

# **Machine Learning (Artificial Neural Network)**

Dr. Emad Natsheh

# Types of Machine Learning



Supervised Learning

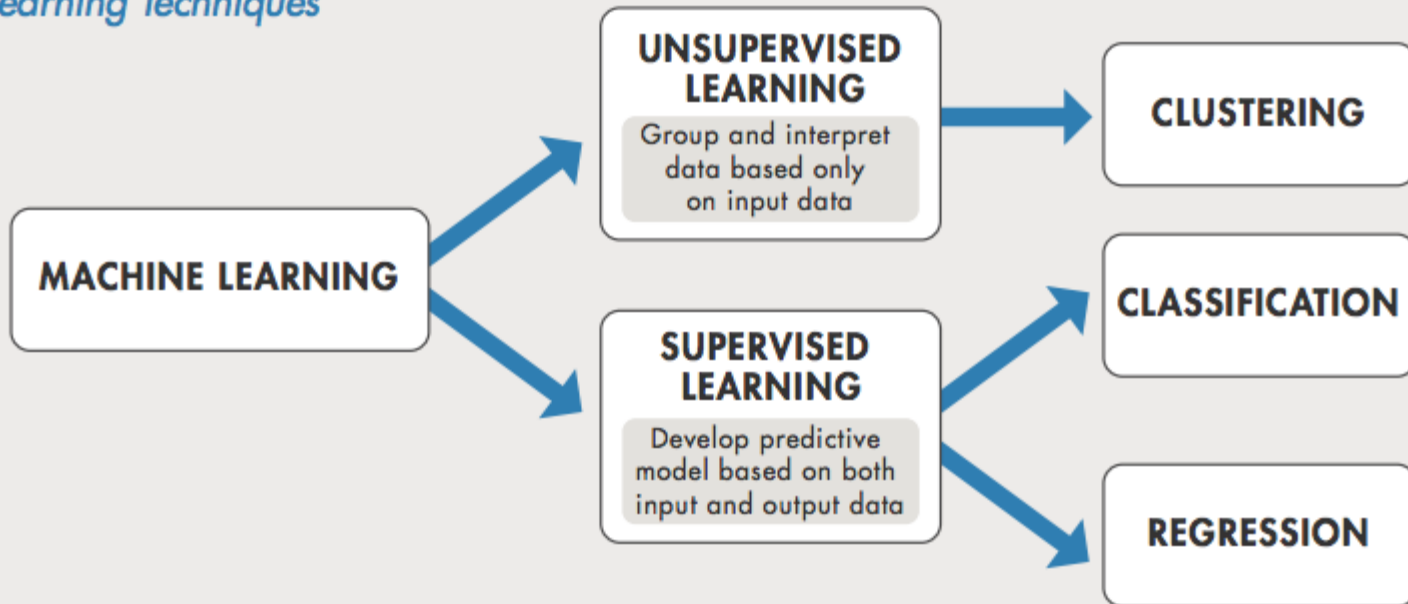


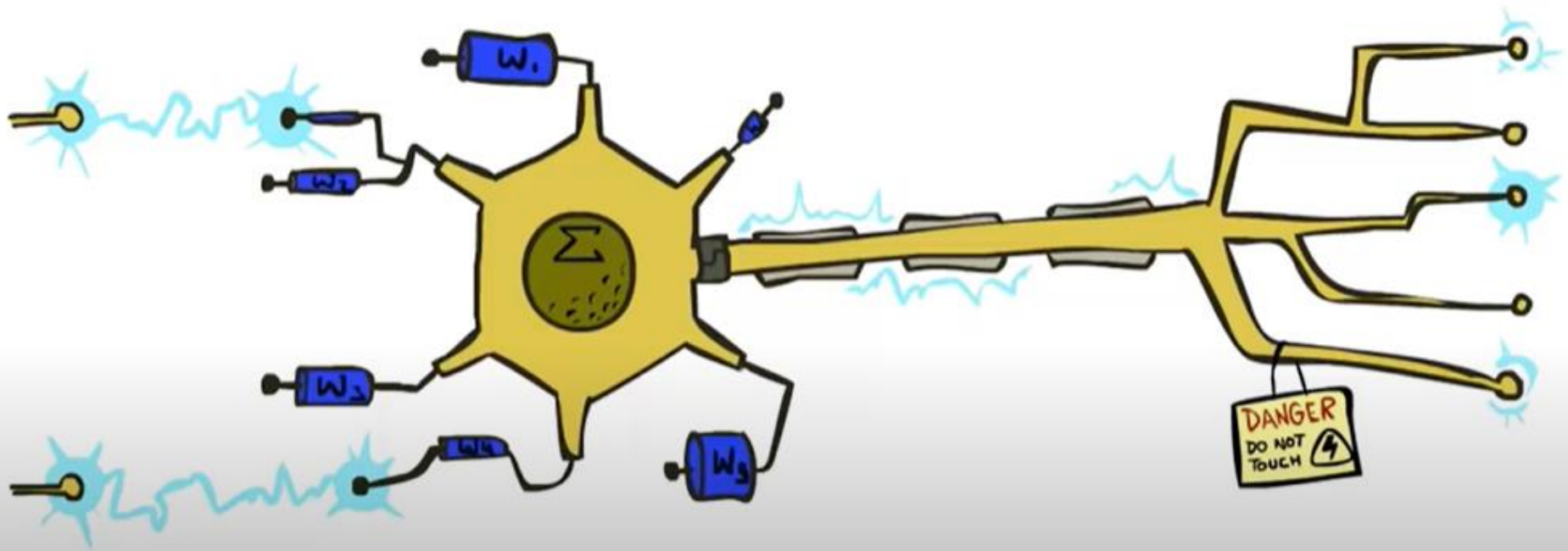
Unsupervised Learning



Reinforcement Learning

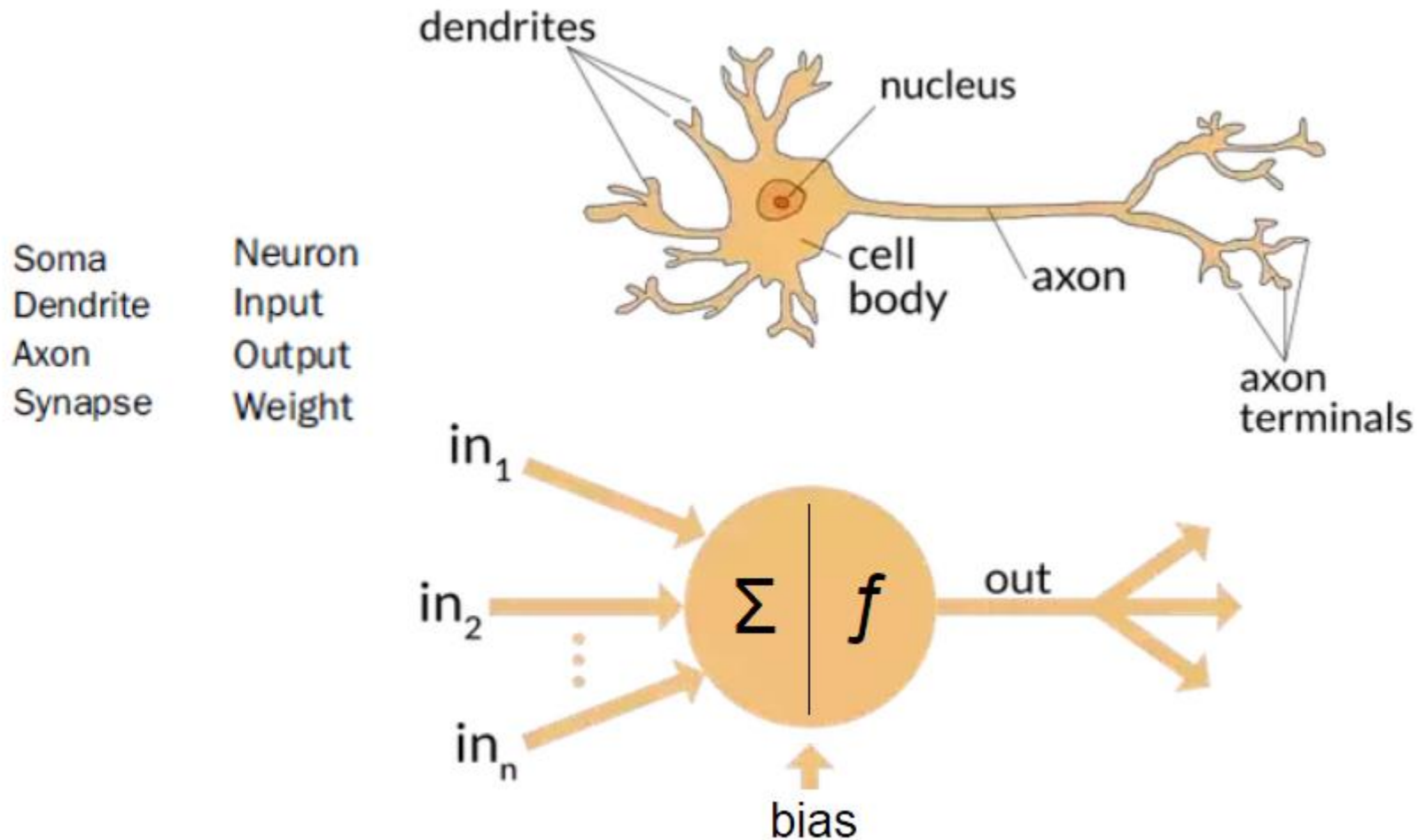
## *Machine Learning Techniques*





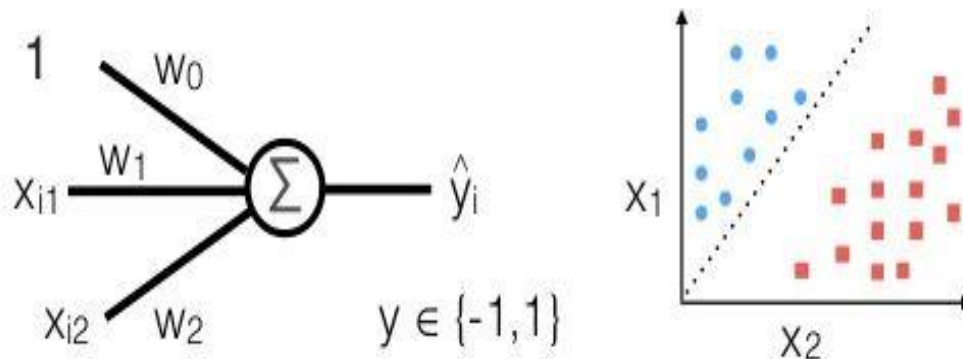
