3.1 Single Layer Feed-forward Network

The Single Layer Feed-forward Network consists of a single layer of weights, where the inputs are directly connected to the outputs, via a series of weights. The synaptic links carrying weights connect every input to every output, but not other way. This way it is considered a network of **feed-forward** type. The sum of the products of the weights and the inputs is calculated in each neuron node, and 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**).

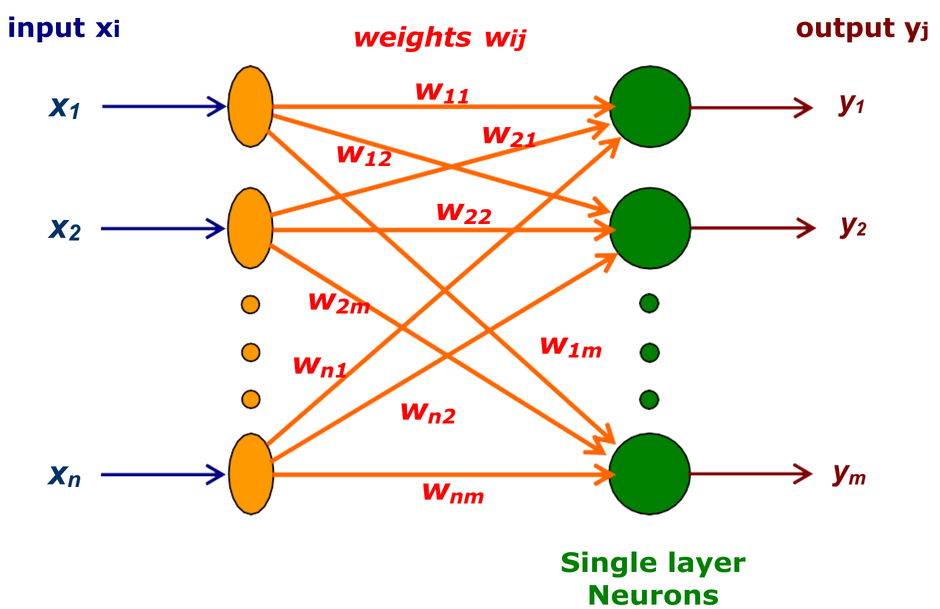


Fig. Single Layer Feed-forward Network

3.2 Multi-Layer Feed-forward Network

The name suggests, it consists of multiple layers. The architecture of this class of network, besides having the input and the output layers, also have one or more intermediary layers called hidden layers. The computational units of the hidden layer are known as hidden neurons.

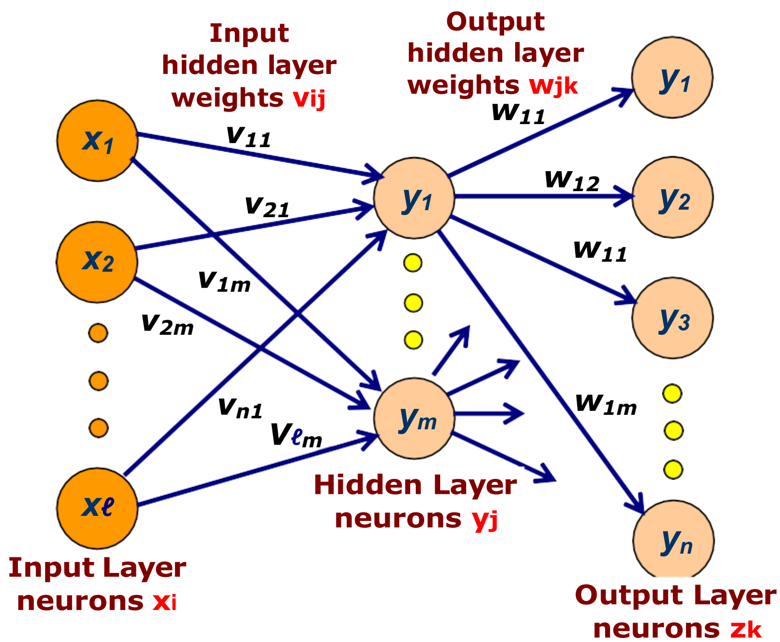


Fig. Multilayer feed-forward network in *(ℓ – m – n)* configuration.

