6. Applications of Artificial Intelligence

(Artificial Intelligence and Neural Networks)

Artificial Neural Network (ANN)

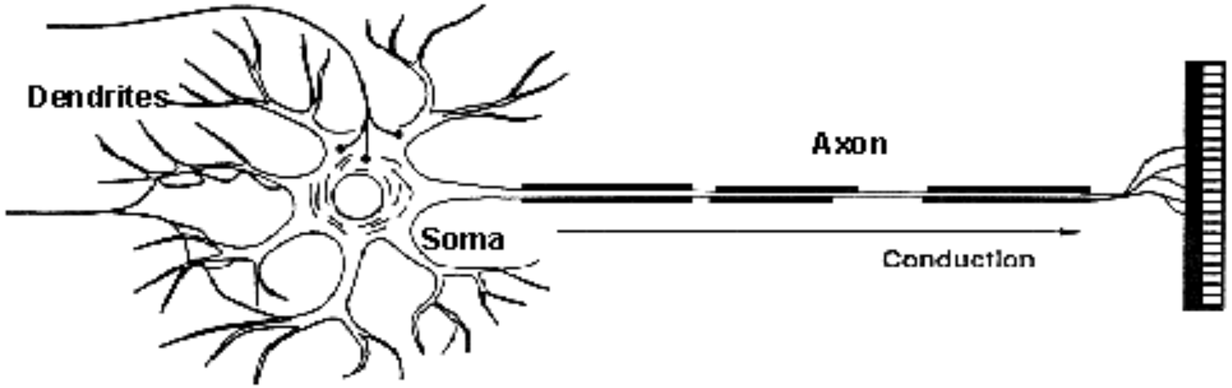
Dr. Udaya Raj Dhungana
Assist. Professor
Pokhara University, Nepal
Guest Faculty
Hochschule Darmstadt University of Applied Sciences, Germany
E-mail: udaya@pu.edu.np and udayas.epost@gmail.com

Overview

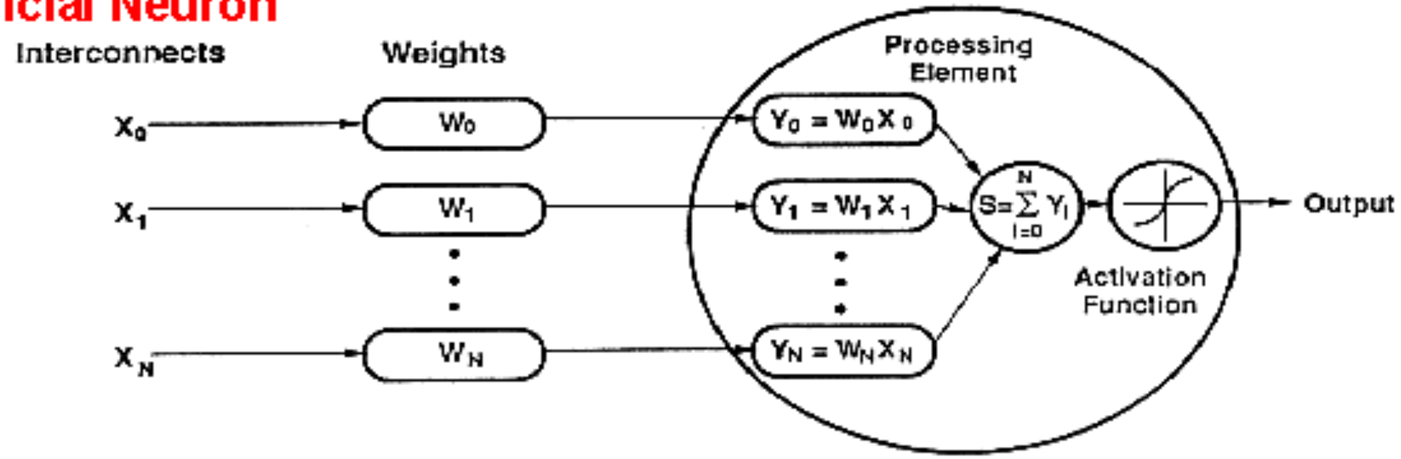
- Network Structure
- Adaline, Madaline, Perceptron
- Multi-layer Perceptron
- Radial basis function
- Hopfield Network
- Kohonen Network
- Elastic net model
- Back-propagation
- Boltzmann Machines

Biological Inspiration

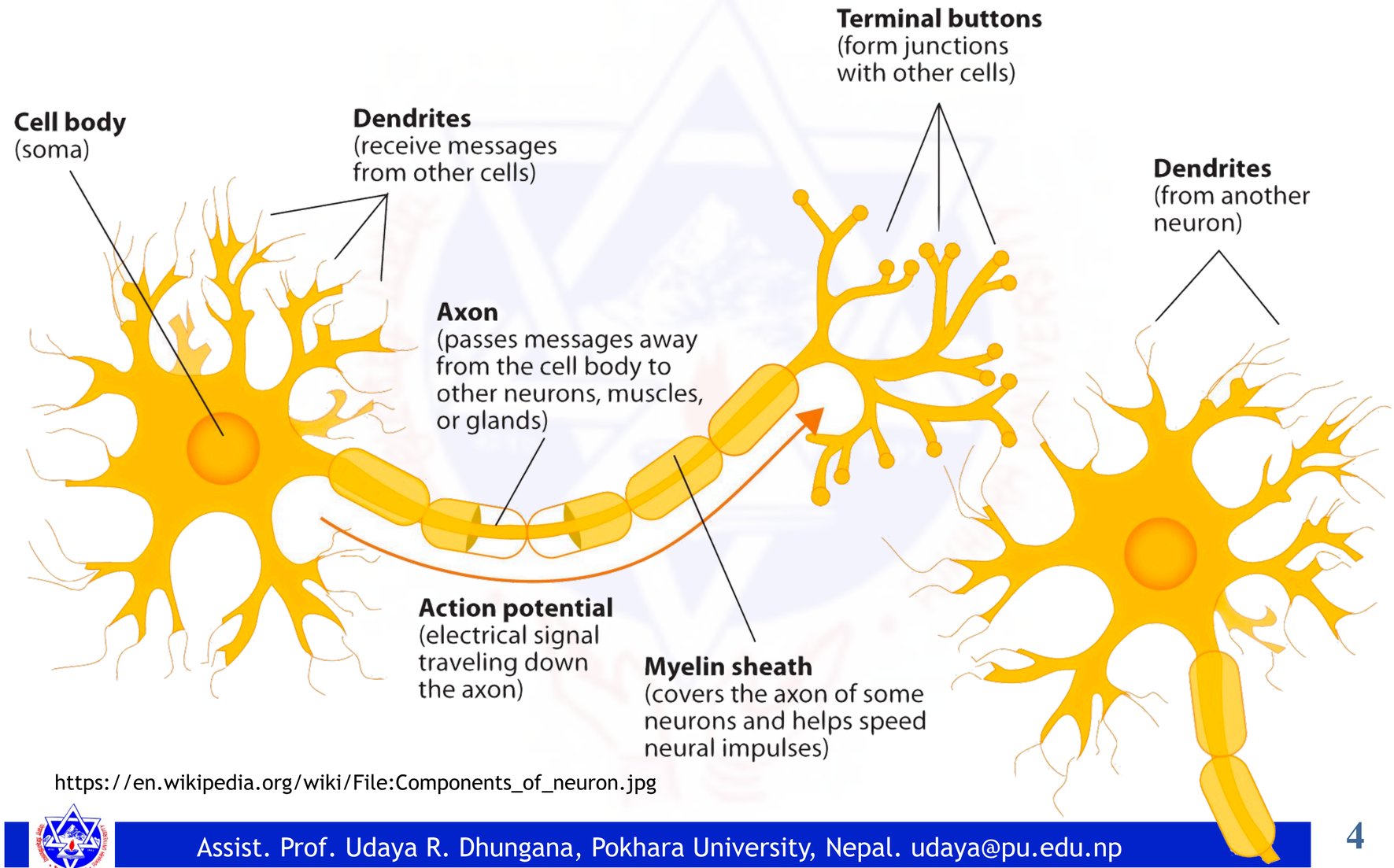
Biological Neuron



Artificial Neuron



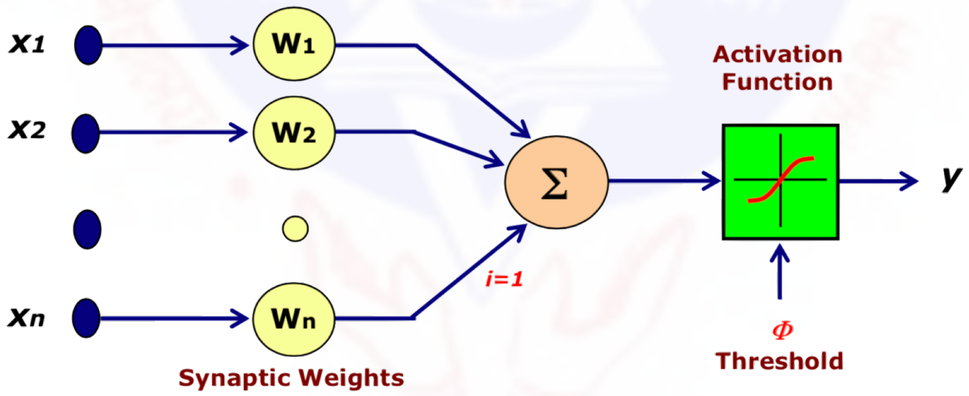
Biological Inspiration



https://en.wikipedia.org/wiki/File:Components_of_neuron.jpg

Biological Inspiration

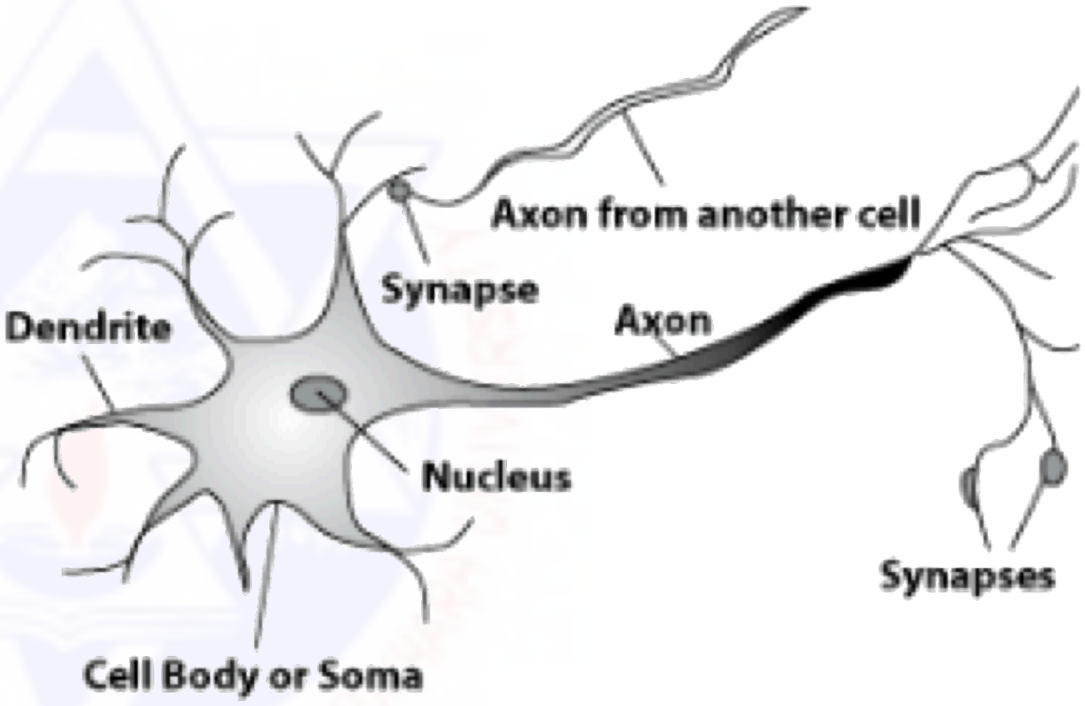
- Humans perform complex tasks: like vision, motor control, or language understanding very well.
- The brain is a highly complex, non-linear, and parallel computing part, composed of some 10^{11} neurons that are densely connected (~10^4 connection per neuron).
- One way to build intelligent machines that can learn is to try to imitate the organizational principles of human brain.



Artificial Neuron

Biological Inspiration

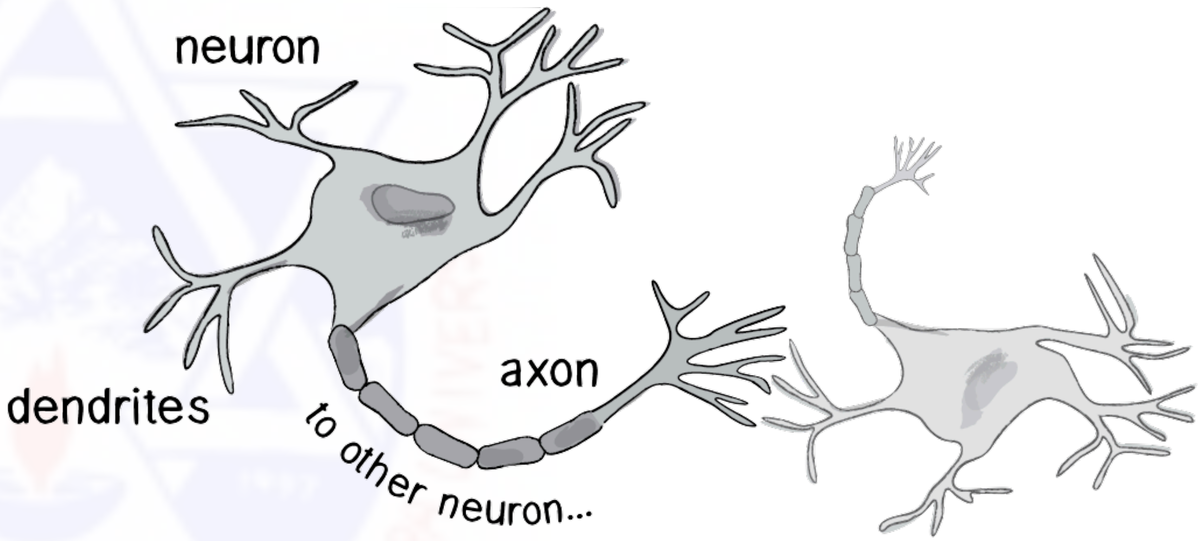
- Dendrites: nerve fibres carrying electrical signals to the cell.
- Cell body: computes a non-linear function of its inputs.
- Axon: single long fiber that carries the electrical signal from the cell body to other neurons
- Synapse: the point of contact between the axon of one cell and the dendrite of another, regulating a chemical connection whose strength affects the input to the cell.



Dendrites: Input
Cell body: Processor
Synaptic: Weight/Link
Axon: Output

Biological Inspiration

- Neurons: simple processing units in the human brain
- **How neurons works:**
  - Inputs are received on dendrites
  - If the input levels are over a threshold, the neuron fires,
  - When a neuron is fired, it passes a signal through the axon to the synapse which then connects to another neuron.



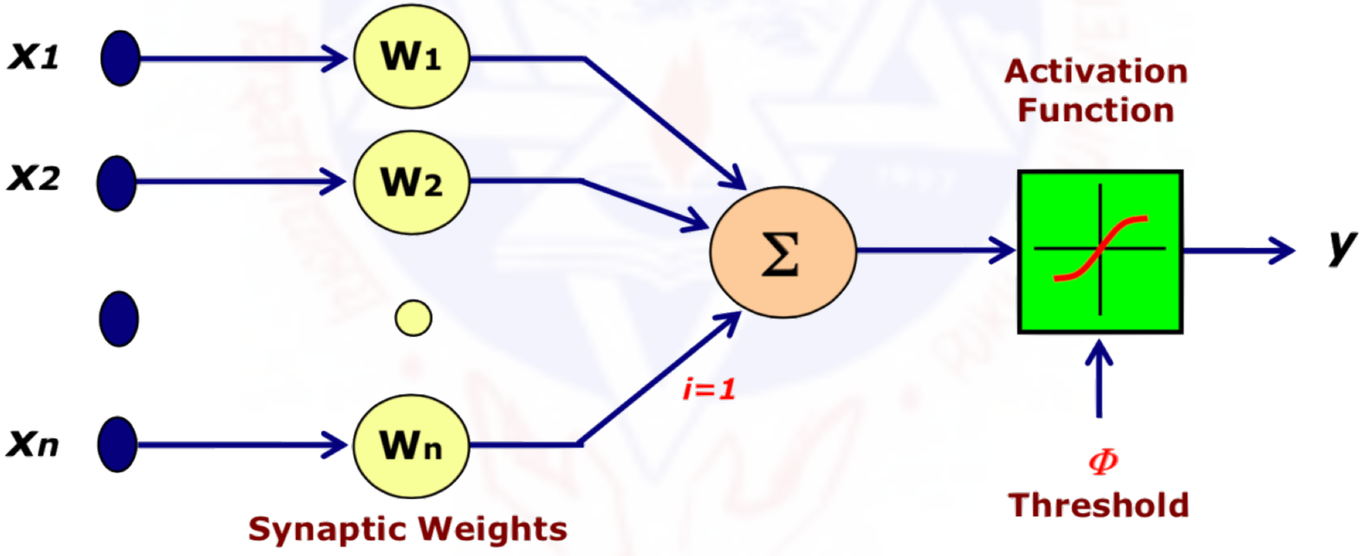
Dendrites: Input
Cell body: Processor
Synaptic: Weight/Link
Axon: Output

Artificial Neural Network- ANN

- Artificial Neural Network (ANN) is analogous to a biological neural network.
- Artificial Neuron are computational and reasoning model based on biological neurons.
- ANN like biological neurons are
  - Massively parallel
  - distributed system
  - made up of simple processing units (artificial neurons)
- ANN is a non-linear model that is widely used in Machine Learning
- ANN is simply called as Neural Network

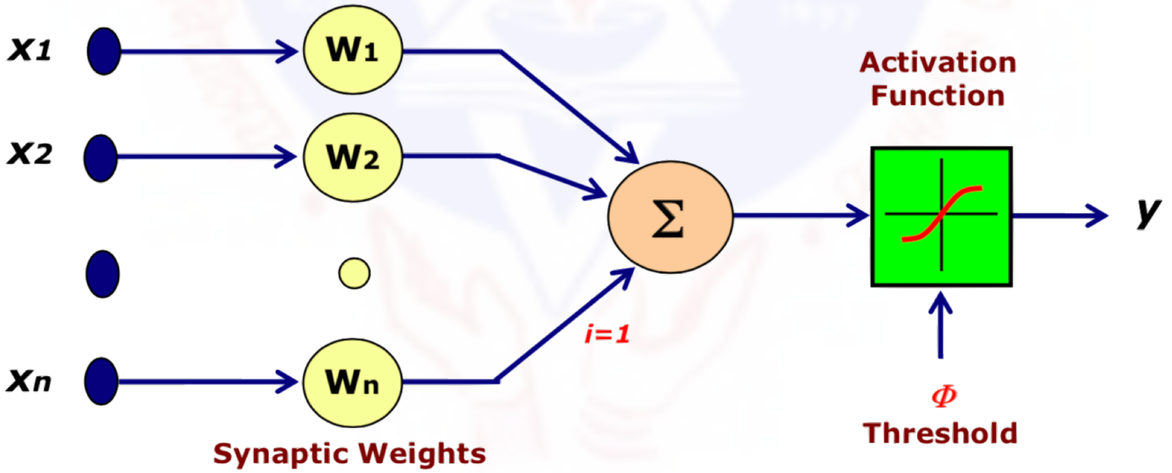
Artificial Neural Network- ANN

- ANN consists of three basic components:
  1. Weight
  2. Threshold (or bias) and
  3. Activation function



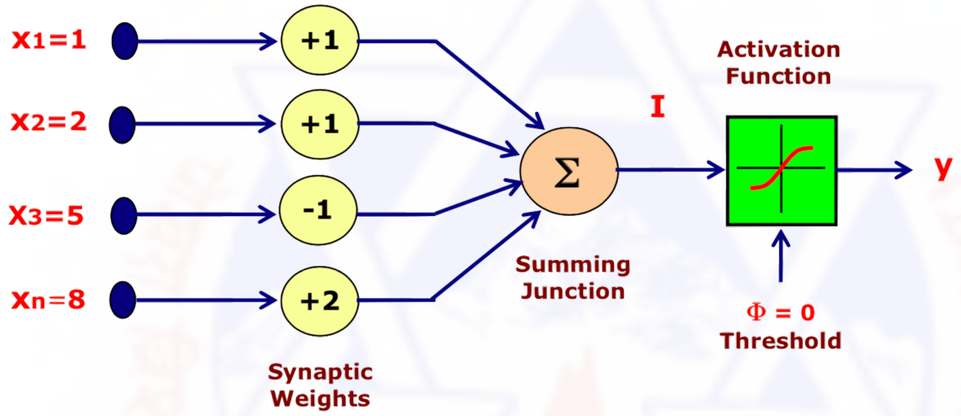
Artificial Neuron- A Simple Computing Element

- An Artificial Neuron:
  - Computes the weighted sum of its input signals (net input)
  - Compares the sum with threshold value passing the sum through an activation function
  - (In case of binary threshold activation function) If it is less than the threshold, the neuron output is 0.
  - If it is greater than or equal to the threshold, the neuron becomes activated (We say the neuron is fired) and its output attains a value 1.



Artificial Neuron- A Simple Computing Element

Example: The figure shows a Neuron with four inputs with weights.



Binary Activation Function:

$$y = f(I) = \begin{cases} 1, & \text{if } I > \theta \\ 0, & \text{if } I < \theta \end{cases}$$

Where y = output of neuron
I = weighted sum
θ = threshold value

The output I prior to activation function is computed as:

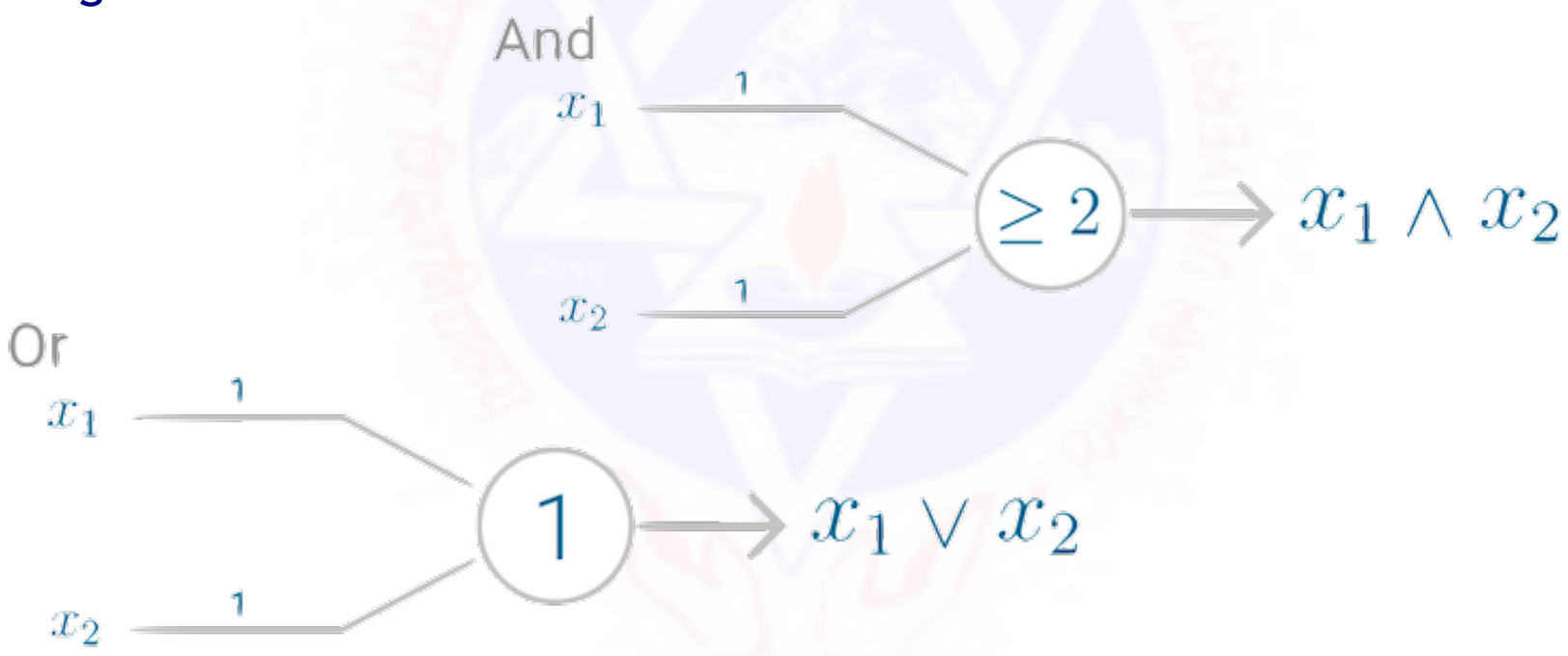
$$I = X^T . W = \begin{bmatrix} 1 & 2 & 5 & 8 \end{bmatrix} \bullet \begin{pmatrix} +1 \\ +1 \\ -1 \\ +2 \end{pmatrix} = 14$$

$$= (1 \times 1) + (2 \times 1) + (5 \times -1) + (8 \times 2) = 14$$

With a binary activation function, the output, **y = f(I) = f (14) = 1**

Artificial Neuron- A Simple Computing Element

Example: The first application of the neuron replicated a logic gate, where you have one or two binary inputs, and a boolean function that only gets activated given the right inputs and weights.



