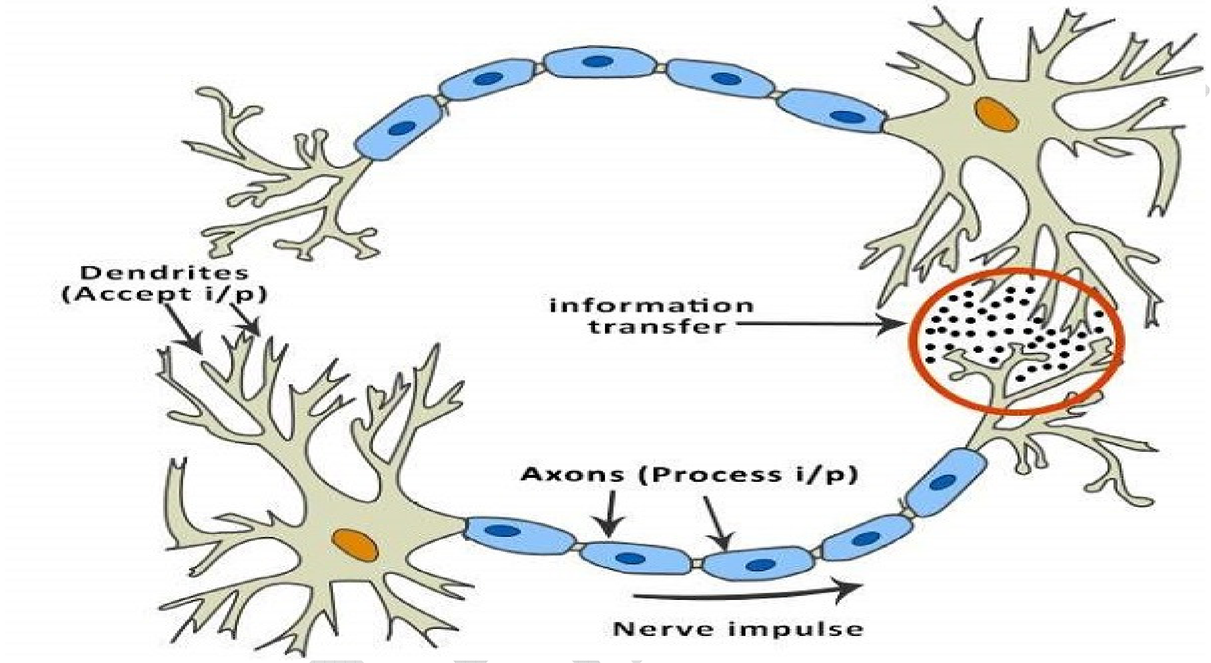
Artificial Neutral Networks

What are Artificial Neural Networks (ANNs)?
The inventor of the first neuro computer, Dr. Robert Hecht-Nielsen, defines a neural network as –
"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."

The idea of ANNs is based on the belief that working of human brain by making the right connections, can be imitated using silicon and wires as living **neurons** and **dendrites**.

The human brain is composed of 86 billion nerve cells called **neurons.** They are connected to other thousand cells by **Axons.** Stimuli from external environment or inputs from sensory organs are accepted by dendrites. These inputs create electric impulses, which quickly travel through the neural network. A neuron can then send the message to other neuron to handle the issue or does not send it forward.



Artificial Neural Network (ANN) is an efficient computing system whose central theme is borrowed from the analogy of biological neural networks. ANNs are also named as "artificial neural systems," or "parallel distributed processing systems," or "connectionist systems." ANN acquires a large collection of units that are interconnected in some pattern to allow communication between the units. These units, also referred to as nodes or neurons, are simple processors which operate in parallel.
Every neuron is connected with other neuron through a connection link. Each connection link is associated with a weight that has information about the input signal. This is the most useful information for neurons to solve a particular problem because the weight usually excites or inhibits the signal that is being communicated. Each neuron has an internal state, which is called an activation signal. Output signals, which are produced after combining the input signals and activation rule, may be sent to other units.

*ANNs are composed of multiple **nodes**, which imitate biological **neurons** of human brain. The neurons are connected by links and they interact with each other. The nodes can take input data and perform simple operations on the data. The result of these operations is passed to other neurons. The output at each node is called its **activation** or **node value.**Each link is associated with **weight.** ANNs are capable of learning, which takes place by altering weight values.*

A Brief History of ANN

The history of ANN can be divided into the following three eras −
ANN during 1940s to 1960s
Some key developments of this era are as follows −

- **1943** − It has been assumed that the concept of neural network started with the work of physiologist, Warren McCulloch, and mathematician, Walter Pitts, when in 1943 they modeled a simple neural network using electrical circuits in order to describe how neurons in the brain might work.
- **1949** − Donald Hebb's book, *The Organization of Behavior*, put forth the fact that repeated activation of one neuron by another increases its strength each time they are used.
- **1956** − An associative memory network was introduced by Taylor.
- **1958** − A learning method for McCulloch and Pitts neuron model named Perceptron was invented by Rosenblatt.
- **1960** − Bernard Widrow and Marcian Hoff developed models called "ADALINE" and "MADALINE."

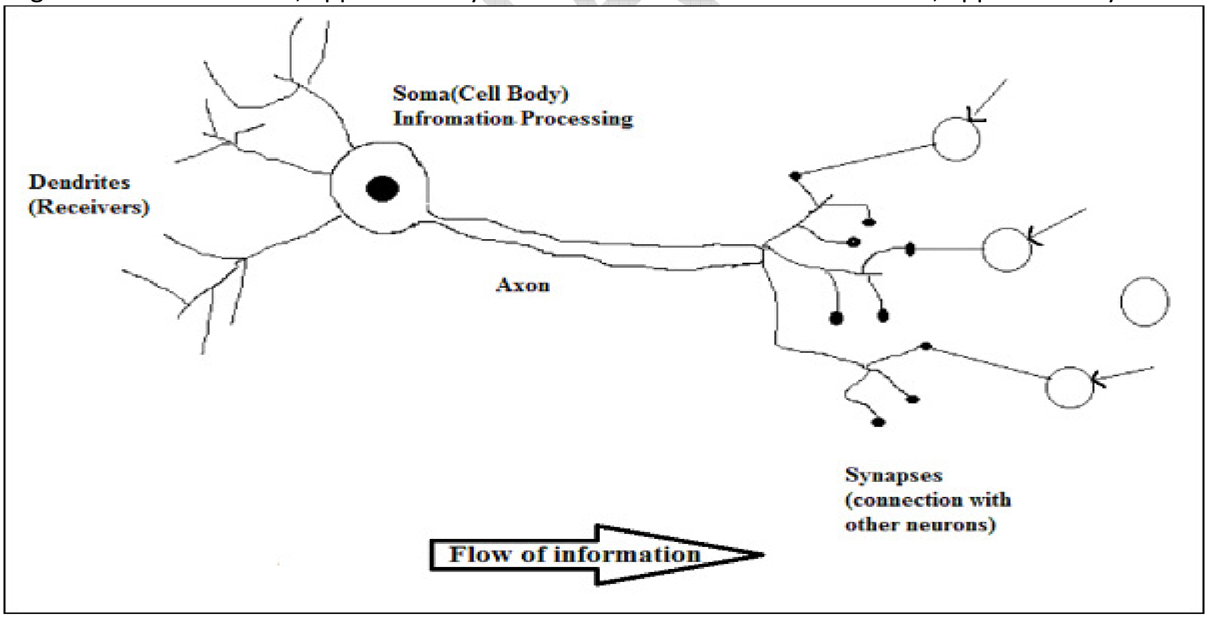
ANN during 1960s to 1980s
Some key developments of this era are as follows −

- **1961** − Rosenblatt made an unsuccessful attempt but proposed the "backpropagation" scheme for multilayer networks.
- **1964** − Taylor constructed a winner-take-all circuit with inhibitions among output units.
- **1969** − Multilayer perceptron (MLP) was invented by Minsky and Papert.
- **1971** − Kohonen developed Associative memories.
- **1976** − Stephen Grossberg and Gail Carpenter developed Adaptive resonance theory.

ANN from 1980s till Present
Some key developments of this era are as follows −

- **1982** − The major development was Hopfield's Energy approach.
- **1985** − Boltzmann machine was developed by Ackley, Hinton, and Sejnowski.
- **1986** − Rumelhart, Hinton, and Williams introduced Generalised Delta Rule.
- **1988** − Kosko developed Binary Associative Memory (BAM) and also gave the concept of Fuzzy Logic in ANN.

The historical review shows that significant progress has been made in this field. Neural network based chips are emerging and applications to complex problems are being developed. Surely, today is a period of transition for neural network technology.
Biological Neuron
A nerve cell (neuron) is a special biological cell that processes information. According to an estimation, there are huge number of neurons, approximately 10^{11} with numerous interconnections, approximately 10^{15}.



As shown in the above diagram, a typical neuron consists of the following four parts with the help of which we can explain its working −

- **Dendrites** − They are tree-like branches, responsible for receiving the information from other neurons it is connected to. In other sense, we can say that they are like the ears of neuron.
- **Soma** − It is the cell body of the neuron and is responsible for processing of information, they have received from dendrites.
- **Axon** − It is just like a cable through which neurons send the information.

- **Synapses** – It is the connection between the axon and other neuron dendrites.

Difference between Biological Neurons and Artificial Neurons?

Before taking a look at the differences between Artificial Neural Network (ANN) and Biological Neural Network (BNN), let us take a look at the similarities based on the terminology between these two.

Biological Neural Network (BNN)	Artificial Neural Network (ANN)
Soma	Node
Dendrites	Input
Synapse	Weights or Interconnections
Axon	Output

Criteria	BNN	ANN
Processing	Massively parallel, slow but superior than ANN	Massively parallel, fast but inferior than BNN
Size	10^{11} neurons and 10^{15} interconnections	10^{2} to 10^{4} nodes (mainly depends on the type of application and network designer)
Learning	They can tolerate ambiguity	Very precise, structured and formatted data is required to tolerate ambiguity
Fault tolerance	Performance degrades with even partial damage	It is capable of robust performance, hence has the potential to be fault tolerant
Storage capacity	Stores the information in the synapse	Stores the information in continuous memory locations

BIOLOGICAL NEURONS	ARTIFICIAL NEURONS
Major components: Axions, Dendrites, Synapse	Major Components: Nodes, Inputs, Outputs, Weights, Bias
Information from other neurons, in the form of electrical impulses, enters the dendrites at connection points called synapses. The information flows from the dendrites to the cell where it is processed. The output signal, a train of impulses, is then sent down the axon to the synapse of other neurons.	The arrangements and connections of the neurons made up the network and have three layers. The first layer is called the input layer and is the only layer exposed to external signals. The input layer transmits signals to the neurons in the next layer, which is called a hidden layer. The hidden layer extracts relevant features or patterns from the received signals. Those features or patterns that are considered important are then directed to the output layer, which is the final layer of the network.

A synapse is able to increase or decrease the strength of the connection. This is where information is stored.	The artificial signals can be changed by weights in a manner similar to the physical changes that occur in the synapses.
Approx 10^{11} neurons.	10^{2}– 10^{4} neurons with current technology

Difference between the human brain and computers in terms of how information is processed.

HUMAN BRAIN(BIOLOGICAL NEURON NETWORK)	COMPUTERS(ARTIFICIAL NEURON NETWORK)
The human brain works asynchronously	Computers(ANN) work synchronously.
Biological Neurons compute slowly (several ms per computation)	Artificial Neurons compute fast (<1 nanosecond per computation)
The brain represents information in a distributed way because neurons are unreliable and could die any time.	In computer programs every bit has to function as intended otherwise these programs would crash.
Our brain changes their connectivity over time to represents new information and requirements imposed on us.	The connectivity between the electronic components in a computer never change unless we replace its components.
Biological neural networks have complicated topologies.	ANNs are often in a tree structure.
Researchers are still to find out how the brain actually learns.	ANNs use Gradient Descent for learning.

Advantage of Using Artificial Neural Networks:

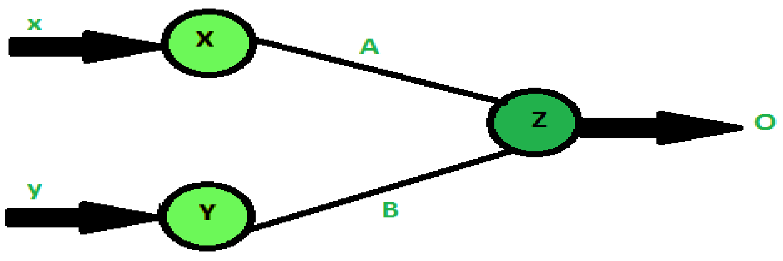
- Problem in ANNs can have instances that are represented by many attribute-value pairs.
- ANNs used for problems having the target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes.
- ANN learning methods are quite robust to noise in the training data. The training examples may contain errors, which do not affect the final output.
- It is used generally used where the fast evaluation of the learned target function may be required.
- ANNs can bear long training times depending on factors such as the number of weights in the network, the number of training examples considered, and the settings of various learning algorithm parameters.

Characteristics of Artificial Neural Network

- It is neurally implemented mathematical model
- It contains huge number of interconnected processing elements called neurons to do all operations
- Information stored in the neurons are basically the weighted linkage of neurons
- The input signals arrive at the processing elements through connections and connecting weights.
- It has the ability to learn , recall and generalize from the given data by suitable assignment and adjustment of weights.
- The collective behavior of the neurons describes its computational power, and no single neuron carries specific information .

How simple neuron works ?

Let there are two neurons **X** and **Y** which is transmitting signal to another neuron **Z** . Then , **X** and **Y** are input neurons for transmitting signals and **Z** is output neuron for receiving signal . The input neurons are connected to the output neuron , over a interconnection links (**A** and **B**) as shown in figure .



Architecture of a Simple Artificial Neuron Net

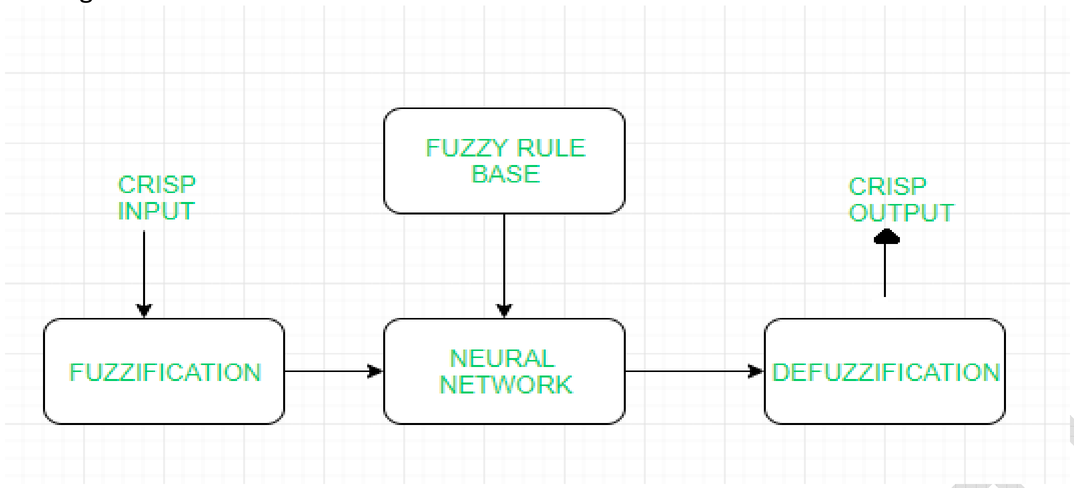
For above neuron architecture , the net input has to be calculated in the way .
I = xA + yB
where x and y are the activations of the input neurons X and Y . The output z of the output neuron Z can be obtained by applying activations over the net input .
O = f(I)
Output = Function (net input calculated)
The function to be applied over the net input is called *activation function* . There are various activation function possible for this.

Application of Neural Network
1. Every new technology need assistance from previous one i.e. data from previous ones and these data are analyzed so that every pros and cons should be studied correctly . All of these things are possible only through the help of neural network.
2. Neural network is suitable for the research on *Animal behavior, predator/prey relationships and population cycles* .
3. It would be easier to do *proper valuation* of property, buildings, automobiles, machinery etc. with the help of neural network.
4. Neural Network can be used in betting on horse races, sporting events and most importantly in stock market .
5. It can be used to predict the correct judgement for any crime by using a large data of crime details as input and the resulting sentences as output.
6. By analyzing data and determining which of the data has any fault (files diverging from peers) called as *Data mining, cleaning and validation* can be achieved through neural network.
7. Neural Network can be used to predict targets with the help of echo patterns we get from sonar, radar, seismic and magnetic instruments .
8. It can be used efficiently in *Employee hiring* so that any company can hire right employee depending upon the skills the employee has and what should be it's productivity in future .
9. It has a large application in *Medical Research* .
10. It can be used to for *Fraud Detection* regarding credit cards , insurance or taxes by analyzing the past records .

Hybrid systems: A Hybrid system is an intelligent system which is framed by combining atleast two intelligent technologies like Fuzzy Logic, Neural networks, Genetic algorithm, reinforcement Learning, etc. The combination of different techniques in one computational model make these systems possess an extended range of capabilities. These systems are capable of reasoning and learning in an uncertain and imprecise environment. These systems can provide human-like expertise like domain knowledge, adaptation in noisy environment etc.
Types of Hybrid Systems:
- Neuro Fuzzy Hybrid systems
- Neuro Genetic Hybrid systems
- Fuzzy Genetic Hybrid systems

(A) Neuro Fuzzy Hybrid systems:
Neuro fuzzy system is based on fuzzy system which is trained on the basis of working of neural network theory. The learning process operates only on the local information and causes only local changes in the underlying fuzzy system. A neuro-fuzzy system can be seen as a 3-layer feedforward neural network. The first layer represents input variables, the middle (hidden) layer represents fuzzy rules and the third layer represents output variables. Fuzzy sets are

encoded as connection weights within the layers of the network, which provides functionality in processing and training the model.



Working flow:

- In input layer, each neuron transmits external crisp signals directly to the next layer.
- Each fuzzification neuron receives a crisp input and determines the degree to which the input belongs to input fuzzy set.
- Fuzzy rule layer receives neurons that represent fuzzy sets.
- An output neuron, combines all inputs using fuzzy operation UNION.
- Each defuzzification neuron represents single output of neuro-fuzzy system.

Advantages:

- It can handle numeric, linguistic, logic, etc kind of information.
- It can manage imprecise, partial, vague or imperfect information.
- It can resolve conflicts by collaboration and aggregation.
- It has self-learning, self-organizing and self-tuning capabilities.
- It can mimic human decision-making process.

Disadvantages:

- Hard to develop a model from a fuzzy system
- Problems of finding suitable membership values for fuzzy systems
- Neural networks cannot be used if training data is not available.

Applications:

- Student Modelling
- Medical systems
- Traffic control systems
- Forecasting and predictions

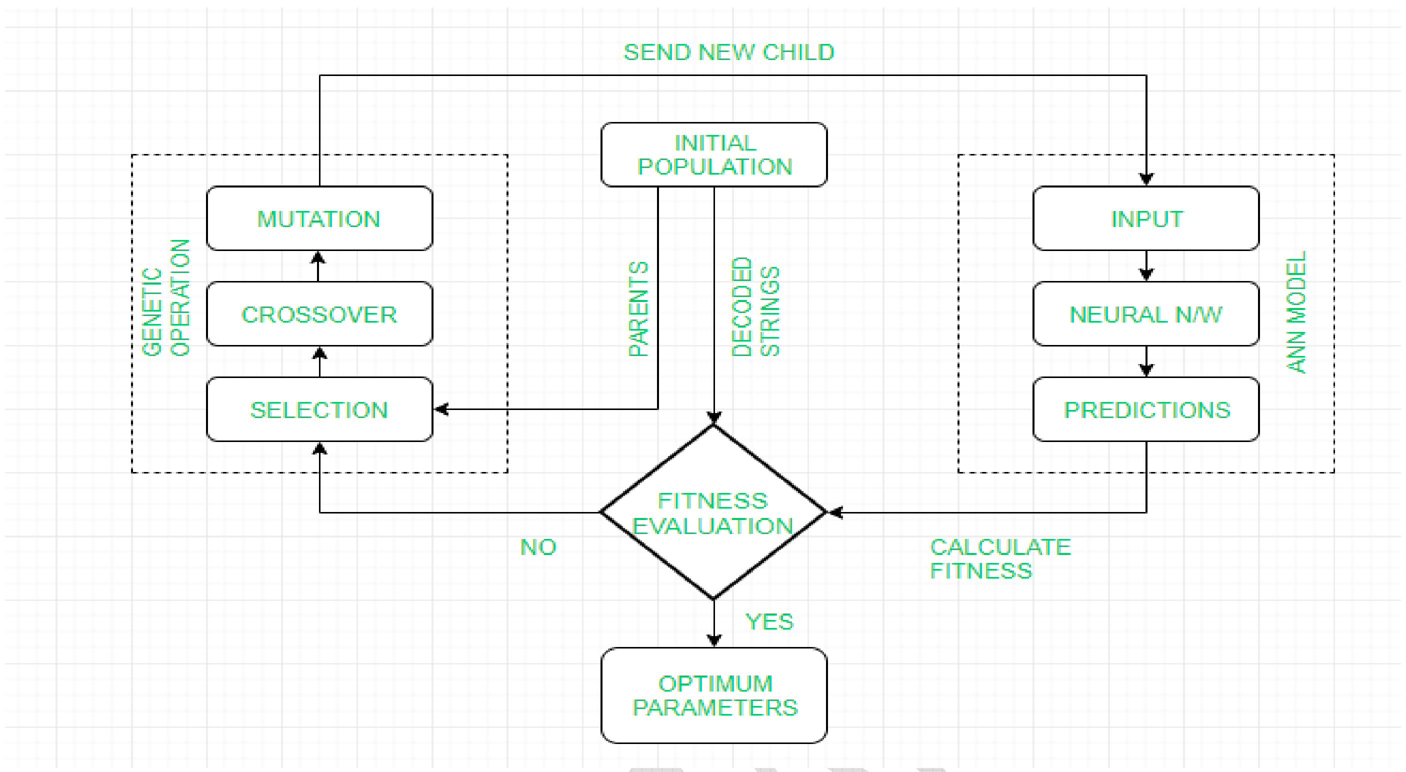
(B) Neuro Genetic Hybrid systems:

A Neuro Genetic hybrid system is a system that combines **Neural networks**: which are capable to learn various tasks from examples, classify objects and establish relation between them and **Genetic algorithm**: which serves important search and optimization techniques. Genetic algorithms can be used to improve the performance of Neural Networks and they can be used to decide the connection weights of the inputs. These algorithms can also be used for topology selection and training network.

Working Flow:

- GA repeatedly modifies a population of individual solutions. GA uses three main types of rules at each step to create the next generation from the current population:
    1. **Selection** to select the individuals, called parents, that contribute to the population at the next generation
    2. **Crossover** to combine two parents to form children for the next generation
    3. **Mutation** to apply random changes to individual parents in order to form children

- GA then sends the new child generation to ANN model as new input parameter.
- Finally, calculating of the fitness by developed ANN model is performed.



Advantages:

- GA is used for topology optimization i.e to select number of hidden layers, number of hidden nodes and interconnection pattern for ANN.
- In GAs, the learning of ANN is formulated as a weight optimization problem, usually using the inverse mean squared error as a fitness measure.
- Control parameters such as learning rate, momentum rate, tolerance level, etc are also optimized using GA.
- It can mimic human decision-making process.

Disadvantages:

- Highly complex system.
- Accuracy of the system is dependent on the initial population.
- Maintaintainance costs are very high.

Applications:

- Face recognition
- DNA matching
- Animal and human research
- Behavioral system

(C) Fuzzy Genetic Hybrid systems:

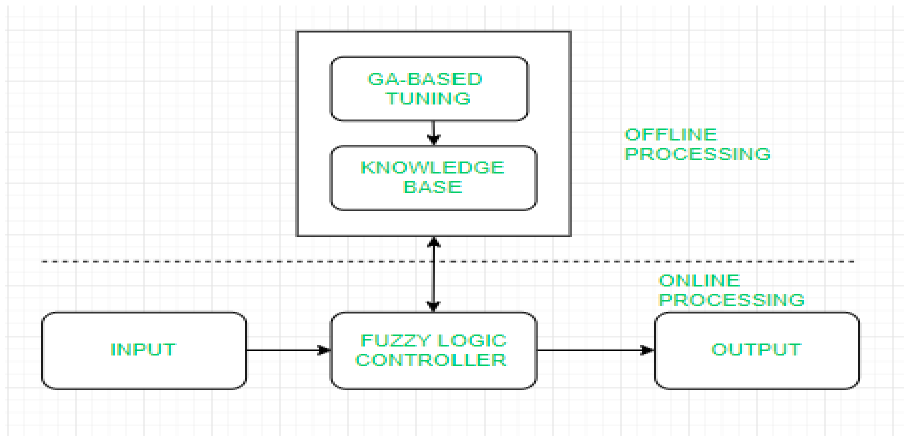
A Fuzzy Genetic Hybrid System is developed to use fuzzy logic based techniques for improving and modelling Genetic algorithms and vice-versa. Genetic algorithm has proved to be a robust and efficient tool to perform tasks like generation of fuzzy rule base, generation of membership function etc.
Three approaches that can be used to develop such system are:

- Michigan Approach
- Pittsburgh Approach
- IRL Approach

Working Flow:

- Start with an initial population of solutions that represent first generation.
- Feed each chromosome from the population into the Fuzzy logic controller and compute performance index.

- Create new generation using evolution operators till some condition is met.



Advantages:
- GAs are used to develop the best set of rules to be used by a fuzzy inference engine
- GAs are used to optimize the choice of membership functions.
- A Fuzzy GA is a directed random search over all discrete fuzzy subsets.
- It can mimic human decision-making process.

Disadvantages:
- Interpretation of results is difficult.
- Difficult to build membership values and rules.
- Takes lots of time to converge.

