3

Artificial Neural Network and Supervised Learning

3.1 Introduction

Artificial neural networks are biologically inspired but *not necessarily biologically plausible.* Researchers are usually thinking about the organization of the brain when considering network configurations and algorithms. But the knowledge about the brain's overall operation is so limited that there is little to guide those who would emulate it. Hence, at present time biologists, psychologists, computer scientists, physicists and mathematicians are working all over the world to learn more and more about the brain. Interests in neural network differ according to profession like neurobiologists and psychologists try to understanding brain. Engineers and physicists use it as tool to recognize patterns in noisy data, business analysts and engineers use to model data, computer scientists and mathematicians viewed as a computing machines that may be *taught* rather than programmed and artificial intelligentsia, cognitive scientists and philosophers use as sub-symbolic processing (reasoning with patterns, not symbols), etc.

A conventional computer will never operate as brain does, but it can be used to simulate or model human thought. In 1955, Herbert Simon and Allen Newell announced that they had invented a thinking machine. Their program, the logic theorist, dealt with problems of proving theories based on assumptions it was given. Simon and Newell later developed the general problem solver, which served as the basis of artificial intelligence (AI) systems. Simon and Newell believed that the main task of AI was figuring out the nature of the symbols and rules that the mind uses. For many years AI engineers have used the "top-down" approach to create intelligent machinery. The top-down approach starts with the highest level of complexity, in this case thought, and breaks it down into smaller pieces to work with. A procedure is followed step by step. AI engineers write very complex computer programs to solve problems. Another approach to the modelling of brain functioning starts with the lowest level, the single neuron. This could be referred to as a bottom-up approach to modelling intelligence.

3.2 Comparison of Neural Techniques and Artificial Intelligence

Artificial intelligence (AI) is a branch of computer science that has evolved to study the techniques of construction of computer programs capable of displaying intelligent behavior. It is the study of computations that make it possible to perceive, reason, and act. Growth of artificial intelligence based on the hypothesis that thought processes could be modeled using a set of symbols and applying a set of logical transformation rules. It is important to note that any artificially intelligent system must possess three essential components:

1. A representation mechanism to handle knowledge which could be general or domain specific, implicit or explicit and of different level of abstraction.
2. An inference mechanism to get the appropriate conclusion for the given information or fact.
3. A mechanism for learning from new information or data without disturbing much to the existing set of rule.

The languages commonly used for AI model development are list processing language (LISP) and programming in logic (PROLOG).

The symbolic approach has a number of limitations:

- It is essentially sequential and difficult to parallelize.
- When the quantity of data increases, the methods may suffer a combinatorial explosion.
- An item of knowledge is represented by a rule. This localized representation of knowledge does not lend itself to a robust system.
- The learning process seems difficult to simulate in a symbolic system.

The ANN approach offers the following advantages over the symbolic approach:

- Parallel and real-time operation of many different components
- The distributed representation of knowledge
- Learning by modifying connection weights.

Both approaches are combined to utilize the advantages of both the techniques. A brief comparison of these techniques is given in Table 3.1.

3.3 Artificial Neuron Structure

The human nervous system, built of cells called neurons is of staggering complexity. An estimated 10^{11} interconnections over transmission paths are there that may range for a meter or more. Each neuron shares many characteristics with the other cells in the body, but has unique capabilities to receive, process, and transmit electrochemical signals over neural pathways that comprise the

Table 3.1. Comparison between ANN and AI

	ANN	AI
Type of information	Quantitative	Qualitative
Input	Measurements	Facts
Output	Predictions	Decision
Type of model	Mathematical	Logical
Requirement for model development	Historical data	Human experts
Adaptability	Learning capability	No learning capability
Flexibility	Re-trained for other problems	Completely change the knowledge base if problem changes
Model accuracy	Depends on learning	Depends on the knowledge acquired
Explanation	No explanation	Explanation depends on the depth of knowledge
Processing	Parallel and distributed	Sequential and logical
Representational structure of knowledge	Store global patterns or function information	Declarative (a collection of facts) or procedural (specifying an algorithm code to process information)

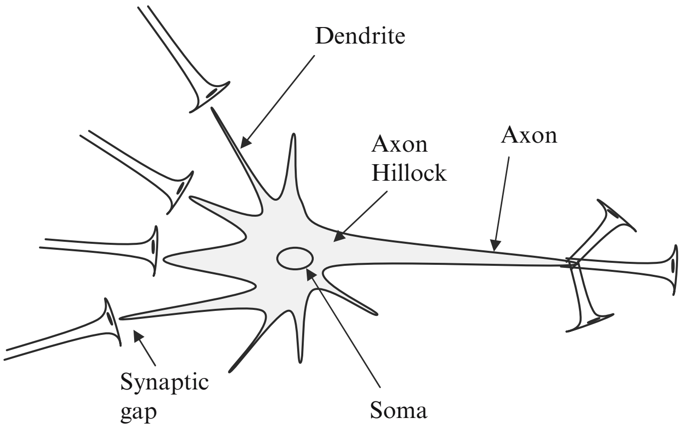


Fig. 3.1. Structure of biological neuron

brain's communication system. Figure 3.1 shows the structure of typical biological neurons. Biological neuron basically consists of three main components cell body, dendrite and axon. Dendrites extend from the cell body to other neurons where they receive signals at a connection point called a synapse. On the receiving side of the synapse, these inputs are conducted to the cell body, where they are summed up. Some inputs tend to excite the cell causing a reduction in the potential across the cell membrane; others tend to inhibit its

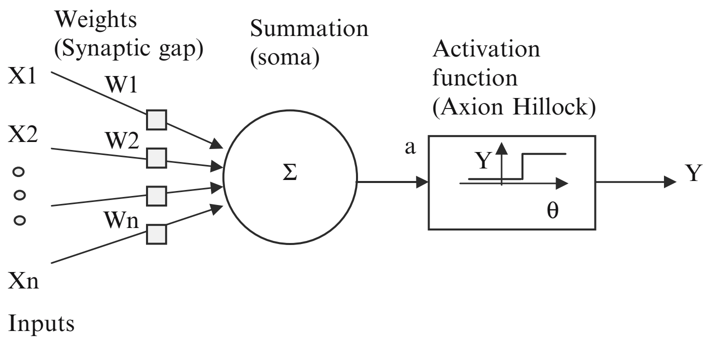


Fig. 3.2. Artificial neuron structure (perceptron model)

firing causing an increase in the polarization of the receiving nerve cell. When the cumulative excitation in the cell body exceeds a threshold, the cell fires and *action potential* is generated and propagates down the axon towards the synaptic junctions with other nerve cells.

The artificial neuron was designed to mimic the first order characteristics of the biological neuron. McCulloch and Pitts suggested the first synthetic neuron in the early 1940s. In essence, a set of inputs are applied, each representing the output of another neuron. Each input is multiplied by a corresponding weight, analogous to a synaptic strength, and all of the weighted inputs are then summed to determine the activation level of the neuron. If this activation exceeds a certain threshold the unit produces an output response. This functionality is captured in the artificial neuron known as the threshold logic unit (TLU) originally proposed by McCulloch and Pitts. Figure 3.2 shows a model that implement this idea. Despite of the diversity of network paradigms, nearly all are based upon this neuron configuration. Here a set of input labeled X_1, $\mathrm{X}_2, \ldots, \mathrm{X}_\mathrm{n}$ is applied from the input space to artificial neuron. These inputs, collectively referred as the input vector "X" corresponds to the signal into the synapses of biological neuron. Each signal is multiplied by an associated weight W_1, $\mathrm{W}_2, \ldots \mathrm{W}_\mathrm{n}$, before it is applied to the summation block.

