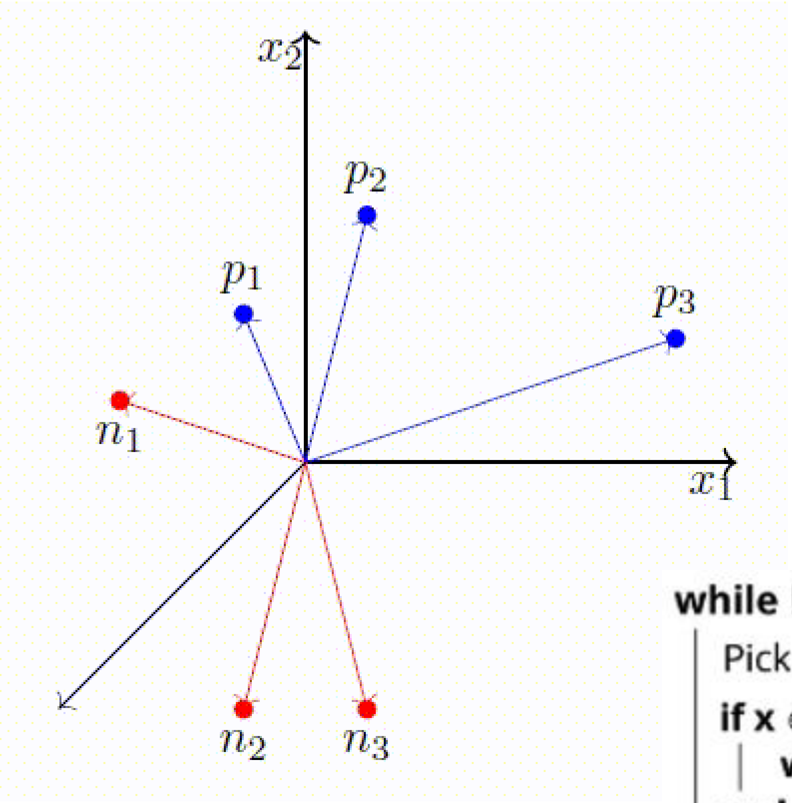
Intuition behind perceptron learning Algorithm

$$(\alpha_{new}) \text{ when } \mathbf{w_{new}} = \mathbf{w} + \mathbf{x}$$

$$\begin{aligned} cos(\alpha_{new}) &\propto \mathbf{w_{new}}^T\mathbf{x} \\ &\propto (\mathbf{w} + \mathbf{x})^T\mathbf{x} \\ &\propto \mathbf{w}^T\mathbf{x} + \mathbf{x}^T\mathbf{x} \\ &\propto cos\alpha + \mathbf{x}^T\mathbf{x} \\ cos(\alpha_{new}) &> cos\alpha \end{aligned}$$