- The hidden layer does intermediate computation before directing the input to output layer.
- The input layer neurons are linked to the hidden layer neurons; the weights on these links are referred to as **input-hidden layer weights**.
- The hidden layer neurons and the corresponding weights are referred to as **output-hidden layer weights**.
- A multi-layer feed-forward network with ℓ input neurons, m_1 neurons in the first hidden layers, m_2 neurons in the second hidden layers, and n output neurons in the output layers is written as **(ℓ - m_1 - m_2 – n).**

The Fig. above illustrates a multilayer feed-forward network with a configuration ***(ℓ - m – n).***

3.3 Recurrent Networks

The Recurrent Networks differ from feed-forward architecture. A Recurrent network has at least one feed-back loop.

Example:

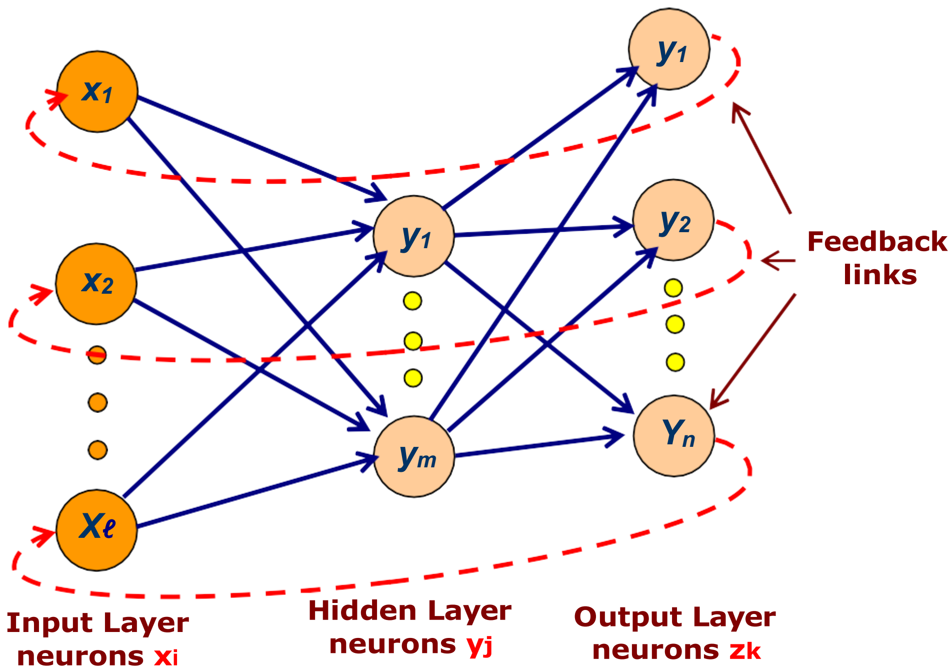


Fig Recurrent Neural Network

There could be neurons with self-feedback links; that is the output of a neuron is fed back into itself as input.

4. Learning Methods in Neural Networks

The learning methods in neural networks are classified into three basic types:

- Supervised Learning,
- Unsupervised Learning and
- Reinforced Learning

These three types are classified based on:

- presence or absence of **teacher** and
- the information provided for the system to learn.

These are further categorized, based on the **rules** used, as

- Hebbian,
- Gradient descent,
- Competitive and
- Stochastic learning.

- **Classification of Learning Algorithms**

  Fig. below indicate the hierarchical representation of the algorithms mentioned in the previous slide. These algorithms are explained in subsequent slides.

