5

Development of Generalized Neuron and Its Validation

More recently, ANNs and fuzzy set theoretic approach have been proposed for many different industrial applications. A number of papers have been published in the last two decades. An illustrative list is given in bibliography. Both techniques have their own advantages and disadvantages. The integration of these approaches can give improved results.

In the previous chapter, the performance aspect of ANN has been discussed in detail. To overcome some of the problems of ANN and improve its training and testing performance, the simple neuron is modified and a generalized neuron is developed in this chapter.

In the common neuron model generally the aggregation function is summation, which has been modified to obtain a generalized neuron (GN) model using fuzzy compensatory operators as aggregation operators to overcome the problems such as large number of neurons and layers required for complex function approximation, which not only affect the training time but also the fault tolerant capabilities of the artificial neural network (ANN) (Chaturvedi 1997).

5.1 Existing Neuron Model

The general structure of the common neuron is an aggregation function and its transformation through a filter. It is shown in the literature (Widrow and Lehr 1990) that the ANNs can be universal function approximators for given input–output data. The common neuron structure has summation or product as the aggregation function with linear or nonlinear (sigmoid, radial basis, tangent hyperbolic, etc.) as the threshold function as shown in Fig. 5.1.

If variation at aggregation is only considered at the neuron level, two types of neurons are possible:

1. Summation type neuron ($\sum$ - Neuron)
   In summation type neuron summation function at aggregation level and sigmoid function at activation level is considered as shown in Fig. 5.1a.

D.K. Chaturvedi: *Soft Computing Techniques and its Applications in Electrical Engineering*, Studies in Computational Intelligence (SCI) **103**, 87–122 (2008)
www.springerlink.com

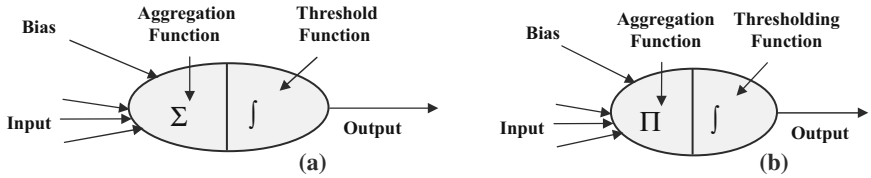


Fig. 5.1. (**a**) Simple summation neuron model. (**b**) Simple product neuron model

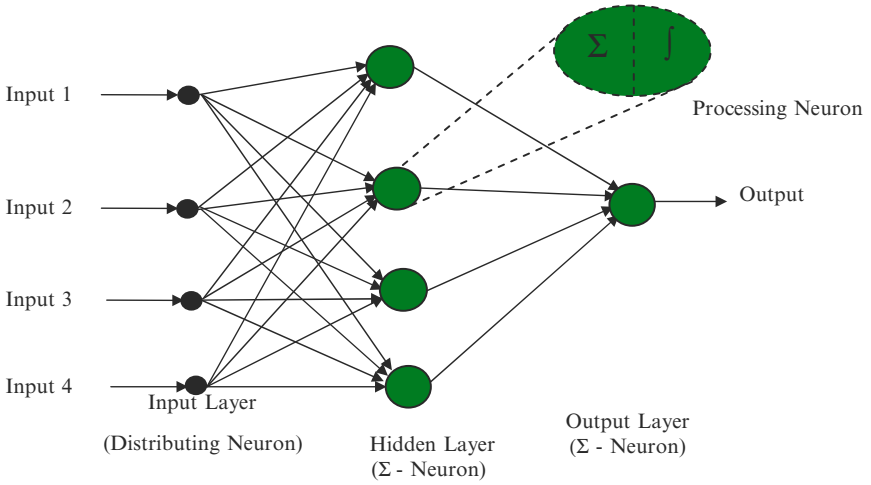


Fig. 5.2. Summation type neural network ($\sum$ - ANN)

2. Product type neuron (Π – Neuron)
   It consists of product function at aggregation level and sigmoid function at activation level as shown in Fig. 5.1b.

Now using these neuron models four type of neural network could be developed:

a. Summation type neural network ($\sum$ - ANN)
   It contains all summation neuron at hidden layer as well as output layer as shown in Fig. 5.2.
b. Product type neural network (Π – ANN)
   It is made up of all product type neurons at both hidden layer and output layer as shown in Fig. 5.3.
c. Mixed type neural network
   In mixed type neural networks both summation and product type neurons could be kept in two ways in the network as mentioned below:
   1. Summation – product type neural network ($\sum -\Pi$ – ANN)
      Here summation type ($\sum$) neuron is considered at the hidden layer and product type (Π) neuron is considered at the output layer as shown in Fig. 5.4.

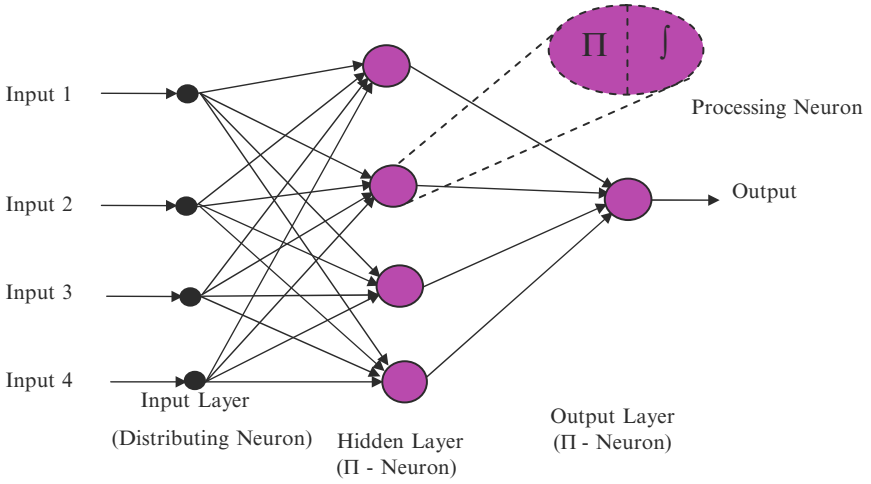


Fig. 5.3. Product type neural network (Π - ANN)

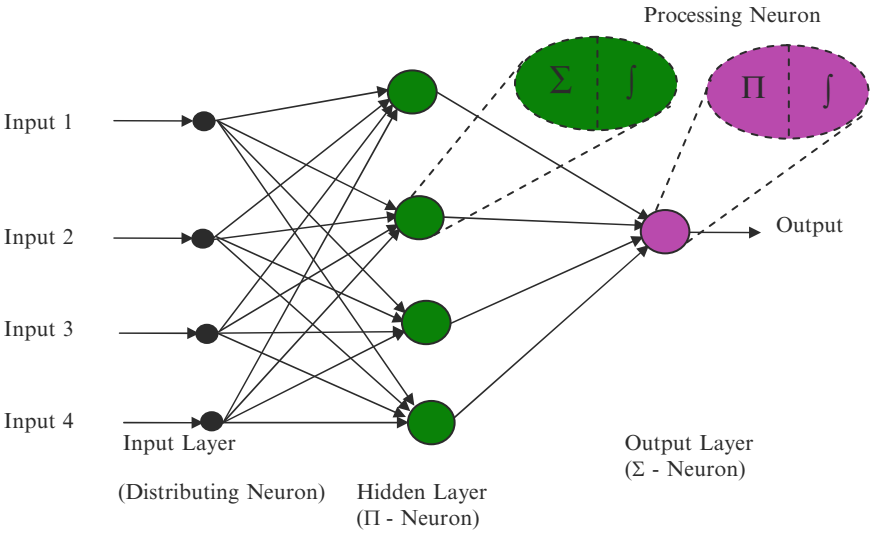


Fig. 5.4. Summation - Product type neural network ($\sum -\Pi$ - ANN)

2. Product – summation type neural network (Π-$\sum$ – ANN)
   This is a network in which Π – neurons are taken at the hidden layer and $\sum$ – neurons are at output layer as shown in Fig. 5.5.