$$(\alpha_{new}) \text{ when } \mathbf{w_{new}} = \mathbf{w} - \mathbf{x}$$

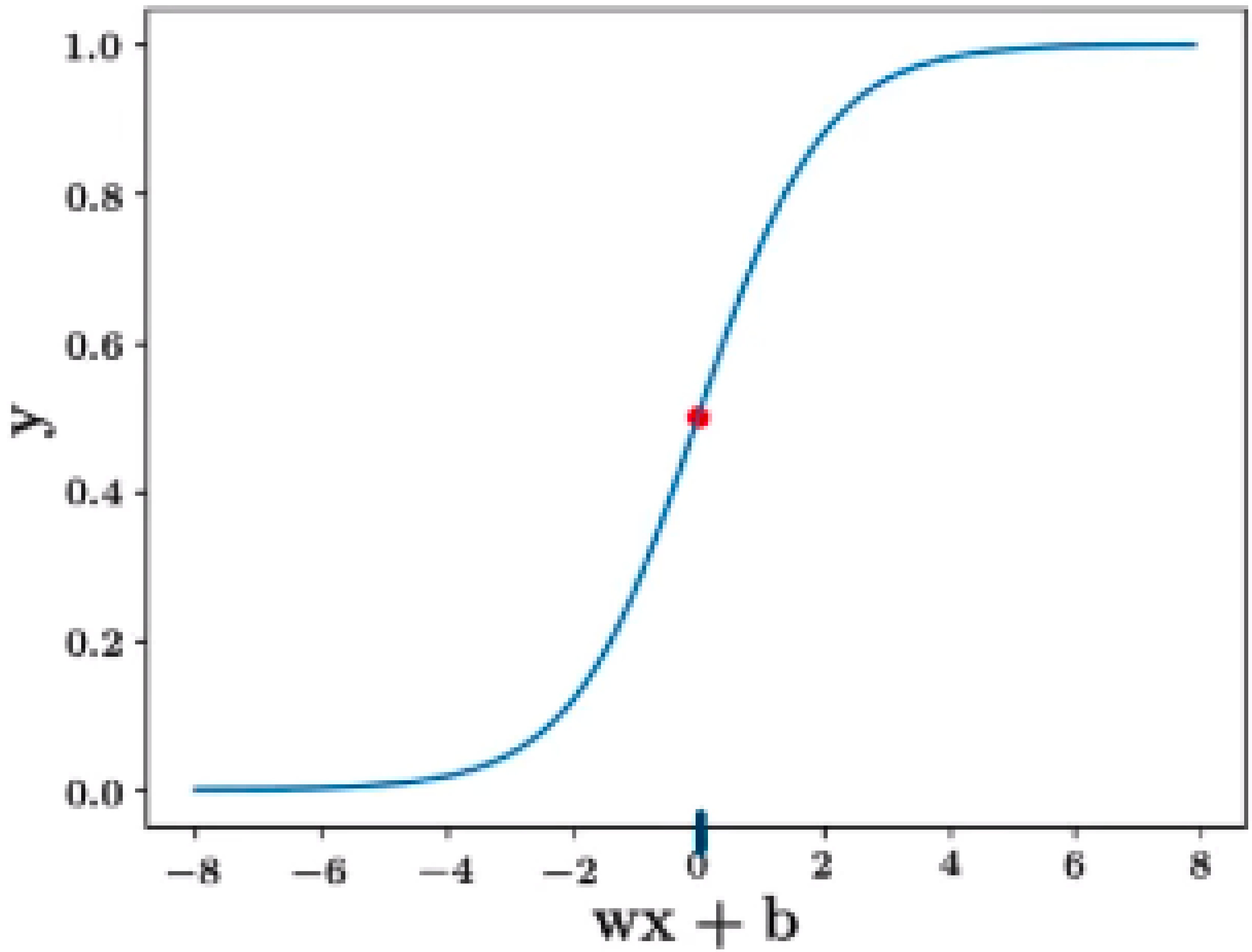
$$\begin{aligned} cos(\alpha_{new}) &\propto \mathbf{w_{new}}^T\mathbf{x} \\ &\propto (\mathbf{w} - \mathbf{x})^T\mathbf{x} \\ &\propto \mathbf{w}^T\mathbf{x} - \mathbf{x}^T\mathbf{x} \\ &\propto cos\alpha - \mathbf{x}^T\mathbf{x} \\ cos(\alpha_{new}) &< cos\alpha \end{aligned}$$



So when we are adding **x** to **w**, which we do when x belongs to P and **w.x** < 0 (Case 1), we are essentially **increasing the *cos(alpha)*** value, which means, we are **decreasing the *alpha* value**, the angle between **w** and **x**, **which is what we desire**. And the similar intuition works for the case when **x** belongs to *N* and **w.x** ≥ 0 (Case 2).

while !*convergence* **do**
  Pick random $\mathbf{x} \in P \cup N$;
  if $\mathbf{x} \in P$ *and* $\sum_{i=0}^{n} w_i * x_i < 0$ **then**
    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
  end
  if $\mathbf{x} \in N$ *and* $\sum_{i=0}^{n} w_i * x_i \geq 0$ **then**
    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
  end
end

$$\hat{y} = 1 \text{ (if } \mathbf{w.x} \geq 0)$$
$$\hat{y} = 0 \text{ (otherwise)}$$


