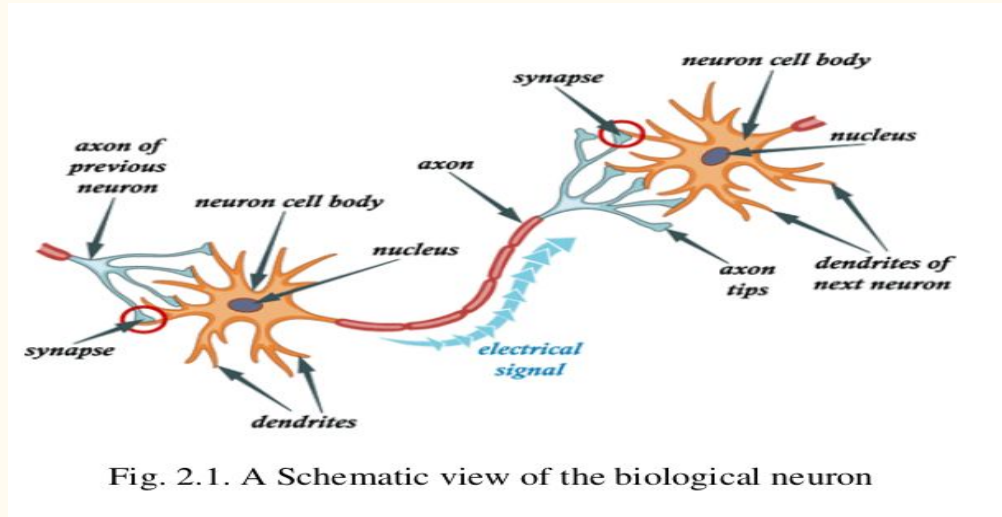


# MODULE 4

—

# THE BIOLOGICAL NEURON

- The human brain consists of a large number, more than a billion of neural cells that process information.
- Each cell works like a simple processor.
- The massive interaction between all cells and their parallel processing only makes the brain's abilities possible.



- Dendrites are branching fibres that extend from the cell body or soma. Soma or cell body of a neuron contains the nucleus and other structures, support chemical processing and production of neurotransmitters.
- Axon is a singular fiber carries information away from the soma to the synaptic sites of other neurons (dendrites and somas), muscles, or glands.
- Synapse is the point of connection between two neurons or a neuron and a muscle or a gland.
- Electrochemical communication between neurons take place at these junctions.

# FUNDAMENTALS OF ANN

- Neural computing is an information processing paradigm, inspired by biological system, composed of a large number of highly interconnected processing elements(neurons) working in union to solve specific problems.
- Artificial neural networks (ANNs), like people, learn by example.
- An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process.
- Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well.

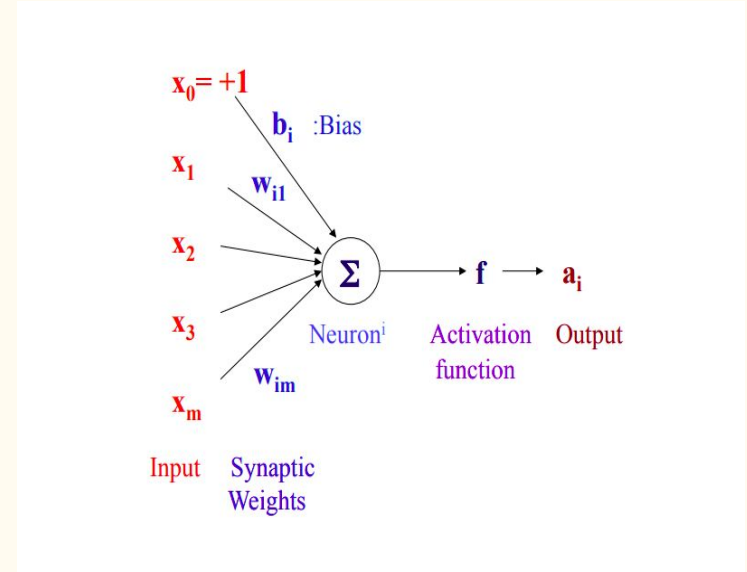
# ARTIFICIAL NEURON MODEL

- An artificial neuron is a mathematical function conceived as a simple model of a real (biological) neuron.
- Inputs to the network are represented by the mathematical symbol,  $x_n$ .
- Each of these inputs are multiplied by a connection weight,  $w_n$