# PERCEPTRON

# Biological Neuron vs Perceptron



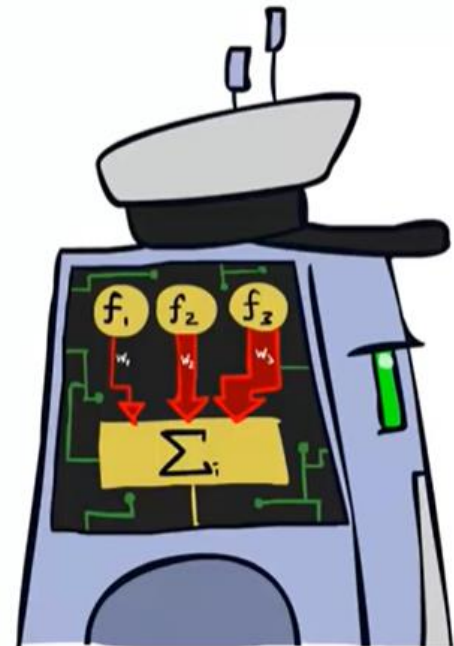
# What is Perceptron

- It's a single node neural network that can take different inputs but produce only one output
- Perceptron is usually used to classify the data into two parts. Therefore, it is also known as a [Linear Binary Classifier](#).