The activation a, is given by

$$\mathrm{a} = \mathrm{w}_1\mathrm{x}_1 + \mathrm{w}_2\mathrm{x}_2 + \ldots \mathrm{w}_\mathrm{n}\mathrm{x}_\mathrm{n} + \theta. \tag{3.1}$$

This may be represented more compactly as

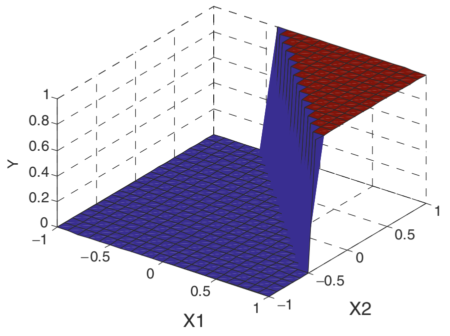
$$a = \sum_{i=1}^{n} X_i W_i + \theta, \tag{3.2}$$

the output y is then given by $\mathrm{y} = \mathrm{f(a)}$, where f is a activation function.

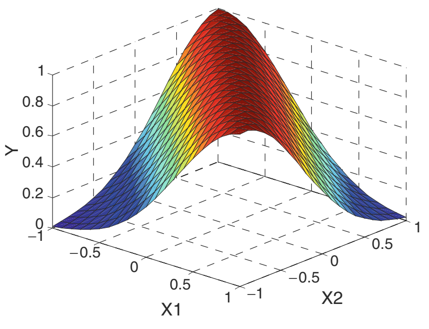
In McCulloh–Pitts Perceptron model hard limiter as activation function was used and defined as:

$$y = \begin{cases} 1 & \text{if a} >= ß \\ 0 & \text{if a} < ß \end{cases}$$

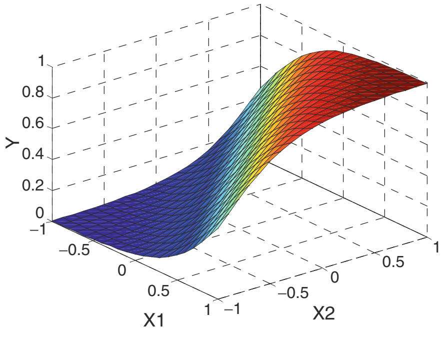
The threshold ß will often be zero. The activation function is sometimes called a *step-function*. Some more non-linear activation functions also tried by the researchers like sigmoid, Gaussian, etc. and the neuron responses for different activation functions shown in Fig. 3.3 with the Matlab program.



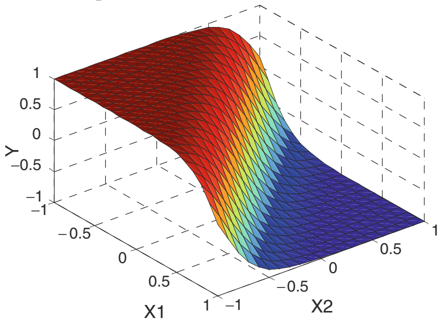
(a) hard-limiter Threshold function.



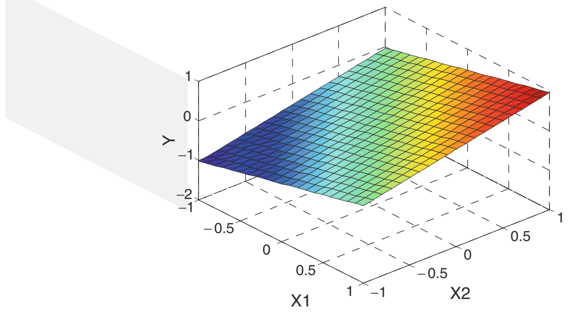
(c) Gaussian Threshold function.
Y = exp(− (x1+x2).^2);



(b) log-sigmoid Threshold function.
Y=1./(1+exp(− 3.0*(x1+x2)));



(d) tanh Threshold function.
Y = tanh(− 3*(x1+x2))



(e) Linear Threshold function.
Y=0.5*(x1+x2)

Fig. 3.3. Effect of different activation function on summation type simple neuron model

```
% 3-D surface generation program for simple neuron with
  different threshold functions
% Inputs
x1=-1:.1:1;
x2=-1:.1:1;
[m n]=size(x1);
Th=0;
for i=1:n
    for j=1:n
         sum=x1(i)+x2(j)-0.05;
         if sum>=Th      Y(i,j)=1 else Y(i,j)=0 end
    end
end
Y1=exp(-Y.^2);
surf(x1(1:n),x2(1:n),Y1)
```

3.4 Adaline

The next major development after the M & P neural model was proposed, occurred in 1949 when Hebb (1949) proposed a learning mechanism for the brain that became the starting point for artificial neural network learning (training) algorithms. He postulated that as brain learns, it changes its connectivity patterns. More specifically, his learning hypothesis is as follows: "When the axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that the A's efficiency, as one of the cells firing cell B, is increased." Hebb further proposed that if one cell repeatedly assists in firing another, the knobs of the synapse, are the junction, between the cells would grow so as to increase the area of contact. The Hebb's learning hypothesis is schematically shown in Fig. 3.4 (Levine 1983). This idea of learning mechanism was first incorporated in artificial neural network by (Rosenblatt 1958). He combined the simple M & P model with the adjustable synaptic weights based on Hebbian learning hypothesis to form the first artificial neural network with the capability to learn. The delta rule or the least mean squares (LMS) learning algorithm, was developed by Widrow and Hoff (1960). This model was called ADALINE for ADAptive LInear NEuron which is shown in Fig. 3.5. This learning algorithm first introduced the concept of supervised learning using a teacher which guides the learning process. It is the recent generalization of this learning rule into the backpropagation algorithm that has led to the resurgence in biologically based neural network research today. This states that if there is a difference between the actual output pattern and the desired output pattern during training, then the weights are changed to reduce the difference. The amount of change of weights is equal to the error on