1.0
0.8
0.6
0.4
0.2
0.0
y
−8
−6
−4
−2
0
2
4
6
8
wx + b

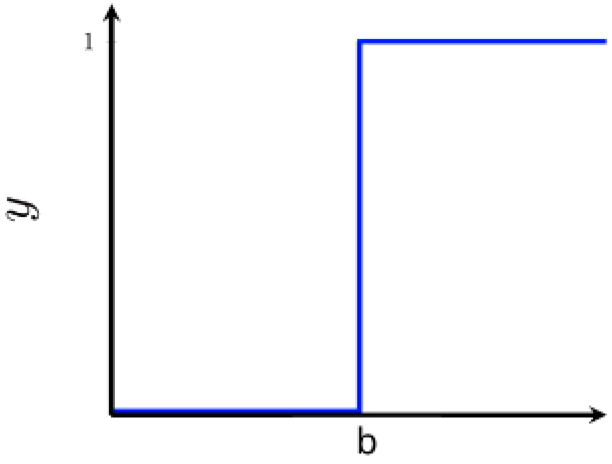
Sigmoid Neuron-

A smoother activation function

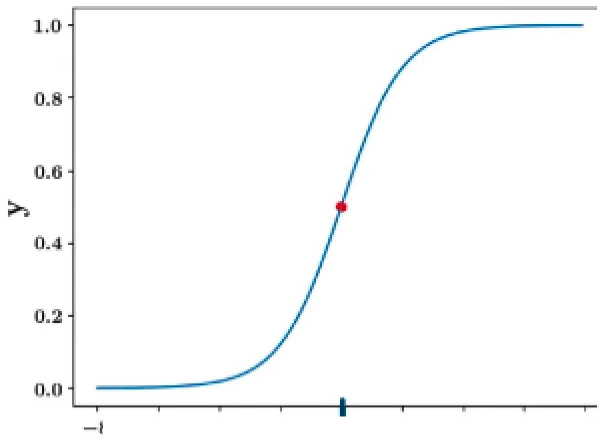
The Sigmoid Function curve looks like a S-shape.

The function is **differentiable**.That means, we can find the slope of the sigmoid curve at any two points. It is **monotonic**

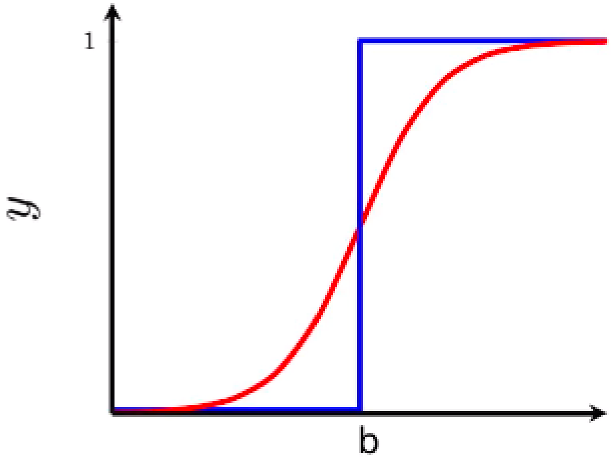
It exists between **(0 to 1).** especially used for models where we have to predict the probability as an output.



. $\sum_{i=1}^{n} w_i x_i$



$$y = \frac{1}{1 + e^{-(wx+b)}}$$



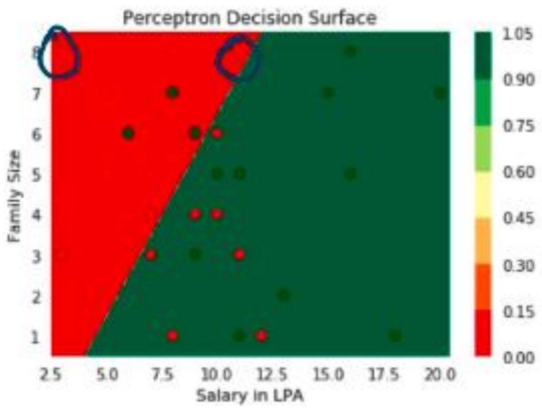
. $\sum_{i=1}^{n} w_i x_i$

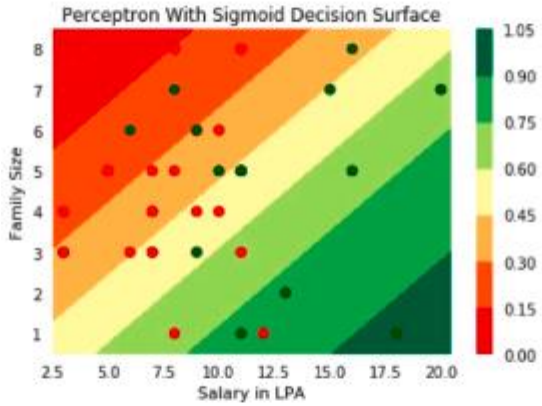
So the output is σ(w·x+b), where σ is called the Sigmoid Function and is def by:- $\sigma(z) \equiv \frac{1}{1+e^{-z}}.$

A monotonic function is a **function which is either entirely nonincreasing or nondecreasing.**

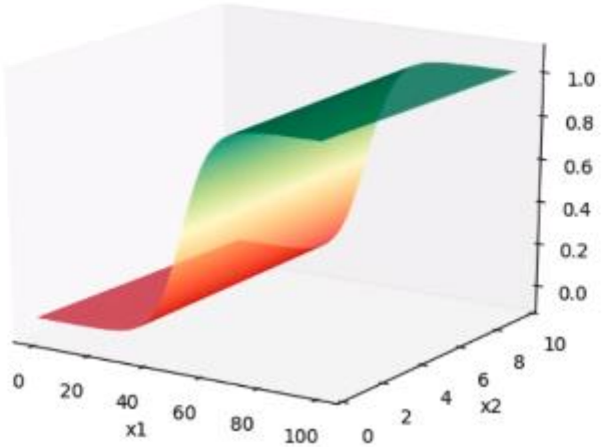
Sigmoid (cont.)