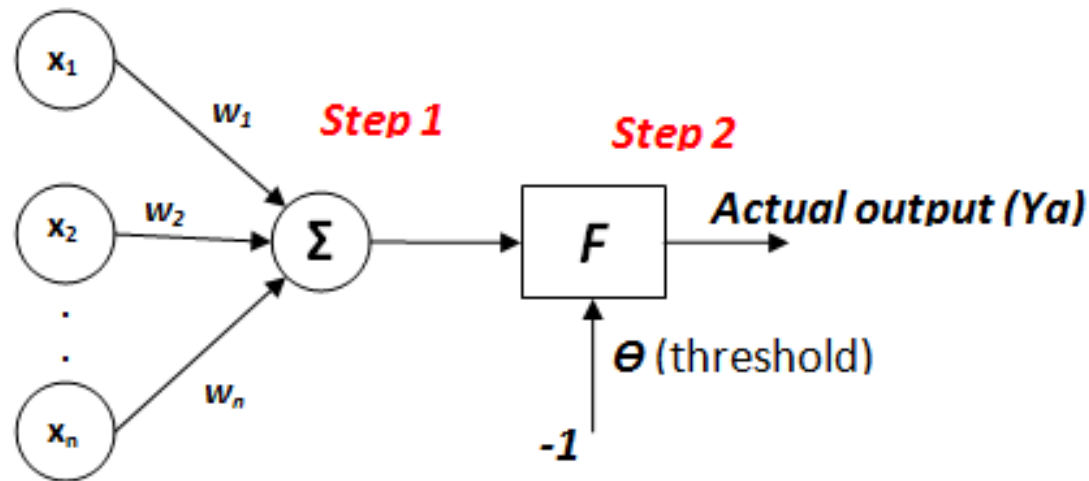


# How does the neuron determine its output?

1. Computes the weighted sum input
2. Apply the value to the activation function (step, sign, sigmoid, linear)



# How does the neuron determine its output?

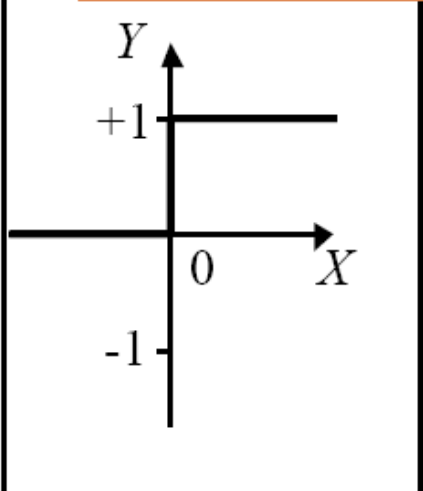
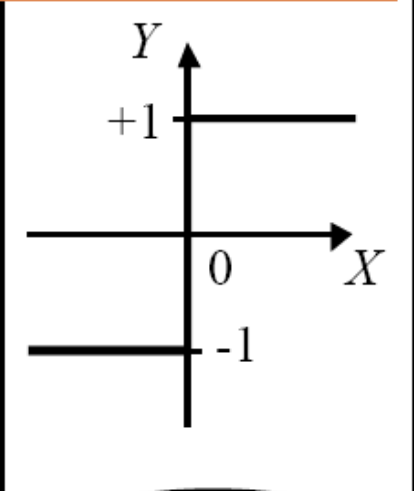
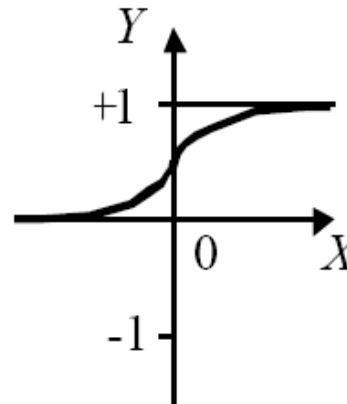
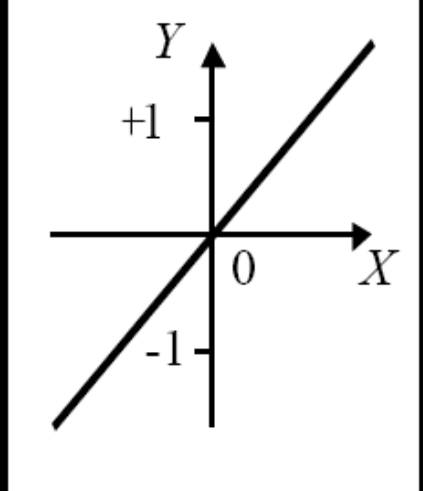


$$x_1w_1 + x_2w_2 + \dots + x_nw_n$$

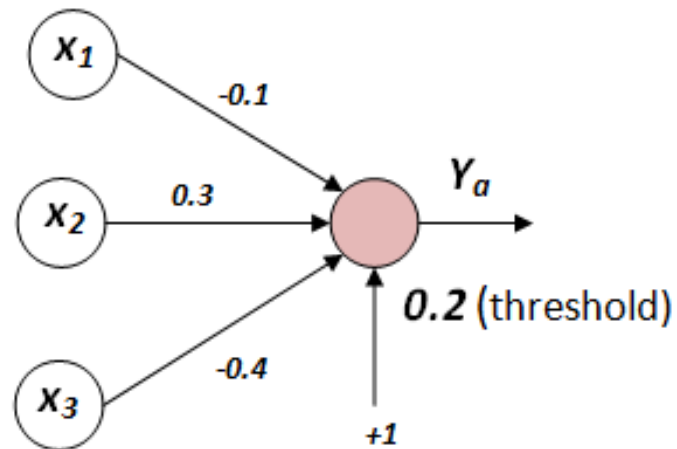
$$Y_a = F\left(\sum_{i=1}^n x_i w_i - \theta\right)$$



# Activation Functions

| Step function  | Sign function  | Sigmoid function  | Linear function  |
|--|--|---|--|
| hard limit functions   |  |   |  |
|        |          |  |  |
| $Y^{step} = \begin{cases} 1, & \text{if } X \geq 0 \\ 0, & \text{if } X < 0 \end{cases}$ | $Y^{sign} = \begin{cases} +1, & \text{if } X \geq 0 \\ -1, & \text{if } X < 0 \end{cases}$ | $Y^{sigmoid} = \frac{1}{1 + e^{-X}}$  | $Y^{linear} = X$   |

