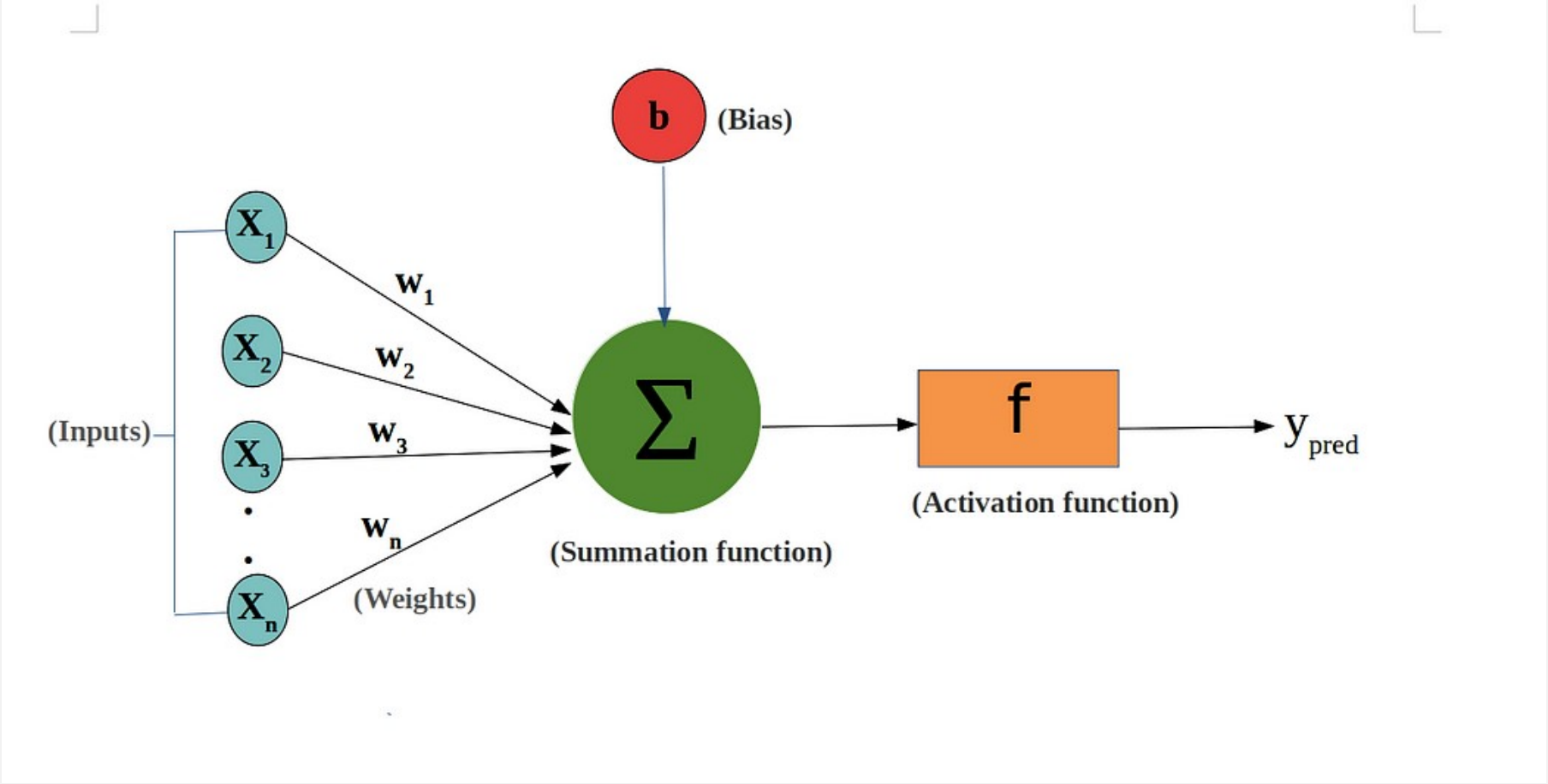
Perceptron Learning Algorithm

Components of the basic Artificial Neuron



What do the weights in a Neuron convey to us?

- Weights associated with each feature, convey the importance of that feature in predicting the output value. Features with weights that are close to zero said to have lesser importance in the prediction process compared to the features with weights having a larger value.
- If the weight associated with a feature is positive it implies that there is a direct relationship between that feature and the target value, and if the weight associated with the feature is negative it implies that there is an inverse relationship between the feature and the target value.

Importance of weights in ANN

- Weights play an important role in changing the orientation or slope of the line that separates two or more classes of data points.
- Weights tell the importance of a feature in predicting the target value.
- Weights tell the relationship between a feature and a target value.

Role of Bias in the ANN?

- **Bias is used for shifting the activation function towards the left of right.**
- **In a simple way, the weights control the strength of the connection between neurons, the bias helps shift the activation function of each neuron.**
- Consider this 1-input, 1-output network that has no bias:

- The output of the network is computed by multiplying the input (x) by the weight (w_o) and passing the result through some kind of activation function (e.g. a sigmoid function.)

- Let us consider a sigmoid activation function to demonstrate the use of bias, we can represent the sigmoid activation function with the mathematical expression as follows:

$$\text{sigmoid function} = \frac{1}{1+e^{-x}}$$

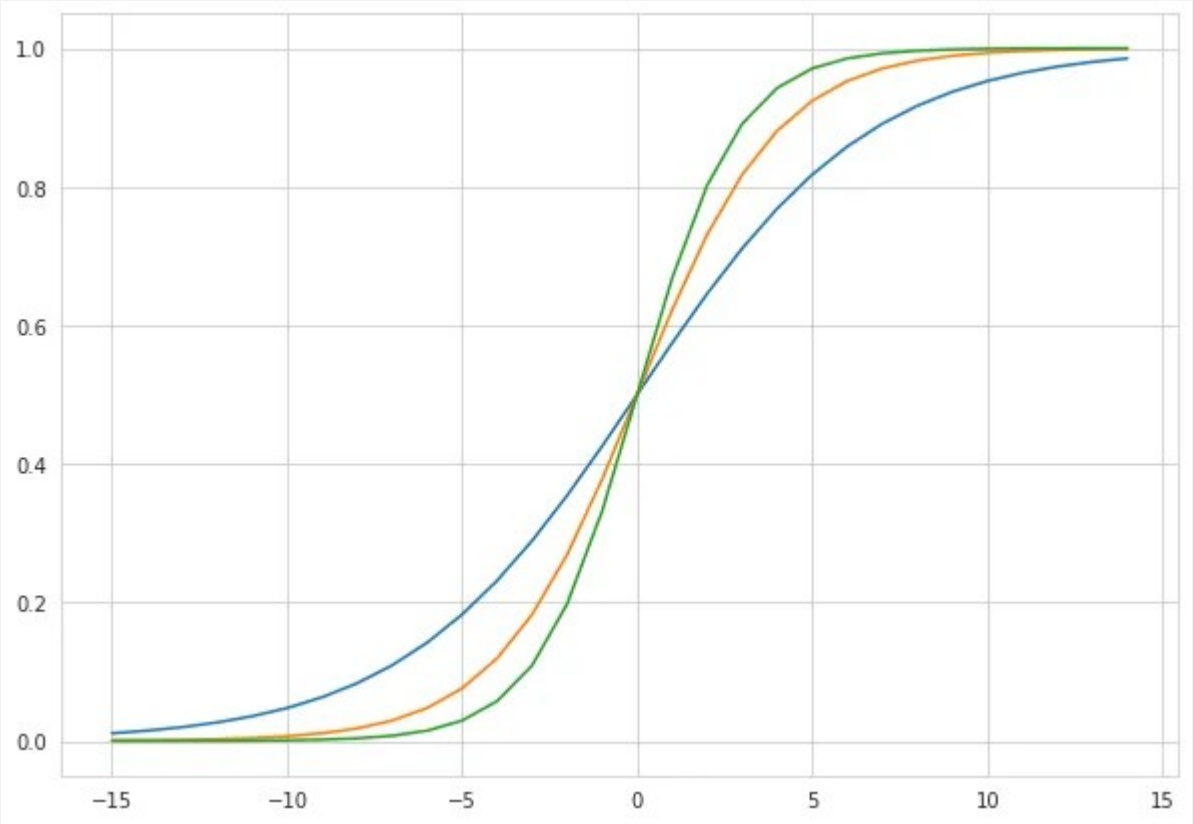
- Replacing x with the equation of a straight line:

$$\text{sigmoid function} = \frac{1}{1+e^{-(W_0 * x + b)}}$$

- Without bias the equation of line passing through centre:

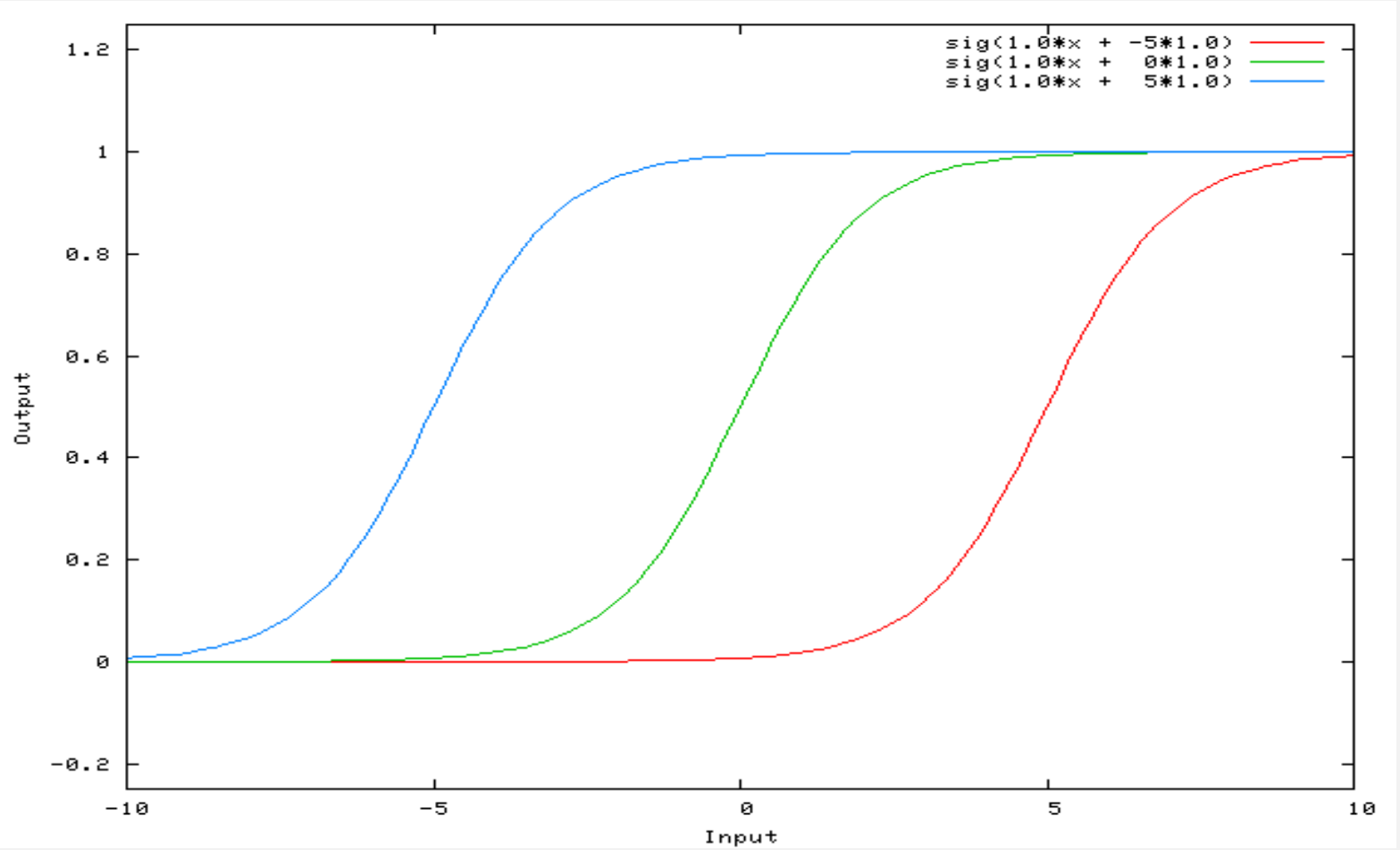
$$\text{sigmoid function} = \frac{1}{1+e^{-(W_0 * x)}}$$

Let us vary different values of w_o with bias b=0



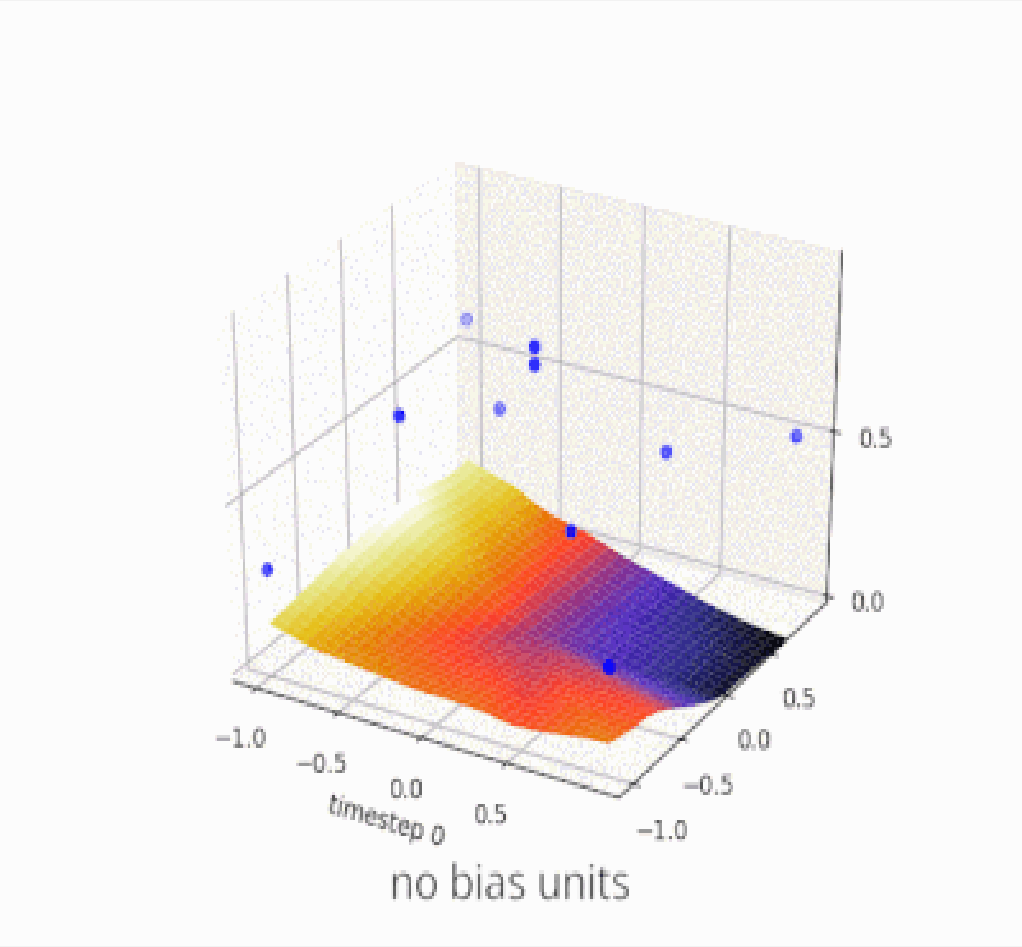
w_o = 0.3 – the blue line in the plot, w_o = 0.5 – the red line in the plot and w_o = 0.7 – the green line in the plot

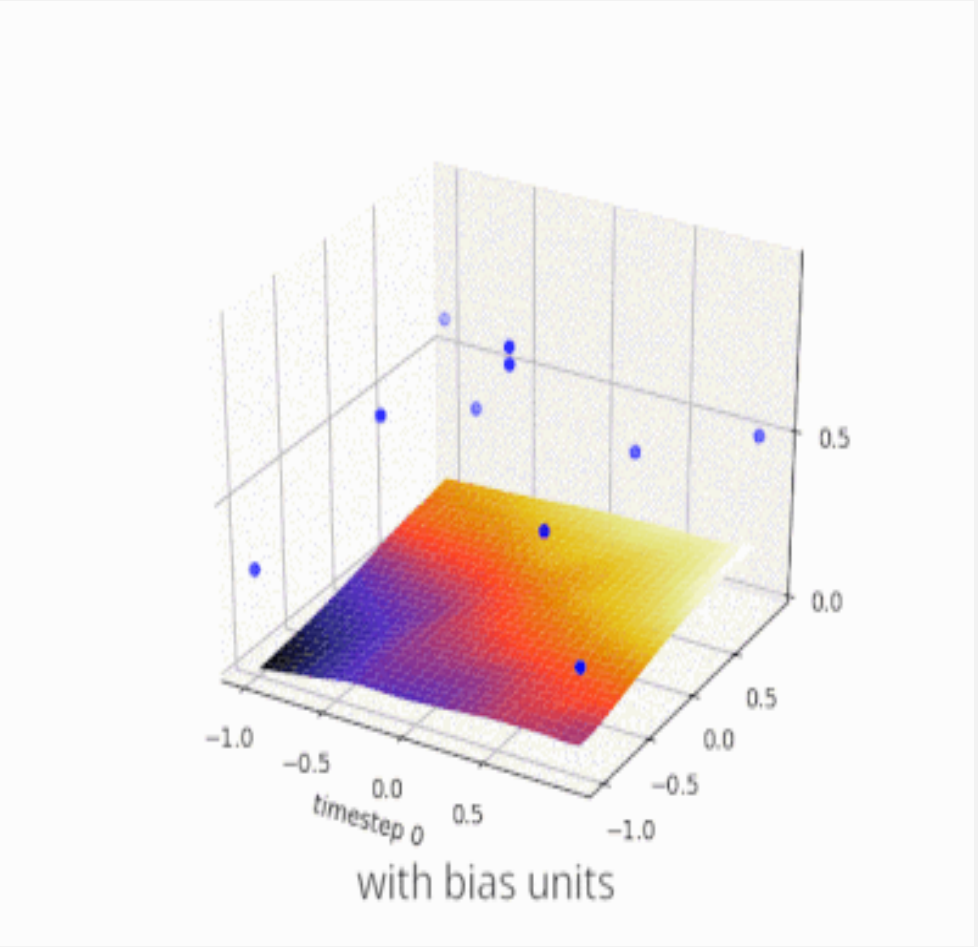
- Even while giving different values of w_o, we could not shift the center of the activation function, in this case, the sigmoid function. Changing the weight w_o essentially changes the "**steepness**" of the sigmoid.
- However, what if you wanted the network to output 0 when x is 2?
- Just changing the steepness of the sigmoid won't really work – you want to be able to shift the entire curve to the right.
- That's exactly what the bias allows you to do. If we add a bias to that network, response of the sigmoid function as shown below.



