Neural Networks
2. NEURAL NETWORKS

Introduction

Initially the computing was meant only for numerical calculations but later on it was realized that symbols like text also can be manipulated by algorithmic approach. This development motivated the implementation of logical inferences on computer. Through logic it was possible to incorporate heuristics in reducing the search for solution. These techniques gained popularity as artificial intelligence and were implemented on digital computers. At that time it was felt that all AI problems including speech and image understanding could be solved by using such heuristic search. But these methods require very particular representation of the problems as in some puzzles and games. To overcome these representational problems rule based approaches came in practice. In these approaches, rules for solution of any problem are obtained by human expert. The rule-base together with inference mechanism resulted in development of expert system. Drawback of this approach is that all the knowledge used by human being in solving a problem can not be represented by set of explicit rules.

Currently the term like AI systems, intelligent systems, knowledge base systems, expert systems etc. are used to convey the meaning that it is possible to build machines that can demonstrate intelligence similar to human being in performing some simple tasks. A machine is said to be intelligent if the performance of the machine and human being are same. To solve a problem with machine which a human being can perform requires inclusion of common sense knowledge and learning. These features are very difficult to include in a machine. Recently significant development has taken place in device technologies and computer architectures. It is now realized that single or multiple processors in Von Neumann model have limitation of speed.
Therefore the current trend is to explore architectures based on parallel distributed processing models motivated by our understanding of structure and function of the biological neural network (details are given in Appendix-A). Artificial neural network (ANN) is one such development in which the driving force is to make machines more and more intelligent.

2.1 Why ANNs Are Needed

The main difference between human and machine intelligence is that human perceive everything as a pattern, whereas for a machine everything is data. Even if the highly complex algorithm can be implemented and computed with the current technology, the pattern description and knowledge can not be completely derived for a given problem. At present time computers and other machines can be built to perform variety of well defined tasks with high speed and reliability. Most of the information processing at the moment is based on John Von Neumann’s adding machine concepts. But before such a computer can be programmed to carry out an information processing function, some person has to understand that function and devise an algorithm for implementing it. Digital computers can solve problems like matrix inversion and solution of differential equation for which we have definite algorithm but it can not solve complex problems like speech recognition for which it is very difficult to have definite algorithm. Even worse, there may be tasks for which algorithm do not yet exist or for which it is virtually impossible to write down a series of logical steps that will arrive at the answer. No human can rival the speed of modern workstation in solving complex problem, still there exist many problems to be solved to our satisfaction by any man made machine but are easily solved by the perceptual or cognitive power of human, and often lower mammals, or even fish and insects. No computer vision system can rival the human ability to recognize visual images formed by objects of all shapes and orientations under a wide range of conditions. The problems easily solved by the brain than by the digital computer typically have two characteristics.
they are ill defined, and they usually require enormous amount of processing. The pattern nature in storage and recall automatically gives robustness and fault tolerance for the human system. Functionally also brain and digital computer differ in sense that brain understands pattern, whereas computers can be made to recognize pattern in data. Human brains get the whole object in data even though there is no clear identification of sub-patterns in the data. For example, consider the name of a person written in a handwritten cursive script. Even though the individual patterns for each letter may not be evident, the name is understood due to visual hints provided in the written script. Likewise speech is understood even though the patterns corresponding to the individual sounds may be distorted. Another example is recognition of character of an object from its image on television; this involves resolving ambiguities associated with distortion and lighting. It also involves filling in information about three dimensional scene, which is missing from the two dimensional image on the screen. Our brain accomplishes all these by utilizing massive parallelism, with millions and even billions of neurons in parts of the brain working together to solve complex problems.

Hence it will be advantageous to attack certain problems by designing naturally parallel computers, which process information and learn by principles borrowed from the nervous system of biological creatures (see appendix-A). ANNs are such attempts to mimic the functioning of human brain.

2.2 Working Principle of Artificial Neural Network

A neural network is modeled on the gross structure of human brain; a collection of nerve cells or neuron, each of which is connected to as many as 10000 others, from which it receives stimuli (inputs and feedback) to which it sends stimuli. Some of these connections are strong and others are weak. The brain accepts inputs and generates response to them in accordance with its
genetically programmed structure, but mainly through learning, organizing itself in reaction to input rather than doing only by rote what it is told.