Then all these four types of networks shown in Figs. 5.2–5.5 are used to model the non-linear starting speed – torque characteristic of induction motor and their performance have been compared for same initial weights and same ANN learning parameters as shown in Table 5.1.

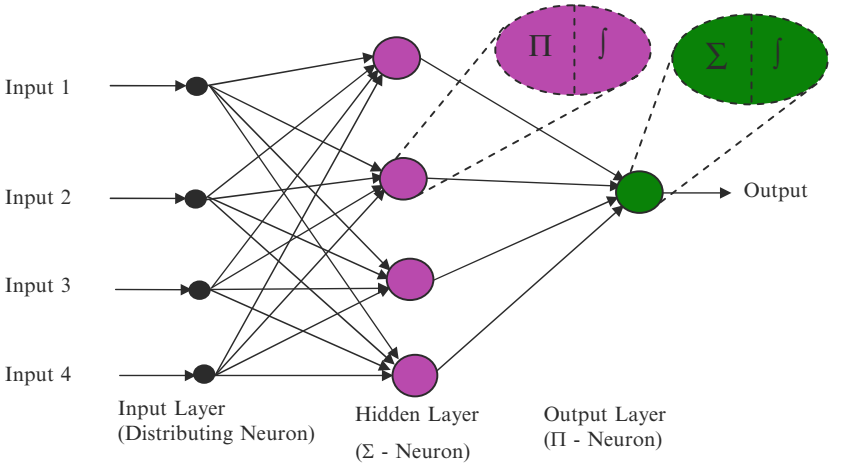


Fig. 5.5. Product-summation type neural network ($\Pi - \sum$ -ANN)

Table 5.1. ANN learning parameters

Learning rate	η	0.4
Momentum factor	α	0.6
Gain scale factor	λ	1.0
Error tolerance	E	0.005

Table 5.2. Performance of different ANN models for mapping induction motor characteristics

Models	Training performance	Testing performance		
		RMS error	Min error	Max error
$\sum$ - **ANN**	1150	0.02568	0.000376	0.888082
Π – **ANN**	4090	0.14097	0.000632	0.287738
$\sum -\Pi$ – **ANN**	50	0.01661	0.000430	0.250279
$\Pi - \sum$ - **ANN**	50	0.01358	0.000614	0.297254

The training and testing performance of all four type neural network is given in Table 5.2. It is quite clear that the combination of these different types of neuron layers in the network gave very interesting results. The mixed type neural network only needed 50 iterations (epochs) during training and testing results are also quite good for these type of networks.

5.2 Development of a Generalized Neuron (GN) Model

It is very clear from the above discussion that the combinations of summation ($\sum$) neurons and product (Π) neurons at different layers are giving quite good results as compared to only summation neuron or product neuron in

the whole network; which motivated to explore the possibilities of different combinations. Thus, a generalized neuron model has been developed that uses the fuzzy compensatory operators (listed in Table 5.3) that are partly union and partly intersection given by Mizumoto in his paper on pictorial representation of fuzzy connectives II in 1989.

Use of the sigmoid threshold function and ordinary summation or product as aggregation functions in the existing models fails to cope with the

Table 5.3. Compensatory operators suggested by Mizumoto (1989)