Artificial Neural Networks (ANN)

- An Artificial Neural Network is specified by:

  1. **Neuron Model:** the information processing unit of the NN,
  2. **An Architecture:** a set of neurons and links connecting neurons. Each link has a weight
  3. **A Learning Algorithm:** used for training the NN by modifying the weights in order to model a particular learning task correctly on the training examples.

- The aim is to obtain a NN that is trained and well generalized.
- It should behaves correctly on new instances of the learning task.

Artificial Neural Networks (ANN)

- The neuron is the basic information processing unit of a NN. It consists of:

  1. A set of links, describing the neuron inputs, with weights $W_1, W_2, \ldots, W_m$

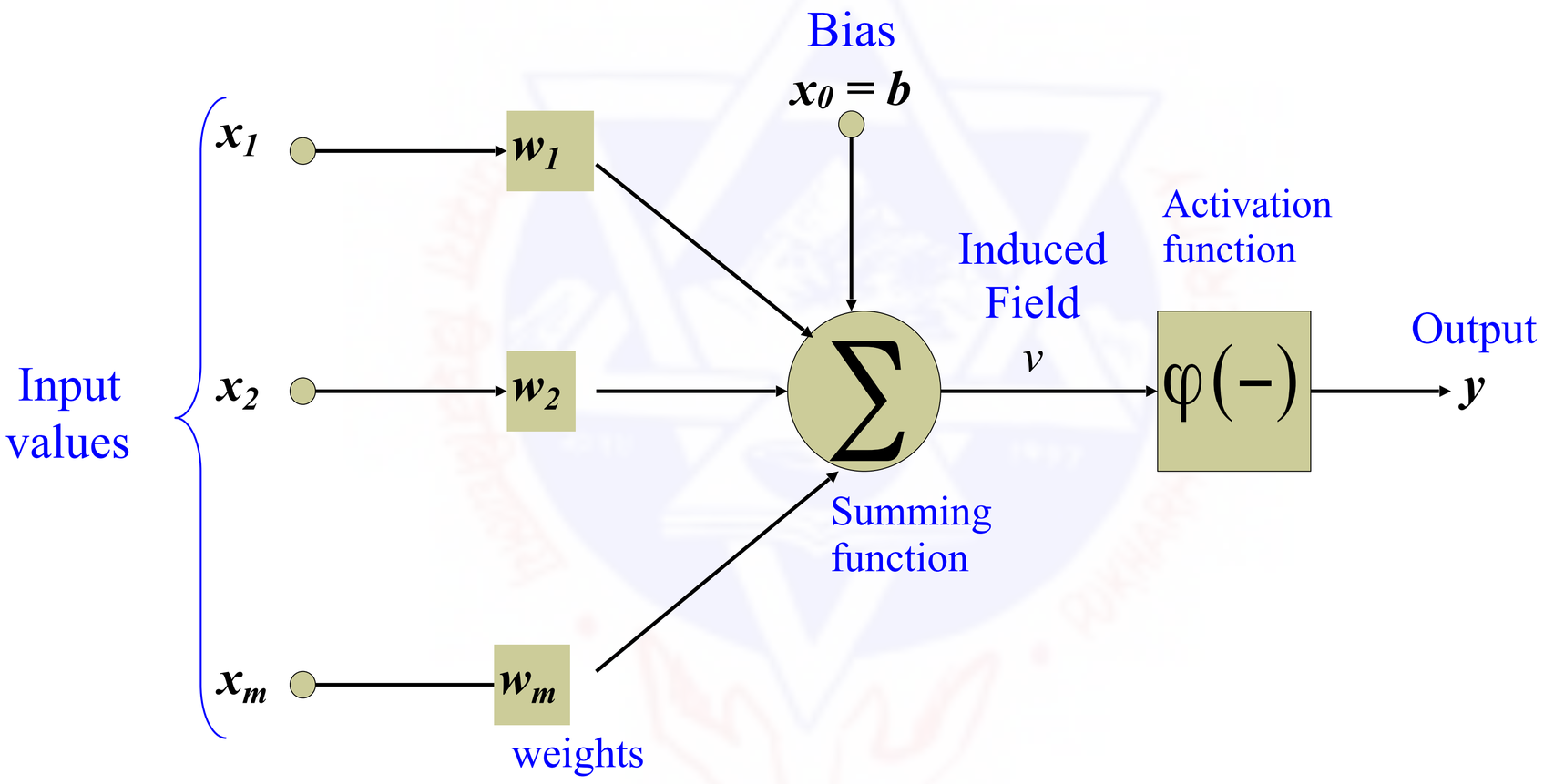
  2. An adder function (linear combiner) for computing the weighted sum of the inputs: (real numbers)

$$u = \sum_{j=1}^{m} w_j x_j$$

  3. Activation function φ for limiting the amplitude of the neuron output. Here 'b' denotes bias.

$$y = \varphi(u + b)$$

Artificial Neural Networks (ANN)



Some Terminologies

- Bias
- Activation functions
- Weights
- Threshold
- Learning rat
- Momentum factor
- Vigilance parameter

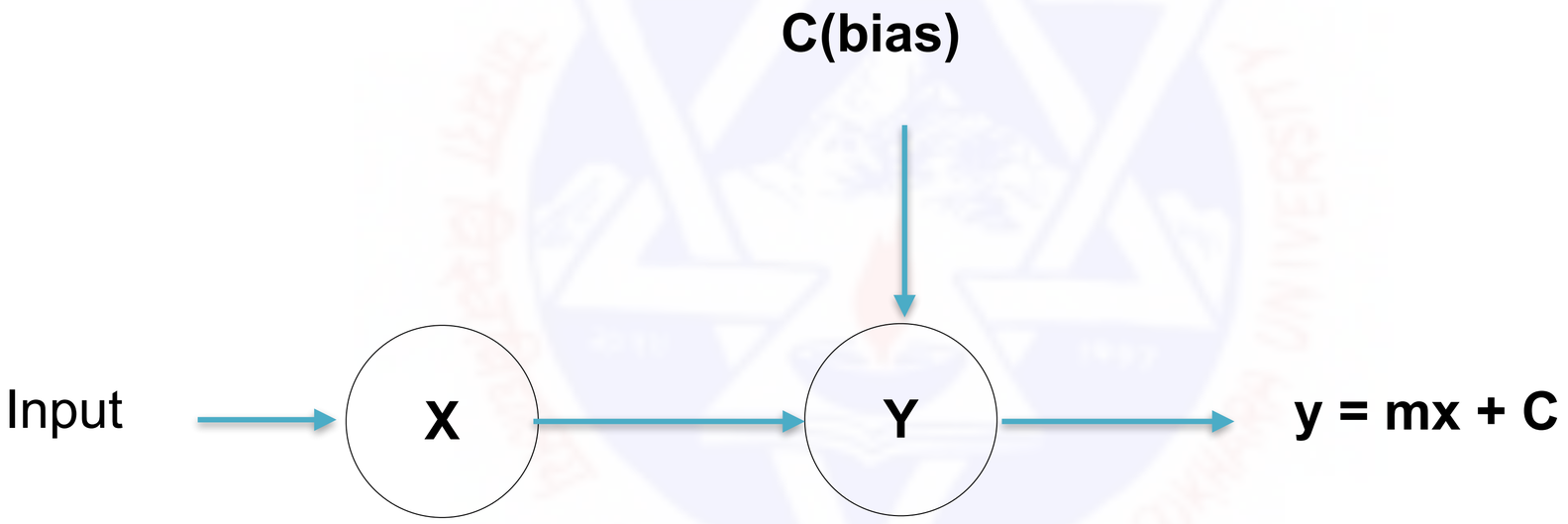
Bias

- An artificial neuron:
  - computes the weighted sum of its input (called its net input)
  - adds its bias
  - passes this value through an activation function
- We say that the neuron "fires" (i.e. becomes active) if its output is above zero.
- 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.

$$y_i = \varphi(n_i) = \varphi\left(\sum_{j=1}^{n} w_{ij}x_j + b_i\right)$$

Why Bias is need?

- The relationship between input and output given by the equation of straight line y=mx+c



Bias

- Bias is like another weight. Its included by adding a component $x_0=1$ to the input vector X.
- $X=(1,X_1,X_2...X_i,...X_n)$
- Bias is of two types
  - Positive bias: increase the net input
  - Negative bias: decrease the net input

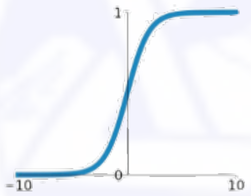
$$y_i = \varphi(n_i) = \varphi\left(\sum_{j=1}^{n} w_{ij}x_j + b_i\right)$$

Activation Functions

- Also called the squashing function as it limits the amplitude of the output of the neuron.
- The choice of activation function φ determines the neuron model.

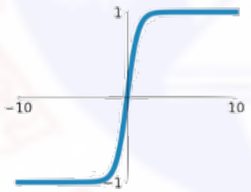
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



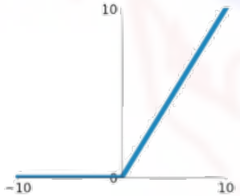
tanh

$\tanh(x)$



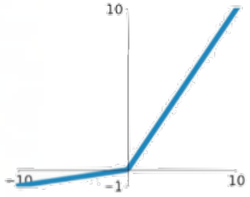
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

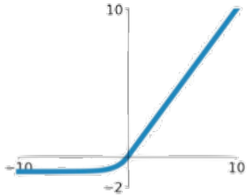


Maxout

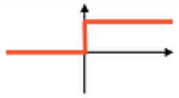
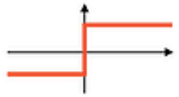
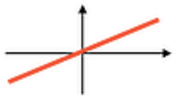
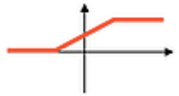
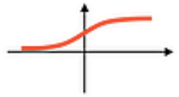
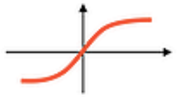
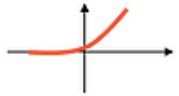
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Activation Functions

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^{z})$	Multi-layer Neural Networks	

Activation Functions

- Used to calculate the output response of a neuron.
- Sum of the weighted input signal is applied with an activation to obtain the response.
- Activation functions can be linear or non linear
- we use the activation functions to propagate the output of a neuron forward. This output is received by the neurons of the next layer to which this neuron is connected (up to the output layer included).

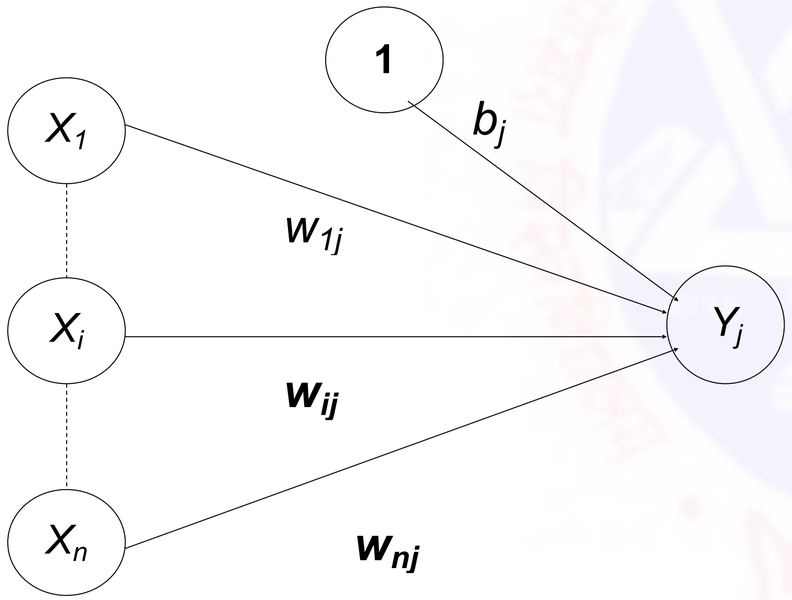
Weights

- Each neuron is connected to every other neuron by means of directed links
- Links are associated with weights
- Weights contain information about the input signal and is represented as a matrix
- Weight matrix also called <u>connection matrix</u>

$$W = \begin{bmatrix} w_1^T \\ w_2^T \\ w_3^T \\ . \\ . \\ . \\ . \\ . \\ w_n^T \end{bmatrix} = \begin{bmatrix} w_{11} w_{12} w_{13} \cdots w_{1m} \\ w_{21} w_{22} w_{23} \cdots w_{2m} \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ w_{n1} w_{n2} w_{n3} \cdots w_{nm} \end{bmatrix}$$

Weights

- w_{ij} is the weight from processing element "*i*" (source node) to processing element "*j*" (destination node)



$$y_{inj} = \sum_{i=0}^{n} x_i w_{ij}$$

$$= x_0 w_{0j} + x_1 w_{1j} + x_2 w_{2j} + \dots + x_n w_{nj}$$

$$= w_{0j} + \sum_{i=1}^{n} x_i w_{ij}$$

$$y_{inj} = b_j + \sum_{i=1}^{n} x_i w_{ij}$$

Threshold

- Set value based upon which the final output of the network may be calculated
- Used in activation function
- The activation function using threshold can be defined as

$$f(x) = \begin{cases} 1 & if\ x \geq \theta \\ -1 & if\ x < \theta \end{cases}$$

Learning rate

- Denoted by α.
- Used to control the amount of weight adjustment at each step of training
- Learning rate ranging from 0 to 1 determines the rate of learning in each time step

- Momentum factor:
  - used for convergence when momentum factor is added to weight updation process.
- Vigilance parameter:
  - Denoted by ρ
  - Used to control the degree of similarity required for patterns to be assigned to the same cluster

Learning Methods

- A neural network is made up of neurons connected to each other; at the same time, each connection of our neural network is associated with a weight that dictates the importance of this relationship in the neuron when multiplied by the input value.

- Each neuron has an **activation function** that defines the output of the neuron. The activation function is used to introduce non-linearity in the modeling capabilities of the network.

- Training our neural network, that is, learning the values of our parameters (weights *wij* and *bj* biases) is the most genuine part of Deep Learning and we can see this learning process in a neural network as an iterative process of "going and return" by the layers of neurons. The "going" is a forward propagation of the information and the "return" is a back propagation of the information.

Learning Methods

- The first phase **forward propagation** occurs when the network is exposed to the training data and these cross the entire neural network for their predictions (labels) to be calculated. That is, passing the input data through the network in such a way that all the neurons apply their transformation to the information they receive from the neurons of the previous layer and sending it to the neurons of the next layer.

- When the data has crossed all the layers, and all its neurons have made their calculations, the final layer will be reached with a result of label prediction for those input examples.

Learning Methods

- Next, we will use a **loss function** to estimate the loss (or error) and to compare and measure how good/bad our prediction result was in relation to the correct result (remember that we are in a supervised learning environment and we have the label that tells us the expected value).

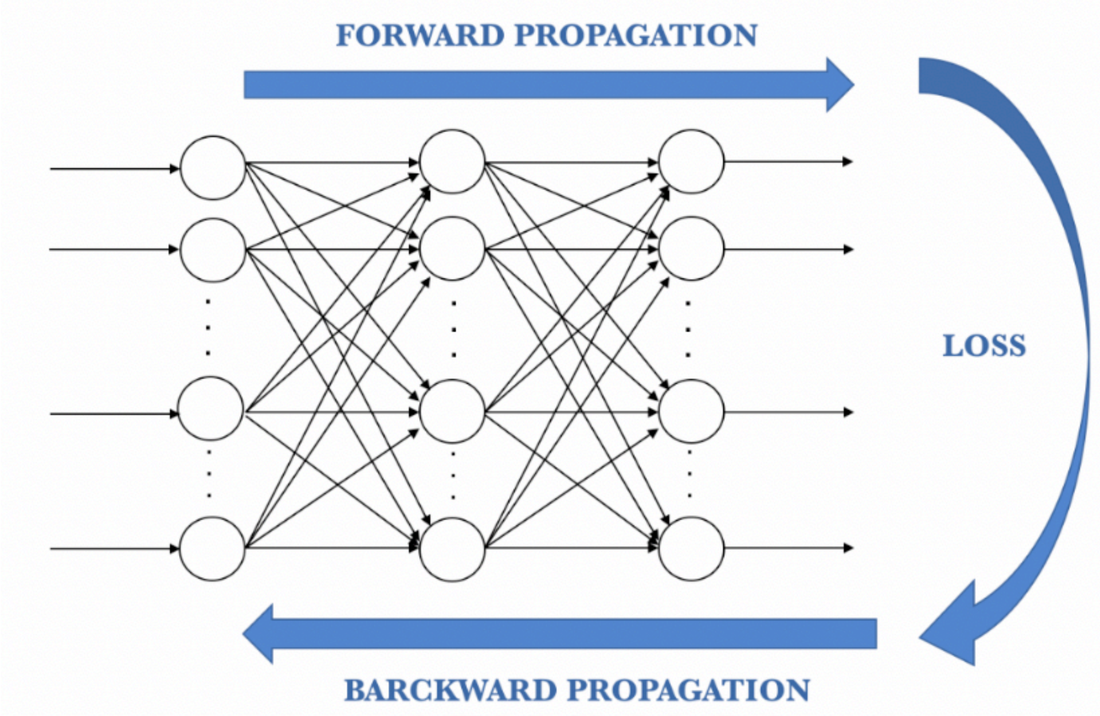
- Ideally, we want our cost to be zero, that is, without divergence between estimated and expected value. Therefore, as the model is being trained, the weights of the interconnections of the neurons will gradually be adjusted until good predictions are obtained.

Learning Methods

- Once the loss has been calculated, this information is propagated backwards. Hence, its name: **back propagation**.
- Starting from the output layer, that loss information propagates to all the neurons in the hidden layer that contribute directly to the output.
- However, the neurons of the hidden layer only receive a fraction of the total signal of the loss, based on the relative contribution that each neuron has contributed to the original output.
- This process is repeated, layer by layer, until all the neurons in the network have received a loss signal that describes their relative contribution to the total loss.

Learning Methods

Visually, we can summarize what we have explained with this visual scheme of the stages:



Learning Methods

- we have spread this information back, we can adjust the weights of connections between neurons.

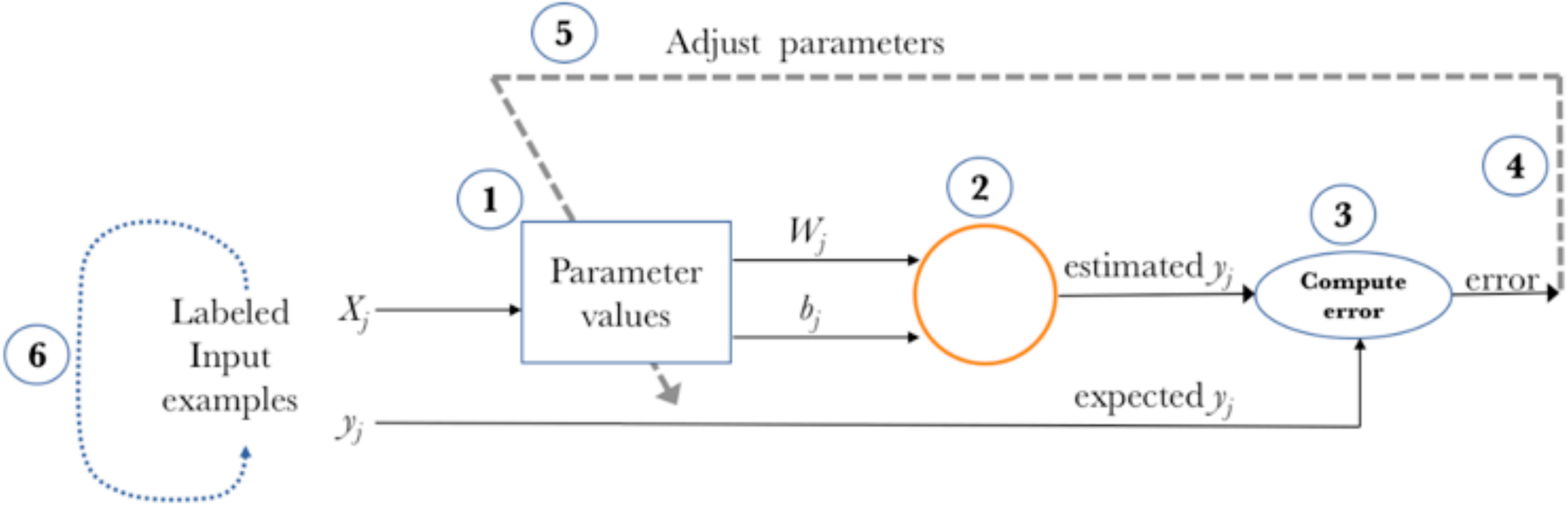
- What we are doing is making the loss as close as possible to zero the next time we go back to using the network for a prediction. For this, we will use a technique called **gradient descent**. This technique changes the weights in small increments with the help of the calculation of the derivative (or gradient) of the loss function, which allows us to see in which direction "to descend" towards the global minimum; this is done in general in batches of data in the successive iterations (epochs) of all the dataset that we pass to the network in each iteration.

Learning Algorithm

To recap, the learning algorithm consists of:

1. Start with values (often random) for the network parameters (*wij* weights and *bj* biases).
2. Take a set of examples of input data and pass them through the network to obtain their prediction.
3. Compare these predictions obtained with the values of expected labels and calculate the loss with them.
4. Perform the **back propagation** in order to propagate this loss to each and every one of the parameters that make up the model of the neural network.
5. Use this propagated information to update the parameters of the neural network with the gradient descent in a way that the total loss is reduced and a better model is obtained.
6. Continue iterating in the previous steps until we consider that we have a good model.

Learning Algorithm



Learning

- It's a process by which a NN adapts itself to a stimulus by making proper parameter adjustments, resulting in the production of desired response
- Two kinds of learning
  - Parameter learning:- connection weights are updated
  - Structure Learning:- change in network structure

NN Learning Rules

- During ANN learning, to change the input/output behavior, we need to adjust the weights.
- Hence, a method is required with the help of which the weights can be modified.
- These methods are called Learning rules, which are simply algorithms or equations.
- Following are some learning rules for the neural network –
    - Hebbian Learning Rule
    - Perceptron Learning Rule
    - Delta Learning Rule
    - Competitive Learning Rule
    - Outstar Learning Rule

.