$$\text{sum} = w_1 x_1 + \dots + w_n x_n$$

- These products are simply summed, fed through the transfer function,  $f(\ )$  to generate a result and then output.

$$\text{Output} = f(w_1 x_1 + \dots + w_n x_n)$$



**Weight :** Each weight represents the strength of the connection between the two nodes it connects. The weight values are first randomly initialized and then learned, updated, and optimized by the network during the training process.

**Bias:** Bias can be incorporated as another weight clamped to a fixed input of  $+1.0$

This extra free variable (bias) makes the neuron more powerful.

**Learning Rate :** The amount that the weights are updated during training is referred to as the step size or the “learning rate.”

Specifically, the learning rate is a configurable hyperparameter used in the training of neural networks that has a small positive value, often in the range between 0.0 and 1.0.

## Biological Neuron versus Artificial Neural Network

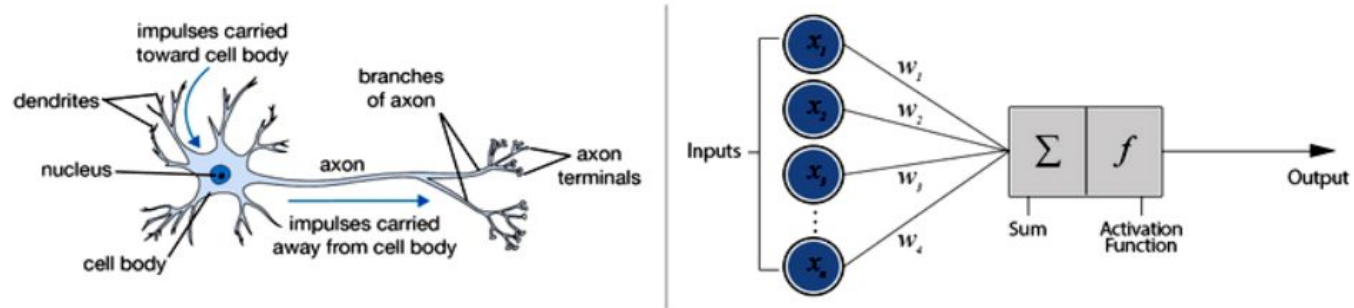


Fig.2.2 Model representation of a biological neuron with multiple inputs

Human	Artificial
Neuron	Processing Element
Dendrites	Combining Function
Cell Body	Transfer Function
Axons	Element Output
Synapses	Weights

# Layers

- An artificial neural network is made of multiple neural layers that are stacked on top of one another.
- Each layer is made up of several neurons stacked in a row.
- We distinguish three types of layers: Input, Hidden and Output layer.
- **Input Layer** : The input layer of the model receives the data that we introduce to it from external sources like a images or a numerical vector. It is the only layer that can be seen in the entire design of a neural network that transmits all of the information from the outside world without any processing.
- **Hidden Layer** : They are intermediary layers that do all calculations and extract the features of the data. The search for hidden features in data may comprise many interlinked hidden layers.
- **Output Layer** : The final prediction is made by the output layer using data from the preceding hidden layers. It is the layer from which we acquire the final result, hence it is the most important.



