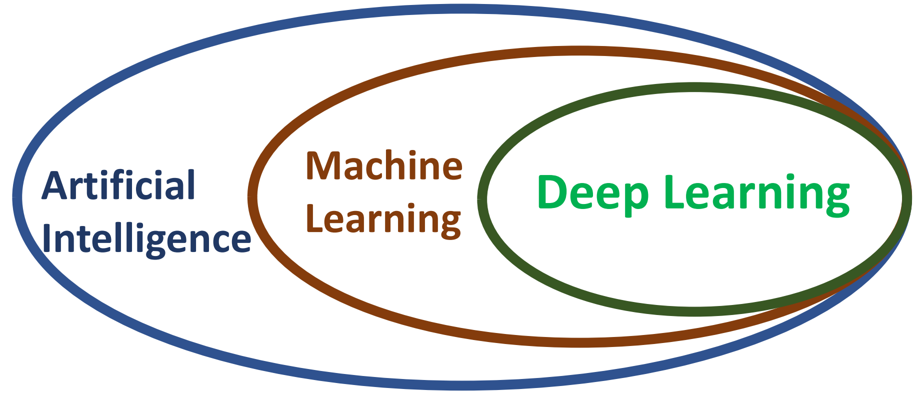
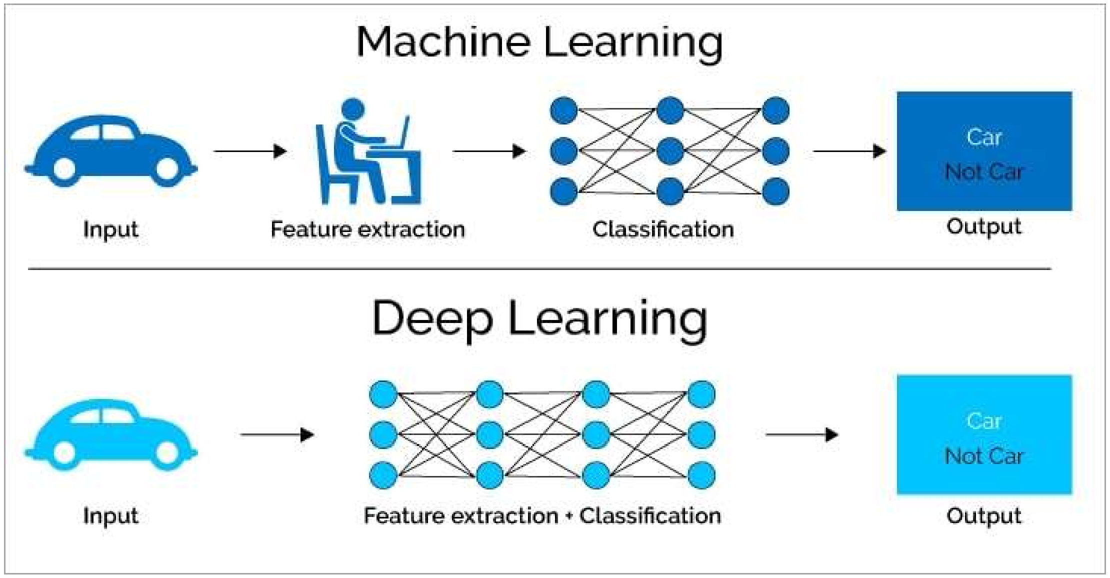
Perceptrons

Building Blocks of complex neural networks

Deep Learning



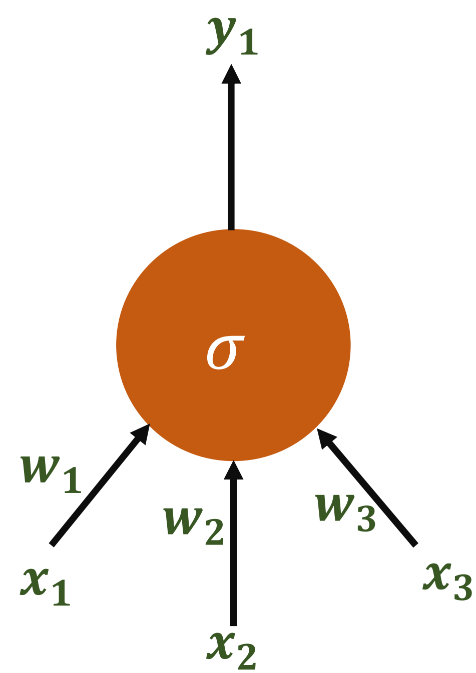
- Deep Learning is a specialized subset of Machine learning
- Relies on a layered structure of algorithms called as Artificial Neural Network
- Deep learning models require a large amount of data to train, but requires little human intervention to function properly



The Deep Learning algorithm doesn't need a software engineer to identify features but is capable of automatic feature engineering through its neural network.
(Source: softwaretestinghelp.com)

Artificial Neuron

- Neural networks are a set of algorithms that have been developed to imitate the human brain in the way we identify patterns
- The most fundamental unit of a Deep Neural Network is called an artificial Neuron
- The inspiration for the neuron comes from our understanding on biological neurons, also called neural processing units



Artificial Neuron

Biological Neurons

Average human brain is estimated to have around 10^{11} neurons.

- Neurons takes an input signal i.e., receives signals from other neurons through their **dendrites**
- **Soma** helps in the processing of the information in the neuron cell body
- The output/information is passed of the neuron to other neurons through **axon**
- Synapse is the point of connection between the axon branches and other neurons' dendrites

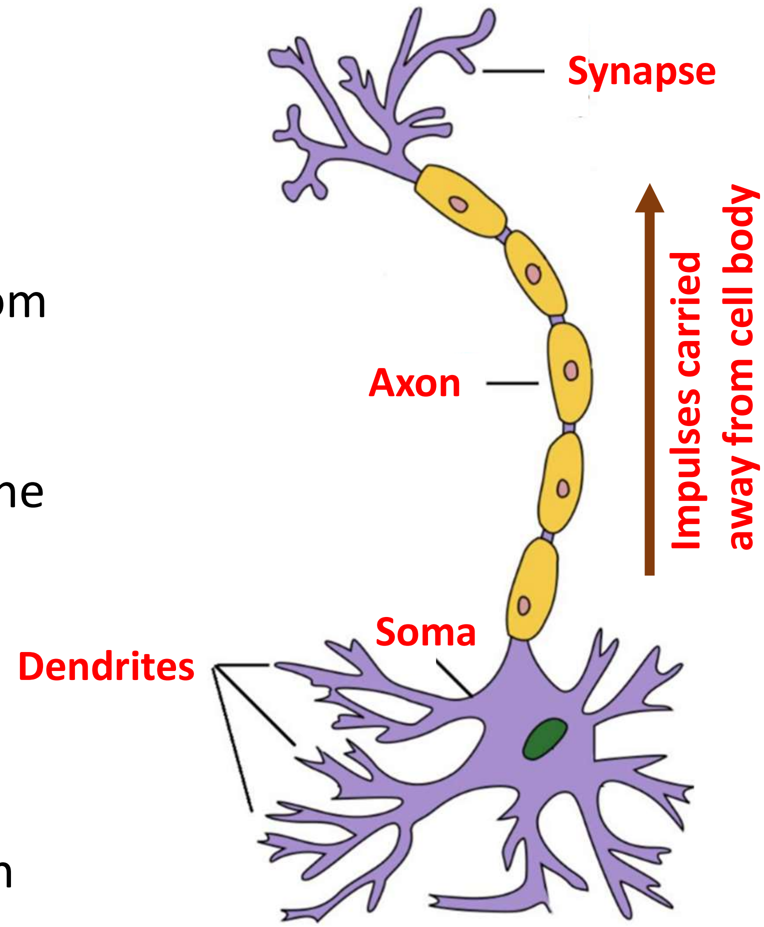


Image Source: https://maelfabien.github.io/

Neurons (Biological vs Artificial)