$$y = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2 + b)}}$$

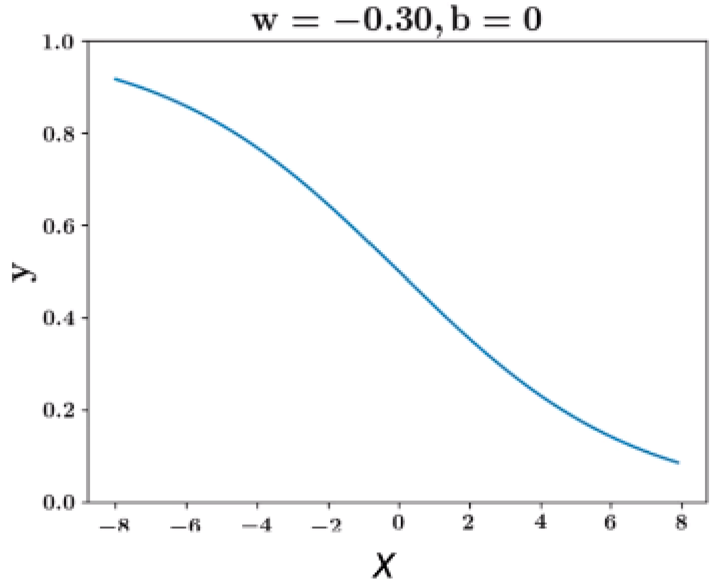




	Salary in LPA	Family Size	Buys Car?
0	11	8	1
1	20	7	1
2	4	8	0
3	8	7	0
4	11	5	1



Loss Function-Sigmoid

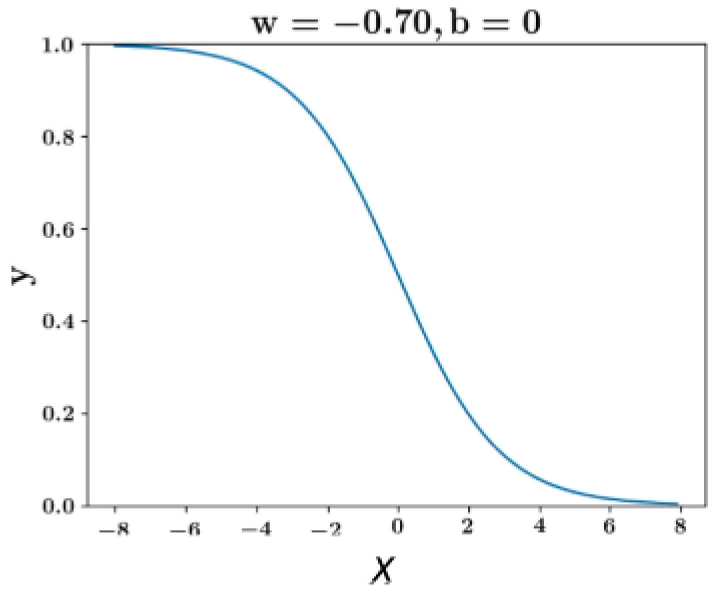


$$\hat{y} = \frac{1}{1 + e^{-(wx+b)}}$$

$$Loss = \sum_{i=1}^{4} (y - \hat{y})^2 = 0.18$$

x_1	x_2	y	$\hat{y}$
1	1	0.5	0.6
2	1	0.8	0.7
1	2	0.2	0.2
2	2	0.9	0.5

Loss Function-Sigmoid

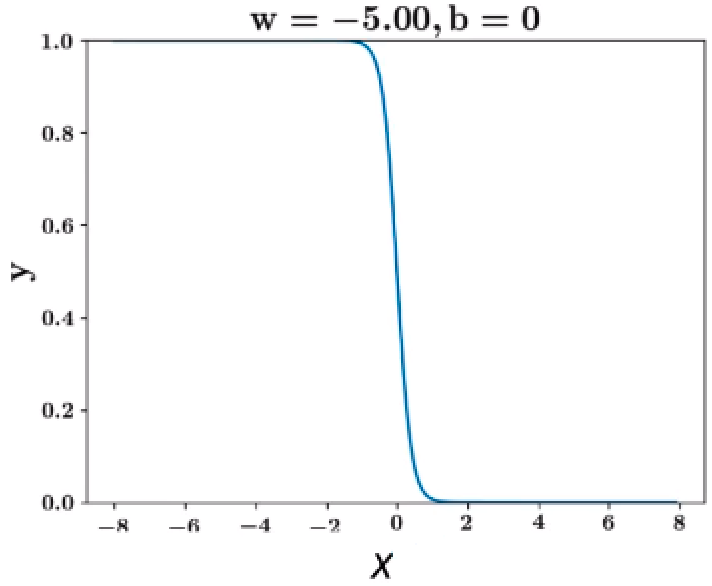


$$\hat{y} = \frac{1}{1 + e^{-(wx+b)}}$$

$$Loss = \sum_{i=1}^{4} (y - \hat{y})^2 = 0.18$$

x_1	x_2	y	$\hat{y}$