S. No.	Summation type operator	Product type operator
1.	$[X_1 \cap X_2]*W + [X_1 \cup X_2]*(1 - W)$	$[X_1 \cap X_2]^W*[X_1 \cup X_2]^{(1-W)}$
2.	$(X_1*X_2)*W + (X_1 + X_2 - X_1*X_2)*(1 - W)$	$(X_1*X_2)^W*(X_1 + X_2 - X_1*X_2)^{(1-W)}$
3.	$[0 \cup (X_1*X_2)]*W] + [1 \cap (X_1 + X_2)]*(1 - W)$	$[0 \cup (X_1*X_2)]^W + [1 \cap (X_1 + X_2)]^{(1-W)}$
4.	$[X_1 \cap X_2]*W + (X_1 + X_2 - X_1*X_2)*(1 - W)$	$[X_1 \cap X_2]^W*(X_1 + X_2 - X_1X_2)^{(1-W)}$
5.	$[X_1 \cup X_2]*W + (X_1*X_2)*(1 - W)$	$[X_1 \cup X_2]^W*(X_1*X_2)^{(1-W)}$
6.	$[X_1 \cap X_2]*W + [1 \cap (X_1 + X_2)]*(1 - W)$	$[X_1 \cap X_2]^W*[1 \cap (X_1 + X_2)]^{(1-W)}$
7.	$[X_1 \cup X_2]*W + [0 \cup (X_1 + X_2 - 1)]*(1 - W)$	$[X_1 \cup X_2]^W*[0 \cup (X_1 + X_2 - 1)]^{(1-W)}$
8.	$[X_1*X_2]*W + [1 \cap (X_1 + X_2)]*(1 - W)$	$[X_1*X_2]^W*[1 \cap (X_1 + X_2)]^{(1-W)}$
9.	$[X_1 + X_2 - X_1*X_2]*W + [0 \cup (X_1 + X_2 - 1)]*(1 - W)$	$[X_1 + X_2 - X_1*X_2]^W*[0 \cup (X_1 + X_2 - 1)]^{(1-W)}$
10.	$[X_1 \cap X_2]*W + [0 \cup (X_1 + X_2 - 1)]*(1 - W)$	$[X_1 \cap X_2]^W*[0 \cup (X_1 + X_2 - 1)]^{(1-W)}$
11.	$[X_1 \cup X_2]*W + [1 \cap (X_1 + X_2)]*(1 - W)$	$[X_1 \cup X_2]^W*[1 \cap (X_1 + X_2)]^{(1-W)}$
12.	$[X_1*X_2]*W + [0 \cup (X_1 + X_2 - 1)]*(1 - W)$	$[X_1*X_2]^W*[0 \cup (X_1 + X_2 - 1)]^{(1-W)}$
13.	$(X_1 + X_2 - X_1*X_2)*W + [1 \cap (X_1 + X_2)]*(1 - W)$	$(X_1 + X_2 - X_1*X_2)^W*[1 \cap (X_1 + X_2)]^{(1-W)}$
14.	$[X_1 \cap X_2]*W + [X_1*X_2]*(1 - W)$	$[X_1 \cap X_2]^W*[X_1*X_2]^{(1-W)}$
15.	$[X_1 \cup X_2]*W + (X_1 + X_2 - X_1*X_2)*(1 - W)$	$[X_1 \cup X_2]^W*(X_1 + X_2 - X_1X_2)^{(1-W)}$
16.	$[X_1 \cap X_2]*W + [(X_1 + X_2)/2]*(1 - W)$	$[X_1 \cap X_2]^W*[(X_1 + X_2)/2]^{(1-W)}$
17.	$[X_1 \cup X_2]*W + [(X_1 + X_2)/2]*(1 - W)$	$[X_1 \cup X_2]^W*[(X_1 + X_2)/2]^{(1-W)}$
18.	$[X_1*X_2]*W + [(X_1 + X_2)/2]*(1 - W)$	$[X_1*X_2]^W*[(X_1 + X_2)/2]^{(1-W)}$
19.	$[X_1 + X_2 - X_1*X_2]*W + [(X_1 + X_2)/2]*(1 - W)$	$[X_1 + X_2 - X_1*X_2]^W*[(X_1 + X_2)/2]^{(1-W)}$
20.	$[0 \cup (X_1 + X_2 - 1)]*W + [(X_1 + X_2)/2]*(1 - W)$	$[0 \cup (X_1 + X_2 - 1)]^W*[(X_1 + X_2)/2]^{(1-W)}$
21.	$[1 \cap (X_1 + X_2)]*W + [(X_1 + X_2)/2]*(1 - W)$	$[1 \cap (X_1 + X_2)]^W*[(X_1 + X_2)/2]^{(1-W)}$
22.	$[(X_1 \cap X_2)]*W + [(X_1 + X_2)/2]*(1 - W)$	$[(X_1 \cap X_2)]^W*[(X_1 + X_2)/2]^{(1-W)}$
23.	$[X_1 \cup X_2]*W + [1 - \sqrt{(1 - X_1)(1 - X_2)}]*(1 - W)$	$[X_1 \cup X_2]^W*[1 - \sqrt{(1 - X_1)(1 - X_2)}]^{(1-W)}$
24.	$[X_1 \cap X_2]*W + [1 - \sqrt{(1 - X_1)(1 - X_2)}]*(1 - W)$	$[X_1 \cap X_2]^W*[1 - \sqrt{(1 - X_1)(1 - X_2)}]^{(1-W)}$
25.	$[X_1 \cup X_2]*W + [\sqrt{(X_1*X_2)}]*(1 - W)$	$[X_1 \cup X_2]^W*[\sqrt{(X_1*X_2)}]^{(1-W)}$
26.	$\sqrt{(X_1*X_2)}*W + [1 - \sqrt{(1 - X_1)(1 - X_2)}]*(1 - W)$	$\sqrt{(X_1*X_2)}^W*[1 - \sqrt{(1 - X_1)(1 - X_2)}]^{(1-W)}$
27.	$[2X_1X_2/(X_1 + X_2)]*W + [(X_1 + X_2 - 2X_1X_2)/(2 - X_1 - X_2)]*(1 - W)$	$[2X_1X_2/(X_1 + X_2)]^W*[(X_1 + X_2 - 2X_1X_2)/(2 - X_1 - X_2)]^{(1-W)}$

Table 5.3. (*Continued*)

S. No.	Summation type operator	Product type operator
28.	$[(X_1 + X_2)/2]*W + \sqrt{(X_1X_2)}]*(1 - W)$	$[(X_1 + X_2)/2]^W * \sqrt{(X_1X_2)^{(1-W)}}$
29.	$[(X_1 + X_2)/2]*W + [1 - \sqrt{(1 - X_1)(1 - X_2)}]*(1 - W)$	$[(X_1 + X_2)/2]^W * [1 - \sqrt{(1 - X_1)(1 - X_2)}]^{(1-W)}$
30.	$[(X_1 + X_2)/2]*W + [2X_1X_2/(X_1 + X_2)]*(1 - W)$	$[(X_1 + X_2)/2]^W * [2X_1X_2/(X_1 + X_2)]^{(1-W)}$
31.	$[(X_1 + X_2)/2]*W + [(X_1 + X_2 - 2X_1X_2)/(2 - X_1 - X_2)]*(1 - W)$	$[(X_1 + X_2)/2]^W * [(X_1 + X_2 - 2X_1X_2)/(2 - X_1 - X_2)]^{(1-W)}$
32.	$[(X_1X_2)(X_1 + X_2 - 2X_1X_2)]*W + [X_1 + X_2 - X_1X_2(X_1 + X_2 - 2X_1X_2)]*(1 - W)$	$[(X_1X_2)(X_1 + X_2 - 2X_1X_2)]^W * [X_1 + X_2 - X_1X_2(X_1 + X_2 - 2X_1X_2)]^{(1-W)}$
33.	$[(X_1X_2)(X_1 \cap X_2)]*W + [X_1 + X_2 - X_1X_2 + X_1 \cup X_2 - (X_1 + X_2 - X_1X_2)(X_1 \cap X_2)]*(1 - W)$	$[(X_1X_2)(X_1 \cap X_2)^W * [X_1 + X_2 - X_1X_2 + X_1 \cup X_2 - (X_1 + X_2 - X_1X_2)(X_1 \cap X_2)]^{(1-W)}$
34.	$[(X_1X_2) + (X_1 \cap X_2) - (X_1X_2)(X_1 \cap X_2)]*W + [(X_1 + X_2 - X_1X_2)(X_1 \cup X_2)]*(1 - W)$	$[(X_1X_2) + (X_1 \cap X_2) - (X_1X_2)(X_1 \cap X_2)]^W * [(X_1 + X_2 - X_1X_2)(X_1 \cup X_2)]^{(1-W)}$

X_1 - Input # 1 for Σ – aggregation and
X_2 – Input # 2 for Π - aggregation
W - Weight or parameter of the operator varies between 0 and 1

Note: Output of Σ – part of neuron may be considered as union operator of fuzzy and Output of Π – part of neuron may be considered as intersection operator of fuzzy system

non-linearities involved in real life problems. To deal with these, the proposed model has both sigmoid and Gaussian functions with weight sharing. The generalized neuron model has flexibility at both the aggregation and threshold function level to cope with the non-linearity involved in the type of applications dealt with. The neuron has both Σ and π aggregation functions. The Σ aggregation function has been used with the sigmoid characteristic function while the π aggregation function has been used with the Gaussian function as a characteristic function. The final output of the neuron is a function of the two outputs O_Σ and O_π with the weights W and (1–W) respectively as shown in Figs. 5.6 and 5.7. Mathematically the output of summation type generalized neuron (GN) may be written as

$$\text{GN output} = O_{\sum} * W + O_{\Pi} * (1 - W),$$

where

$O_{\sum}$ – output of the summation part of the neuron Σ_1
W – weight associated with $O_{\sum}$
O_{Π} – output of the product part of the neuron (π).