- Firing Rates of different input neurons combine to influence the firing rate of other neurons
- The activation corresponds to a “sort of” firing rate
- The weights between neurons model whether neurons excite or inhibit each other
- The activation function and bias model the thresholded behaviour of action potentials

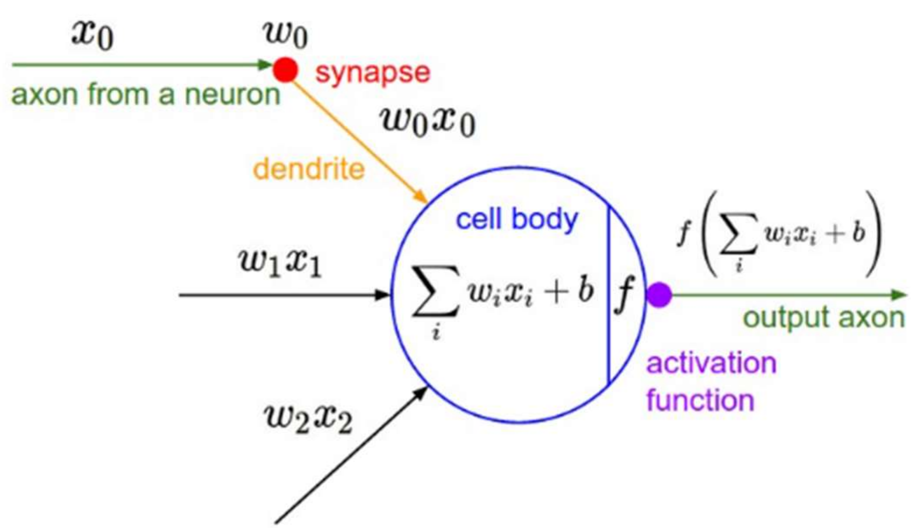


Image Source: https://www.cs.toronto.edu/~lczhang/aps360_20191/lec/w02/term

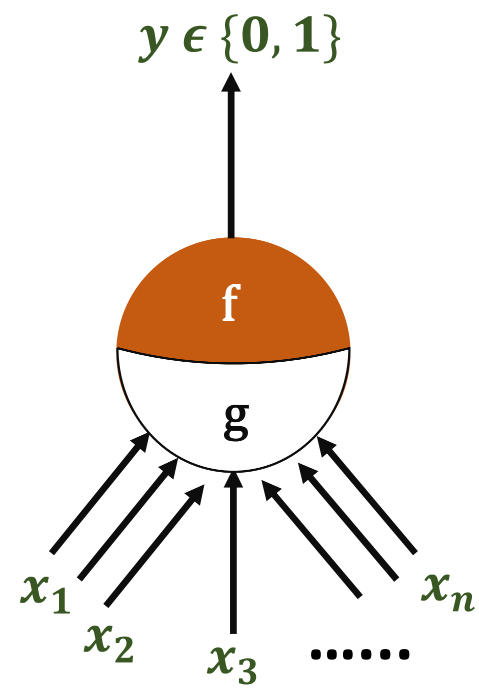
McCulloch-Pitts Neuron

McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified computational model of the neuron (1943)

The output y = 0, if any x_i is inhibitory, otherwise

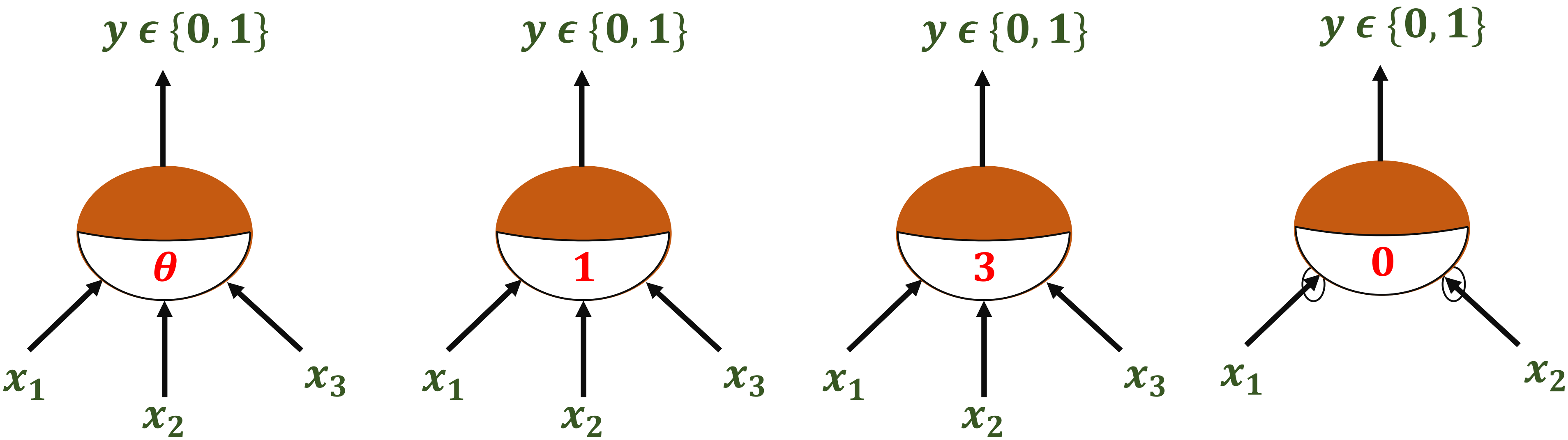
$$\begin{aligned} y = f\big(g(x)\big) &= 1 \quad if\ g(x) \geq \theta \\ &= 0 \quad if\ g(x) < \theta \end{aligned}$$