1	1	0.5	0.6
2	1	0.8	0.7
1	2	0.2	0.2
2	2	0.9	0.5

Loss Function-Sigmoid

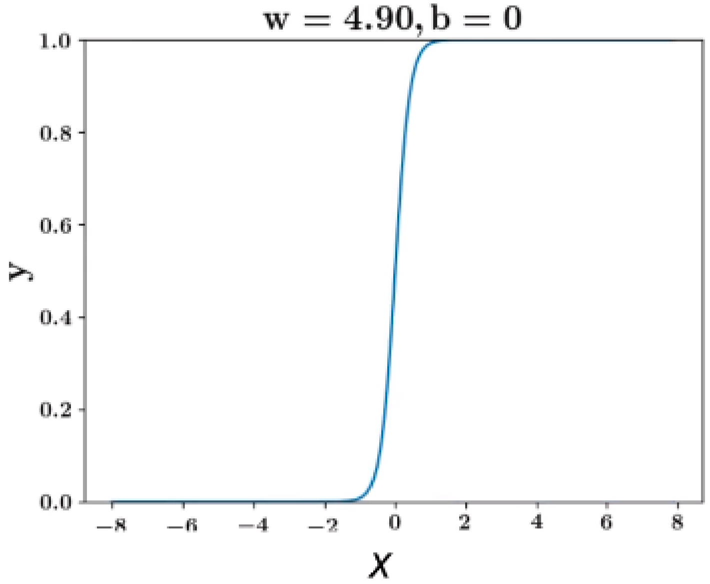


$$\hat{y} = \frac{1}{1 + e^{-(wx+b)}}$$

$$Loss = \sum_{i=1}^{4} (y - \hat{y})^2 = 0.18$$

x_1	x_2	y	$\hat{y}$
1	1	0.5	0.6
2	1	0.8	0.7
1	2	0.2	0.2
2	2	0.9	0.5

Loss Function-Sigmoid

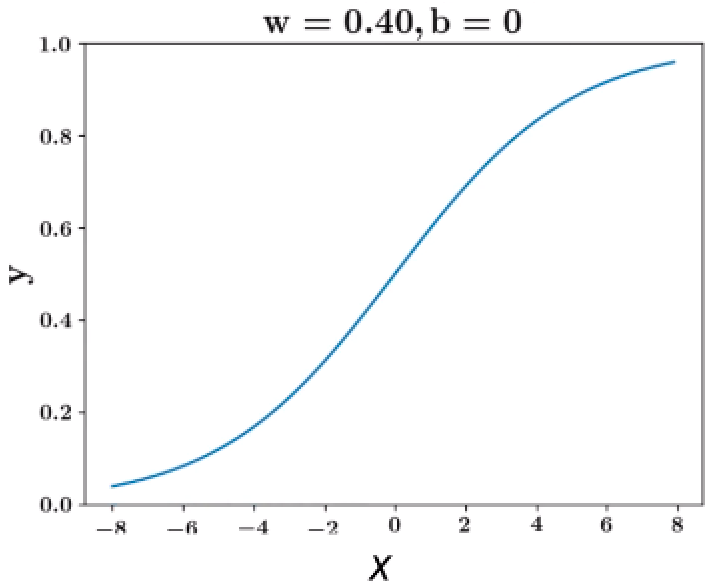


$$\hat{y} = \frac{1}{1 + e^{-(wx+b)}}$$

$$Loss = \sum_{i=1}^{4} (y - \hat{y})^2 = 0.18$$

x_1	x_2	y	$\hat{y}$
1	1	0.5	0.6
2	1	0.8	0.7
1	2	0.2	0.2
2	2	0.9	0.5

Loss Function-Sigmoid

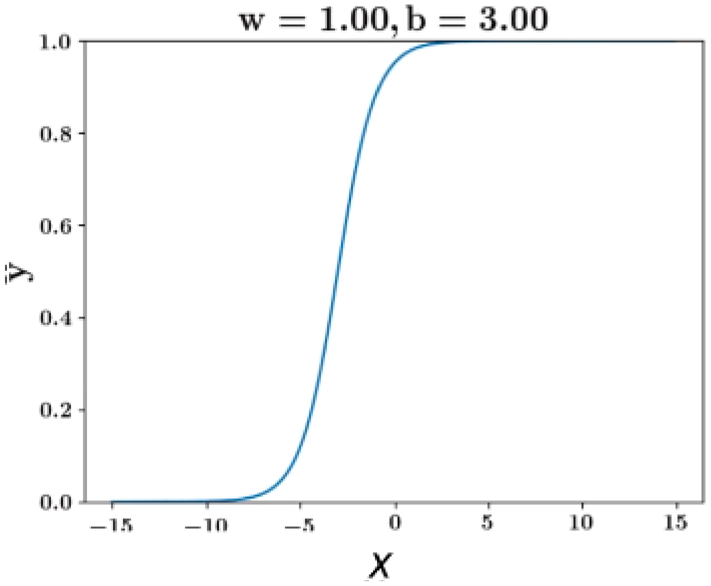