Some Learning Algorithm

- **Supervised:**
  - Adaline, Madaline
  - Perceptron
  - Back Propagation
  - multilayer perceptrons
  - Radial Basis Function Networks
- **Unsupervised**
  - Competitive Learning
  - Kohenen self organizing map
  - Learning vector quantization
  - Hebbian learning

Training

- The process of modifying the weights in the connections between network layers with the objective of achieving the expected output is called training a network.
- This is achieved through
  - Supervised learning
  - Unsupervised learning
  - Reinforcement learning

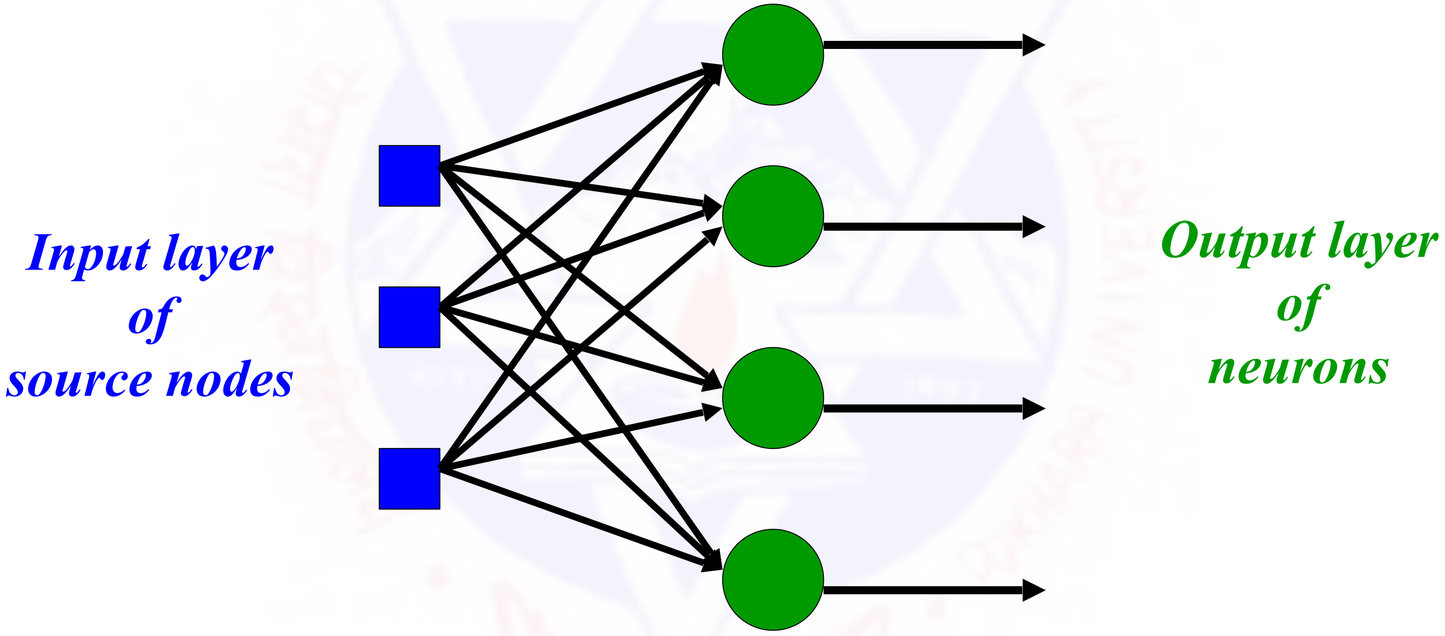
Neural Network Architecture

- Three different classes of network architectures
  - single-layer feed-forward
  - multi-layer feed-forward
  - recurrent

Single-layer Feed-forward NN

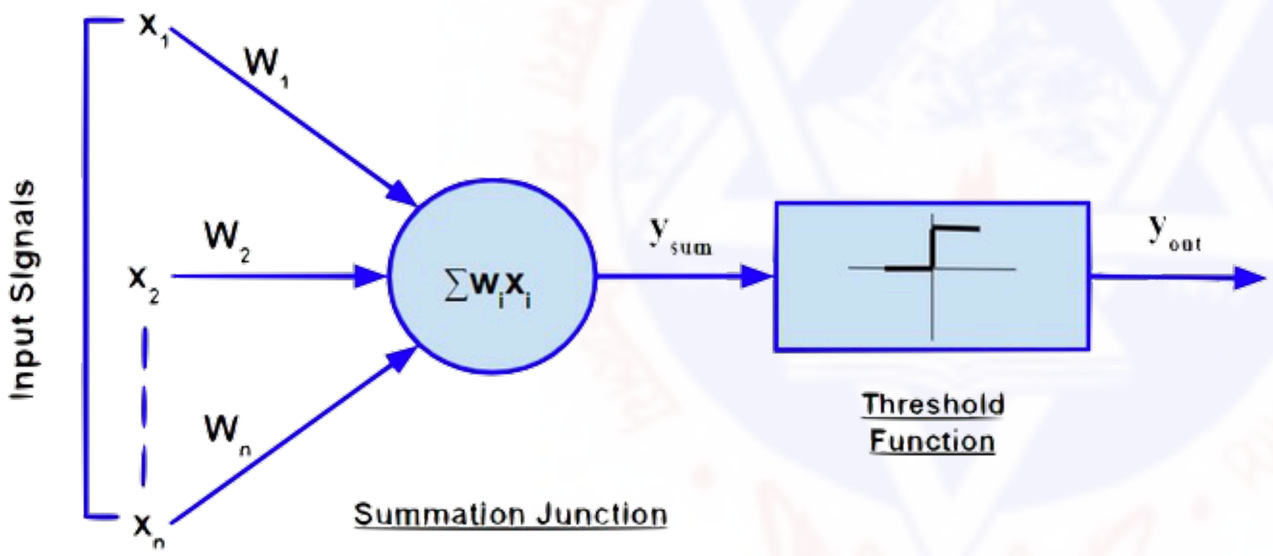
- A single-layer feed forward network consists of a single layer of output nodes
- The inputs are fed directly to the outputs via a series of weights.
- The sum of the products of the weights and the inputs is calculated in each node
- If the value is above some threshold (typically 0) the neuron fires and takes the activated value (typically 1); otherwise it takes the deactivated value (typically -1).
- They have no hidden layers
- Example:
  - McCulloch and Pitts Neuron
  - single-layer perceptron

Single-layer Feed-forward NN

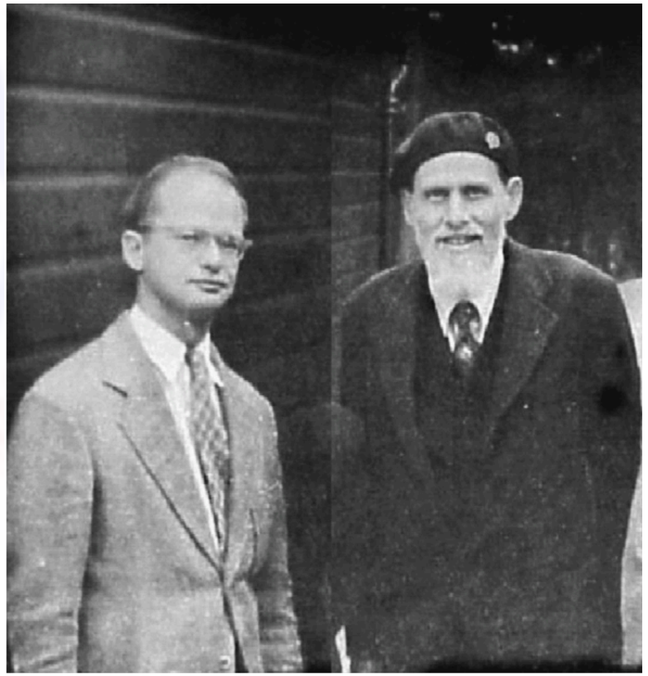


McCulloch-Pitts Neuron

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



McCulloch-Pitts Model



McCulloch (right) and Pitts (left) in 1949

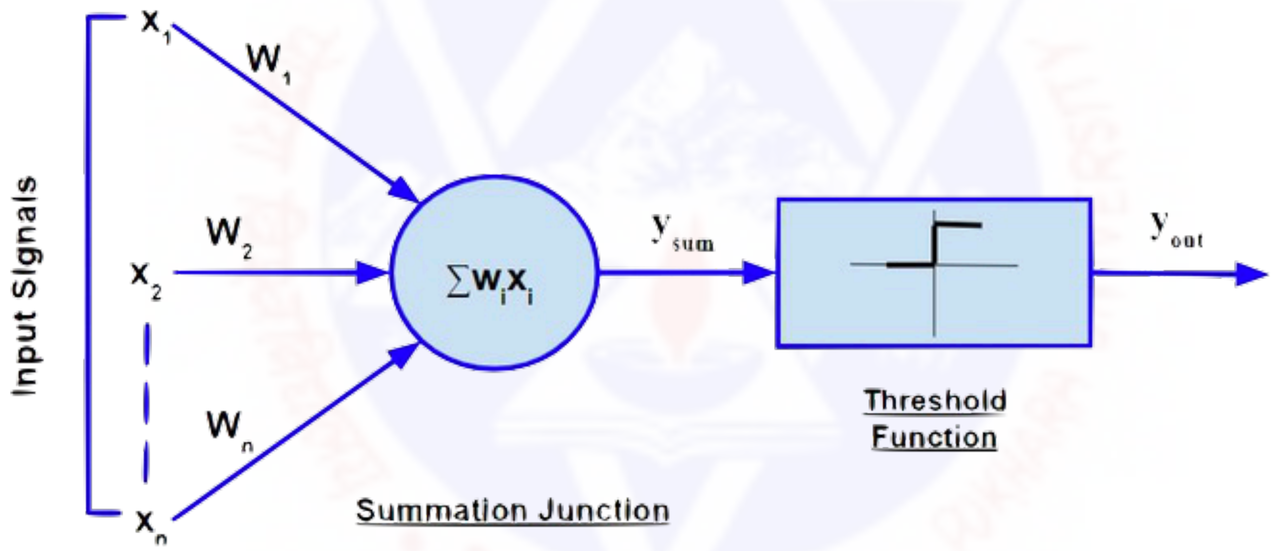
Image Source: www.semanticscholar.org

McCulloch-Pitts Neuron

- This neuron has only two types of inputs —
  - **Excitatory: The excitatory inputs have weights of positive magnitude**
  - **Inhibitory.** the inhibitory weights have weights of negative magnitude.
- The inputs of the McCulloch-Pitts neuron could be either 0 or 1.
- It has a threshold function as an activation function.
- So, the output signal *yout* is 1 if the input *ysum* is greater than or equal to a given threshold value, else 0.

McCulloch-Pitts Neuron

- The diagrammatic representation of the model is as follows:



McCulloch-Pitts Model

McCulloch-Pitts Neuron

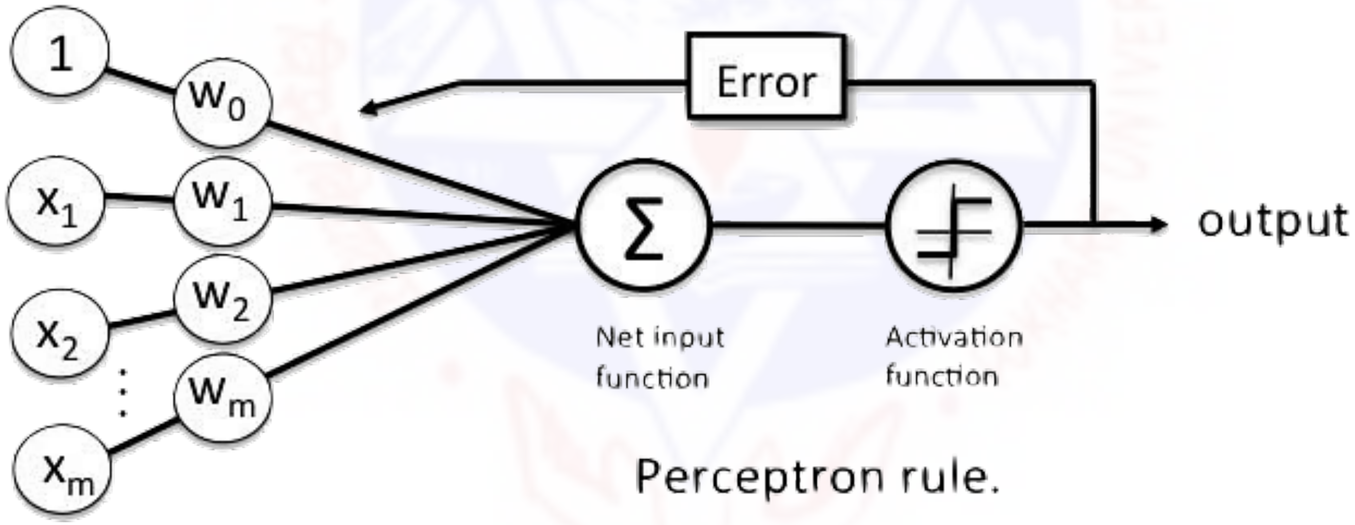
- This approach was powerful in so far it was able to emulate a wide variety of boolean functions by combining simple binary computational units.
- However, the architecture lacked several characteristics of biological networks:
  - complex connectivity patterns,
  - processing of continuous values (rather than just binary), and
  - critically, a learning procedure.
- To overcome these limitations, Frank Rosenblatt introduced the so-called "**perceptron**" in 1958.

Perceptron

- Perceptron is based on McCulloch-Pitts neuron model
- The perceptron algorithm was invented in 1958 at the Cornell Aeronautical Laboratory by Frank Rosenblatt
- Its first implementation was in software for the IBM 704- for image recognition
- A Perceptron is an algorithm for supervised learning of binary classifiers.
- Perceptron consists of a single neuron
- Perceptron can classify data (patterns) into two classes if the data are linearly separable

Perceptron

- Perceptron consists of a single neuron with an arbitrary number of inputs along with adjustable weights, but the output of the neuron is 1 or 0 depending upon the threshold. It also consists of a bias whose weight is always 1.



Perceptron rule.

Perceptron

- A perceptron can be described by:
  - a linear function that aggregates the input signals
  - a threshold function that determines if the response neuron fires or not
  - a learning procedure to adjust connection weights
- A perceptron uses error-corrective reinforcement learning.
- A perceptron trained by error-correction achieves its goal is by iteratively adjusting its weights values until it finds a set of weights that allow for the separation of classes in the set.

Perceptron

- **The Linear Aggregation Function:**
  - The linear function that aggregates the input signals for a single neuron or processing unit is defined as:

$$z = b + \sum_{i=1}^{n} w_i x_i = w_0 x_0 + w_1 x_1 + \ldots + w_n x_n$$

  - The z output is used as input for the threshold function $f(z)$.
  - The b constant added at the beginning, the bias term, is a way to simplify learning a good threshold value for the network.

Perceptron

- **The Threshold Function:**
  - The threshold function for the perceptron is defined as:

$$\hat{y} = f(z) = \begin{cases} +1, & \text{if } z > 0 \\ -1, & \text{Otherwise} \end{cases}$$

  - Where, z is output of the linear aggregation function.

Perceptron

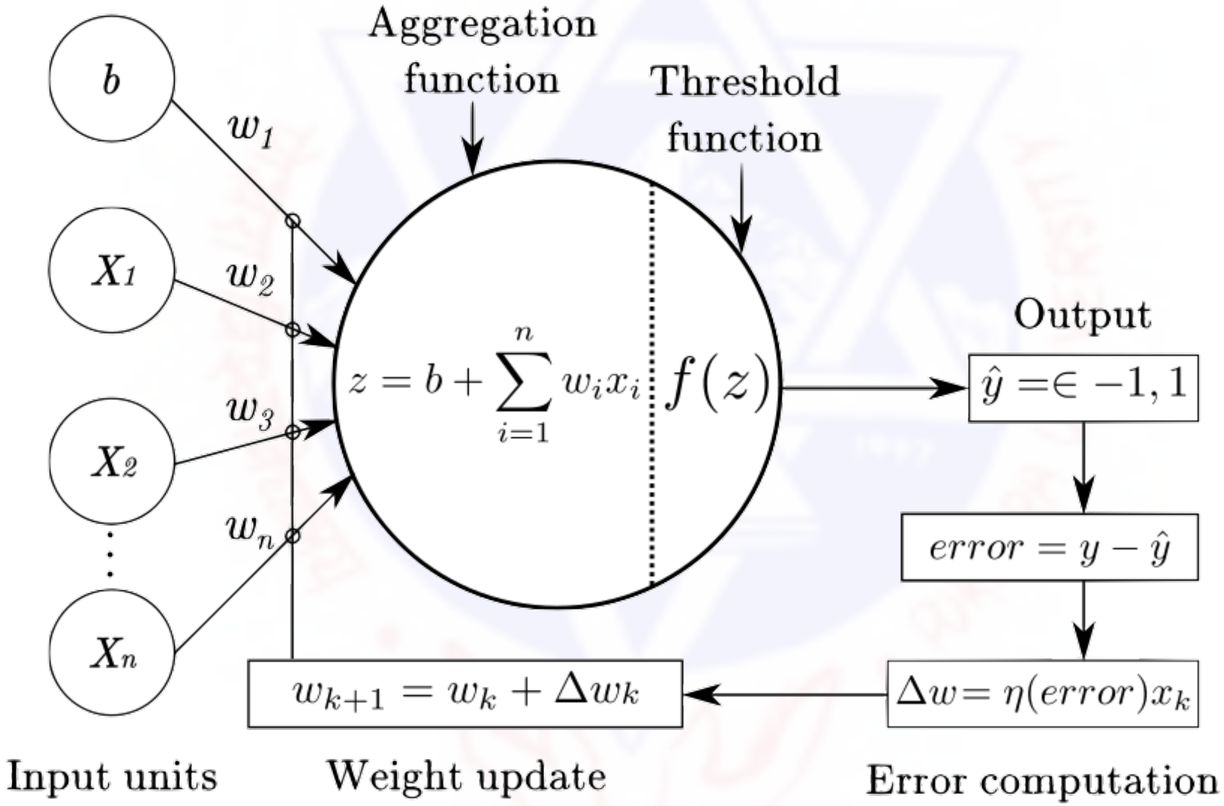
- **The Learning Algorithm:**
  - Rosenblatt (1961) defined as an error-corrective reinforcement learning procedure
  - The error-corrective learning rule is as follow:
    - compute the mismatch between the obtained value and the expected value for the training example
    - if the obtained and expected values match, do nothing
    - if the obtained and expected values do not match, compute the difference or delta between those values then, use a portion of that value to update the weights of the network

Perceptron

- **The Learning Algorithm:**
  - Compute error as: $error = y - \hat{y}$
  - Computer update in weight (Δw) as:
    $\Delta w = \eta(error)x_k = \eta(y - \hat{y})x_k$
  - Compute new weight as: $w_{k+1} = w_k + \Delta w$
- Where, w_k is the weight for case k
  - η is the learning rate whose values are between (0, 1)
  - y is the actual output value given in trading example
  - $\hat{y}$ is the predicted output value
  - x_k is the vector of input values for case k

Perceptron

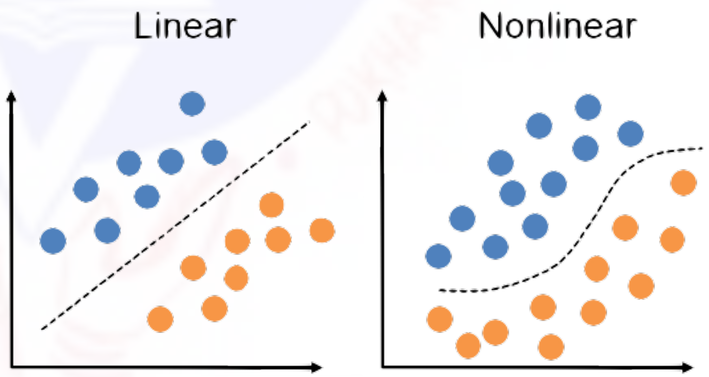
- **The Learning Algorithm:**



The perceptron with the learning procedure

Perceptron

- Single layer perceptron is a linear classifier.
- Single layer Perceptrons can learn only linearly separable patterns.
- Perceptrons can implement the linearly separable logic Gates like AND, OR, or Complement.
- It cannot model XOR function as it is non linearly separable.
- Multilayer Perceptron or feedforward neural network with two or more layers have the greater processing power and can process non-linear patterns as well.

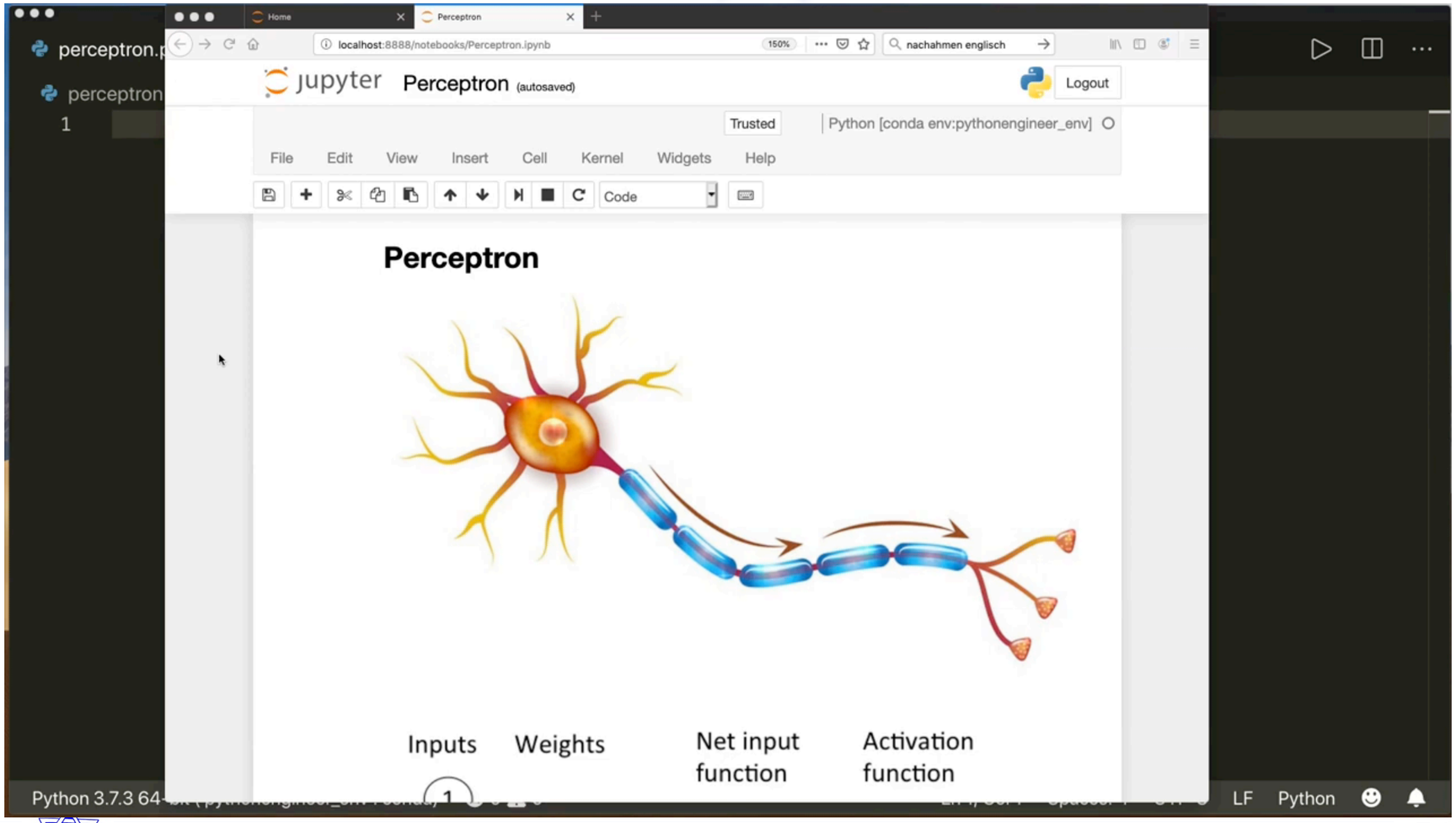


Example: Perceptron to learn OR

- Initially consider w1 = -0.2 and w2 = 0.4
- Training data say, x1 = 0 and x2 = 0, output is 0.
- Compute y = Step(w1*x1 + w2*x2) = 0. Output is correct so weights are not changed.
- For training data x1=0 and x2 = 1, output is 1
- Compute y = Step(w1*x1 + w2*x2) = 0.4 = 1. Output is correct so weights are not changed.
- Next training data x1=1 and x2 = 0 and output is 1
- Compute y = Step(w1*x1 + w2*x2) = - 0.2 = 0. Output is incorrect, hence weights are to be changed.
- Assume a = 0.2 and error e=1
- **wi = wi + (a * xi * e) gives w1 = 0 and w2 =0.4**
- With these weights, test the remaining test data.
- Repeat the process till we get stable result.