Applications:
- Mechanical Engineering
- Electrical Engine
- Artificial Intelligence
- Economics

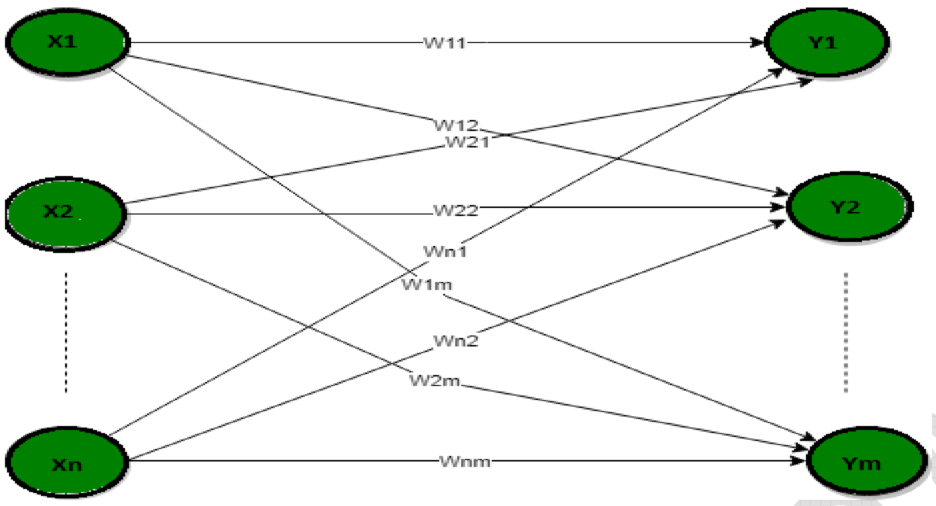
Network Architectures

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired from the brain. ANNs, like people, learn by examples. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning largely involves adjustments to the synaptic connections that exist between the neurons.
The model of Artificial neural network which can be specified by three entities:
- **Interconnections**
- **Activation functions**
- **Learning rules**

Interconnections:
Interconnection can be defined as the way processing elements (Neuron) in ANN are connected to each other. Hence, the arrangements of these processing elements and geometry of interconnections are very essential in ANN.
These arrangements always have two layers which are common to all network architectures, Input layer and output layer where input layer buffers the input signal and output layer generates the output of the network. The third layer is the Hidden layer, in which neurons are neither kept in the input layer nor in the output layer. These neurons are hidden from the people who are interfacing with the system and acts as a blackbox to them. On increasing the hidden layers with neurons, the system's computational and processing power can be increased but the training phenomena of the system gets more complex at the same time.
There exist five basic types of neuron connection architecture:
1. Single-layer feed forward network
2. Multilayer feed forward network

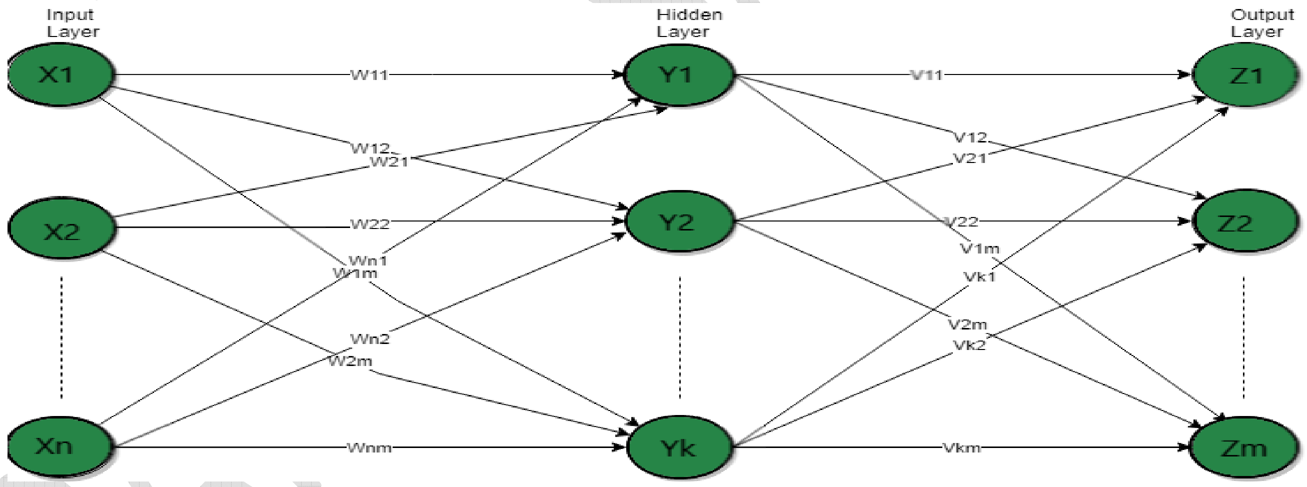
3. Single node with its own feedback
4. Single-layer recurrent network
5. Multilayer recurrent network

1. **Single-layer feed forward network**



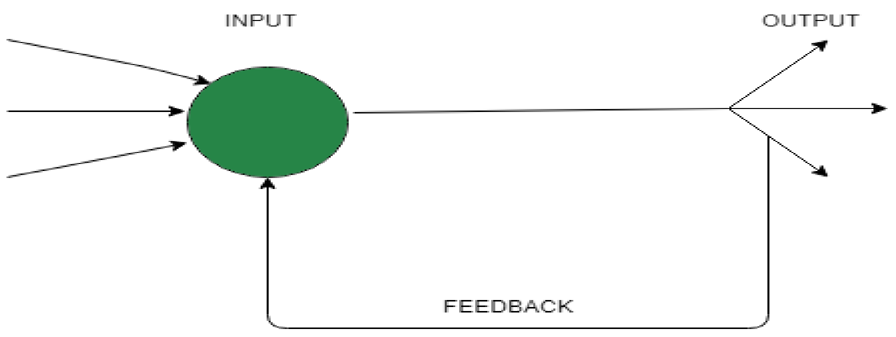
In this type of network, we have only two layers input layer and output layer but input layer does not count because no computation performed in this layer. Output layer is formed when different weights are applied on input nodes and the cumulative effect per node is taken. After this the neurons collectively give the output layer compute the output signals.

2. **Multilayer feed forward network**



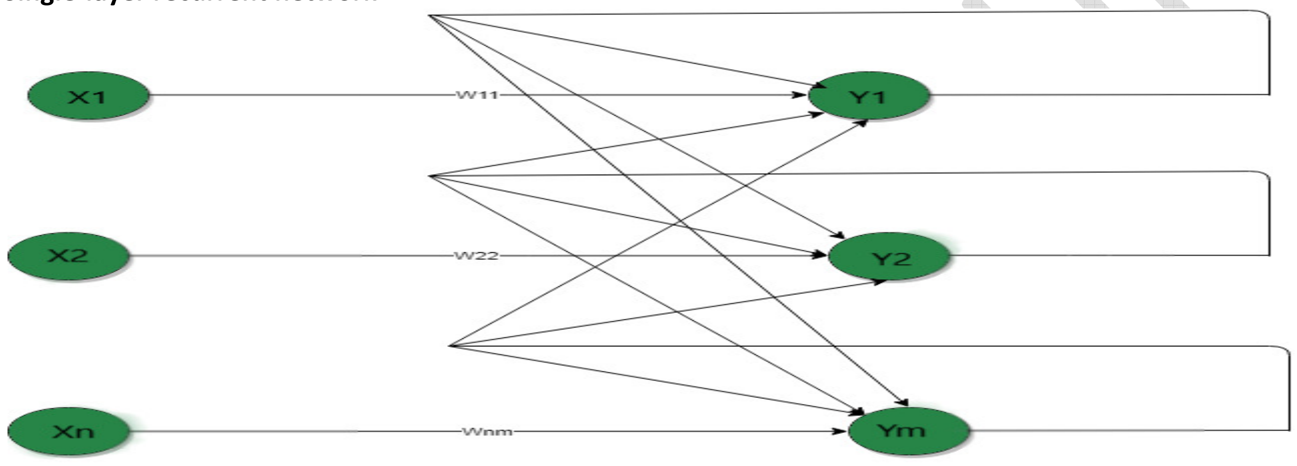
This layer also has hidden layer which is internal to the network and has no direct contact with the external layer. Existence of one or more hidden layers enable the network to be computationally stronger, feed-forward network because information ?ows through the input function, and the intermediate computations used to de?ne the output Z. There are no feedback connections in which outputs of the model are fed back into itself.

3. **Single node with its own feedback**



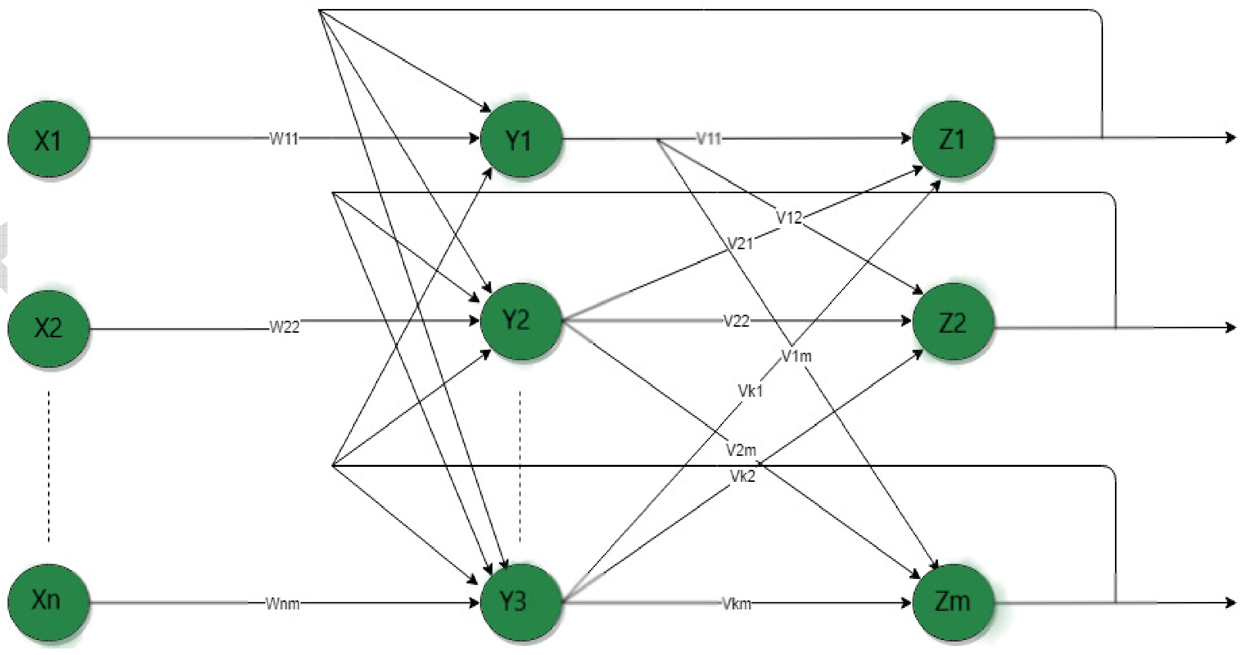
When outputs can be directed back as inputs to the same layer or preceeding layer nodes, then it results in feedback networks. Recurrent networks are feedback networks with closed loop. Above figure shows a single recurrent network having single neuron with feedback to itself.

4. **Single-layer recurrent network**



Above network is single layer network with feedback connection in which processing element's output can be directed back to itself or to other processing element or both. Recurrent neural network is a class of artificial neural network where connections between nodes form a directed graph along a sequence. This allows it to exhibit dynamic temporal behavior for a time sequence. Unlike feed forward neural networks, RNNs can use their internal state (memory) to process sequences of inputs.

5. **Multilayer recurrent network**



In this type of network, processing element output can be directed to the processing element in the same layer and in the preceding layer forming a multilayer recurrent network. They perform the same task for every element of a sequence, with the output being depended on the previous computations. Inputs are not needed at each time step. The main feature of an Recurrent Neural Network is its hidden state, which captures some information about a sequence.

Activation functions in Neural Networks

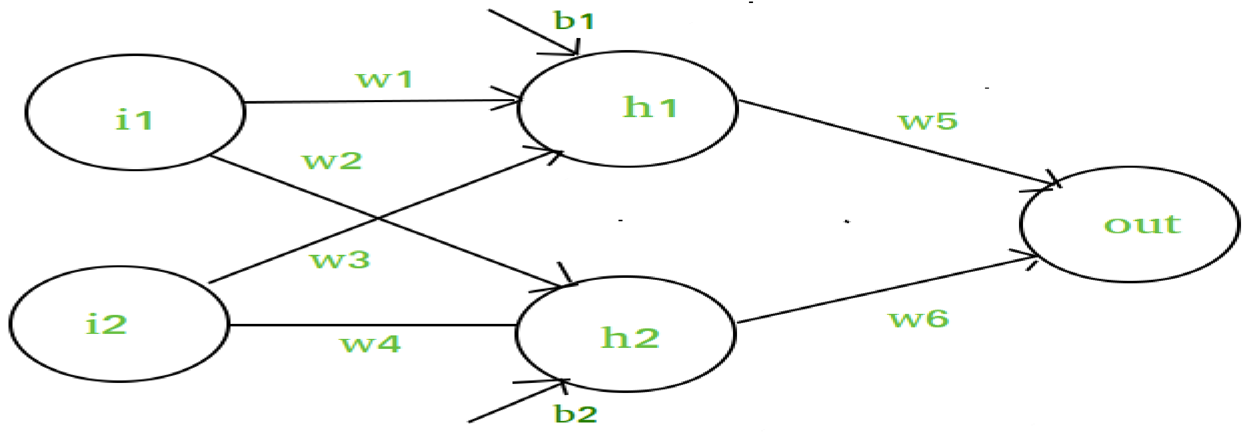
Elements of a Neural Network :-

Input Layer :- This layer accepts input features. It provides information from the outside world to the network, no computation is performed at this layer, nodes here just pass on the information(features) to the hidden layer.
Hidden Layer :- Nodes of this layer are not exposed to the outer world, they are the part of the abstraction provided by any neural network. Hidden layer performs all sort of computation on the features entered through the input layer and transfer the result to the output layer.
Output Layer :- This layer bring up the information learned by the network to the outer world.

What is an activation function and why to use them?
Definition of activation function:- Activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it. The purpose of the activation function is to **introduce non-linearity** into the output of a neuron.

Explanation :-
We know, neural network has neurons that work in correspondence of *weight, bias* and their respective activation function. In a neural network, we would update the weights and biases of the neurons on the basis of the error at the output. This process is known as *back-propagation*. Activation functions make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases.

Why do we need Non-linear activation functions :-
A neural network without an activation function is essentially just a linear regression model. The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.
Mathematical proof :- *Suppose we have a Neural net like this :-*



Elements of the diagram :-
Hidden layer i.e. layer 1 :-
z(1) = W(1)X + b(1)
a(1) = z(1)
Here,

- *z(1) is the vectorized output of layer 1*
- *W(1) be the vectorized weights assigned to neurons of hidden layer i.e. w1, w2, w3 and w4*
- *X be the vectorized input features i.e. i1 and i2*
- *b is the vectorized bias assigned to neurons in hidden layer i.e. b1 and b2*
- *a(1) is the vectorized form of any linear function.*

*(**Note:** We are not considering activation function here)*

Layer 2 i.e. output layer :-
// **Note :** Input for layer
// 2 is output from layer 1
z(2) = W(2)a(1) + b(2)
a(2) = z(2)

Calculation at Output layer:
// Putting value of z(1) here

z(2) = (W(2) * [W(1)X + b(1)]) + b(2)

z(2) = [W(2) * W(1)] * X + [W(2)*b(1) + b(2)]

Let,
[W(2) * W(1)] = W
[W(2)*b(1) + b(2)] = b

Final output : z(2) = W*X + b
Which is again a linear function

This observation results again in a linear function even after applying a hidden layer, hence we can conclude that, doesn't matter how many hidden layer we attach in neural net, all layers will behave same way because ***the composition of two linear function is a linear function itself***. Neuron can not learn with just a linear function attached to it. A non-linear activation function will let it learn as per the difference w.r.t error.
Hence we need activation function.

VARIANTS OF ACTIVATION FUNCTION :-
1). Linear Function :-
- **Equation :** Linear function has the equation similar to as of a straight line i.e. **y = ax**
- No matter how many layers we have, if all are linear in nature, the final activation function of last layer is nothing but just a linear function of the input of first layer.
- **Range :** -inf to +inf
- **Uses : Linear activation function** is used at just one place i.e. output layer.
- **Issues :** If we will differentiate linear function to bring non-linearity, result will no more depend on *input "x"* and function will become constant, it won't introduce any ground-breaking behavior to our algorithm.

For example : Calculation of price of a house is a regression problem. House price may have any big/small value, so we can apply linear activation at output layer. Even in this case neural net must have any non-linear function at hidden layers.

2). Sigmoid Function :-
- It is a function which is plotted as **'S'** shaped graph.
- **Equation :**
  $A = 1/(1 + e^{-x})$
- **Nature :** Non-linear. Notice that X values lies between -2 to 2, Y values are very steep. This means, small changes in x would also bring about large changes in the value of Y.
- **Value Range :** 0 to 1
- **Uses :** Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easily to be ***1*** if value is greater than **0.5** and ***0*** otherwise.

3). Tanh Function :- The activation that works almost always better than sigmoid function is Tanh function also knows as Tangent Hyperbolic function. It's actually mathematically shifted version of the sigmoid function. Both are similar and can be derived from each other.

Equation :-

f(x) = tanh(x) = 2/(1 + e-2x) - 1
OR
tanh(x) = 2 * sigmoid(2x) - 1

- Value Range :- -1 to +1
- Nature :- non-linear
- Uses :- Usually used in hidden layers of a neural network as it's values lies between -1 to 1 hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in centering the data by bringing mean close to 0. This makes learning for the next layer much easier.

4).RELU :- Stands for *Rectified linear unit*. It is the most widely used activation function. Chiefly implemented in *hidden layers* of Neural network.

- **Equation :-** ***A(x) = max(0,x)***. It gives an output x if x is positive and 0 otherwise.
- **Value Range :-** [0, inf)
- **Nature :-** non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.
- **Uses :-** ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.

In simple words, RELU learns *much faster* than sigmoid and Tanh function.

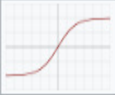
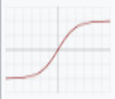
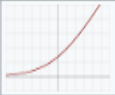
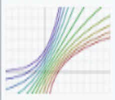
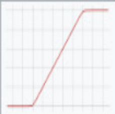
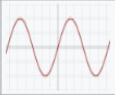
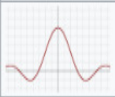
5). Softmax Function :- The softmax function is also a type of sigmoid function but is handy when we are trying to handle classification problems.

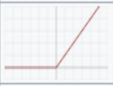
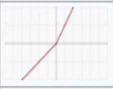
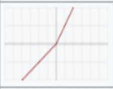
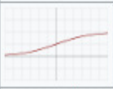
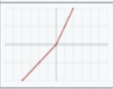
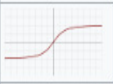
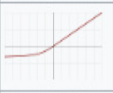
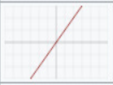
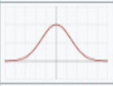
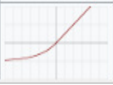
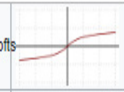
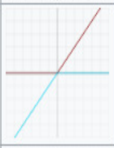
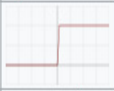
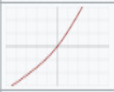
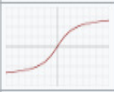
- **Nature :-** non-linear
- **Uses :-** Usually used when trying to handle multiple classes. The softmax function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs.
- **Ouput:-** The softmax function is ideally used in the output layer of the classifier where we are actually trying to attain the probabilities to define the class of each input.

CHOOSING THE RIGHT ACTIVATION FUNCTION

- The basic rule of thumb is if you really don't know what activation function to use, then simply use *RELU* as it is a general activation function and is used in most cases these days.
- If your output is for binary classification then, *sigmoid function* is very natural choice for output layer.

Foot Note :-
The **activation function** does the non-linear transformation to the input making it capable to learn and perform more complex tasks.

Name	Plot	Equation	Derivative (with respect to *x*)	Range	Order of continuity	Monotonic	Monotonic derivative	Approximates identity near the origin
TanH		$f(x)=\tanh(x)=\frac{(e^x-e^{-x})}{(e^x+e^{-x})}$	$f'(x)=1-f(x)^2$	$(-1,1)$	C^∞	Yes	No	Yes
Square Nonlinearity (SQNL)[10]		$f(x)=\begin{cases}1 & : x>2.0\\ x-\frac{x^2}{4} & : 0\le x\le 2.0\\ x+\frac{x^2}{4} & : -2.0\le x<0\\ -1 & : x<-2.0\end{cases}$	$f'(x)=1\mp\frac{x}{2}$	$(-1,1)$	C^2	Yes	No	Yes
SoftPlus[23]		$f(x)=\ln(1+e^x)$	$f'(x)=\frac{1}{1+e^{-x}}$	$(0,\infty)$	C^∞	Yes	Yes	No
SoftExponential[28]		$f(\alpha,x)=\begin{cases}-\frac{\ln(1-\alpha(x+\alpha))}{\alpha} & \text{for } \alpha<0\\ x & \text{for } \alpha=0\\ \frac{e^{\alpha x}-1}{\alpha}+\alpha & \text{for } \alpha>0\end{cases}$	$f'(\alpha,x)=\begin{cases}\frac{1}{1-\alpha(\alpha+x)} & \text{for } \alpha<0\\ e^{\alpha x} & \text{for } \alpha\ge 0\end{cases}$	$(-\infty,\infty)$	C^∞	Yes	Yes	Yes iff $\alpha=0$
Soft Clipping[29]		$f(\alpha,x)=\frac{1}{\alpha}\log\frac{1+e^{\alpha x}}{1+e^{\alpha(x-1)}}$	$f'(\alpha,x)=\frac{1}{2}\sinh\left(\frac{p}{2}\right)\operatorname{sech}\left(\frac{px}{2}\right)\operatorname{sech}\left(\frac{p}{2}(1-x)\right)$	$(0,1)$	C^∞	Yes	No	No
Sinusoid[30]		$f(x)=\sin(x)$	$f'(x)=\cos(x)$	$[-1,1]$	C^∞	No	No	Yes
Sinc		$f(x)=\begin{cases}1 & \text{for } x=0\\ \frac{\sin(x)}{x} & \text{for } x\ne 0\end{cases}$	$f'(x)=\begin{cases}0 & \text{for } x=0\\ \frac{\cos(x)}{x}-\frac{\sin(x)}{x^2} & \text{for } x\ne 0\end{cases}$	$[\approx -.217234,1]$	C^∞	No	No	No
Sigmoid Linear Unit (SiLU)[25] (AKA SiL[26] and Swish-1[27])			No	Approximates identity/2				
Scaled exponential linear unit (SELU)[20]		$f(\alpha,x)=\lambda\begin{cases}\alpha(e^x-1) & \text{for } x<0\\ x & \text{for } x\ge 0\end{cases}$ with $\lambda=1.0507$ and $\alpha=1.67326$	$f'(\alpha,x)=\lambda\begin{cases}\alpha(e^x) & \text{for } x<0\\ 1 & \text{for } x\ge 0\end{cases}$	$(-\lambda\alpha,\infty)$	C^0	Yes	No	No

S-shaped rectified linear activation unit (SReLU)[21]		$f_{t_l,a_l,t_r,a_r}(x) = \begin{cases} t_l + a_l(x - t_l) & \text{for } x \le t_l \\ x & \text{for } t_l < x < t_r \\ t_r + a_r(x - t_r) & \text{for } x \ge t_r \end{cases}$ t_l, a_l, t_r, a_r are parameters.	$f'_{t_l,a_l,t_r,a_r}(x) = \begin{cases} a_l & \text{for } x \le t_l \\ 1 & \text{for } t_l < x < t_r \\ a_r & \text{for } x \ge t_r \end{cases}$	$(-\infty, \infty)$	C^0	No	No	No
Rectified linear unit (ReLU)[14]		$f(x) = \begin{cases} 0 & \text{for } x \le 0 \\ x & \text{for } x > 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \le 0 \\ 1 & \text{for } x > 0 \end{cases}$	$[0, \infty)$	C^0	Yes	Yes	No
Randomized leaky rectified linear unit (RReLU)[18]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \ge 0 \end{cases}$ [3]	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \ge 0 \end{cases}$	$(-\infty, \infty)$	C^0	Yes	Yes	No
Parameteric rectified linear unit (PReLU)[17]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \ge 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \ge 0 \end{cases}$	$(-\infty, \infty)$[2]	C^0	Yes iff $\alpha \ge 0$	Yes	Yes iff $\alpha = 1$
Logistic (a.k.a. Sigmoid or Soft step)		$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$[1]	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$	C^∞	Yes	No	No
Leaky rectified linear unit (Leaky ReLU)[16]		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \ge 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ 1 & \text{for } x \ge 0 \end{cases}$	$(-\infty, \infty)$	C^0	Yes	Yes	No
Inverse square root unit (ISRU)[13]		$f(x) = \frac{x}{\sqrt{1 + \alpha x^2}}$	$f'(x) = \left(\frac{1}{\sqrt{1 + \alpha x^2}}\right)^3$	$\left(-\frac{1}{\sqrt{\alpha}}, \frac{1}{\sqrt{\alpha}}\right)$	C^∞	Yes	No	Yes
Inverse square root linear unit (ISRLU)[13]		$f(x) = \begin{cases} \frac{x}{\sqrt{1 + \alpha x^2}} & \text{for } x < 0 \\ x & \text{for } x \ge 0 \end{cases}$	$f'(x) = \begin{cases} \left(\frac{1}{\sqrt{1 + \alpha x^2}}\right)^3 & \text{for } x < 0 \\ 1 & \text{for } x \ge 0 \end{cases}$	$\left(-\frac{1}{\sqrt{\alpha}}, \infty\right)$	C^2	Yes	Yes	Yes
Identity								
GELU[24]		$f(x) = x\Phi(x) = x(1 + \operatorname{erf}(x/\sqrt{2}))/2$	$f'(x) = \Phi(x) + x\phi(x)$		C^∞	No	No	No
Gaussian		$f(x) = e^{-x^2}$	$f'(x) = -2xe^{-x^2}$	$(0, 1]$	C^∞	No	No	No
Exponential linear unit (ELU)[19]		$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x \le 0 \\ x & \text{for } x > 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} f(\alpha, x) + \alpha & \text{for } x \le 0 \\ 1 & \text{for } x > 0 \end{cases}$	$(-\alpha, \infty)$	$\begin{cases} C_1 & \text{when } \alpha = 1 \\ C_0 & \text{otherwise} \end{cases}$	Yes iff $\alpha \ge 0$	Yes iff $0 \le \alpha \le 1$	Yes iff $\alpha = 1$
ElliotSig[8][9][10] Softs		$f(x) = \frac{x}{1 + \lvert x \rvert}$	$f'(x) = \frac{1}{(1 + \lvert x \rvert)^2}$	$(-1, 1)$	C^1	Yes	No	Yes
Bipolar rectified linear unit (BReLU)[15]		$f(x_i) = \begin{cases} ReLU(x_i) & \text{if } i \bmod 2 = 0 \\ -ReLU(-x_i) & \text{if } i \bmod 2 \ne 0 \end{cases}$	$f'(x_i) = \begin{cases} ReLU'(x_i) & \text{if } i \bmod 2 = 0 \\ -ReLU'(-x_i) & \text{if } i \bmod 2 \ne 0 \end{cases}$	$(-\infty, \infty)$	C^0	Yes	Yes	No
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \ge 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \ne 0 \\ ? & \text{for } x = 0 \end{cases}$	$\{0, 1\}$	C^{-1}	Yes	No	No
Bent identity		$f(x) = \frac{\sqrt{x^2 + 1} - 1}{2} + x$	$f'(x) = \frac{x}{2\sqrt{x^2 + 1}} + 1$	$(-\infty, \infty)$	C^∞	Yes	Yes	Yes
ArSinH		$f(x) = \sinh^{-1}(x) = \ln\left(x + \sqrt{x^2 + 1}\right)$	$f'(x) = \frac{1}{\sqrt{x^2 + 1}}$	$(-\infty, \infty)$	C^∞	Yes	No	Yes
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$	$\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$	C^∞	Yes	No	Yes
Adaptive piecewise linear (APL)[22]		$f(x) = \max(0, x) + \sum_{s=1}^{S} a_i^s \max(0, -x + b_i^s)$	$f'(x) = H(x) - \sum_{s=1}^{S} a_i^s H(-x + b_i^s)$[4]	$(-\infty, \infty)$	C^0	No	No	No

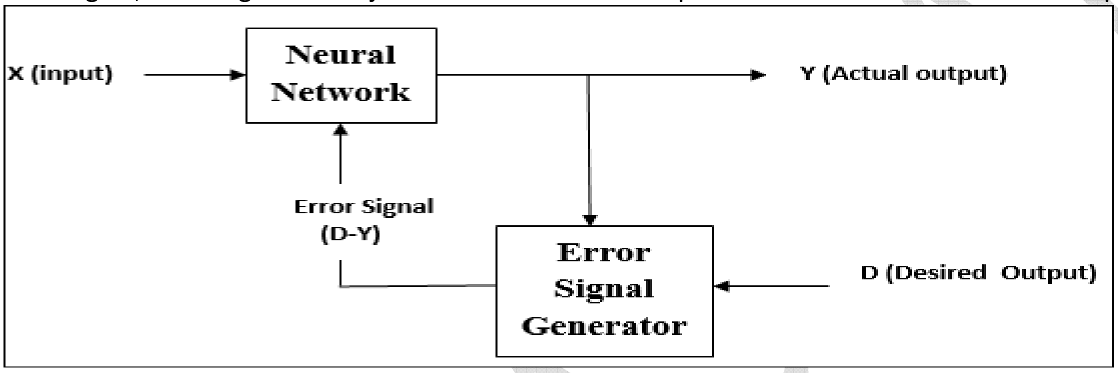
Name	Equation	Derivatives	Range	Order of continuity
Softmax	$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}$ for $i = 1, \ldots, J$	$\frac{\partial f_i(\vec{x})}{\partial x_j} = f_i(\vec{x})(\delta_{ij} - f_j(\vec{x}))$[5]	$(0,1)$	C^∞
Maxout[31]	$f(\vec{x}) = \max_i x_i$	$\frac{\partial f}{\partial x_j} = \begin{cases} 1 & \text{for } j = \operatorname{argmax}_i x_i \\ 0 & \text{for } j \neq \operatorname{argmax}_i x_i \end{cases}$	$(-\infty, \infty)$	C^0

Adjustments of Weights or Learning

Learning, in artificial neural network, is the method of modifying the weights of connections between the neurons of a specified network. Learning in ANN can be classified into three categories namely supervised learning, unsupervised learning, and reinforcement learning.