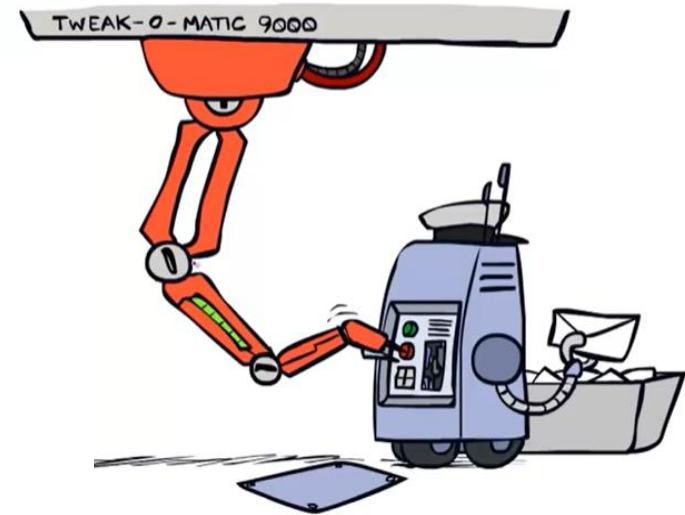
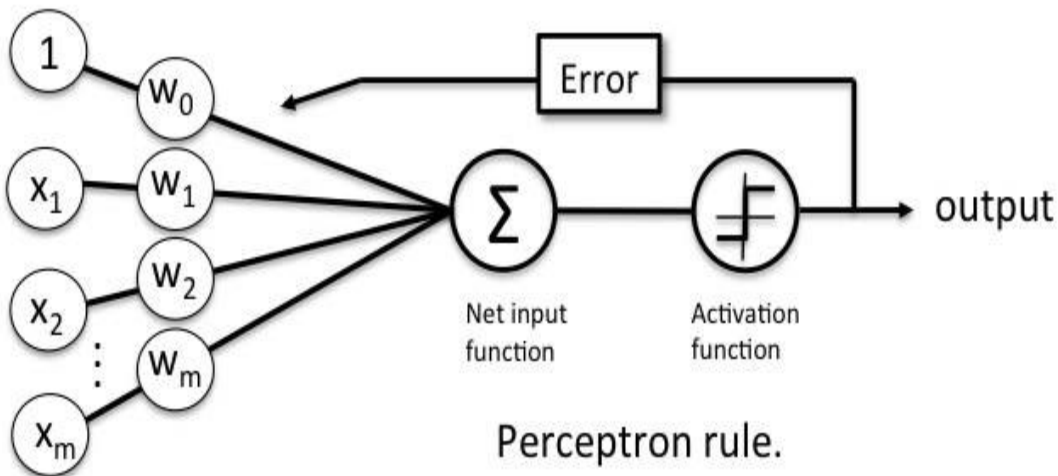
# Example



| $x_1$ | $x_2$ | $x_3$ | $Y_a$ (step) |
|-------|-------|-------|--------------|
| 1     | 0     | -1    | 1            |
| -1    | -1    | 0.5   | 0            |

# How does a perceptron learn

- ❖ This is done by making small adjustments in the weights to reduce the difference between the actual and desired outputs of the perceptron



# Perceptron learn

## □ Step 1: Initialization

- Set initial weights  $w_1, w_2, \dots, w_n$  and threshold to random numbers in the range  $[-0.5, 0.5]$

## □ Step 2: Activation

- Activate the perceptron by applying inputs  $x_1(p), x_2(p), x_3(p), \dots, x_n(p)$ , and desired output  $y_d(p)$ . Calculate the actual output at iteration  $p = 1$

$$Y(p) = \text{step} \left[ \sum_{i=1}^n x_i(p) w_i(p) - \theta \right]$$

- where  $n$  is the number of the perceptron inputs, and  $\text{step}$  is a step activation function.

# Perceptron learn

## □ Step 3: Weight training

$$\text{error } e = Y_{\text{expected}} - Y_{\text{actual}}$$

- ▣ Update the weights of the perceptron

$$w_i(p+1) = w_i(p) + \Delta w_i(p),$$

where  $\Delta w$  is the weight correction at iteration  $p$ . The weight correction is computed by the delta rule:

$$\Delta w_i(p) = \alpha \times x_i(p) \times e(p)$$

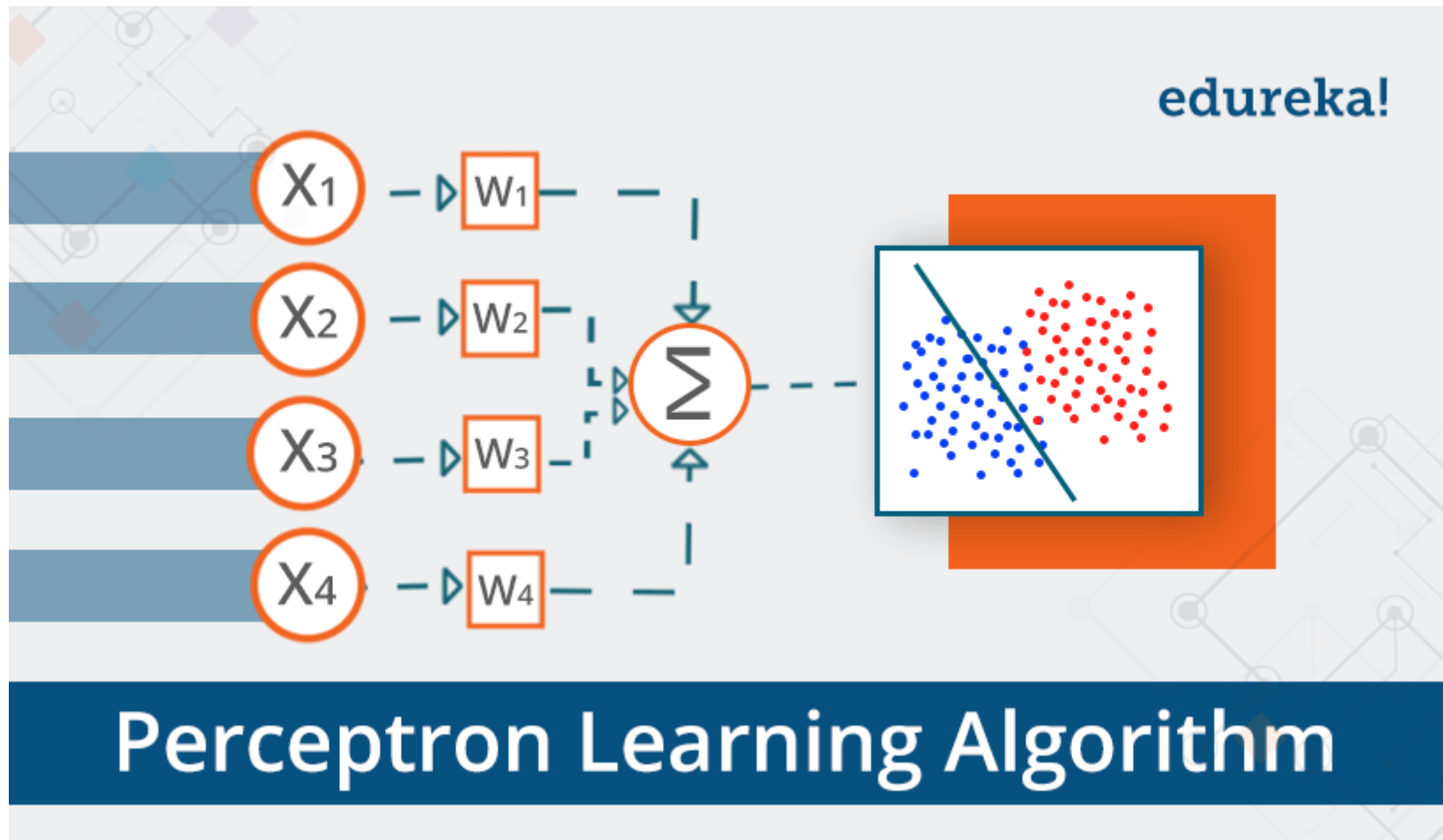
## □ Step 4: Iteration

$\alpha$  is the *learning rate* (between 0 and 1)

- ▣ Increase iteration  $p$  by one, go back to Step 2 and repeat the process until convergence.

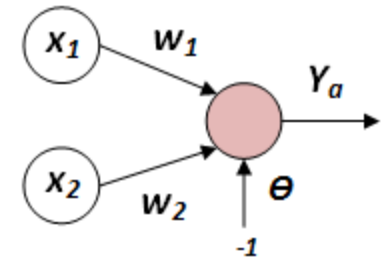
# Perceptron learn

edureka!



# Perceptron Example – AND

- Train a perceptron to recognize logical AND



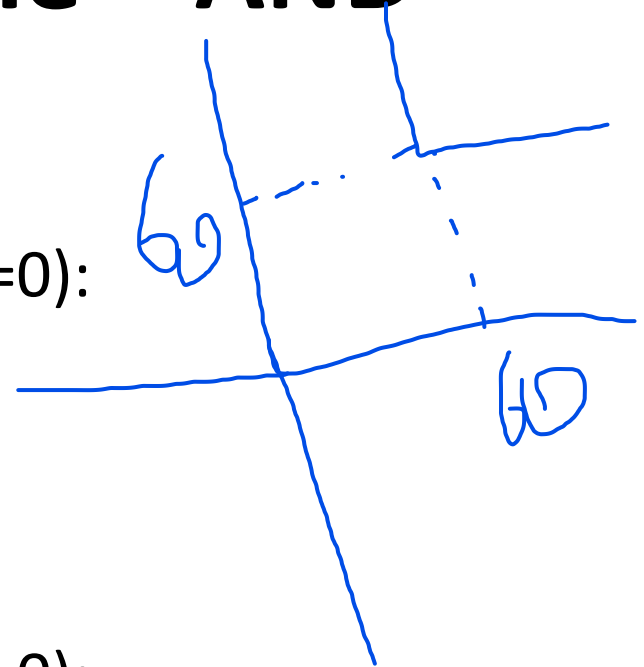
| Epoch | Inputs |       | Desired output<br>$Y_d$ | Initial weights |       | Actual output<br>$Y$ | Error<br>$e$ | Find weights |       |
|-------|--------|-------|-------------------------|-----------------|-------|----------------------|--------------|--------------|-------|
|       | $x_1$  | $x_2$ |                         | $w_1$           | $w_2$ |                      |              | $w_1$        | $w_2$ |
| 1     | 0      | 0     | 0                       | 0.3             | -0.1  | 0                    | 0            | 0.3          | -0.1  |
|       | 0      | 1     | 0                       | 0.3             | -0.1  | 0                    | 0            | 0.3          | -0.1  |
|       | 1      | 0     | 0                       | 0.3             | -0.1  | 1                    | -1           | 0.2          | -0.1  |
|       | 1      | 1     | 1                       | 0.2             | -0.1  | 0                    | 1            | 0.3          | 0.0   |
| 2     | 0      | 0     | 0                       | 0.3             | 0.0   | 0                    | 0            | 0.3          | 0.0   |
|       | 0      | 1     | 0                       | 0.3             | 0.0   | 0                    | 0            | 0.3          | 0.0   |
|       | 1      | 0     | 0                       | 0.3             | 0.0   | 1                    | -1           | 0.2          | 0.0   |
|       | 1      | 1     | 1                       | 0.2             | 0.0   | 1                    | 0            | 0.2          | 0.0   |



Use threshold  $\Theta = 0.2$  and  
learning rate  $\alpha = 0.1$

# Perceptron Example – AND

- Epoch 1
  - Iteration 1 (input  $X_1=0$ ;  $X_2=0$ ;  $Y_d=0$ ):
    - $(0 * 0.3 + 0 * -0.1) - 0.2 = -0.2$
    - $Y_a = \text{step}(-0.2) = 0$
    - $\text{Error} = Y_d - Y_a = 0$
    - $\Delta w_1 = 0$ ;  $\Delta w_2 = 0$ ;
  - Iteration 2 (input  $X_1=0$ ;  $X_2=1$ ;  $Y_d=0$ ):
    - $(0 * 0.3 + 1 * -0.1) - 0.2 = -0.3$
    - $Y_a = \text{step}(-0.3) = 0$
    - $\text{Error} = Y_d - Y_a = 0$
    - $\Delta w_1 = 0$ ;  $\Delta w_2 = 0$ ;





# Perceptron Example – AND

– Iteration 3 (input  $X_1=1$ ;  $X_2=0$ ;  $Y_d=0$ )

- $(1 * 0.3 + 0 * -0.1) - 0.2 = +0.1$
- $Y_a = \text{step}(+0.1) = 1$
- $\text{Error} = Y_d - Y_a = -1$
- $\Delta w_1 = 0.1 * 1 * -1 = -0.1$
- $W_1(\text{new}) = w_1(\text{old}) + \Delta w_1 = 0.3 - 0.1 = \underline{\underline{0.2}}$
- $\Delta w_2 = 0$