Python Code: Perceptron

- https://www.youtube.com/watch?v=t2ym2a3pb_Y



Python Code: Perceptron to learn AND

- https://pythonmachinelearning.pro/perceptrons-the-first-neural-networks/

Perceptron

```python
class Perceptron(object):
    """Implements a perceptron network"""
    def __init__(self, input_size, lr=1, epochs=100):
        self.W = np.zeros(input_size+1)
        # add one for bias
        self.epochs = epochs
        self.lr = lr

    def activation_fn(self, x):
        #return (x >= 0).astype(np.float32)
        return 1 if x >= 0 else 0

    def predict(self, x):
        z = self.W.T.dot(x)
        a = self.activation_fn(z)
        return a

    def fit(self, X, d):
        for _ in range(self.epochs):
            for i in range(d.shape[0]):
                x = np.insert(X[i], 0, 1)
                y = self.predict(x)
                e = d[i] - y
                self.W = self.W + self.lr * e * x
```

Training data set for AND gate

```python
if __name__ == '__main__':
    X = np.array([
        [0, 0],
        [0, 1],
        [1, 0],
        [1, 1]
    ])
    d = np.array([0, 0, 0, 1])

    perceptron = Perceptron(input_size=2)
    perceptron.fit(X, d)
    print(perceptron.W)
```

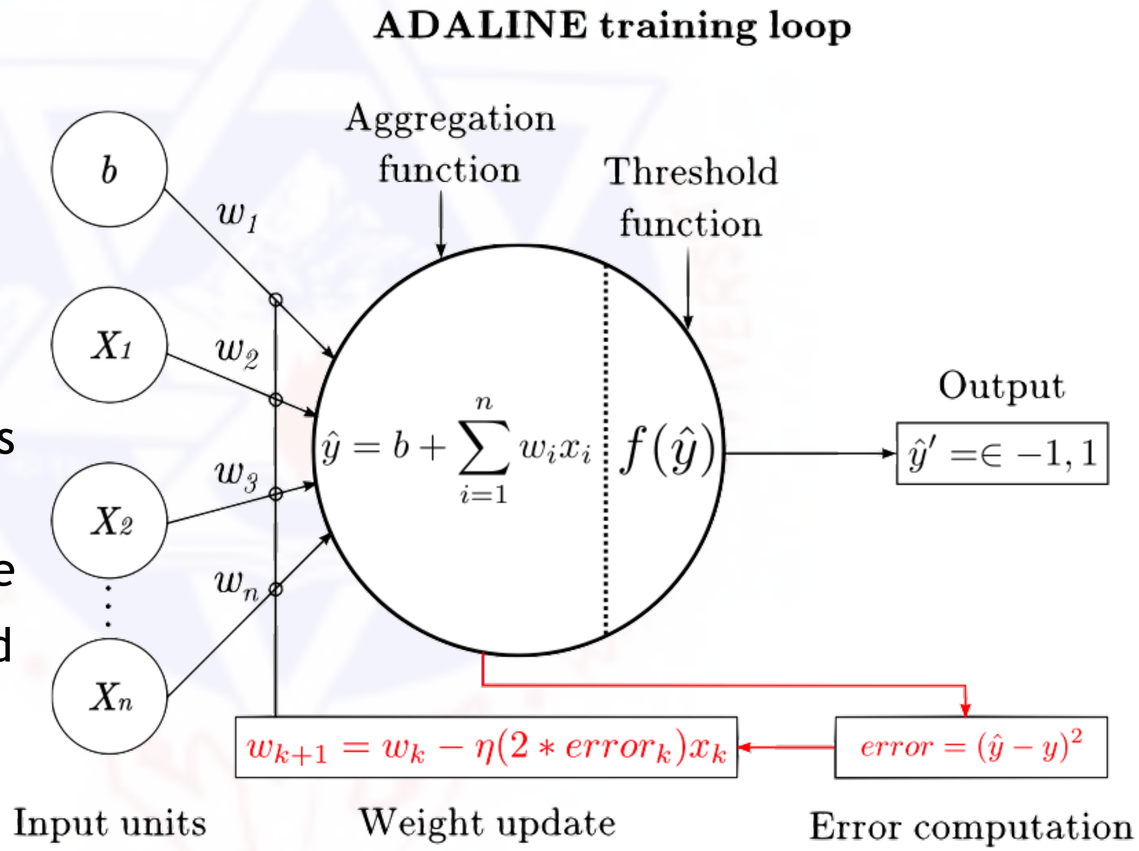
Adaline

- The ADALINE (Adaptive Linear Neuron) was introduced in 1959, shortly after Rosenblatt's perceptron, by Bernard Widrow and his doctoral student Ted Hoff at Stanford University.
- At the time, implementing an algorithm in a mainframe computer was slow and expensive, so they decided to build a small electronic device capable of being trained by the ADALINE algorithm to learn to classify patterns of inputs.
- Adaline is also based on the McCulloch–Pitts neuron.
- The main difference between the perceptron and the ADALINE is that the Adaline works by minimizing the mean squared error of the predictions of a linear function. This means that the learning procedure is based on the outcome of a linear function rather than on the outcome of a threshold function as in the perceptron.

Difference between Adaline and Perceptron

- The difference is in the ***learning phase***:

- (**In Adaline**), the weights are adjusted according to the weighted sum of the inputs (the net $\hat{y}$).
- (**In Perceptron**), the net is passed to the activation (transfer) function and the function's output $\hat{y}'$ is used for adjusting the weights.



Adaline

- The error in the ADALINE is calculated as $(y - \hat{y})^2$, (i.e. by squaring the difference between the expected value and the predicted value) for all cases, j=1 to j=n, in a given dataset.
- Once we add the squared difference for the entire dataset and divide by the total, we obtained the so-called **mean of squared errors** (MSE).
- The goal of the least-squares algorithm is to generate as little cumulative error as possible

stopping point for summation

expected value (+1, -1)

predicted value (linear function output)

$$MSE = \frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2 = \underbrace{(y_1 - \hat{y}_1)^2 + (y_2 - \hat{y}_2)^2 + \ldots + (y_n - \hat{y}_n)^2}_{\text{sum the sequence of squared errors from j=1 to j=n}}$$

division by total n cases

summation index

starting point for summation

Adaline

- The ADALINE acts as a binary classifier, splitting its input pattern space into two regions.

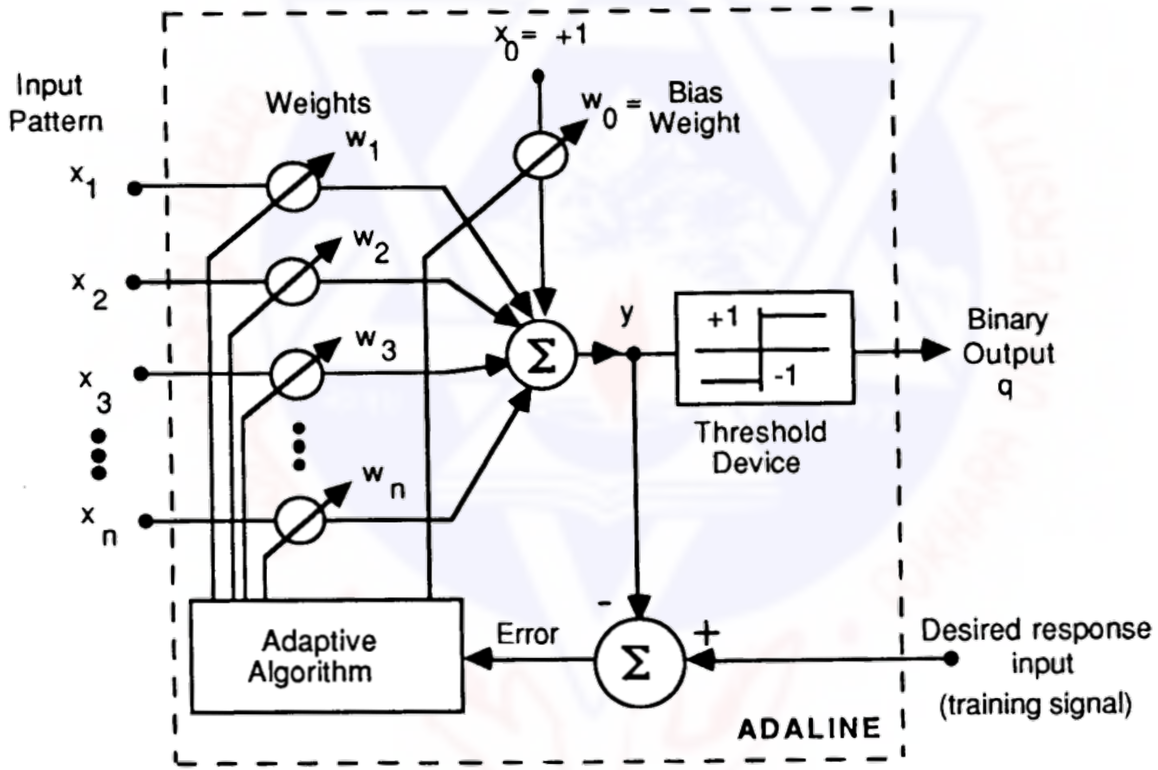
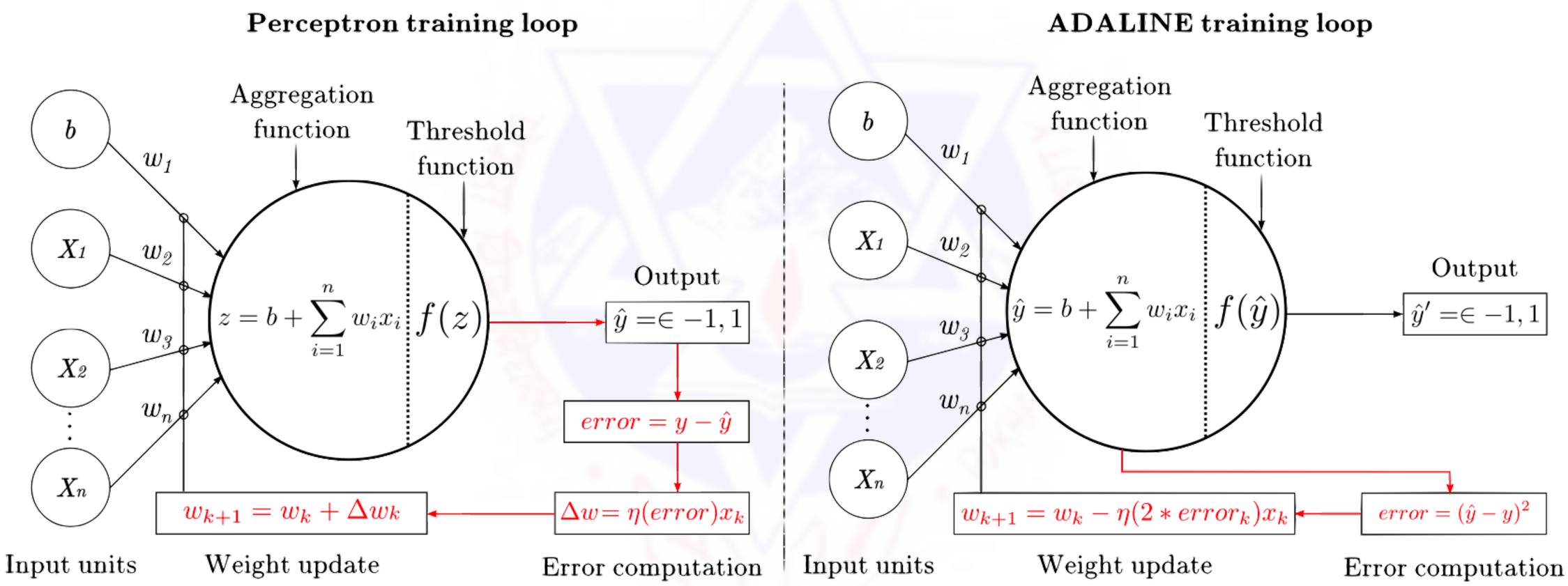


Figure 2: Adaptive linear neuron (ADALINE)

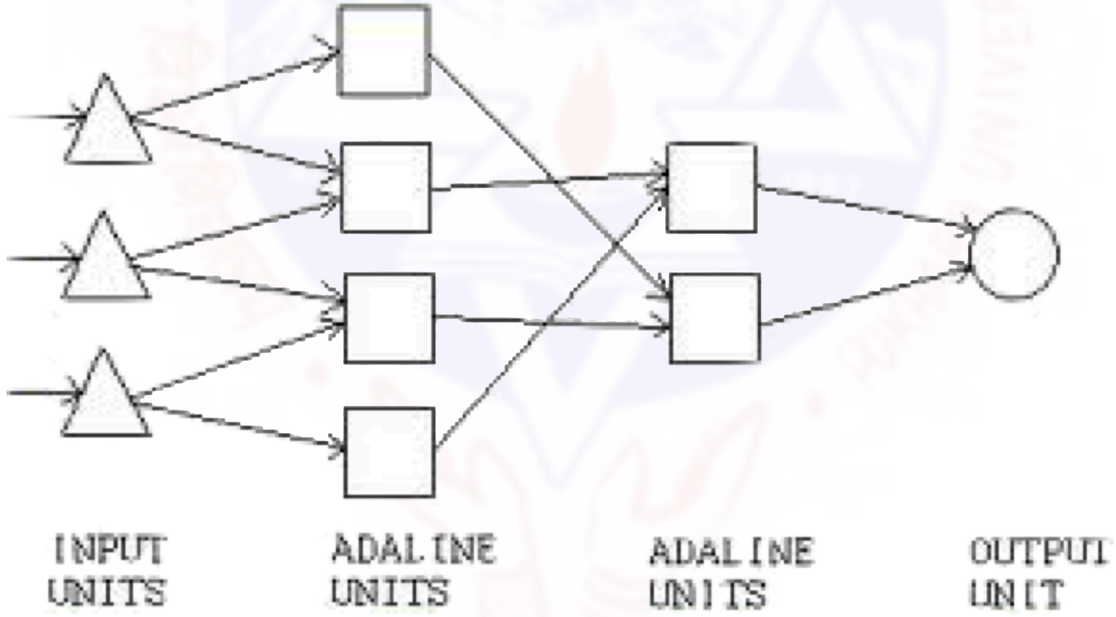
Difference between Adaline and Perceptron



Adaline

- The ADALINE was initially applied to:
  - speech and pattern recognition
- The main application of the ADALINE was in
  - adaptive filtering and
  - adaptive signal processing.
- Technologies that uses ADALINE:
  - adaptive antennas
  - adaptive noise canceling and
  - adaptive equalization in high-speed modems (which makes Wifi works well)

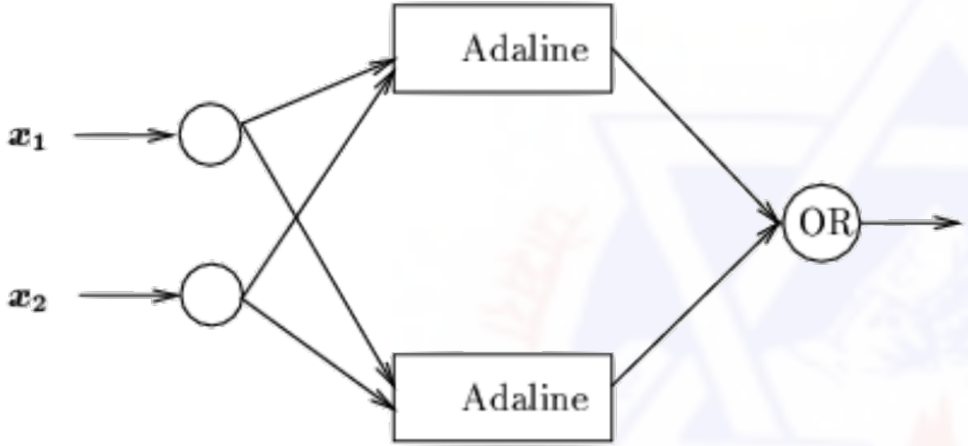
Madaline

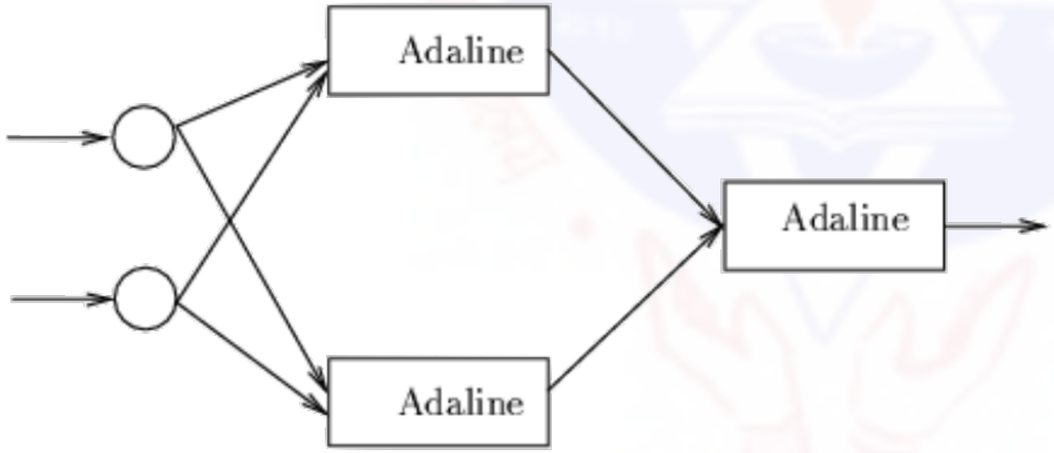
- Madaline stands for Multiple Adaptive Linear Neuron
- It is a network which consists of many Adalines in parallel in hidden layer.
- It has a single output unit.
- The training process of madaline is similar to that of adaline
- 

Madaline

- MADALINE (Many ADALINE) is a three-layer (input, hidden, output), fully connected, feed-forward artificial neural network architecture for classification that uses ADALINE units in its hidden and output layers
- its activation function is the sign function.
- It cannot be learned using backpropagation because the sign function is not differentiable
- Three different training algorithms for MADALINE networks:
  - MADALINE Rule I (MRI)
  - MADALINE Rule II (MRII)
  - MADALINE Rule III (MRIII)

Madaline





MRI net:

Output nodes with logic function

MRII net:

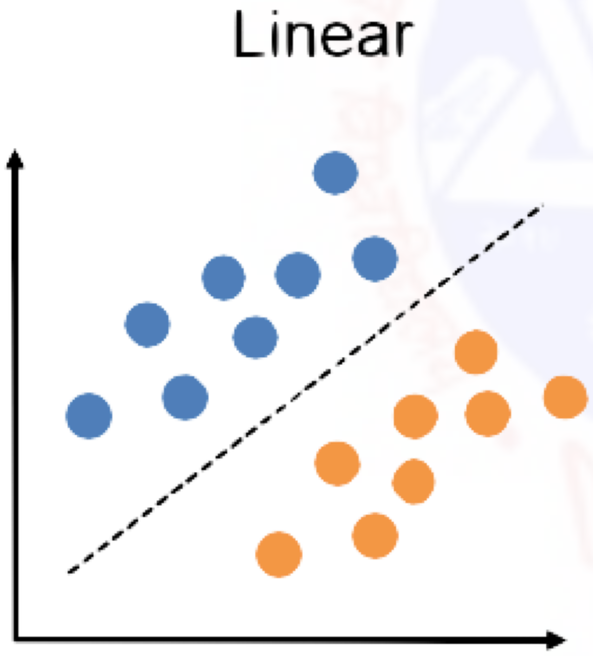
Output nodes are adalines

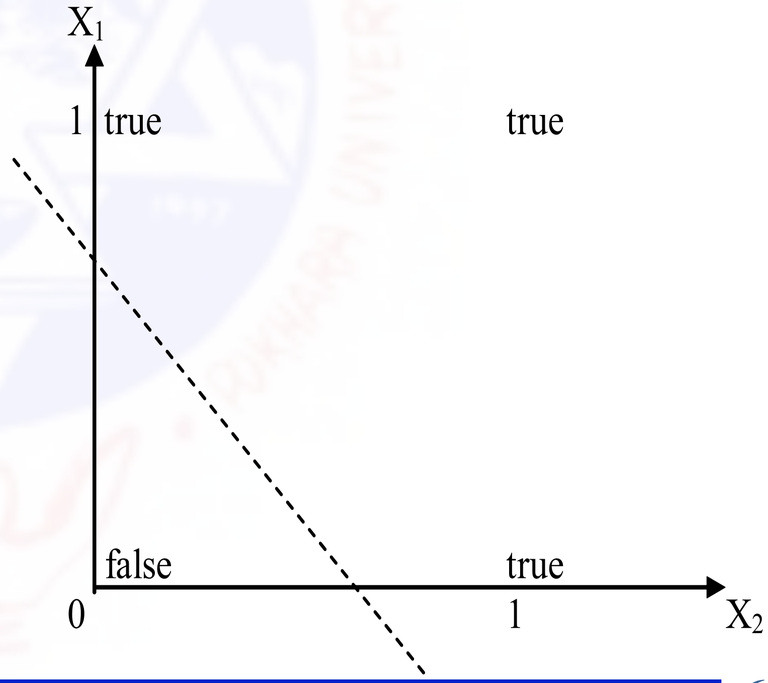
MRIII net:

Same as MRII, except the nodes with sigmoid function

Multi-layer Feed-Forward NN

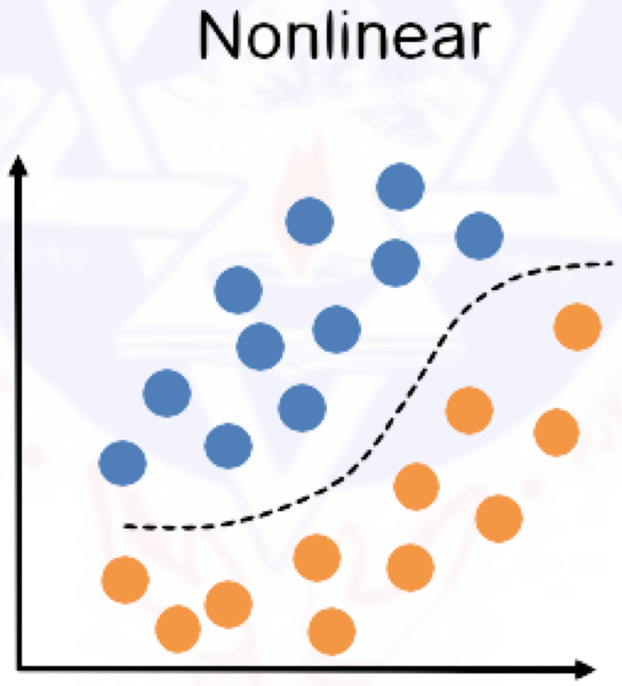
- **Linearly Separable problems:**
  - The perceptron can only model **linearly separable functions**
  - **Examples:** Boolean functions: OR, AND and Complement





Multi-layer Feed-Forward NN

- **Non-linearly Separable problems:**
  - The perceptron cannot model non linearly separable problem
  - **Examples: XOR function**