Supervised Learning

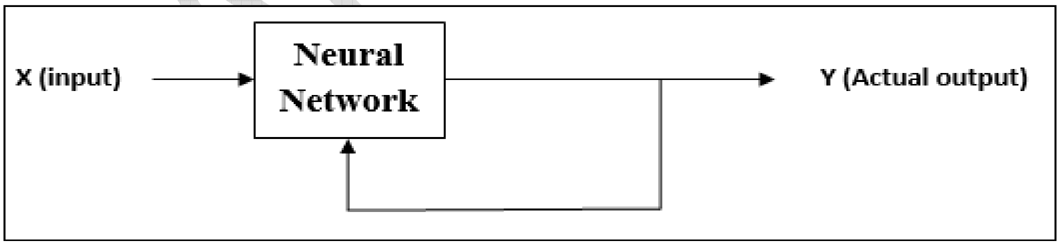
As the name suggests, this type of learning is done under the supervision of a teacher. This learning process is dependent.During the training of ANN under supervised learning, the input vector is presented to the network, which will give an output vector. This output vector is compared with the desired output vector. An error signal is generated, if there is a difference between the actual output and the desired output vector. On the basis of this error signal, the weights are adjusted until the actual output is matched with the desired output.



Unsupervised Learning

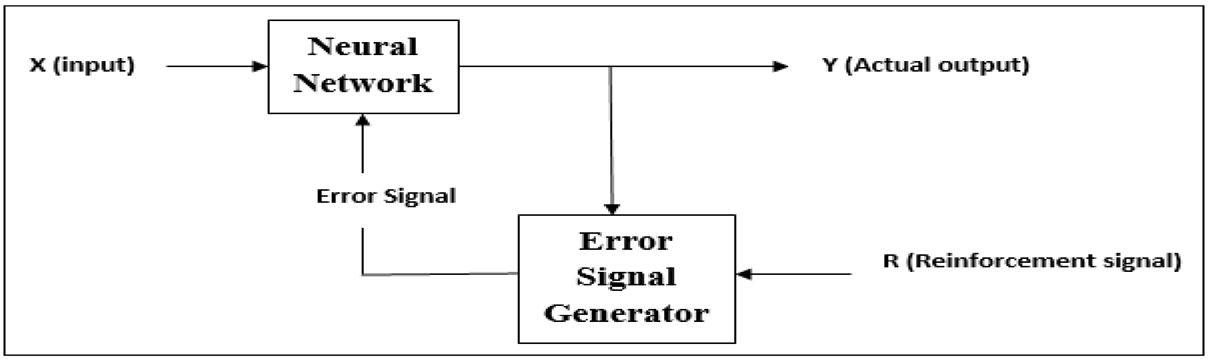
As the name suggests, this type of learning is done without the supervision of a teacher. This learning process is independent.

During the training of ANN under unsupervised learning, the input vectors of similar type are combined to form clusters. When a new input pattern is applied, then the neural network gives an output response indicating the class to which the input pattern belongs.There is no feedback from the environment as to what should be the desired output and if it is correct or incorrect. Hence, in this type of learning, the network itself must discover the patterns and features from the input data, and the relation for the input data over the output.



Reinforcement Learning

As the name suggests, this type of learning is used to reinforce or strengthen the network over some critic information. This learning process is similar to supervised learning, however we might have very less information. During the training of network under reinforcement learning, the network receives some feedback from the environment. This makes it somewhat similar to supervised learning. However, the feedback obtained here is evaluative not instructive, which means there is no teacher as in supervised learning. After receiving the feedback, the network performs adjustments of the weights to get better critic information in future.



What Is Learning in ANN?
Basically, learning means to do and adapt the change in itself as and when there is a change in environment. ANN is a complex system or more precisely we can say that it is a complex adaptive system, which can change its internal structure based on the information passing through it.

Why Is It important?
Being a complex adaptive system, learning in ANN implies that a processing unit is capable of changing its input/output behavior due to the change in environment. The importance of learning in ANN increases because of the fixed activation function as well as the input/output vector, when a particular network is constructed. Now to change the input/output behavior, we need to adjust the weights.

Classification
It may be defined as the process of learning to distinguish the data of samples into different classes by finding common features between the samples of the same classes. For example, to perform training of ANN, we have some training samples with unique features, and to perform its testing we have some testing samples with other unique features. Classification is an example of supervised learning.

Neural Network Learning Rules
We know that, 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 –** It identifies, how to modify the weights of nodes of a network.
- **Perceptron learning rule –** Network starts its learning by assigning a random value to each weight.
- **Delta learning rule –** Modification in sympatric weight of a node is equal to the multiplication of error and the input.
- **Correlation learning rule –** The correlation rule is the supervised learning.
- **Outstar learning rule –** We can use it when it assumes that nodes or neurons in a network arranged in a layer

1) <u>Hebbian Learning Rule</u>
The **Hebbian rule** was the first learning rule. In 1949 *Donald Hebb* developed it as learning algorithm of the unsupervised neural network. We can use it to identify how to improve the weights of nodes of a network.
The **Hebb learning rule** assumes that – If two neighbor neurons activated and deactivated at the same time. Then the weight connecting these neurons should increase. For neurons operating in the opposite phase, the weight between them should decrease. If there is no signal correlation, the weight should not change.

When inputs of both the nodes are either positive or negative, then a strong positive weight exists between the nodes. If the input of a node is positive and negative for other, a strong negative weight exists between the nodes. At the start, values of all weights are set to zero. This learning rule can be used0 for both soft- and hard-activation functions. Since desired responses of neurons are not used in the learning procedure, this is the unsupervised learning rule. The absolute values of the weights are usually proportional to the learning time, which is undesired.

$$w_{ij} = x_i * x_j$$

2) <u>Perceptron Learning Rule</u>

As you know, each connection in a neural network has an associated weight, which changes in the course of learning. According to it, an example of supervised learning, the network starts its learning by assigning a random value to each weight.

Calculate the output value on the basis of a set of records for which we can know the expected output value. This is the learning sample that indicates the entire definition. As a result, it is called a learning sample.
The network then compares the calculated output value with the expected value. Next calculates an error function ∈, which can be the sum of squares of the errors occurring for each individual in the learning sample.
Computed as follows:

$$\sum_i \sum_j \left(E_{ij} - O_{ij}\right)^2$$

Perform the first summation on the individuals of the learning set, and perform the second summation on the output units. Eij and Oij are the expected and obtained values of the jth unit for the ith individual.
The network then adjusts the weights of the different units, checking each time to see if the error function has increased or decreased. As in a conventional regression, this is a matter of solving a problem of least squares.
Since assigning the weights of nodes according to users, it is an example of supervised learning.

3) Delta Learning Rule

Developed by *Widrow* and *Hoff*, the delta rule, is one of the most common learning rules. It depends on supervised learning.
This rule states that the modification in sympatric weight of a node is equal to the multiplication of error and the input.
In Mathematical form the delta rule is as follows:

$$\Delta w = \eta\,(t - y)\,x_i$$

For a given input vector, compare the output vector is the correct answer. If the difference is zero, no learning takes place; otherwise, adjusts its weights to reduce this difference. The change in weight from ui to uj is: dwij = r* ai * ej. where r is the learning rate, ai represents the activation of ui and ej is the difference between the expected output and the actual output of uj. If the set of input patterns form an independent set then learn arbitrary associations using the delta rule.
It has seen that for networks with linear activation functions and with no hidden units. The error squared vs. the weight graph is a paraboloid in n-space. Since the proportionality constant is negative, the graph of such a function is concave upward and has the least value. The vertex of this paraboloid represents the point where it reduces the error. The weight vector corresponding to this point is then the ideal weight vector.
We can use the delta learning rule with both single output unit and several output units.
While applying the delta rule assume that the error can be directly measured.
The aim of applying the delta rule is to reduce the difference between the actual and expected output that is the error.

4) Correlation Learning Rule

The **correlation learning rule** based on a similar principle as the Hebbian learning rule. It assumes that weights between responding neurons should be more positive, and weights between neurons with opposite reaction should be more negative.
Contrary to the Hebbian rule, the correlation rule is the supervised learning. Instead of an actual
The response, oj, the desired response, dj, uses for the weight-change calculation.
In Mathematical form the correlation learning rule is as follows:

$$\Delta w_{ij} = \eta x_i d_j$$

Where dj is the desired value of output signal. This training algorithm usually starts with the initialization of weights to zero.Since assigning the desired weight by users, the correlation learning rule is an example of supervised learning.

5) Out Star Learning Rule

We use the Out Star Learning Rule when we assume that nodes or neurons in a network arranged in a layer. Here the weights connected to a certain node should be equal to the desired outputs for the neurons connected through those weights. The out start rule produces the desired response t for the layer of n nodes.

Apply this type of learning for all nodes in a particular layer. Update the weights for nodes are as in Kohonen neural networks.
In Mathematical form, express the out star learning as follows:

$$w_{jk} = \begin{cases} \eta(y_k - w_{jk}) \text{if node j wins the competition} \\ 0 \quad \text{if node j losses the competition} \end{cases}$$

Supervised learning takes place under the supervision of a teacher. This learning process is dependent. During the training of ANN under supervised learning, the input vector is presented to the network, which will produce an output vector. This output vector is compared with the desired/target output vector. An error signal is generated if there is a difference between the actual output and the desired/target output vector. On the basis of this error signal, the weights would be adjusted until the actual output is matched with the desired output.

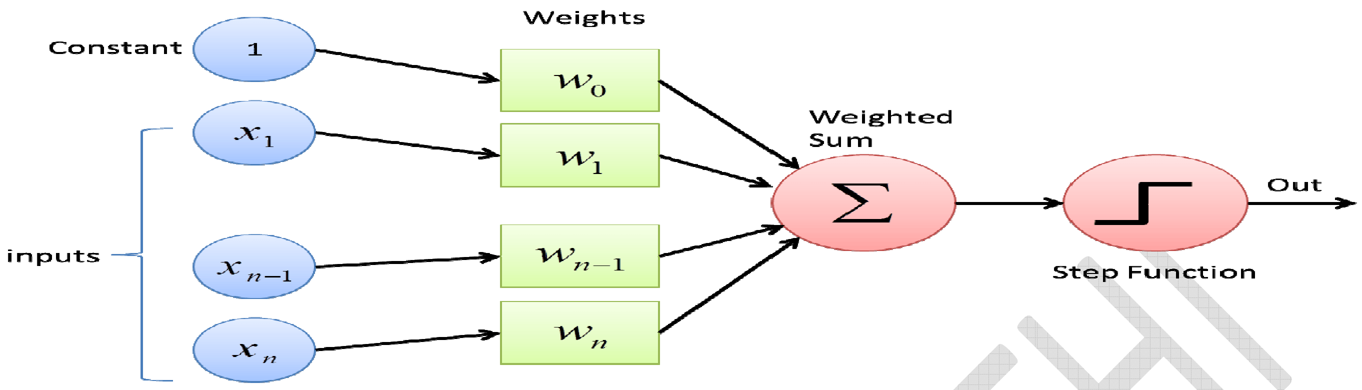
Supervised learning in neural networks is usually performed in the following sequence:
1. Set an appropriate structure of a neural network, having, for example, (n + 1) input neurons (n for the input variables and 1 for the bias, x0) and m output neurons and set initial values of the connection weights of the network.
2. Supply an input vector x from the set of the training examples X to the network.
3. Calculate the output vector o as produced by the neural network.
4. Compare the desired output vector y (answer, from the training data) and the output vector o produced
by the network; if possible, evaluate the error.
5. Correct the connection weights in such a way that the next time x is presented to the network, the produced output o becomes closer to the desired output y.
6. If necessary, repeat steps 2 to 5 until the network reaches a convergence state.
Evaluating an error of approximation can be done in many ways, the most used being instantaneous error:
Err = (o - y), or Err = |o - y|;
mean-square error (MSE):
Err = (o- y)2/2;
a total MSE sums the error over all individual examples and all the output neurons in the network:

$$\boxed{Err = \left(\sum_{k=1}^{p} \sum_{j=1}^{m} (o_j^{(k)} - y_j^{(k)})^2\right) \Big/ p \cdot m}$$

The above general algorithm for supervised learning in a neural network has different implementations, mainly distinguished by the way the connection weights are changed through training. Some of the algorithms are **perceptron learning (Rosenblatt 1958); ADALINE (Widrow and Hoff 1960); the backpropagation algorithm (Rumelhart et al. 1986b; and others); and (learning vector quantization) LVQ1,2,3 algorithms (Kohonen 1990).**

Perceptron
Developed by Frank Rosenblatt by using McCulloch and Pitts model, perceptron is the basic operational unit of artificial neural networks. It employs supervised learning rule and is able to classify the data into two classes.

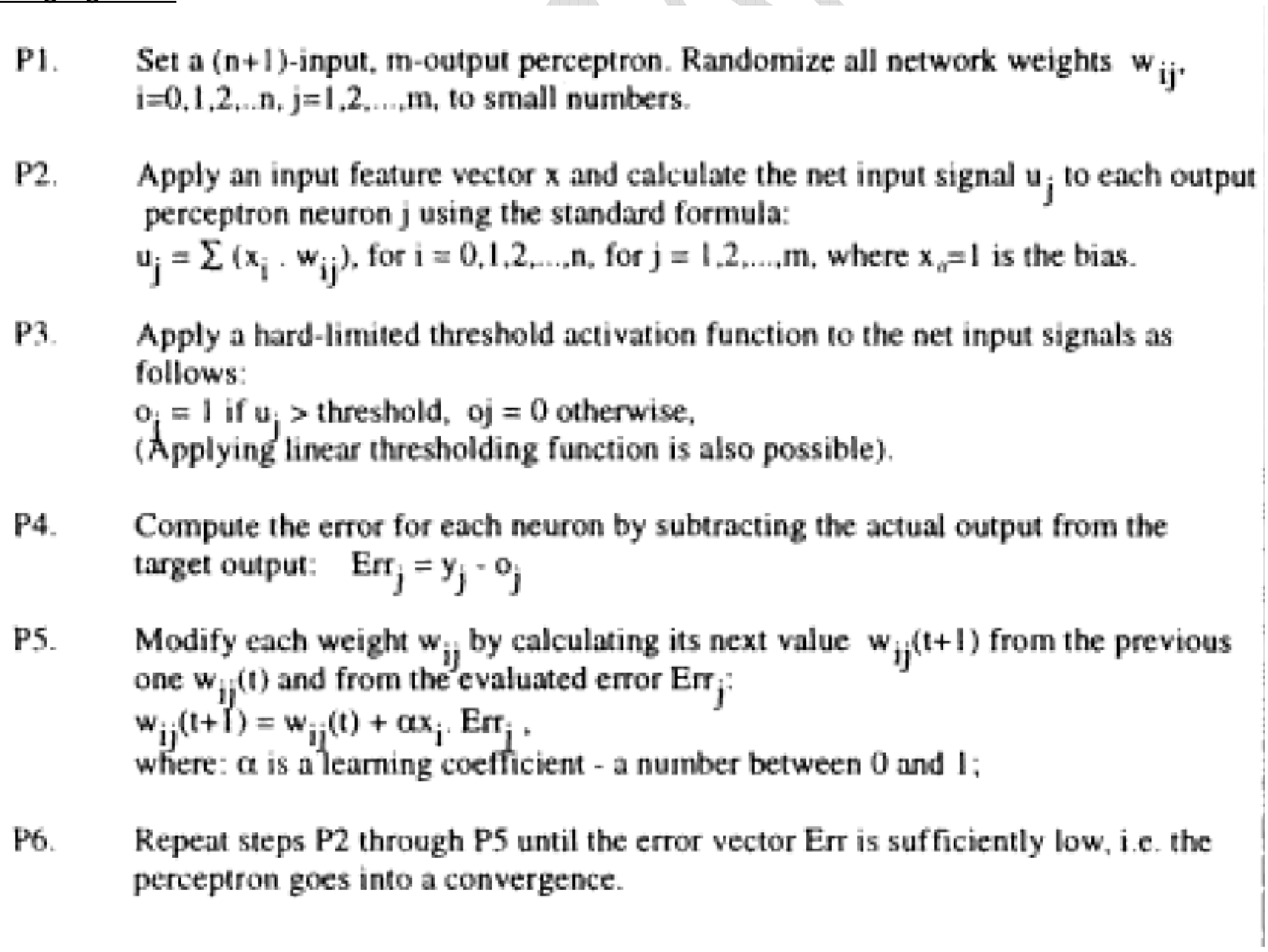
Operational characteristics of the perceptron: It consists of a single neuron with an arbitrary number of inputs along with adjustable weights, but the output of the neuron is 1 or 0 depending upon the threshold. It also consists of a bias whose weight is always 1. Following figure gives a schematic representation of the perceptron.



Perceptron thus has the following three basic elements –

- **Links** – It would have a set of connection links, which carries a weight including a bias always having weight 1.
- **Adder** – It adds the input after they are multiplied with their respective weights.
- **Activation function** – It limits the output of neuron. The most basic activation function is a Heaviside step function that has two possible outputs. This function returns 1, if the input is positive, and 0 for any negative input.

Training Algorithm



P1. Set a (n+1)-input, m-output perceptron. Randomize all network weights w_{ij}, i=0,1,2,..n, j=1,2,...,m, to small numbers.

P2. Apply an input feature vector x and calculate the net input signal u_j to each output perceptron neuron j using the standard formula:
$u_j = \Sigma (x_i \cdot w_{ij})$, for i = 0,1,2,...,n, for j = 1,2,...,m, where x_o=1 is the bias.

P3. Apply a hard-limited threshold activation function to the net input signals as follows:
$o_j = 1$ if $u_j >$ threshold, oj = 0 otherwise,
(Applying linear thresholding function is also possible).

P4. Compute the error for each neuron by subtracting the actual output from the target output: $Err_j = y_j - o_j$

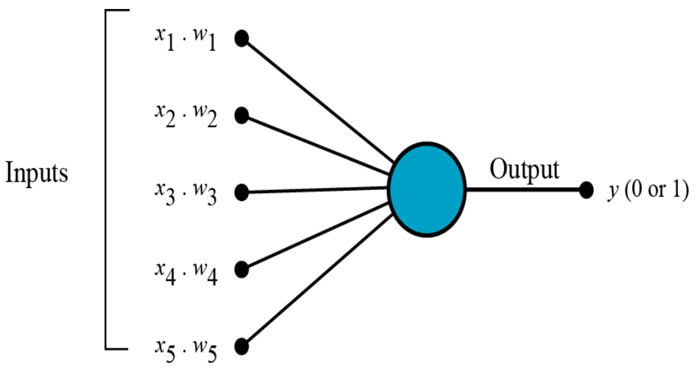
P5. Modify each weight w_{ij} by calculating its next value $w_{ij}(t+1)$ from the previous one $w_{ij}(t)$ and from the evaluated error Err_j:
$w_{ij}(t+1) = w_{ij}(t) + \alpha x_i \cdot Err_j$,
where: α is a learning coefficient - a number between 0 and 1;

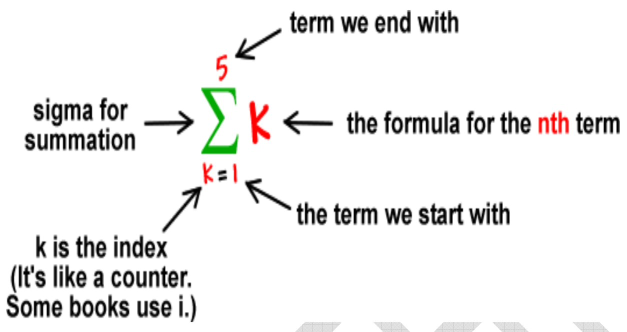
P6. Repeat steps P2 through P5 until the error vector Err is sufficiently low, i.e. the perceptron goes into a convergence.

How does it work?

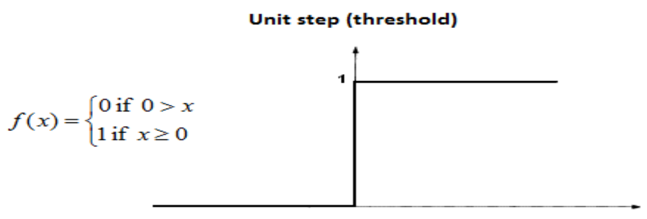
The perceptron works on these simple steps

a. All the inputs x are multiplied with their weights w. Let’s call it k.
b. Add all the multiplied values and call them Weighted Sum.
c. Apply that weighted sum to the correct Activation Function.

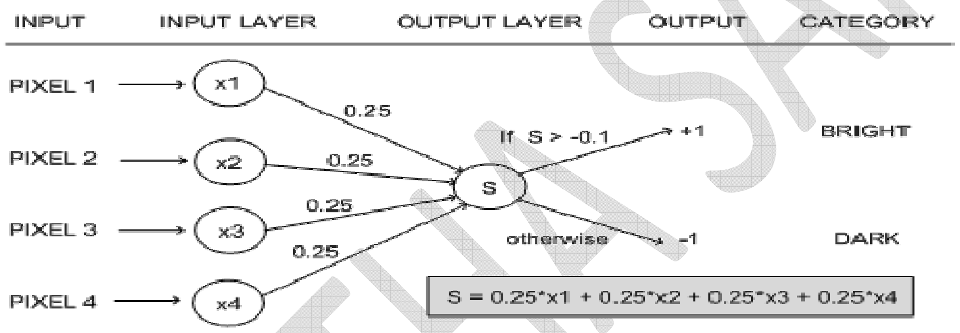




term we end with
5
sigma for summation
$\sum k$
the formula for the nth term
k=1
the term we start with
k is the index (It's like a counter. Some books use i.)



Why do we need Weights and Bias?
Weights shows the strength of the particular node. ***A bias*** value allows you to shift the activation function curve up or down.



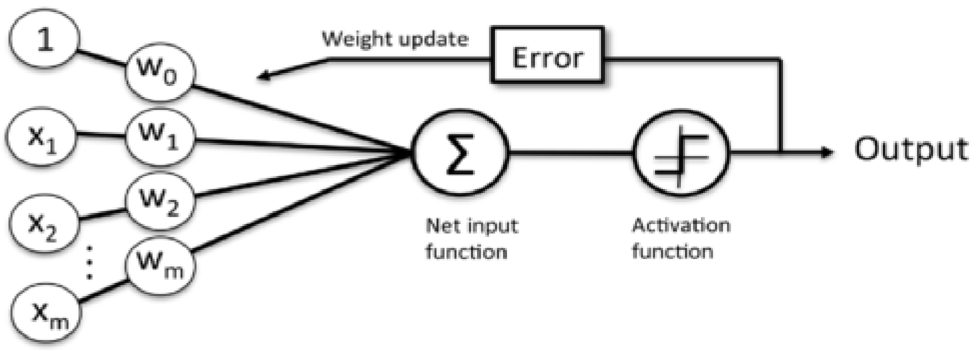
Why do we need Activation Function?
In short, **the activation functions are used to map the input between the required values like (0, 1) or (-1, 1)**.
Where we use Perceptron?
Perceptron is usually used to classify the data into two parts. Therefore, it is also known as a Linear Binary Classifier.

Adaptive Linear Neuron

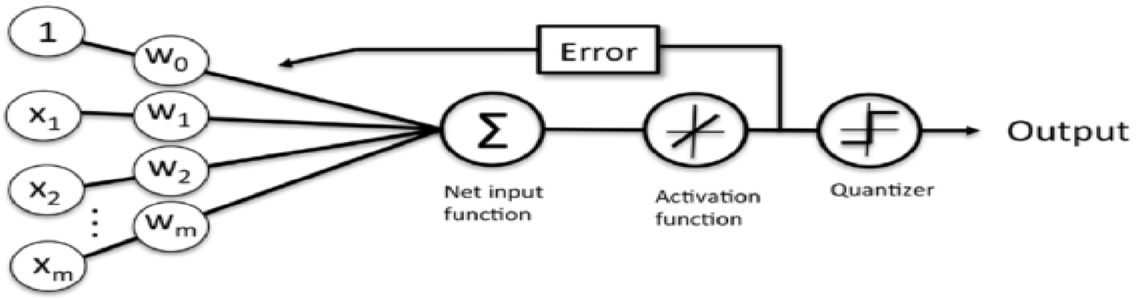
Adaline which stands for Adaptive Linear Neuron, is a network having a single linear unit. It was developed by Widrow and Hoff in 1960. Some important points about Adaline are as follows −

- It uses bipolar activation function.
- It uses delta rule for training to minimize the Mean-Squared Error (MSE) between the actual output and the desired/target output.
- The weights and the bias are adjustable.

The key difference between the Adaline rule (also known as the Widrow-Hoff rule) and Rosenblatt's perceptron is that the weights are updated based on a linear activation function rather than a unit step function.



Perceptron



Adaptive linear neuron

The difference is that we're going to use the continuous valued output from the linear activation function to compute the model error and update the weights, rather than the binary class labels.

In contrast to the perceptron rule, the delta rule of the adaline updates the weights based on a linear activation function rather than a unit step function; here, this linear activation function g(z)g(z) is just the identity function of the net input g(wTx)=wTx. In the next section, we will see why this linear activation is an improvement over the perceptron update and where the name "delta rule" comes from

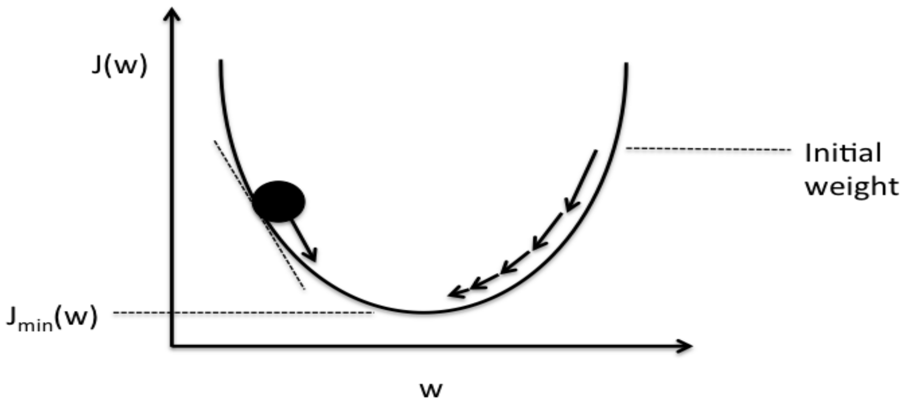
Gradient Descent

Being a continuous function, one of the biggest advantages of the linear activation function over the unit step function is that it is differentiable. This property allows us to define a cost function J(w)J(w) that we can minimize in order to update our weights. In the case of the linear activation function, we can define the cost function J(w)J(w) as the *sum of squared errors* (SSE), which is similar to the cost function that is minimized in ordinary least squares (OLS) linear regression.