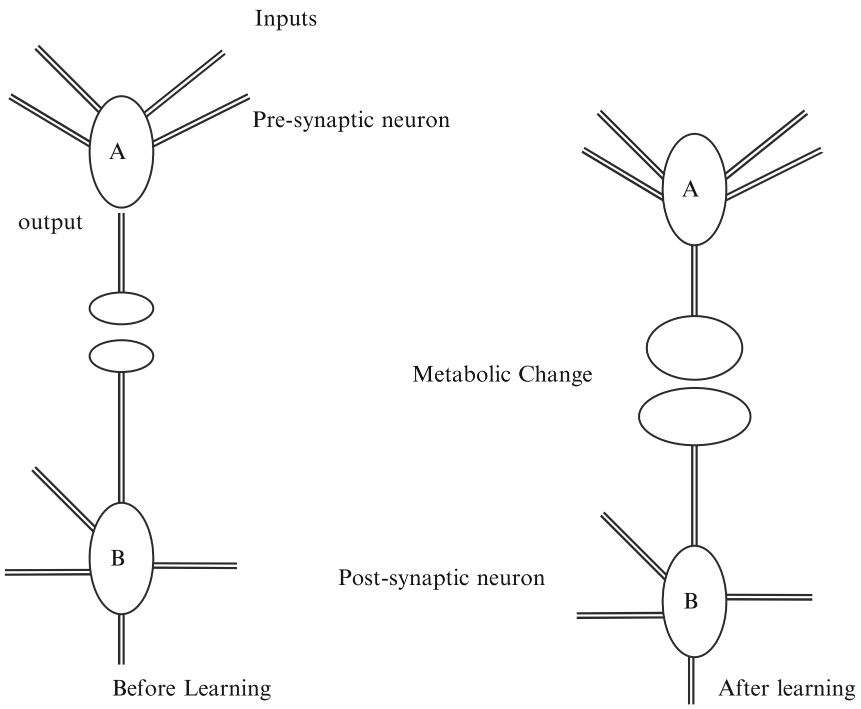


Fig. 3.4. Hebbian learning

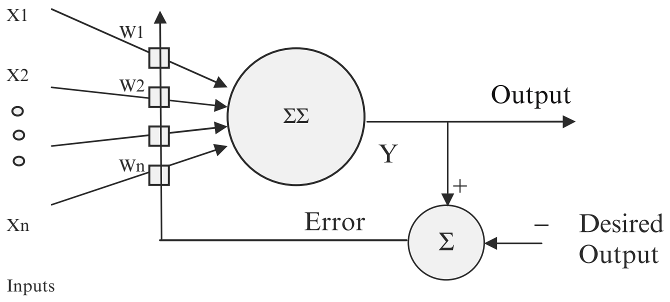


Fig. 3.5. ADLINE model

the outputs times the values of the inputs, times the learning rate. Many networks use some variation of this formula for training. In 1969 research in the field of artificial neural networks suffered a serious setback. Minsky and Papert published a book called Perceptrons (Minsky and Papert 1969) in which they proceed that single layer neural networks have limitations in their abilities to process data, and are capable of any mapping that is linearly separable. They pointed out, carefully applying mathematical techniques that the logical exclusive OR (XOR) function could not be realized by perceptrons. Further, Minsky and Papert argued that research into multi-layer neural network could be unproductive. Due to this pessimistic view of Minsky and Papert, the field of artificial neural networks entered into an almost total eclipse for nearly two decades. Fortunately, Minsky and Papert's judgement has been disproved; all non-linear separable problems can be solved by multi-layer perceptron networks. Nevertheless, a few dedicated researchers such as Kohonen, Grossberg, Anderson, Hopfield continued their efforts. A renaissance in the field of neural networks started in 1982 with the publication of the dynamic neural architecture by Hopfield (1982). This was followed by the landmark publication "Parallel Distributed Processing" by McClelland and Rumelhart (1986) who introduced into the back-propagation learning technique for multi-layer neural networks. Back-propagation, developed independently by Werbos (1974), provides a systematic means for training multi-layer neural networks. This development resulted in renewed interest in the field of neural networks and since mid-nineties a tremendous explosion of research has been occurring. Most of the neural network structures used presently for engineering applications is feed-forward neural networks (static). These neural networks comprising of a number of neurons respond instantaneously to the inputs. In other words, the response of static neural networks depends on the current inputs and the weights. The absence of feedback in static neural networks ensures that networks are conditionally stable. However, these networks suffer from the following limitations:

(1) In feed forward neural networks, where the information flows from A to B, to C, to D and never comes back to A. On the other hand, biological neural systems almost always have feedback signals about their functioning.
(2) The structure of the computational (artificial) neuron is not dynamic in nature and performs a simple summation operation. On the other hand, a biological neuron is highly complex in structure and provides much more computational functions than just summation.
(3) The static neuron model does not take into account the time delays that affect the dynamics of the system; inputs produce an instantaneous output with no memory involved. Time delays are inherent characteristics of biological neurons during information transmission.
(4) Static networks do not include the effects of synchronism or the frequency modulation function of biological neurons. In recent years, many researchers are involved in developing artificial neural networks to overcome the limitations of static neural networks mentioned above. Instead of summation as an aggregation function, the product (Π) is used as an aggregation function as shown in Fig. 3.6. The effect of different aggregation functions are also studied and shown in Fig. 3.7. The aggregation function could also be the combination of summation and product.

In product neuron the activation *a*, is given by

$$\mathrm{a} = \mathrm{w}_1\mathrm{x}_1{}^*\mathrm{w}_2\mathrm{x}_2{}^* \ldots \mathrm{w}_\mathrm{n}\mathrm{x}_\mathrm{n} + \theta.$$

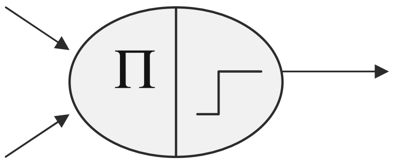
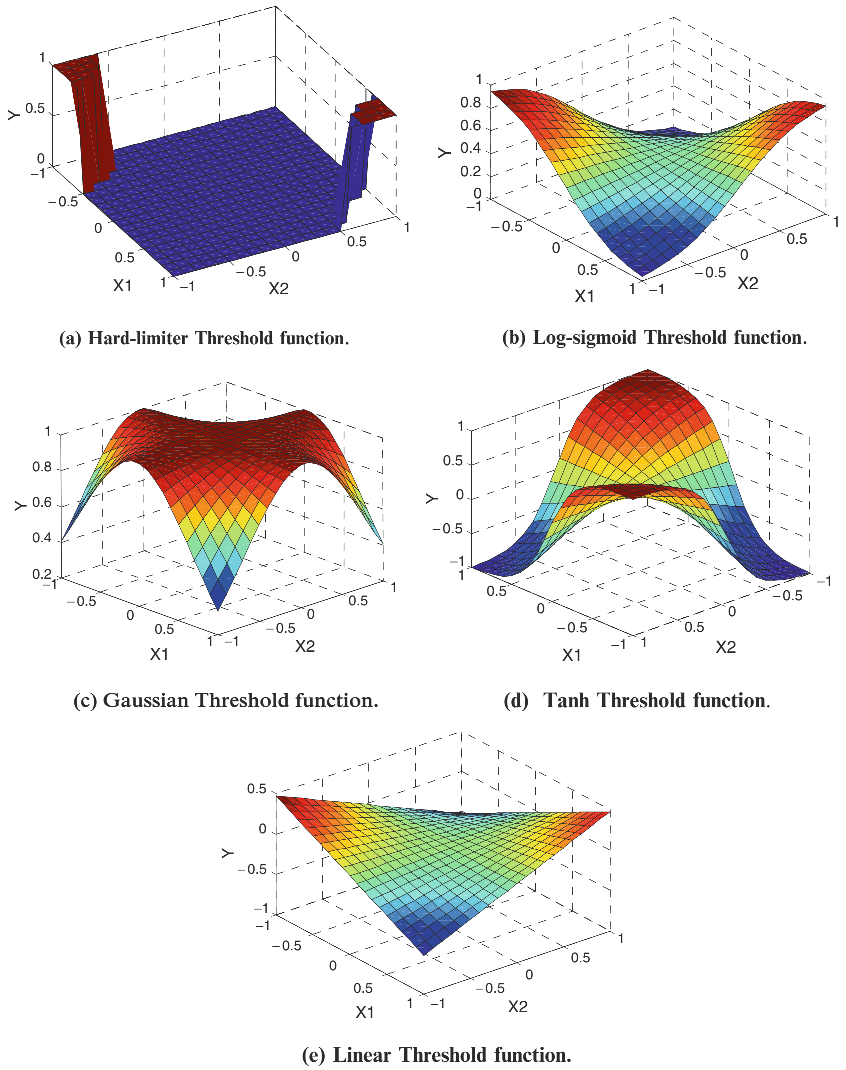


Fig. 3.6. Product type neuron



(a) Hard-limiter Threshold function.

(b) Log-sigmoid Threshold function.

(c) Gaussian Threshold function.

(d) Tanh Threshold function.

(e) Linear Threshold function.

Fig. 3.7. Effect of different activation functions on the product type neuron

This may be represented more compactly as

$$a = \prod_{i=1}^{n} X_i W_i + \theta,$$

the output y is then given by y = f(a).

A major characteristic of perceptron (summation type neuron) is the linear separation due to its threshold function. It can identify linearly separable regions easily as shown in Fig. 3.8.

To be able to divide the area into two regions for the problem in Fig. 3.8a, only one perceptron is required, since the whole area is divided into two separate regions by a single line. However, the threshold area in Fig. 3.8b is formed by many lines, thus we need more than one perceptron in different layers to generate a solution to this problem as shown below in Fig. 3.9.

Here, θ_1, θ_2 and θ_3 are the threshold values and w_{1j}, w_{2j}, and w_{3j} are the interconnection weights for the processing elements in the hidden layer, 1, 2 and 3, respectively. Each of the processing elements is connected to the third layer, which is the output perceptron through equal weight of 1, and a threshold value of 2.5 to be able to perform the operation. X and Y represent the x-y co-ordinates of the point selected from the region specified in Fig. 3.8b. Each perceptron separates the area into two regions by a line, but the solution is the intersection of these areas. Therefore, one more perceptron is needed to combine the outputs of these perceptrons to identify the marked area. This special perceptron combines the outputs from other perceptrons with unity weighing and the threshold value of "n–0.5", where n is the number of separation lines created.