- Having a weight of -5 for w1 shifts the curve to the right, which allows us to have a network that outputs 0 when x is 2.

3D example


0.5
0.0
−1.0
−0.5
0.0
0.5
timestep 0
0.5
0.0
−0.5
−1.0
no bias units


0.5
0.0
−1.0
−0.5
0.0
0.5
timestep 0
0.5
0.0
−0.5
−1.0
with bias units

Why we need weights and bias in the Artificial Neuron Network.

- **Changing the value of w only changes the steepness of the curve, but there is no way we cans shift the curve towards left or right, the only way to shift the curve towards left or right is by changing the value of bias(b).**

Relationship between decision boundary line and weight vector

- Decision boundary line in 1D input can be replicated for higher dimensions (more than 1D) for plane and hyperplane.
- Linear Algebra is one of the foundational concepts for understanding the relation between hyperplane and weight vector.
- Dot products: Let's first see the concept of dot products in vectors. If a and b are 2-dimensional vectors, the dot product is:

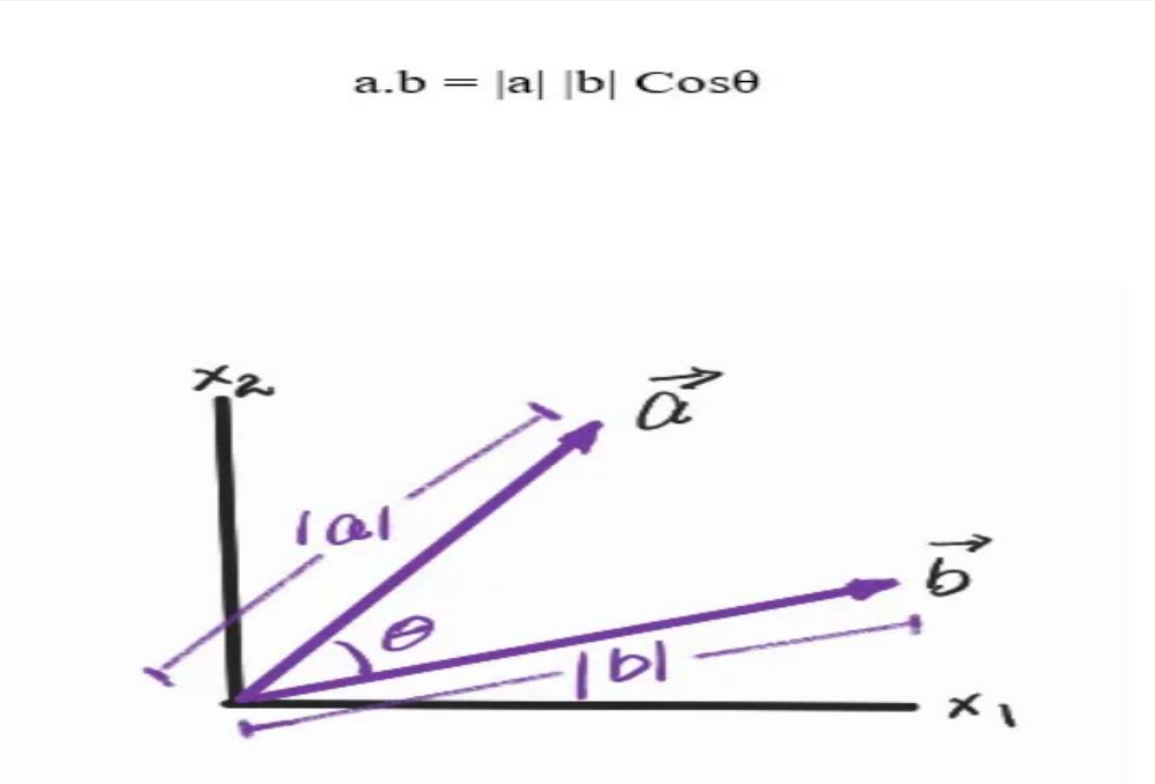
$$a = \begin{bmatrix} a1 \\ a2 \end{bmatrix} \qquad b = \begin{bmatrix} b1 \\ b2 \end{bmatrix}$$

$$a.b = a_1 b_1 + a_2 b_2$$

$$= [a_1\ a_2] \begin{bmatrix} b1 \\ b2 \end{bmatrix}$$

$$= a^T b$$

- Geometrically, dot product can be shown as:



- Equation of Line / Hyperplane
- $y = mx + c$, m = slope and c = y-intercept

Equation of Line / Hyperplane

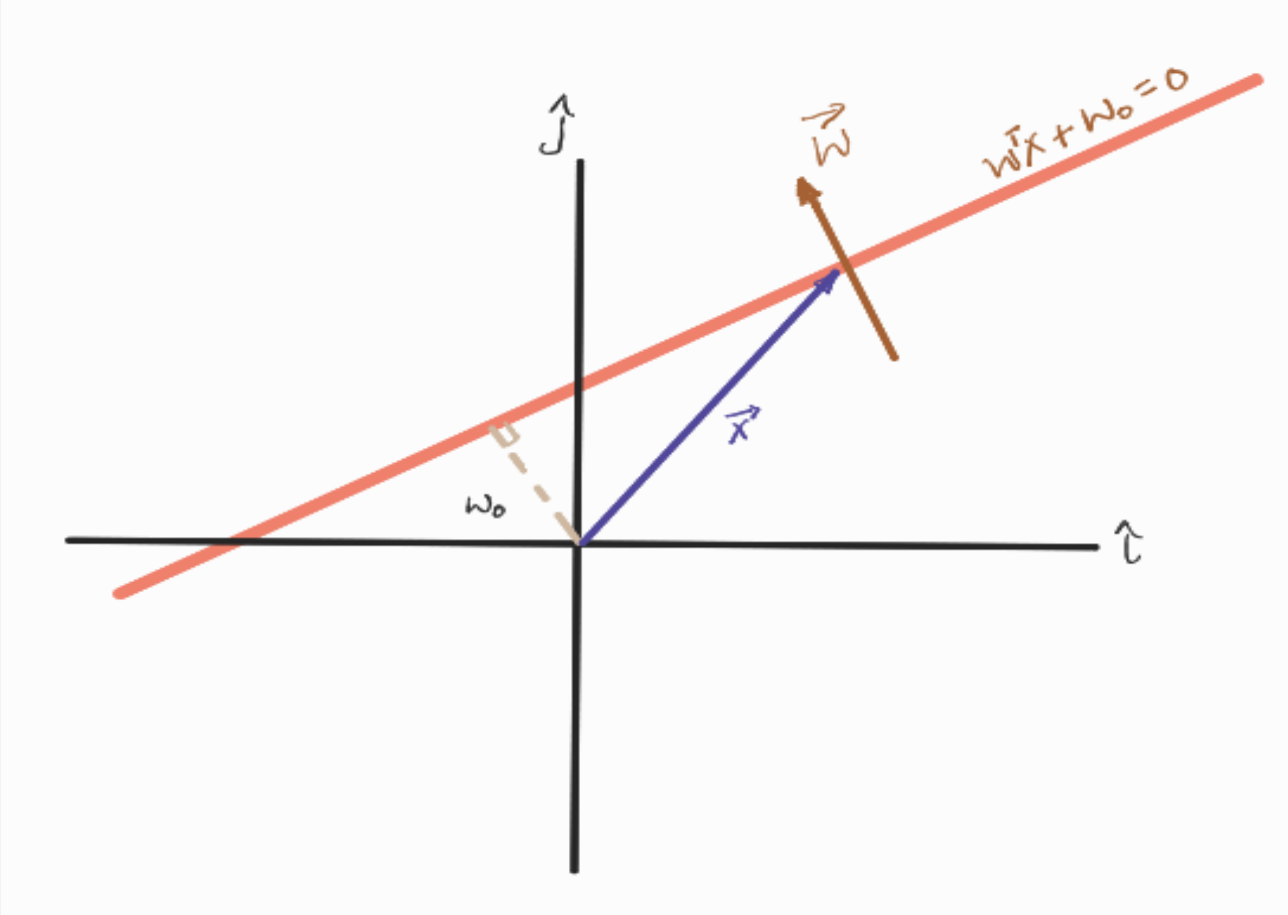
- We can rewrite the above equation in another form:

$$w_1 x_1 + w_2 x_2 + w_0 = 0$$

$$[w_1 \; w_2] \begin{bmatrix} x1 \\ x2 \end{bmatrix} + w_0 = 0$$

$$w^T x + w_{0=0}$$

- Note: The same equation can be extended to higher dimensions for planes and hyperplanes.
- Equation of the decision line $g(x) = w^T x + w_o$
- Any point on the decision line can be expressed as: $w^T x + w_o = 0$



- wTx: This part is the dot product of vector w and vector x. The product is always a constant value equal to -w_o to satisfy the equation $w^T \mathrm{x} + w_o = 0$.
- w_o : This is the constant term. This is the perpendicular distance from the origin to the line (hyperplane).