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i}(\mathbf{target}^{(i)} - \mathbf{output}^{(i)})^2 \qquad \mathbf{output}^{(i)} \in \mathbb{R}$$

(The fraction 1/2 is just used for convenience to derive the gradient as we will see in the next paragraphs.)
In order to minimize the SSE cost function, we will use gradient descent, a simple yet useful optimization algorithm that is often used in machine learning to find the local minimum of linear systems.
Before we get to the fun part (calculus), let us consider a convex cost function for one single weight. As illustrated in the figure below, we can describe the principle behind gradient descent as "climbing down a hill" until a local or global minimum is reached. At each step, we take a step into the opposite direction of the gradient, and the step size is determined by the value of the learning rate as well as the slope of the gradient.

In order to minimize the SSE cost function, we will use gradient descent, a simple yet useful optimization algorithm that is often used in machine learning to find the local minimum of linear systems.
Before we get to the fun part (calculus), let us consider a convex cost function for one single weight. As illustrated in the figure below, we can describe the principle behind gradient descent as "climbing down a hill" until a local or global minimum is reached. At each step, we take a step into the opposite direction of the gradient, and the step size is determined by the value of the learning rate as well as the slope of the gradient.



Schematic of gradient descent.

Now, as promised, onto the fun part – deriving the Adaline learning rule. As mentioned above, each update is updated by taking a step into the opposite direction of the gradient $\Delta \mathbf{w} = -\eta \nabla J(\mathbf{w})$, thus, we have to compute the partial derivative of the cost function for each weight in the weight vector: $\Delta w_j = -\eta \frac{\partial J}{\partial w_j}$.

The partial derivative of the SSE cost function for a particular weight can be calculated as follows:

$$
\begin{aligned}
\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j} \frac{1}{2} \sum_i (t^{(i)} - o^{(i)})^2 \\
&= \frac{1}{2} \sum_i \frac{\partial}{\partial w_j} (t^{(i)} - o^{(i)})^2 \\
&= \frac{1}{2} \sum_i 2(t^{(i)} - o^{(i)}) \frac{\partial}{\partial w_j} (t^{(i)} - o^{(i)}) \\
&= \sum_i (t^{(i)} - o^{(i)}) \frac{\partial}{\partial w_j} \left(t^{(i)} - \sum_j w_j x_j^{(i)} \right) \\
&= \sum_i (t^{(i)} - o^{(i)})(-x_j^{(i)})
\end{aligned}
$$

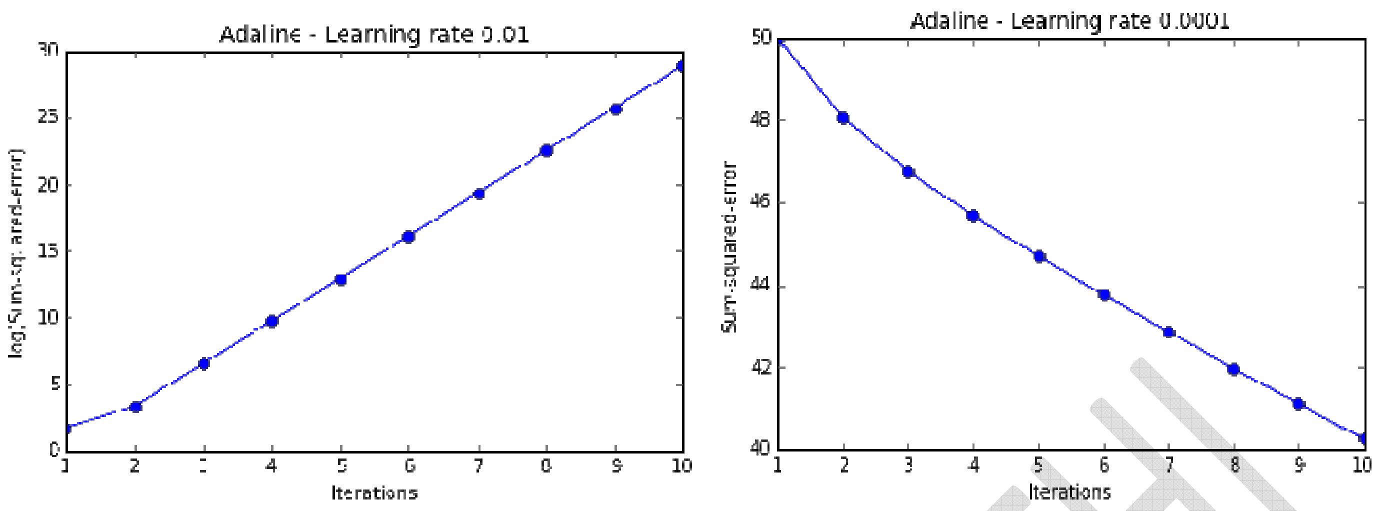
(t = target, o = output)
And if we plug the results back into the learning rule, we get

$$\Delta w_j = -\eta \frac{\partial J}{\partial w_j} = -\eta \sum_i (t^{(i)} - o^{(i)})(-x_j^{(i)}) = \eta \sum_i (t^{(i)} - o^{(i)}) x_j^{(i)}$$

Eventually, we can apply a simultaneous weight update similar to the perceptron rule: $w := w + \Delta w$.

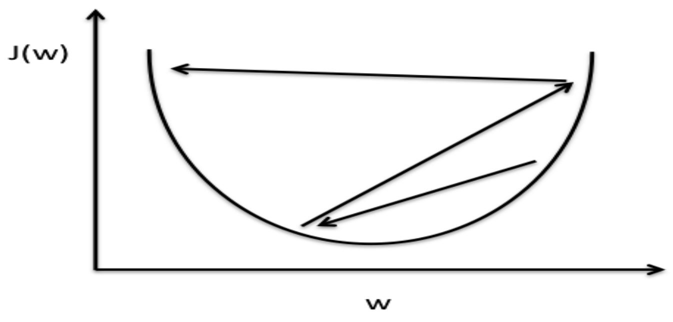
Although, the learning rule above looks identical to the perceptron rule, we shall note the two main differences:

1. Here, the output “o” is a real number and not a class label as in the perceptron learning rule.
2. The weight update is calculated based on all samples in the training set (instead of updating the weights incrementally after each sample), which is why this approach is also called “batch” gradient descent.

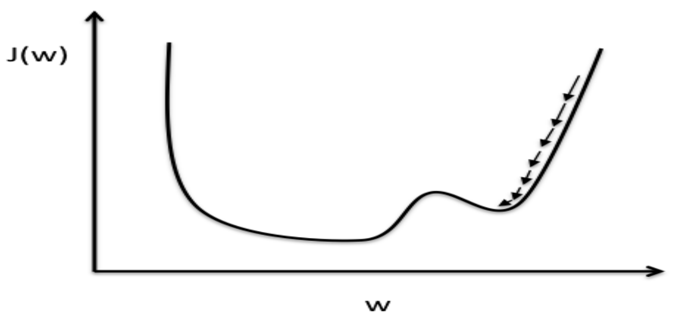


The two plots above nicely emphasize the importance of plotting learning curves by illustrating two most common problems with gradient descent:

1. If the learning rate is too large, gradient descent will overshoot the minima and diverge.
2. If the learning rate is too small, the algorithm will require too many epochs to converge and can become trapped in local minima more easily.



Large learning rate: Overshooting.



Small learning rate: Many iterations until convergence and trapping in local minima.

Gradient descent is also a good example why feature scaling is important for many machine learning algorithms. It is not only easier to find an appropriate learning rate if the features are on the same scale, but it also often leads to faster convergence and can prevent the weights from becoming too small (numerical stability). A common way of feature scaling is standardization

$$\mathbf{x}_{j,std} = \frac{\mathbf{x}_j - \mu_j}{\sigma_j}$$

where μj is the sample mean of the feature xj and σj the standard deviation, respectively. After standardization, the features will have unit variance and are centred around mean zero.

Online Learning via Stochastic Gradient Descent

Batch gradient descent learning the "batch" updates refers to the fact that the cost function is minimized based on the complete training data set. If we think back to the perceptron rule, we remember that it performed the weight update incrementally after each individual training sample. This approach is also called "online" learning, and in fact, this is also how Adaline was first described by Bernard Widrow.

The process of incrementally updating the weights is also called "stochastic" gradient descent since it approximates the minimization of the cost function. Although the stochastic gradient descent approach might sound inferior to gradient descent due its "stochastic" nature and the "approximated" direction (gradient), it can have certain advantages in practice. Often, stochastic gradient descent converges much faster than gradient descent since the updates are applied immediately after each training sample; stochastic gradient descent is computationally more efficient, especially for very large datasets. Another advantage of online learning is that the classifier can be immediately updated as new training data arrives, e.g., in web applications, and old training data can be discarded if storage is an issue. In large-scale machine learning systems, it is also common practice to use so-called “mini-batches”, a compromise with smoother convergence than stochastic gradient descent.

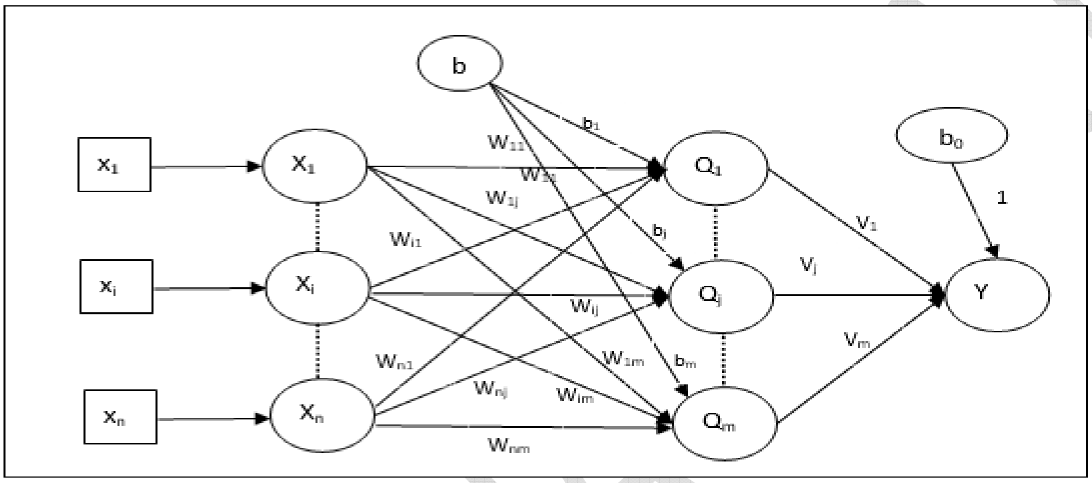
Multiple Adaptive Linear Neuron (Madaline)

Madaline which stands for Multiple Adaptive Linear Neuron, is a network which consists of many Adalines in parallel. It will have a single output unit. Some important points about Madaline are as follows −

- It is just like a multilayer perceptron, where Adaline will act as a hidden unit between the input and the Madaline layer.
- The weights and the bias between the input and Adaline layers, as in we see in the Adaline architecture, are adjustable.
- The Adaline and Madaline layers have fixed weights and bias of 1.
- Training can be done with the help of Delta rule.

Architecture

The architecture of Madaline consists of **“n”** neurons of the input layer, **“m”** neurons of the Adaline layer, and 1 neuron of the Madaline layer. The Adaline layer can be considered as the hidden layer as it is between the input layer and the output layer, i.e. the Madaline layer.

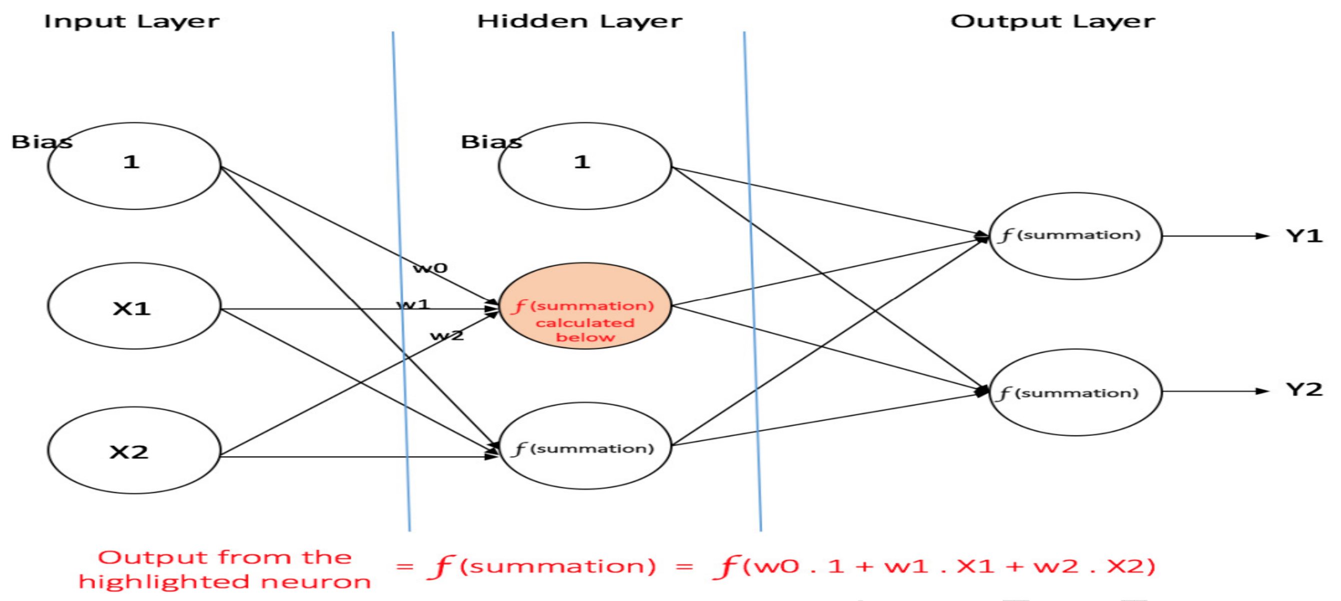


Multilayer perceptron (Feedforward Neural Networks)

A **multilayer perceptron** (MLP) is a class of feedforward artificial neural network. An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called back propagation for training.Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable.Multilayer perceptrons are sometimes colloquially referred to as "vanilla" neural networks, especially when they have a single hidden layer.

Input Layer: The Input layer has three nodes. The Bias node has a value of 1. The other two nodes take X1 and X2 as external inputs (which are numerical values depending upon the input dataset). As discussed above, no computation is performed in the Input layer, so the outputs from nodes in the Input layer are 1, X1 and X2 respectively, which are fed into the Hidden Layer.

Hidden Layer: The Hidden layer also has three nodes with the Bias node having an output of 1. The output of the other two nodes in the Hidden layer depends on the outputs from the Input layer (1, X1, X2) as well as the weights associated with the connections (edges). Below figure shows the output calculation for one of the hidden nodes (highlighted). Similarly, the output from other hidden node can be calculated. Remember that ***f*** refers to the activation function. These outputs are then fed to the nodes in the Output layer.


Input Layer
Hidden Layer
Output Layer
Bias
1
X1
X2
Bias
1
w0
w1
w2
ƒ(summation) calculated below
ƒ(summation)
ƒ(summation)
ƒ(summation)
Y1
Y2
Output from the highlighted neuron = ƒ(summation) = ƒ(w0 . 1 + w1 . X1 + w2 . X2)

Output Layer: The Output layer has two nodes which take inputs from the Hidden layer and perform similar computations as shown for the highlighted hidden node. The values calculated (Y1 and Y2) as a result of these computations act as outputs of the Multi Layer Perceptron.

Given a set of features **X = (x1, x2, …)** and a target **y**, a Multi Layer Perceptron can learn the relationship between the features and the target, for either classification or regression.
Lets take an example to understand Multi Layer Perceptrons better. Suppose we have the following student-marks dataset:

Hours Studied	Mid Term Marks	Final Term Result
35	67	1 (Pass)
12	75	0 (Fail)
16	89	1 (Pass)
45	56	1 (Pass)
10	90	0 (Fail)

The two input columns show the number of hours the student has studied and the mid term marks obtained by the student. The Final Result column can have two values 1 or 0 indicating whether the student passed in the final term. For example, we can see that if the student studied 35 hours and had obtained 67 marks in the mid term, he / she ended up passing the final term.
Now, suppose, we want to predict whether a student studying 25 hours and having 70 marks in the mid term will pass the final term.

Hours Studied	Mid Term Marks	Final Term Result
25	70	?

This is a binary classification problem where a multi layer perceptron can learn from the given examples (training data) and make an informed prediction given a new data point. We will see below how a multi layer perceptron learns such relationships.

Training our MLP: The Back-Propagation Algorithm

The process by which a Multi Layer Perceptron learns is called the Back-proagation algorithm.
Backward Propagation of Errors, often abbreviated as BackProp is one of the several ways in which an artificial neural network (ANN) can be trained. It is a supervised training scheme, which means, it learns from labeled training data (there is a supervisor, to guide its learning).
To put in simple terms, BackProp is like "**learning from mistakes**". The supervisor ***corrects*** the ANN whenever it makes mistakes.

An ANN consists of nodes in different layers; input layer, intermediate hidden layer(s) and the output layer. The connections between nodes of adjacent layers have "weights" associated with them. The goal of learning is to assign correct weights for these edges. Given an input vector, these weights determine what the output vector is.

In supervised learning, the training set is labeled. This means, for some given inputs, we know the desired/expected output (label).

BackProp Algorithm:
Initially all the edge weights are randomly assigned. For every input in the training dataset, the ANN is activated and its output is observed. This output is compared with the desired output that we already know, and the error is "propagated" back to the previous layer. This error is noted and the weights are "adjusted" accordingly. This process is repeated until the output error is below a predetermined threshold.
Once the above algorithm terminates, we have a "learned" ANN which, we consider is ready to work with "new" inputs. This ANN is said to have learned from several examples (labeled data) and from its mistakes (error propagation).

The Multi Layer Perceptron shown in below Figure has two nodes in the input layer (apart from the Bias node) which take the inputs 'Hours Studied' and 'Mid Term Marks'. It also has a hidden layer with two nodes (apart from the Bias node). The output layer has two nodes as well – the upper node outputs the probability of 'Pass' while the lower node outputs the probability of 'Fail'.

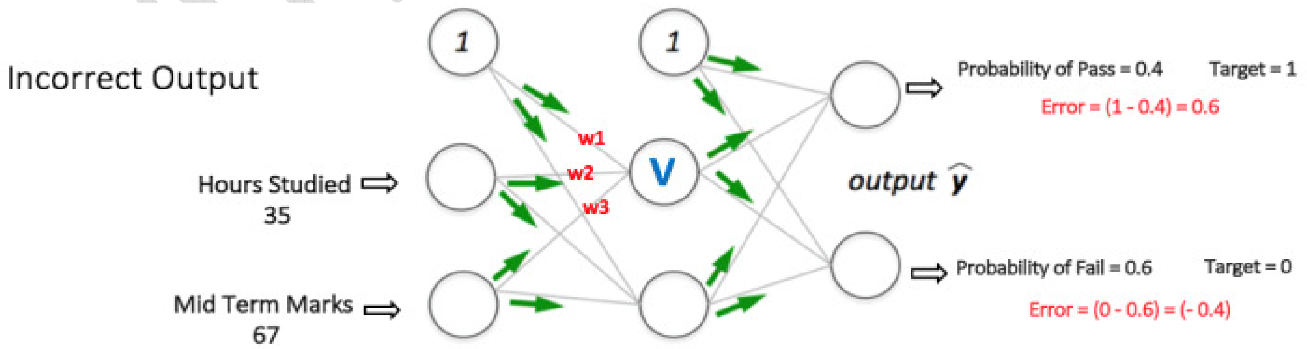
In classification tasks, we generally use a Softmax function as the Activation Function in the Output layer of the Multi Layer Perceptron to ensure that the outputs are probabilities and they add up to 1. The Softmax function takes a vector of arbitrary real-valued scores and squashes it to a vector of values between zero and one that sum to one. So, in this case, Probability (Pass) + Probability (Fail) = 1

Step 1: Forward Propagation
All weights in the network are randomly assigned. Lets consider the hidden layer node marked **V** in above Figure below. Assume the weights of the connections from the inputs to that node are w1, w2 and w3 (as shown).
The network then takes the first training example as input (we know that for inputs 35 and 67, the probability of Pass is 1).

- Input to the network = [35, 67]
- Desired output from the network (target) = [1, 0]

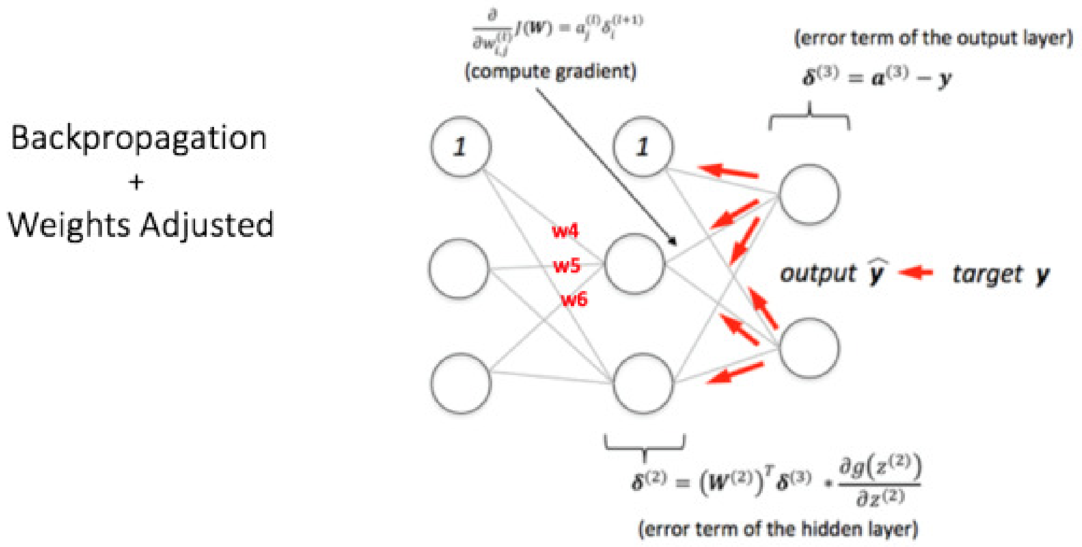
Then output V from the node in consideration can be calculated as below (***f*** is an activation function such as sigmoid): V = ***f*** (1*w1 + 35*w2 + 67*w3)
Similarly, outputs from the other node in the hidden layer is also calculated. The outputs of the two nodes in the hidden layer act as inputs to the two nodes in the output layer. This enables us to calculate output probabilities from the two nodes in output layer.

Suppose the output probabilities from the two nodes in the output layer are 0.4 and 0.6 respectively (since the weights are randomly assigned, outputs will also be random). We can see that the calculated probabilities (0.4 and 0.6) are very far from the desired probabilities (1 and 0 respectively), hence the network in Figure 5 is said to have an 'Incorrect Output'.

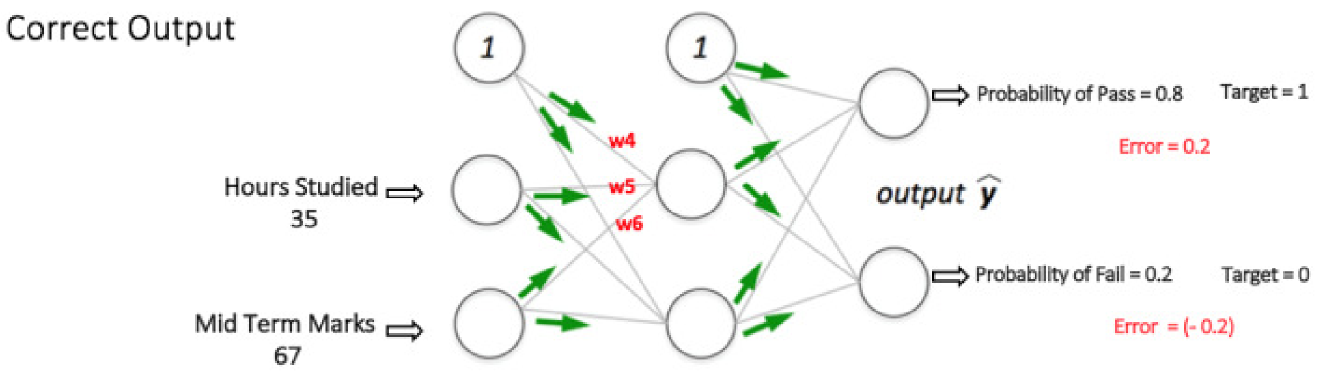


Step 2: Back Propagation and Weight Updation
We calculate the total error at the output nodes and propagate these errors back through the network using Back propagation to calculate the *gradients*. Then we use an optimization method such as *Gradient Descent* to 'adjust' **all** weights in the network with an aim of reducing the error at the output layer. This is shown in the Figure 6 below (ignore the mathematical equations in the figure for now).

Suppose that the new weights associated with the node in consideration are w4, w5 and w6 (after Backpropagation and adjusting weights).

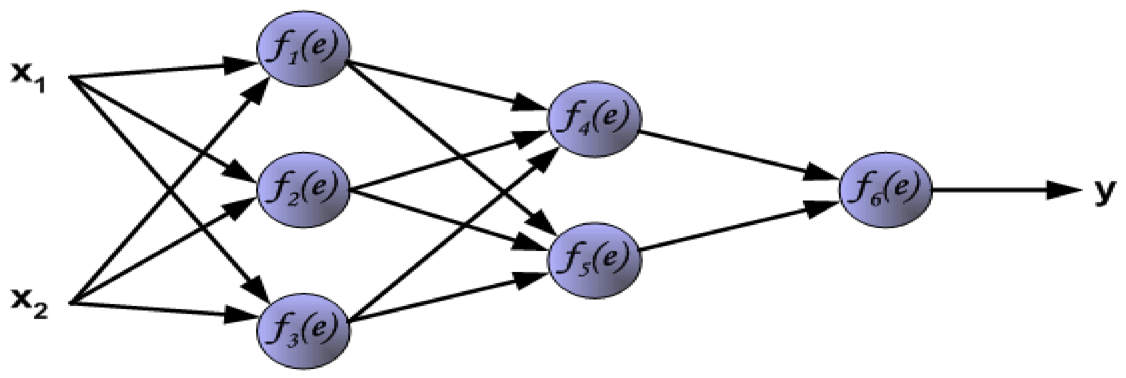

Backpropagation
+
Weights Adjusted
$\frac{\partial}{\partial w_{i,j}^{(l)}} J(W) = a_j^{(l)} \delta_i^{(l+1)}$
(compute gradient)
(error term of the output layer)
$\delta^{(3)} = a^{(3)} - y$
1
1
w4
w5
w6
output $\hat{y}$
target y
$\delta^{(2)} = (W^{(2)})^T \delta^{(3)} \cdot \frac{\partial g(z^{(2)})}{\partial z^{(2)}}$
(error term of the hidden layer)

If we now input the same example to the network again, the network should perform better than before since the weights have now been adjusted to minimize the error in prediction. As shown in Figure 7, the errors at the output nodes now reduce to [0.2, -0.2] as compared to [0.6, -0.4] earlier. This means that our network has learnt to correctly classify our first training example.