$$\hat{y} = \frac{1}{1 + e^{-(wx+b)}}$$

$$Loss = \sum_{i=1}^{4} (y - \hat{y})^2 = 0.18$$

x_1	x_2	y	$\hat{y}$
1	1	0.5	0.6
2	1	0.8	0.7
1	2	0.2	0.2
2	2	0.9	0.5

Loss Function-Sigmoid

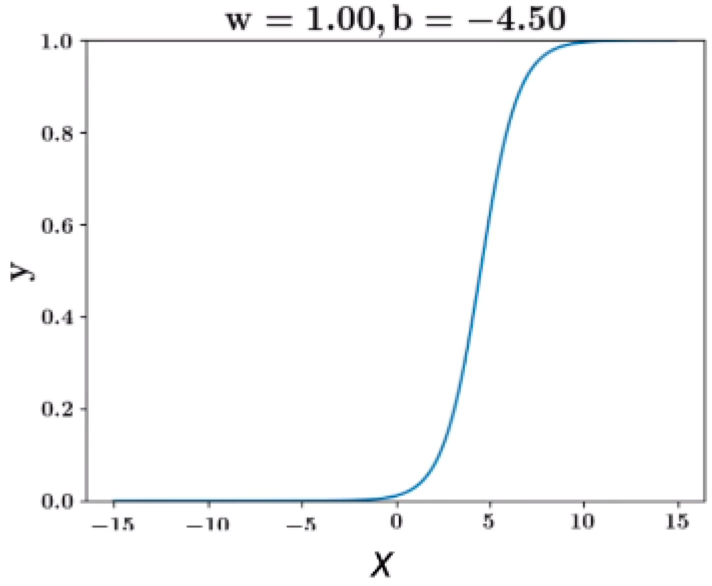


$$\hat{y} = \frac{1}{1 + e^{-(wx+b)}}$$

$$Loss = \sum_{i=1}^{4} (y - \hat{y})^2 = 0.18$$

x_1	x_2	y	$\hat{y}$
1	1	0.5	0.6
2	1	0.8	0.7
1	2	0.2	0.2
2	2	0.9	0.5

Loss Function-Sigmoid

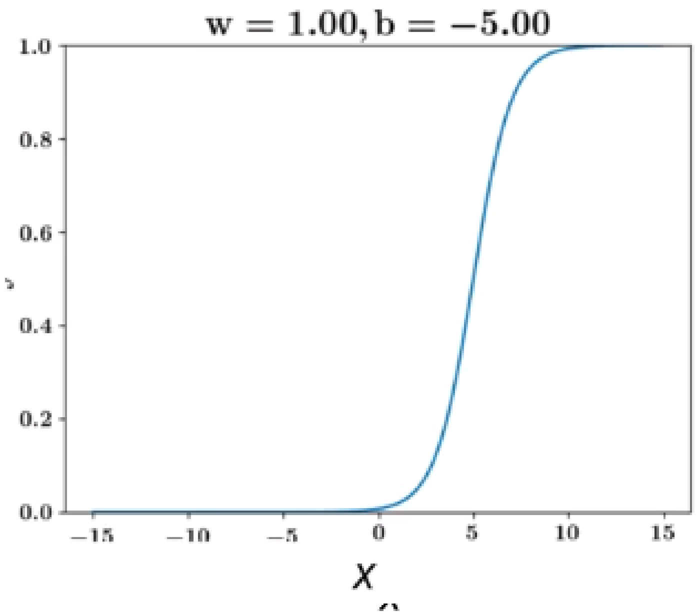


$$\hat{y} = \frac{1}{1 + e^{-(wx+b)}}$$

$$Loss = \sum_{i=1}^{4}(y - \hat{y})^2 = 0.18$$

x_1	x_2	y	$\hat{y}$
1	1	0.5	0.6
2	1	0.8	0.7
1	2	0.2	0.2
2	2	0.9	0.5