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



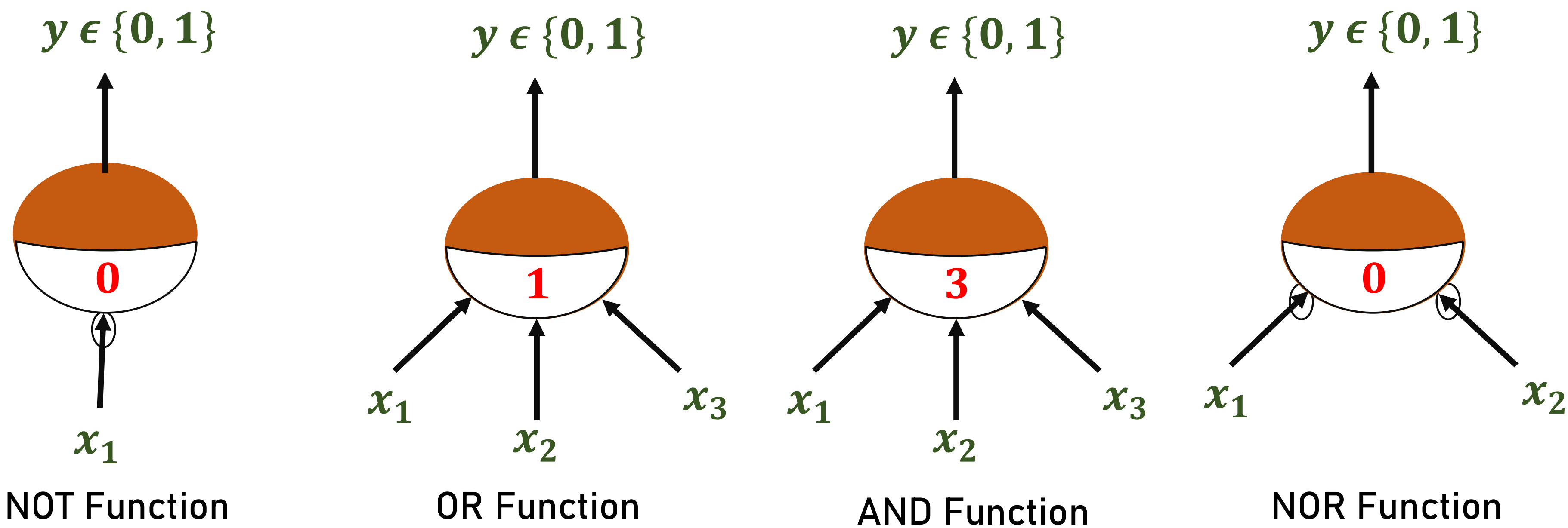
$$x_1, x_2, x_3, x_4, \ldots \ldots, x_n \in \{0, 1\}$$

Boolean Functions using MP Neurons



A McCulloch Pitts Unit | OR Function | AND Function | NOR Function

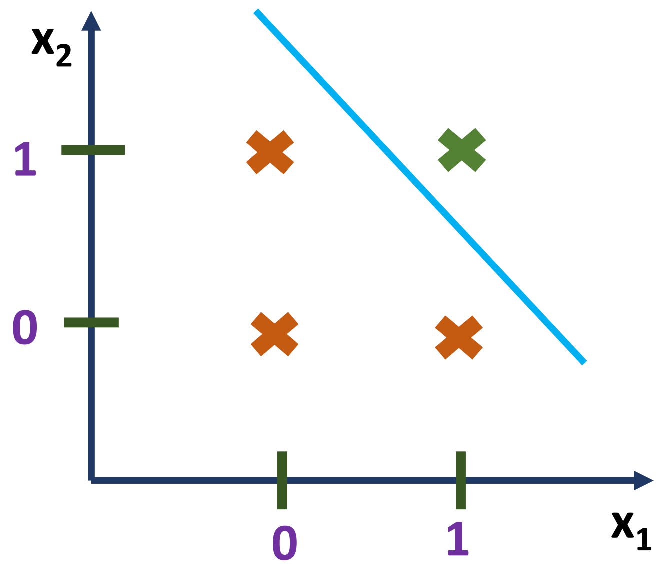
Boolean Functions using MP Neurons

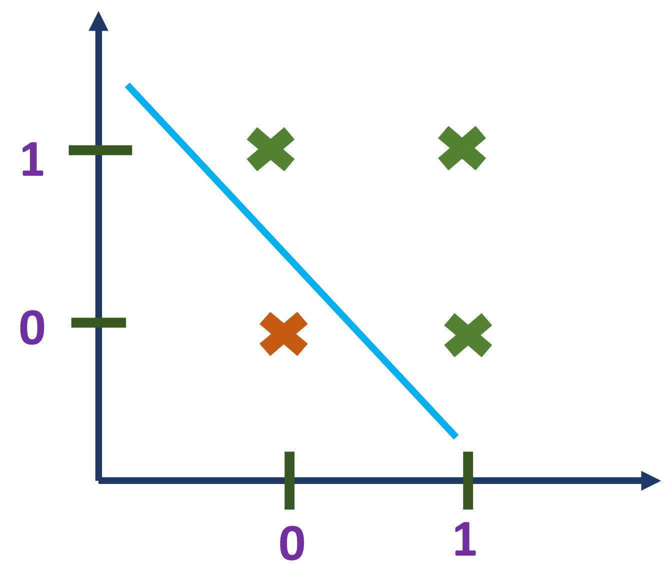


McCulloch Pitts Neuron

- McCulloch Pitts Neuron can be used to represent Boolean functions which are linearly separable
- It produces a linear separability for Boolean functions, such that all inputs which produce a 1 lie on one side of the line (plane) and all points which produce a 0 lie on other side of the line (plane)

x_1	x_2	AND	OR
0	**0**	**0**	**0**
0	**1**	**0**	**1**
1	**0**	**0**	**1**
1	**1**	**1**	**1**





McCulloch Pitts Neuron

- McCulloch Pitts Neuron can be used to represent Boolean functions which are linearly separable
- It produces a linear separability for Boolean functions, such that all inputs which produce a 1 lie on one side of the line (plane) and all points which produce a 0 lie on other side of the line (plane)
- What about non-Boolean inputs?
- All the inputs are given equal weights? What if some inputs are more important?
- The threshold θ must be chosen by hand
- What about the data which are not linearly separable?