The vector w orthogonal to the line wTx + w0 = 0 ?

- To prove this we are going to take two points A and B on the line $w^T x + w_o = 0$.
- Given these points are on the line they must satisfy the equation of line which means:

$$g(x_A) = w^T x_A + w_o = 0$$

$$g(x_B) = w^T x_B + w_o = 0$$

- We can also say that f(xA) – f(xB) should also be equal to 0 because individually they are equal to zero.

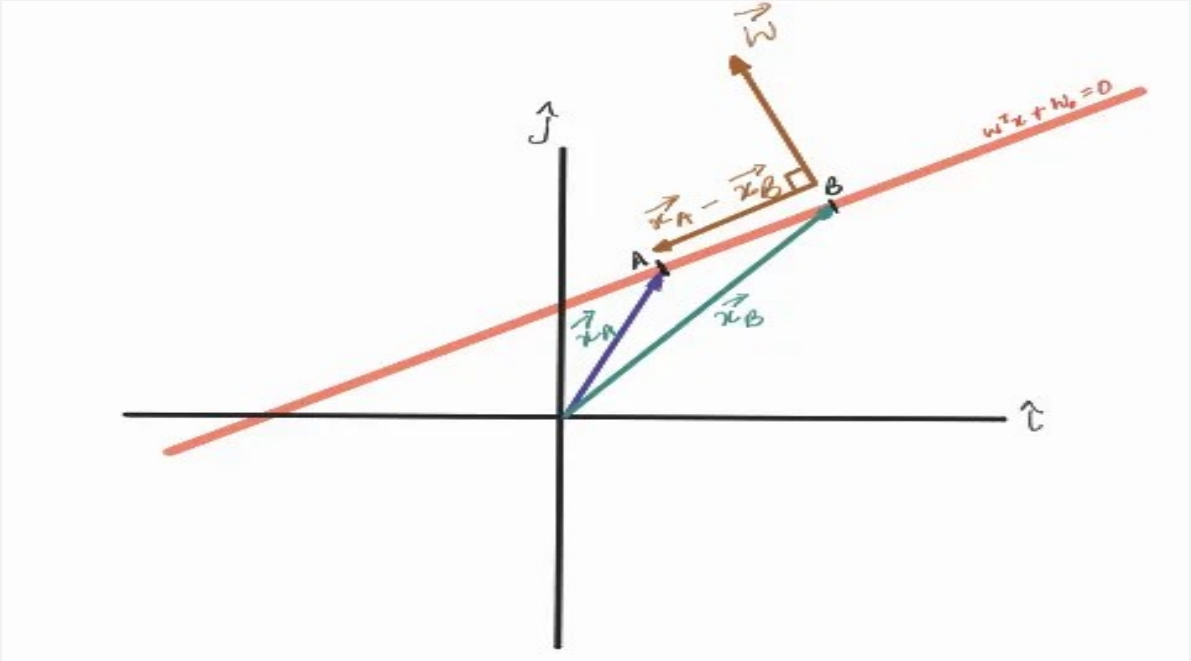
$$f(x_A) - f(x_B) = 0$$

$$(w^T x_A + w_0) - (w^T x_B + w_0) = 0$$

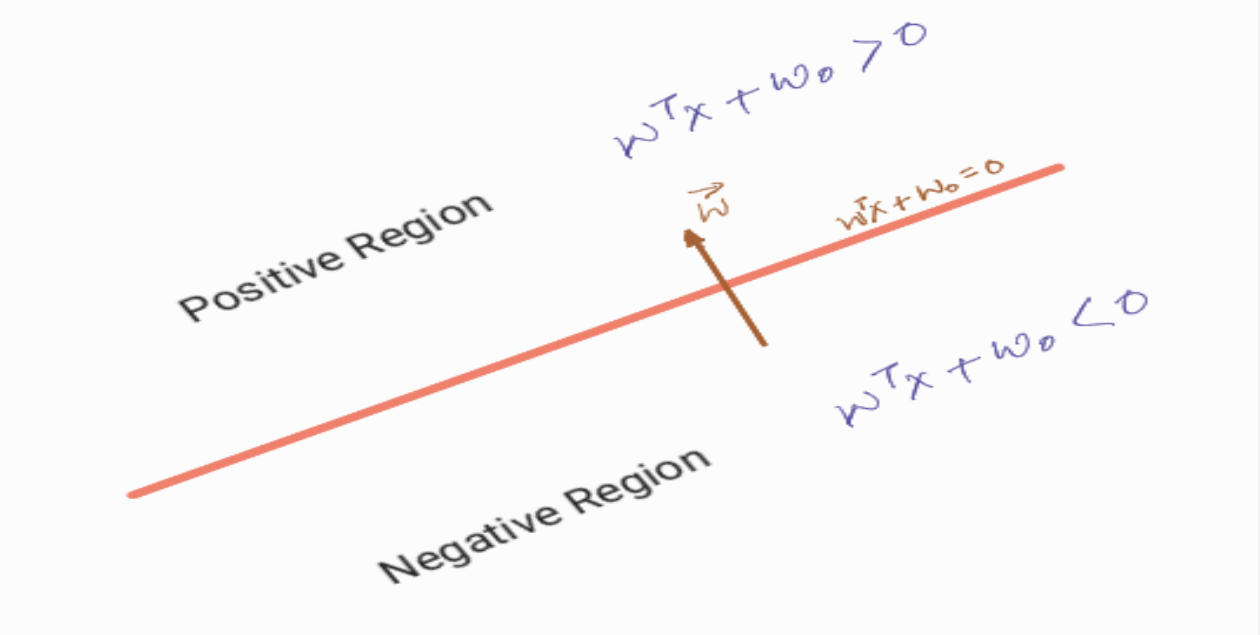
$$w^T (x_A - x_B) = 0$$

$$|w| \, | \, x_A - x_B \, | \, \text{Cos}\theta = 0$$

- From the previous Eqn., it is clear that the angle between w and $(x_A - x_B)$ should be 90^o ($\theta = 90^o$).
- $(x_A - x_B)$ is a vector along the line $w^T x + w_o = 0$ and w must be orthogonal to it.
- This proves that vector w is always orthogonal to the line (hyperplane). which can be seen in the diagram.



- The hyperplane $w^T x + w_o = 0$ acts as a divider that divides the space into two parts.
- Any point on the divider or hyperplane, $w^T x + w_o = 0$
- Any point on the same side of the direction of w, $w^T x + w_o > 0$
- Any point on the opposite side of the direction of w, $w^T x + w_o < 0$



Labeled data and Unlabeled data

- Terms frequently used in Learning:
- **Labeled data**: Data consisting of a series of training examples[(x_1, y_1), (x_2, y_2), …, (x_n, y_n)], where each example is a pair consisting of the sample's input and output value. Y_i is the "ground truth" or "desired response" or "right answer" for each sample in the data. Here y_1, …, y_n are the desired response for each samples in the dataset.

- **Unlabeled data**: Data consisting of only of input values [(x_1), (x_2), …, (x_n)]. It does not come with the "ground truth".

Learning Processes

- The property that is of primary significance for a neural network is **the ability of the network to learn from its environment (dataset)**, and to improve it's performance through learning.
- The improvement in performance takes place over time in accordance with predefined learning algorithm.
- In the learning process, the neural network learns from data points by updating its synaptic weights and bias levels (free parameters) iteratively. This phase of adjustment of free parameters is called training phase.
- Neural network becomes more knowledgeable about its ernvironment after each iteration of the learning process.

- We define learning in the context of neural networks as follows:
- **Learning is a process in which the free parameters of a neural network are adjusted through interactions with the environment in which the network operates.**
- This definition of learning process implies the following sequence of events:
  1. The neural network is simulated by an environment.
  2. The neural network undergoes changes in its free parameters iteratively through interactions with the environment.
  3. The neural network responds in a new way to the environment because of the changes that have occurred in its internal structure.