Loss Function-Sigmoid



$$\hat{y} = \frac{1}{1 + e^{-(wx+b)}}$$

$$Loss = \sum_{i=1}^{4} (y - \hat{y})^2 = 0.18$$

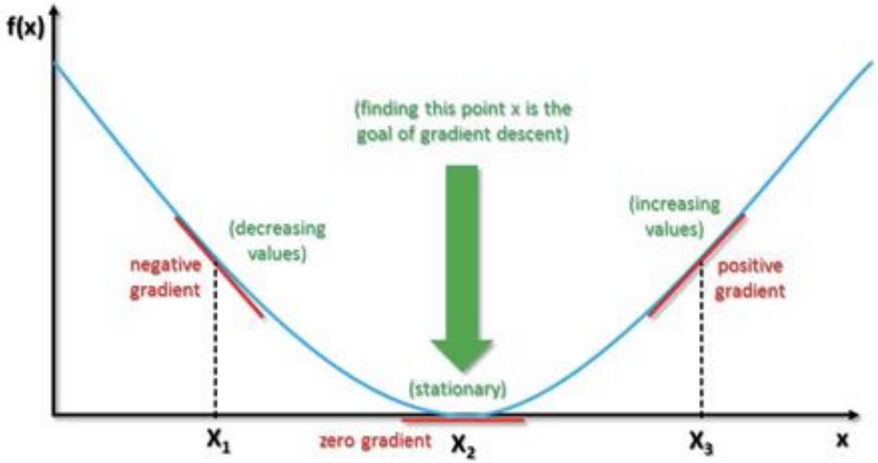
x_1	x_2	y	$\hat{y}$
1	1	0.5	0.6
2	1	0.8	0.7
1	2	0.2	0.2
2	2	0.9	0.5

Gradient Descent

Parameter Update rule

$$w_{t+1} = w_t - \eta \Delta w_t$$
$$b_{t+1} = b_t - \eta \Delta b_t$$

$$where\ \Delta w_t = \frac{\partial \mathscr{L}(w,b)}{\partial w}_{at\ w=w_t, b=b_t}, \Delta b_t = \frac{\partial \mathscr{L}(w,b)}{\partial b}_{at w=w_t, b=b_t}$$



$$\min_{w,b} \quad Loss\mathscr{L}(w,b)$$

Initialise w, b

Iterate over data:

$compute\ \hat{y}$

$compute\ \mathscr{L}(w,b)$

$w_{t+1} = w_t - \eta \Delta w_t$

$b_{t+1} = b_t - \eta \Delta b_t$

till satisfied

I/P	O/P
3	0.268
4	0.73
5	0.952
6	0.994
8	0.999

$$\hat{y} = \frac{1}{1 + e^{-(wx+b)}}$$

$$Loss \mathscr{L}(w,b) = \sum_{i=1}^{5} (y_i - \hat{y}_i)^2$$

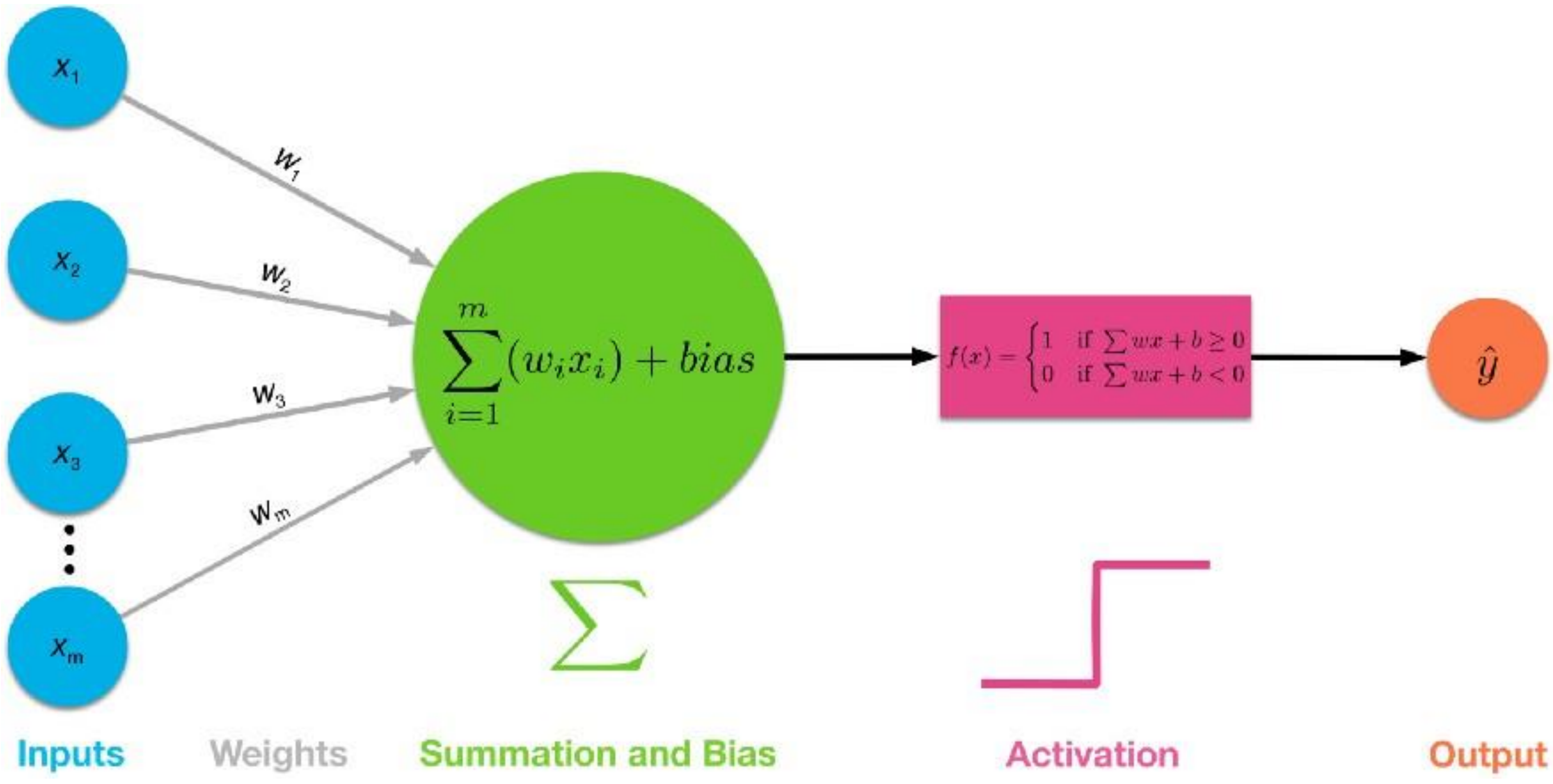
$$\mathscr{L} = \frac{1}{5} \sum_{i=1}^{i=5} (f(x_i) - y_i)^2$$

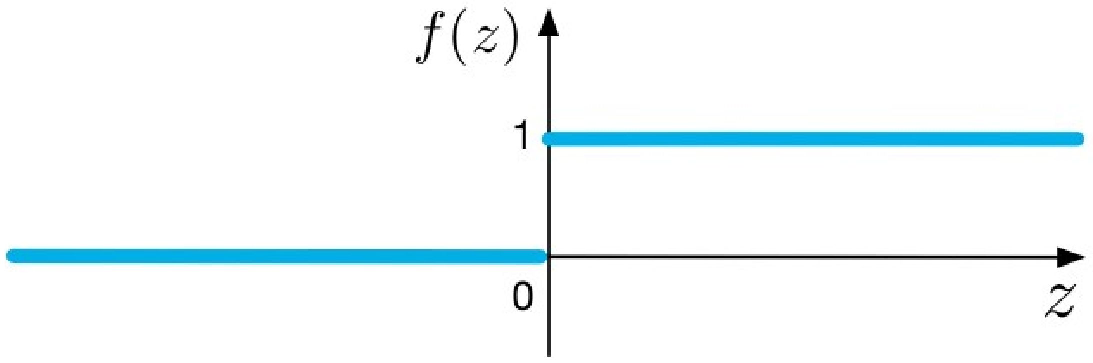
$$\frac{\partial \mathscr{L}}{\partial w} = \frac{\partial}{\partial w} [\frac{1}{5} \sum_{i=1}^{i=5} (f(x_i) - y_i)]^2$$

$$\Delta w = \frac{\partial \mathscr{L}}{\partial w} = \frac{1}{5} \sum_{i=1}^{i=5} \frac{\partial}{\partial w} (f(x_i) - y_i)^2$$

$$(f(x) - y) * f(x) * (1 - f(x)) * x$$

Procedures of a Single-layer Perceptron Network





Step -Activation Function

(Can be sigmoid as well)

Courtesy: towardsdatascience.com

References

Namah Shivaya