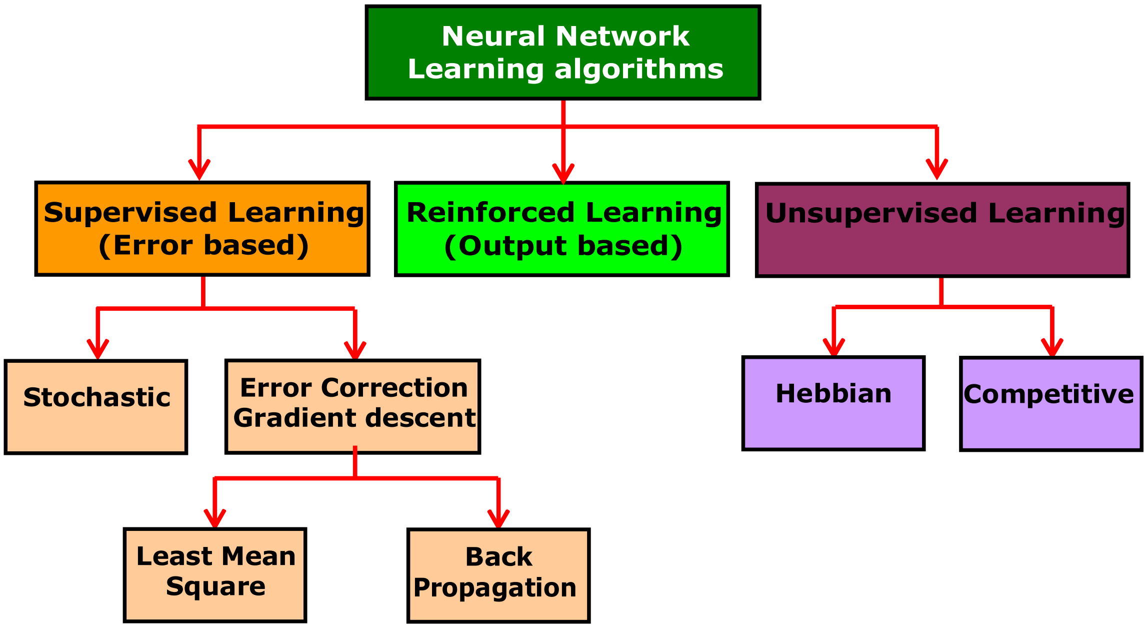


Fig. Classification of learning algorithms

- **Supervised Learning**
  - A teacher is present during learning process and presents expected output.
  - Every input pattern is used to train the network.
  - Learning process is based on comparison, between network's computed output and the correct expected output, generating "error".
  - The "error" generated is used to change network parameters that result improved performance.

- **Unsupervised Learning**
  - No teacher is present.
  - The expected or desired output is not presented to the network.
  - The system learns of it own by discovering and adapting to the structural features in the input patterns.

- **Reinforced learning**
  - A teacher is present but does not present the expected or desired output but only indicated if the computed output is correct or incorrect.
  - The information provided helps the network in its learning process.
  - A reward is given for correct answer computed and a penalty for a wrong answer.

Note: The Supervised and Unsupervised learning methods are most popular forms of learning compared to Reinforced learning.

- **Hebbian Learning**

  Hebb proposed a rule based on correlative weight adjustment.

  In this rule, the input-output pattern pairs (X_i, Y_i) are associated by the weight matrix W, known as correlation matrix, computed as

$$W = \sum_{i=1}^{n} X_i \, Y_i^T$$

  where Y_i^T is the transpose of the associated output vector Y_i

  There are many variations of this rule proposed by the other researchers (Kosko, Anderson, Lippman).

- **Gradient descent Learning**

  This is based on the minimization of errors E defined in terms of weights and the activation function of the network.

  - Here, the activation function of the network is required to be differentiable, because the updates of weight is dependent on the gradient of the error E.