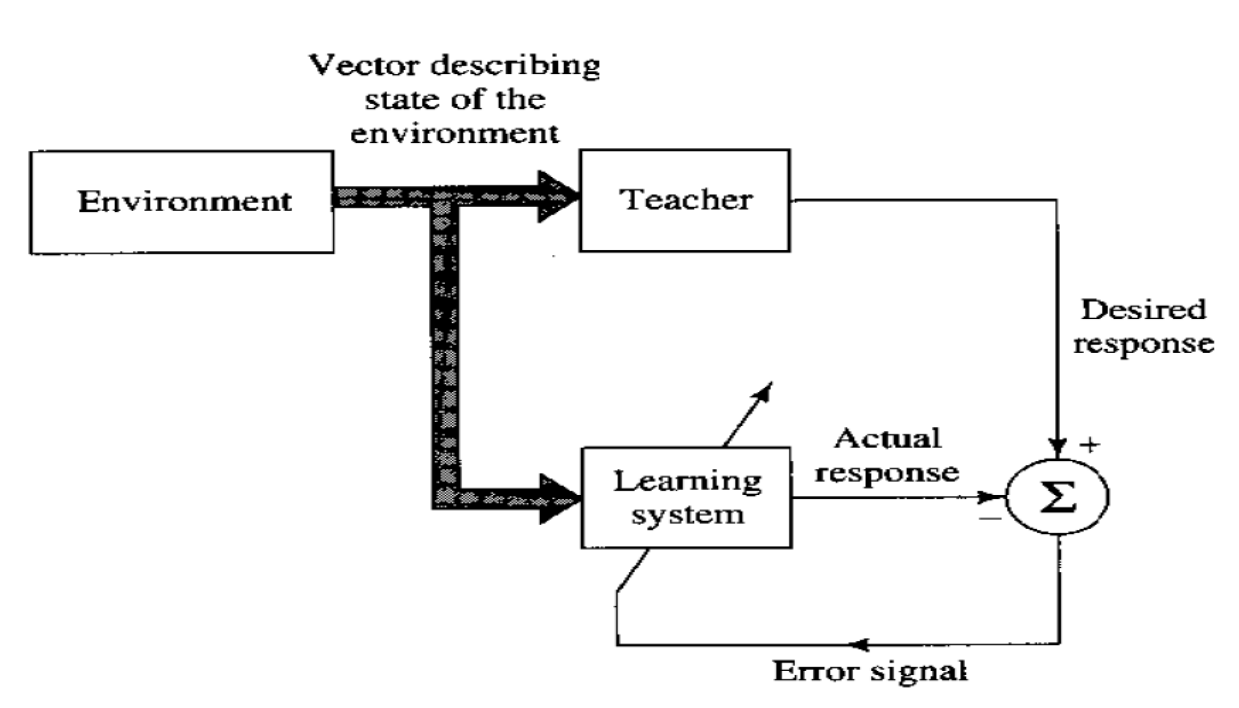
- A set of well-defined rules for solving a learning process is called a learning algorithm.
- A learning algorithm that refers to a model (optimal synaptic weights parameters) of the environment in which neural network operates.
- There is no single learning algorithm for designing neural networks. Various tasks require different learning algorithms to update the free parameters of neural networks.
- It is also evident that the choice of learning algorithm depends on the type of neural network interconnection, such as feedforward or feedback neural network architectures.

Types of Neural Network Architectures

- For different neural networks, different learning algorithm are used. A few neural networks architectures are given below:
- Perceptron (Single Layer Perceptron)
- Adaline (Adaptive Linear Neuron)
- Madaline (Multiple Adaptive Linear Neuron)
- MLP (Multi-Layer Perceptron)
- RBFN (Radial Basis Functional Neural Network)
- SOM (Self Organizing Map)
- RNN (Recurrent Neural Network)
- CNN (Convolutional Neural Network)

Learning with a Teacher

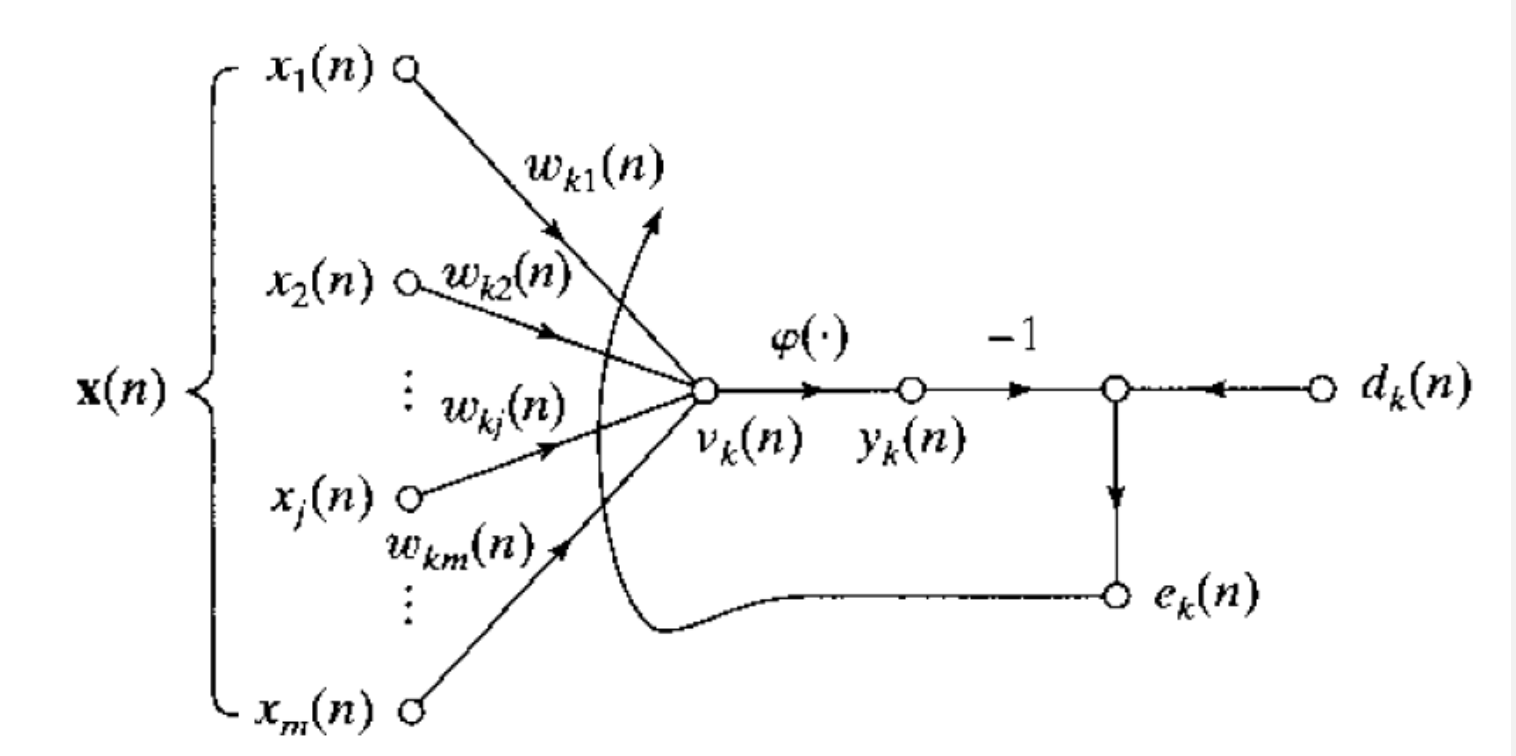
- Learning with a teacher is also referred as supervised learning.



Block diagram of learning with a teacher

- Supervised learning is also termed as Error-Correction Learning.
- Consider the simple case of a neuron k constituting the only computational node in the output layer as shown in following Fig.
- During the learning process, in the n^{th} iteration weights of the network are adjusted. The output of the neuron k, actual response or predicted output is denoted by $y_k(n)$.
- This output signal is compared to a desired response or target output, denoted by $d_k(n)$.
- Consequently, an error signal, denoted by $e_k(n)$, is produced. By definition, we thus have $e_k(n) = d_k(n) - y_k(n)$.

- The corrective adjustments are designed to make the output signal $y_k(\mathrm{n})$ *come closer to the desired response* $d_k(\mathrm{n})$ in a step-by-step manner. This objective is achieved by minimising a cost function.