# Learning

- In artificial neural networks, learning refers to the method of modifying the weights of connections between the nodes of a specified network.
- The learning ability of a neural network is determined by its architecture and by the algorithmic method chosen for training.
- Supervised Learning : Learns by using labelled data , supervision
- Unsupervised Learning : Trained using unlabelled data without any guidance , No supervision
- Reinforcement Learning : Works on interacting with the environment, No supervision

# Learning Rules

- Learning rule updates the weights and bias levels of a network when certain conditions are met in the training process.
- It is a crucial part of the development of the Neural Network.
- Applying learning rule is an iterative process. It helps a neural network to learn from the existing conditions and improve its performance.
- Different learning rules in the Neural network are,
  - a. Hebbian Learning Rule
  - b. Perceptron Learning Rule
  - c. Competitive Learning Rule
  - d. Delta Learning Rule
  - e. Outstar Learning Rule

# Hebbian Learning Rule

Hebbian learning is an unsupervised learning rule that works by adjusting the weights between two neurons in proportion to the product of their activation.

$$\Delta w = \alpha x_i y$$

$\Delta w$  is the change in weight

$\alpha$  is the learning rate

$x_i$  is the input vector and,

$y$  is the output

$$W \text{ (new)} = W \text{ (old)} + X_i * Y$$

# Perceptron Learning Rule

Developed by Rosenblatt, the perceptron learning rule is an error correction rule, used in a feed forward network.

This rule works by finding the difference between the actual and desired outputs and adjusts the weights according to that.

$$\Delta w = \eta(y - \hat{y})x$$

$\Delta w$  is the change in weight

$\eta$  is the learning rate

$x$  is the input vector

$y$  is the actual label of the input vector

$\hat{y}$  is the predicted label of the input vector

# Competitive Learning Rule

- The competitive learning rule is unsupervised in nature and as the name says, is based on the principle of competition amongst the nodes.
- That is why it is also known as the ‘Winner takes it all’ rule.

$$\Delta w_{ij} = \eta * (x_i - w_{ij})$$

$\Delta w_{ij}$  is the changes in weight between the  $i$ th input neuron and  $j$ th output neuron

$\eta$  is the learning rate

$x_i$  is the input vector

$w_{ij}$  is the weight between the  $i$ th input neuron and  $j$ th output neuron

# Delta Learning Rule

Developed by Bernard Widrow and Marcian Hoff, Delta learning rule is a supervised learning rule with a continuous activation function.

The main aim of this rule is to minimize error in the training patterns and thus, it is also known as the least mean square method.

$$\Delta w_{ij} = \eta * (d_j - y_j) * f'(h_j) * x_i$$

$\Delta w_{ij}$  is the changes in weight between the  $i$ th input neuron and  $j$ th output neuron

$\eta$  is the learning rate

$d_j$  is the target output

$y_j$  is the actual output

$f'(h_j)$  is the derivative of the activation function

$x_i$  is the  $i$ th input

# Outstar Learning Rule

Developed by Grossberg, Out Star learning is a supervised learning rule that works with a pair of nodes arranged in a network layer.