# Perceptron Example – AND

| Epoch | Inputs |       | Desired output<br>$Y_d$ | Initial weights |       | Actual output<br>$Y$ | Error<br>$e$ | Final weights |       |
|-------|--------|-------|-------------------------|-----------------|-------|----------------------|--------------|---------------|-------|
|       | $x_1$  | $x_2$ |                         | $w_1$           | $w_2$ |                      |              | $w_1$         | $w_2$ |
| 3     | 0      | 0     | 0                       | 0.2             | 0.0   | 0                    | 0            | 0.2           | 0.0   |
|       | 0      | 1     | 0                       | 0.2             | 0.0   | 0                    | 0            | 0.2           | 0.0   |
|       | 1      | 0     | 0                       | 0.2             | 0.0   | 1                    | -1           | 0.1           | 0.0   |
|       | 1      | 1     | 1                       | 0.1             | 0.0   | 0                    | 1            | 0.2           | 0.1   |
| 4     | 0      | 0     | 0                       | 0.2             | 0.1   | 0                    | 0            | 0.2           | 0.1   |
|       | 0      | 1     | 0                       | 0.2             | 0.1   | 0                    | 0            | 0.2           | 0.1   |
|       | 1      | 0     | 0                       | 0.2             | 0.1   | 1                    | -1           | 0.1           | 0.1   |
|       | 1      | 1     | 1                       | 0.1             | 0.1   | 1                    | 0            | 0.1           | 0.1   |



# Perceptron Example – AND

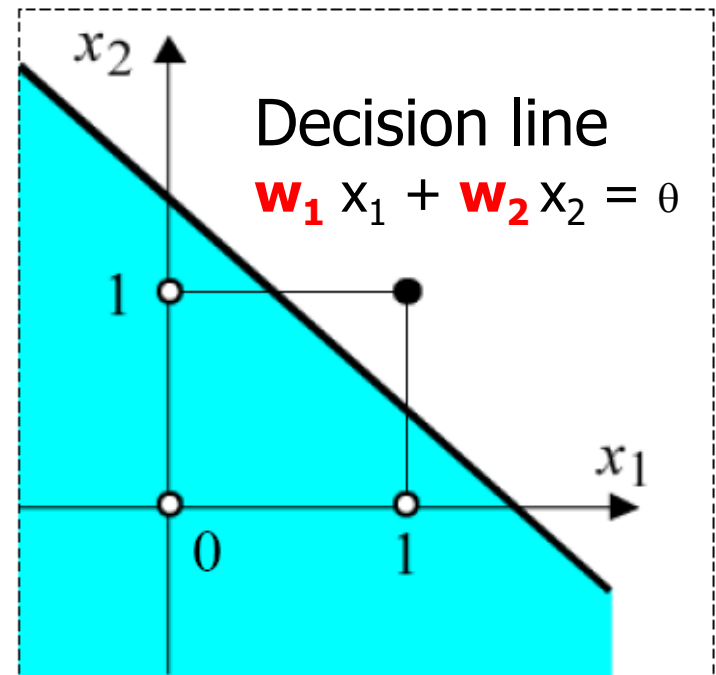
- Repeat until convergence
  - ▣ i.e. final weights do not change and *no error*

| Epoch | Inputs |       | Desired output<br>$Y_d$ | Initial weights |       | Actual output<br>$Y$ | Error<br>$e$ | Final weights |       |
|-------|--------|-------|-------------------------|-----------------|-------|----------------------|--------------|---------------|-------|
|       | $x_1$  | $x_2$ |                         | $w_1$           | $w_2$ |                      |              | $w_1$         | $w_2$ |
| 5     | 0      | 0     | 0                       | 0.1             | 0.1   | 0                    | 0            | 0.1           | 0.1   |
|       | 0      | 1     | 0                       | 0.1             | 0.1   | 0                    | 0            | 0.1           | 0.1   |
|       | 1      | 0     | 0                       | 0.1             | 0.1   | 0                    | 0            | 0.1           | 0.1   |
|       | 1      | 1     | 1                       | 0.1             | 0.1   | 1                    | 0            | 0.1           | 0.1   |

**Epoch** refers to one cycle through the full training dataset.  
Usually, training a neural network takes more than a few epochs

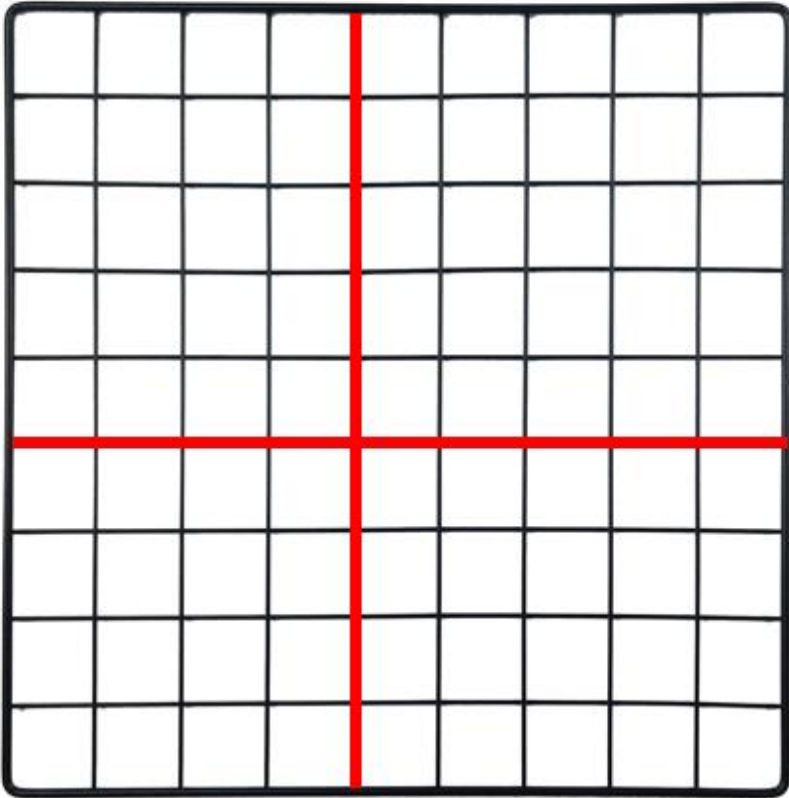
# Summary

- Single perceptron can be trained to recognize any linear separable function
- Perceptron is able to represent a function only if there is some line that separates all the black dots from all the white dots