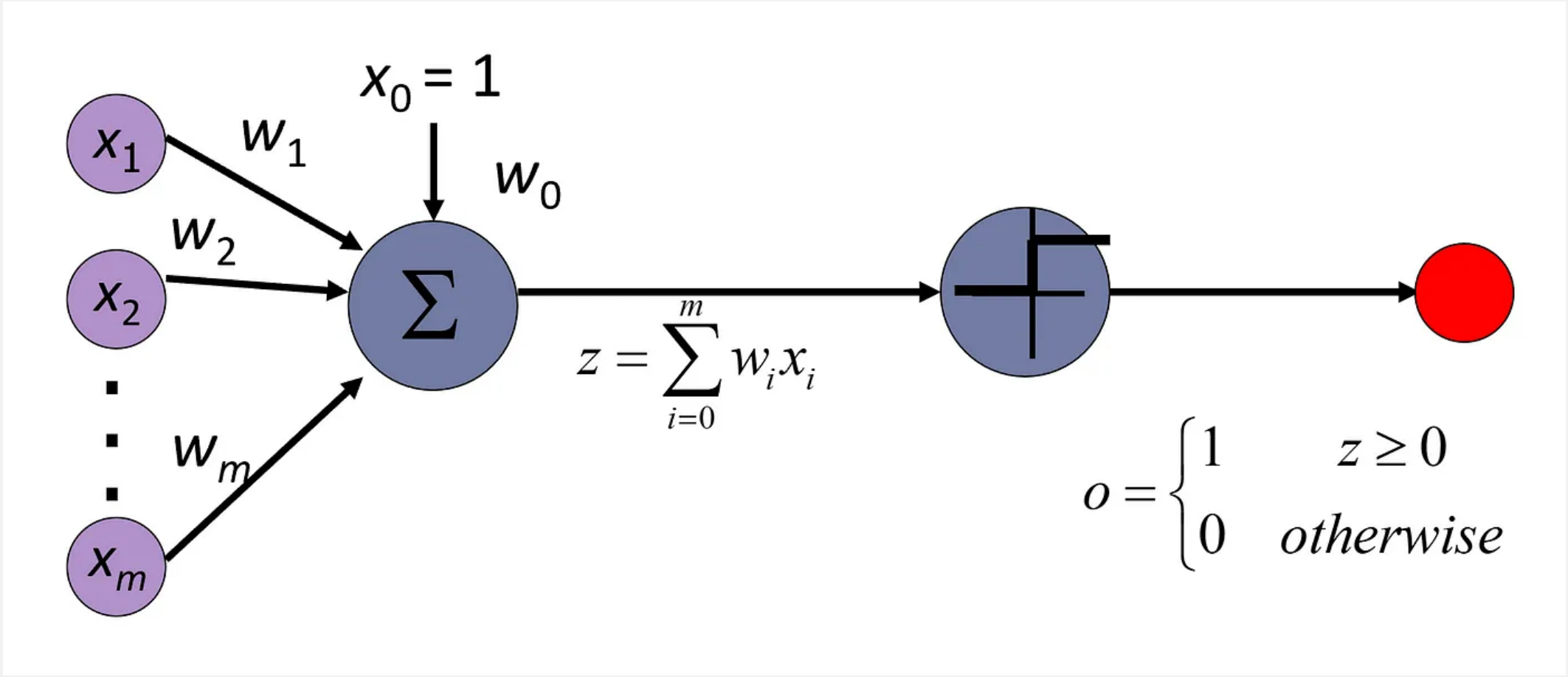
Illustrating error-correction learning

- The goal is to learn a function that maps the input feature vector to the correct class label or predicted output.
- Supervised learning is often used for tasks such as **classification** and **regression**.
- **Classification:** In classification tasks, the algorithm learns to assign data points (feature vectors) to predefined categories or classes. For example, it can classify emails as spam or not spam, detect diseases based on medical images, or recognize handwritten digits.
- **Regression:** Regression tasks involve predicting a continuous numerical value. For instance, it can predict housing prices based on features like square footage, number of bedrooms, and location, or forecast stock prices based on historical data.

Perceptron Model

- Frank Rosenblatt (1958) proposed the Perceptron as the first model for learning with a teacher (supervised learning).
- The perceptron is the simplest form of a neural network used for classification of data points said to be linearly separable (i.e., data points lie on opposite sides of the hyperplane).
- Basically, it consists of a single neuron with adjustable synaptic weights and bias.
- The perceptron build around a single neuron is limited to performing data classification with only two classes and works effectively only when the data distribution of these two classes is linearly separable.

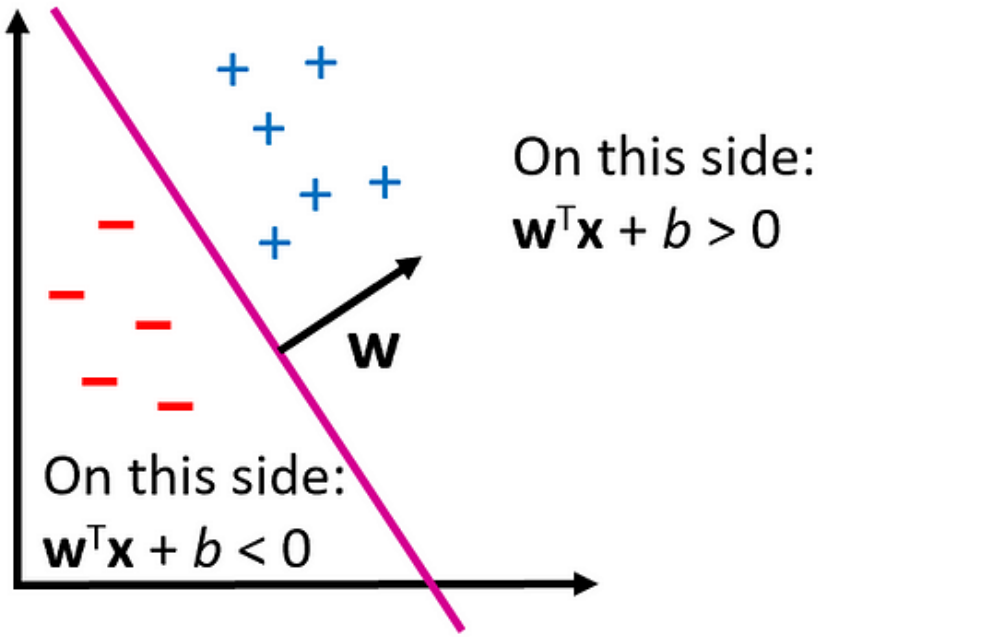
Perceptron Architecture



Perceptron

- Perceptron is built around a nonlinear neuron, namely, the McCulloch-Pitts model of a neuron with additional weights for each input connection and an added bias term.
- Perceptron is a linear classifier that means it is capable of learning only linearly separable problems, i.e., problems where the decision boundary between the positive and the negative examples is a linear surface (a hyperplane).