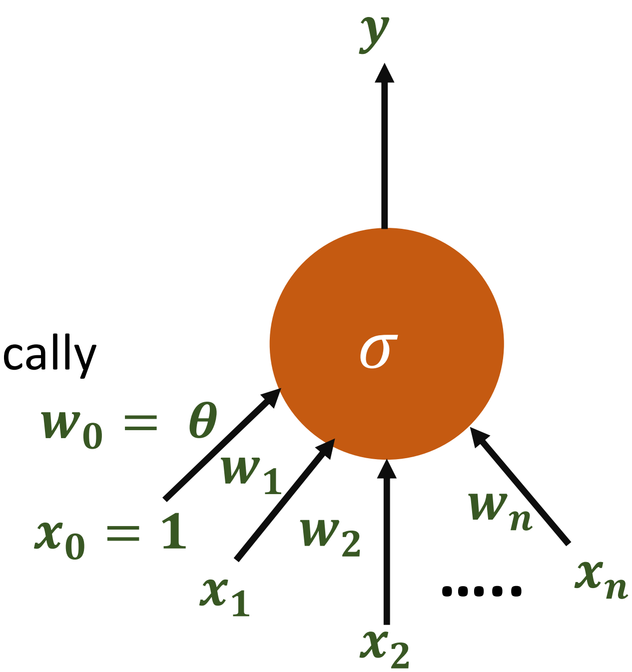
The Rosenblatt's Perceptron (1957)

Features:

- It can process non-Boolean inputs
- Different weights can be assigned to each input automatically
- The threshold θ is assigned automatically

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

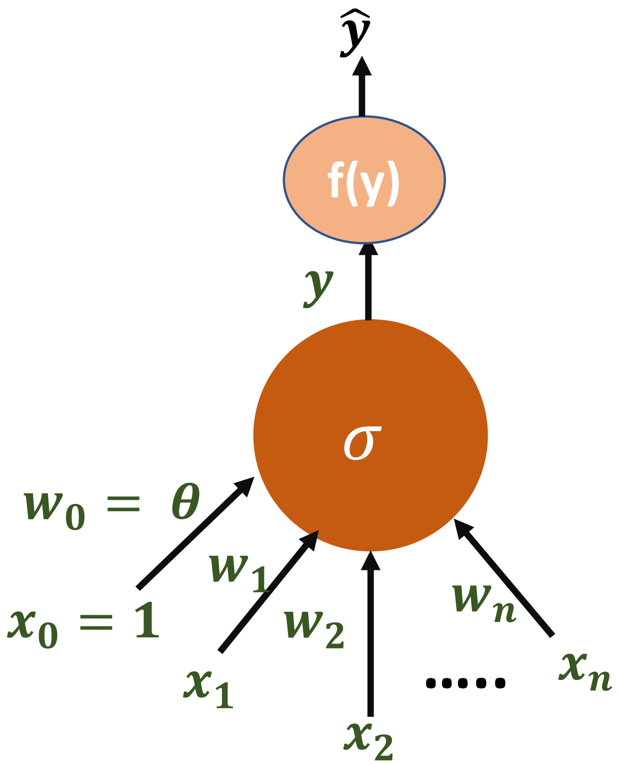
$$= 0 \ \ if \sum_{i=0}^{n} w_i * x_i < 0$$



Minsky and Papert (1969)

Features:

- Introduction of the activation function
- Heaviside function is harsh (0.49 & 0.51 leading different values)
- Considers smooth, differentiable activation function



NOT Logical Function

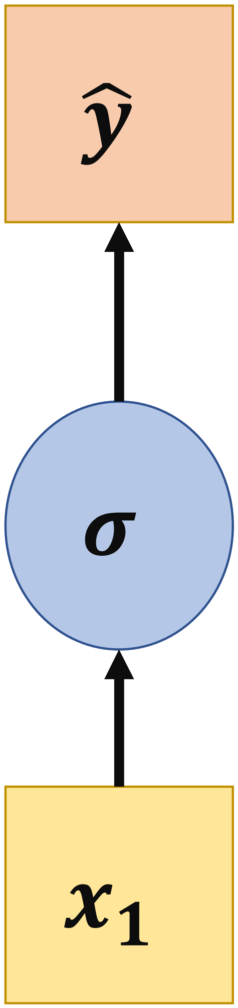
NOT Function can be implemented with:

$$\boldsymbol{y = w_1 . x_1 + w_0}$$

$$\boldsymbol{\hat{y} = \begin{matrix} 1 \; if \; y \geq 0 \\ 0 \; if \; y < 0 \end{matrix}}$$

Say: $w_1 = -1, w_0 = 0.5$

$\boldsymbol{Case\ 1:\ x_1 = 0,\quad y = 0.5,\quad \hat{y} = 1}$
$\boldsymbol{Case\ 2:\ x_1 = 1,\quad y = -0.5,\quad \hat{y} = 0}$



AND Logical Function

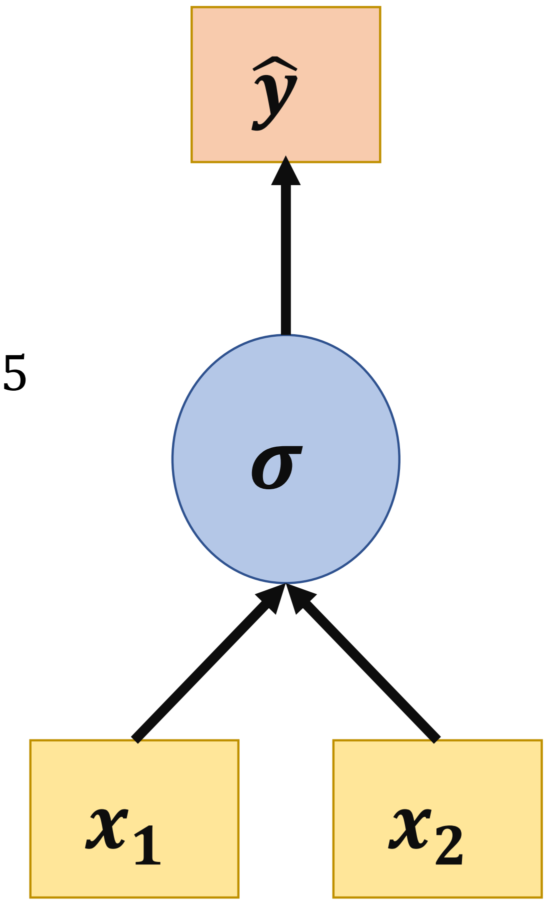
AND Function can be implemented with:

$$y = w_1 * x_1 + w_2 * x_2 + w_0$$

$$\hat{y} = \begin{matrix} 1 \text{ if } y \geq 0 \\ 0 \text{ if } y < 0 \end{matrix}$$