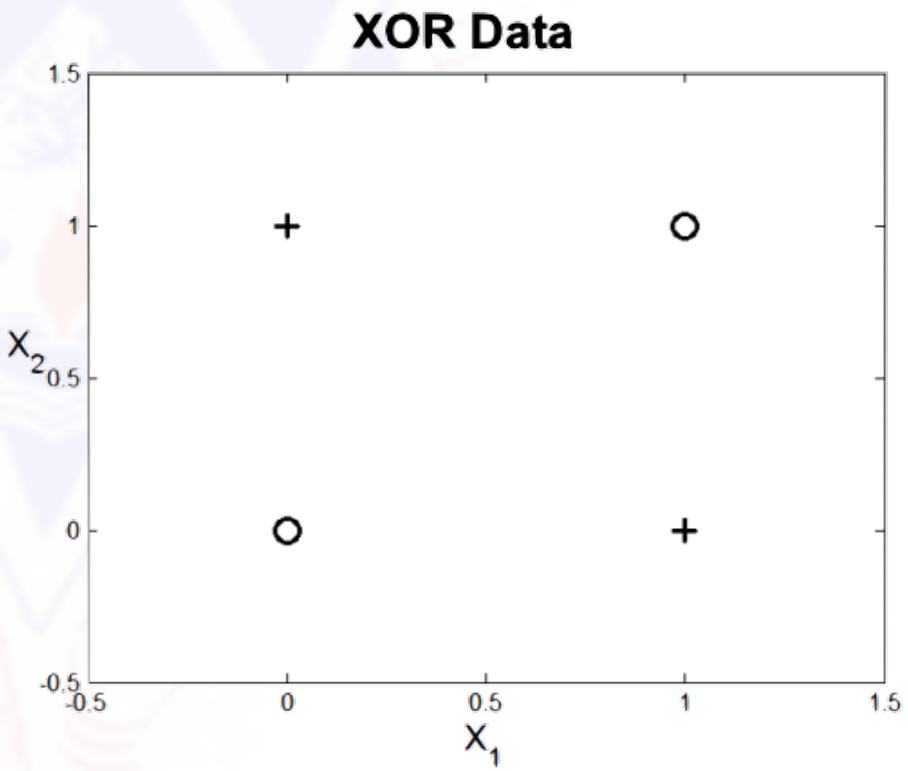


Multi-layer Feed-Forward NN

- **Non-linearly Separable problems:**
  - The perceptron cannot model **XOR function** as it is non linearly separable

$$y = x_1 \oplus x_2$$

x_1	x_2	y
0	0	-1
1	0	1
0	1	1
1	1	-1



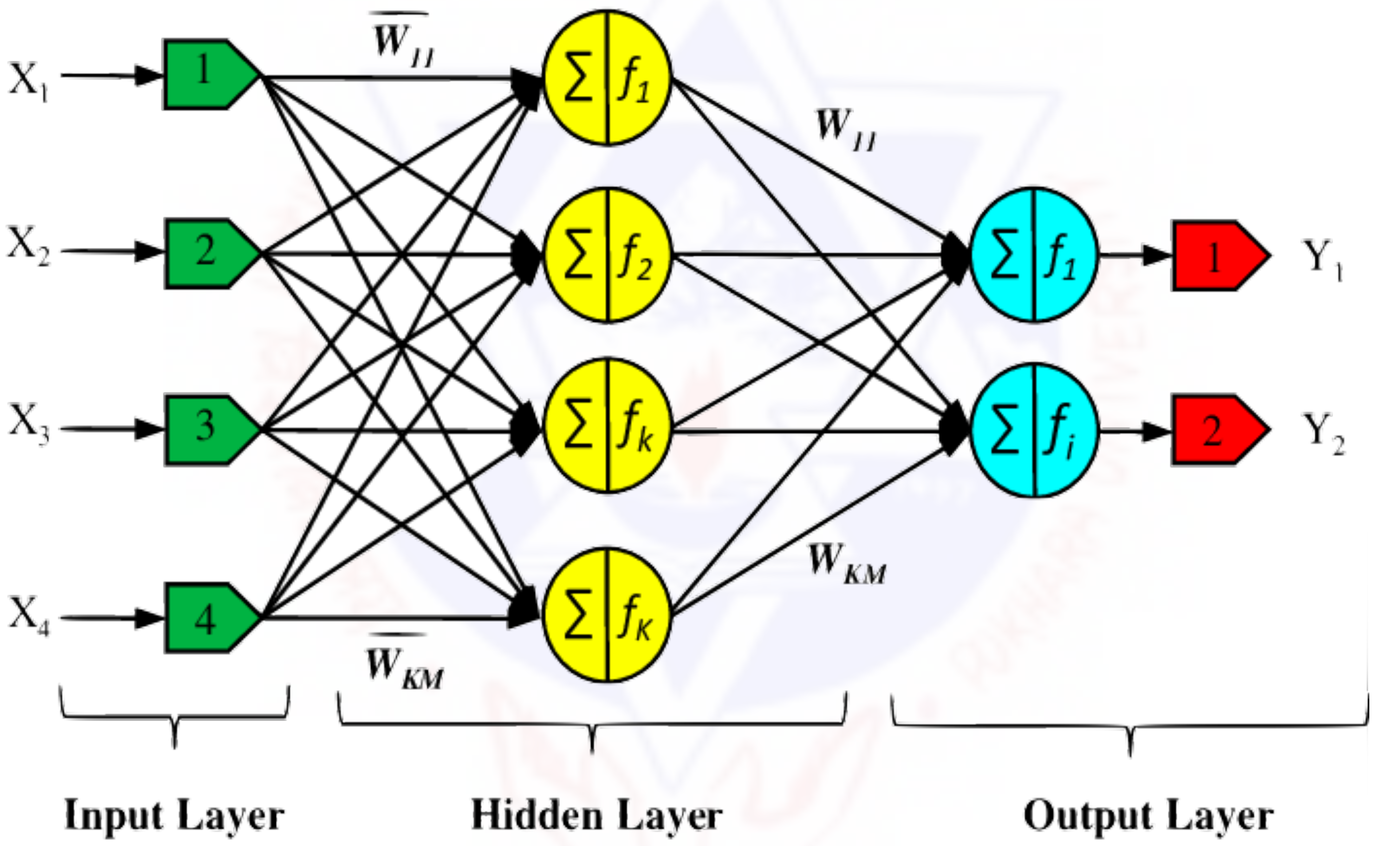
Solution:

- Multi-layer feed forward NN
- (e.g. multi-layer perceptron)

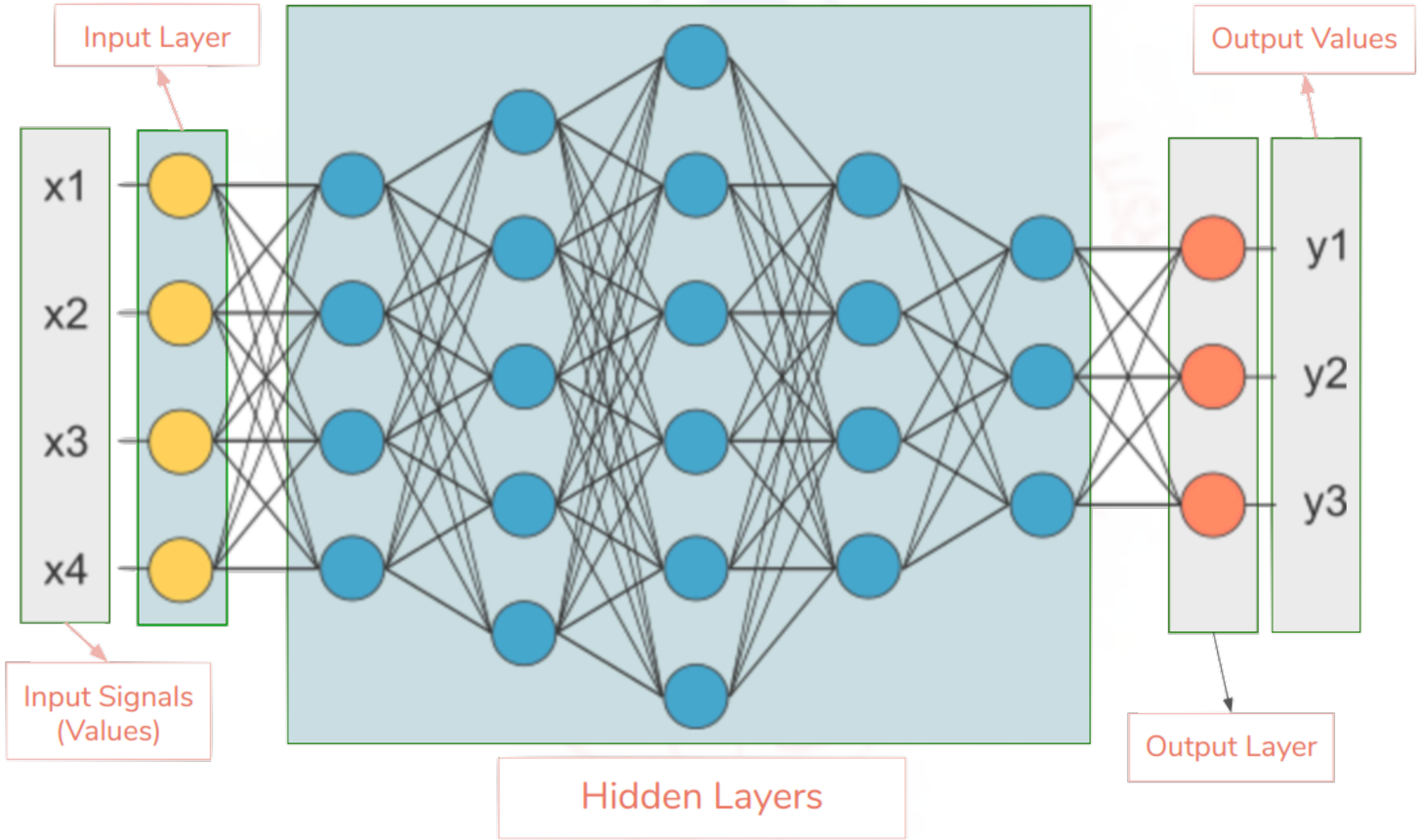
Multi-layer Feed-Forward NN

- Multi-layer FFNN is a more general network architecture, which consists:
    - Input layer
    - One or more hidden layers and
    - output layer
- The hidden nodes do not directly receive inputs nor send outputs to the external environment.
- Multi-layer FFNNs can handle non-linearly separable learning tasks.
- Multi-layer Perceptron is the class of multi-layer feed-forward neural network

Multi-layer Feed-Forward NN

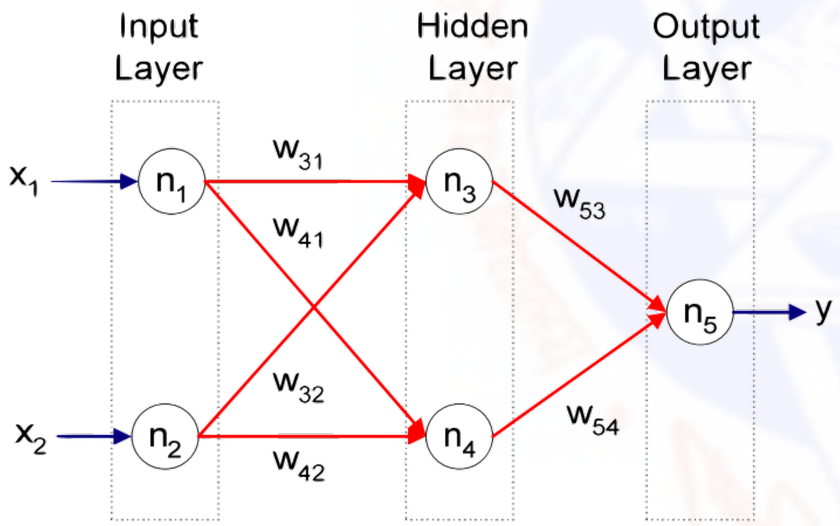


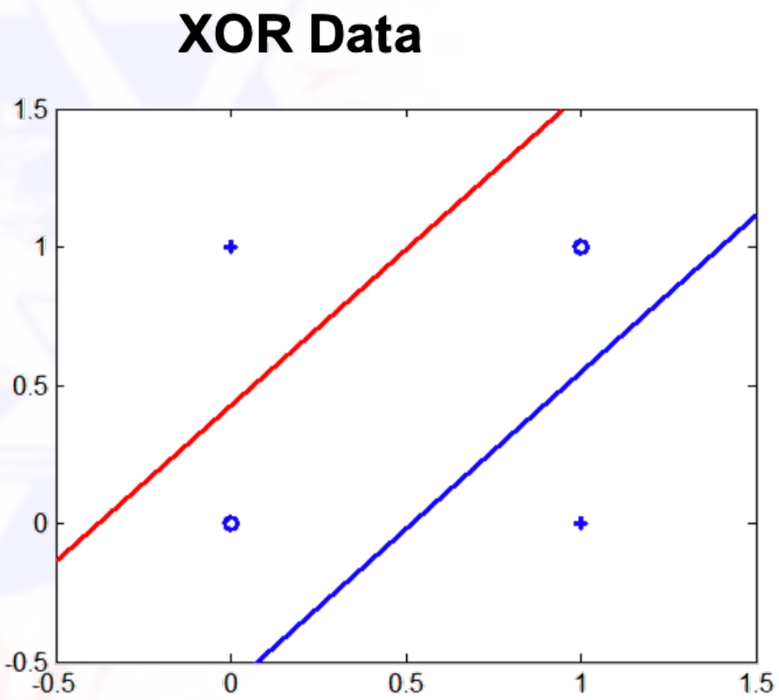
Multi-layer Feed-Forward NN



Multi-layer Feed-Forward NN

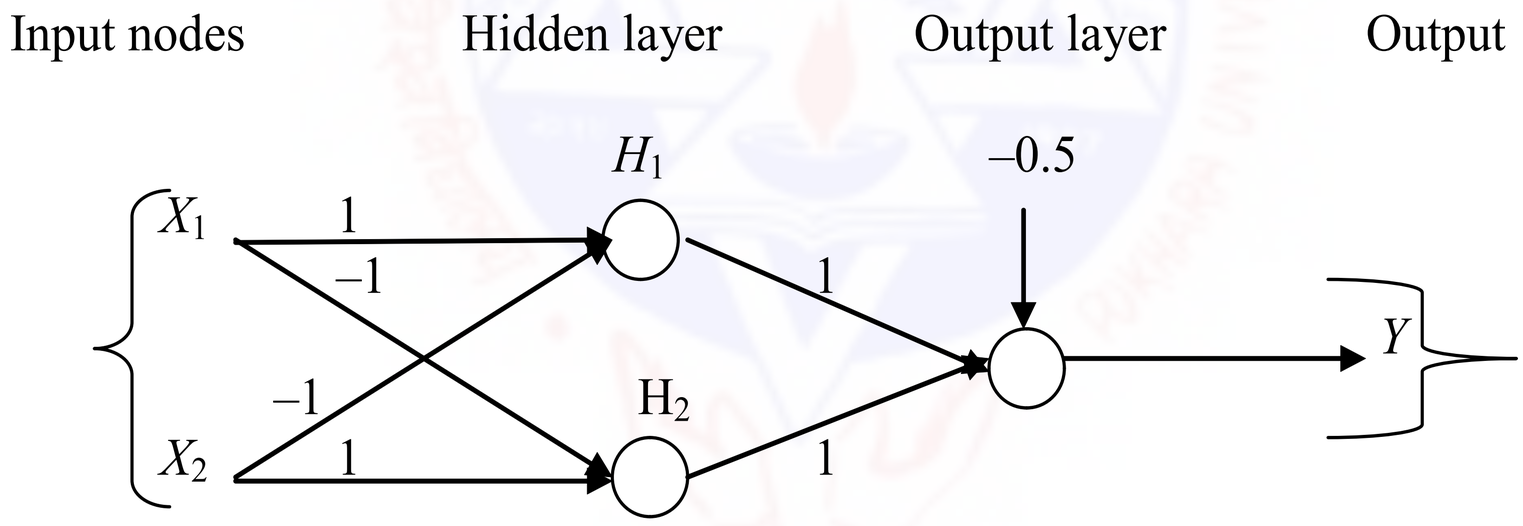
- Multi-layer feed-forward NN with at least one hidden layer can solve any type of classification task involving non-linear decision surface.





Multi-layer Feed-forward: XOR

- The ANN for XOR has two hidden nodes that realizes this non-linear separation and uses the sign (step) activation function.
- Arrows from input nodes to two hidden nodes indicate the directions of the weight vectors (1,-1) and (-1,1).
- The output node is used to combine the outputs of the two hidden nodes.



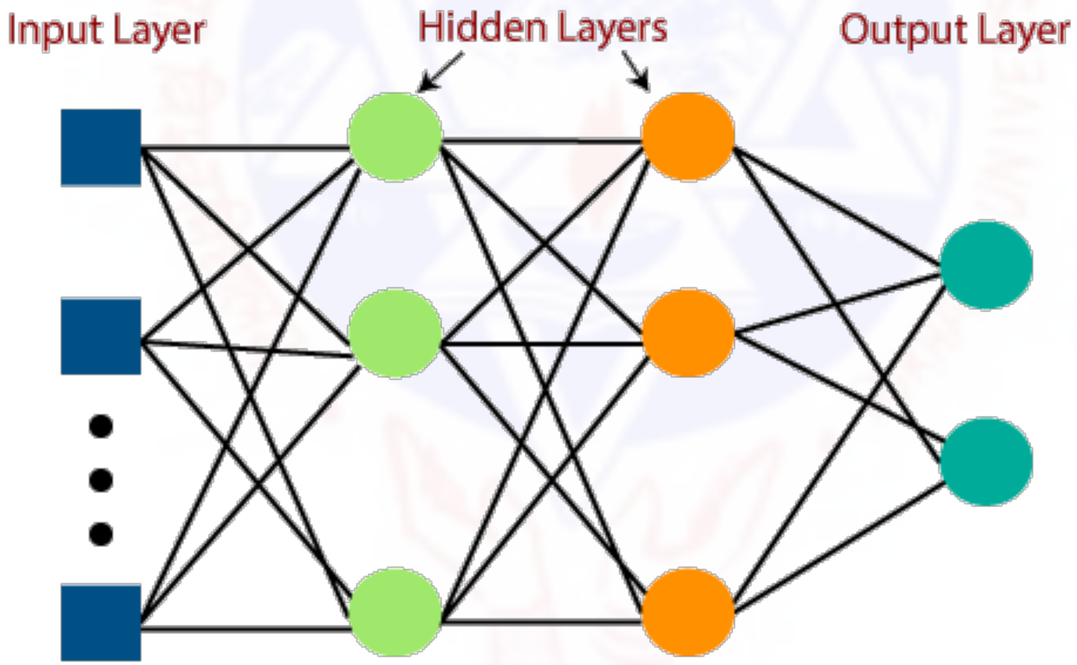
Multi-layer Feed-forward: XOR

Since we are representing two states by 0 (false) and 1 (true), we will map negative outputs (-1, -0.5) of hidden and output layers to 0 and positive output (0.5) to 1.

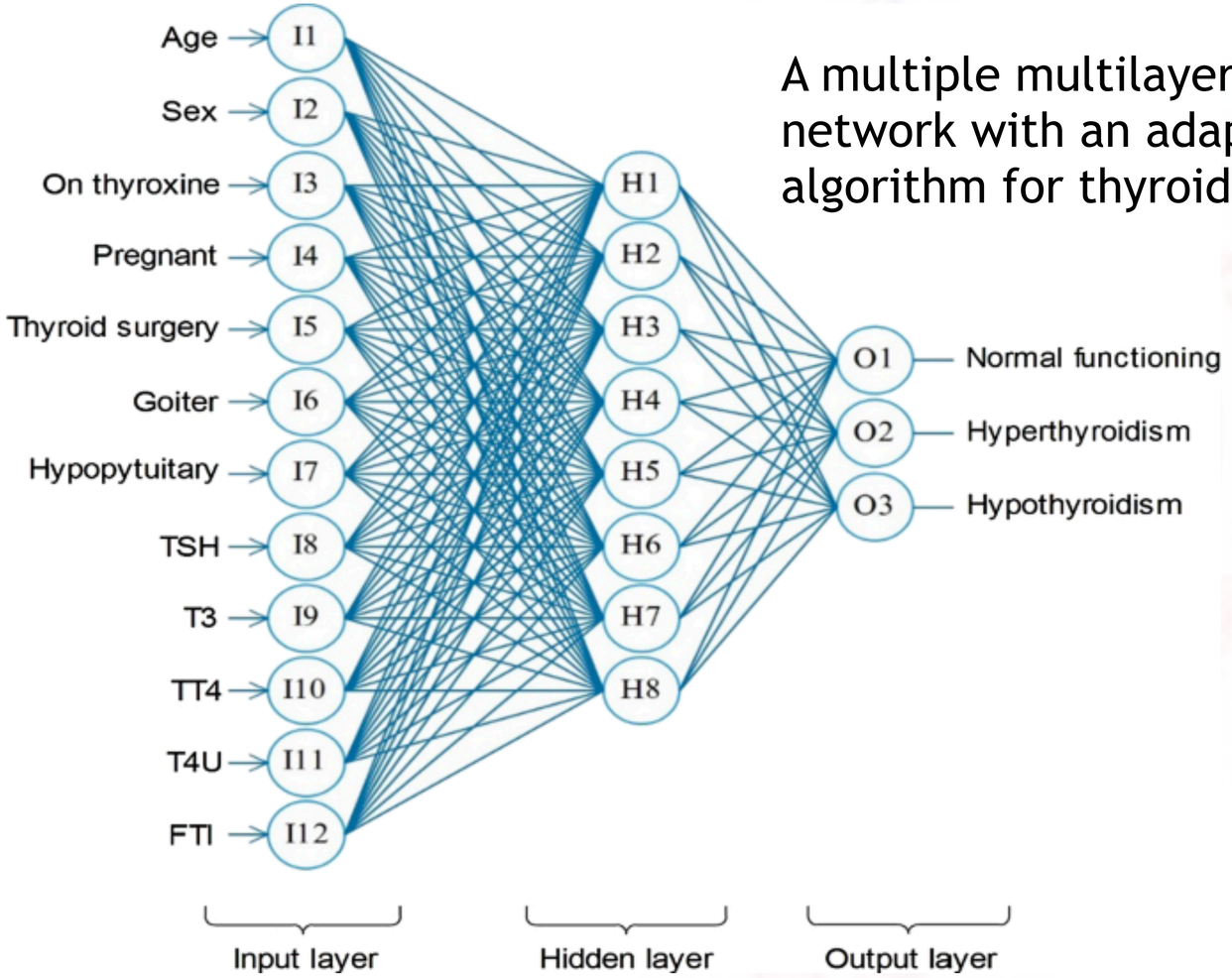
Inputs		Output of Hidden Nodes		Output Node	X_1 XOR X_2
X_1	X_2	H_1	H_2		
0	0	0	0	–0.5 → 0	0
0	1	–1 → 0	1	0.5 → 1	1
1	0	1	–1 → 0	0.5 → 1	1
1	1	0	0	–0.5 → 0	0

Multilayer Perceptron

- A **multilayer perceptron** (MLP) is a fully connected, feed-forward artificial neural network with at least three layers (**input, output, and at least one hidden layer**) where each layer is fully connected to the adjacent layer.