The perceptron architecture is also able to perform an Exclusive OR operation, for example, in identifying the truth table of the region given in Fig. 3.10.

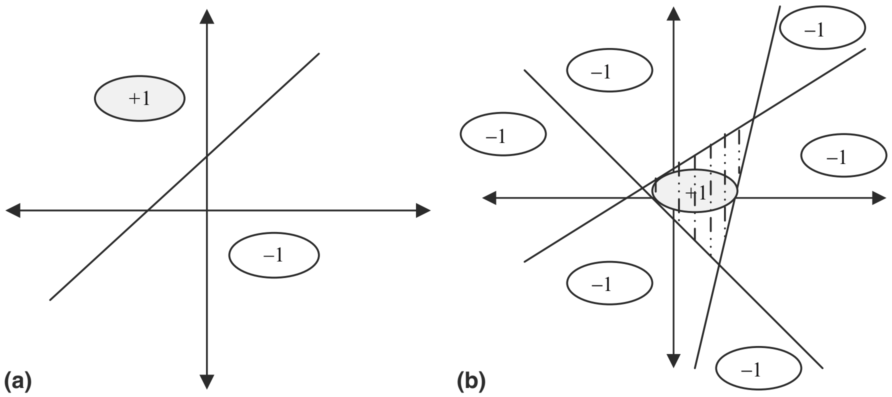


Fig. 3.8. (**a**) Two separate regions defined. (**b**) A selected region defined by a single perceptron intersection of areas created by perceptrons

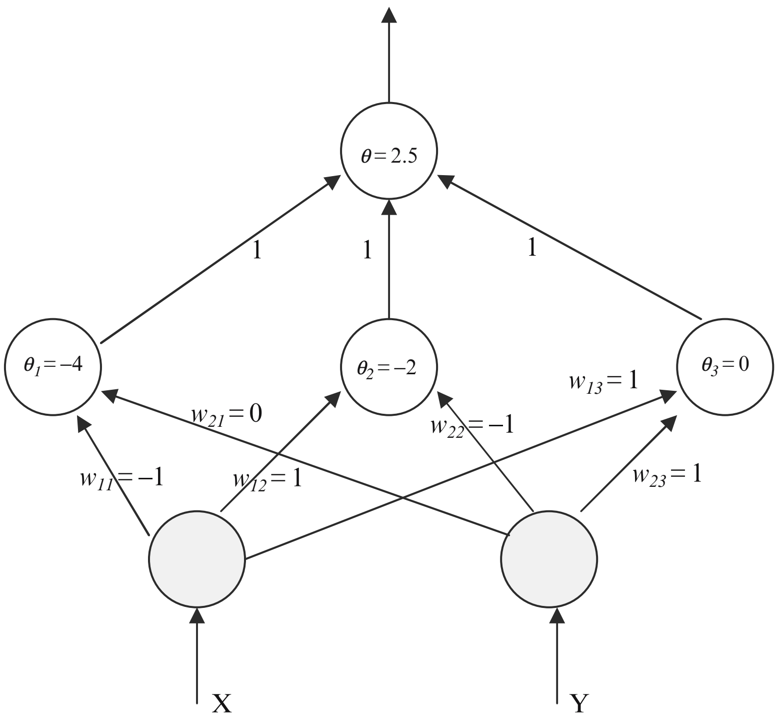
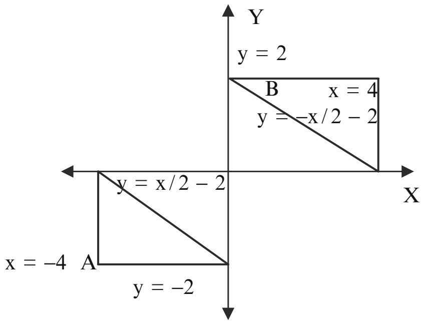


Fig. 3.9. A solution perceptron network



=

Truth Table		
Point 1	Point 1	Outcome
In A	In B	0
In A	Out B	1
Out A	In B	1
Out A	Out B	0

Fig. 3.10. The exclusive-OR problem

Lets identify the region A using a three-layer perceptron network. Hence we need to compute the weights, w_{ij}s associated with the interconnections between the input units, which represent the x-y co-ordinates of the points given in Fig. 3.8b and a corresponding threshold value, θ_i to be able to make the distinction between the two regions identified as $+1$ and -1 in the figure. These values also correspond to the output values of perceptron model as shown in Figs. 3.11–3.15.

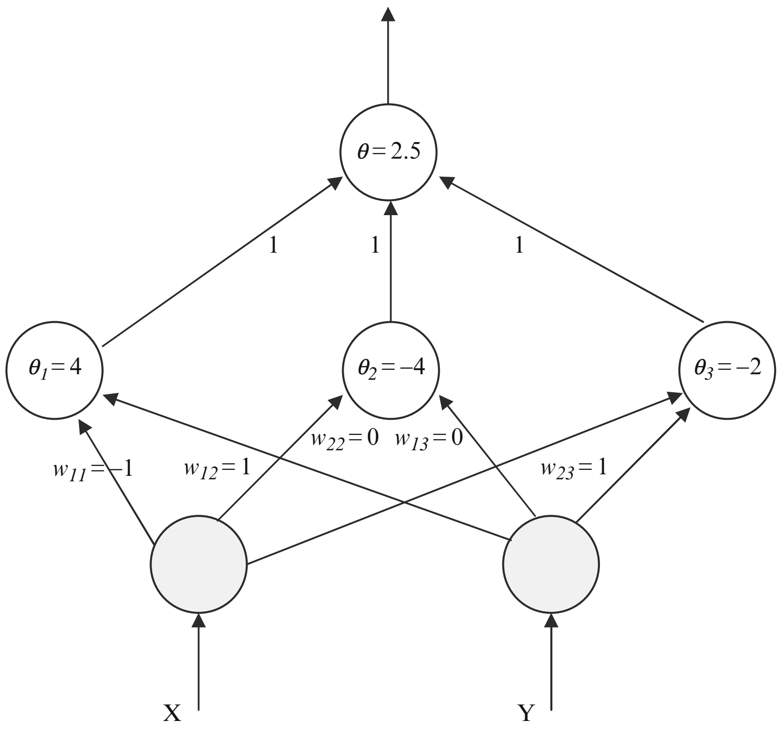


Fig. 3.11. Perceptron network for region A

$$\begin{aligned}&y < -x/2 - 2; \quad x > -4; \quad y > -2\\ &x/2 + y + 2 < 0;\ x + 4 > 0;\ y + 2 > 0\\ &-x - 2y - 4 > 0\end{aligned}$$

Similarly for region B;

$$\begin{aligned}&y > -x/2 + 2; \quad x < 4; \quad y < 2\\ &x + 2y - 4 > 0;\ -x + 4 > 0;\ -y + 2 > 0\end{aligned}$$

To be able to produce result as in the truth table shown in Fig. 3.6, which is an Exclusive OR operation, we need another three-layer perceptron as shown in Fig. 3.9.