  - If ΔW_{ij} is the weight update of the link connecting the i_{th} and the j_{th} neuron of the two neighboring layers, then ΔW_{ij} is defined as

$$\Delta W_{ij} = \eta \, (\partial E / \partial W_{ij})$$

  where η is the learning rate parameters and $(\partial E / \partial W_{ij})$ is error gradient with reference to the weight W_{ij}.

  Note: The Widrow-Hoff (Delta) rule and Back-propagation learning rule are the examples of Gradient descent learning.

- **Competitive Learning**
  - In this method, those neurons which respond strongly to the input stimuli have their weights updated.
  - When an input pattern is presented, all neurons in the layer compete, and the winning neuron undergoes weight adjustment.
  - This strategy is called "winner-takes-all".

- **Stochastic Learning**
  - In this method the weights are adjusted in a probabilistic fashion.
  - Example : Simulated annealing which is a learning mechanism employed by Boltzmann and Cauchy machines.

5. Taxonomy of Neural Network Systems

In the previous sections, the Neural Network Architectures and the Learning methods have been discussed. Here the popular neural network systems are listed. The grouping of these systems in terms of architectures and the learning methods are presented in the next slide.

- **Neural Network Systems**
  - ADALINE (Adaptive Linear Neural Element)
  - ART (Adaptive Resonance Theory)
  - AM (Associative Memory)
  - BAM (Bidirectional Associative Memory)
  - Boltzmann machines
  - BSB (Brain-State-in-a-Box)
  - Cauchy machines
  - Hopfield Network
  - LVQ (Learning Vector Quantization)
  - Neo-cognition
  - Perceptron
  - RBF (Radial Basis Function)
  - RNN (Recurrent Neural Network)
  - SOFM (Self-organizing Feature Map)

- **Classification of Neural Network**

A taxonomy of neural network systems based on Architectural types and the Learning methods is illustrated below.

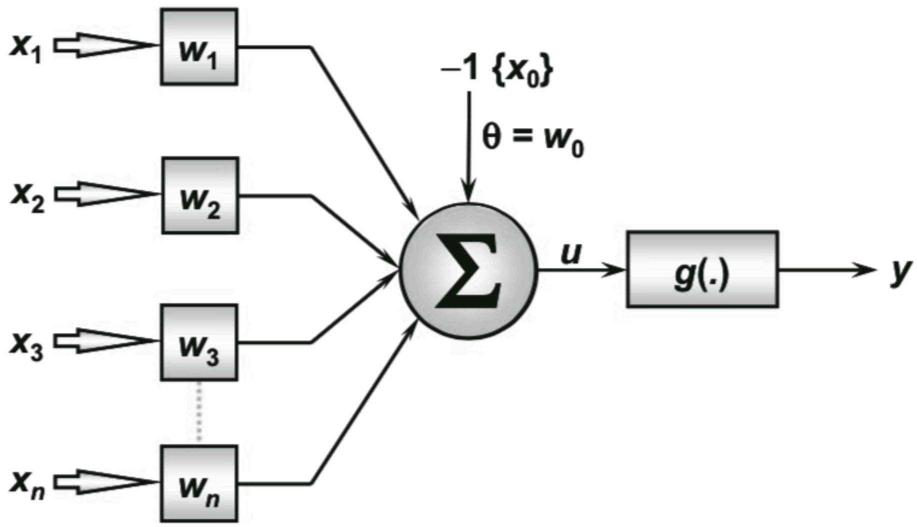
Types of Architecture		Learning Methods: Gradient descent	Hebbian	Competitive	Stochastic
	Single-layer feed-forward	ADALINE, Hopfield, Perceptron,	AM, Hopfield,	LVQ, SOFM	-
	Multi-layer feed- forward	CCM, MLFF, RBF	Neocognition		
	Recurrent Networks	RNN	BAM, BSB, Hopfield,	ART	Boltzmann and Cauchy machines

Table: Classification of Neural Network Systems with respect to learning methods and Architecture types

Single- Layer Perceptron

Perceptron Networks are single-layer feed-forward networks. These are also called Single Perceptron Networks. The Perceptron consists of an input layer, a hidden layer, and output layer.