The neural networks used by engineers are loosely based upon biology. Only comparison is that they behave vaguely in a similar way. Since the working of biological neural network is not yet fully understood, it will be a long time before we can recreate in a machine all the capabilities of brain. Even so the neural networks used by engineers are offering some valuable, specialized, brain like capabilities that in likelihood is beyond the reach of algorithmic programming. To differentiate them from the biological neural network the term 'artificial neural network' (ANN) is frequently used.

An ANN consists of a collection of processing elements having many input signals but only one output signal. This output signal is fed as input to other elements. Each processing element has its own local small memory which stores the value of some previous computations along with the adaptive coefficients basic to neural network learning. The processing that each element does is determined by a transfer function. Often an ANN is divided into layers (groups of processing elements) all having the same transfer function. Depending on the design of ANN, the processing elements either operate continuously or updated serially. A scheduling function determines in which way and how often each processing element is to apply its transfer function.

Each processing element is completely self sufficient and work away in total disregard to the processing going on inside its neighbours. In any ANN a great deal of independent computation is usually underway. At the same time all neurons intimately affect the behaviour of entire network. Every connection entering a neuron has an adaptive coefficient called a weight assigned to it. This weight which is stored in the local memory of neuron is generally used to amplify, attenuate and also to change the sign of the signal in the incoming connection. These weights are not fixed but may change. Most
transfer functions include a learning law (an equation that modifies all or some of weights) in the local memory in response to input and desired output.

2.3 Statistical Nature of ANN

A critical fact about the ANN is that they are statistical associative models. A typical network model has a set of input and a set of output patterns. The role of the network is to perform a function that associates each input pattern with an output pattern. A learning algorithm uses the statistical properties of set of input / output pairs, called the training set, to generalize. That is, generate output from new inputs. Without the ability to generalize, neural network model would be like lookup tables, which are not very interesting [31].

It is important to understand the difference between statistical and rule-based inference. Statistical inference allows for exceptions and randomness in the association between two variables, whereas rules are deterministic. In neural network model, the history of the system determines the system response to a new stimulus, often, rule based systems are non-adaptive, that is they do not respond to observed changes in the stimulus environment, although they can be made to be adaptive at the expense of making the rules more complex.

2.4 Models of Neurons

In this section discussion on three classical models for an artificial neuron is presented. These neurons are also referred as processing element hence these terms will be used interchangeably in the succeeding discussions.

2.4.1 McCulloch-Pitts Model

In McCulloch-Pitts (MP) model (see Fig. 2.1) the activation ‘x’ is given by weighted sum of its M input values ‘aᵢ’ and a bias term ‘b’. The output signal
's' is typically a nonlinear function $f(x)$ of the activation value 'x'. The following equations describe the operation of a MP model.

\[ x = \sum_{i=1}^{M} w_i a_i + b \]  \hspace{1cm} (2.1)

Activation:

\[ s = f(x) \]  \hspace{1cm} (2.2)

Output signal:

The output function originally used with MP model was binary type with threshold values 'zero' and 'one'. In this model of neuron the weights are fixed. Hence the network using this neuron does not have the capability of learning.

2.4.2 Perceptron

The Rosenblatt’s perceptron model (see Fig. 2.2) for an artificial neuron consists of outputs from sensory units to a fixed set of association units, the output of which are fed to an MP neuron. The association units perform predetermined manipulation on their inputs. The main deviation from MP neuron is that adjustment of weight is incorporated in the operation of the unit. The desired output 'd' is compared with actual binary output 's' and the error
‘e’ is used to adjust the weights. The following equations describe the operation of the perceptron model of a neuron.

\[
\begin{align*}
\text{Activation:} & \quad x = \sum_{i=1}^{M} w_i a_i + b \\
\text{Output signal:} & \quad s = f(x) \\
\text{Error:} & \quad e = d - s \\
\text{Weight change:} & \quad \Delta w_i = \eta e a_i
\end{align*}
\]  

Where \( \eta \) is the learning rate parameter.

There is a perceptron learning law, which gives step-by-step procedure for adjusting the weights. Convergence of weight adjustment depends on nature of input-output pairs. If the weight values converge, then the corresponding problem is said to be represented by the perceptron network.