Thus the entire solution architecture is as shown above.

```
% Matlab Program for solving OR problem
clc; clear all;
X=[0 0 1 1;
    0 1 0 1];                         % Row wise inputs
D=[0 1 1 1];                          % Row wise output
```

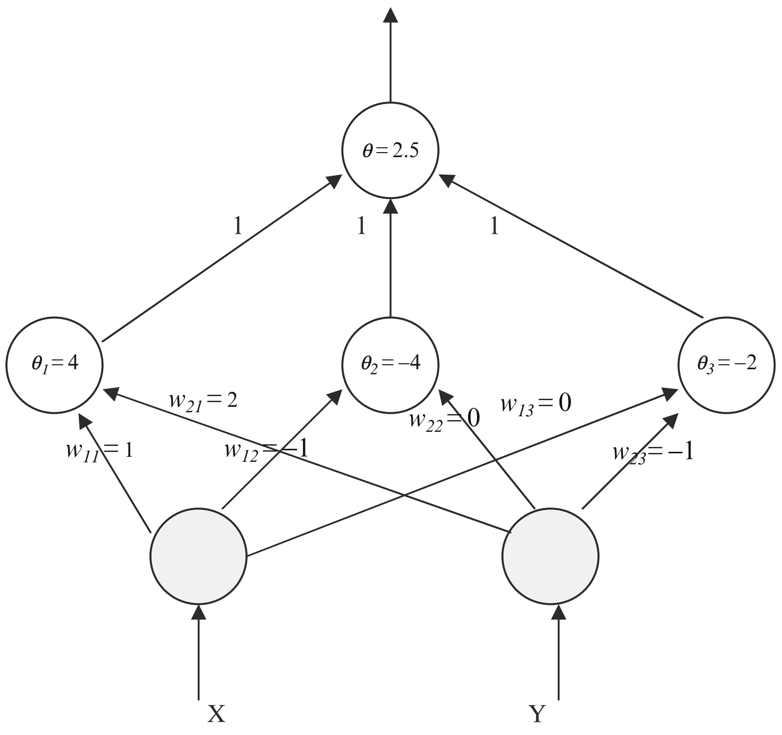
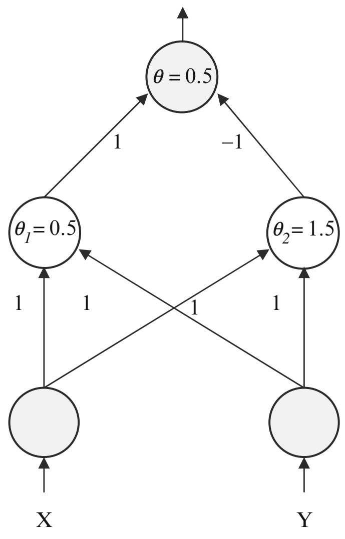


Fig. 3.12. Perceptron network for region B



Truth Table		
x	y	Output
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 3.13. Perceptron network for XOR

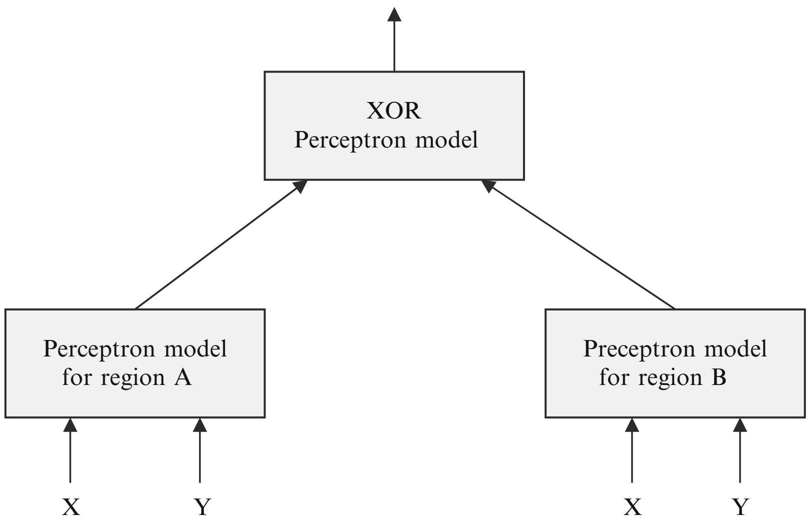


Fig. 3.14. Combined architecture for the subject problem

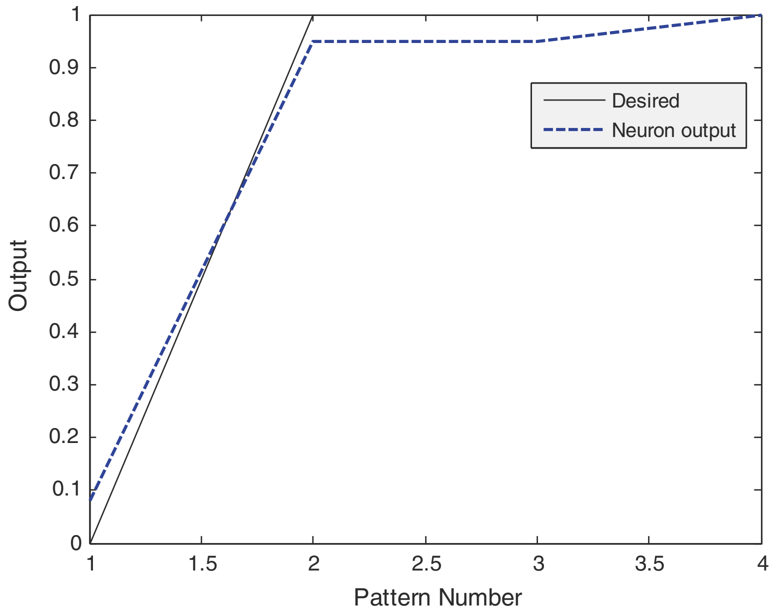


Fig. 3.15. Comparison of neuron output with actual results

```
% Initialization
W=randn(2,1);                   % Weight
B=randn(1,1);                   % Bias
LR=0.9;                         % learning rate
MF=0.1;                         % momentum factor
dW=zeros(2,1);dB=0;             % change in weight
for i=1:500
  wSum=W'*X+B;                  % Calculating activation
  O=1./(1+exp(-wSum));          % Output of neuron
  e=(D-O);                      % Error calculation
  sse=sum(e.^2)/2;              % Sum squared error
  de=(e.*O.*(1-O));             % Derivative of error
  dW=LR*X*de'+MF* dW;
  dB=LR*sum(de')+MF*dB;
  B=B+dB;                       % New value of bias
  W=W+dW                        % New value of weight
end
wSum=W'*X+B;                    % Testing
O=1./(1+exp(-wSum))
plot(D);
hold
plot(O, '--')
xlabel('Pattern Number')
ylabel ('Output')
```

The neurons or processing elements of an ANN are connected together and the overall system behaviour is determined by the structure and strength of these connections. This network structure consists of the processing elements arranged in groups or layers. A single structure of interconnected neurons produces an auto-associative system and is often used as a content-addressable memory. Connections between same neurons are referred as *lateral connections* and those that loop back and connected to the same neuron are called *recurrent connections*. Multi-layered systems contain input and output neuron layers that receive or emit signals to the environment and neurons, which are neither, called *hidden neuron layers*. This hidden layer provides networks with the ability to perform non-linear mappings as well as contributing to the complexity of reliable training of the system. Inter-field connections or connections between neurons in different layers can propagate signals in one of two ways:

- Feed-forward signals only allow information to flow along connections in one directions, while
- Feedback signals allow information to flow in either direction and/or recursively.

Artificial neural networks can provide content-addressable memory (CAM), which stores data at stable states in the weight matrix, and associative memory (AM), which provides output responses from, input stimuli. In this hetero-associative memory system, the ANN recall mechanism is a function g(.) that takes the weight matrix **W** and an input stimulus Input_k, and produces the output response Output_k. The two primary recall mechanisms are

1. Nearest-neighbour recall and
2. Interpolative recall.

Nearest-neighbour recall finds the stored input that closely matches the stimulus and responds with the corresponding output using a distance measure such as Hamming or Euclidean distance.

Interpolative recall takes the stimulus and interpolates the entire set of stored inputs to produce the corresponding output.

3.5 ANN Learning

Learning in an ANN is defined as any progressive systematic change in the memory (weight matrix) and can be supervised or unsupervised.

Unsupervised learning or self-organisation is a process that does not incorporate any external teacher and relies only upon local information and internal control strategies. Examples include:

- Adaptive Resonance Theory (ART1 and ART2) (Carpenter and Grossberg 1988)
- Hopfield Networks (Hopfield 1982)
- Bi-directional Associative Memory (BAM) (Kosko 1987, 1988)
- Learning Vector Quantisation (LVQ) (Kohonen 1988, 1997)
- Counter-Propagation networks (Hecht-Nielsen 1987, 1988, 1990).