Linearly separable problem

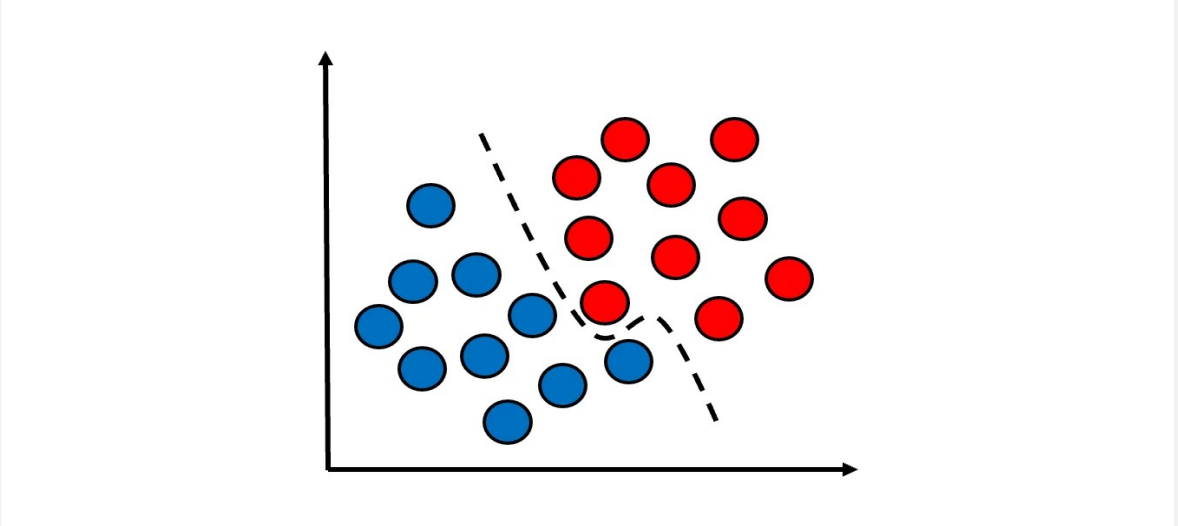


Hyperplane perpendicular to $\mathbf{w}$

$H = \{\mathbf{x}: \mathbf{w}^T\mathbf{x} + b = 0\}$

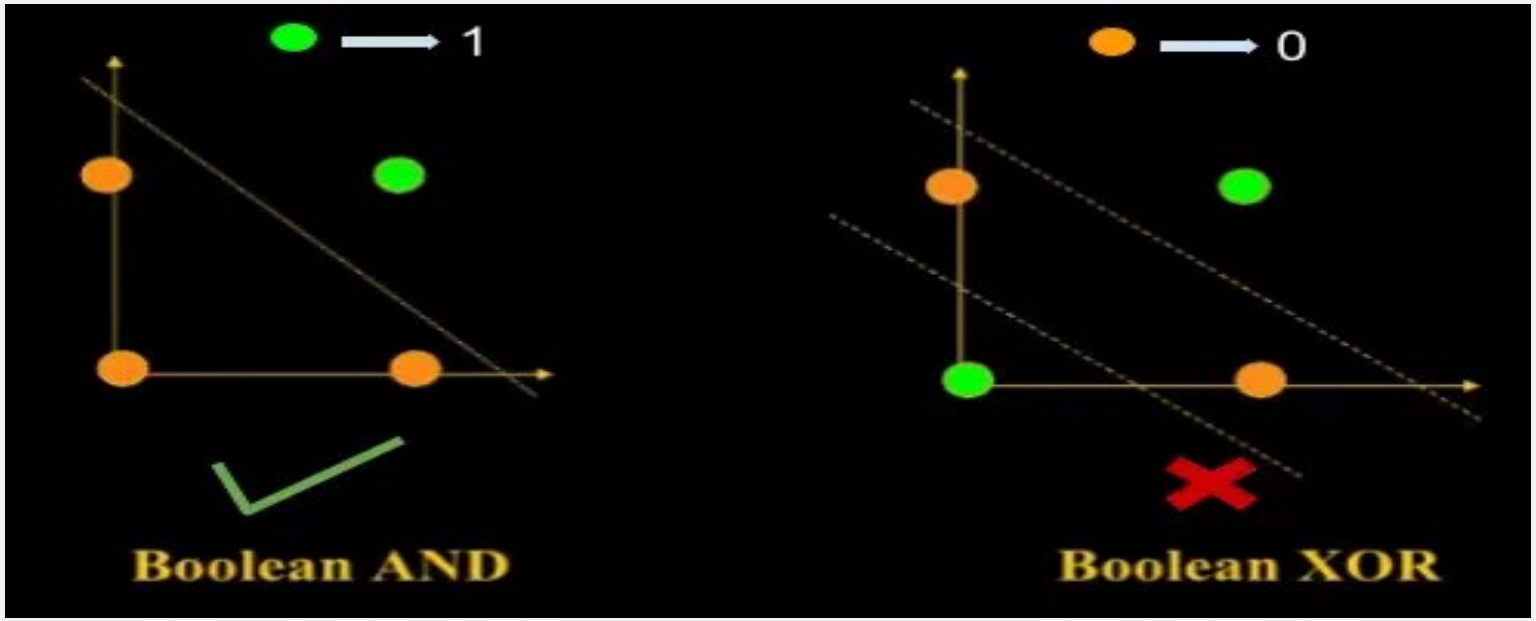
Nonlinearly separable problem

- When dataset is nonlinearly separable, then perceptron cannot classify correctly all the data points in the dataset by using a single hyperplane.



Nonlinearly separable dataset

Perceptron: Linearly separable classifier



- Perceptron can only learn linearly separable problems for examples Logic Gates like AND, OR, or NAND.
- For non-linear problems such as the Boolean XOR problem, it does not work.

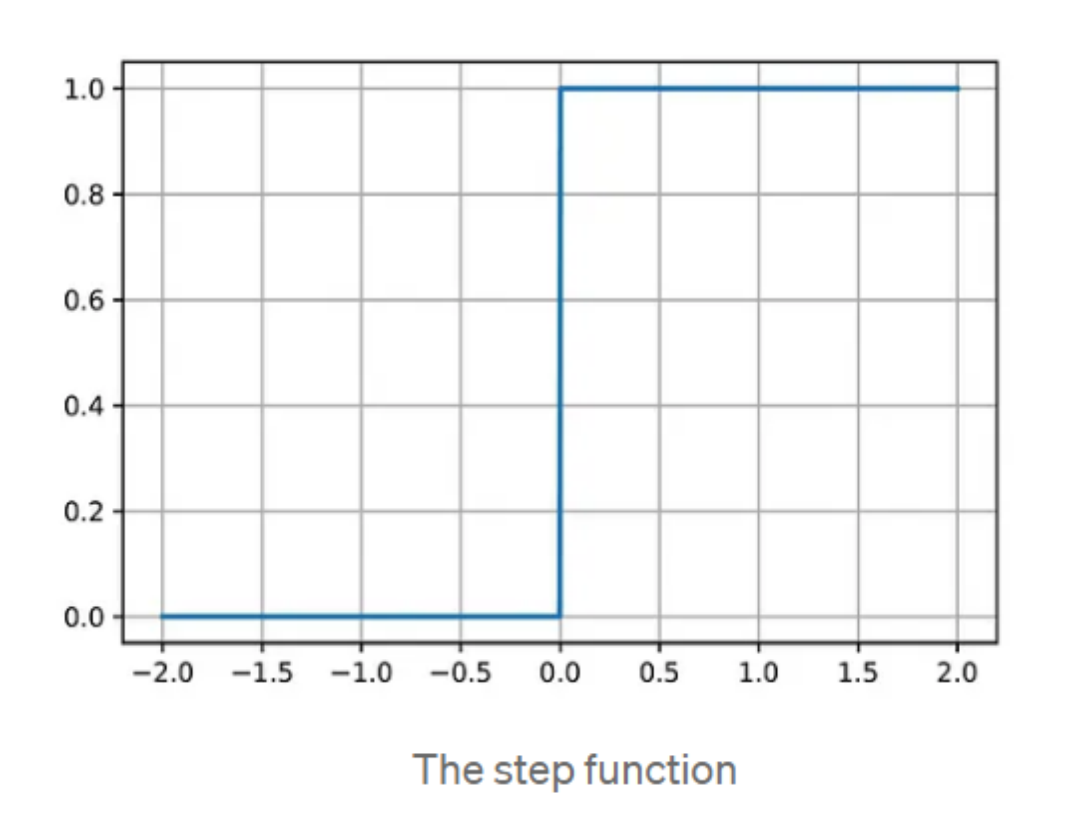
Activation Function for Perceptron

- The two most common activation functions used in perceptron are:
  1. Step function or Heaviside function: It is a function whose value is 0 for negative inputs and 1 for non-negative inputs.

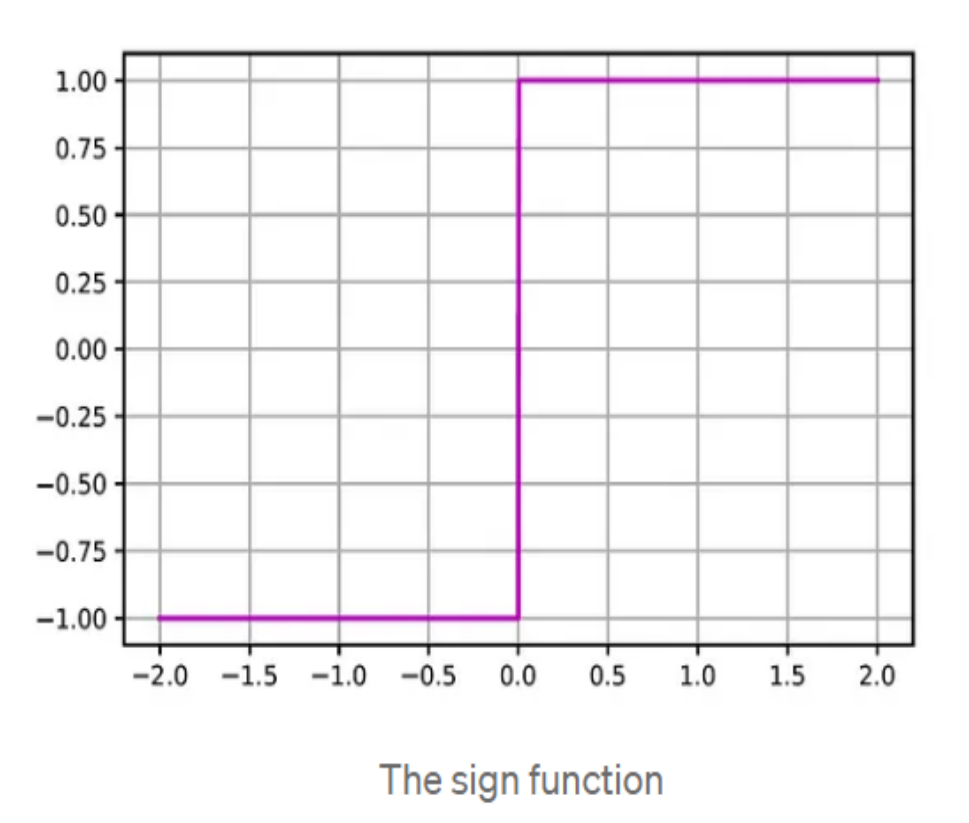
     When $z \geq 0; f(z) \rightarrow 1, otherwise\ f(z) \rightarrow 0.$
  1. Signum (sign) function is a function whose value is -1 for negative inputs and 1 for non-negative inputs.