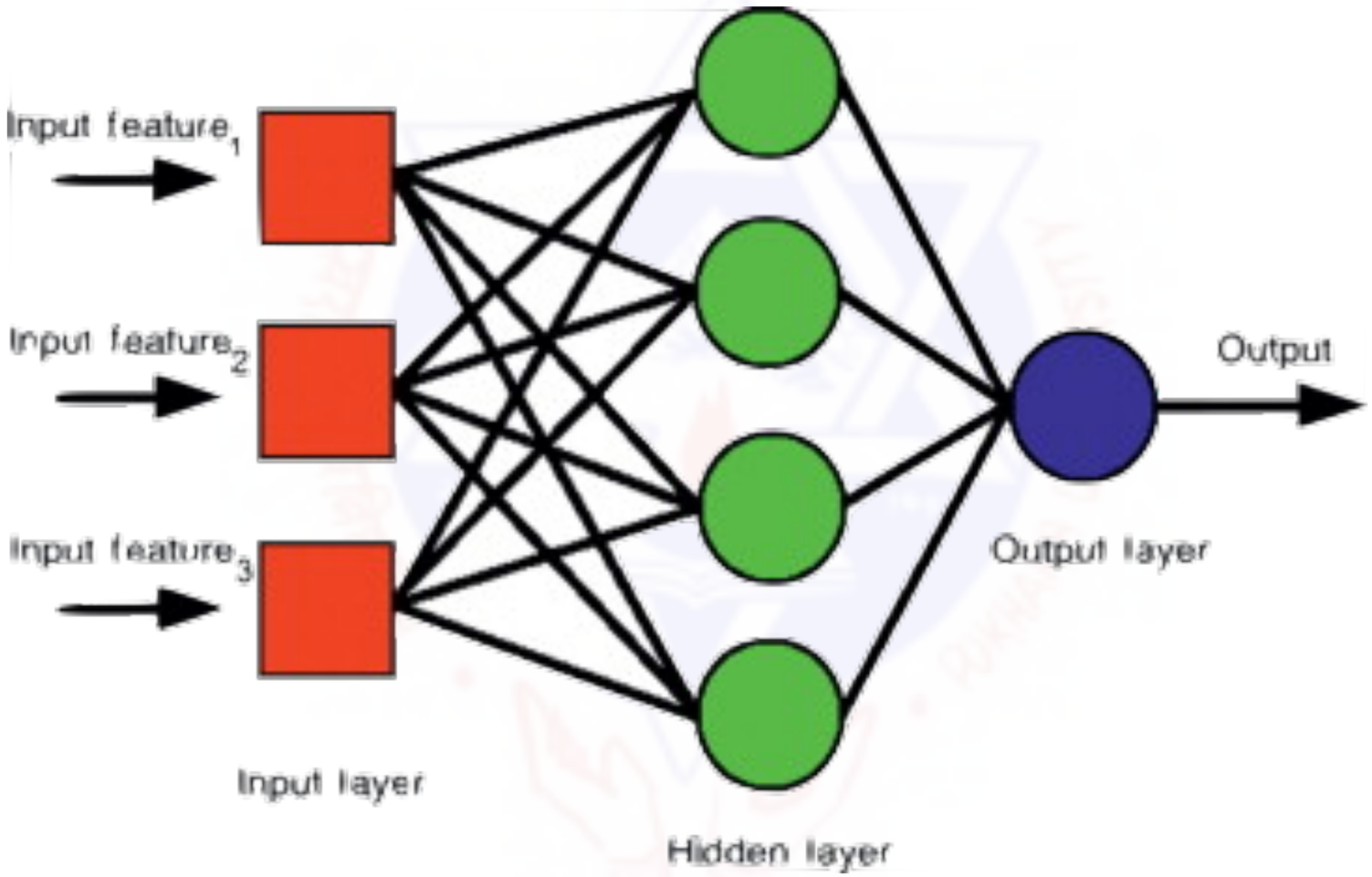


Multilayer Perceptron



A multiple multilayer perceptron neural network with an adaptive learning algorithm for thyroid disease diagnosis

Multilayer Perceptron



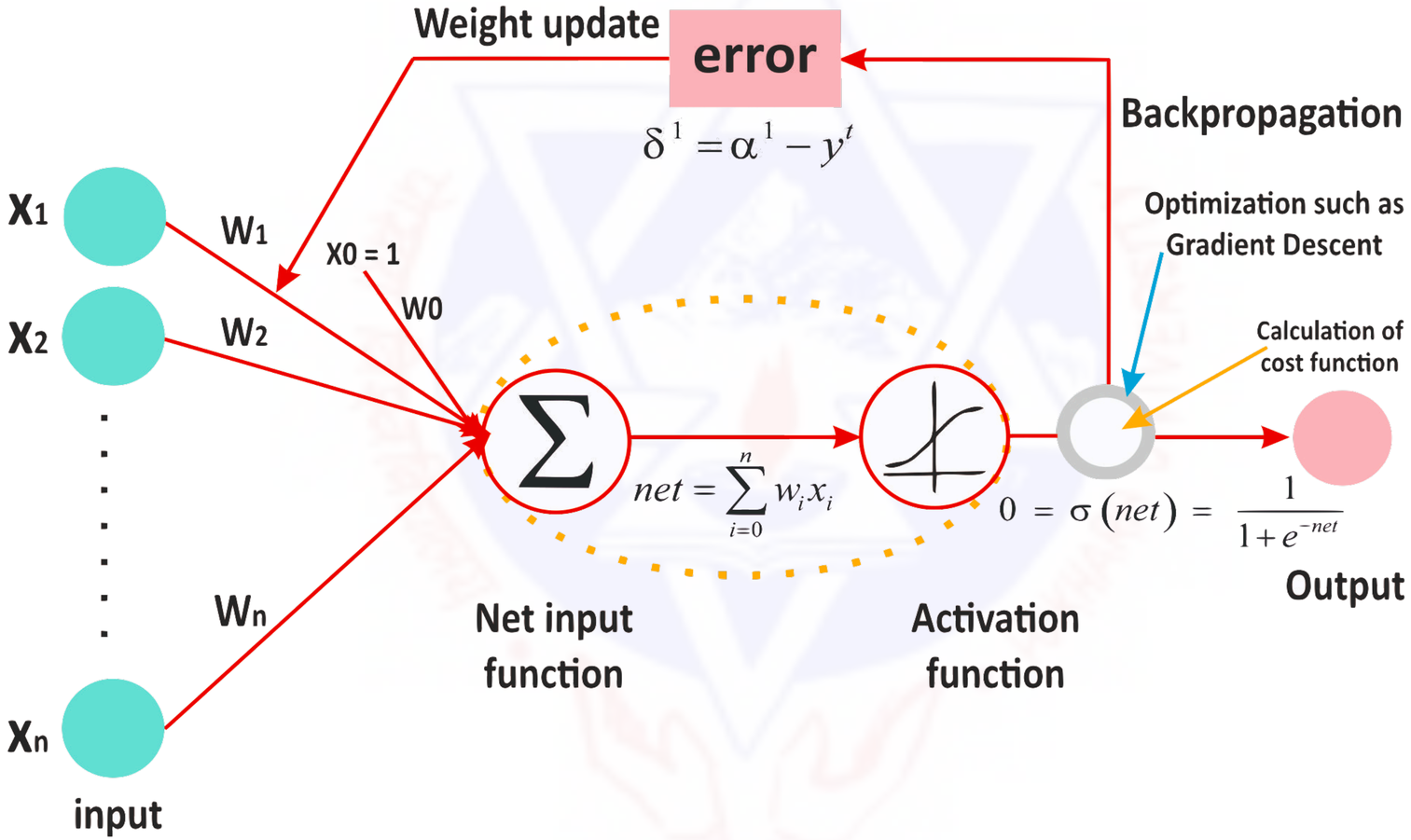
Multilayer Perceptron

- Each neuron usually is a **non-linear functional** processing unit.
- Except for the input nodes, each node is a neuron that uses a nonlinear activation function T
- The neurons in the MLP are trained with a supervised learning technique called **back propagation** learning algorithm.
- MLPs are designed to approximate any continuous function and can solve problems which are not linearly separable (like XOR function)
- The **Multilayer Perceptron** was developed to tackle this limitation of single layer perceptron which can only model the linearly separable problems
- The major use cases of MLP are pattern classification, recognition, prediction and approximation.

Backpropagation

- The backpropagation algorithm is a popular **technique to train ANNs**
- ***Back-propagation*** is the practice of fine-tuning the weights of a neural net based on the error rate (i.e. loss) obtained in the previous epoch (i.e. iteration).
- The optimization function (Gradient Descent) will help us find the weights that will yield a smaller loss in the next iteration.
- Proper tuning of the weights ensures lower error rates, making the model reliable by increasing its generalization.

Backpropagation



Backpropagation

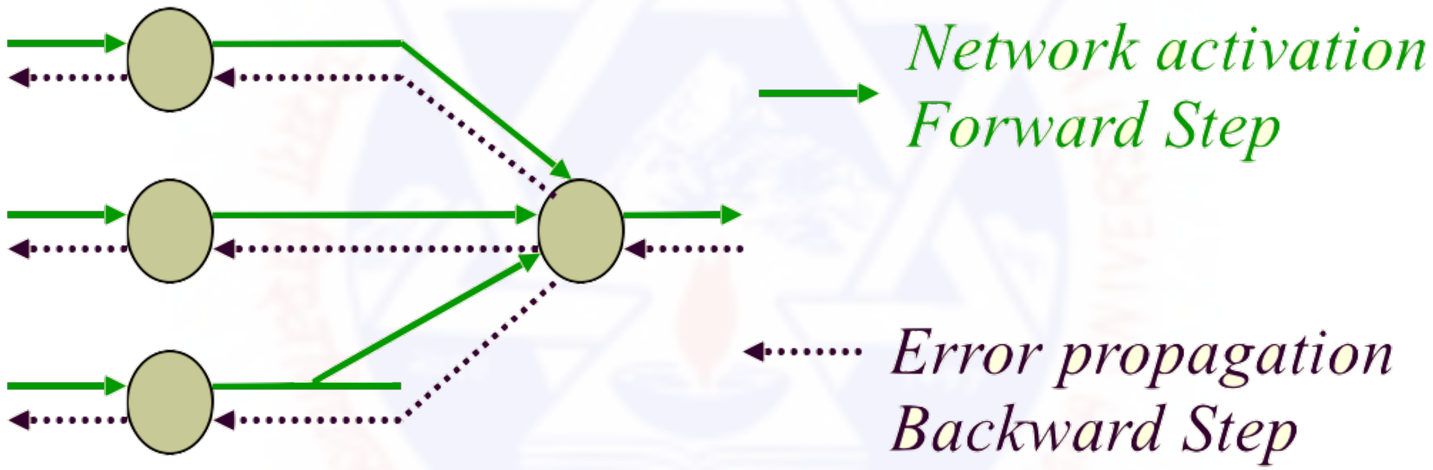
- Backpropagation algorithm works as follows:
  1. The input data vectors are propagated forward through the neural network to generate the output values from the output layer.
  2. Compare the generated output vector with the actual/desired output vector for that input data vector.
  3. Compute difference in error at the output units.
  4. If there is error, then Backpropagate error values to generate deltas at each node/neuron.
  5. Compute weight gradients by multiplying the output delta (error) and input activation.
  6. Use learning rate to determine percentage of the gradient to be subtracted from original weight and update the weight of the nodes.
- This is repeated multiple times with multiple iterations/epochs until we get satisfactory results.
- Typically backpropagation is used along with optimization algorithms or functions like stochastic gradient descent.

Backpropagation

- The Backpropagation algorithm learns in the same way as single perceptron.
- It searches for weight values that minimize the total error of the network over the set of training examples (training set).
- Backpropagation consists of the repeated application of the following two passes:
  - **Forward pass**: In this step, the network is activated on one example and the error of (each neuron of) the output layer is computed.
  - **Backward pass**: in this step the network error is used for updating the weights. The error is propagated backwards from the output layer through the network layer by layer. This is done by recursively computing the local gradient of each neuron.

Backpropagation

- Back-propagation training algorithm



- Backpropagation adjusts the weights of the NN in order to minimize the network total mean squared error.

Backpropagation

- Consider a network of three layers.
- Let us use i to represent nodes in input layer, j to represent nodes in hidden layer and k represent nodes in output layer.
- w_{ij} refers to weight of connection between a node in input layer and node in hidden layer.
- The following equation is used to derive the output value Yj of node j

$$Yj = \frac{1}{1+e^{-X_j}}$$

where, $X_j = \sum x_i \cdot w_{ij} - \theta_j$, $1 \leq i \leq n$; n is the number of inputs to node j, and θ_j is threshold for node j

Back propagation- Total Mean Squared Error

- The error of output neuron *k* after the activation of the network on the *n-th* training example (x(n), d(n)) is:

  $e_k(n) = d_k(n) - y_k(n)$

- The network error is the sum of the squared errors of the output neurons:

$$E(n) = \sum e_k^2(n)$$

- The total mean squared error is the average of the network errors of the training examples.

$$E_{AV} = \frac{1}{N}\sum_{n=1}^{N} E(n)$$

Back propagation- Weight Update Rule

- The Backprop weight update rule is based on the **gradient descent method:**
  - It takes a step in the direction yielding the maximum decrease of the network error E.
  - This direction is the opposite of the gradient of E.
- Iteration of the Backprop algorithm is usually terminated when the sum of squares of errors of the output values for all training data in an epoch is less than some threshold such as 0.01

$$w_{ij} = w_{ij} + \Delta w_{ij} \qquad \Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

Back propagation- Learning Algorithm

n=1;

initialize **weights** randomly;

while (stopping criterion not satisfied or n <max_iterations)

for each example (**x**,*d*)

- run the network with input x and compute the output y
- update the weights in backward order starting from those of the output layer:

$$w_{ji} = w_{ji} + \Delta w_{ji}$$

with Δw_{ji} computed using the (generalized) Delta rule

end-for

n = n+1;

end-while;

Backpropagation and Gradient Decent

- https://www.youtube.com/watch?v=odlgtjXduVg

Backpropagation and Gradient Decent

- https://www.youtube.com/watch?v=GJXKOrqZauk (Video)
- https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ (Explanation in text)

Radial Bias Function (RBF)

- A radial basis function network is a type of supervised artificial neural network
- An RBF network is a type of feed forward neural network
- The RBF network model was proposed by Broomhead and Lowe in 1988
- RBF network consists of an input layer, a **single hidden layer**, and an output layer.
- Since RBFs have only one hidden layer, the convergence of optimization objective is much faster. So, RBFs are proven to be universal approximators.
- RBF networks are a commonly used for function approximation problems.

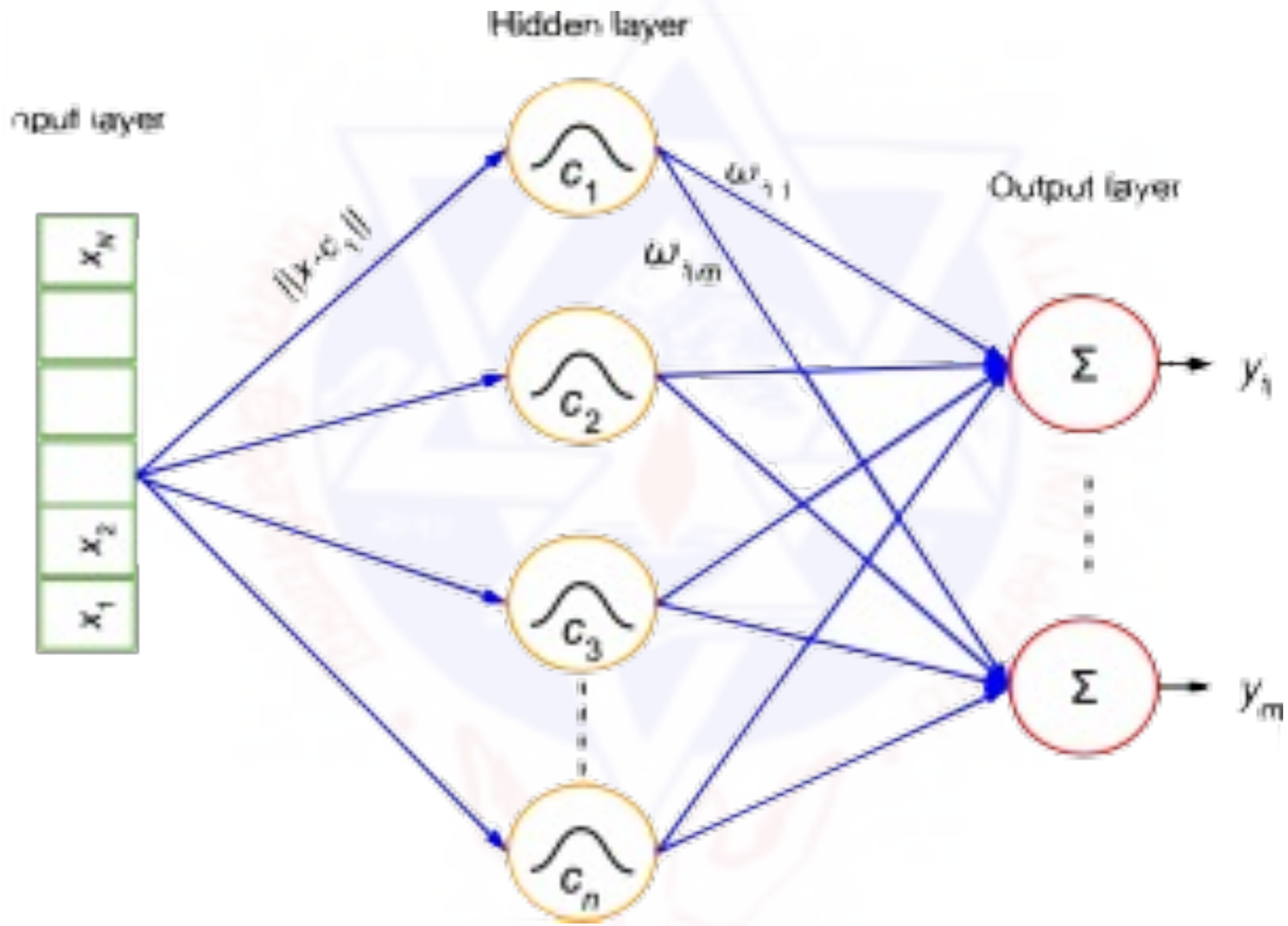
Radial Bias Function (RBF)

- The multilayer perceptron (MLP) trained with backpropagation (BP) rule is one of the most important neural network models.
- Due to its universal function approximation capability, the MLP is widely used in system identification, prediction, regression, classification, control, feature extraction, and associative memory
- The RBF network has equivalent capabilities as the MLP model, but with a much faster training speed, and thus it has become a good alternative to the MLP
- RBF networks have many applications like function approximation, interpolation, classification and time series prediction.
- All these applications serve various industrial interests like stock price prediction, anomaly detection in data, fraud detection in financial transaction etc.

RBF- Architecture

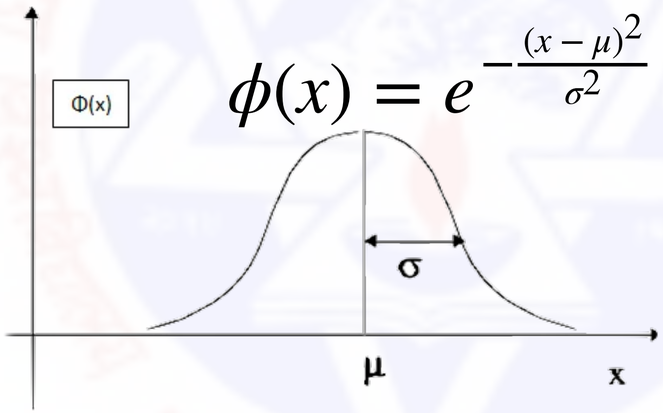
- RBF network has an input layer, a hidden layer, and an output layer.
- RBNN is **strictly limited** to have exactly **one hidden layer**. We call this hidden layer as **feature vector.**
- The Hidden layer of RBF consists of hidden neurons, and activation function of these neurons is a **Gaussian function**
- Hidden layer generates a signal corresponding to an input vector in the input layer, and corresponding to this signal, network generates a response

RBF- Architecture



RBF- Architecture

- To generate an output, neuron process the input signal through a function called activation function of the neuron.
- In RBF activation function of hidden neuron is $\phi(X)$ i.e. for an input vector X output produced by the hidden neuron will be $\phi(X)$.



$$\phi(x) = e^{-\frac{(x-\mu)^2}{\sigma^2}}$$

μ is center

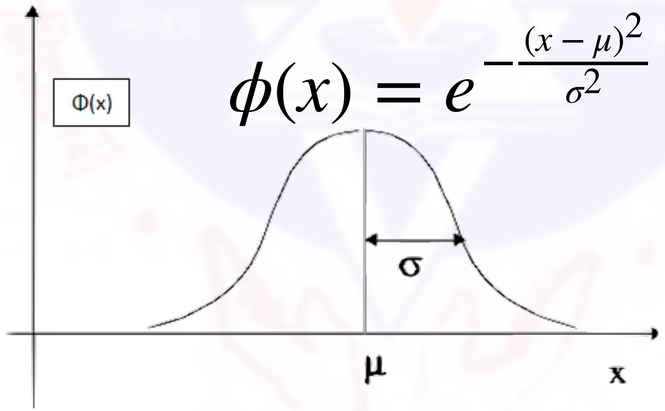
σ is radius

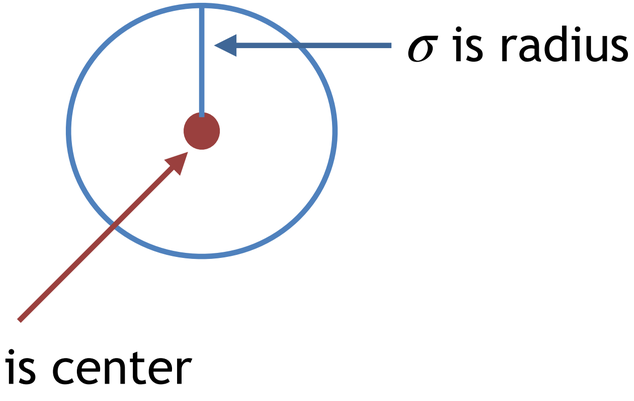
- Above figure represents a Gaussian neural activation function for 1-D input x with center mean μ and σ is the width of the neuron
- Here $\phi(x)$ represents the output of Gaussian node for given value of x.

RBF- Architecture

- It can be clearly seen that the signal strength decreases as the input (X) move away from the center.
- The basic idea of this model is that the entire feature vector space is partitioned by Gaussian neural nodes, where each node generates a signal corresponding to an input vector, and strength of the signal produced by each neuron depends on the distance between its center and the input vector.

$$\phi(x) = e^{-\frac{(x-\mu)^2}{\sigma^2}}$$





RBF- Architecture

- The hidden layer takes the input in which the pattern might not be linearly separable
- The computation in the hidden layer can be mathematically written as follow:

$$\phi_i(X) = e^{\frac{-||\bar{X} - \bar{\mu}||^2}{2\sigma_i^2}}$$

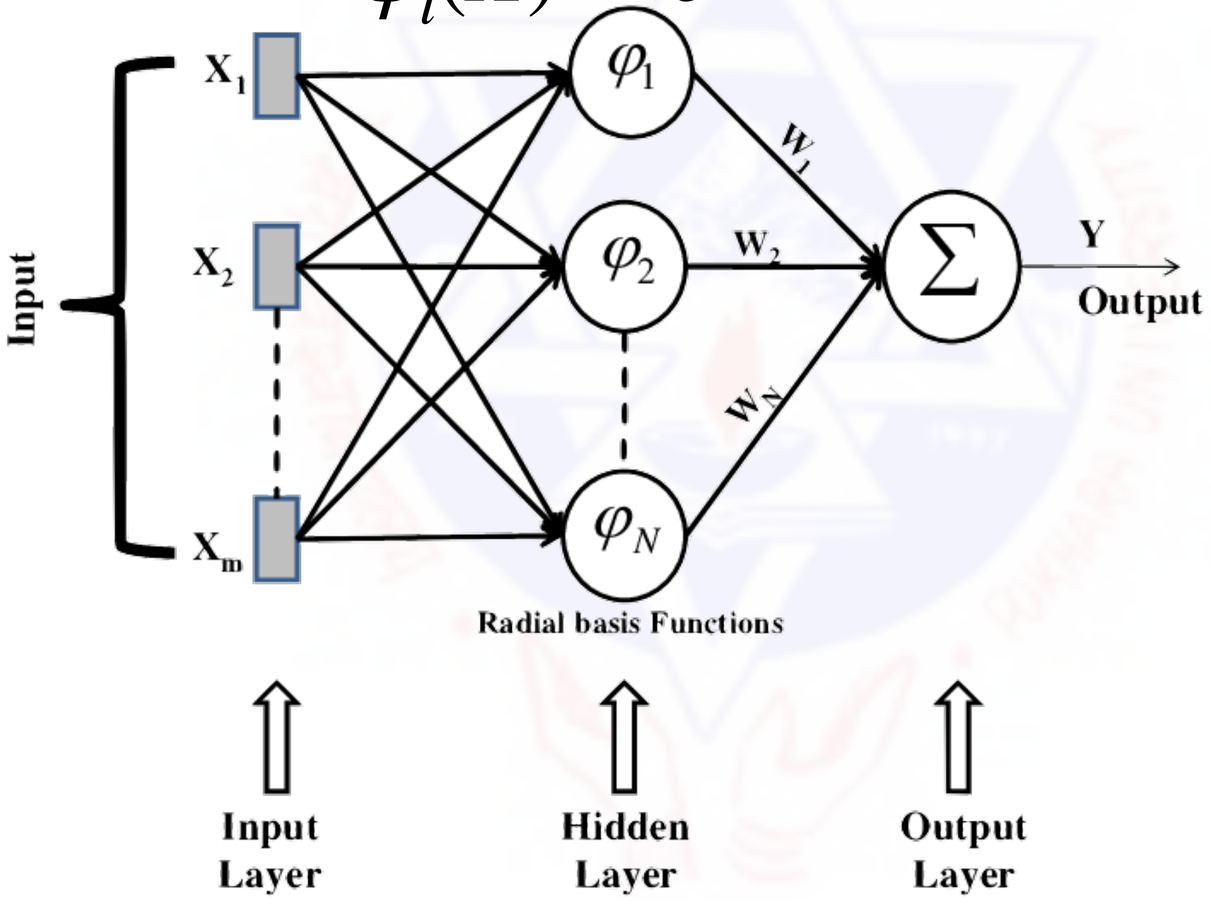
RBF- Architecture

- 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
- The computation in the output layer can be mathematically written as follow:(y the prediction result is given by)