Say: $w_1 = 1, w_2 = 1, w_0 = -1.5$

x_1	x_2	y	$\hat{y}$
0	0	-1.5	0
0	1	-0.5	0
1	0	-0.5	0
1	1	0.5	1



OR Logical Function

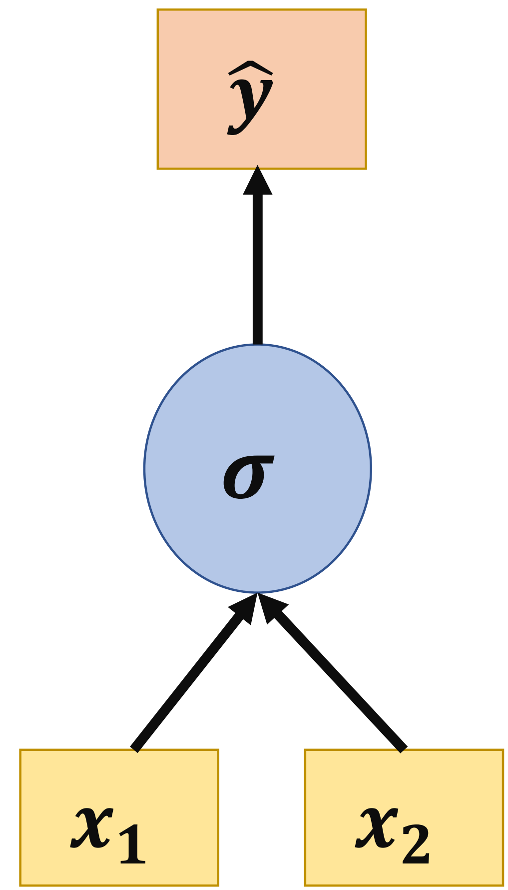
OR Function can be implemented with:

$$\boldsymbol{y = w_1 * x_1 + w_2 * x_2 + w_0}$$

$$\hat{y} = \begin{cases} \mathbf{1} \ if \ y \geq 0 \\ \mathbf{0} \ if \ y < 0 \end{cases}$$

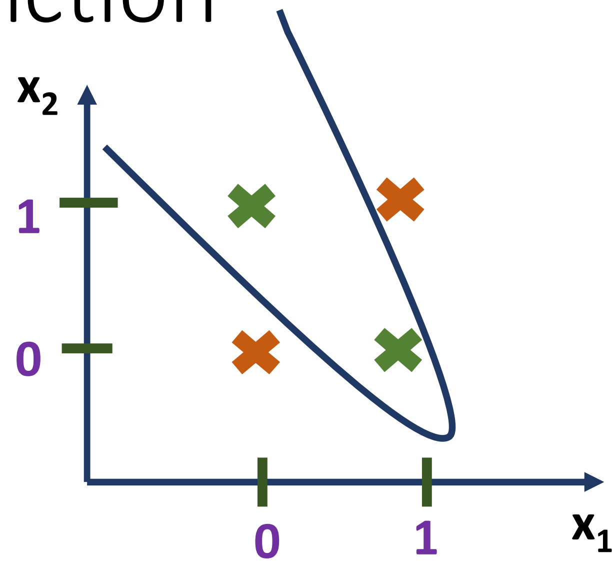
Say: $w_1 = 1, w_2 = 1, w_0 = -0.5$

x_1	x_2	y	$\hat{y}$
0	0	-0.5	0
0	1	0.5	1
1	0	0.5	1
1	1	1.5	1



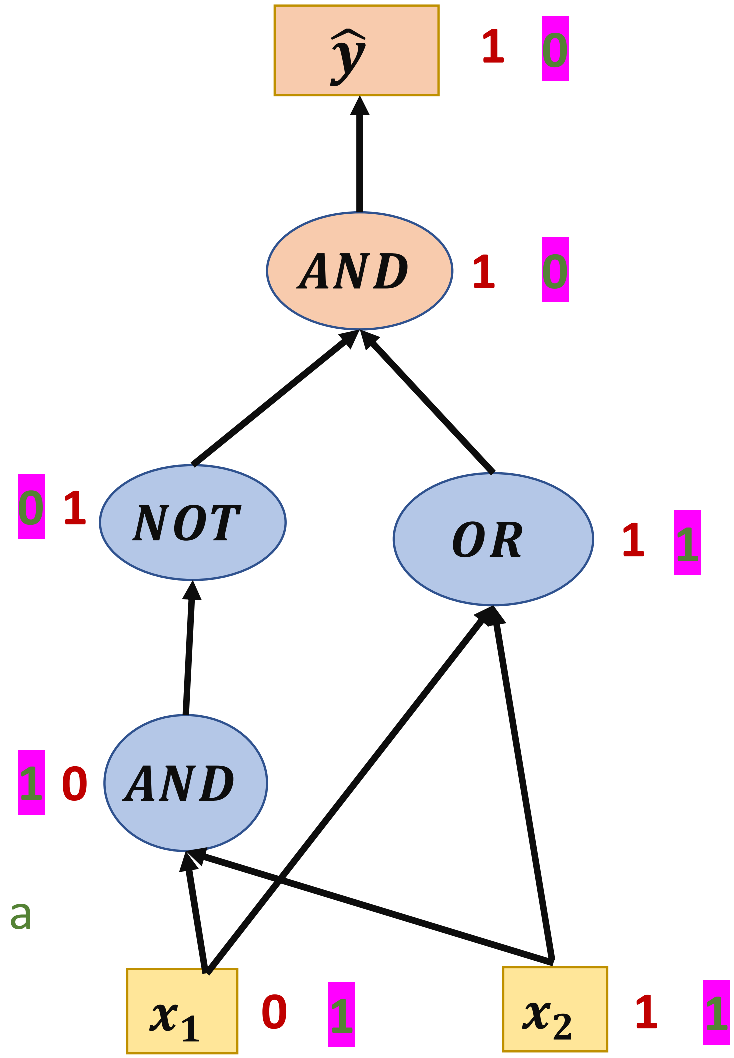
XOR Logical Function

x_1	x_2	XOR
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**0**



$$\text{XOR}(x_1,x_2) = \text{AND}(\text{NOT}(\text{AND}(x_1,x_2)),\text{OR}(x_1,x_2))$$

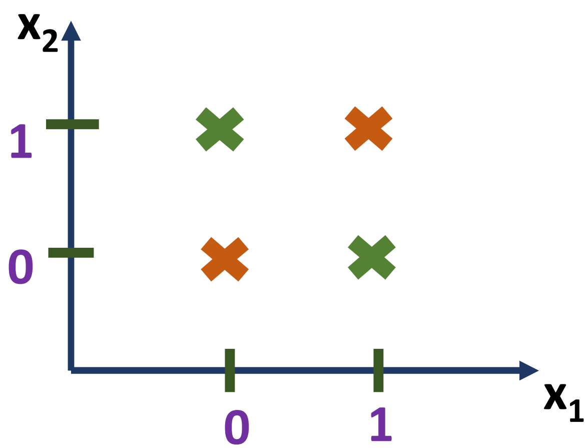
A single perceptron cannot deal with nonlinear data, however a network of perceptrons can indeed deal with such data