The neuron model described above is known as the summation type compensatory neuron model, since the outputs of the sigmoidal and Gaussian functions are summed up. Similarly, the product type compensatory neuron models may also be developed. It is found that in most of the applications

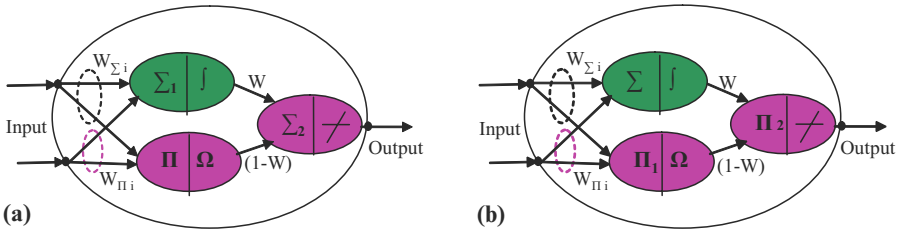


Fig. 5.6. (**a**) Internal structure of summation type. (**b**) Internal structure of product type generalized neuron

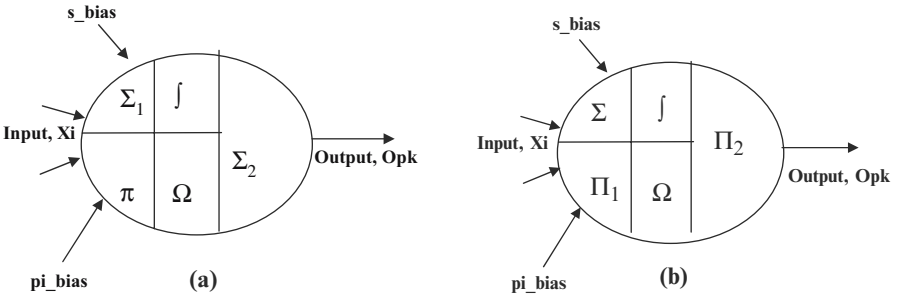


Fig. 5.7. (**a**) Symbolic representation of summation type generalized neuron model. (**b**) Symbolic representation of product type generalized neuron model

summation type compensatory neuron model works well (Chaturvedi 2002). Mathematically the output of product type generalized neuron may be written as –

$$\text{GN Output} = O_{\sum}{}^{W} * O_{\Pi}^{(1-W)}$$

5.3 Advantages of GN

1. Less number of unknown weights
   The number of weights in the case of a GN is equal to twice the number of inputs plus one, which is very low in comparison to a multi-layer feed-forward ANN.
2. Less training time
   The weights are determined through training. Hence, by reducing the number of unknown weights, training time can be reduced.
3. Less number of training patterns
   The number of training patterns required for GN training is dependent on the number of unknown weights. The number of training patterns must be greater or equal to number of GN weights. As mentioned above the number of GN weights are lesser than multi layered ANN, hence the number of training patterns required is also lesser.

4. Size of hidden layers
   There is no hidden layer required in case of GN and single neuron is capable to solve most of the problems.
5. Complexity of GN
   GN model is less complex as compared to multilayered ANN models.
6. Structural level flexibility
   GN models are more flexible at structural level. The aggregation and activations functions could be chosen depending on the problem in hand.

5.4 Learning Algorithm of a Summation Type Generalized Neuron

The following steps are involved in the training of a summation type generalized neuron:

1. *Foreward calculations*
   Step-1: The output of the Σ_1 part of the summation type generalized neuron is
   $$O_{\Sigma} = \frac{1}{1 + e^{-\lambda s * s\_net}} \tag{5.1}$$
   where $s\_net = \sum W_{\Sigma i} X_i + X_{o\Sigma}$.
   Step-2: The output of the π part of the summation type generalized neuron is
   $$O_{\Pi} = e^{-\lambda p * pi\_net^2} \tag{5.2}$$
   where $pi\_net = \prod W_{\Pi i} X_i * X_{o\Pi}$.
   Step-3: The output of the summation type generalized neuron can be written as
   $$O_{pk} = O_{\Pi} * (1 - W) + O_{\Sigma} * W \tag{5.3}$$
2. *Reverse calculation*
   Step-4: After calculating the output of the summation type generalized neuron in the forward pass, as in the feed-forward neural network, it is compared with the desired output to find the error. Using back-propagation algorithm the summation type GN is trained to minimize the error. In this step, the output of the single flexible summation type generalized neuron is compared with the desired output to get error for the ith set of inputs:
   $$\text{Error } Ei = (Yi - Oi) \tag{5.4}$$
   Then, the sum-squared error for convergence of all the patterns is
   $$Ep = 0.5 \sum Ei^2 \tag{5.5}$$
   A multiplication factor of 0.5 has been taken to simplify the calculations.
   Step-5: Reverse pass for modifying the connection strength.

(a) Weight associated with the Σ_1 and Σ_2 part of the summation type generalized neuron is:

$$W(k) = W(k-1) + \Delta W \tag{5.6}$$

where $\Delta W = \eta\delta_k(O_\Sigma - O_\Pi)Xi + \alpha W(k-1)$
and $\delta_k = \sum(Yi - Oi)$

(b) Weights associated with the inputs of the Σ_1 part of the summation type generalized neuron are:

$$W_{\Sigma i}(k) = W_{\Sigma i}(k-1) + \Delta W_{\Sigma i} \tag{5.7}$$

where $\Delta W_{\Sigma i} = \eta\delta_{\Sigma j}Xi + \alpha W_{\Sigma i}(k-1)$
and $\delta_{\Sigma j} = \sum \delta_k W(1 - O_\Sigma) * O_\Sigma$

(c) Weights associated with the input of the π- part of the summation type generalized neuron are:

$$W_{\Pi i}(k) = W_{\Pi i}(k-1) + \Delta W_{\Pi i} \tag{5.8}$$

where $\Delta W_{\Pi i} = \eta\delta_{\Pi j}Xi + \alpha W_{\Pi i}(k-1)$
and $\delta_{\Pi j} = \sum \delta_k(1 - W) * (-2 * pi\_net) * O_\Pi$
α – momentum factor for better convergence
η – learning rate

Range of these factors is from 0 to 1 and is determined by experience.