Fig. 2.2, Rosenblatt’s Perceptron Model of Neuron
2.4.3 Adaline

ADAptive LINear Element (Adaline) is a computing model proposed by Widrow as shown in Fig. 2.3. The main difference between Rosenblatt’s perceptron model and Widrow’s Adaline is that, in this model the analog activation value ‘x’ is compared with the desired output ‘d’. Output here is linear function of activation value ‘x’. The equations describing the operation of an Adaline are as follows:

\[
x = \sum_{i=1}^{M} w_i a_i + b \quad (2.7)
\]

Output signal: \[s = f(x) = x \quad (2.8)\]

Error: \[e = d - s = d - x \quad (2.9)\]

Weight change: \[\Delta w_i = \eta e a_i \quad (2.10)\]

Where \(\eta\) is the learning rate parameter. The weight rule minimizes the squared error ‘\(e^2\)’ averaged over all inputs. This rule is also called least mean squared (LMS) error learning law. This law is derived using negative gradient of the error surface in the weight space therefore this is also referred to as gradient descent algorithm.

Fig. 2.3, Widrow’s Adaline Model of Neuron
2.5 Connecting Method of Processing Units

ANNs can perform useful tasks when the processing units are organized in a suitable manner. The arrangement of processing units, connections and pattern input / output is referred to as topology.

ANNs are normally organized into layers of processing units. The units of a layer have same activation dynamics and output functions. Connections may be interlayer, intralayer or both interlayer and intralayer. Also the connections across the layers and among the units within a layer can be in feedforward manner or in a feedback manner.

There are six basic structures which form the building blocks for complex neural network architectures. Let us consider only two layers $F_1$ and $F_2$ with $M$ and $N$ processing units respectively to demonstrate the six basic structures. By connecting $j^{th}$ unit in $F_2$ with all the units in $F_1$ we get two network structures ‘instar’ (Fig. 2.4) and ‘outstar’ (Fig. 2.5). In instar connection signals propagate from $F_1$ to $F_2$ layer and during learning the normalized weight vector $w_j = (w_{j1}, w_{j2}, \ldots, w_{jM})^T$ approaches the normalized input vector, when an input vector $a = (a_1, a_2, \ldots, a_M)^T$ is presented at layer $F_1$. Hence the activation $w_j^T a = \sum_{i=1}^{M} w_{ji} a_i$ of $j^{th}$ unit in layer $F_2$ will approach maximum value during learning. After learning, whenever the input is given to $F_2$, then $j^{th}$ unit will be activated maximally, this operation of instar can be viewed as content addressing the memory. In the case of an outstar during learning, the weight vector for the connection from $j^{th}$ unit in $F_2$ approaches the activating pattern in $F_1$. When an input vector $a$ is presented at $F_1$. During recall, whenever the unit $j$ is activated, the signal pattern $(s_j w_{1j}, s_j w_{2j}, \ldots, s_j w_{Mj})$ will be transmitted to $F_1$, where $s_j$ is output of $j^{th}$ unit. This signal pattern then produces the original activity pattern $a$. This operation of an outstar can be viewed as memory addressing the content.

When the connections from the units in $F_1$ to $F_2$ are made as in Fig. 2.6 then this network is called a hetroassociation network. This network can be
Fig. 2.4, Instar

Fig. 2.5, Outstar

Fig. 2.6, Group of Instar

Fig. 2.7, Group of Outstar

Fig. 2.8, Bidirectional Associative Memory

Fig. 2.9, Autoassociative Memory
viewed as group of instars, if the flow is from $F_1$ to $F_2$. On the other hand if the flow is from $F_2$ to $F_1$, then the network can be viewed as group of outstars (see Fig. 2.7).

When the flow is bidirectional the structure is called bidirectional associative memory (see Fig. 2.8). In this structure either of the layers can be viewed as input or output.

If the two layers $F_1$ and $F_2$ coincide and the weights are symmetric i.e. $w_{ij} = w_{ji}$, $i \neq j$ then this structure is called autoassociative memory in which each unit is connected to every other unit and to itself (see Fig. 2.9).

2.6 Basic Functional Units

The simplest functional blocks of artificial neural network are of three types. These are (i) Feedforward (ii) Feedback (iii) A combination of both. These are called functional block because these can perform some simple pattern recognition task by themselves.

Fig. 2.10. Basic Feedforward Neural Network
The simplest feedforward network has two layers with $M$ input units and $N$ output units (see Fig. 2.10). Each input unit is connected to each of the output unit, and each connection is associated with weight of connection. The input units are linear and do the task of fanout. The output units may be linear or nonlinear depending on the problem. These types of networks are used for pattern association, pattern classification and pattern mapping.