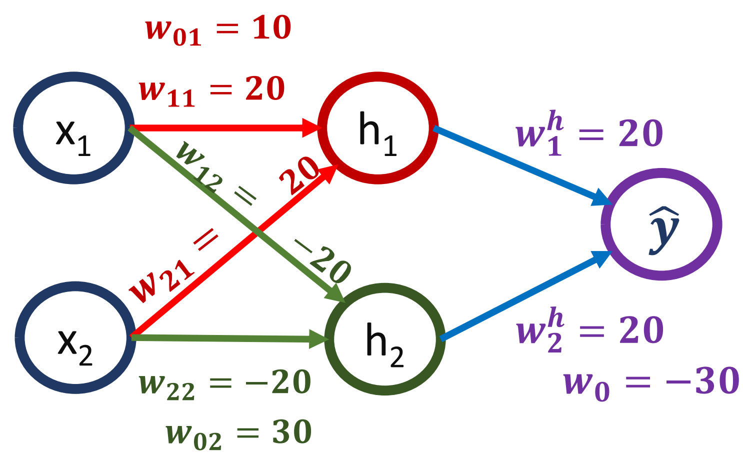


XOR Logical Function

$\cancel{\hat{y} = \sigma(w_1 x_1 + w_2 x_2 + w_0)}$

x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0





$$h_1 = w_{11}\, x_1 + w_{12}\, x_2 + w_{01}$$
$$= 20 \times x_1 + 20 \times x_2 + 10$$

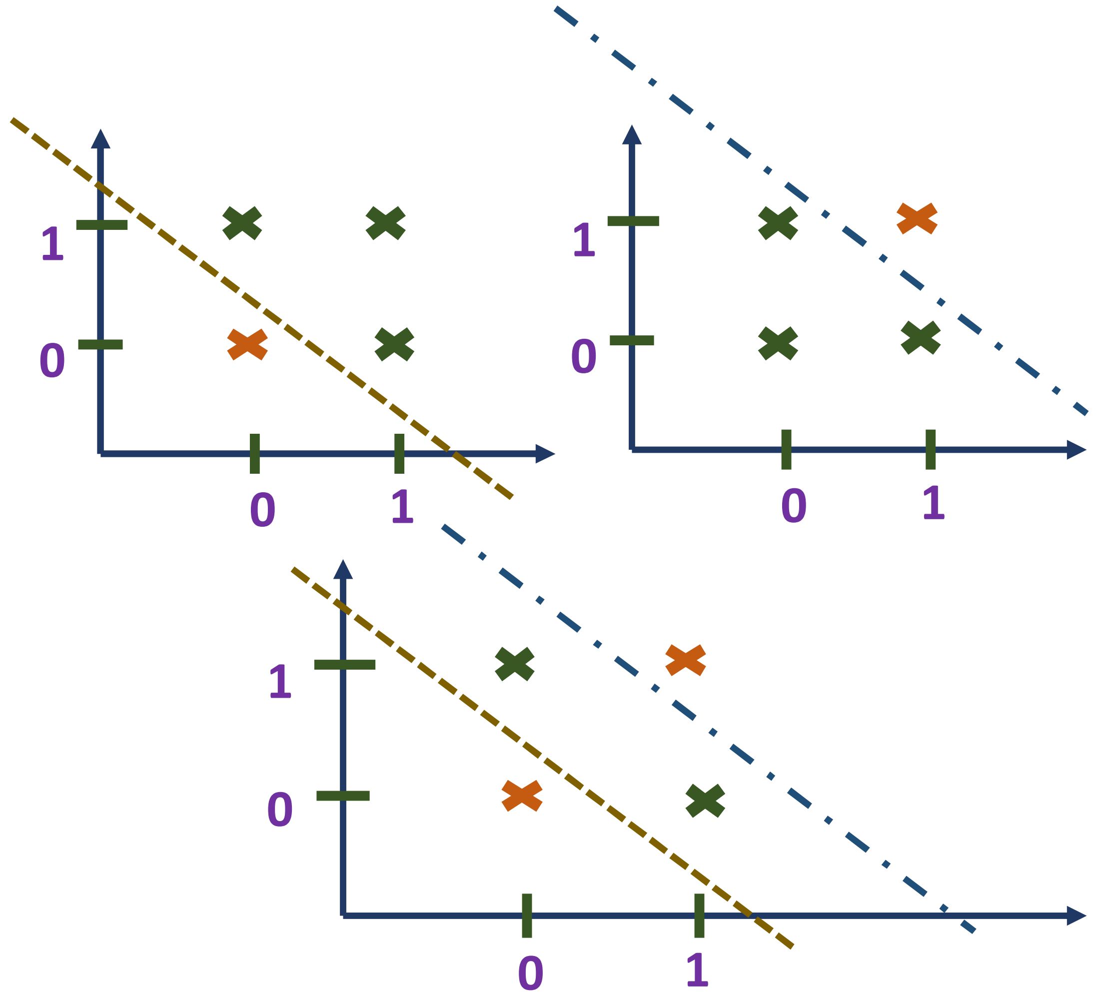
$$h_2 = w_{21}\, x_1 + w_{22}\, x_2 + w_{02}$$
$$= -20 \times x_1 - 20 \times x_2 + 30$$

$$y = w_1^h\, h_1 + w_2^h\, h_2 + w_0$$
$$\hat{y} = \sigma(y) = \sigma(20 \times h_1 + 20 \times h_2 - 30)$$

x_1	x_2	h_1	h_2	y	$\hat{y}$
0	0	0	1	-10	0
0	1	1	1	10	1
1	0	1	1	10	1
1	1	1	0	-10	0