Matlab Program for Summation type GN model

```
% Main Programm for Summation type Generalized neuron (GN)
clear all;
clc;
tr_exor;     % training file name (tr_pat.m)
[i_row i_col]=size(x_tr);
patterns=i_row;
% Initialization of GN model
% weight Initialization
w=randn(in,on)*0.1;              % Weight of sum part (size [in x on])
wpi=ones(in,on)+randn(in,on)*0.1;   % Weight of product part
                                    (size [in x on])
w1=0.6;                          % weight of sum-sum part
pi_bais=0.05;                    % bais of product part
s_bais=0.05;                     % bais of sum part
delta_w1=0.0;                    % Change in weights for sum-sum part
delta_w=zeros(in,on)             % Change in weights for sum part
delta_wpi=zeros(in,o             % Change in weights for product part
delta_s_bais=0.0;                % Change in bais for sum part
delta_pi_bais=0.0;                    % Change in bais for sum part
ss_err=0;                             % sum quared error
```

```
% Network parameters
eta=input('Value of Learning rate='); %learning rate
alpha=input('Value of momentum factor='); % momentum factor
lemda_s=1;                          % gain scale factor of sigma part
lemda_pi=1;                         % gain scale factor of pi part
err_tol=0.001;                      % error tolrence
max_epoch=20000;                    % maximum number of iterations
disp_itr=max_epoch/100;             % Display training results after
                                      so many epochs
x_in_tr=x_tr(:,1:in);               % input pattern for training
y_desired=x_tr(:,(in+1):i_col);     % Desired output
count1=0;
for epoch=1:max_epoch               % Loop or cycle (both forward and
                                      reverse calculation)
for i=1:patterns                    % Loop for calculating output
                                      for GN model
s_net(i,:)=x_in_tr(i,:)*w+s_bais;  % sigma of (xi*wi)
x_wpi=x_in_tr(i,:).*wpi';           % product of (xi*wi)
pi_net(i,:)=pi_bais+prod(x_wpi);
end
s_out=1./(1+exp(-lemda_s*s_net));          % output of sigma part
pi_out=exp(-lemda_pi*(pi_net.^2));         % output of pi part


y_cnn=(w1*s_out+(1-w1)*pi_out);     % Final output

% Reverse calculation for adjusting weights and train GN model
% Network error
error=y_desired-y_cnn;              % Error of GN model
s_err=(error.^2)./2;                       % square error
ss_err1=ss_err;
ss_err=sum(s_err);                         % sum square error
err_dot=ss_err1-ss_err;             % change in sum square error
tr_res(epoch,:)=[epoch ss_err err_dot];    % training results

if ss_err<=err_tol; break; end
if count1==disp_itr
ss_err
count1=0;
end
count1=count1+1;

% weight adjustment of sigma-sigma part
delta_w1=eta*error'*(s_out-pi_out)+alpha*delta_w1;
w1=w1+delta_w1;

% weight adjustment of input-sigma part
f_desh=s_out.*(1-s_out);
delta_w=(lemda_s*eta*w1*(error.*f_desh)'*x_in_tr)'+alpha*delta_w;
w=w+delta_w;                        % New weights for sum part
```

```
% weight adjustment of input-pi part
fdesh_pi=-2*pi_out.*pi_net;
delta_wpi=lemda_pi*eta*(1-w1)*sum(error.*fdesh_pi.*(pi_net-pi_bais))./
          wpi+alpha*delta_wpi;
wpi=wpi+delta_wpi;                % New weights for product part

% modify bais of first sigma part
delta_s_bais=lemda_s*eta*w1*error'*f_desh+alpha*delta_s_bais;
s_bais=s_bais+delta_s_bais;

% modify bais of pi part
delta_pi_bais=lemda_pi*eta*(1-w1)*error'*fdesh_pi+alpha*delta_pi_bais;
pi_bais=pi_bais+delta_pi_bais;
end

% Testing of GN model
tst_exor;                         % Test File name (tst_pat.m)
[x_R x_c]=size(x_tst);
patterns1=x_R;

x_in1=x_tst(:,1:in);
y_desired1=x_tst(:,(in+1):i_col);

for i=1:patterns1
 s_net1(i,:)=x_in1(i,:)*w+s_bais;       % sigma of (xi*wi)
 x_wpi1=x_in1(i,:).*wpi';               % product of (xi*wi)
 pi_net1(i,:)=pi_bais+prod(x_wpi1);
end

s_out1=1./(1+exp(-lemda_s*s_net1));    % output of sigma part
pi_out1=exp(-lemda_pi*(pi_net1.^2));   % output of pi part

y_cnn1=(w1*s_out1+(1-w1)*pi_out1);     % GN output
err_tst=(y_desired1-y_cnn1);           % Error during testing

% plotting of training results
subplot(1,2,1);                        % Divide the display screen
                                         in 1 row and 2 columns
plot(tr_res(:,1),tr_res(:,2),'k-');    % plot (x,y)
xlabel('Number of Epochs') ;           % Label x-axis
ylabel('training ss_err');             % Label y-axis
title('Error');                        % Title for the graph
subplot(2,2,2)
plot(tr_res(:,1),tr_res(:,3),'k-');
axis([0 20000 -0.001 0.001]);
xlabel('Number of Epochs')
ylabel('Err_dot')
title('Derivative of Error')

% plotting of test results
subplot(1,2,2)
```

```
plot(y_desired1,'k--');hold on
plot(y_cnn1,'k-')
xlabel('Number of Output')
ylabel('Generalized Neuron output')
title('GN output during testing')
subplot(2,2,4)
plot(1:x_R, err_tst,'k-*')
xlabel('Number of Output')
ylabel('Error during testing')
title('testing Error')
```

5.5 Benchmark Testing of Generalized Neuron Model

The generalized neuron model developed must be verified on Benchmark problems and compared with feed-forward multi-layered ANN under same training conditions such as same gain scale factor, learning rate, momentum, initial weights and error function used in back-propagation learning algorithm.

5.5.1 Ex-OR Problem

The multi-layered feed-forward ANNs are trained to produce an output of one (zero) when binary input has an odd (even) number of bits. The Ex-OR problem is a classification problem, which is linearly non-separable. It requires minimum one hidden layer having two neurons for its solution. The input–output pattern of Ex-OR problem is given in Table 5.4.

It arises in the case of XOR problem, which may be viewed as a special case of points in the unit hypercube. Each point in the hypercube is class 0 or class 1. However, in the special case of the XOR problem, we need only the four corners of the unit square that corresponds to the input patterns (0,0), (0,1), (1,0) and (1,1). The first and third patterns are in class 0 and the input patterns (0,1) and (1,0) are also at the opposite corners of the square, but are classified together as output 1.

The use of a single neuron with two inputs results in a straight line for decision boundary in the input space. For all points on one side of the line, the neuron outputs 1; for all points on the other side of the line, it outputs 0. The position and orientation of the line in the input space are determined by the synaptic weights of the neuron connected to the input nodes, and the

Table 5.4. Input–output patterns for Ex-OR problem

Inputs		Output
0	0	0
0	1	1
1	0	1
1	1	0

threshold applied to the neuron. With the input patterns (0,0) and (1,1) located on opposite corners of the unit square and likewise for the other two input patterns (0,1) and (1,0), it is clear that we cannot construct a straight line for a decision boundary so that (0,0) and (1,1) lie in one decision region and (0,1) and (1,0) lie in the other decision region. In other words, an elementary perceptron cannot solve the XOR problem (Minsky and Papert 1969).

We may solve the XOR problem by using a single hidden layer with two neurons. The signal flow graph is shown in Fig. 5.8 and the decision boundaries formed in Fig. 5.9. The following assumptions are made here:

- Each neuron is represented by a McCulloch–Pitts model.
- Bits 0 and 1 are represented by 0 and +1.