Supervise learning, which includes:

- Back-propagation
- The Boltzmann Machine (Ackley et al. 1985)

It incorporates an external teacher and/or global information and includes such techniques as error-correction learning, reinforcement learning and stochastic learning. *Error-correction learning* adjusts the correction weight matrix in proportion to the difference between the desired and the computed values of each neuron in the outer layer. *Reinforcement learning* is a technique by which the weights are reinforced for properly performed actions and punished for inappropriate ones where the performance of the outer layer is captured in a single scalar error value. *Stochastic learning* works by making a random change in the weight matrix and then determining a property of the network called the *resultant energy*. If the change has made this energy value lower than it was previously, then the change is accepted, otherwise the change is accepted according to a pre-chosen probability distribution. This random acceptance of change that temporarily degrades the performance of the system allows it to escape from local energy minima in its search for the optimal system state.

In a multi-layered net using supervised learning, the input stimuli can be recorded into an internal representation and the outputs generated are then representative of this internal representation instead of just the original pattern pair. The network is provided with a set of example input–output pairs (a training set) and the weight matrix modified so as to approximate the function from which the training set has been derived. In the ideal case, the net would be able, after training, to generalise or produce reasonable results for input simulation that it has never been exposed to. Currently, the most popular technique for accomplishing this type of learning in an ANN is the multi-layered perceptron employing back-propagation. It evolved from Rosenblatt's perceptron; a two-layered supervised ANN, which provides nearest neighbour pattern matching via the perceptron error-correction procedure. This procedure effectively works to place a hyper plane between two classes of data in an n-dimensional pattern space. It has been shown that this algorithm will find a solution for any linearly separable problem in a finite amount of time.

3.6 Back-Propagation Learning

Error back-propagation through non-linear systems has existed in variational calculus for many years but the first application of gradient descent to the training of multi-layered nets was proposed by Amari (1967) who used a single hidden layer to perform a non-linear classification. Werbos (1974) discovered *dynamic feedback* and Parker and Chau (1987) talked about *learning logic*, but the greatest impact on ANN field came when Rumelhart, Hinton and Williams published their version of the *Back-propagation algorithm*.

One of the major reasons for the development of the back-propagation algorithm was the need to escape one of the constraints on two layer ANNs, which is that similar inputs lead to similar output(s). But, while ANNs like the perceptron may have trouble with non-linear mappings, there is a guaranteed learning rule for all problems that can be solved without hidden units. Unfortunately, it is known that there is no equally powerful rule for multi-layered perceptrons.

The simplest multi-layered perceptron implementing back-propagation is a three-layered perceptron with feed-forward connections from the input layer to the hidden layer and from the hidden layer to the output layer. This function-estimating ANN stores pattern pairs using a multi-layered gradient error correction algorithm. It achieves its internal representation of the training set by minimising a *cost function*. The most commonly used cost function is the sum squared error or the summation of the difference between the computed and desired output values for each output neuron across all patterns in the training set. Other cost functions include the Entropic cost function, Linear error and the Minkuouski-r or the rth power of the absolute value of the error.

In all cases, the changes made to the weight matrix are derived by computing the change in the cost function with respect to the change in each weight. The most basic version of the algorithm minimises the sum-squared error and is also known as the *generalised delta rule* (Simpson 1990).

Step 1: Assign small random weights to all weights on connections between all layers of the network as well as to all neuron thresholds. The activation used is logistic sigmoid function as given by (3.2) with $\lambda = 1$.

Step 2: For each pattern pair in the training set:

(a) Read the environmental stimuli into the neurons of the input layer and proceed to calculate the new activations for the neurons in the hidden layer using

$$hidden_i = f\left(\sum_{h=1}^{n} input_h \; w_{hi} + \theta_i\right) \tag{3.3}$$

where $f(.)$ is the activation function, there are n input neurons and θ_i is the threshold for the ith hidden neuron.

(b) Use these new hidden layer activations and the weights on the connections between the hidden layer and the output to calculate the new output activations using

$$output_j = f\left(\sum_{i=1}^{n} hidden_i \; w_{ij} + \Gamma_j\right) \tag{3.4}$$

where $f(.)$ is the activation function, there are n hidden neurons and Γ_j is the threshold for the jth output neuron.

(c) Determine the difference between the computed and the desired values of the output layer activations using

$$diff_j = output_j\,(1 - output_j)\,(desired_j - output_j) \tag{3.5}$$

and calculate the error between each neuron in the hidden layer relative to the $diff_j$ using

$$err_i = hidden_i\,(1 - hidden_i)\sum_{i=1}^{n} w_{ij}\,diff_j \tag{3.6}$$

(d) Modify each connection between the hidden and output layers, and $\Delta w_{ij} = \alpha \; hidden_i \; diff_j$, which is the amount of change to be made to the weight on the connection from the ith neuron in the hidden layer to the jth neuron in the output layer. α is a positive constant that controls the rate of modification or learning.

(e) Perform a similar modification to the weights on the input to hidden layer connections with $\Delta w_{hi} = \beta \; err_i$ for the hidden units and $\Delta\Gamma_j = \alpha \; diff_j$ for the output units.

Step 3: Repeat Step 2 until all the diff_js are either zero or sufficiently low.

Step 4: After the BP-ANN has been trained; recall consists of two feed-forward operations, which create hidden neuron values.

$$hidden_i = f\left(\sum_{h=1}^{n} input_i \; w_{hi} + \theta_j\right) \tag{3.7}$$

and then we use them to create new output neuron values

$$output_j = f\left(\sum_{i=1}^{n} hidden_i \; w_{ij} + \Gamma_j\right) \tag{3.8}$$

Back-propagation is guaranteed only to find the local, not the global error minimum. And while this technique has proven extremely successful for many practical applications, it is based on gradient descent, which can proceed very slowly because it is working only with local information. Practical implementation factors that must be considered include:

- The number of units in the hidden layer.
- The value of the learning rate constants.
- The amount of data that is necessary to create the proper mapping.

Once these issues have been addressed, the power of back-propagation is realised in a system that has the ability to store many more patterns that the number of dimensions inherent in the size of its input layer. It also has the ability to acquire arbitrarily complex non-linear mappings. This is possible if the application allows for a reasonably long training time in an off-line mode.

Current research in the area of back-propagation improvements is looking at:

- Optimising the number of units in the hidden layer and the effect of the inclusion of more than one layer of hidden units.
- Improving the rate of learning by dynamic manipulation of the learning rates and by the use of techniques such as momentum.
- The effects of dynamically changing and modular connection topologies.
- Analysing the scaling and generalisation properties of this ANN model.
- Employing higher-order correlations and arbitrary threshold functions.

During training the nodes in the hidden layers organize themselves such that different nodes learn to recognize different features of the total input space.

During the recall phase of operation the network will respond to inputs that exhibit features similar to those learned during training. Incomplete or noisy inputs may be completely recovered by the network.

In its learning phase, you give it a training set of examples with known inputs and outputs.

An overview of training

The objective of training the network is to adjust the weights so that application of a set of inputs produces the desired set of outputs. For reasons of brevity, these input–output sets can be referred to as vectors. Training assumes that each input vector is paired with a target vector representing the desired output; together these are called a training pair. Usually, a network is trained over a number of training pairs. For example, the input part of a training pair might consist of a pattern of ones and zeros representing a binary image of a letter of the alphabet. A set of inputs for the letter A drawn on a grid. If a line passes through square, the corresponding neuron's input is one; otherwise, that neuron's input is zero. The output might be a number that represents the letter A, or perhaps another set of ones and zeros that could be used to produce an output pattern. If one wished to train the network to recognize all the letters of the alphabet, 26 training pairs would be required. This group of training pairs is called a training set.

Before starting the training process, the weights must be initialized to small random numbers. This ensures that the network is not saturated by large values of the weights, and prevents certain other training pathologies. For example, if the weights all start at equal values and the desired performance requires unequal values, the network will not learn.