The simplest feedback network shown in Fig. 2.11, has $N$ processing units each connected to all other units. The connection weights are assumed to be symmetric $w_{ij} = w_{ji}$, for $i \neq j$. Depending on the application, the output function may be linear or nonlinear. These types of networks are typically used for autoassociation or pattern storage problem.

![Fig. 2.11, Basic Feedback Neural Network](image)

The simplest combination network is shown in Fig. 2.12. This is also called competitive learning network. It has input layer of units feeding to output layer in feedforward manner. The units in the output layer feed the other units in output and to itself. Usually the feedforward connections are adjustable and feedback connections are fixed to some specific value depending on the problem. The input units are linear and output units may be
linear or nonlinear depending on the task. These types of network are used for pattern clustering or grouping tasks.

![Basic Competitive Learning Network](image)

**Fig. 2.12, Basic Competitive Learning Network**

### 2.7 Commonly Used Output Function

There are various types of output functions. Choice of the output function depends on problem under consideration. Some output functions are discussed here.

#### 2.7.1 Binary Function

This function is defined as

\[
f(x) = 1, \text{ for } x \geq 0 \tag{2.11}
\]

\[
f(x) = 0, \text{ for } x < 0 \tag{2.12}
\]

It has saturation values 0 and 1 (see Fig. 2.13)
2.7.2 Signum Function

This function indicates the sign of activating variable. It is defined as

\[ f(x) = -1.0, \text{ for } x < 0 \] (2.13)

\[ f(x) = 1.0, \text{ for } x \geq 0 \] (2.14)

This has saturation values -1.0 and 1.0 (see Fig. 2.14)

2.7.3 Linear Function

In cases where above discussed threshold nonlinearities can not be used linear function is helpful. It was first used in ADALINE model of ANN. This is defined as below

\[ f(x) = x, \text{ for } -\infty < x < \infty \] (2.15)
This function does not saturate hence sometimes causes difficulty in convergence of weight updates.

![Diagram of f(x)](image)

Fig. 2.15, Linear Output Function

### 2.7.4 Logistic Function

For gradient search methods of weight updates, differentiable functions are needed. For this purpose function with sigmoidal nonlinearity are used. Logistic function is one such function which is defined as,

\[ f(x) = \frac{1}{1 + \exp(-x / T)}, \quad \text{for } -\infty < x < \infty \]  \hspace{1cm} (2.16)

where T is referred to as temperature. It shows the degree of nonlinearity of the function.

For T = 1,

If \( x = 0 \), \( f(x) = 0.5 \) \hspace{1cm} (2.17)

If \( x \to \infty \), \( f(x) \to 1.0 \) \hspace{1cm} (2.18)

If \( x \to -\infty \), \( f(x) \to 0.0 \) \hspace{1cm} (2.19)
This function has saturation value of ‘0’ and ‘1’ (see Fig. 2.16). At $T = 0$ logistic function reduces to binary function which is strictly nonlinear. As $T \to \infty$ its nonlinearity decreases and finally $f(x) \to 0.5$.

Derivative of logistic function for $T = 1$ is,

$$\frac{df(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = f(x) [1-f(x)] \quad (2.20)$$

It can be seen from this equation that $df(x)/dx$ has maximum value of 0.25 when $f(x) = 0.5$ (see Fig. 2.16) and has minimum value of 0 when $f(x) = 0$ or 1. The weight change depends on derivative of output function hence this change is maximum in the midrange of the activation value. This feature contributes to stability of learning law.

![Logistic Output Function and its Derivative](image)

Fig. 2.16, Logistic Output Function and its Derivative

### 2.7.5 Hyperbolic Tangent Function

This is also a sigmoidal nonlinear function which is differentiable. It is defined as below,

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \text{ or}$$
\[ f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}, \quad \text{for } -\infty < x < \infty \]  
\hspace{1cm} (2.21)

If \( x = 0 \), \( f(x) = 0 \)  
\hspace{1cm} (2.22)

If \( x \to \infty \), \( f(x) \to 1.0 \)  
\hspace{1cm} (2.23)

If \( x \to - \infty \), \( f(x) \to -1.0 \)  
\hspace{1cm} (2.24)

This function has saturation values -1.0 and 1.0 (see Fig. 2.17). Its derivative is,
\[
\frac{df(x)}{dx} = 1 - f^2(x) 
\hspace{1cm} (2.25)
\]

Derivative equation shows that for \( f(x) = +1 \) or -1; \( df(x)/dx = 0 \) and for \( f(x) = 0 \) this attains the maximum value of unity (see Fig. 2.17). The asymmetry of hyperbolic tangent function makes the learning faster during training.