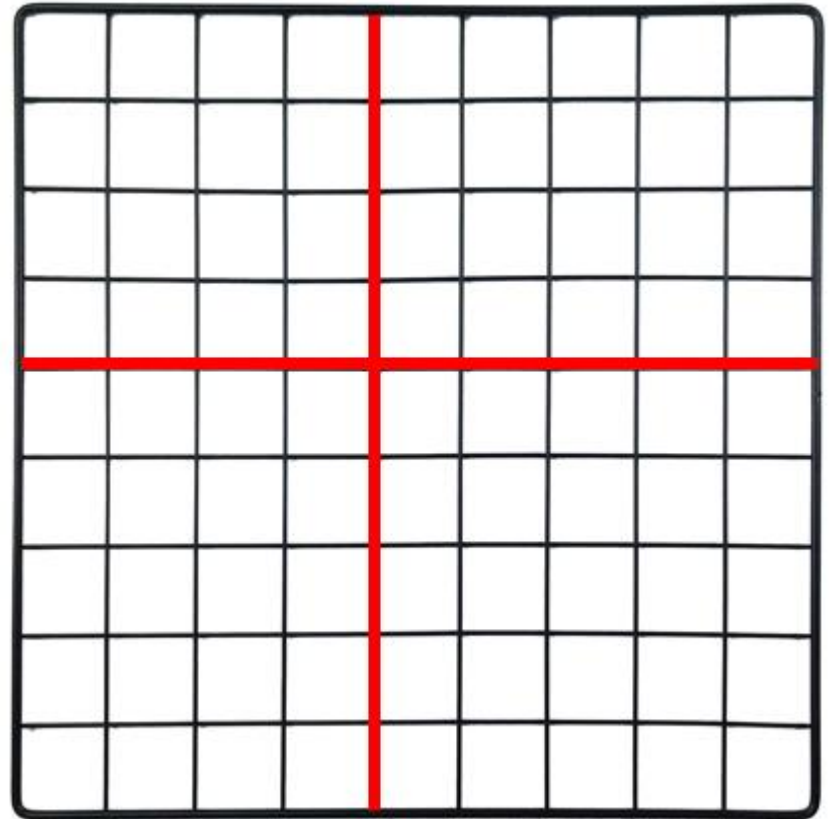


# Drawing Line

$$Y = 2x - 2$$



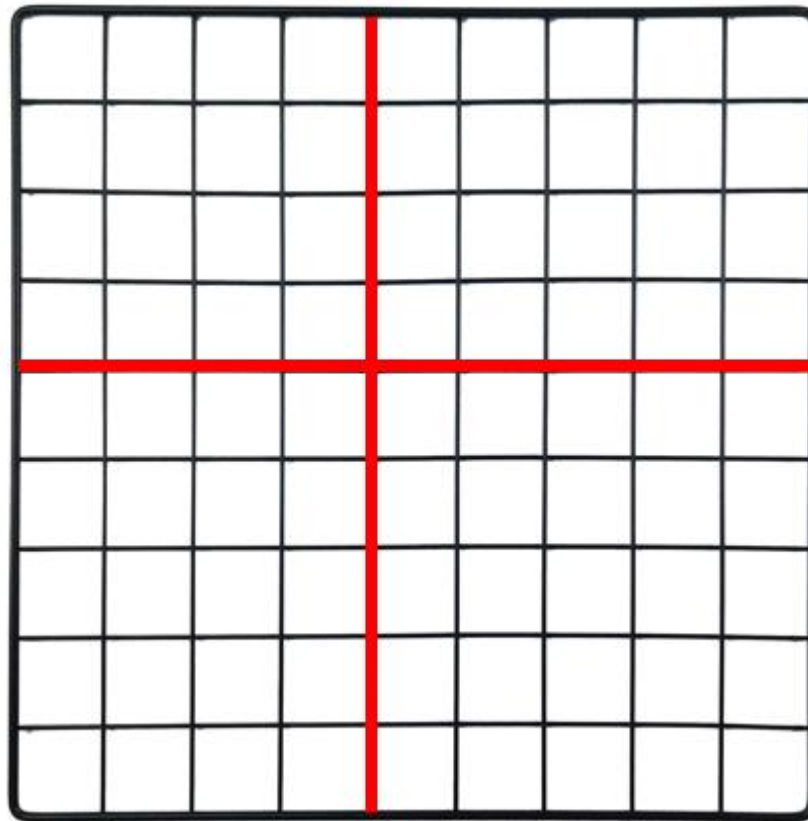
$$Y = -\frac{1}{2}x + 1$$



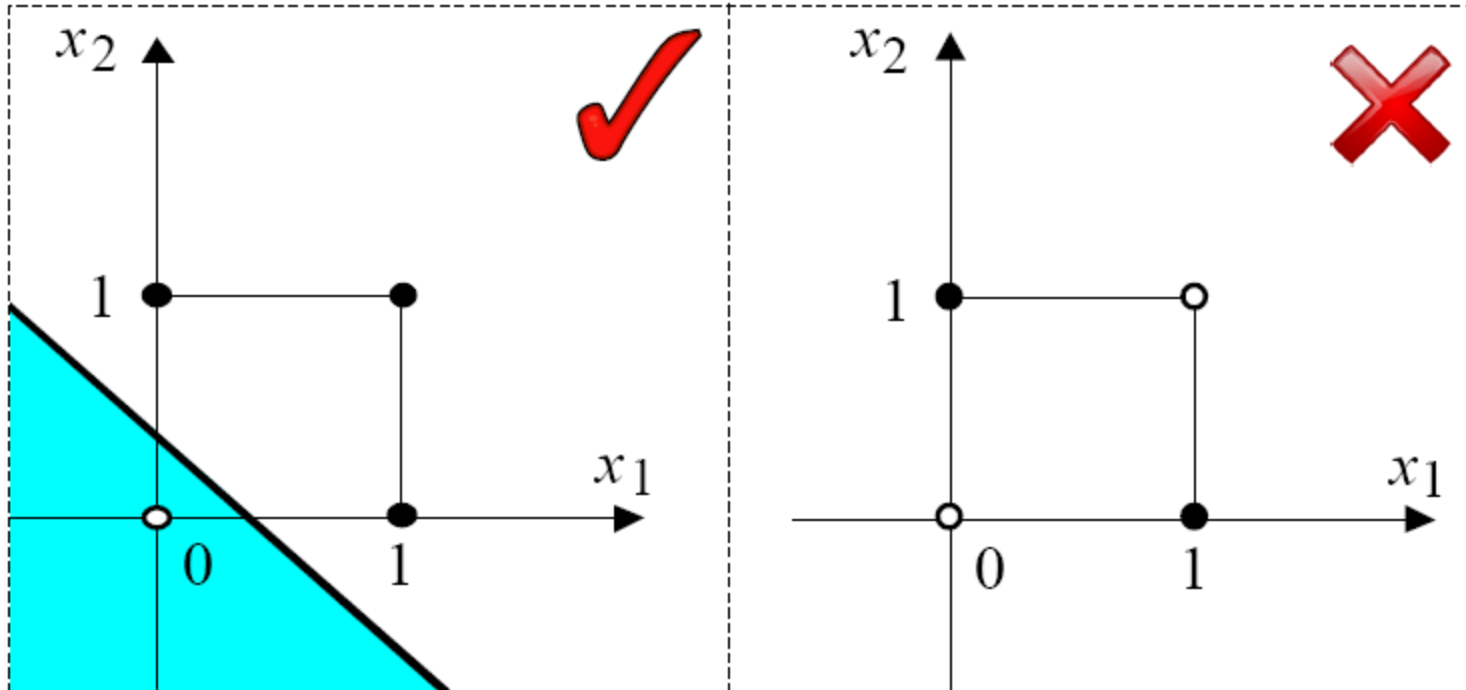
# Drawing Line (AND example)