Training the back-propogation network requires the steps that follow:

Step 1. Select the training pair from the training set; apply the input vector to the network input.
Step 2. Calculate the output of the network.
Step 3. Calculate the error between the network output and the desired output (the target vector from the training pair).
Step 4. Adjust the weights of the network in a way that minimizes error.
Step 5. Repeat steps 1 through 4 for each vector in the training set until the error for the entire set is acceptably low.

The operations required in steps 1 and 2 above are similar to the way in which the trained network will ultimately be used; that is, an input vector is applied and the resulting output is calculated. Calculations are performed on layer-by-layer basis.

In step 3, each of the network outputs is subtracted from its corresponding component of the target of the network, where the polarity and magnitude of the weight changes are determined by the training algorithm.

After enough repetitions of these four steps, the error between actual outputs and target outputs should be reduced to an acceptable value, and the network is said to be trained. At this point, the network is used for recognition and weights are not changed.

It may be seen that steps 1 and 2 constitute "forward pass" in that the signal propagates from the network input to its output. Steps 3 and 4 are a "reverse pass"; here the calculated error signal propagates backward through the network where it is used to adjust weights. These two passes are now expanded and expressed in a somewhat more mathematical form in Chap. 4.

% Matlab Program for Backpropagation for single hidden layer

% Input–output Pattern for EX-OR problem

```
clear all;
clc;
X=[0.1 0.1 0.1; 0.1 0.9 0.9; 0.9 0.1 0.9; 0.9 0.9 0.1];
% Training parameters
eta=1.0; % learning rate
alpha=0.6; % Momentum rate
err_tol=0.001; % Error tolerance
[row_x col_x]=size(X);
sum_err=0;
% ANN architecture
In=2; % number of input neurons
Hn=2; % number of hidden neurons
On=1; % number of output neurons
% Weight/delta weight Intialization
Wih=2*rand(In+1,Hn)-1;
Who=2*rand(Hn+1,On)-1;
DeltaWih=zeros(In+1,Hn);
DeltaWho=zeros(Hn+1,On);
deltaWihold=zeros(In+1,Hn);
deltaWhoold=zeros(Hn+1,On);
deltah=zeros(1,Hn+1);
deltao=zeros(1,On);
X_in=[ones(row_x,1) X(:,1:In)];
D_out=X(:,1:On);
sum_err=2*err_tol;
while (sum_err>err_tol)
 sum_err=0;
 for i=1:row_x
  sum_h=X_in(i,:)*Wih;
  out_h=[1 1./(1+exp(-sum_h))];
  sum_o=out_h* Who;
  out_o=1./(1+exp(-sum_o));
  error=D_out(i) - out_o;
  deltao=error.*out_o.*(1-out_o);
  for j=1:Hn+1
   DeltaWho(j,:)=deltao*out_h(j);
  end
  for k=2:Hn+1
   deltah(k)=(deltao*Who(k,:)')*out_h(k)*(1-out_h(k));
  end
  for l=2:In+1
   deltaWih(l,:)=deltah(2:Hn+1)*X_in(i,l);
  end
  Wih=Wih+DeltaWih+alpha*deltaWihold;
  Who=Who+DeltaWho+alpha*deltaWhoold;
  deltaWihold=DeltaWih;
  deltaWhoold=DeltaWho;
 sum_err=sum_err+sum(error.^2);
 end
 sum_err
end
```

Summary of back-propagation training:

- Objective is to find the global minimum on the error surface.
- Solution is obtained through gradient descent algorithm and ANN weights are adjusted to follow the steepest downhill slope.
- The error surface is not known in advance, so explore it in many small steps and the possibility to stuck in local minima is always there as shown in Fig. 3.16.

The algorithm finds the nearest local minimum, not always the global minimum. There can be two causes for this:

a. *Over-fitted ANN*
   The overfitting of data is a common problem found in ANN during approximating a function, specially when ANN has too many weights. Too many weights (free parameters) in ANN approximate the function very accurately, but the generalization capability for unforeseen data is not so good. On the other hand, a network with too few weights will also give poor generalization capability as the ANN has very low flexibility and is

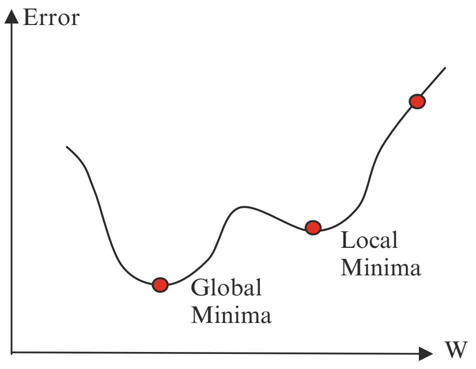


Fig. 3.16. Error curve with respect to weight

unable to approximate the function. Hence, there is a trade off between the number of training data and the size of the network.

b. *Too many hidden nodes*

- One node can model a linear function
- More nodes can model higher-order functions, or more input patterns
- Too many nodes model the training set too closely, preventing generalization.

This problem could be resolved by optimizing the ANN size.

3.7 Properties of Neural Networks

1. Neural networks are inherently parallel and implementation can be done on parallel hardware.
2. It has a capacity for adaptation.
3. In neural networks "memory" corresponds to an activation map of the neurons. Memory is thus distributed over many units giving resistance to noise. In distributed memories, such as neural networks, it is possible to start with noisy data and to recall the correct data.
4. Fault tolerant capability
   Distributed memory is also responsible for fault tolerance. In most neural networks, if some neurons are destroyed or their connections altered slightly, then the behavior of the network as a whole is only slightly degraded. The characteristic of *graceful degradation* makes neural computing systems extremely well suited for applications where failure of control equipment means disaster.
5. Capacity for generalization
   Designers of expert systems have difficulty in formulation rules which encapsulate an expert's knowledge in relation to some problem. A neural system may learn the rules simply from a set of examples. The generalization capacity of a neural network is its capacity to give a satisfactory response for an input which is not part of the set of examples on which it was trained. The capacity for generalization is an essential feature of a classification system. Certain aspects of generalization behavior are interesting because they are intuitively quite close to human generalization.
6. Ease of construction.

3.8 Limitations in the Use of Neural Networks

1. Neural systems are inherently parallel but are normally simulated on sequential machines.
   - ◦ Processing time can rise quickly as the size of the problem grows.
   - ◦ A direct hardware approach would lose the flexibility offered by a software implementation.

2. The performance of a network can be sensitive to the quality and type of preprocessing of the input data.
3. Neural networks cannot explain the results they obtain; their rules of operation are completely unknown.
4. Performance is measured by statistical methods giving rise to distrust on the part of potential users.
5. Many of the design decisions required in developing an application are not well understood.

Fruit identification problem

This is a very simple example of identification of fruits. The inputs for this problem are shape, size and colour of fruits.

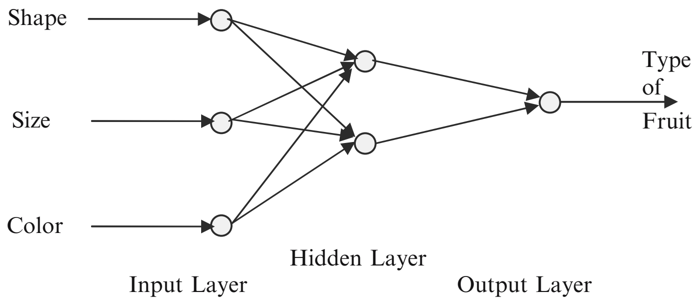
Step-1 What the neural network is to learn?
Input 1 Shape = {Round, Large}
Input 2 Size = {Small, Large}
Input 3 Colour = {Red, Orange, Yellow, Green}.
Output Type of fruit = {Grape, Apple, Cherry, Orange, Banana}.
Step-2 Pre-processing of data

a. Representation of data
Neural Network could not work with qualitative information. Hence the input must be converted into quantitative information like 0 and 1.
Shape = {0,1} zero stands of round and 1 for large.
Size = {0,1} small is zero and large is 1.
Colour = {0.0, 0.25, 0.5, 0.75, 1.0}.
Where 0.0 - Red; 0.25 - Orange, 0.5 - Yellow, 0.75 - Green.
Output = {0, 0.25, 0.5, 0.75, 1.0}.
Where 0.0 Grape, 0.25 - Apple, 0.5 - Cherry, 0.75 - Orange, 1.0 Banana.
b. Sequence of presentation of data

Input 1 shape	Input 2 size	Input 3 color	Ouptut - fruit
0 - Round	0 - Small	0.0 - Red	0 - Grape
0 - Round	0 - Small	0.25 - Orange	0.5 - Cherry
0 - Round	1 - large	0.25 - Orange	0.75 - Orange
1 - Large	1 - Large	0.5 - Yellow	1.0 - Banana
0 - Round	1 - Large	0.75 - Green	0.25 - Apple

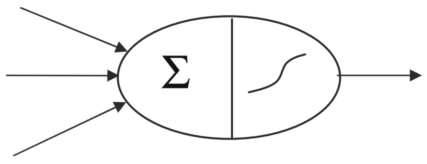
The inputs need not be the exact value as given in the table; we could assign some other value depending on the situation. It also helps us to incorporate the uncertainty in the model.