The input layer is connected to the hidden layer through weights which may be inhibitory or excitery or zero (-1, +1 or 0). The activation function used is a binary step function for the input layer and the hidden layer.



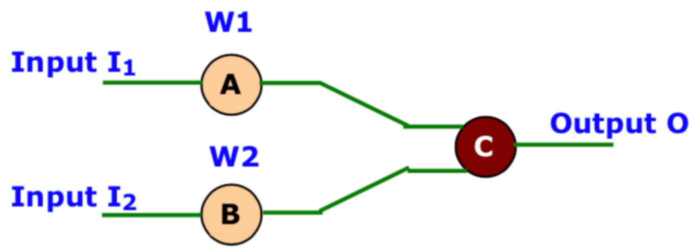
The Perceptron algorithm states that:

Prediction (y`) = 1 if Wx+b > 0 and 0 if Wx+b ≤ 0

AND function

Implementation of AND function in the neural network.

AND		
X1	X2	Y
0	0	0
0	1	0
1	0	0
1	1	1



AND function implementation

- there are 4 inequalities in the AND function and they must be satisfied.

  $w_1 0 + w_2 0 < \theta$, $w_1 0 + w_2 1 < \theta$,

  $w_1 1 + w_2 0 < \theta$, $w_1 1 + w_2 1 > \theta$

- one possible solution :

  if both weights are set to 1 and the threshold is set to 1.5, then

  $(1)(0) + (1)(0) < 1.5$ assign 0 , $(1)(0) + (1)(1) < 1.5$ assign 0

  $(1)(1) + (1)(0) < 1.5$ assign 0 , $(1)(1) + (1)(1) > 1.5$ assign 1

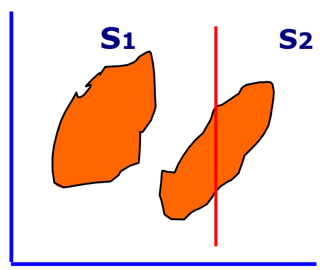
Although it is straightforward to explicitly calculate a solution to the AND function problem, but the question is "how the network can learn such a solution". That is, given random values for the weights can we define an incremental procedure which will cover a set of weights which implements AND function.

- **Perceptron and Linearly Separable Task**

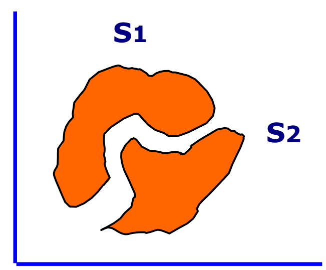
Perceptron cannot handle tasks which are not separable.

- Definition: Sets of points in 2-D space are linearly separable if the sets can be separated by a straight line.
- Generalizing, a set of points in n-dimensional space are linearly separable if there is a hyper plane of (n-1) dimensions separates the sets.

Example



(a) Linearly separable patterns



(b) Not Linearly separable patterns

Note: Perceptron cannot find weights for classification problems that are not linearly separable.

- **XOR Problem:**

Exclusive OR operation

Input x1	Input x2	Output
0	0	0
1	1	0
0	1	1
1	0	1

} Even parity •

} Odd parity °

XOR truth table

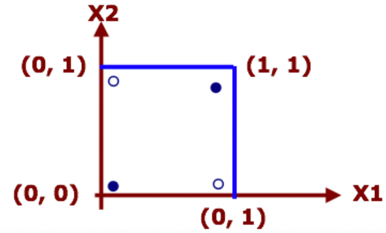


Fig. Output of XOR in X1, x2 plane

Even parity is, even number of 1 bits in the input

Odd parity is, odd number of 1 bits in the input

- There is no way to draw a single straight line so that the circles are on one side of the line and the dots on the other side.
- Perceptron is unable to find a line separating even parity input patterns from odd parity input patterns.