![Fig. 2.17, Hyperbolic Tangent Output Function and its Derivative](image-url)
2.8 Learning Methods

Learning means adjustment of weights of ANN. Learning is a slow process and patterns have to be presented to ANN repeatedly before the pattern information is captured in the weight. Interesting thing about learning is that pattern features are distributed across all the weights slowly and training samples themselves are never stored. Learning should be able to capture complex nonlinear mapping between input-output pattern pairs, as well as between adjacent patterns in some cases. Also it should capture as many patterns as possible into the network.

Learning can be viewed as search in weight space to determine the weight vector for optimal objective function. This search depends on criterion used for learning. Basically there are two criteria ‘supervised’ and ‘unsupervised’. In supervised methods the weight adjustment is based on deviation of actual output form desired output. This learning can be used for structural or for temporal learning. Structural learning means capturing in weights the relationship between input-output patterns on the other hand temporal learning is concerned with capturing the relationship with neighbouring patterns in a sequence of patterns. In unsupervised learning the objective is to categorize or discover features in training data. Unlike supervised learning there in no teacher signal. Unsupervised learning mostly uses the local information in the pattern itself or in the outputs to update the weights.

2.8.1 Error Correction Rules

These rules were initially proposed for training of single processing unit. These rules drive the output error of a single unit to zero and fall in the category of supervised learning. Perceptron rule, May’s rule and $\alpha$ – LMS are some error correction rules. Perceptron and $\alpha$ – LMS rules are briefly discussed here.
Perceptron Rule

Let \((a, d)\) be a sample of input-output pair of vector for which a network is to be trained. If \(w\) be the weight vector and \(s_i\) be the output from \(i^{th}\) unit then change in weight vector is given by.

\[
\Delta w_i = \eta \left[ d_i - s_i \right] a, \tag{2.26}
\]

\[
s_i = \text{sgn}(w_i^T a), \tag{2.27}
\]

where \(\text{sgn}(x)\) is sign of \(x\). Therefore, we have

\[
\Delta w_{ij} = \eta \left[ d_i - s_i \right] a_j, \text{ for } j = 1, 2, \ldots, M, \tag{2.28}
\]

This law is applicable for bipolar output function \(f(.)\). This law is also referred as discrete perceptron law. The \(\Delta w_{ij}\) has non zero value only when actual output \(s_i\) is incorrect. Weights can be initialized to any random value and by repeated use of input-output pair gradually they converge to some final value if the patterns are representable by the system.

\(\alpha -\) LMS Learning Rule

This rule was proposed by widrow and Hoff. This was originally used to train ADALINE. This is an example of error correcting rule with a quadratic criterion function. This is based on minimal disturbance principle. Later on it was discovered that this rule causes convergence corresponding to least mean square (LMS) of output error if all input patterns are of same length.

Assuming a linear output function for each unit i.e. \(f(x) = x\). The corresponding weight update equation is given by,

\[
\Delta w_i = \eta \left[ d_i - w_i^T a \right] a, \tag{2.29}
\]
Hence we have, $\Delta w_{ij} = \eta [d_i - w_i^T a] a_j$, for $j = 1,2,\ldots,M$ \hfill (2.30)

Originally proposed equation is as below,

$\Delta w_i = \alpha [d_i - w_i^T a] a / |a|^2$ \hfill (2.31)

From the above equation relation between $\alpha$ and $\eta$ will be,

$\eta = \alpha / |a|^2$ \hfill (2.32)

Although this law was founded on minimal disturbance principle but it is also a gradient descent minimizer of an appropriate quadratic criterion function.

2.8.2 Gradient Descent-Based Learning Rules

These rules are applicable for differentiable output functions. In this an appropriate criterion function is optimized by iterative gradient search procedure. These are also supervised learning rules. Delta rule, $\mu$-LMS rule, backpropagation etc. are some examples of this type of rule. Delta rule is discussed briefly in this section and backpropagation rule is discussed in Chapter-5.