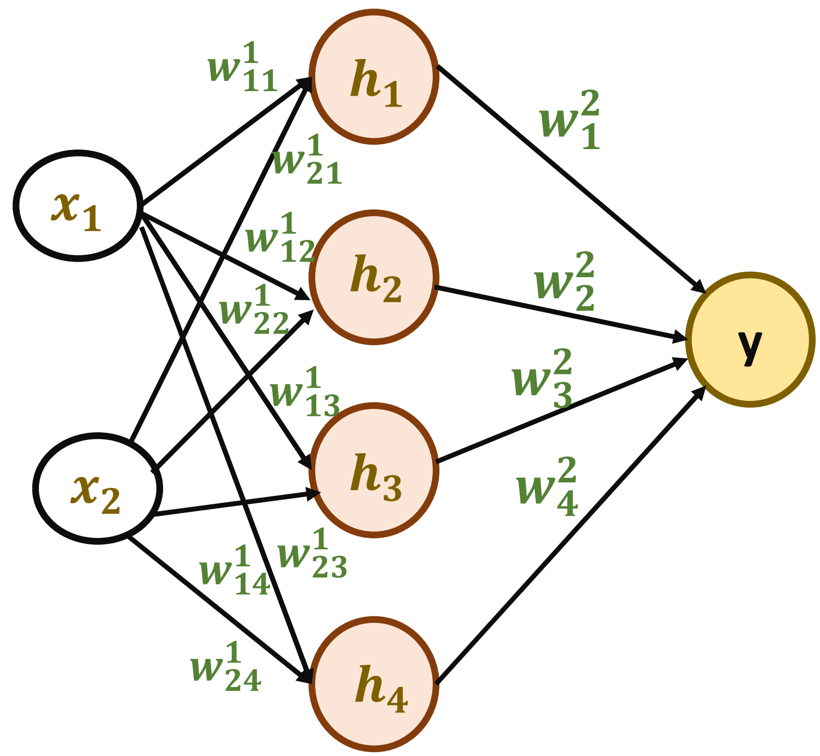
XOR Logical Function

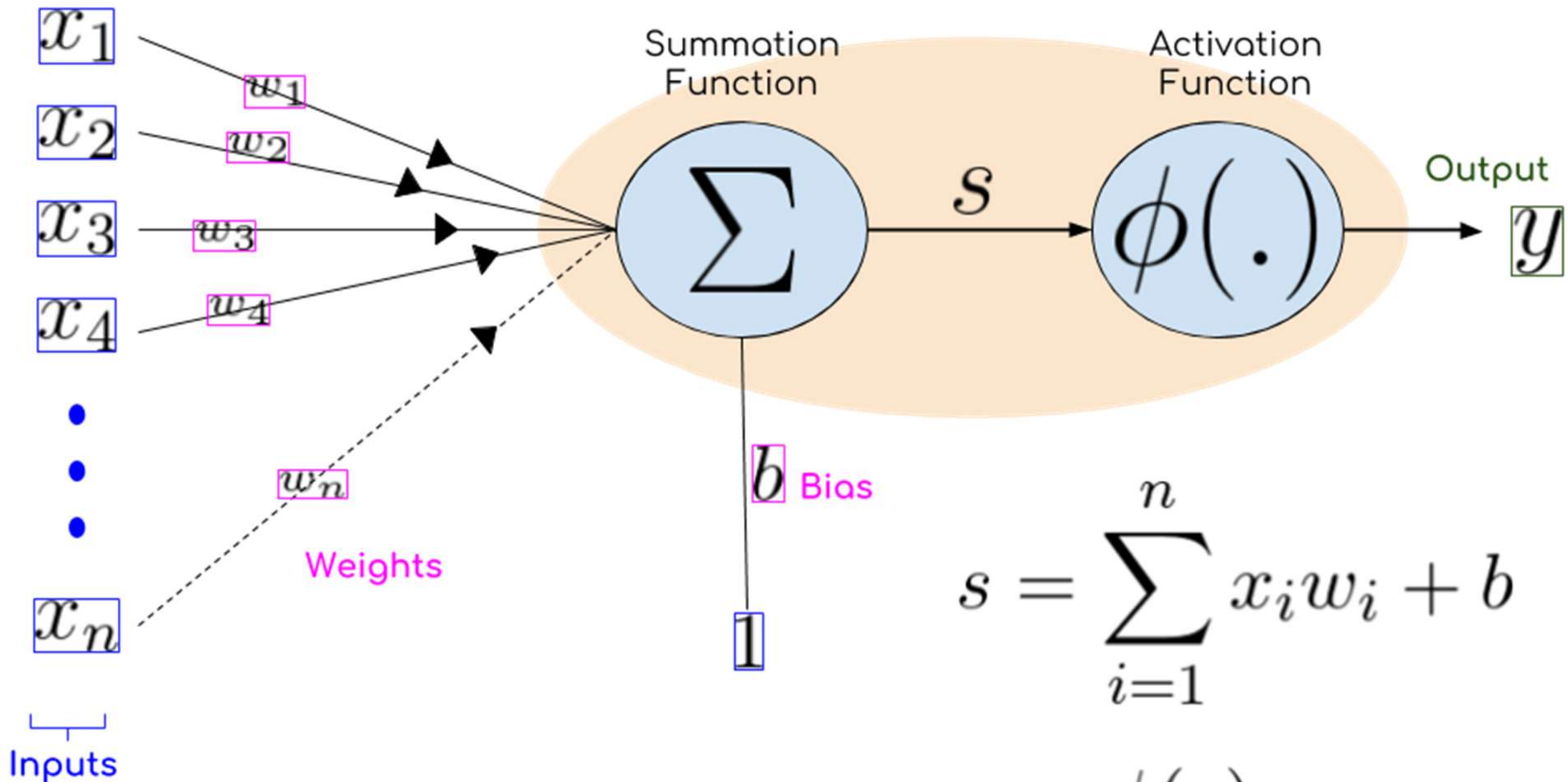
x_1	x_2	h_1	h_2	$\hat{y}$
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0
		OR	NAND	AND



Multilayer Perceptrons

- The network contains 3 fully-connected layers
- The layer containing the inputs (x_1,x_2) is called the **input layer**
- The middle layer containing the 4 perceptrons is called the **hidden layer**
- The outputs of the 4 perceptrons in the hidden layer are denoted by (h_1, h_2, h_3, h_4)
- The final layer containing one output neuron is called the **output layer**





$$s = \sum_{i=1}^{n} x_i w_i + b$$

$$y = \phi(s)$$

- How to learn w_i and b automatically?

Loss Function and Gradient Descent

Loss Function

- Also known as cost function, objective function, and error function
- The loss function provides the cost of being wrong, by measuring the quality of a particular set of parameters based on how well the output of the network agrees with the ground truth labels in the training data

input label, ground truth

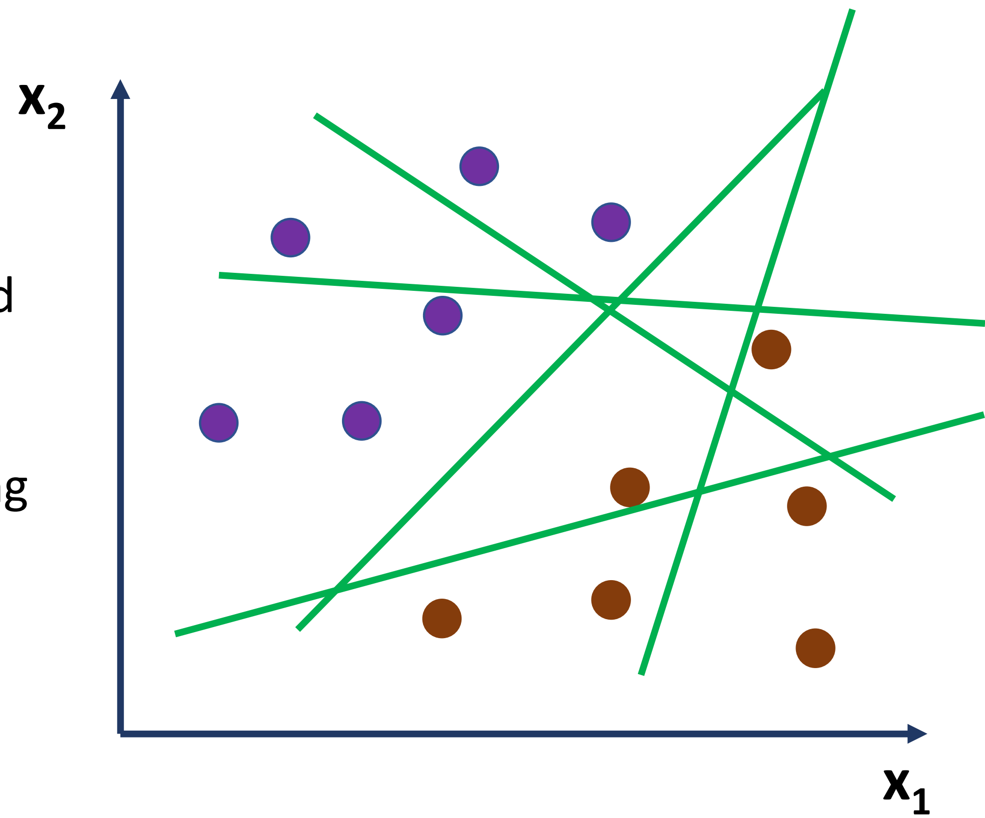
$$\boldsymbol{\mathcal{J}(w) = distance(f_{\theta}(x), y))}$$

error

parameters(weights, biases)