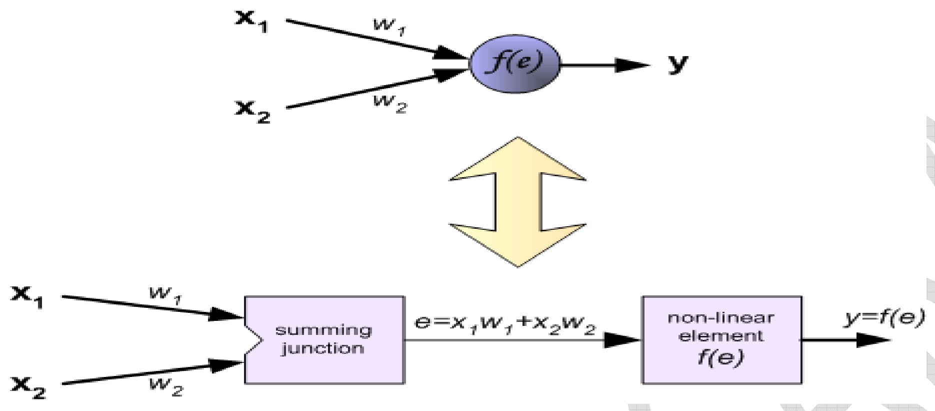


We repeat this process with all other training examples in our dataset. Then, our network is said to have *learnt* those examples.
If we now want to predict whether a student studying 25 hours and having 70 marks in the mid term will pass the final term, we go through the forward propagation step and find the output probabilities for Pass and Fail.

Principles of training multi-layer neural network using backpropagation
The project describes teaching process of multi-layer neural network employing *backpropagation* algorithm. To illustrate this process the three layer neural network with two inputs and one output,which is shown in the picture below, is used:

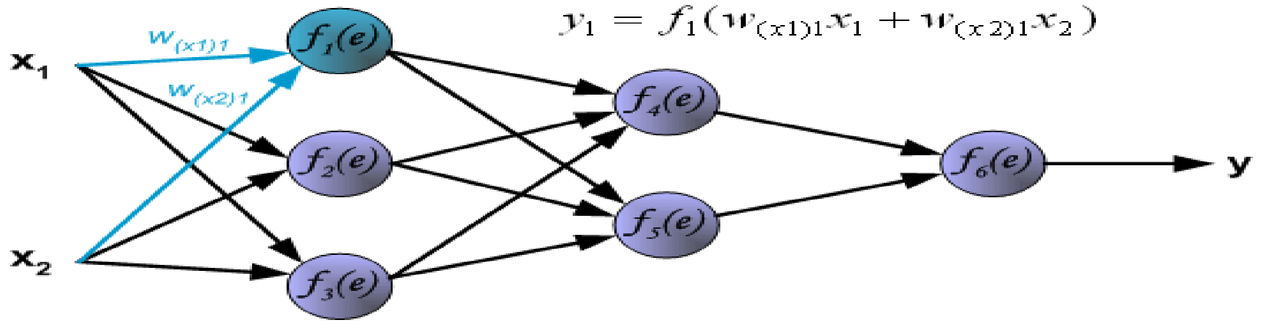


Each neuron is composed of two units. First unit adds products of weights coefficients and input signals. The second unit realise nonlinear function, called neuron activation function. Signal *e* is adder output signal, and *y = f(e)* is output signal of nonlinear element. Signal *y* is also output signal of neuron.

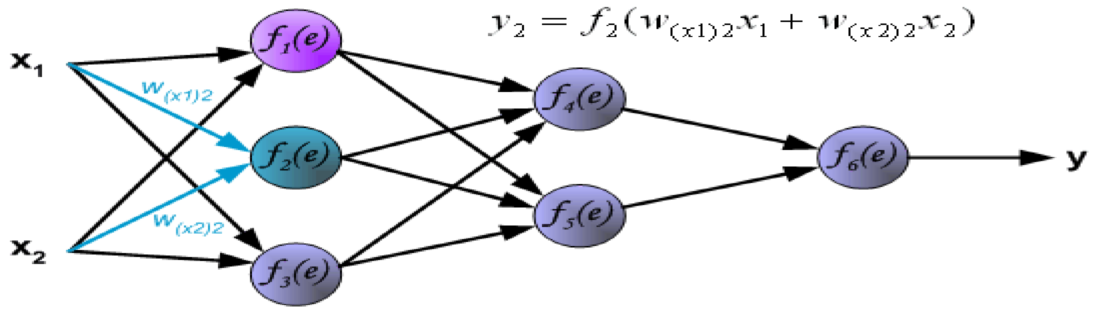


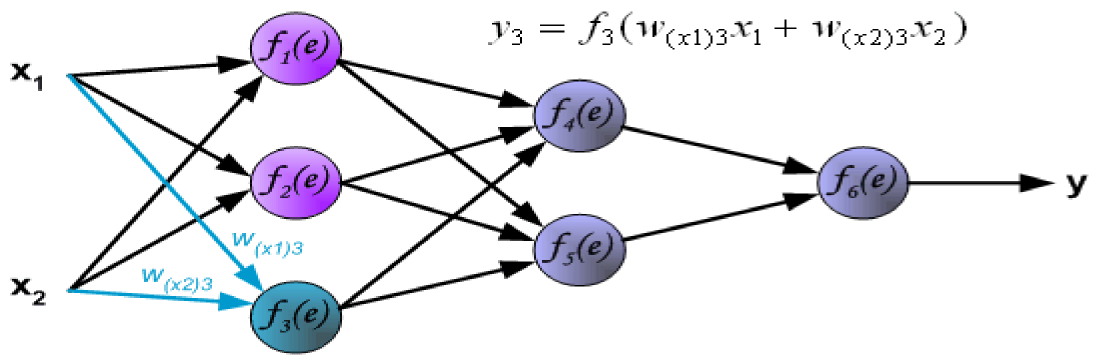
To teach the neural network we need training data set. The training data set consists of input signals (x_1 and x_2) assigned with corresponding target (desired output) *z*. The network training is an iterative process. In each iteration weights coefficients of nodes are modified using new data from training data set. Modification is calculated using algorithm described below: Each teaching step starts with forcing both input signals from training set. After this stage we can determine output signals values for each neuron in each network layer. Pictures below illustrate how signal is propagating through the network, Symbols $w_{(xm)n}$ represent weights of connections between network input x_m and neuron *n* in input layer. Symbols y_n represents output signal of neuron *n*.

$$y_1 = f_1(w_{(x1)1}x_1 + w_{(x2)1}x_2)$$

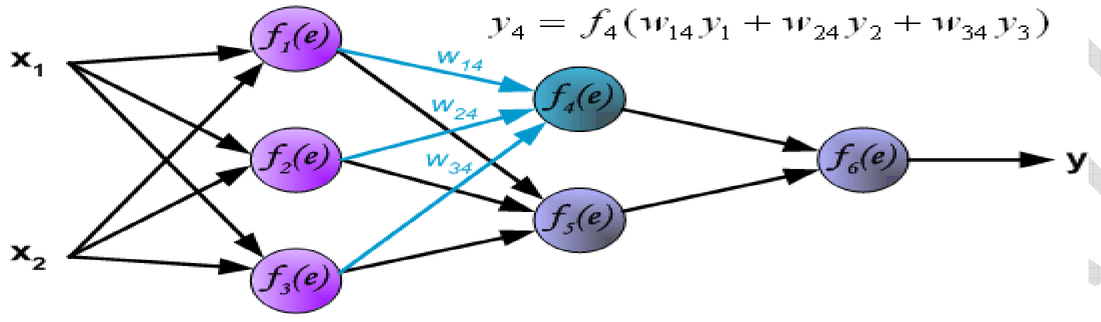


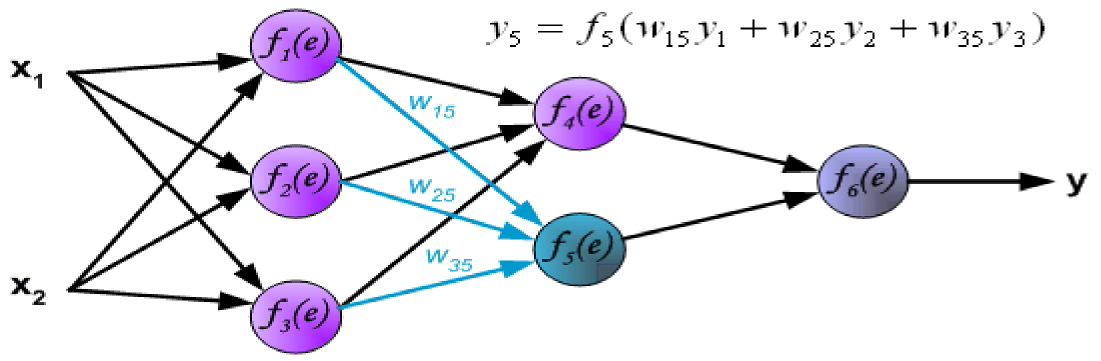
$$y_2 = f_2(w_{(x1)2}x_1 + w_{(x2)2}x_2)$$



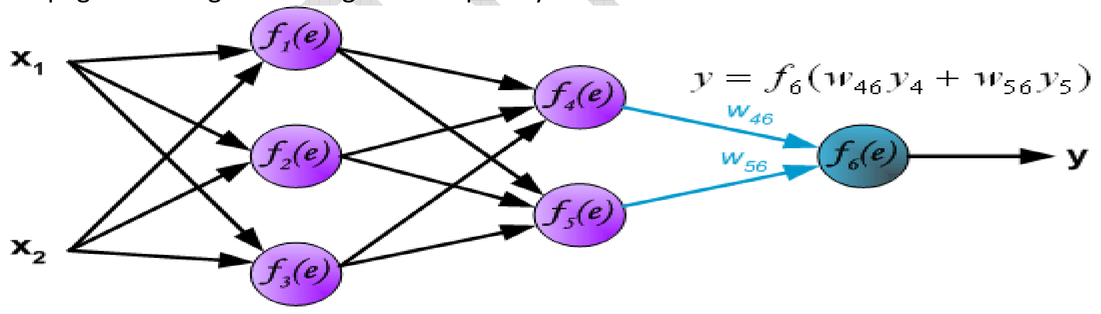


Propagation of signals through the hidden layer. Symbols w_{mn} represent weights of connections between output of neuron *m* and input of neuron *n* in the next layer.

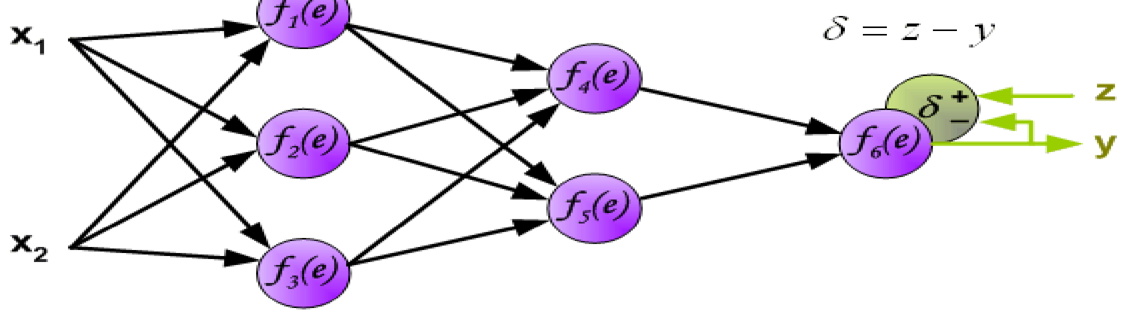




Propagation of signals through the output layer.

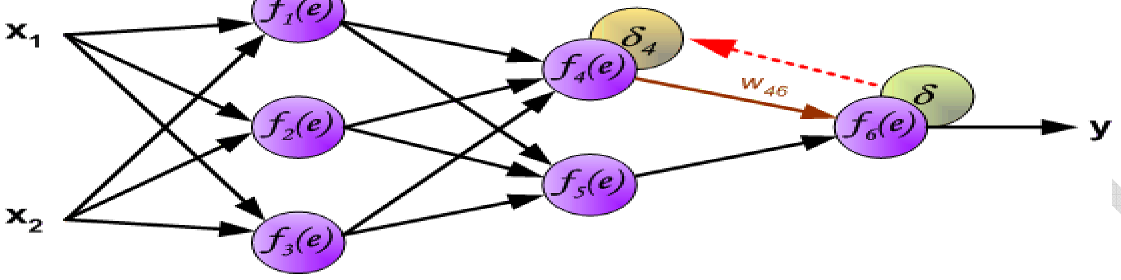


In the next algorithm step the output signal of the network *y* is compared with the desired output value (the target), which is found in training data set. The difference is called error signal *d* of output layer neuron.

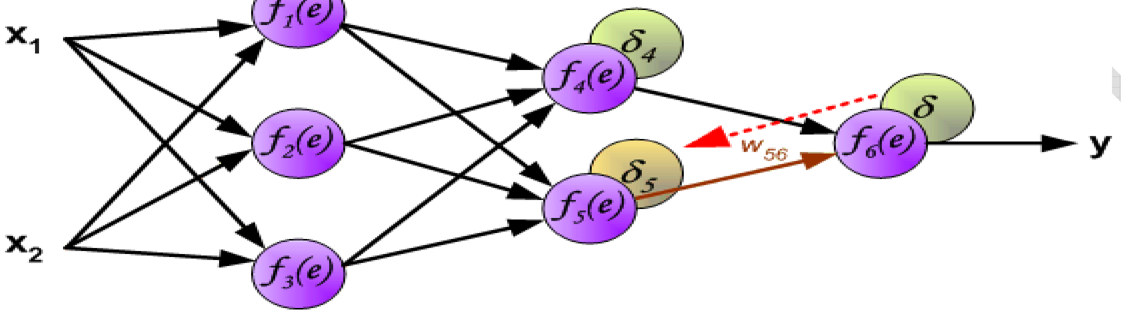


It is impossible to compute error signal for internal neurons directly, because output values of these neurons are unknown. For many years the effective method for training multiplayer networks has been unknown. Only in the middle eighties the backpropagation algorithm has been worked out. The idea is to propagate error signal *d* (computed in single teaching step) back to all neurons, which output signals were input for discussed neuron.

$$\delta_4 = w_{46}\delta$$

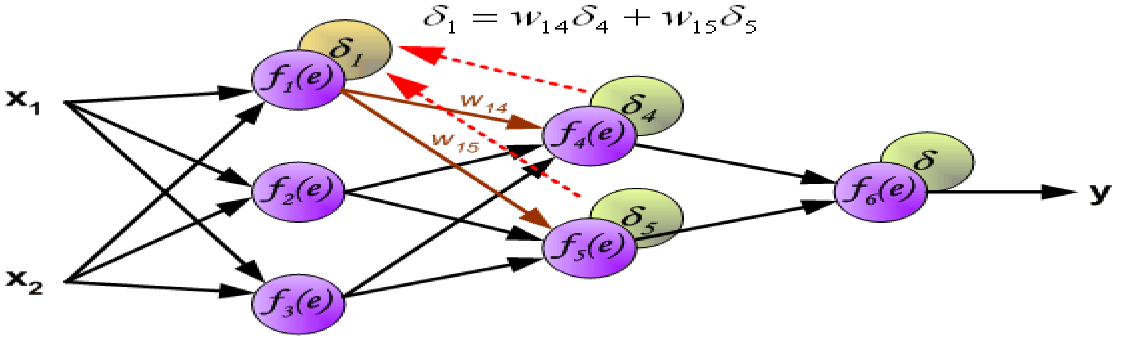


$$\delta_5 = w_{56}\delta$$

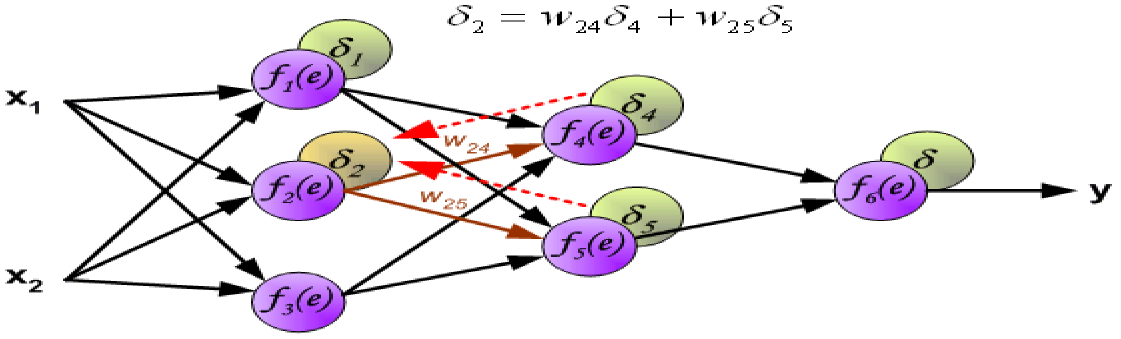


The weights' coefficients w_{mn} used to propagate errors back are equal to this used during computing output value. Only the direction of data flow is changed (signals are propagated from output to inputs one after the other). This technique is used for all network layers. If propagated errors came from few neurons they are added. The illustration is below.

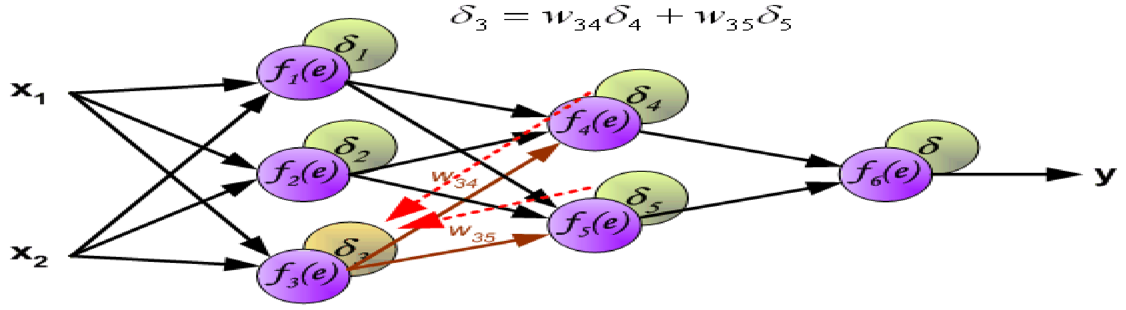
$$\delta_1 = w_{14}\delta_4 + w_{15}\delta_5$$



$$\delta_2 = w_{24}\delta_4 + w_{25}\delta_5$$



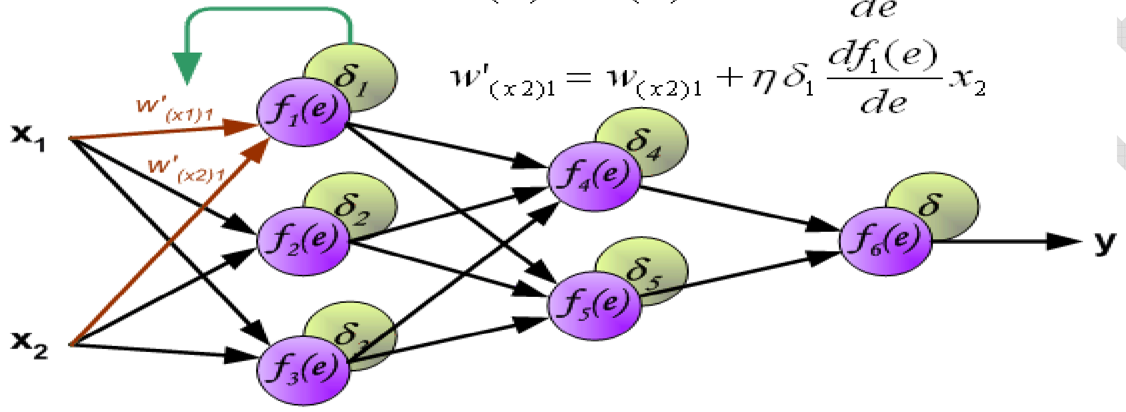
$$\delta_3 = w_{34}\delta_4 + w_{35}\delta_5$$



When the error signal for each neuron is computed, the weights coefficients of each neuron input node may be modified. In formulas below *df(e)/de* represents derivative of neuron activation function (which weights are modified).

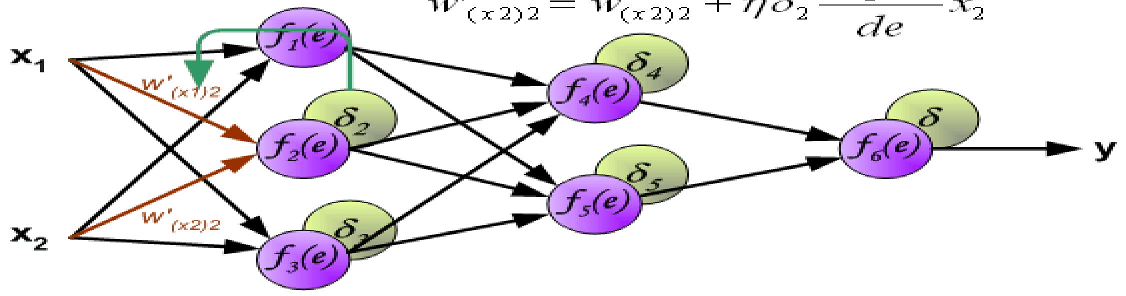
$$w'_{(x1)1} = w_{(x1)1} + \eta\delta_1 \frac{df_1(e)}{de} x_1$$

$$w'_{(x2)1} = w_{(x2)1} + \eta\delta_1 \frac{df_1(e)}{de} x_2$$



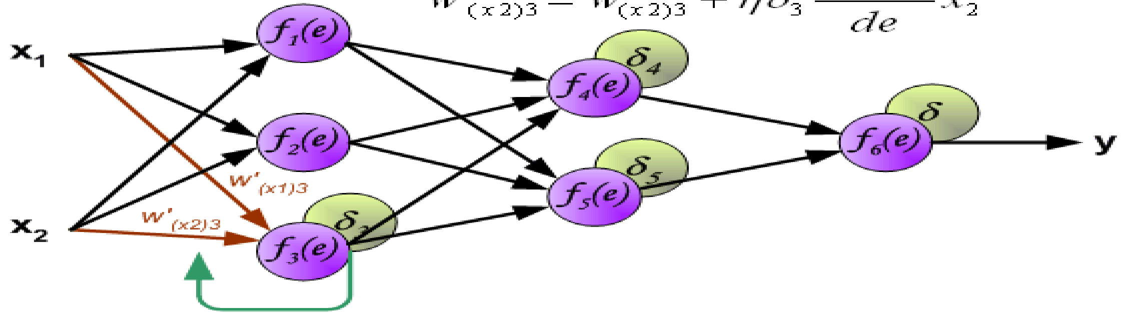
$$w'_{(x1)2} = w_{(x1)2} + \eta\delta_2 \frac{df_2(e)}{de} x_1$$

$$w'_{(x2)2} = w_{(x2)2} + \eta\delta_2 \frac{df_2(e)}{de} x_2$$



$$w'_{(x1)3} = w_{(x1)3} + \eta\delta_3 \frac{df_3(e)}{de} x_1$$

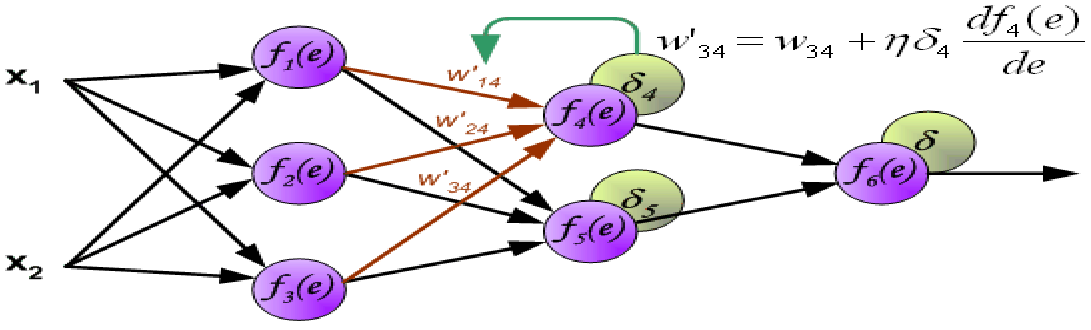
$$w'_{(x2)3} = w_{(x2)3} + \eta\delta_3 \frac{df_3(e)}{de} x_2$$



$$w'_{14} = w_{14} + \eta\delta_4 \frac{df_4(e)}{de} y_1$$

$$w'_{24} = w_{24} + \eta\delta_4 \frac{df_4(e)}{de} y_2$$

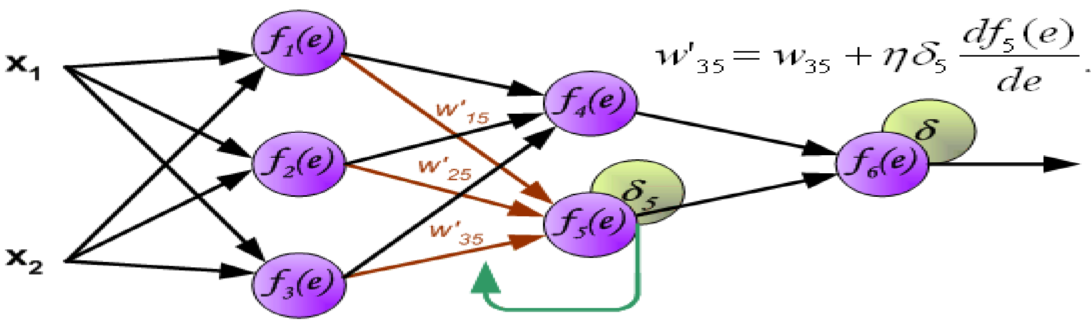
$$w'_{34} = w_{34} + \eta\delta_4 \frac{df_4(e)}{de} y_3$$



$$w'_{15} = w_{15} + \eta\delta_5 \frac{df_5(e)}{de} y_1$$

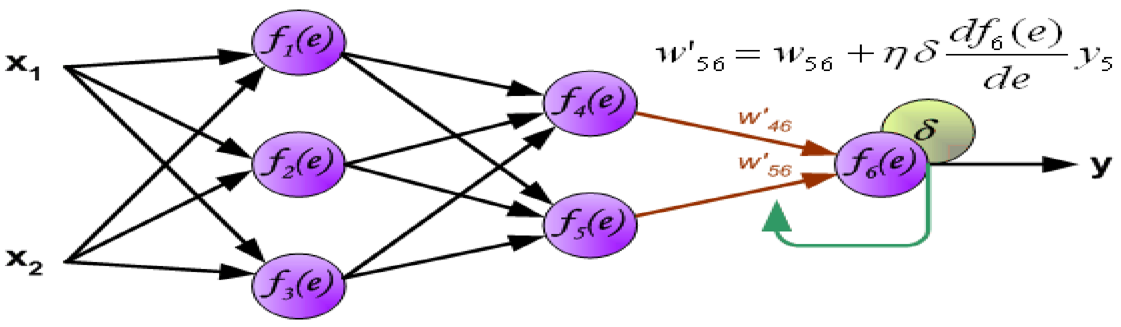
$$w'_{25} = w_{25} + \eta\delta_5 \frac{df_5(e)}{de} y_2$$

$$w'_{35} = w_{35} + \eta\delta_5 \frac{df_5(e)}{de} y_3$$

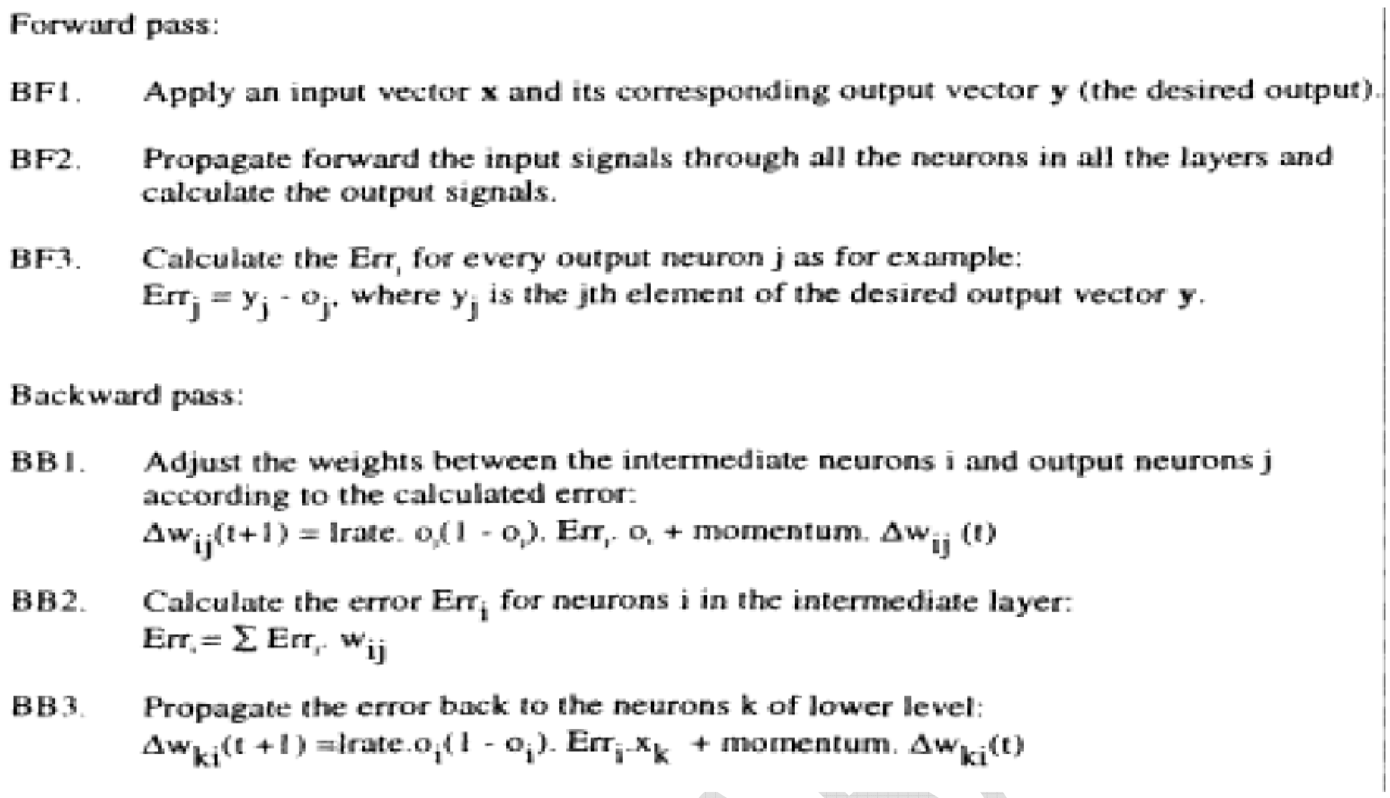


$$w'_{46} = w_{46} + \eta\delta \frac{df_6(e)}{de} y_4$$

$$w'_{56} = w_{56} + \eta\delta \frac{df_6(e)}{de} y_5$$



Coefficient *h* affects network teaching speed. There are a few techniques to select this parameter. The first method is to start teaching process with large value of the parameter. While weights coefficients are being established the parameter is being decreased gradually. The second, more complicated, method starts teaching with small parameter value. During the teaching process the parameter is being increased when the teaching is advanced and then decreased again in the final stage. Starting teaching process with low parameter value enables to determine weights coefficients signs.