$$y = \sum_{i=1}^{n} w_i \phi_i(x)$$

RBF- Architecture

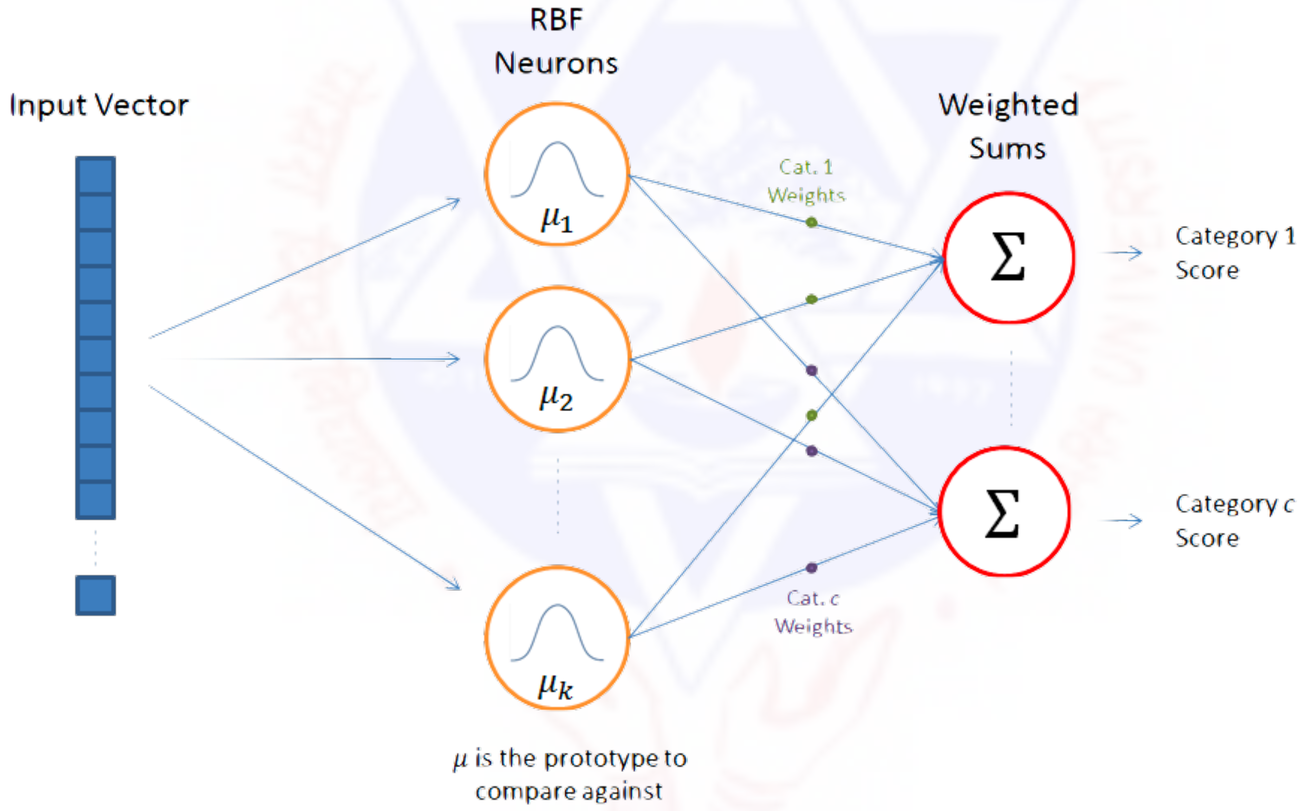
$$\phi_i(X) = e^{\frac{-||\bar{X}-\bar{\mu}||^2}{2\sigma_i^2}}$$



$$y = \sum_{i=1}^{n} w_i \phi_i(x)$$

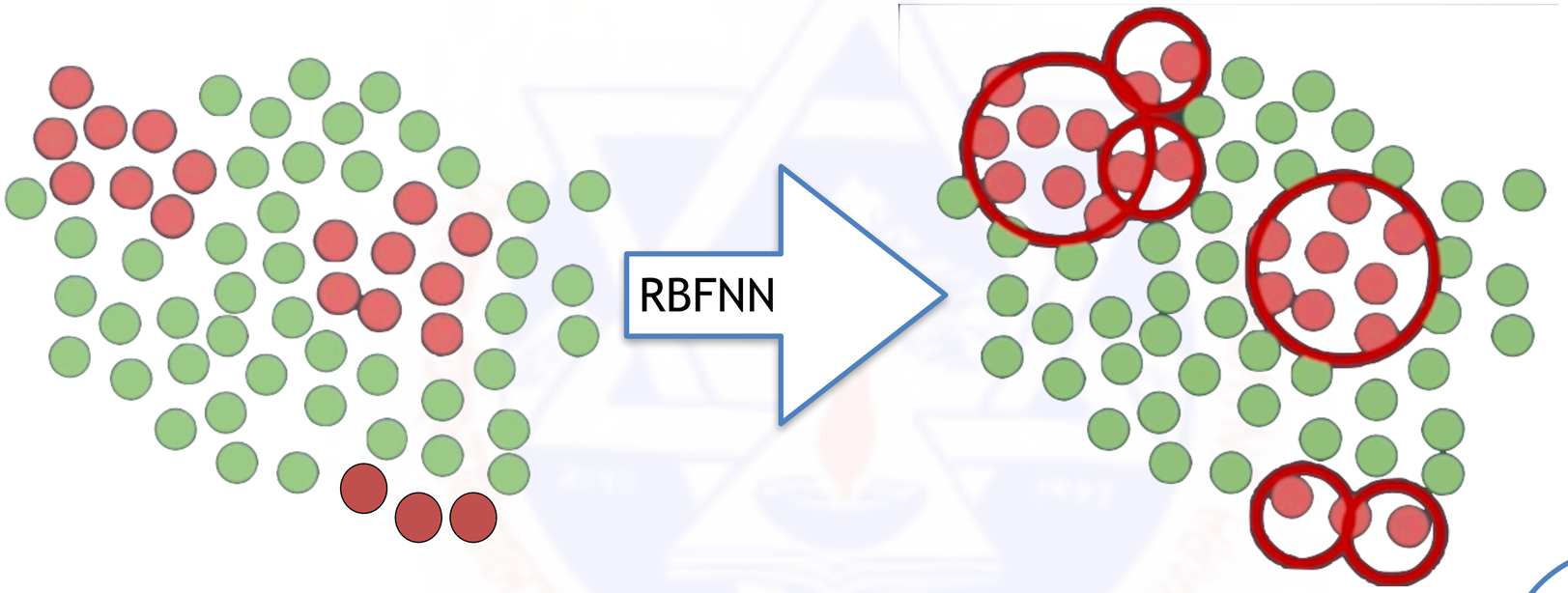
RBF- Architecture

$$\phi_i(X) = e^{\frac{-||\bar{X}-\bar{\mu}||^2}{2\sigma_i^2}}$$



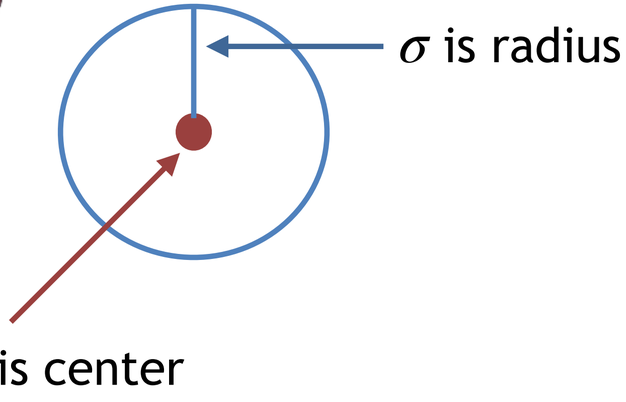
$$y = \sum_{i=1}^{n} w_i \phi_i(x)$$

Radial Bias Function



$$\phi_i(X) = e^{\frac{-||\bar{X}-\bar{\mu}||^2}{2\sigma_i^2}}$$

$$y = \sum_{i=1}^{n} w_i \phi_i(x)$$



https://www.youtube.com/watch?v=1Cw45yNm6VA

Radial Bias Function

A number of functions can be used as the RBF [6, 13, 14]

$$\phi(r) = e^{-r^2/2\sigma^2}, \quad \text{Gaussian}, \tag{2.1}$$

$$\phi(r) = \frac{1}{\left(\sigma^2 + r^2\right)^{\alpha}}, \quad \alpha > 0, \tag{2.2}$$

$$\phi(r) = \left(\sigma^2 + r^2\right)^{\beta}, \quad 0 < \beta < 1, \tag{2.3}$$

$$\phi(r) = r, \quad \text{linear}, \tag{2.4}$$

$$\phi(r) = r^2 \ln(r), \quad \text{thin-plate spline}, \tag{2.5}$$

$$\phi(r) = \frac{1}{1 + e^{(r/\sigma^2)-\theta}}, \quad \text{logistic function}, \tag{2.6}$$

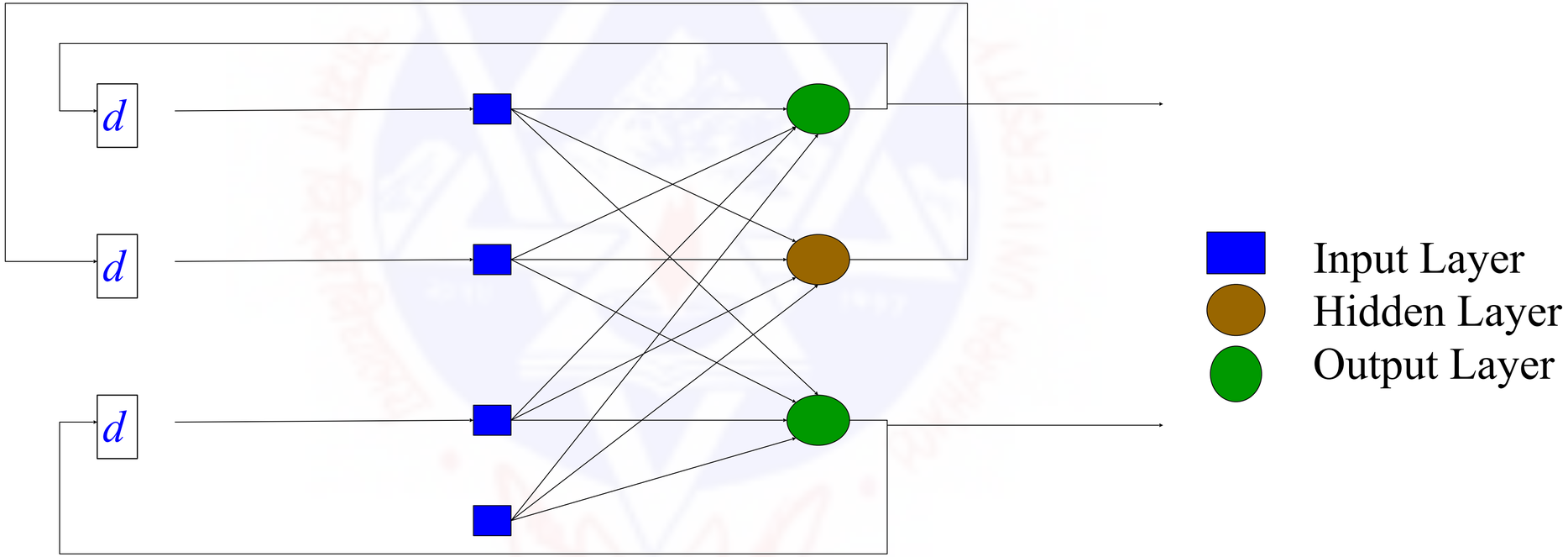
where $r > 0$ denotes the distance from a data point $\vec{x}$ to a center $\vec{c}$, σ in (2.1), (2.2), (2.3), and (2.6) is used to control the smoothness of the interpolating function, and θ in (2.6) is an adjustable bias. When β in (2.3) takes the value of 1/2, the RBF becomes Hardy's multiquadric function, which is extensively used in surface interpolation with very good results [6]. When α in (2.2) is unity, $\phi(r)$ is suitable for DSP implementation [18].

Recurrent Network

- **Feed Forward Neural Network (FFNN)** is acyclic where data passes from input to the output nodes and not vice versa.
  - Once the FFNN is trained, its state is fixed and does not alter as new data is presented to it. It does not have memory.
- **Recurrent network** can have connections that go backward from output to input nodes and models dynamic systems.
  - In this way, a recurrent network's internal state can be altered as sets of input data are presented. It can be said to have memory.
  - It is useful in solving problems where the solution depends not just on the current inputs but on all previous inputs.
- Applications
  - predict stock market price,
  - weather forecast

Recurrent Network

- Recurrent Network with ***hidden neuron***: unit delay operator ***d*** is used to model a dynamic system

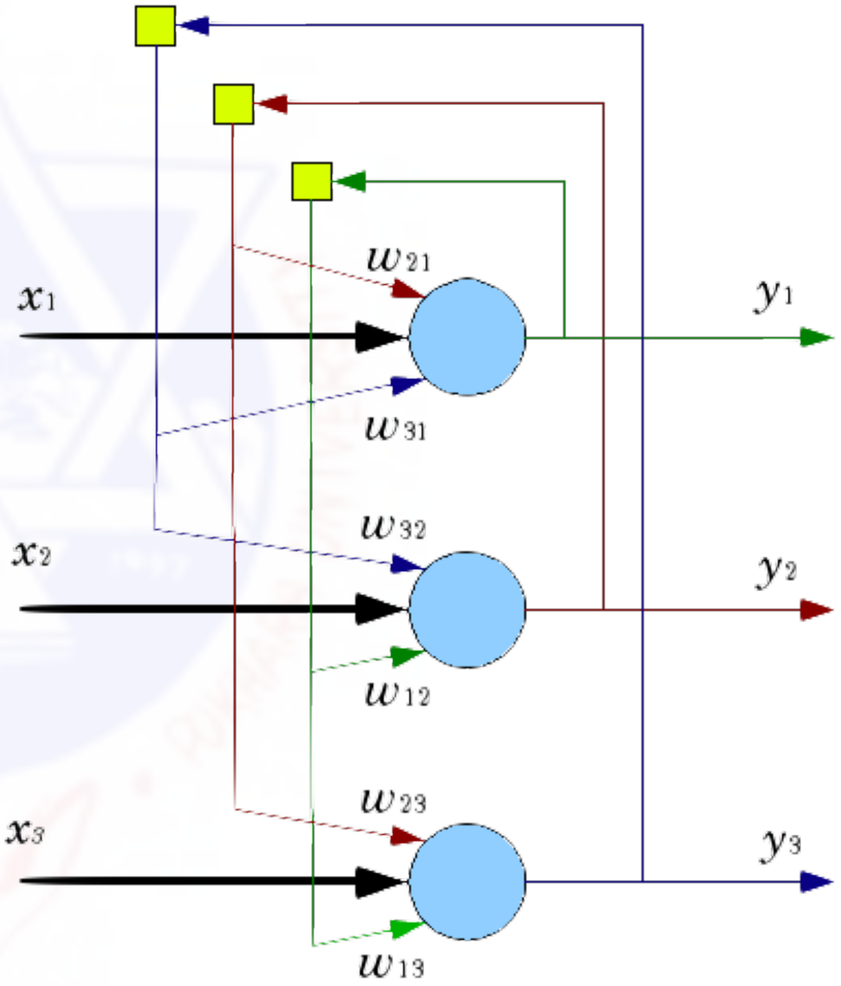


Recurrent Network

- During learning phase,
  - a recurrent network feeds its inputs through the network, including feeding data back from outputs to inputs
  - process is repeated until the values of the outputs do not change.
- This state is called equilibrium or stability
- Recurrent networks can be trained by using back-propagation algorithm.
- In this method, at each step, the activation of the output is compared with the desired activation and errors are propagated backward through the network.
- Once this training process is completed, the network becomes capable of performing a sequence of actions.

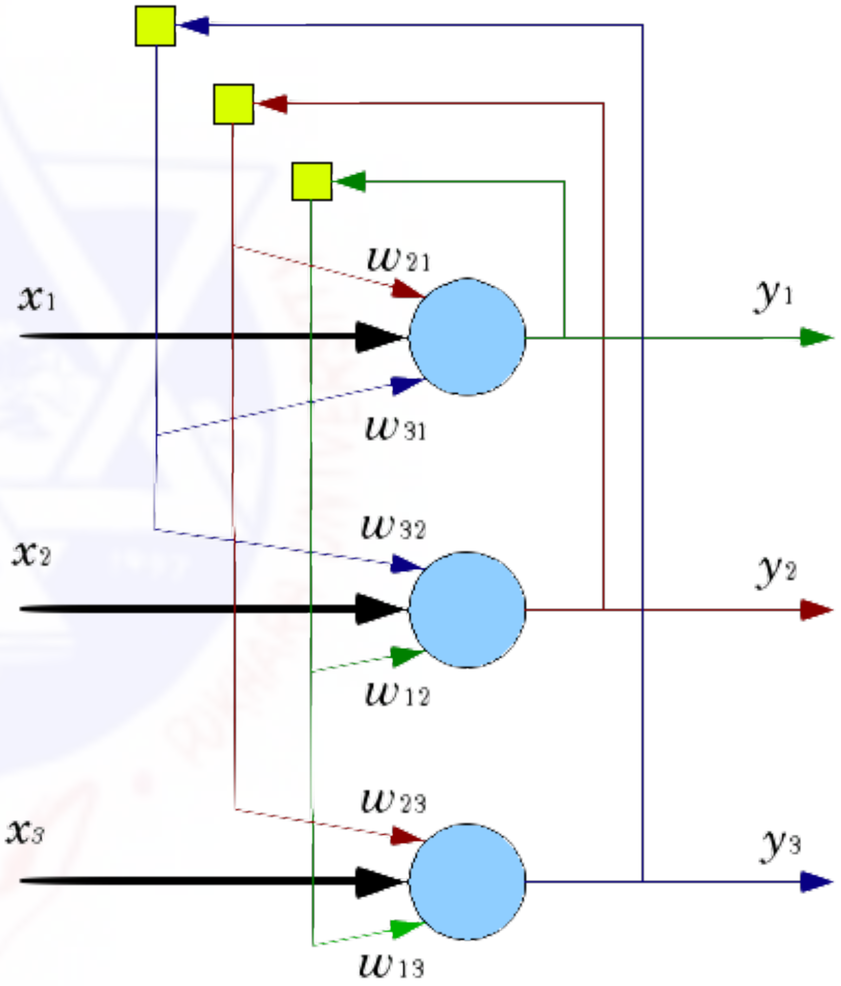
Hopfield Network

- The Hopfield Neural Networks is invented by Dr John J. Hopfield in 1982
- It consists of a single layer of *one or more* fully connected recurrent neurons.
- All the neurons act as input and output
- Every output is connected to every other neuron but no self feedback in the network
- It is generally used in performing auto association and optimization tasks.
- It is a form of recurrent artificial neural network



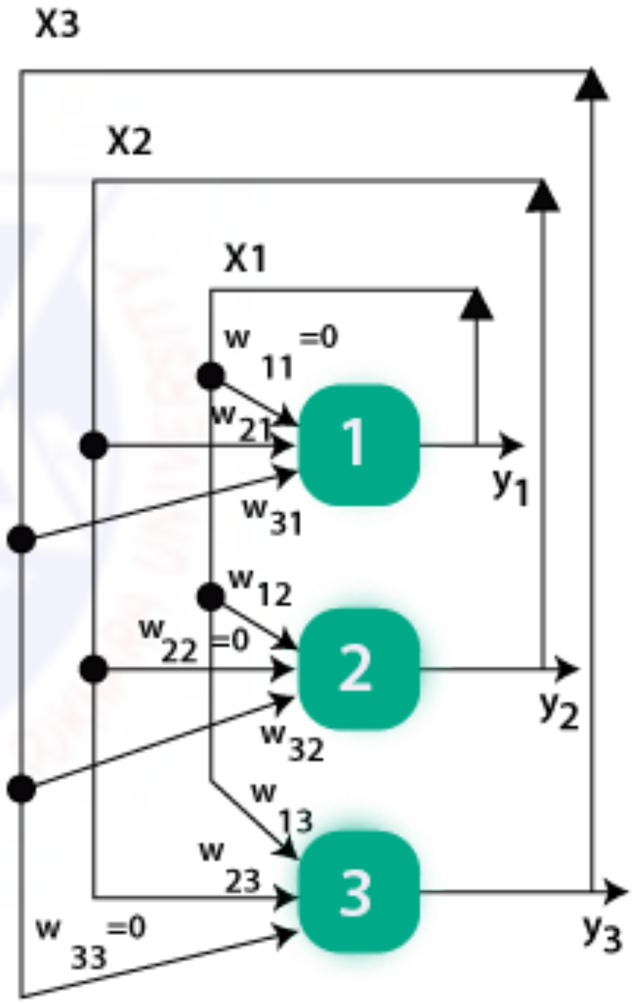
Hopfield Network

- **Hopfield Network is** introduced to collect and retrieve memory like the human brain.
- Here, a neuron is either on or off the situation.
- The state of a neuron(on +1 or off 0) will be restored, relying on the input it receives from the other neuron.
- A Hopfield network is at first prepared to store various patterns or memories.
- Afterward, it is ready to recognize any of the learned patterns by uncovering partial or even some corrupted data about that pattern, i.e., it eventually settles down and restores the closest pattern.
- Thus, similar to the human brain, the Hopfield model has stability in pattern recognition.



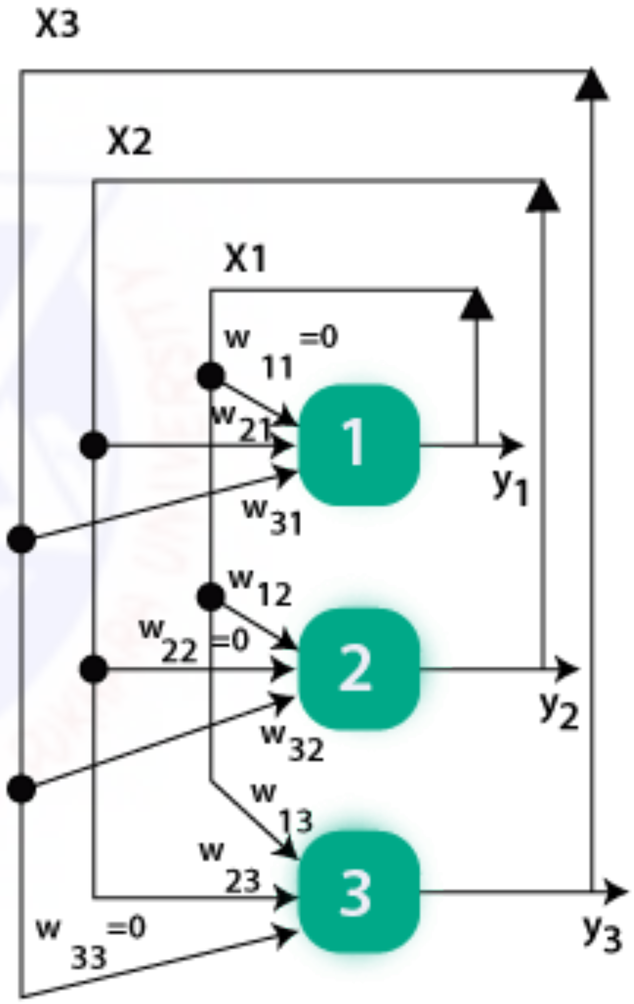
Hopfield Network

- A Hopfield network is a single-layered and recurrent network
- Each neuron is associated with other neurons.
- If there are two neurons i and j, then there is a connectivity weight w_{ij} lies between them which is symmetric $w_{ij} = w_{ji}$.
- There is no self connectivity, $W_{ii} = 0$.
- In figure, the given three neurons having values **i = 1, 2, 3** with values $X_i = \pm 1$ have connectivity weight W_{ij}.



Hopfield Network- Properties

- A recurrent network with all nodes connected to all other nodes.
- Nodes have binary outputs (either binary 0,1 or bipolar -1,1).
- Weights between the nodes are symmetric .
- No connection from a node to itself is allowed.
- Nodes are updated asynchronously (i.e. nodes are selected at 5 random).
- The network has no hidden nodes or layer.



Hopfield Network

- Hopfield neural networks represent a new neural computational paradigm by implementing an autoassociative memory.
- There are two versions of Hopfield neural networks:
  - in the **binary version** all neurons are connected to each other but there is no connection from a neuron to itself, and
  - in the **continuous version** all connections including self-connections are allowed.