     When $z \geq 0; f(z) \rightarrow 1, otherwise\ f(z) \rightarrow -1.$

Activation Functions for Perceptron



The step function

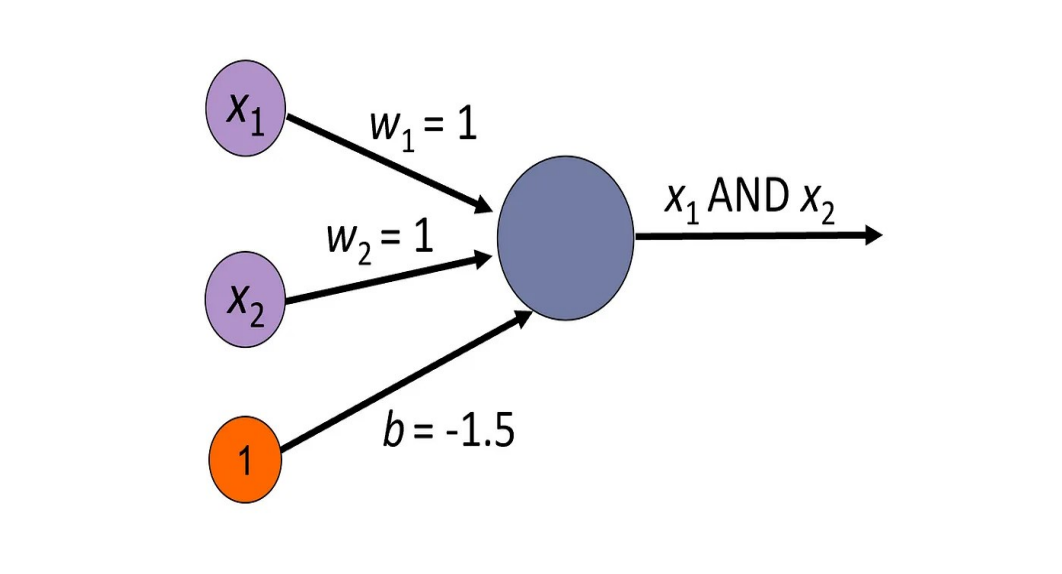


The sign function

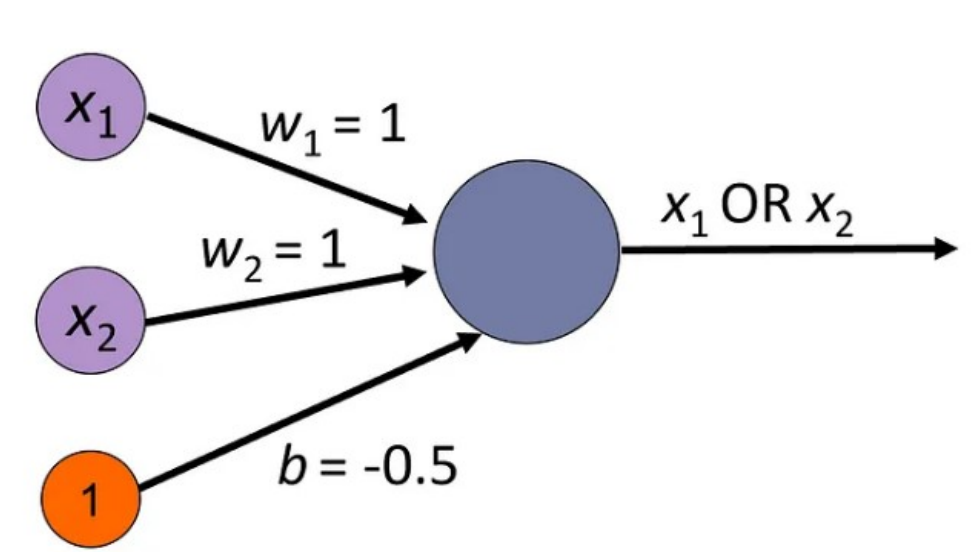
Applying an activation function f(z) on the net input (z) that generates a binary output (o) --> (0/1 or -1/+1).

Implementing Logic Gates with Perceptrons

- To demonstrate how perceptron works, let's try to build perceptron that computes the logical functions AND and OR.



A perceptron that computes the logical AND function



A perceptron that computes the logical OR function

Assignment: Try to build a perceptron for the NAND function

Augmented input and weight vectors

- The synaptic weights $w_1, w_2, \ldots, w_n$ and bias of the perceptron can be adapted on an iteration-by-iteration basis. For the adaptation we may use an error-correction rule known as the perceptron convergence algorithm.

- Hyperplane is expressed as $\mathrm{g(x)} = \sum_{i=0}^{m} w_i x_i + b = 0$.

- The bias b is treated as a synaptic weight driven by a fixed input equal to +1.

$x(n) = [1, x_1(n), x_2(n), \ldots, x_m(n)]^T$ and $\mathrm{w}(n) = [w_0(n), w_1(n), w_2(n), \ldots, w_m(n)]^T$

- To simplify the hyperplane equation as:

$$\sum_{i=0}^{m} w_i x_i = w^T x = 0$$

The Perceptron Learning Rule

- For the perceptron to function properly, the two classes C_1 and C_2 must be linearly separable.
- This, in turn, means that the patterns to be classified must be sufficiently separated from each other then perceptron learning can draw a hyperplane as a decision boundary.
- The training process involves the adjustment of the weight vector w in such a way that the two classes C_1 and C_2 are linearly separable by placing the hyperplane in a position that produce zero or very less error. Then, the training phase is terminated.

- For an input x, there exit a weight vector w such that satisfy following conditions:
  - $w^T x > 0$ for every input vector x belonging to class C_1
  - $w^T x < 0$ for every input vector x belonging to class C2
  - $w^T x = 0$ for some points lie on the hyperplane

1. If the nth member of the training set, $\mathbf{x}(n)$, is correctly classified by the weight vector $\mathbf{w}(n)$ computed at the nth iteration of the algorithm, no correction is made to the weight vector of the perceptron in accordance with the rule:

$$\mathbf{w}(n+1) = \mathbf{w}(n) \quad \text{if } \mathbf{w}^T\mathbf{x}(n) > 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1$$

$$\mathbf{w}(n+1) = \mathbf{w}(n) \quad \text{if } \mathbf{w}^T\mathbf{x}(n) \leq 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2$$

2. Otherwise, the weight vector of the perceptron is updated in accordance with the rule

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta(n)\mathbf{x}(n) \quad \text{if } \mathbf{w}^T(n)\mathbf{x}(n) > 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2$$

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta(n)\mathbf{x}(n) \quad \text{if } \mathbf{w}^T(n)\mathbf{x}(n) \leq 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1$$

where the *learning-rate parameter* $\eta(n)$ controls the adjustment applied to the weight vector at iteration n.

Simplification of Perceptron rules

d_i	y_i	$d_i - y_i$
1	1	0
0	0	0
0	1	-1
1	0	1

$$w(n+1) = w(n) + \eta(d_i - y_i)x_i$$

When $x_i \in C_2$ and $W^T x > 0$; $w(n+1) = w(n) - \eta x_i$

When $x_i \in C_1$ and $W^T x < 0$; $w(n+1) = w(n) + \eta x_i$

Variables and Parameters:

$\mathbf{x}(n)$ = $(m+1)$-by-1 input vector
$= [+1, x_1(n), x_2(n), \ldots, x_m(n)]^T$
$\mathbf{w}(n)$ = $(m+1)$-by-1 weight vector
$= [b(n), w_1(n), w_2(n), \ldots, w_m(n)]^T$
$b(n)$ = bias
$y(n)$ = actual response (quantized)
$d(n)$ = desired response
η = learning-rate parameter, a positive constant less than unity

1. *Initialization.* Set $\mathbf{w}(0) = \mathbf{0}$. Then perform the following computations for time step $n = 1, 2, \ldots$.

2. *Activation.* At time step n, activate the perceptron by applying continuous-valued input vector $\mathbf{x}(n)$ and desired response $d(n)$.

3. *Computation of Actual Response.* Compute the actual response of the perceptron:

$$y(n) = \text{sgn}[\mathbf{w}^T(n)\mathbf{x}(n)]$$

where $\text{sgn}(\cdot)$ is the signum function.

4. *Adaptation of Weight Vector.* Update the weight vector of the perceptron:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)$$

where

$$d(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \\ -1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \end{cases}$$

5. *Continuation.* Increment time step n by one and go back to step 2.

- This learning rule is applied to all the training samples sequentially (in an arbitrary order). It typically requires more than one iteration over the entire training set (called an epoch) to find the correct weight vector (i.e., the vector of a hyperplane that linearly separates two group of data points.
- Note that the weights are typically initialized to small random values in order to break the symmetry (if all the weights were equal, then the output of the perceptron would be constant for every input), while the bias is initialized to zero.

Limitations of the Perceptron Model

- Single layer perceptron is limited in its computational power since they can solve only linearly separable problems.
- The XOR problem is not linearly separable, therefore linear models such as perceptron cannot solve it.
- A Multilayer Perceptron consists of more than two layers (input, hidden, and output) to solve more complex and non-linear problems such as the XOR problem.