Forward pass:

BF1. Apply an input vector **x** and its corresponding output vector **y** (the desired output).

BF2. Propagate forward the input signals through all the neurons in all the layers and calculate the output signals.

BF3. Calculate the Err_j for every output neuron j as for example:
$Err_j = y_j - o_j$, where y_j is the jth element of the desired output vector **y**.

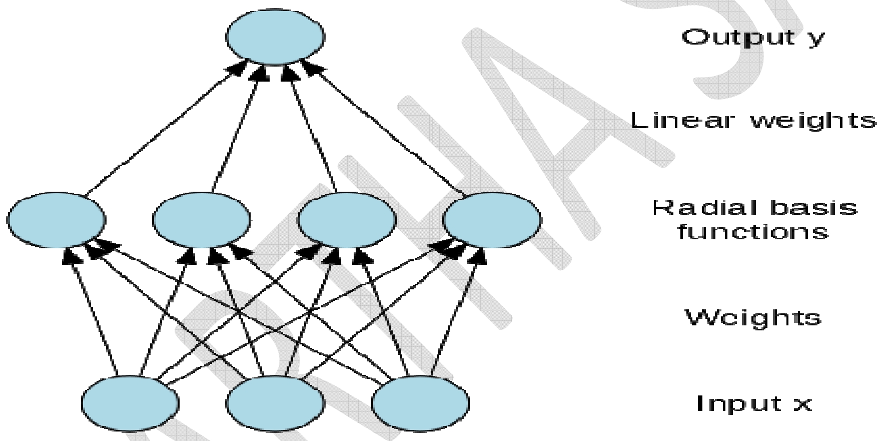
Backward pass:

BB1. Adjust the weights between the intermediate neurons i and output neurons j according to the calculated error:
$\Delta w_{ij}(t+1) = lrate.\ o_j(1 - o_j).\ Err_j.\ o_i + momentum.\ \Delta w_{ij}(t)$

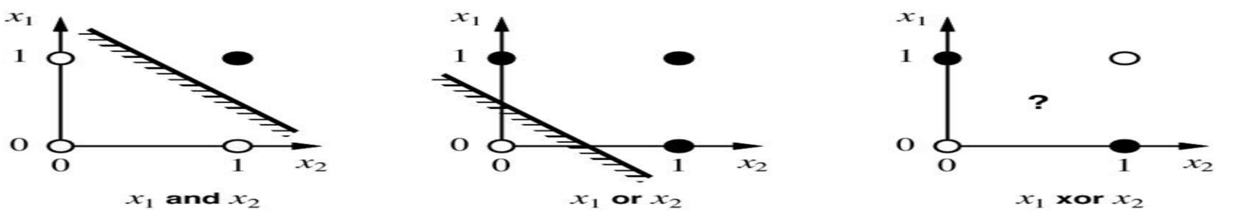
BB2. Calculate the error Err_i for neurons i in the intermediate layer:
$Err_i = \sum Err_j.\ w_{ij}$

BB3. Propagate the error back to the neurons k of lower level:
$\Delta w_{ki}(t+1) = lrate.o_i(1 - o_i).\ Err_i.x_k + momentum.\ \Delta w_{ki}(t)$

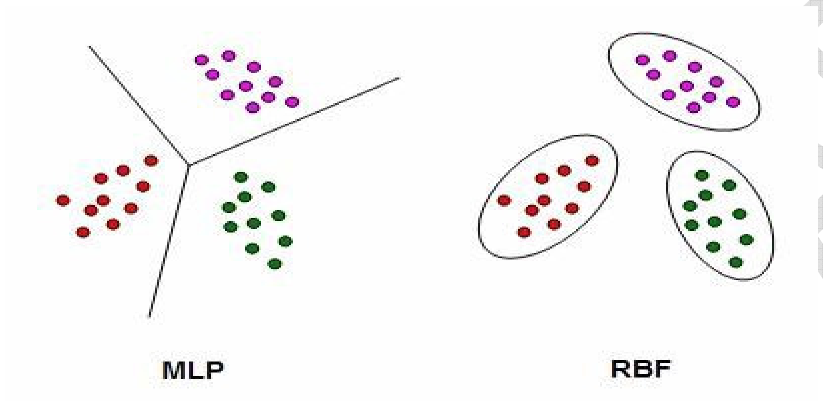
Radial Basis Functions Neural Networks



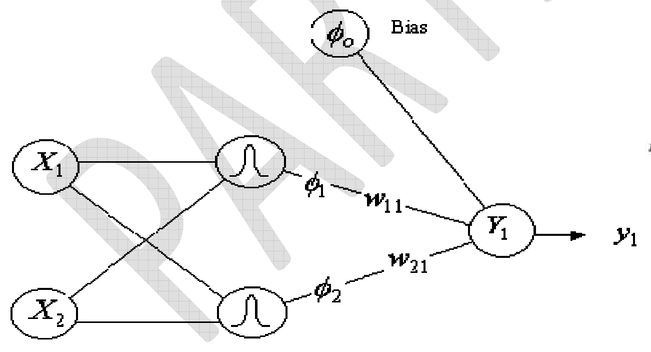
- In **Single Perceptron** / **Multi-layer Perceptron(MLP)**, we only have linear separability because they are composed of input and output layers(some hidden layers in MLP)
- For example, AND, OR functions are **linearly**-separable & XOR function is **not** linearly separable.

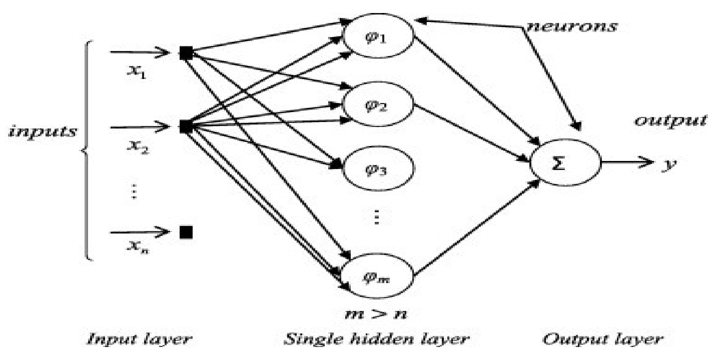


- We atleast need one hidden layer to derive a non-linearity separation.
- Our RBNN what it does is, it transforms the input signal into another form, which can be then feed into the network to get linear separability.
- RBNN is structurally same as perceptron(MLP).



- RBNN is composed of input, hidden, and output layer. RBNN is strictly limited to have exactly one hidden layer. We call this hidden layer as feature vector.
- RBNN increases dimenion of feature vector.

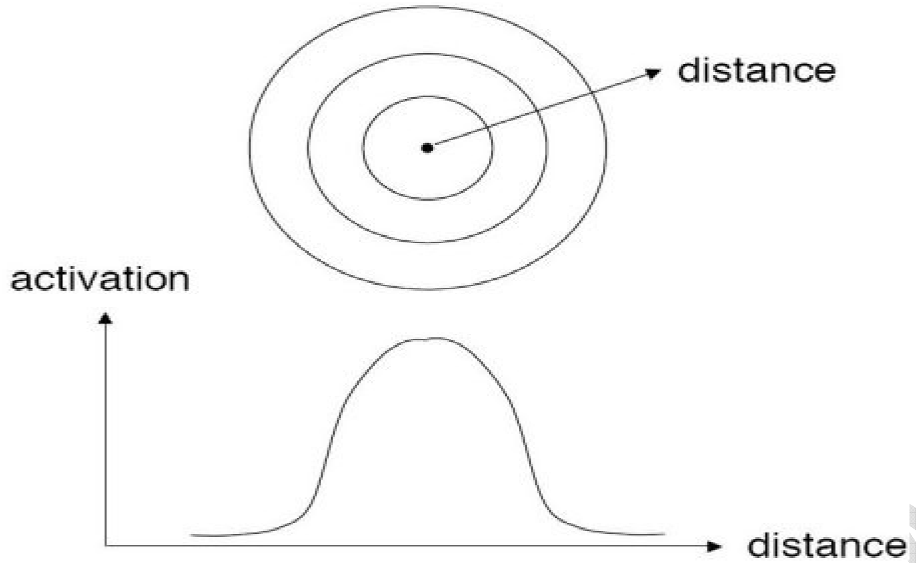




- We apply non-linear transfer function to the feature vector before we go for classification problem.
- When we increase the dimension of the feature vector, the linear separability of feature vector increases.

A non-linearity separable problem(pattern classification problem) is highly separable in high dimensional space than it is in low dimensional space.
[Cover's Theorem]
- What is a Radial Basis Function ?
- we define a receptor = t

- we draw confrontal maps around the receptor.
- Gaussian Functions are generally used for Radian Basis Function(confrontal mapping). So we define the radial distance r = ||x- t||



Gaussian Radial Function := φ(r) = exp (- $r^2/2\sigma^2$)
φ(r) = exp (- $(x-M)^2/2\sigma^2$) *where σ > 0*

$$\varphi(x) = e^{-\beta\|x-\mu\|^2}$$

$$\beta = \frac{1}{2\sigma^2}, \qquad \sigma = \frac{1}{m}\sum_{i=1}^{m}\|x_i - \mu\|$$

Where M and σ are two parameters meaning the mean and the standard deviation of the input variable x. For a particular intermediate node i, its RBFi is centered at a cluster center ci in the n-dimensional input space. The cluster center ci is represented by the vector (w1i . . . ,wni) of connection weights between the n input nodes and the hidden node i. The standard deviation for this cluster defines the range for the RBFi. The RBF is nonmonotonic, in contrast to the sigmoid function. The second layer is connected to the output layer. The output nodes perform a simple

summation function with a linear threshold activation function. Training of an RBFN consists of two phases: (1) adjusting the RBF of the hidden neurons by applying a statistical clustering method; this represents an unsupervised learning phase; (2) applying gradient descent (e.g., the back propagation algorithm) or a linear regression algorithm for adjusting the second layer of connections; this is a Supervised learning phase. During training, the following parameters of the RBFN are adjusted

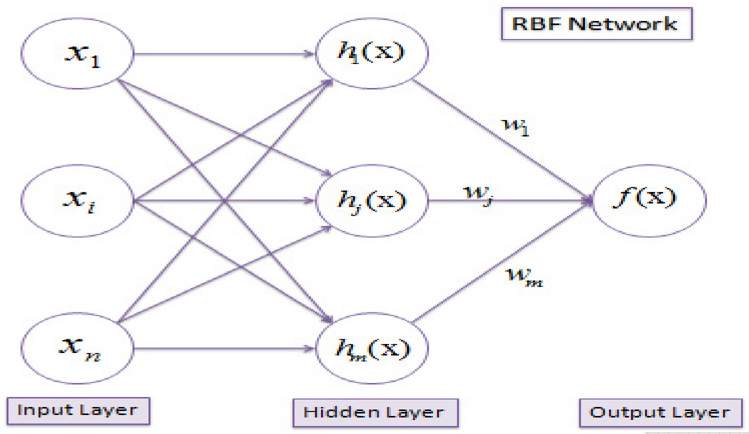
- The n-dimensional position of the centers Ci of the RBFi. This can be achieved by using the kmeans clustering algorithm ; the algorithm finds k (number of hidden nodes) cluster centers which minimize the average distance between the training examples and the nearest centers.
- The deviation scaling parameter **σ** i for every RBFi; it is defined by using average distance to the nearest m-cluster centers: ***"the squared sum of the distances between the respective receptor & the each cluster nearest samples"***

$$\sigma = \frac{1}{m}\sum_{i=1}^{m}\|x_i - \mu\|$$

- The weights of the second layer connections.
  The recall procedure finds through the functions RBFi how close an input vector x' is to the centers ci and then propagates these values to the output layer.
  The following advantages of the RBFN over the MLP with the back propagation algorithm have been

experimentally and theoretically proved:

- Training in RBFNs is an order of magnitude faster than training of a comparably sized feedforward network with the back propagation algorithm.
- A better generalization is achieved in RBFNs.
- RBFNs have very fast convergence properties compared with the conventional multilayer networks with sigmoid transfer functions, since any function can be approximated by a linear combination of locally tuned factorizable basis functions.
- There is no local minima problem.
- The RBF model can be interpreted as a fuzzy connectionist model, as the RBFs can be considered as membership functions.
- The hidden layer has a much clearer interpretation than the MLP with the back propagation algorithm. It is easier to explain what an RBF network has learned than its counterpart MLP with the back propagation algorithm



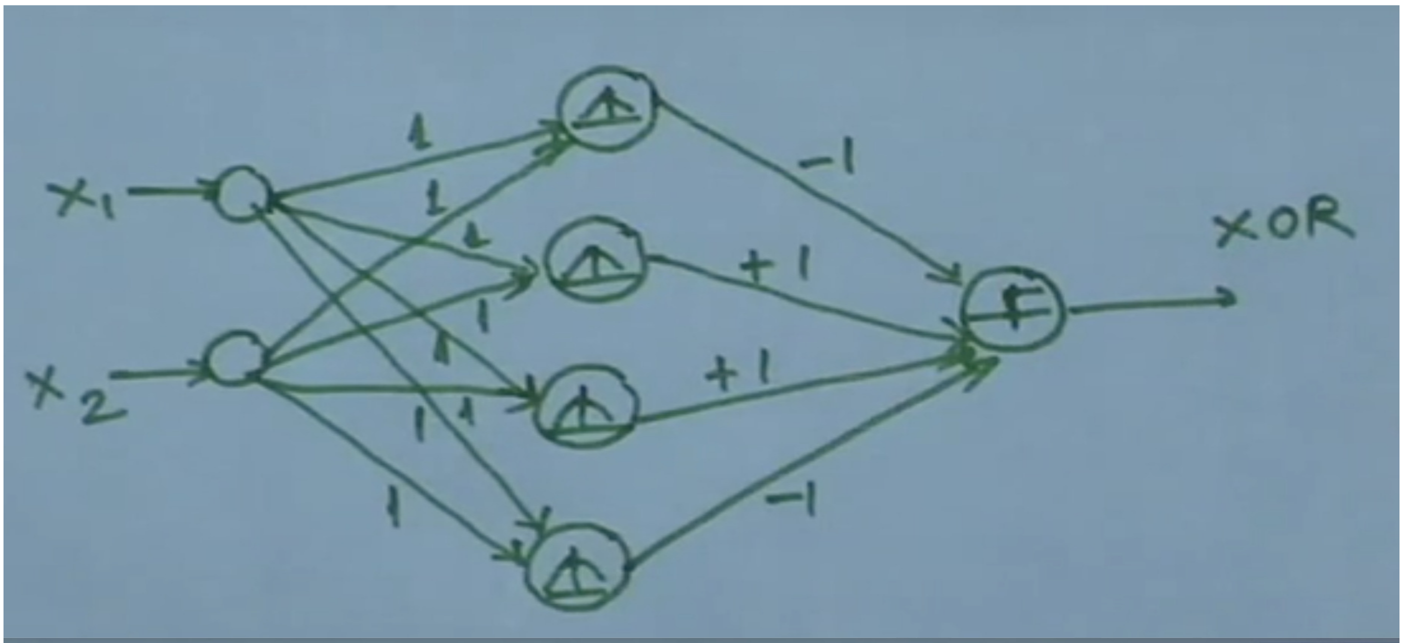
$$f(x) = \sum_{j=1}^{m} w_j h_j(x)$$

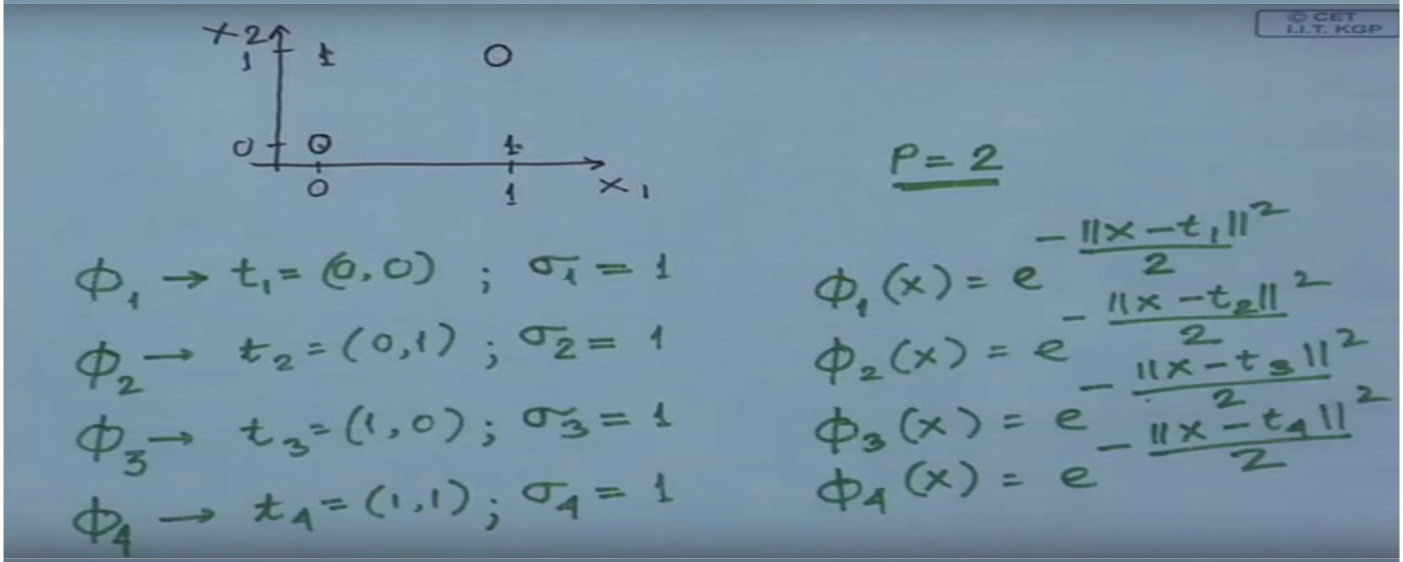
$$h(x) = \exp\left(- \frac{(x-c)^2}{r^2} \right)$$

- Example. XOR function :-

- I have 4 inputs and I will not increase dimension at the feature vector here. So I will select 2 receptors here. For each transformation function φ(x), we will have each receptors t.
- Now consider the RBNN architecture,

- P := # of input features/ values.
- M = # of transformed vector dimensions (hidden layer width). So M ≥ P usually be.
- Each node in the hidden layer, performs a set of non-linear radian basis function.
- Output C will remains the same as for the classification problems(certain number of class labels as predefined).




x_2
1
0
0
1
x_1
1
0
0
1
© CET I.I.T. KGP

$P = 2$

$\phi_1 \rightarrow t_1 = (0,0)\ ;\ \sigma_1 = 1$

$\phi_2 \rightarrow t_2 = (0,1)\ ;\ \sigma_2 = 1$

$\phi_3 \rightarrow t_3 = (1,0)\ ;\ \sigma_3 = 1$

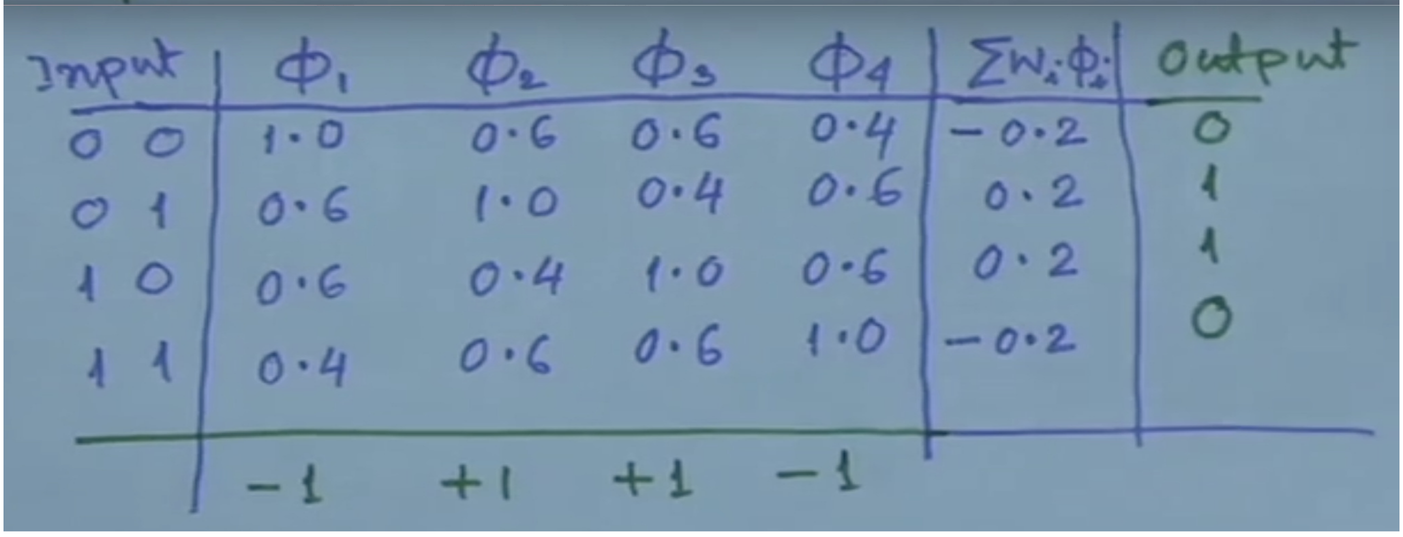
$\phi_4 \rightarrow t_4 = (1,1)\ ;\ \sigma_4 = 1$

$$\phi_1(x) = e^{-\frac{\|x - t_1\|^2}{2}}$$

$$\phi_2(x) = e^{-\frac{\|x - t_2\|^2}{2}}$$

$$\phi_3(x) = e^{-\frac{\|x - t_3\|^2}{2}}$$

$$\phi_4(x) = e^{-\frac{\|x - t_4\|^2}{2}}$$



Input	ϕ_1	ϕ_2	ϕ_3	ϕ_4	$\sum w_i \phi_i$	Output
0 0	1.0	0.6	0.6	0.4	-0.2	0
0 1	0.6	1.0	0.4	0.6	0.2	1
1 0	0.6	0.4	1.0	0.6	0.2	1
1 1	0.4	0.6	0.6	1.0	-0.2	0
	-1	+1	+1	-1		

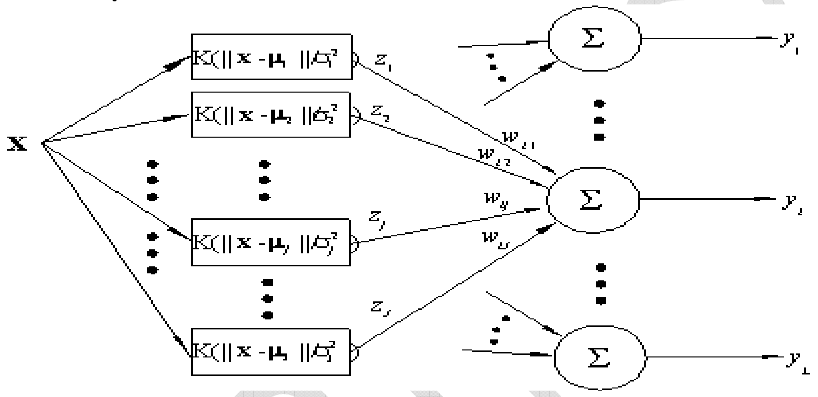
- Only Nodes in the hidden layer perform the radian basis transformation function.
- Output layer performs the linear combination of the outputs of the hidden layer to give a final probabilistic value at the output layer.
- So the classification is only done only @ ***(hidden layer → output layer)***

Training the RBNN :-

- - **First**, we should train the **hidden layer** using **back propagation.**
- Neural Network training(back propagation) is a **curve fitting method**. It **fits** a **non-linear curve** during the **training** phase. It runs through stochastic approximation, which we call the back propagation.
- For each of the node in the hidden layer, we have to find **t**(receptors) & the variance (**σ**)[variance — the spread of the radial basis function]
- On the **second** training phase, we have to **update** the **weighting vectors** between **hidden layers & output layers.**
- In hidden layers, **each** node represents **each** transformation basis function. **Any** of the function could satisfy the non-linear separability OR even **combination** of set of functions could satisfy the non-linear separability.
- So in our hidden layer transformation, all the non-linearity terms are included. Say like $X^2 + Y^2 + 5XY$; its all included in a hyper-surface equation(X & Y are inputs).
- Therefore, the first stage of training is done by **clustering algorithm.** We define the **number of cluster centers** we need. And by clustering algorithm, we compute the cluster centers, which then is assigned as the **receptors** for each hidden neurons.
- I have to cluster N samples or observations into M clusters (N > M).
- So the output "clusters" are the "receptors".
- for each receptors, I can find the variance as "**the squared sum of the distances between the respective receptor & the each cluster nearest samples**" :=

$$\sigma = \frac{1}{m}\sum_{i=1}^{m}\|x_i - \mu\|$$

- The interpretation of the first training phase is that the **"feature vector is projected onto the transformed space".**



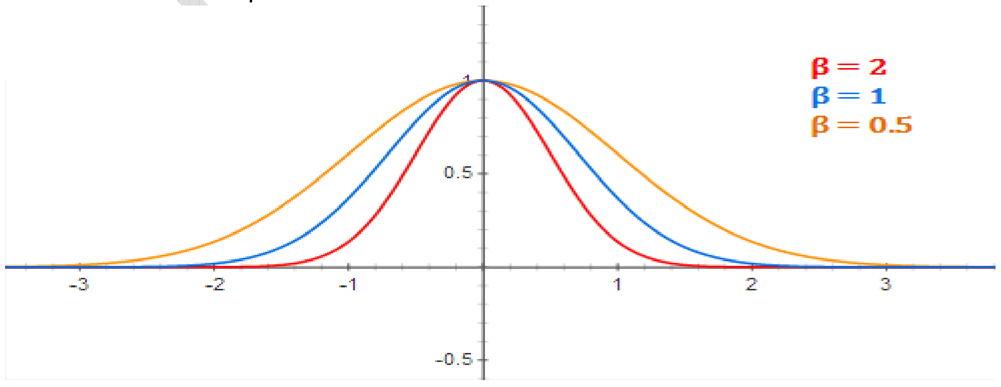
RBF Neuron Width

Recall that each RBF neuron applies a Gaussian to the input. We all know from studying bell curves that an important parameter of the Gaussian is the standard deviation–it controls how wide the bell is. That same parameter exists here with our RBF neurons, you'd probably just interperet it a little differently. It still controls the width of the Gaussian, which means it determines how much of the input space the RBF neuron will respond to.