Delta Rule

This rule is the extension of $\mu$-LMS rule to units with differentiable output function. In delta rule the change in the weight vector is given by,

$\Delta w_i = \eta [d_i - f (w_i^T a)] df (w_i^T a) a \overline{dx}$ \hfill (2.33)

Where $df(.)/dx$ is the derivative with respect to $x$. Hence,
\[ \Delta w_{ij} = \eta \left[ d_i - f(w_i^T a_i) \right] \frac{df(w_i^T a_i)}{dx} a_j \]  \hspace{1cm} (2.34)

\[ \Delta w_{ij} = \eta \left[ d_i - s_i \right] \frac{df(x_i)}{dx} a_j; \quad \text{for} \ j = 1, 2, \ldots, M \]  \hspace{1cm} (2.35)

As the change in weight is based on the error between the desired and the actual output values for a given input, this law can be viewed as a continuous perceptron rule. The weights converge to final values by repeated application of input output pairs. Convergence is almost guaranteed if the number of layers is more. This rule can be generalized for training multiple layers of feedforward network. This extended version is known as generalized delta rule which is popularly known as backpropagation rule.

2.8.3 Reinforcement Learning

In cases where desired output is known for inputs, we can find the error between them and error correction based on supervised learning can be applied. On the other hand there are many situations where desired output is not known but we definitely have binary result that the output is right or wrong. This output can be used as reinforcement signal. Learning based on this reinforcement signal is called learning with critic or reinforcement learning. Depending on reinforcement signal credit or blame for the overall outcome is assigned to different units of the network. This is called ‘structural credit assignment’. If the credit is assigned on the basis of series of action then it is called ‘temporal credit assignment’. The reinforcement signal is basically feedback from the environment. If the feedback signal for the given input-output pair is fixed with time, then it is called fixed credit assignment. On the other hand if the input output pair determines only the probability of positive reinforcement, then it is called probabilistic credit assignment.
2.8.4 Hebb’s Rule

This is unsupervised learning rule. Weight change in this learning is based on correlation of input and output signals of a unit (see Fig. 2.18).

![Fig. 2.18](image)

Change in weight vector is given by,

$$\Delta w_i = \eta f(w_i^T a) a$$  \hspace{1cm} (2.36)

Therefore the jth component of $\Delta w_i$ will be given by,

$$\Delta w_{ij} = \eta f(w_i^T a_j) a_j = \eta s_i a_j, \quad \text{for } j = 1, 2, \ldots, M$$  \hspace{1cm} (2.37)

Where $s_i$ is the output of ith unit. This rule requires weight initialization to small random values near zero i.e. $w_{ij} \approx 0$ prior to learning.

Synaptic dynamic equation for this learning is given as.

$$\frac{dw_{ij}}{dt} = -w_{ij}(t) + s_i s_j$$  \hspace{1cm} (2.38)

Solution of this equation is,

$$w_{ij}(t) = w_{ij}(0) e^{t} + \int_{0}^{t} s_i(\tau) s_j(\tau) e^{t-\tau} d\tau$$  \hspace{1cm} (2.39)
Where \( w_{ij}(0) \) is the initial value of weight at time \( t = 0 \). The first term shows the past knowledge will decay exponentially to zero, which is equivalent to forgetting. Second term accumulates the correlation term with more weightage to recent terms.

2.8.5 Competitive Learning

This method modulates the difference between output signal and synaptic weight. This is based on the notion of competition among the units and specialization of units to tackle a different class of problems involving clustering of unlabeled data. The synaptic learning equation for this learning is given as,

\[
dw_{ij}(t)/dt = s_i [s_j - w_{ij}(t)] = -s_i w_{ij}(t) + s_i s_j
\]  

(2.40)

Where \( s_i = f_i(x_i(t)) \), is the output signal of unit ‘i’ and \( s_j = f_j(x_j(t)) \), is the output signal of unit ‘j’. This dynamic equation is similar to Hebbian learning equation except the forgetting term \((-s_i w_{ij}(t))\), which is nonlinear in this case. If \( s_i = 0 \), there is no change in the weight. Also with \( s_i = 0 \), forgetting term is also zero hence the network will not forget the knowledge already acquired.

When an external input is presented at input layer of units and these units feed signals to output units. The signals from the output units compete with each other, leaving one unit as the winner. The output of winning unit is nonzero hence weights corresponding to this unit are adjusted according to above explained law to match the input vector [32-34].

All the rules discussed in this section, show the possible varieties in the existing methods of learning. There are many other rules which are not possible to discuss here. Fig. 2.19 shows most of the rules with network type and output functions to be used.
Fig. 2.19, Various Learning Rules

**********