$$\Delta w_{ij} = \eta * (x_i - w_j) * f(w_j)$$

$\Delta w_{ij}$  is the change in weight

$\eta$  is the learning rate

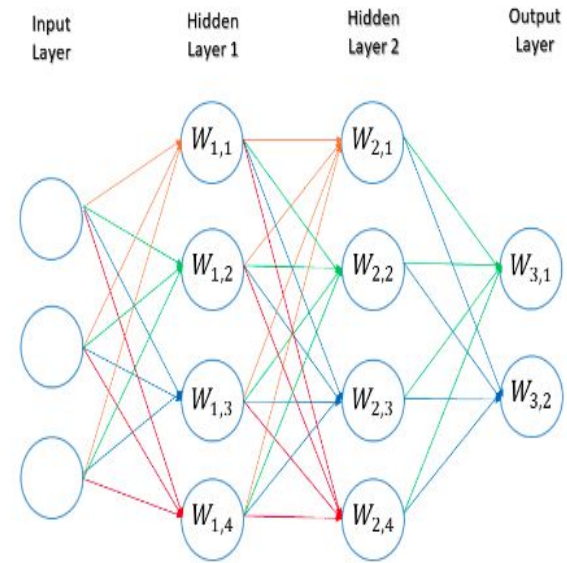
$x_i$  is the input pattern

$w_j$  is the weight vector

$f(w_j)$  is the activation function

# Feed-forward Neural Networks

- In a feed-forward network, signals can only move in one direction.
- These networks are considered non-recurrent network with inputs, outputs, and hidden layers.
- A layer of processing units receives input data and executes calculations there.
- Based on a weighted total of its inputs, each processing element performs its computation.
- The newly derived values are subsequently used as the new input values for the subsequent layer.
- This process continues until the output has been determined after going through all the layers.





# Backpropagation

- The typical algorithm to train this type of network is back-propagation.
- It is a technique for adjusting a neural network weights based on the error rate recorded in the previous epoch.
- Proper tuning of the weights ensures lower error rates, making the model reliable by increasing its generalization.
- The algorithm is used to effectively train a neural network through a method called chain rule.

# Activation Functions

- Activation functions are functions used in a neural network to compute the weighted sum of inputs and biases, which is in turn used to decide whether a neuron can be activated or not.
- The activation functions are also referred to as transfer functions in some literature.
- Activation function also helps to normalize the output of any input in the range between 1 to -1 or 0 to 1.
- The Activation Functions can be basically divided into 2 types-
  - Linear Activation Function
  - Non-linear Activation Functions

## Linear Activation Function

- Equation :  $f(x) = x$
- Range : (-infinity to infinity)

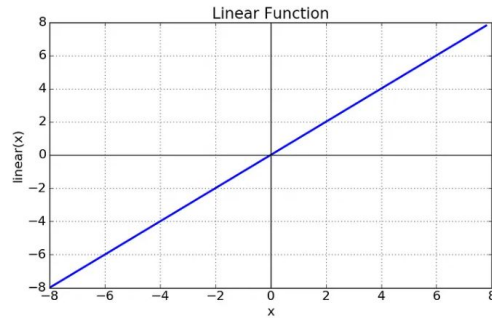


Fig: Linear Activation Function

## Non-linear Activation Function

- The Nonlinear Activation Functions are the most used activation functions.
- The Nonlinear Activation Functions are mainly divided on the basis of their range or curves.

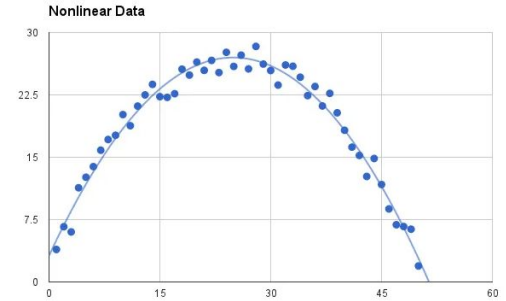


Fig: Non-linear Activation Function

## 1. Sigmoid or Logistic Activation Function

- The Sigmoid Function curve looks like a S-shape.
- The main reason why we use sigmoid function is because it exists between (0 to 1).
- Therefore, it is especially used for models where we have to predict the probability as an output.
- Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice.

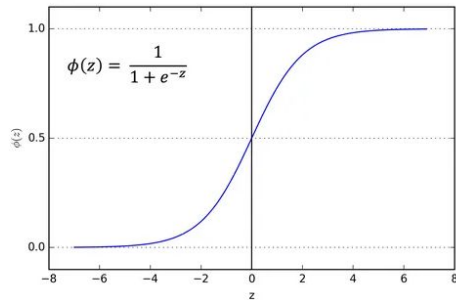


Fig: Sigmoid Function

## 2. Tanh or hyperbolic tangent Activation Function

- Tanh is also like logistic sigmoid but better. The range of the tanh function is from (-1 to 1).
- tanh is also sigmoidal (s - shaped).
- The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph.