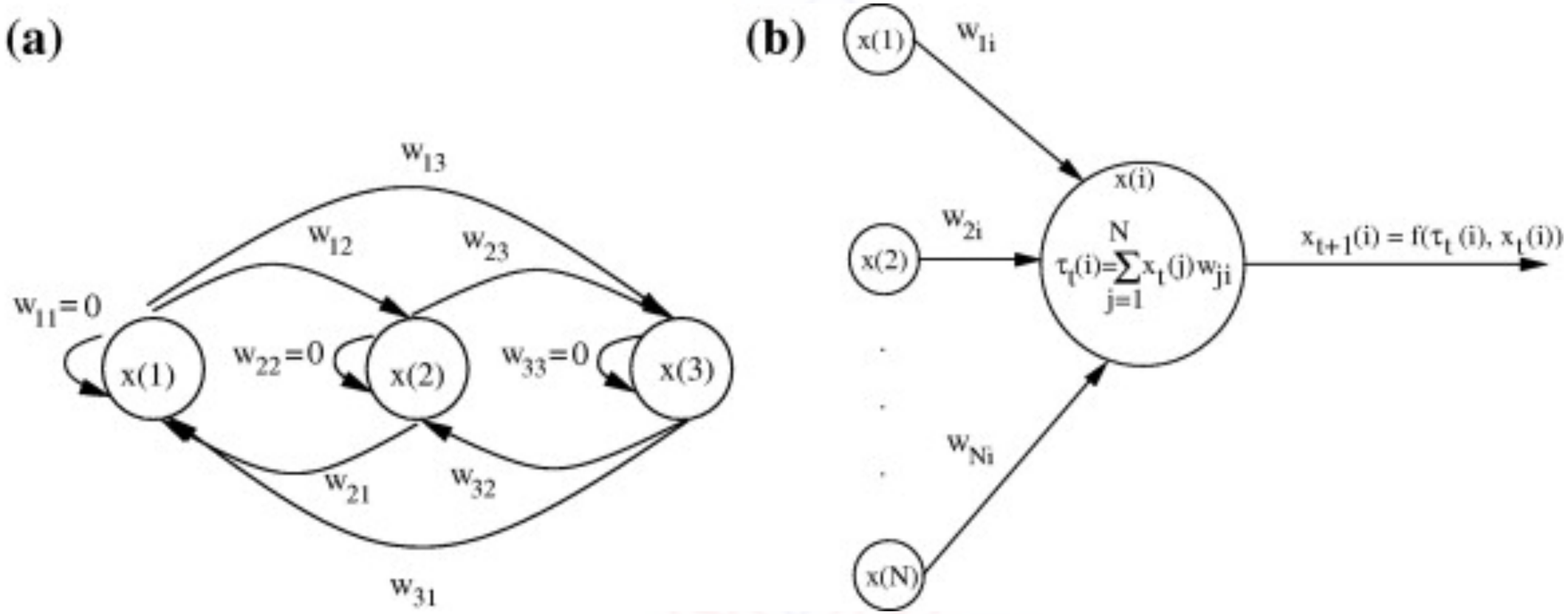
Hopfield Network

- A pattern, in N-node Hopfield neural network parlance, is an N -dimensional vector $p = [p_1, p_2, \ldots, p_N]$ from the space $P = \{-1,1\}^N$.
- A special subset of P represents the set of stored or reference patterns $E = \{e^k : 1 \leq k \leq K\}$, where $e^k = [e_1^k, e_2^K, \ldots, e_N^k]$
- The Hopfield net associates a vector from P with a certain stored (reference) pattern in E .
- The neural net splits the binary space P into classes whose members bear in some way similarity to the reference pattern that represents the class.
- The Hopfield network finds a broad application area in image restoration and segmentation.

Hopfield Network- Structure

- **Neurons:**
  - The Hopfield network has a finite set of neurons $x(i)$ where $1 \leq i \leq N$, which serve as processing units.
  - Each neuron has a value (or state) at time t described by $x_t(i)$
  - A neuron in the Hopfield net has one of the two states, either -1 or $+1$; that is, $x_t(i) \in \{-1, +1\}$
- **Synaptic connections:**
  - The learned information of a neural net resides within the interconnections between its neurons.
  - For each pair of neurons, $x(i)$ and $x(j)$, there is a connection w_{ij} called the synapse between $x(i)$ and $x(j)$.
  - The design of the Hopfield net requires that $w_{ij} = w_{ji}$ and $w_{ii} = 0$.

Hopfield Network



(a) Hopfield neural network and (b) propagation rule and activation function for the Hopfield network.

Hopfield Network- Structure

- **Propagation rule:**
  - This defines how states and synapses influence the input of a neuron.
  - The propagation rule $\tau_t(i)$ is defined by

$$\tau_t(i) = \sum_{j=1}^{N} x_t(j) w_{ij} + b_i$$

  - Where b_i is the externally applied bias to the neuron

Hopfield Network- Structure

- **Activation function:**
  - The activation function f determines the next state of the neuron $x_{t+1}(i)$ based on the value $\tau_t(i)$ computed by the propagation rule and the current value $x_t(i)$.
  - The activation function for the Hopfield net is the hard limiter defined here:

$$\mathbf{x}_{t+1}(i) = f(\tau_t(i), \mathbf{x}_t(i)) = \begin{cases} 1, & \text{if} \quad \tau_t(i) > 0 \\ -1, & \text{if} \quad \tau_t(i) < 0 \end{cases}$$

Hopfield Network- Structure

- **Updating nodes**:
  - Updates in the Hopfield network can be performed in two different ways:
    - **Asynchronous:** Only one unit is updated at a time. This unit can be picked at random, or a pre-defined order can be imposed from the very beginning.
    - **Synchronous:** All units are updated at the same time. This requires a central clock to the system in order to maintain synchronization.

Hopfield Network- Learning

The network learns patterns that are N-dimensional vectors from the space $\mathbf{P} = \{-1,1\}^N$. Let $\mathbf{e^k} = [e_1^k, e_2^k, \ldots, e_n^k]$ define the kth exemplar pattern where $1 \leq k \leq K$. The dimensionality of the pattern space is reflected in the number of nodes in the net, such that the net will have N nodes $\mathbf{x}(1), \mathbf{x}(2), \ldots, \mathbf{x}(N)$.

1. **Learning:** Assign weights w_{ij} to the synaptic connections:

$$w_{ij} = \begin{cases} \sum_{k=1}^{K} e_i^k e_j^k, & \text{if} \quad i \neq j \\ 0, & \text{if} \quad i = j \end{cases}$$

Keep in mind that $w_{ij} = w_{ji}$, so it is necessary to perform the preceding computation only for $i < j$.

Hopfield Network- Learning

2. **Initialization:** Draw an unknown pattern. The pattern to be learned is now presented to the net. If $\mathbf{p} = [p_1, p_2, \ldots, p_N]$ is the unknown pattern, set

$$\mathbf{x}_0(i) = p_i, \quad 1 \leq i \leq N$$

3. **Adaptation:** Iterate until convergence. Using the propagation rule and the activation function we get for the next state,

$$\mathbf{x}_{t+1}(i) = f\left(\sum_{j=1}^{N} \mathbf{x}_t(j) w_{ij}, \mathbf{x}_t(i)\right)$$

This process should be continued until any further iteration will produce no state change at any node.

4. **Continuation:** For learning a new pattern, repeat steps 2 and 3.

Hopfield Network

- In case of the **continuous version** of the Hopfield neural network, we have to
  - consider neural self-connections $w_{ij} \neq 0$ and
  - choose as an activation function a **sigmoid** function.
- With these new adjustments, the training algorithm operates in the same way in the binary version.

Kohonen Network

- Kohonen networks are a type of neural network that perform clustering
- They also known as a **self-organizing map or** self-organizing feature map (SOFM) or Kohonen map.
- A self-organizing map is an unsupervised learning model
- A valuable detail is that the entire learning occurs without supervision i.e. the nodes are *self-organizing*.
- They are also called *feature maps*, as they are essentially retraining the features of the input data, and simply grouping themselves according to the *similarity* between one another.
- The SOM was introduced by the Finnish professor Teuvo Kohonen in the 1982

Kohonen Network

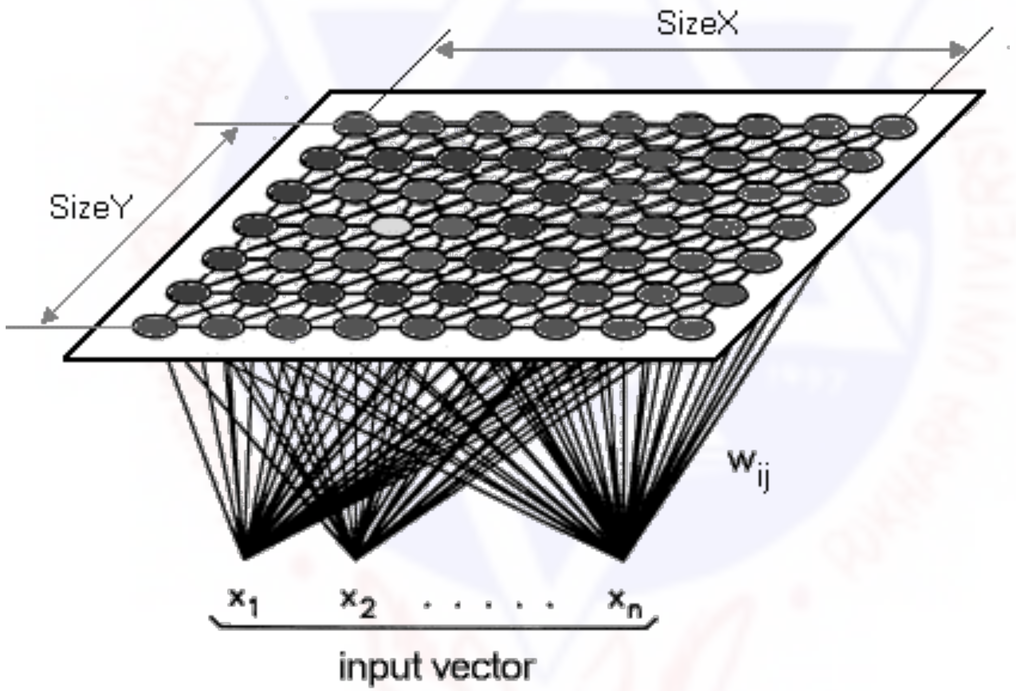
- It is trained using **competitive learning** rather than the error-correction learning (e.g., backpropagation with gradient descent)
- **competitive learning:** The winning output unit is simply the unit with the weight vector that has the smallest Euclidean distance to the input pattern. The competition process suggests that some criteria select a winning processing element.
- This type of network can be used to cluster the dataset into distinct groups when you don't know what those groups are at the beginning.
- Records are grouped so that records within a group or cluster tend to be similar to each other, and records in different groups are dissimilar.

Kohonen Network

- The self-organizing map makes topologically ordered mappings between input data and processing elements of the map.
- Topological ordered implies that if two inputs are of similar characteristics, the most active processing elements answering to inputs that are located closed to each other on the map.
- 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.

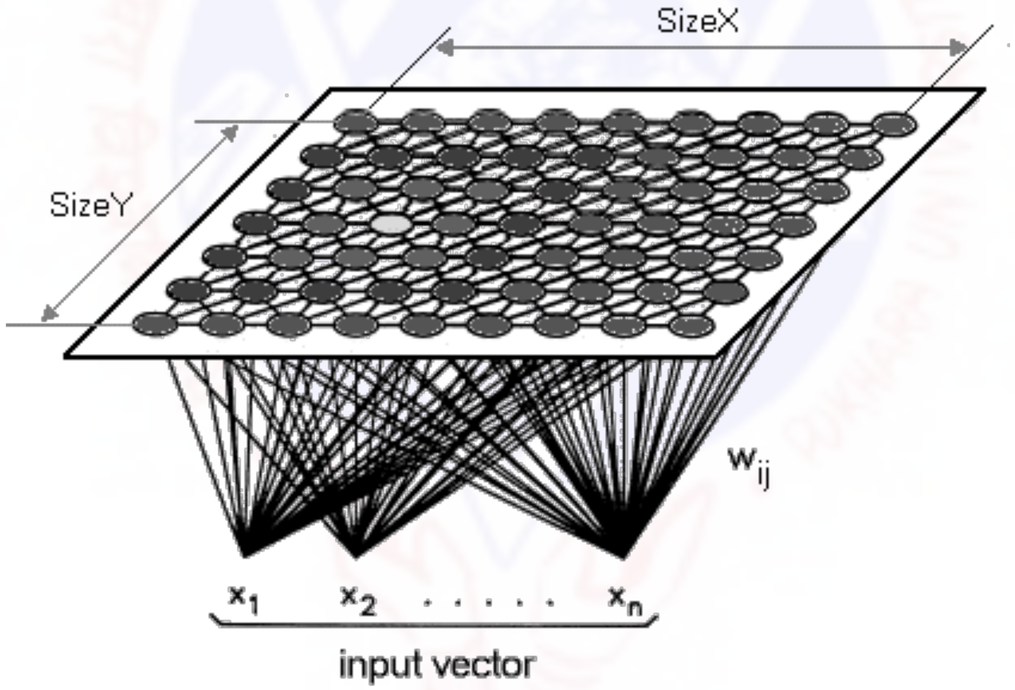
Kohonen Network-Structure

- It comprises of a single-layer linear 2D grid of neurons, instead of a series of layers



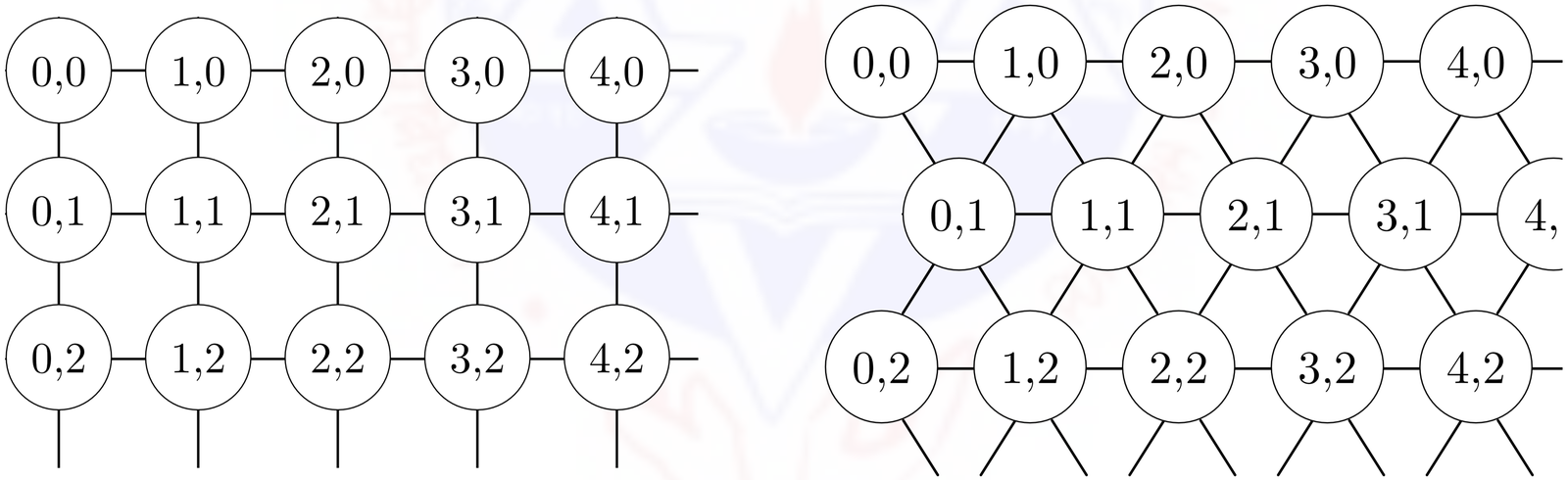
Kohonen Network-Structure

- All the nodes on this grid are connected directly to the input vector, *but not to one another*, meaning the nodes do not know the values of their neighbours and only update the weight of their connections as a function of the given inputs.



Kohonen Network-Structure

- The grid *itself is the map* that organizes itself at each iteration as a function of the input of the input data.
- As such, after clustering, each node has its own *(i,j)* coordinate, which allows one to calculate the Euclidean distance between 2 nodes by means of the Pythagorean theorem

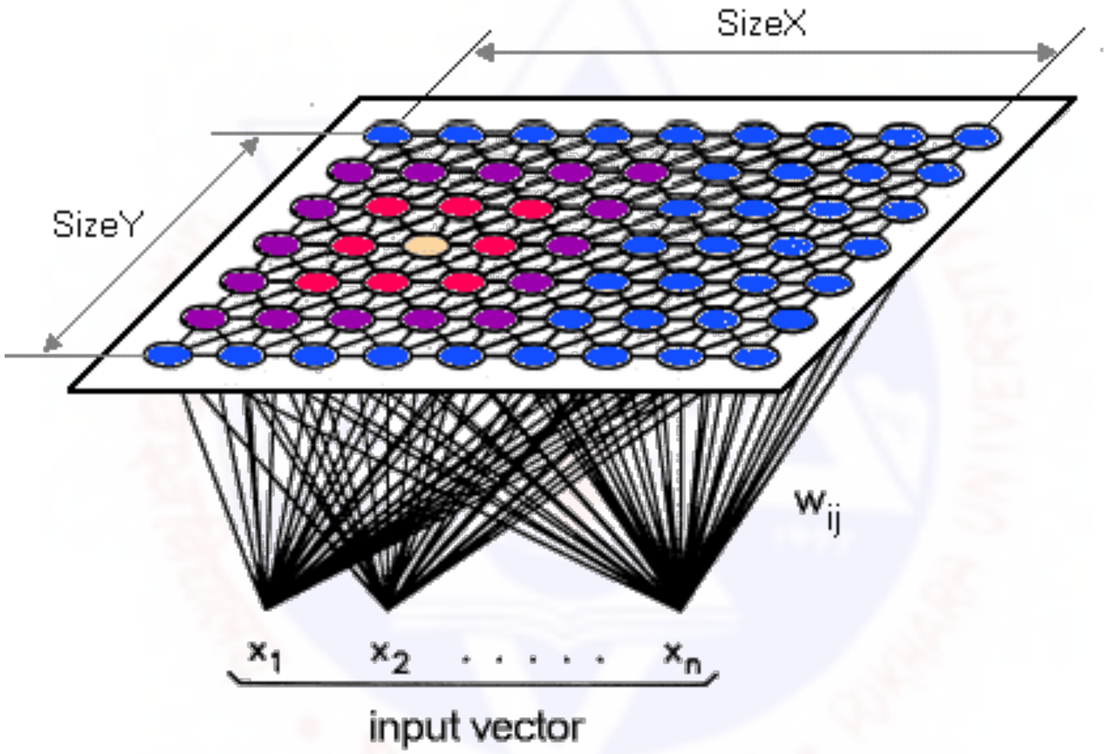


Kohonen network's nodes can be in a rectangular (left) or hexagonal (right) topology.

Kohonen Network-Structure

- A Self-Organizing Map, uses competitive learning to adjust it weights.
  - This means that *only a single node* is activated at each iteration in which the features of an instance of the input vector are presented to the neural network, as all nodes compete for the right to respond to the input.
- The chosen node — the Best Matching Unit (BMU) — is selected according to the similarity, between the current input values and all the nodes in the grid.
- The node with the smallest Euclidean difference between the input vector and all nodes is chosen, *along with its neighbouring nodes* within a certain radius to have their position slightly adjusted to match the input vector.
- By going through all the nodes present on the grid, the entire grid eventually matches the complete input dataset, with similar nodes grouped together towards one area, and dissimilar ones separated.

Kohonen Network-Structure



A Kohonen model with the BMU in yellow, the layers inside the neighbourhood radius in pink and purple, and the nodes outside in blue.

Kohonen Network-Algorithm

Variables Notation:

- t is the current iteration
- n is the iteration limit, i.e. the total number of iterations the network can undergo
- λ is the time constant, used to decay the radius and learning rate
- i is the row coordinate of the nodes grid
- j is the column coordinate of the nodes grid
- d is the distance between a node and the BMU
- w is the weight vector
- $w_{ij}(t)$ is the weight of the connection between the nodes i,j in the grid, and the input vector's instance at the iteration t
- x is the input vector
- x(t) is the input vector's instance at iteration t
- α(t) is the learning rate, decreasing with time in the interval [0,1], to ensure the network converges.
- $\beta_{ij}(t)$ is the neighbourhood function, monotonically decreasing and representing a node i, j's distance from the BMU, and the influence it has on the learning at step t.
- σ(t) is the radius of the neighbourhood function, which determines how far neighbour nodes are examined in the 2D grid when updating vectors. It is gradually reduced over time.

Kohonen Network-Algorithm

Algorithm:

1. Initialize each node's weight w_{ij} to a random value
2. Select a random input vector x_k
3. Repeat point 4. and 5. for all nodes in the map:
4. Compute Euclidean distance between the input vector $x(t)$ and the weight vector w_{ij} associated with the first node, where t, i, j = 0.
5. Track the node that produces the smallest distance t.
6. Find the overall Best Matching Unit (BMU), i.e. the node with the smallest distance from all calculated ones.
7. Determine topological neighbourhood $\beta_{ij}(t)$ its radius $\sigma(t)$ of BMU in the Kohonen Map
8. Repeat for all nodes in the BMU neighbourhood: Update the weight vector w_{ij} of the first node in the neighbourhood of the BMU by adding a fraction of the difference between the input vector $x(t)$ and the weight $w(t)$ of the neuron.
9. Repeat this whole iteration until reaching the chosen iteration limit t=n

Step 1 is the initialization phase, while step 2-9 represent the training phase.

Kohonen Network-Formula

The weights within the neighborhood are updated as:

$$w_{ij}(t+1) = w_{ij}(t) + \alpha_i(t)[x(t) - w_{ij}(t)], \quad \text{or}$$
$$w_{ij}(t+1) = w_{ij}(t) + \alpha_i(t)\beta_{ij}(t)[x(t) - w_{ij}(t)]$$

Where,

$$\sigma(t) = \sigma_0 \cdot \exp(\frac{-t}{\lambda}), \quad \text{where } t = 1, 2, 3 \ldots n$$
$$\alpha(t) = \alpha_0 \cdot \exp(\frac{-t}{\lambda}), \quad \text{where } t = 1, 2, 3 \ldots n$$
$$\beta_{ij}(t) = \exp\left(\frac{-d^2}{2\sigma^2(t)}\right), \quad \text{where } t = 1, 2, 3 \ldots n$$

Kohonen Network-Formula

The Euclidean distance between each node's weight vector and the current input instance is calculated by

$$||\vec{x} - \vec{w_{ij}}|| = \sqrt{\sum_{t=0}^{n} [\vec{x}(t) - \vec{w}_{ij}(t)]^2}$$

The BMU is selected from all the node's calculated distances as the one with the smallest.

$$d = \min(||\vec{x} - \vec{w_{ij}}||) = \min(\sqrt{\sum_{t=0}^{n} [\vec{x}(t) - \vec{w}_{ij}(t)]^2})$$

See at https://towardsdatascience.com/kohonen-self-organizing-maps-a29040d688da

Kohonen Network

- The following points show the main properties which make the Kohonen network useful for clustering:
  1. Low dimension of the network and its simple structure.
  2. Simple representation of clusters by means of vectors associated with each neuron.
  3. Topology of the input data set is somehow mapped in the topology of the weights of the network.
  4. Learning is unsupervised.
  5. Self-organized property.

Elastic Net Model

- A major component of neural networks is *regularization*, adding an additional term to the loss (based on the size of the network's weights) that helps prevent the network from overfitting during training.
- The two most common forms of regularization are Ridge regularization (also known as L1-regularization) and LASSO regularization (also known as L2-regularization). These regularization methods use the L1 and L2 norms, respectively, of the neural network's weights as the regularization term.
- The elastic net is simply a linear combination of Ridge and LASSO regularization. It was created to overcome the limitations of LASSO regularization; namely, the fact that LASSO can be too dependent on the training data (especially data with correlated features), and become unstable.
- Basically, elastic net regularization attempts to find a middle ground between Ridge and LASSO by combining the two.

Elastic Net Model

- **Assignment:** Go through the following links and do the following:
  - What is Elastic Net Model?
  - What is the use of Elastic Net Model? Why do you need it?

- https://www.geeksforgeeks.org/lasso-vs-ridge-vs-elastic-net-ml/
- https://corporatefinanceinstitute.com/resources/knowledge/other/elastic-net/
- https://towardsdatascience.com/preventing-overfitting-with-lasso-ridge-and-elastic-net-regularization-in-machine-learning-d1799b05d382

Boltzmann Machine

- **Assignment:** Write short notes on Boltzmann Machine based on the following articles:
- https://analyticsindiamag.com/beginners-guide-to-boltzmann-machines/
- https://www.cs.toronto.edu/~hinton/csc321/readings/boltz321.pdf

Properties of ANN

Learning from examples
- labeled or unlabeled

Adaptivity
- changing the connection strengths to learn things

Non-linearity
- the non-linear activation functions are essential

Fault tolerance
- if one of the neurons or connections is damaged, the whole network still works quite well

Applications of ANN

Clustering:

A clustering algorithm explores the similarity between patterns and places similar patterns in a cluster. Best known applications include data compression and data mining.

Classification/Pattern recognition:

The task of pattern recognition is to assign an input pattern (like handwritten symbol) to one of many classes. This category includes algorithmic implementations such as associative memory.

Function approximation :

The tasks of function approximation is to find an estimate of the unknown function subject to noise. Various engineering and scientific disciplines require function approximation.

Prediction Systems:

The task is to forecast some future values of a time-sequenced data. Prediction has a significant impact on decision support systems. Prediction differs from function approximation by considering time factor. System may be dynamic and may produce different results for the same input data based on system state (time).

THANK YOU

End of Chapter