For RBFNs, instead of talking about the standard deviation ('sigma') directly, we use the related value 'beta':

$$\beta = \frac{1}{2\sigma^2}$$

Here are some examples of how different beta values affect the width of the Gaussian.



Advantages of using RBNN than the MLP :-

1. Training in RBNN is faster than in Multi-layer Perceptron (MLP) → takes many interactions in MLP.
2. We can easily interpret what is the meaning / function of the each node in hidden layer of the RBNN. This is difficult in MLP.
3. (what should be the # of nodes in hidden layer & the # of hidden layers) this parameterization is difficult in MLP. But this is not found in RBNN.
4. Classification will take more time in RBNN than MLP.

Unsupervised Learning in Neural Networks

Two principle learning rules are implemented in the contemporary unsupervised learning algorithms for neural networks: (1) non competitive learning, that is, many neurons may be activated at a time; and (2) competitive learning, that is, the neurons compete and after that, only one is activated at one time, e.g. only one wins after the competition. This principle is also called "winner takes all."

$$w_{ij}(t+1) = w_{ij}(t) + c \cdot o_i \cdot o_j,$$

differential Hebbian learning law (Kosko 1988(A)): $w_{ij}(t+1) = w_{ij}(t) + c \cdot o_i \cdot o_j + \Delta o_i \cdot \Delta o_j$

Grossberg's competitive law (Grossberg 1982), expressed as: $\Delta w_{ij} = c \cdot o_j \cdot (o_i \cdot w_{ij}),$

The differential competitive learning law (Kosko 1990): $\Delta w_{ij} = c \cdot \Delta o_j (o_i - w_{ij})$

K-means Clustering Algorithm

***k*-means clustering** is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. K-means is one of the most popular clustering algorithm in which we use the concept of partition procedure. We start with an initial partition and repeatedly move patterns from one cluster to another, until we get a satisfactory result.

- Let the set of data points D be {x1 , x2 , …, xn }, where xi = (xi1 , xi2 R⊆, …, xir) is a vector in X r , and r is the number of dimensions.
- The k-means algorithm partitions the given data into k clusters:
  – Each cluster has a cluster center, called centroid.
  – k is specified by the user

- Given k, the k-means algorithm works as follows:
  1. Choose k (random) data points (seeds) to be the initial centroids, cluster centers
  2. Assign each data point to the closest centroid
  3. Re-compute the centroids using the current cluster memberships
  4. If a convergence criterion is not met, repeat steps 2 and 3
- no (or minimum) re-assignments of data points to different clusters, or
- no (or minimum) change of centroids, or
- minimum decrease in the sum of squared error (SSE),

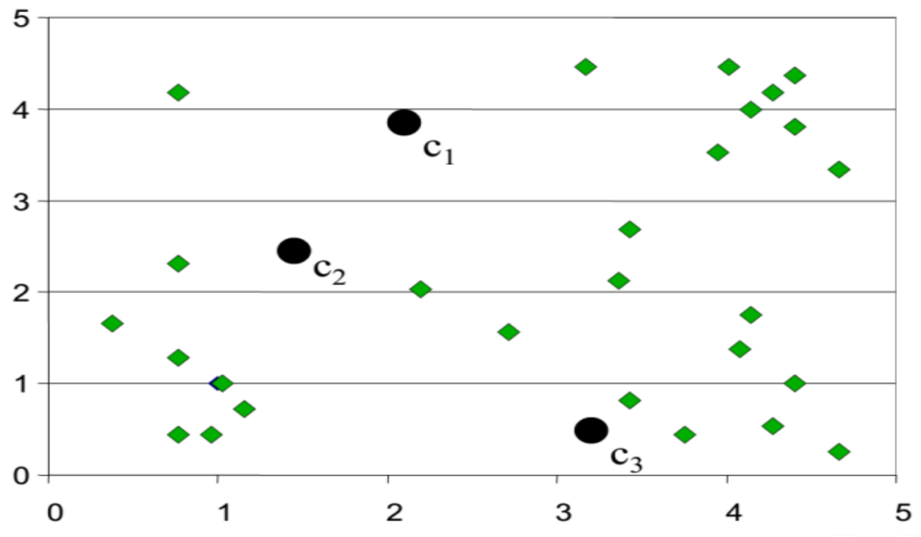
$$SSE = \sum_{j=1}^{k} \sum_{\mathbf{x} \in C_j} d(\mathbf{x}, \mathbf{m}_j)^2$$

– Cj is the jth cluster,
– mj is the centroid of cluster Cj (the mean vector of all the data points in Cj),
– d(x, mj) is the (Eucledian) distance between data point x and centroid mj

K-means clustering example:

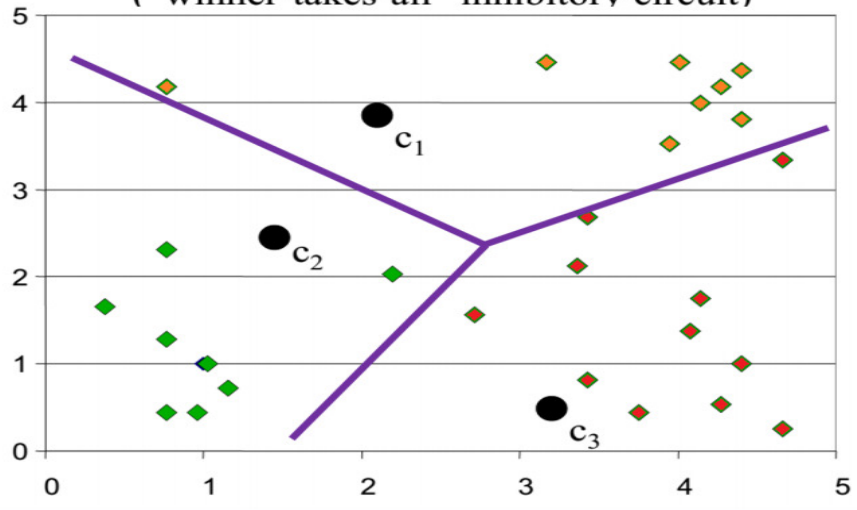
step 1

Randomly initialize the cluster centers (synaptic weights)



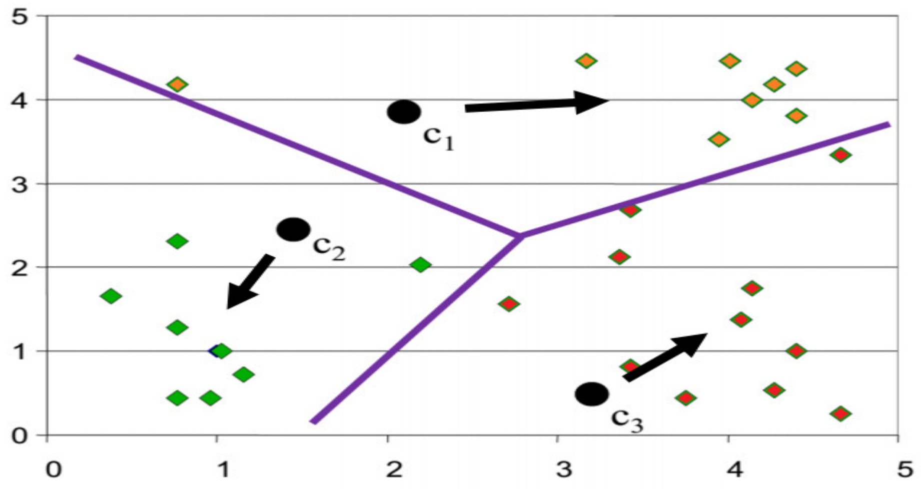
step 2

Determine cluster membership for each input ("winner-takes-all" inhibitory circuit)

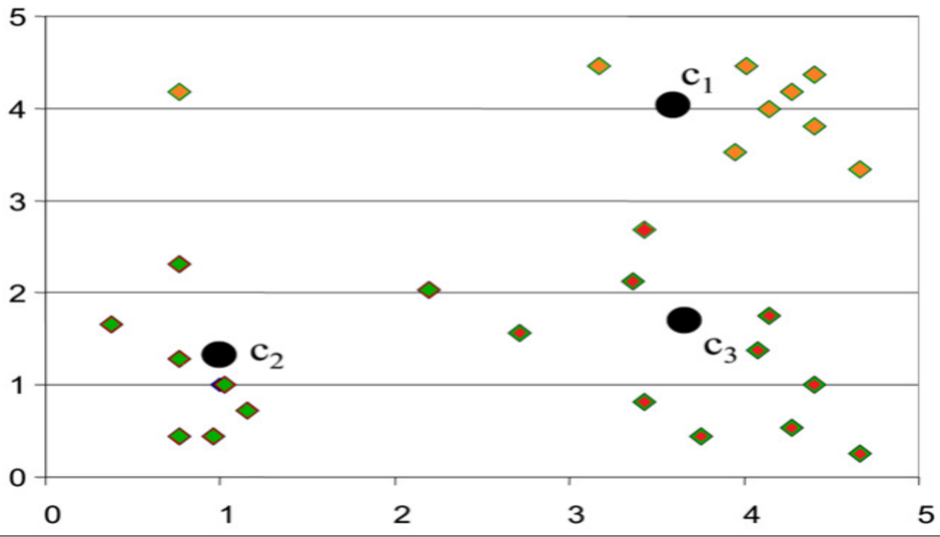


step 3

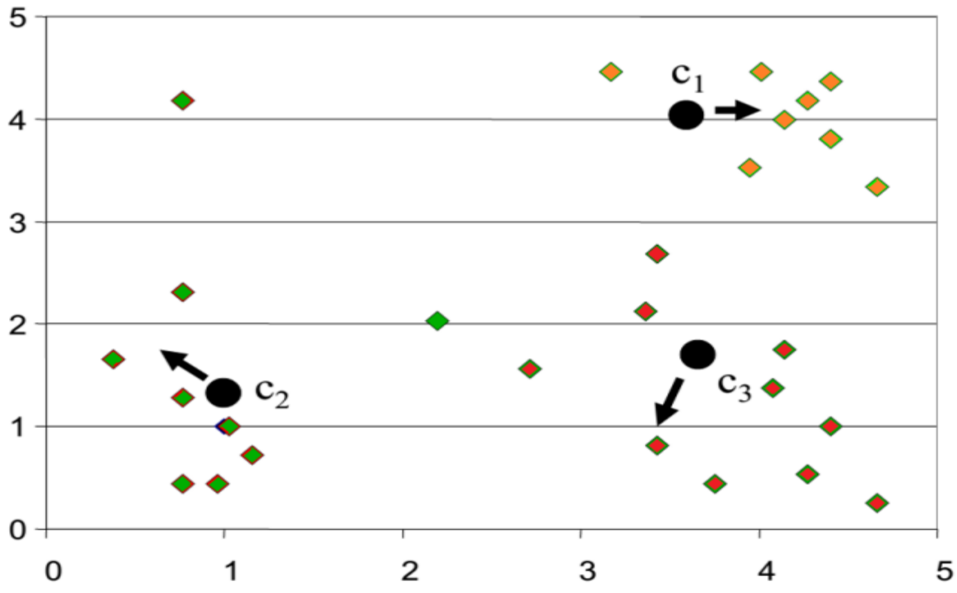
Re-estimate cluster centers (adapt synaptic weights)

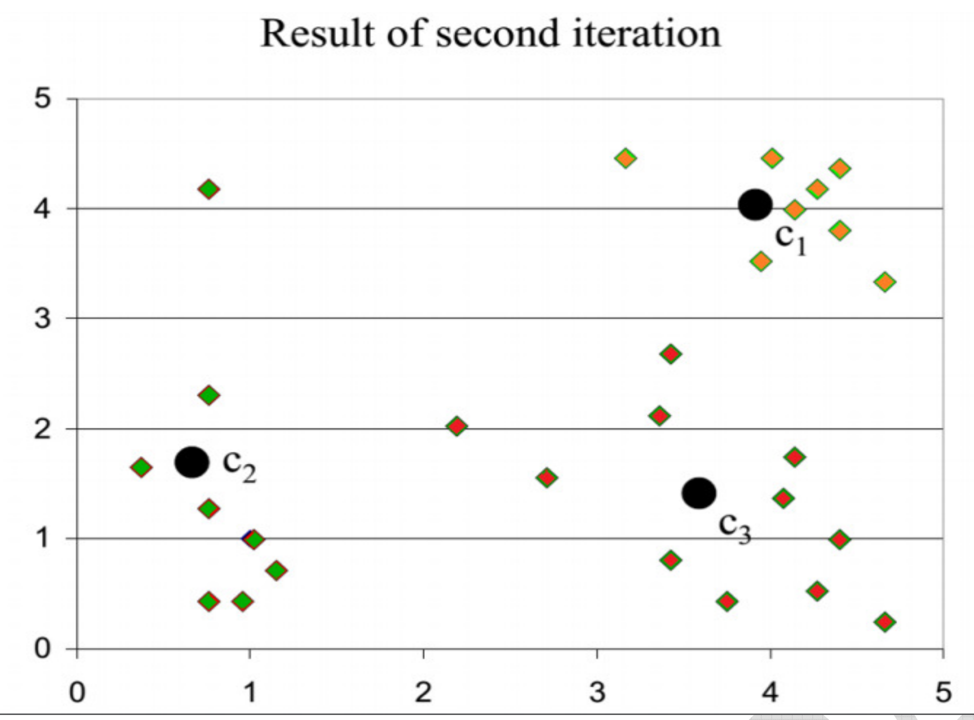


Result of first iteration



Second iteration

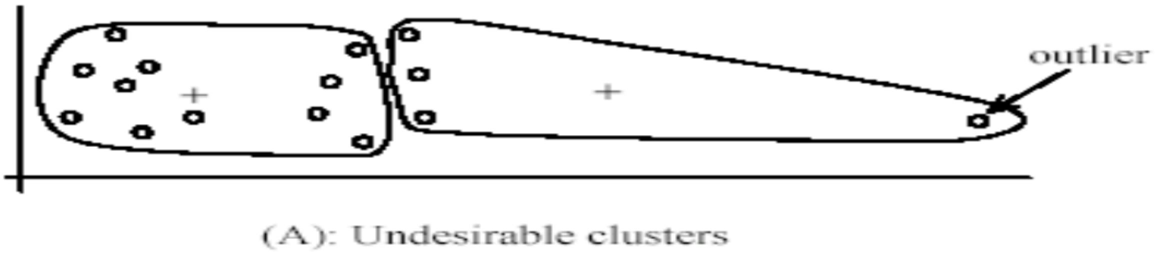




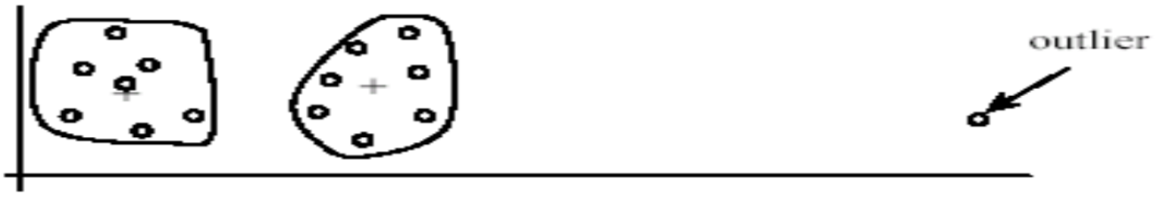
Why use K-means?

- Strengths:
  - Simple: easy to understand and to implement
  - Efficient: Time complexity: O(tkn),
    where n is the number of data points,
    k is the number of clusters,
    and t is the number of iterations.
  - Since both k and t are small. k-means is considered a linear algorithm.
- K-means is the most popular clustering algorithm.
- Note that: it terminates at a local optimum if SSE is used. The global optimum is hard to find due to complexity.
- The algorithm is only applicable if the mean is defined.
  – For categorical data, k-mode - the centroid is represented by most frequent values.
- The user needs to specify k.
- The algorithm is sensitive to outliers
  – Outliers are data points that are very far away from other data points.
  – Outliers could be errors in the data recording or some special data points with very different values.

Outliers



(A): Undesirable clusters



(B): Ideal clusters

Dealing with outliers

- Remove some data points that are much further away from the centroids than other data points
  - To be safe, we may want to monitor these possible outliers over a few iterations and then decide to remove them.
- Perform random sampling: by choosing a small subset of the data points, the chance of selecting an outlier is much smaller
  - Assign the rest of the data points to the clusters by distance or similarity comparison, or classification

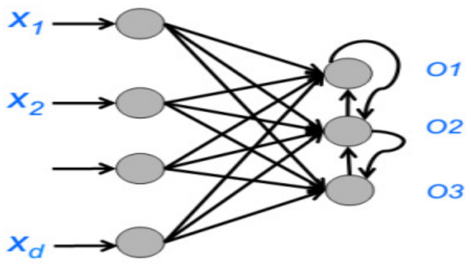
K-means summary

- The k-means algorithm is not suitable for discovering clusters that are not hyper-ellipsoids (or hyper-spheres).
- Despite weaknesses, k-means is still the most popular algorithm due to its simplicity and efficiency.
- No clear evidence that any other clustering algorithm performs better in general .
- Comparing different clustering algorithms is a difficult task. No one knows the correct clusters!

Competitive learning

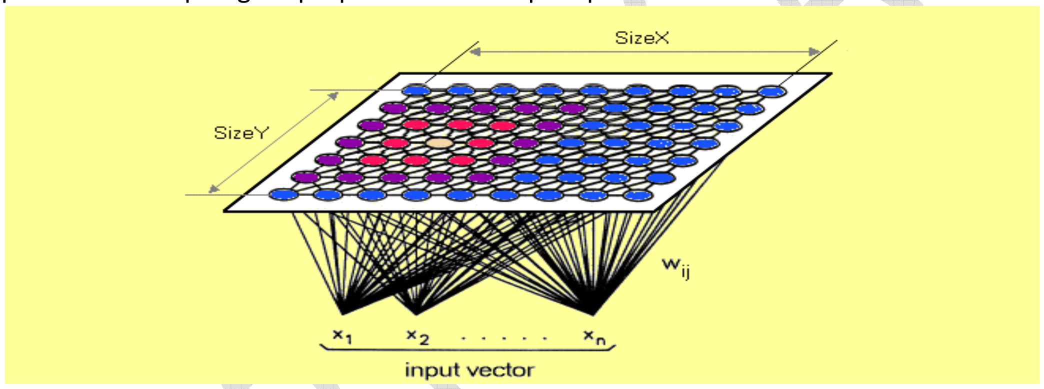
A form of unsupervised training where output units are said to be in competition for input patterns

- During training, the output unit that provides the highest activation to a given input pattern is declared the winner and is moved closer to the input pattern, whereas the rest of the neurons are left unchanged
- This strategy is also called <u>winner-take-all</u> since only the winning neuron is updated
  - Output units may have lateral inhibitory connections so that a winner neuron can inhibit others by an amount proportional to its activation level



Kohonen Self Organization Maps (K-SOM)

A **self-organizing map** (**SOM**) is a type of artificial neural network (ANN) that is trained using unsupervised learning to produce a low-dimensional (typically two-dimensional), discretized representation of the input space of the training samples, called a **map**, and is therefore a method to do dimensionality reduction. Self-organizing maps differ from other artificial neural networks as they apply competitive learning as opposed to error-correction learning (such as back propagation with gradient descent), and in the sense that they use a neighbour hood function to preserve the topological properties of the input space.



SOM was introduced by Finnish professor Teuvo Kohonen in the 1980s is sometimes called a **Kohonen map.**

What really happens in SOM ?

Each data point in the data set recognizes themselves by competeting for representation. SOM mapping steps starts from initializing the weight vectors. From there a sample vector is selected randomly and the map of weight vectors is searched to find which weight best represents that sample. Each weight vector has neighboring weights that are close to it. The weight that is chosen is rewarded by being able to become more like that randomly selected sample vector. The neighbors of that weight are also rewarded by being able to become more like the chosen sample vector. This allows the map to grow and form different shapes. Most generally, they form square/rectangular/hexagonal/L shapes in 2D feature space.

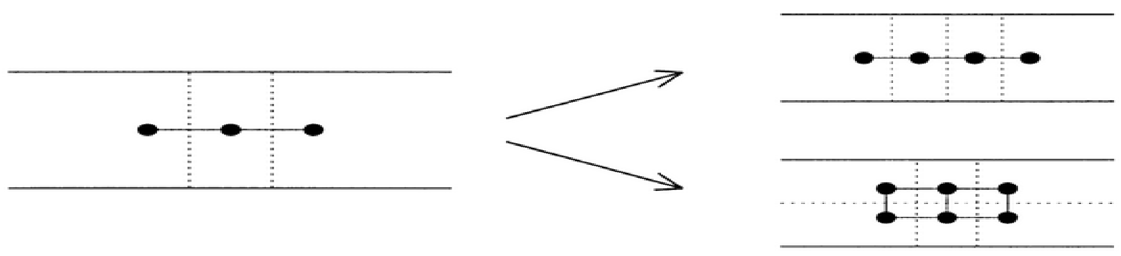
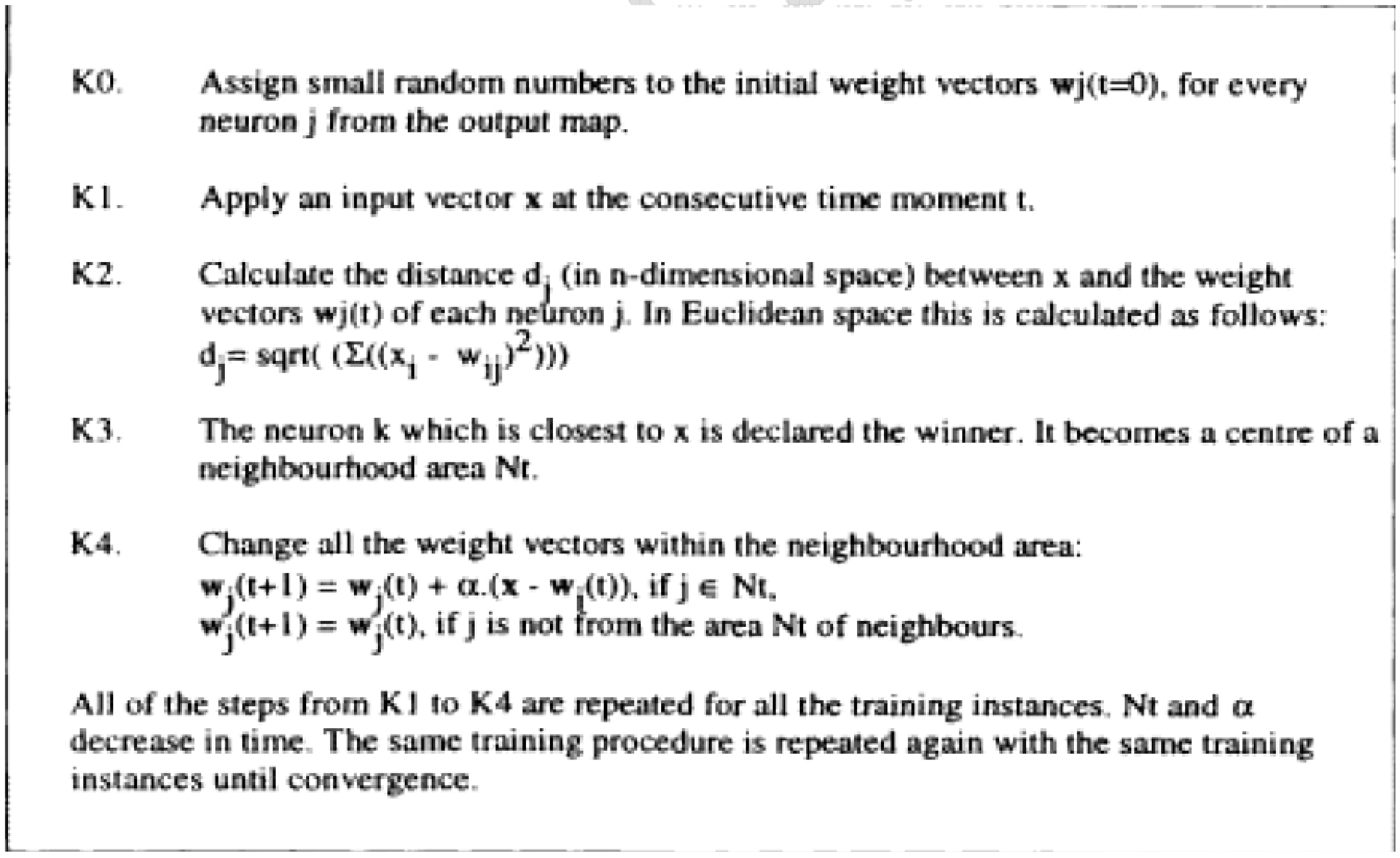


Fig. 1. Illustration of the basic decision to be made during the growth procedure: an output space lattice can either grow in an existing direction or it can open a new direction.