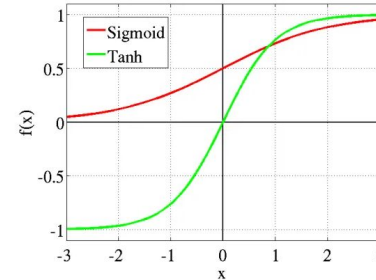
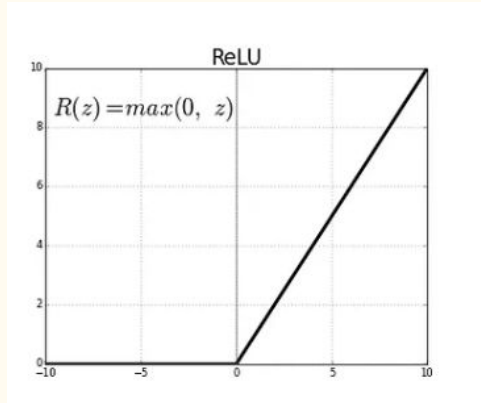


Fig: tanh v/s Logistic Sigmoid

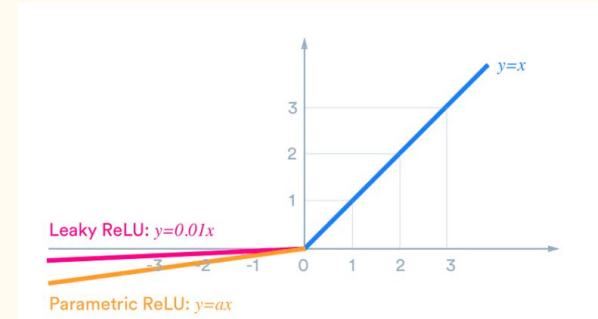
### 3. ReLU (Rectified Linear Unit) Activation Function

- The ReLU is the most used activation function in the world right now.
- Since, it is used in almost all the convolutional neural networks or deep learning.
- ReLU is half rectified (from bottom).
- $f(z)$  is zero when  $z$  is less than zero and  $f(z)$  is equal to  $z$  when  $z$  is above or equal to zero.
- Range:  $[0 \text{ to infinity})$



### 4. Leaky ReLU

- Leaky Rectified Linear Unit, or Leaky ReLU, is a type of activation function based on a ReLU, but it has a small slope for negative values instead of a flat slope.
- The value of the constant is determined before training, i.e. it is not learned during training.




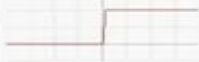







Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Fig: Activation Function Cheetsheet

# Perceptron

Perceptron is a linear classifier (binary). Also, it is used in supervised learning. It helps to classify the given input data.

The perceptron works on these simple steps

1. All the inputs  $x$  are multiplied with their weights  $w$ .
2. Add all the multiplied values and call them Weighted Sum.
3. Apply that weighted sum to the correct Activation Function.

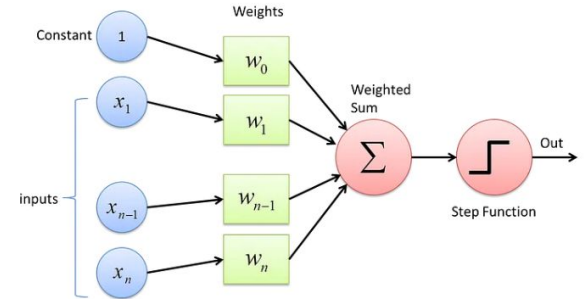


Fig : Perceptron

# Single Layer Perceptron Model

- A single-layer perceptron model is the simplest type of artificial neural network.
- It includes a feed-forward network that can analyze only linearly separable objects while being dependent on a threshold transfer function.
- The model returns only binary outcomes(target) i.e. 1, and 0.
- The algorithm in a single-layered perceptron model does not have any previous information initially.
- The weights are allocated inconsistently, so the algorithm simply adds up all the weighted inputs.
- If the value of the sum is more than the threshold or a predetermined value then the output is delivered as 1 and the single-layer perceptron is considered to be activated.