$$0.1X_1 + 0.1X_2 - 0.2 = 0$$

$$X_2 = -X_1 + 2$$

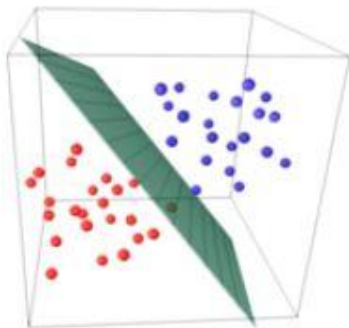
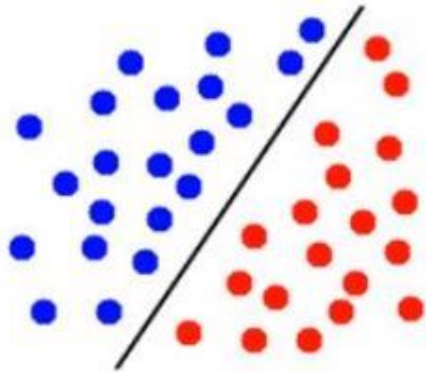


# Summary



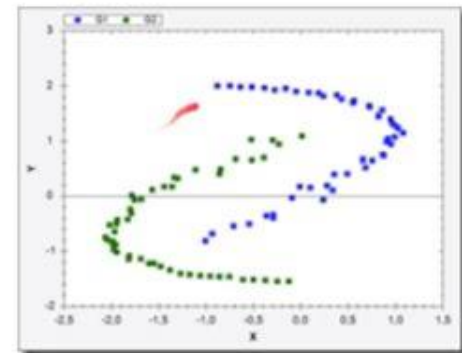
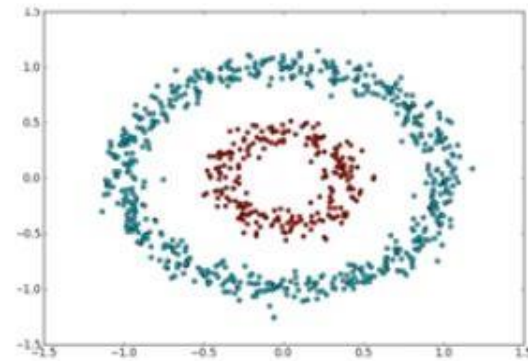
# Summary

Linearly Separable



← 3D

Not Linearly Separable





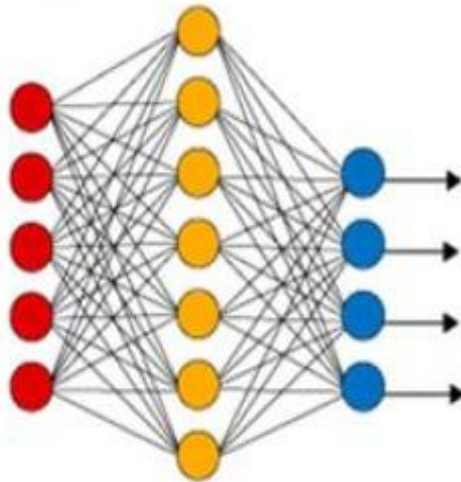
# Summary

- [A Neural Network Playground - TensorFlow](#)

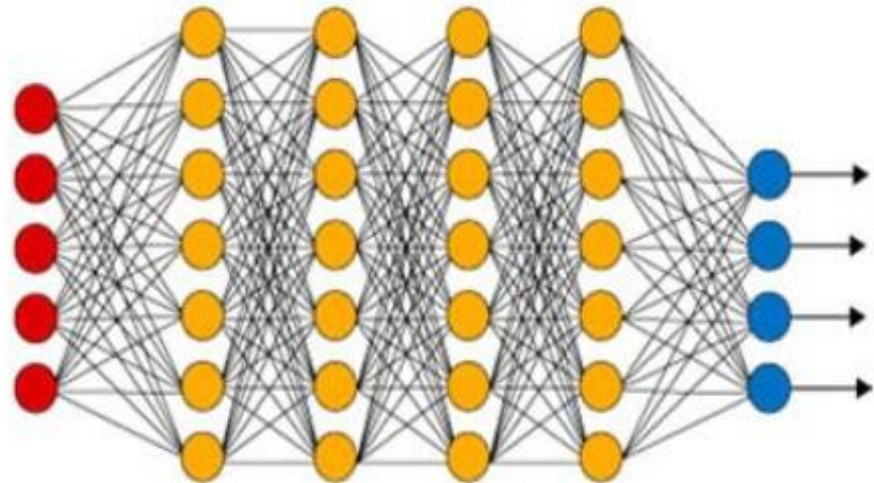
# Next Topic

- Feed-forward neural network

Simple Neural Network



Deep Learning Neural Network



● Input Layer

● Hidden Layer

● Output Layer