The Algorithm:

1. Each node's weights are initialized.
2. A vector is chosen at random from the set of training data.
3. Every node is examined to calculate which one's weights are most like the input vector. The winning node is commonly known as the **Best Matching Unit** (BMU).
4. Then the neighbourhood of the BMU is calculated. The amount of neighbors decreases over time.
5. The winning weight is rewarded with becoming more like the sample vector. The nighbors also become more like the sample vector. The closer a node is to the BMU, the more its weights get altered and the farther away the neighbor is from the BMU, the less it learns.
6. Repeat step 2 for N iterations.

Best Matching Unit is a technique which calculates the distance from each weight to the sample vector, by running through all weight vectors. The weight with the shortest distance is the winner. There are numerous ways to determine the distance, however, the most commonly used method is the *Euclidean Distance, and that's what is used in the following implementation.*


K0. Assign small random numbers to the initial weight vectors wj(t=0), for every neuron j from the output map.

K1. Apply an input vector **x** at the consecutive time moment t.

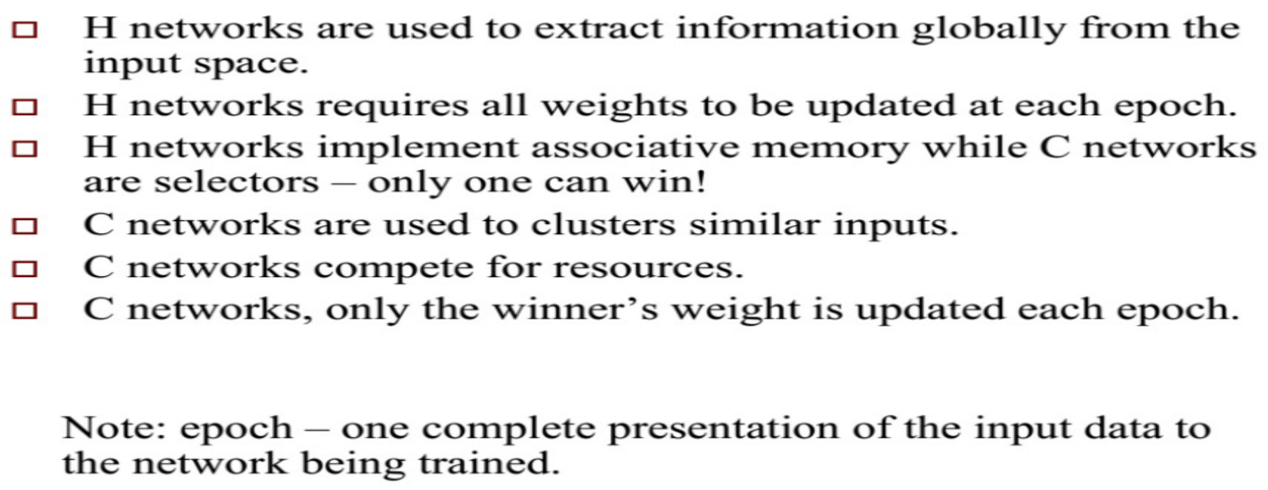
K2. Calculate the distance d_j (in n-dimensional space) between **x** and the weight vectors wj(t) of each neuron j. In Euclidean space this is calculated as follows:
$d_j = \text{sqrt}((\Sigma((x_i - w_{ij})^2)))$

K3. The neuron k which is closest to **x** is declared the winner. It becomes a centre of a neighbourhood area Nt.

K4. Change all the weight vectors within the neighbourhood area:
$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \alpha.(\mathbf{x} - \mathbf{w}_j(t))$, if $j \in$ Nt,
$\mathbf{w}_j(t+1) = \mathbf{w}_j(t)$, if j is not from the area Nt of neighbours.

All of the steps from K1 to K4 are repeated for all the training instances. Nt and α decrease in time. The same training procedure is repeated again with the same training instances until convergence.

<u>Hebbian vs. Competitive learning</u>



- H networks are used to extract information globally from the input space.
- H networks requires all weights to be updated at each epoch.
- H networks implement associative memory while C networks are selectors – only one can win!
- C networks are used to clusters similar inputs.
- C networks compete for resources.
- C networks, only the winner's weight is updated each epoch.

Note: epoch – one complete presentation of the input data to the network being trained.

ART (ADAPTIVE RESONANCE THEORY) NETWORK :

This network was developed by Stephen Grossberg and Gail Carpenter in 1987. It is based on competition and uses unsupervised learning model. Adaptive Resonance Theory (ART) networks, as the name suggests, is always open to new learning (adaptive) without losing the old patterns (resonance). Basically, ART network is a vector classifier which accepts an input vector and classifies it into one of the categories depending upon which of the stored pattern it resembles the most.

Operating Principal

The main operation of ART classification can be divided into the following phases −

- Recognition phase − The input vector is compared with the classification presented at every node in the output layer. The output of the neuron becomes "1" if it best matches with the classification applied, otherwise it becomes "0".
- Comparison phase − In this phase, a comparison of the input vector to the comparison layer vector is done. The condition for reset is that the degree of similarity would be less than vigilance parameter.
- Search phase − In this phase, the network will search for reset as well as the match done in the above phases. Hence, if there would be no reset and the match is quite good, then the classification is over. Otherwise, the process would be repeated and the other stored pattern must be sent to find the correct match.

Classification:

ART is of two types-

1. ART1
2. ART2

ART1 is designed for clustering binary vectors and ART2 is designed to accept continuous-valued vectors.

Adaptive Resonance Theory1 (ART1)

It is a type of ART, which is designed to cluster binary vectors. We can understand about this with the architecture of it.

Architecture of ART1

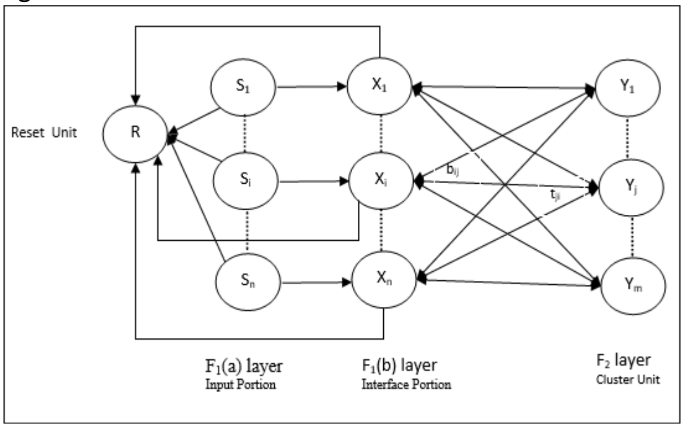
It consists of the following two units −

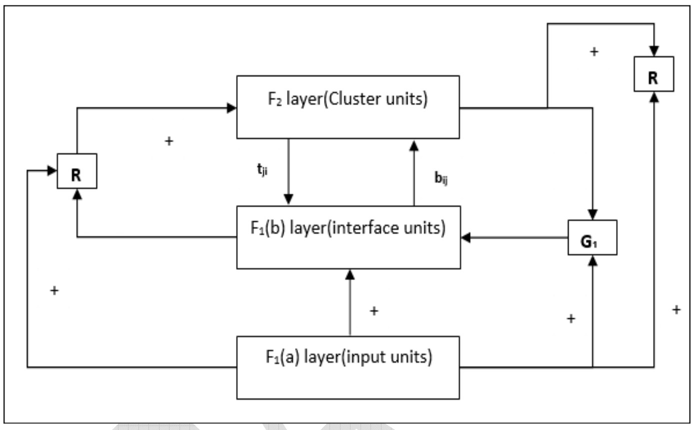
Computational Unit − It is made up of the following −

- Input unit (F_1 layer) − It further has the following two portions −
  - F_1(a) layer (Input portion) − In ART1, there would be no processing in this portion rather than having the input vectors only. It is connected to F_1(b) layer (interface portion).
  - F_1(b) layer (Interface portion) − This portion combines the signal from the input portion with that of F_2 layer. F_1(b) layer is connected to F_2 layer through bottom up weights b_{ij} and F_2 layer is connected to F_1(b) layer through top down weights t_{ji}.
- Cluster Unit (F_2 layer) − This is a competitive layer. The unit having the largest net input is selected to learn the input pattern. The activation of all other cluster unit are set to 0.

- Reset Mechanism − The work of this mechanism is based upon the similarity between the top-down weight and the input vector. Now, if the degree of this similarity is less than the vigilance parameter, then the cluster is not allowed to learn the pattern and a rest would happen.

Supplement Unit − actually the issue with Reset mechanism is that the layer F_2 must have to be inhibited under certain conditions and must also be available when some learning happens. That is why two supplemental units namely, G_1 and G_2 is added along with reset unit, R. They are called gain control units. These units receive and send signals to the other units present in the network. '+' indicates an excitatory signal, while '−' indicates an inhibitory signal.





Parameters Used

Following parameters are used −

- **n** − Number of components in the input vector
- **m** − Maximum number of clusters that can be formed
- **b_{ij}** − Weight from F_1(b) to F_2 layer, i.e. bottom-up weights
- **t_{ji}** − Weight from F_2 to F_1(b) layer, i.e. top-down weights
- **ρ** − Vigilance parameter
- **||x||** − Norm of vector x

Algorithm

Step 0: Initialize the parameters: α>1 and 0<ρ<=1

Initialize the weights: 0<bij(0)<α/α-1+n and tij(0)=1

Step 1: Perform steps 2 to 13 when stopping condition is false.

Step 2: Perform steps 3 to 12 for each of the training input.

Step 3: Set activation of all F2 units to zero. Set the activation of F1(a) units to input vectors.

Step 4: Calculate the norm of s: ||s||= ∑si

Step 5: Send input signal from F1(a) layer to F1(b) layer: xi=si

Step 6: For each F2 node that is not inhibited, the following rule should hold: If yj not=-1, then yj= ∑bij.xi

Step 7: Perform step 8 to 11 when reset is true.

Step 8: Find J for yJ>=yj for all nodes. If yJ =-1, then all the nodes are inhibited and this pattern cannot be clustered.

Step 9: Recalculate activation X of F1(b): xi = si.tJi

Step 10: Calculate the norm of vector x: ||x||=∑xi

Step 11: Test for reset condition.

If ||x||/||s||<ρ, then inhibit node J, yJ= -1.go to step 7 again.

Else if ||x||/||s||>=ρ, then proceed to the next step.

Step 12: Perform weight updation for node J: biJ(new)=αxi / α-1+||x|| and tJi(new)=xi

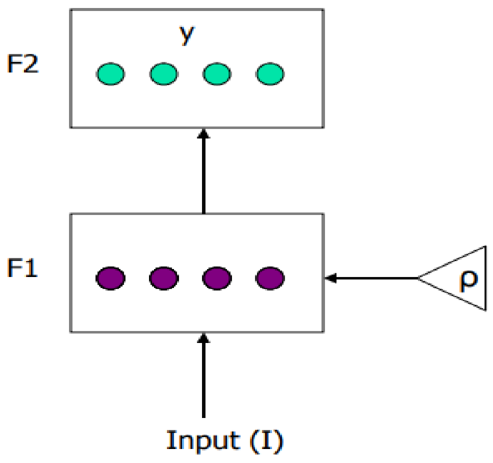
Step 13: Test for stopping condition. The following may be stopping conditions:

i).no change in weights

ii).no reset of units

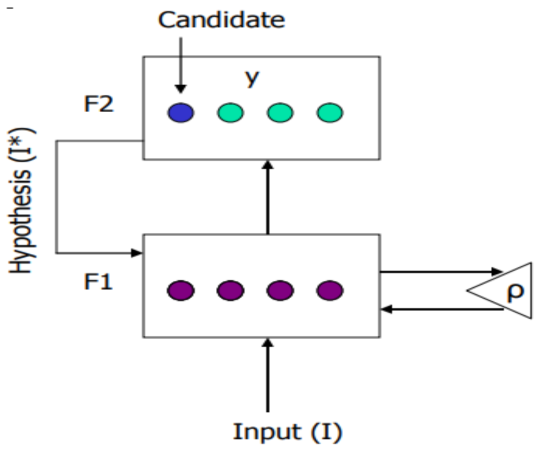
iii).maximum number of epochs reached.

Example:



Step 1: Send input from the F1 layer to F2 layer for processing. The first node within the F2 layer is chosen as the closest match to the input and a hypothesis is formed. This hypothesis represents what the node will look like after learning has occurred, assuming it is the correct node to be updated.

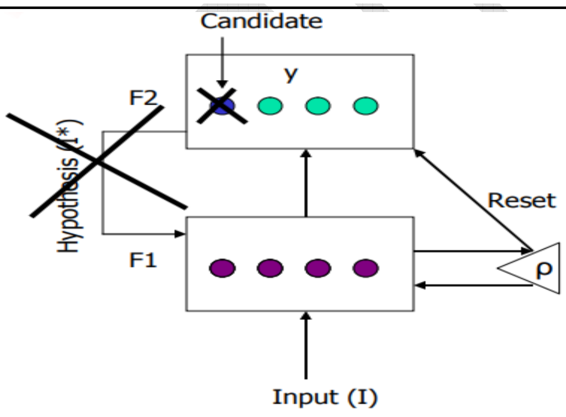
F1 (short term memory) contains a vector of size M, and there are N nodes within F2. Each node within F2 is a vector of size M. The set of nodes within F2 is referred to as "y".



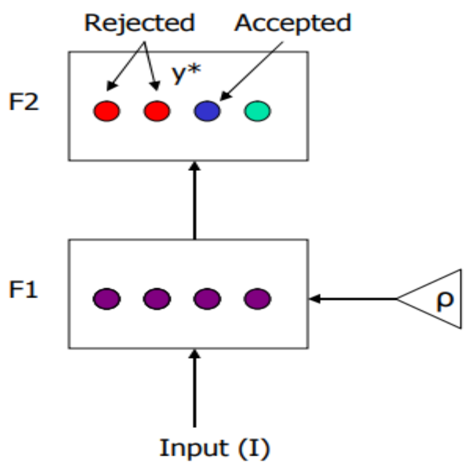
Step 2: Once the hypothesis has been formed, it is sent back to the F1 layer for matching. Let $T_j(I^*)$ represent the level of matching between I and I* for node j. Then:

$$T_j(I^*) = \frac{I \wedge I^*}{I} \quad \text{where} \quad A \wedge B = \min(A, B)$$

If $T_j(I^*) \geq \rho$ then the hypothesis is accepted and assigned to that node. Otherwise, the process moves on to Step 3.



Step 3: If the hypothesis is rejected, a "reset" command is sent back to the F2 layer. In this situation, the jth node within F2 is no longer a candidate so the process repeats for node j+1.



Step 4:

1. If the hypothesis was accepted, the winning node assigns its values to it.
2. If none of the nodes accepted the hypothesis, a new node is created within F2. As a result, the system forms a new memory.

In either case, the vigilance parameter ensures that the new information does not cause older knowledge to be forgotten.

Adaptive Resonance Theory has can be categorized into the following:

1. ART1 – Default ART architecture. Can handle discrete (binary) input.
2. ART2 – An extension of ART1. Can handle continuous input.
3. Fuzzy ART – Introduces fuzzy logic when forming the hypothesis.
4. ARTMAP – An ART network where one ART module attempts to learn based off of another ART module. In a sense, this is supervised learning.
5. FARTMAP – An ARTMAP architecture with Fuzzy logic included.

<u>Adaptive Resonance Theory 2:</u>

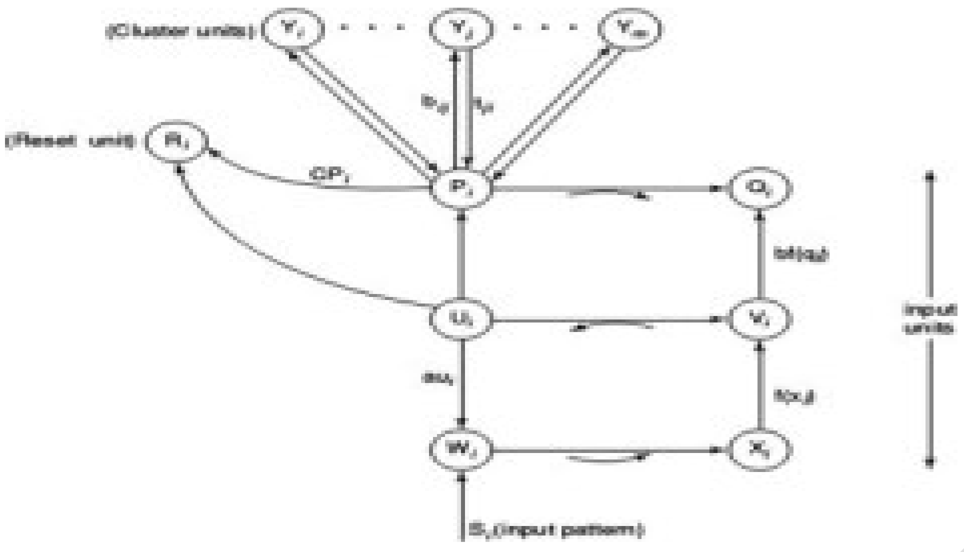
ART2 is for continuous-valued input vectors. In ART2 network complexity is higher than ART1 network because much processing s needed in F1 layer.ART2 network was designed to self-organize recognition categories for analog as well as binary input sequences. The continuous-valued inputs presented to the ART2 network may be of two forms-the first form is a "noisy binary" signal form and the second form of data is "truly continuous".

The major difference between ART1 and ART2 network is the input layer. A three-layer feedback system in the input layer of Art2 network is required :a bottom layer where the input patterns are read in, a top layer where inputs coming from the output layer are read in and a middle layer where the top and bottom patterns are combined together to form a matched pattern which is then fed back to the top and bottom input layers.

<u>ART2 Architecture:</u>

In ART2 architecture F1 layer consist of six types of units-W, X, U, V, P, Q- and there are n unit of each type.The supplemental unit "N" between units W and X receives signals from all "W" units, computes the norm of vector w and sends this signal to each of the X units. Similarly there exit supplemental units between U and V, and P and Q, performing same operation as done between W an X. The connection between Pi f the F1 layer and Yj of the F2 layer show the weighted interconnections, which multiplies the signal transmitted over those paths.

The operation performed in F2 layer are same for both ART1 and ART2.



Training Algorithm for ART2 network:

Step 0: Initialize the parameters :a, b, c, d, e, α, ρ, θ. Also specify the number of epochs of training(nep) and number of leaning iterations(nit).

Step 1: Perform step 2 to 12 (nep) times.

Step 2: Perform steps 3 to 11 for each input vector s.

Step 3: Update F1 unit activations:

ui=0 ; wi=si; Pi=0; qi=0; vi=f(xi);

xi=si / e+||s||

Update F1 unit activation again:

ui=vi / e+||v||; wi=si+a.ui;

Pi=ui; xi=wi / e+||w||;

qi=pi / e+ ||p||; vi= f(xi) + b.f(qi)

In ART2 networks, norms are calculated as the square root of the sum of the squares of the respective values.

Step 4: Calculate signals to F2 units: yj=∑bij.Pi

Step 5: Perform steps 6 and 7 when reset is true.

Step 6: Find F2 unit Yj with largest signal(J is defined such that yj>=yj, j=1 to m).

Step 7: Check for reset:

ui=vi / e + ||v||; Pi=ui + d.tJi; ri=ui + c.Pi / e+||u||+c||p||

If ||r|| < (ρ-e), then yJ =-1(inhibit J).Reset is true; perform step 5.

If ||r|| >= (ρ-e), then wi=si+a.ui; xi=wi / e+||w||; qi=pi / e+||p||; vi=f(xi)+b.f(qi)

Reset is false. Proceed to step 8.

Step 8: Perform steps 9 to 11 for specified number of learning iterations.

Step 9: Update the weights for winning unit J:

tJi=α.d.ui + {[1+α.d(d-1)]}tJi

biJ= α.d.ui + {[1+α.d(d-1)]}biJ

Step 10: Update F1 activations:

ui-vi / e+ ||v||; wi=si+a.ui;

Pi=ui+d.tJi; xi=wi / e+||w||;

qi=Pi / e+||p||; vi=f(xi)+b.f(qi)

Step 11: Check for the stopping condition of weight updation.

Step 12: Check for the stopping condition for number of epochs.

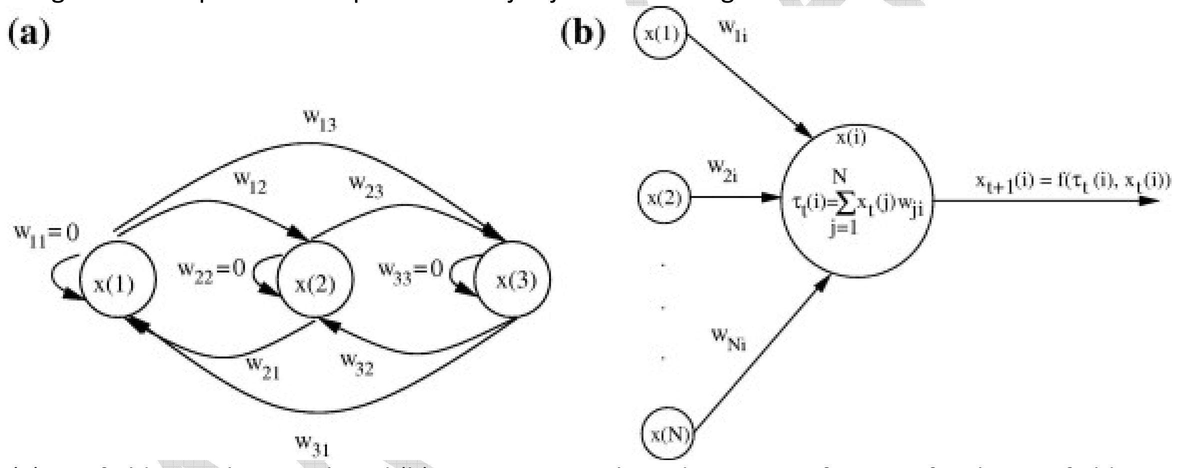
Hopfield Neural Networks

Hopfield neural networks represent a new neural computational paradigm by implementing an auto associative memory. They are recurrent or fully interconnected neural networks. 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 case all connections including self-connections are allowed.

A pattern, in N-node Hopfield neural network parlance, is an N-dimensional vector p=[p1,p2,…,pN] from the space $\mathbf{P} = \{-1,1\}^N$. A special subset of P represents the set of stored or reference patterns $\mathbf{E} = \{\mathbf{e^k} : 1 \leq k \leq K\}$, where $\mathbf{e^k} = [e_1^k, e_2^k, \ldots, e_N^k]$. 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.

As already stated in the Introduction, neural networks have four common components.
For the Hopfield net we have the following:
Neurons: The Hopfield network has a finite set of neurons x(i),1≤i≤N, which serve as processing units. Each neuron has a value (or state) at time t described by xt(i). A neuron in the Hopfield net has one of the two states, either -1 or +1; that is, $\mathbf{x}_t(i) \in \{-1, +1\}$.

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 wij called the synapse between x(i) and x(j). The design of the Hopfield net requires that wij=wji and wii=0. Figure 7.15a illustrates a three-node network.



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

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

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

b_i is the externally applied bias to the neuron.
Activation function: The activation function f determines the next state of the neuron xt+1(i) based on the value τt(i) computed by the propagation rule and the current value $\mathbf{x}_t(i)$. Figure b illustrates this fact. The activation function for the Hopfield net is the hard limiter defined here:

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

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≤k≤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 x(1),x(2),…,x(N).

The training algorithm of the Hopfield neural network is simple and is outlined below:

1. **Learning:** Assign weights wij 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.

2. **Initialization**: Draw an unknown pattern. The pattern to be learned is now presented to the net.
If p=[p1,p2,…,pN] 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.
In case of the continuous version of the Hopfield neural network, we have to consider neural self-connections wij≠0 and choose as an activation function a sigmoid function. With these new adjustments, the training algorithm operates in the same way.

The convergence property of Hopfield's network depends on the structure of W (the matrix with elements wij) and the updating mode. An important property of the Hopfield model is that if it operates in a sequential mode and W is symmetric with nonnegative diagonal elements, then the energy function

$$E_{hs}(t) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} x_i(t) x_j(t) - \sum_{i=1}^{n} b_i x_i(t)$$
$$= -\frac{1}{2} \mathbf{x}^T(t) \mathbf{W} \mathbf{x}(t) - \mathbf{b}^T \mathbf{x}(t)$$

is nonincreasing . The network always converges to a fixed point.

Hopfield neural networks are applied to solve many optimization problems. In medical image processing, they are applied in the continuous mode to image restoration, and in the binary mode to image segmentation and boundary detection.

The recall process can be organized in the following modes:
· Asynchronous updating: Each neuron may change its state at a random moment with respect to the others.
· Synchronous updating: All neurons change their states simultaneously at a given moment.
· Sequential updating: Only one neuron changes its state at any moment; thus all neurons change their states, but sequentially.

Why Artificial Neural Networks?

We need to understand the answer to the above question with an example of a human being. As a child, we used to learn the things with the help of our elders, which includes our parents or teachers. Then later by self-learning or practice we keep learning throughout our life. Scientists and researchers are also making the machine intelligent, just like a human being, and ANN plays a very important role in the same due to the following reasons −

- With the help of neural networks, we can find the solution of such problems for which algorithmic method is expensive or does not exist.
- Neural networks can learn by example, hence we do not need to program it at much extent.
- Neural networks have the accuracy and significantly fast speed than conventional speed.

Areas of Application

Followings are some of the areas, where ANN is being used. It suggests that ANN has an interdisciplinary approach in its development and applications.

Speech Recognition

Speech occupies a prominent role in human-human interaction. Therefore, it is natural for people to expect speech interfaces with computers. In the present era, for communication with machines, humans still need sophisticated languages which are difficult to learn and use. To ease this communication barrier, a simple solution could be, communication in a spoken language that is possible for the machine to understand.

Great progress has been made in this field, however, still such kinds of systems are facing the problem of limited vocabulary or grammar along with the issue of retraining of the system for different speakers in different conditions. ANN is playing a major role in this area. Following ANNs have been used for speech recognition −

- Multilayer networks
- Multilayer networks with recurrent connections
- Kohonen self-organizing feature map

The most useful network for this is Kohonen Self-Organizing feature map, which has its input as short segments of the speech waveform. It will map the same kind of phonemes as the output array, called feature extraction technique. After extracting the features, with the help of some acoustic models as back-end processing, it will recognize the utterance.

Character Recognition

It is an interesting problem which falls under the general area of Pattern Recognition. Many neural networks have been developed for automatic recognition of handwritten characters, either letters or digits. Following are some ANNs which have been used for character recognition −

- Multilayer neural networks such as Backpropagation neural networks.
- Neocognitron

Though back-propagation neural networks have several hidden layers, the pattern of connection from one layer to the next is localized. Similarly, neocognitron also has several hidden layers and its training is done layer by layer for such kind of applications.

Signature Verification Application

Signatures are one of the most useful ways to authorize and authenticate a person in legal transactions. Signature verification technique is a non-vision based technique.

For this application, the first approach is to extract the feature or rather the geometrical feature set representing the signature. With these feature sets, we have to train the neural networks using an efficient neural network algorithm. This trained neural network will classify the signature as being genuine or forged under the verification stage.

Human Face Recognition

It is one of the biometric methods to identify the given face. It is a typical task because of the characterization of "non-face" images. However, if a neural network is well trained, then it can be divided into two classes namely images having faces and images that do not have faces.

First, all the input images must be preprocessed. Then, the dimensionality of that image must be reduced. And, at last it must be classified using neural network training algorithm. Following neural networks are used for training purposes with preprocessed image −

- Fully-connected multilayer feed-forward neural network trained with the help of back-propagation algorithm.
- For dimensionality reduction, Principal Component Analysis (PCA) is used.