# MLP

- The multi-layer perceptron (MLP) is another artificial neural network process containing a number of layers.
- A multi-layer perceptron model uses the backpropagation algorithm.
- Though it has the same structure as that of a single-layer perceptron, it has one or more hidden layers.
- Every node in the multi-layer perceptron uses a sigmoid activation function.
- The sigmoid activation function takes real values as input and converts them to numbers between 0 and 1 using the sigmoid formula.
- In a single perceptron, distinctly linear problems can be solved but it is not well suitable for non-linear cases. To solve these complex problems, MLP can be considered.

# RBF Networks

- Radial basis function (RBF) networks have a fundamentally different architecture than most neural network architectures.
- Most neural network architecture consists of many layers and introduces nonlinearity by repetitively applying nonlinear activation functions.
- RBF network on the other hand only consists of an input layer, a single hidden layer, and an output layer.
- The input layer is not a computation layer, it just receives the input data and feeds it into the special hidden layer of the RBF network.
- The computation that is happened inside the hidden layer is very different from most neural networks, and this is where the power of the RBF network comes from. The output layer performs the prediction task such as classification or regression.

## Input Layer

- The input layer simply feeds the data to the hidden layers.
- The input neurons are fully connected to the hidden neurons and feed their input forward.

## Hidden Layer

- The hidden layer takes the input in which the pattern might not be linearly separable and transform it into a new space that is more linearly separable.
- This is based on Cover's theorem on the separability of patterns, which states that a pattern that is transformed into a higher-dimensional space with nonlinear transformation is more likely to be linearly separable, therefore the number of neurons in the hidden layer should be greater than the number of the input neuron.

- The computations in the hidden layers are based on comparisons with prototype vectors which is a vector from the training set.
- Each neuron in the hidden layer has a prototype vector and a bandwidth denoted by  $\mu$  and  $\sigma$  respectively.
- Each neuron computes the similarity between the input vector and its prototype vector. The computation in the hidden layer can be mathematically written as follow:

$$\phi_i = e^{\left(-\frac{\|\bar{X} - \bar{u}_i\|^2}{2 \cdot \sigma_i^2}\right)}$$

With:

$\bar{X}$  as the input vector

$\bar{u}_i$  as the  $i^{\text{th}}$  neuron's prototype vector

$\sigma$  as the  $i^{\text{th}}$  neuron's bandwidth

$\phi_i$  as the  $i^{\text{th}}$  neuron's output

- The parameters  $\bar{u}_i$  and  $\sigma$  are learned in an unsupervised way, for example using some clustering algorithm.

## Output Layer

- The output layer uses a linear activation function for both classification or regression tasks.
- The computations in the output layer are performed just like a standard artificial neural network which is a linear combination between the input vector and the weight vector. The computation in the output layer can be mathematically written as follow:

With:

$$y = \sum_i^n w_i \Phi_i$$

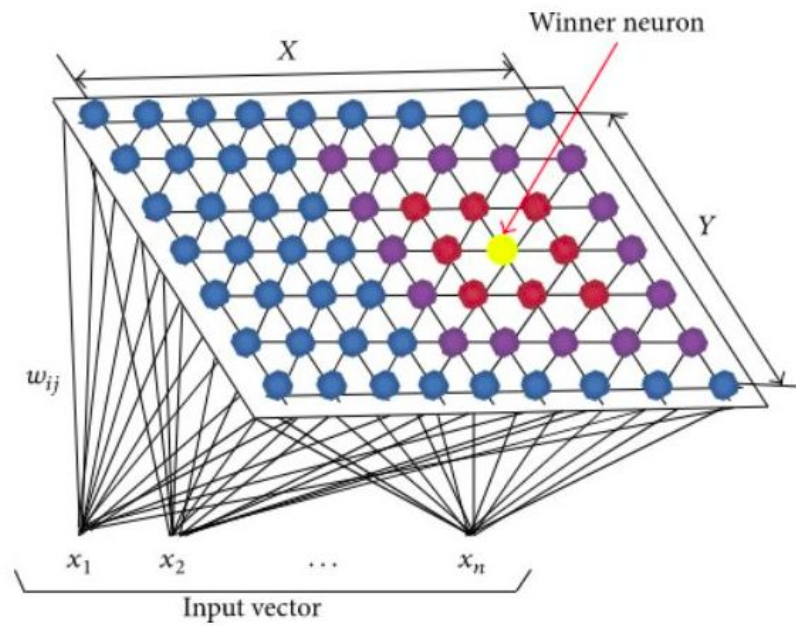
$w_i$  as the weight connection

$\phi_i$  as the  $i^{\text{th}}$  neuron's output from the hidden layer

$y$  as the prediction result

# Self Organizing Maps

- Self Organizing Map (or Kohonen Map or SOM) is a type of Artificial Neural Network which is also inspired by biological models of neural systems from the 1970s.
- SOM is used for clustering and mapping (or dimensionality reduction) techniques to map multidimensional data onto lower-dimensional which allows people to reduce complex problems for easy interpretation.
- SOM has two layers, one is the Input layer and the other one is the Output layer.



## How do SOM works?

- Self organizing maps have two layers, the first one is the input layer and the second one is the output layer or the feature map
- Unlike other ANN types, SOM doesn't have activation function in neurons, we directly pass weights to output layer without doing anything.
- Each neuron in a SOM is assigned a weight vector with the same dimensionality  $d$  as the input space. This type of unsupervised artificial neural network uses competitive learning to update its weights.
- Competitive learning is based on three processes : Competition ,Cooperation Adaptation

### 1)Competition :

- Each neuron in a SOM is assigned a weight vector with the same dimensionality as the input space.
- We compute distance between each neuron (neuron from the output layer) and the input data, and the neuron with the lowest distance will be the **winner** of the competition.
- The Euclidean metric is commonly used to measure distance.



2) Cooperation: The winning neuron determines the spatial location of a topological neighbourhood of excited neurons, thereby providing the basis for cooperation among neighbouring neurons.

3) Adaptation:

After choosing the winner neuron and its neighbors we compute neurons update. Those chosen neurons will be updated but not the same update, more the distance between neuron and the input data grow less we adjust it like shown in the image below :

$$w_j(n+1) = w_j(n) + \alpha h_{ij} \left( d(x, w_j(n)) \right)$$

where:

$x$  is the input vector

$w_{\{j\}}(n)$  is the vector representation of neuron  $j$  at iteration  $n$  ( $w_i(n)$  is the winning neuron)

$\alpha$  is the learning rate

$d$  is a distance function

The training process of SOMs follows:

- Firstly, randomly initialize all the weights.
- Select an input vector  $x = [x_1, x_2, x_3, \dots, x_n]$  from the training set.
- Compare  $x$  with the weights  $w_j$  by calculating Euclidean distance for each neuron  $j$ . The neuron having the least distance is declared as the winner. The winning neuron is known as Best Matching Unit(BMU)
- Update the neuron weights so that the winner becomes and resembles the input vector  $x$ .
- The weights of the neighbouring neuron are adjusted to make them more like the input vector. The closer a node is to the BMU, the more its weights get altered. The parameters adjusted are learning rate and neighbourhood function. (more below on neighbourhood function)
- Repeat from step 2 until the map has converged for the given iterations or there are no changes observed in the weights.

# Hopfield Networks

# Boltzmann Machines

# Probabilistic Neural Networks