Learning Process

1. Start with random values of w_i
2. Evaluate the **goodness** of the line, determined with a loss function, $J(w)$
3. The weights w_i is changed accordingly moving the line to a **better** position
   - Note: $J(w)$ should be minimum when the training samples are correctly classified
4. Repeat 2 and 3 until $J(w) < \tau$



$$y = w_0 + w_1 x_1 + w_2 x_2$$

- **Gradient Descent Algorithm**

Gradient Descent in Action

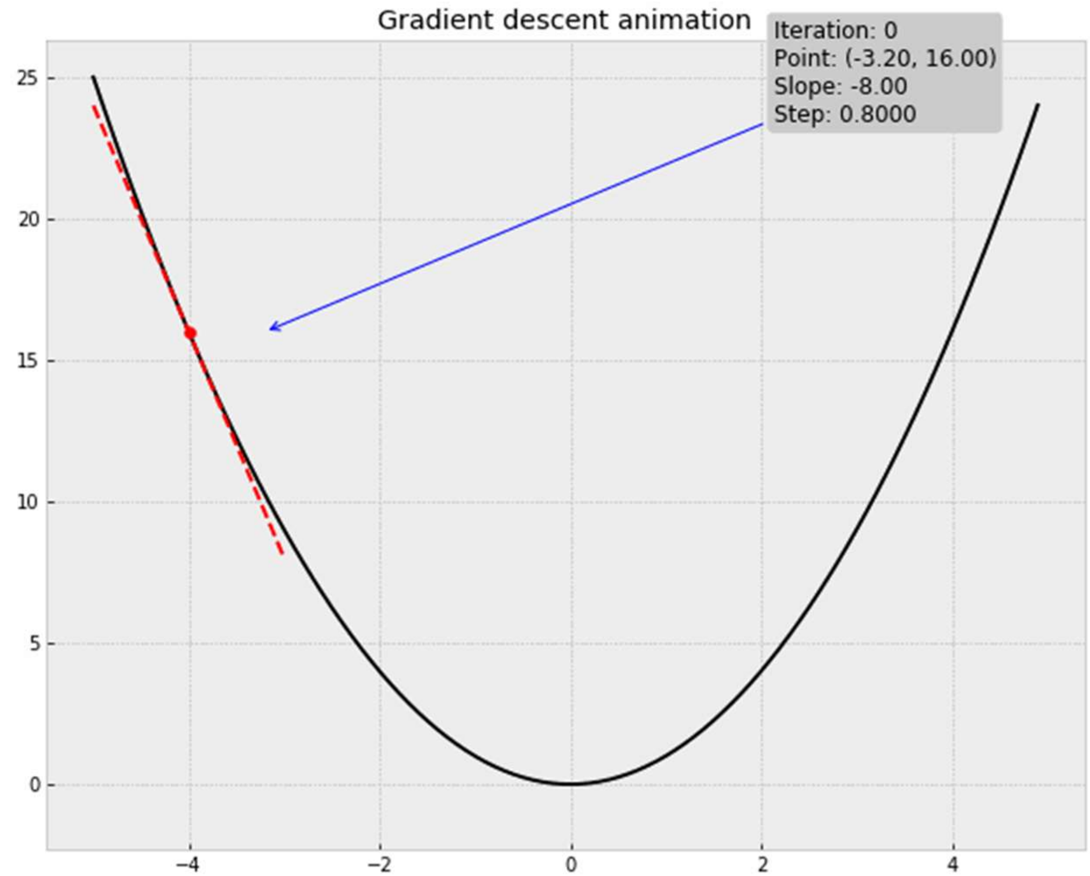


Image Source: Kaggle

Gradient Descent in Action

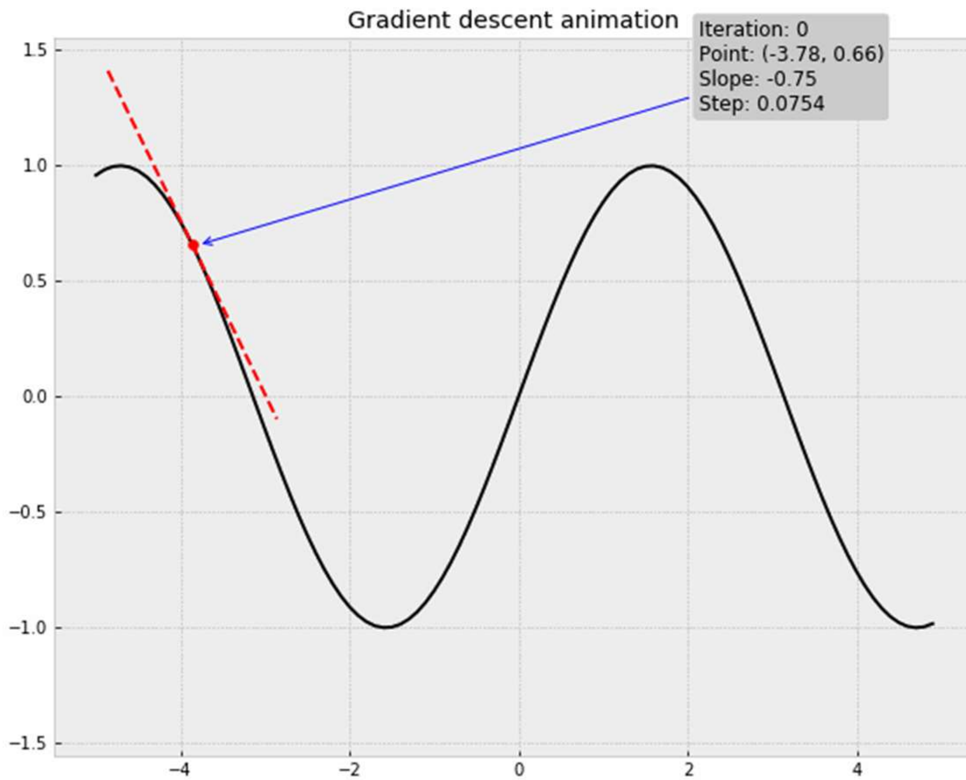


Image Source: Kaggle