Step-3 Define Network Structure –
Number of input layer neurons = 3 (Number of inputs)
Number of output layer neuron = 1 (Number of outputs)
Number of Hidden layer neurons = 2 (generally average of input and output neurons)



Neural Network to recognize Fruits

Step-4 Selection of Neuron Structure



Normally the neuron structure is consisting of summation as aggregation function and sigmoid as threshold function.

Step-5 Usually the network starts with random weight in the range (-0.1 to $+0.1$). Sometimes to reduce the training time, the initialization of weights is done with evolutionary algorithms.

Step-6 Training Algorithms and Error Function selection
Generally, gradient descent back-propagation training algorithm is used with or without adaptive learning and momentum factors. In this the sum squared error is fed back to modify the weight during training.

Step-7 Decision regarding selection of training parameters
Training parameters are

a. Number of epochs = 100 (number of iterations required to reach to the desired goal)
b. Error tolerance = 0.001 (depends on the accuracy required)
c. Learning rate = 0.9 (near 1)
d. Momentum facto = 0.1 (smaller)

Step-8 Training and Testing of Network
The network is trained for the above given data and then test it to check its performance. Generally, the testing data is slightly different from training data (10% new data).

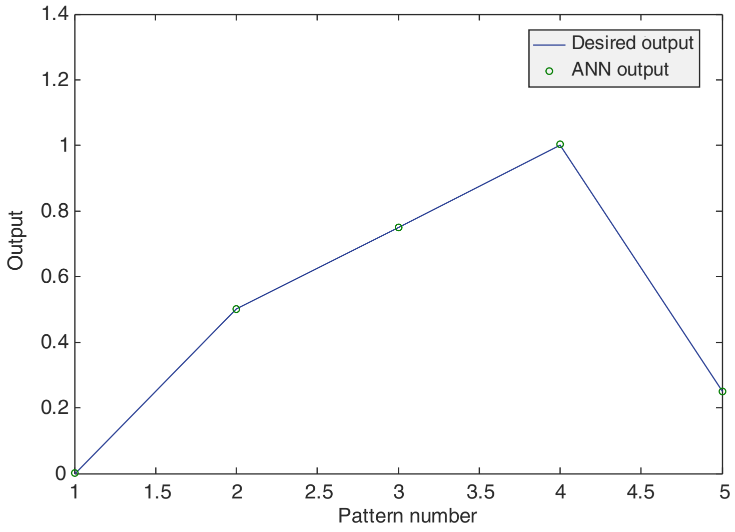
Step-9 Use the trained network for prediction

% Matlab Program for identification of fruit type

```
clc;   clear all;
% Column wise input--output patterns
```

```
x=[0 0 0.0 0; 0..0 .0.25 0.5; 0..1 ..0.25 ..0.75;
   1...1..1.5 ..1.0; 0 ..1 ..0.75....0.25];
P=x(:,1:3)';
T=x(:,4)';
net=newff(minmax(P), [2 1],
          {'tansig' 'purelin'});          % defining~ANN
net.trainParam.epochs=100;                % Define number of epochs
net.trainParam.goal=0.0001;               % Define error~goal
net=init(net)                             % Initialize weights
net=train(net,P,T);                       % Training
Y=sim(net,P);                             % Testing
plot(1:5,T,'-',1:5,Y,'o')                 % Ploting the results
```

Results



3.9 Summary

1. It is clear that the quantum of processing that takes place in biological neurons is far more complex. It integrates hundreds or thousands of temporal signals through their dendrites.
2. Artificial neuron is similar to biological neurons and receives weighted input, which passes through aggregation function and activation function. A hard limiter constitutes the nonlinear element of McCulloch – Pitts neuron.

3. Artificial neural network (ANN) consists of artificial neurons in two different architectures: feed forward and feedback. Feedforward networks are static and the output depends only on input, but in feedback ANN output is also feedback and therefore, it is dynamic in nature.

3.10 Bibliography and Historical Notes

The pioneer work in the area of neural network was done by McCulloch and Pitts way back in 1943. McCulloch was a psychiatrist and neuroanatomist by training. He spent nearly two decades in understanding the event in the nervous system. Pitts was a mathematical prodigy. McCulloch and Pitts developed neuron model at the University of Chicago.

The next major development in the field of neural network came in 1949, when Hebb wrote a book on *The Organization of Behavior*. He proposed that the connections strength between neurons change while learning. Rochester et al. (1956) is probably the first attempt to formulate neural learning theory based on Hebb's work.

Minsky submitted his doctorate thesis at Princeton University on the topic of *Theory of Neural – Analog Reinforcement Systems and Its Application to the Brain-Model Problem* in 1954. Then he published a paper on *Steps Toward Artificial Intelligence*. The significant contributions to the early development of associative memory papers by Taylor (1956), Anderson (1972), and Kohonen (1972).

In 1958 Rossenblatt introduced a novel method of supervised learning. In 1960 Widrow and Hoff gave least mean square (LMS) algorithm to formulate ADALINE. Later on Widrow and his students developed MADALINE. The books by Wasserman Philip (1989) and Nielsen (1990) also contain treatment of back propagation algorithms. Minsky and Papert (1969) demonstrated the fundamental limitations of perceptron.

In 1970s self-organizing maps using competitive learning was introduced (Grossberg 1967, 1972). Carpenter and Grossberg also developed adaptive resonance theory (ART) in 1980 and used it for pattern recognition (Carpenter and Grossberg 1987, 1988, 1990, 1996). Hopfield used an energy function to develop recurrent networks with symmetric synaptic connections. Rumelhart et al. (1986) developed back propagation algorithm. In early 1990s, Vapnik et al. invented a computationally powerful class of supervised learning networks called support vector machines for different applications. An excellent review article is that by Lippmann (1987). Kosko (1988) discusses on bidirectional associative memory (BAM).

3.11 Exercises

1. Explore the method of steepest descent involving a single weigh w by considering the following cost function:

   $\mathrm{f(w)} = 0.7\ \mathrm{A} + \mathrm{W}^{*}\mathrm{B} + \mathrm{C}^{*}\mathrm{W}^{2}$ where A, B, and C are constants.

2. The function expressed by $f(x) = 1/x^2$
   a. Write a matlab program to generate the two sets of data:
      1. Training data
      2. Testing data.
   b. Use a three layer network and train it with back propagation learning algorithm for the data generated in part a. Consider the error tolerance 0.01.
   c. Test the network for generated testing data.
   d. Compare the network performance for
      1. Two or more hidden layers in the network.
      2. Two or more neurons in each hidden layer.
3. In question 2, study the effect of starting (initial) weights.
4. A sigmoid function is $f(x) = 1/(1 + e^{-\lambda x})$. Find its inverse function and plot both the function and its inverse for different values of λ.
5. Find the derivative of the above mentioned function with respect to x.
6. Find appropriate weights and threshold of neuron for logical AND problem.
7. Solve the 3-bit even parity problem with three layer ANN using 3-hidden neurons. Write the matlab program to solve the parity problem.