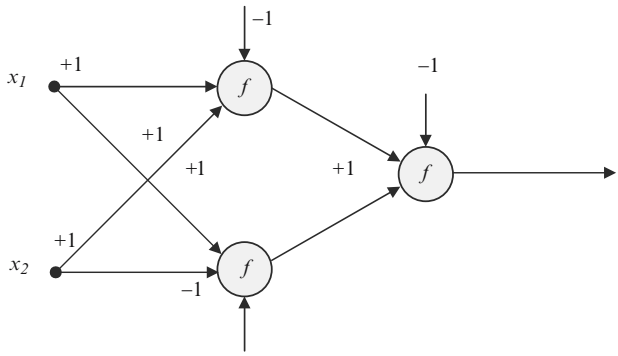


Fig. 5.8. XOR problem network

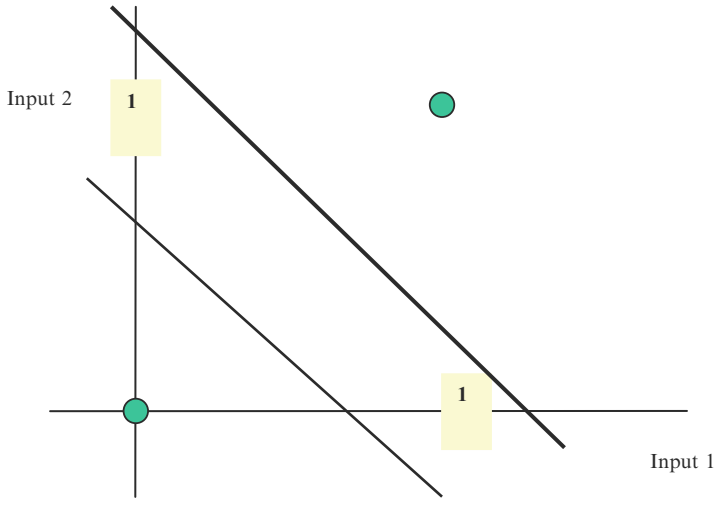


Fig. 5.9. Boundaries for Ex-OR problem

The top neuron, labelled 1 in the hidden layer is characterised as follows:

$$w_{11} = w_{12} = +1$$
$$\theta_1 = +1.5$$

The bottom neuron, labelled 2 in the hidden layer is characterised as follows:

$$w_{21} = w_{22} = +1$$
$$\theta_2 = +0.5$$

The output neuron, labelled 3 is characterised by:

$$w_{13} = -2$$
$$w_{23} = +1$$
$$\theta_3 = +0.5$$

ANN and GN model both have been trained using gradient descent back-propagation learning algorithm for 0.001 error tolerance with same training parameters. The reduction in error during training of ANN and GN Model is shown in Fig. 5.10 for Ex-or problem. Reduction in error during training is faster for GNM in comparison to ANN as given in Table 5.5.

The training and testing performance of GNM are shown in Figs. 5.11 and 5.12 for all four input patterns for Ex-or problem. The testing results in terms of RMS, max and min error of ANN and GNM during testing are presented in Table 5.6. It is found that during testing GN model is giving very good performance than ANN as the errors in output shown by them are considerably less than ANN.

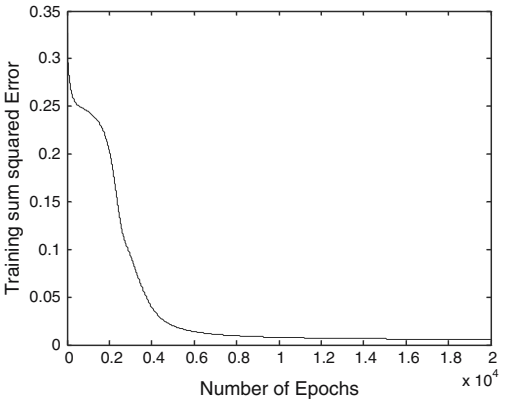


Fig. 5.10. Training performance of GN model for Ex-OR problem

Table 5.5. Training performance of ANN and GN Model (GNM) for EX-OR problem

Models	Structure	Training epochs
ANN	2-2-1	60,680
GN Model	Single neuron	20,435

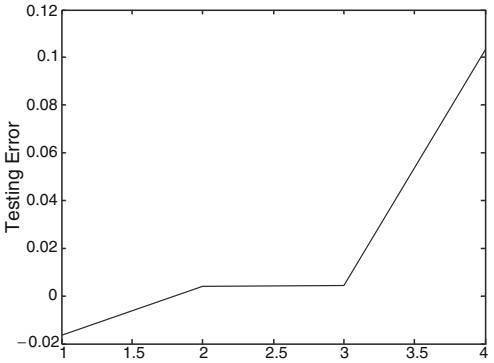


Fig. 5.11. Testing performance of GN model for Ex-OR problem

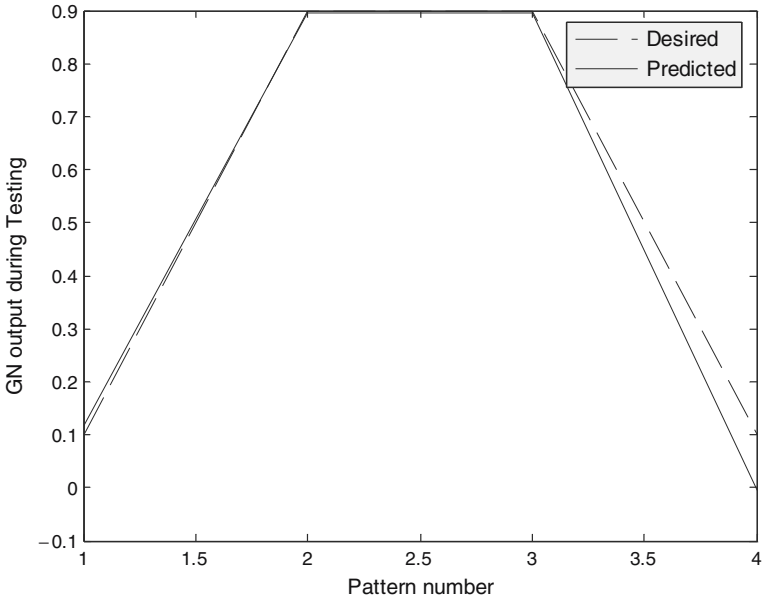


Fig. 5.12. Performance of generalized neuron output during testing

Table 5.6. Testing performance of ANN and GN model For Ex-OR problem with 5% noise

Models	RMS error	MAX error	MIN error
ANN	0.33673	0.21758	−0.11341
GN Model	0.11671	0.11201	−0.01832

```
% Training patterns for Ex-OR problem (train_pat.m)
% Normalized input output patterns
in=2; % number of inputs (X1 and X2)
on=1; % number of outputs (Y)
   %      X1      X2      Y
   x=[    0.9     0.9     0.1
          0.1     0.9     0.9
          0.9     0.1     0.9
          0.1     0.1     0.1];
% Normalized testing file (tst_pat.m)
   %      X1      X2      Y
   x=[    0.89    0.9     0.1
          0.1     0.9     0.9
          0.9     0.1     0.9
          0.12    0.1     0.1];
```

5.5.2 The Mackey-Glass Time Series

The Mackey-Glass (MG) time series is the most common problem to evaluate a network for its prediction capabilities. The MG series is a model of chaotic series. The Mackey-Glass equation represents a model for white blood cells production in leukaemia patients. It mimics the non-linear oscillations in the physiological processes involved. The Mackey–Glass delay difference equation is given below

$$x(t+1) = 0.9x(t) + [0.2 \times (t-\tau)/(1+x^{10}(t-\tau))].$$

The function plot is shown in Fig. 5.13.

The model is complicated due to the addition of a time delay τ in the non-linear equations. The objective of this analysis is to evaluate the efficiency of networks to predict future values using a set of past values. The above M-G equation is implemented with $\tau = 1.7$, $x(0) = 1.2$, $x(t) = 0$ for $t < 0$. A total of 301 points have been generated from $t = 0$ to $t = 300$, all points have been used for training. The 0th, 6th, 12th, and 18th points have been used to predict 19th point and so on.

The training results of ANN and GNM are shown in Table 5.7 and Fig. 5.14 for Mackey–Glass problem for error level 0.002. Reduction in error during training is faster for GN Model in comparison to ANN. GN model is consistently giving good results in training of this time series problem.

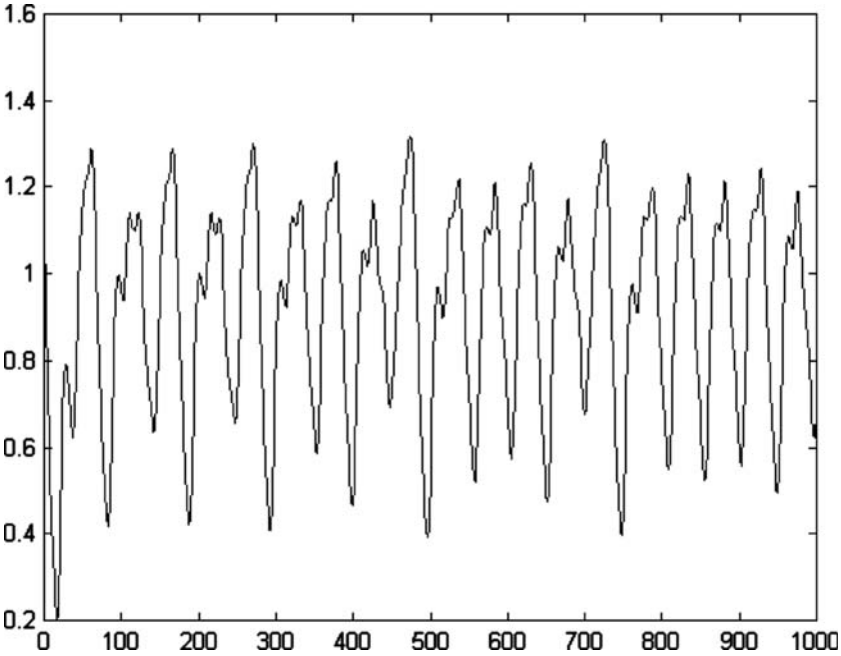


Fig. 5.13. The Mackey-Glass time series

Table 5.7. Training performance of ANN and GN model for Mackey–Glass problem

Models	Structure	Training epochs
ANN	4-4-1	13,340
GN model	Single neuron	20,083

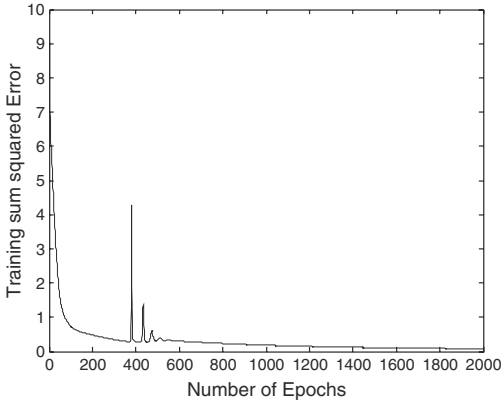


Fig. 5.14. Error during training

Once ANN and GNM are carefully trained for Mackey–Glass Problem, it has been used for testing. The training and testing performance of ANN and GNM are shown in Figs. 5.15 and 5.16. The test output of GN model is nearly coinciding with the actual data for all test patterns. The results in terms of RMS, max and min errors of ANN and GN Model during testing are presented in Table 5.8.

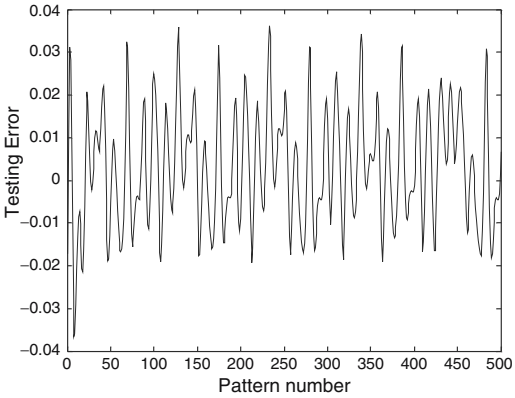


Fig. 5.15. Error during testing

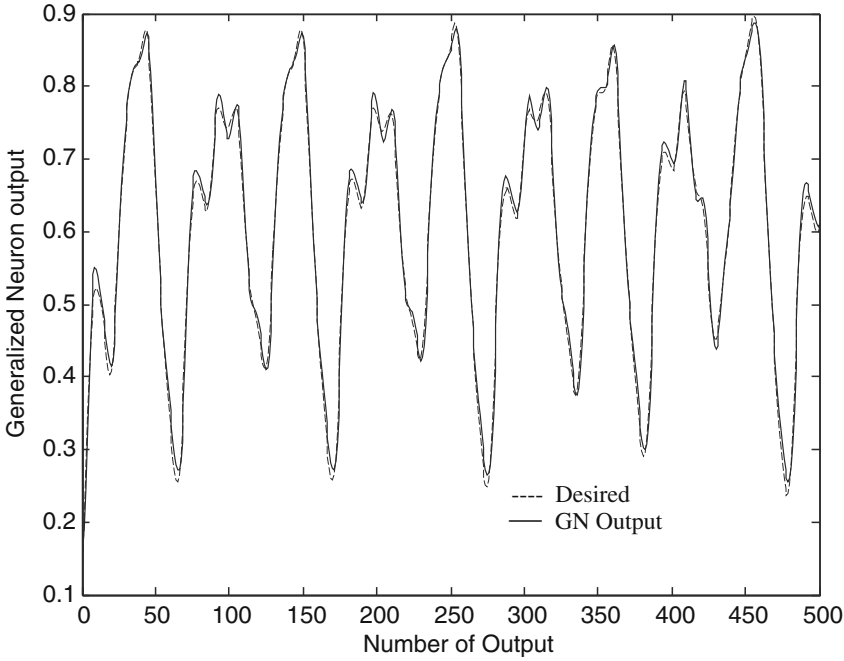


Fig. 5.16. Test results of Mackey–Glass problem during testing

Table 5.8. Testing performance of ANN and GN model with MACKEY-GLASS problem

Models	Errors (after 200 training epochs)		
	RMS error	MAX error	MIN error
ANN	0.17301	0.32675	−0.20943
GN model	0.02101	0.03502	−0.03730

5.5.3 Character Recognition Problem

The GN model is used to distinguish five different characters, A, X, H, B, I. Each character is represented by 5×7 dots. Hence, there are 35 inputs for each character as shown below:

```
input =[ 0 0 1 0 0   0 1 0 1 0   1 0 0 0 1   1 0 0 0 1   1 1 1 1 1   1 0 0 0 1   1 0 0 0 1
         1 0 0 0 1   0 1 0 1 0   0 0 1 0 0   0 0 1 0 0   0 0 1 0 0   0 1 0 1 0   1 0 0 0 1
         1 0 0 0 1   1 0 0 0 1   1 0 0 0 1   1 1 1 1 1   1 0 0 0 1   1 0 0 0 1   1 0 0 0 1
         1 1 1 1 1   1 0 0 0 1   1 0 0 0 1   1 1 1 1 1   1 0 0 0 1   1 0 0 0 1   1 1 1 1 1
         0 0 1 0 0   0 0 1 0 0   0 0 1 0 0   0 0 1 0 0   0 0 1 0 0   0 0 1 0 0   0 0 1 0 0];
```

The output of GN model is these characters. We can assign some values to these characters, because Neural system can give only numerical outputs.

A -	0.1
X -	0.3
H -	0.5
B -	0.7
I -	0.9

Let's train the GN model to recognize these characters. The training file consists of the set of inputs–ouput data. Here the input has value 0 or 1. For 0 input product part of GN model does not work, therefore it is necessary to normalize the data in 0.1–0.9 range and present them to GN model for training. GN model uses the following parameters for training:

Learning rate = 0.1, momentum factor = 0.5, error tolerance = 0.0001 and maximum epochs = 1,000. The following results are obtained – given error tolerance level is achieved in 210 epochs (cycles) and weights of size w1(1×1), w(1×35), wpi(1×35) are as follows:

```
w1 = [1.1256];
 w = [0.0024  − 0.1337 0.1996  − 0.0314 0.0425 0.1887  − 0.2568 0.3413
      −0.5350 0.1317  − 0.1299  − 0.0076 ...
      0.1993  − 0.0968 0.0157  − 0.0183 0.0938 0.4312 0.2867  − 0.1531
      −0.0449  − 0.2633 0.1360  − 0.1022 ...
      −0.0940  − 0.0982  − 0.1757 0.4440  − 0.1252  − 0.0573  − 0.2875
      0.1390 0.7092 0.1367  − 0.3039];
```

wpi = [1.0247 0.8564 1.0149 0.8307 1.0719 1.1142 1.1552 1.1384 0.9242
1.0443 1.0911 0.8926 ...
1.0202 1.0763 0.8712 0.9047 1.0778 0.9994 1.0524 1.1364 1.0482 0.9213
1.0752 0.9833 ...
0.9184 1.2094 1.0080 0.9063 1.0636 1.1682 1.0594 1.0790 1.0105 0.9841
1.0871];

The training and testing performance is graphically shown in Figs. 5.17 and 5.18.

For the same given inputs GN output is [0.1005 0.3021 0.4968 0.7040 0.8870].

Now, one bit is changed in every character and then input data is prepared for testing of GN performance. The GN output for

these set of testing data is [0.0869 0.3221 0.4999 0.7040 0.8737];
and expected output vector is [0.1 0.3 0.5 0.7 0.9].

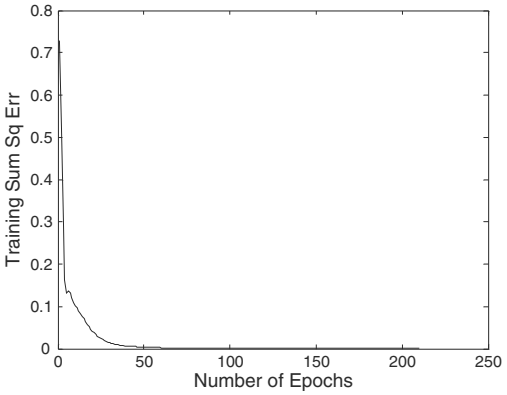


Fig. 5.17. Training performance of GN model for character recognition problem

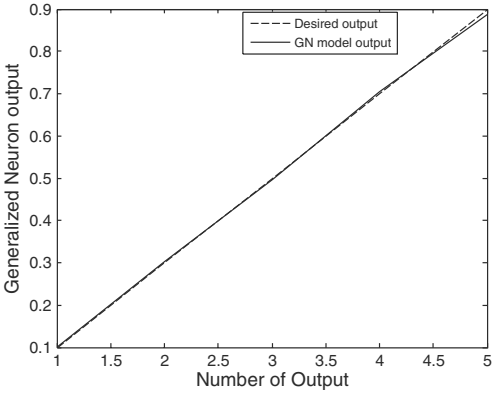


Fig. 5.18. Testing performance of GN model for character recognition problem

Table 5.9. The performance comparison of GN model and ANN model

	ANN	GNN
Network size	35-5-1 (three layer network)	35-1 (Single neuron)
Training cycles	1000	210
Error achieved in training	0.008223	0.0001
Change one bit in each character	0.0120	0.0028

It shows that the GN performance does not detoriate too much if some noise is present in the testing data set. It means the training patterns are learned well. What happens if, we present foreign characters to the GN model? Let us consider the letter M and J, as follows:

```
Testing = [1 0 0 0 1  1 1 0 1 1  1 0 1 0 1  1 0 0 0 1  1 0 0 0 1  1 0 0 0 1  1 0 0 0 1
pattern   0 0 1 0 0  0 0 1 0 0  0 0 1 0 0  0 0 1 0 0  0 0 1 0 0  0 0 1 0 0  0 1 1 1 0];
```

The results should show each foreign character in the category closest to it. The results obtained from the model is [0.1977 0.8741]. In the first pattern, M is categorized as A and J is categorized as I as expected.

The performance of GN model is also compared with ANN model and results are given in Table 5.9.

5.5.4 Sin(X1) ∗ Sin(X2) Problem

This is a functional mapping problem to test the capabilities of GN model for various types of functions. The mapping of two functions sin(x1) and sin(x2) on to their product is used here. This is a popular functional mapping problem Y = sin(x1)* sin(x2) as shown in Fig. 5.19. The training data for sin(x1)* sin(x2) problem of the GNM and ANN is given in Table 5.10.

The ANN and GN model have been trained for Sin(x1) ∗ Sin(x2) problem and training performance of both types of architectures is compared and given in Table 5.9 and Fig. 5.20.

The testing performance of ANN and GN Model are shown in Fig. 5.21 with 5% noise in the testing data. The test output given by GN Model nearly coincides with the actual data for all test patterns; however the test output given by ANN is far away to coincide with actual data. The results in terms of RMS, max and min errors of ANN and GN Model during testing are presented in Table 5.11. It is found that during testing GN Model gives very good performance than ANN as the errors in output shown by it is considerably less than ANN.

5.5.5 Coding Problem

In this problem the network is presented with n distinct binary input patterns, each with different bit positions and the network is trained to produce a

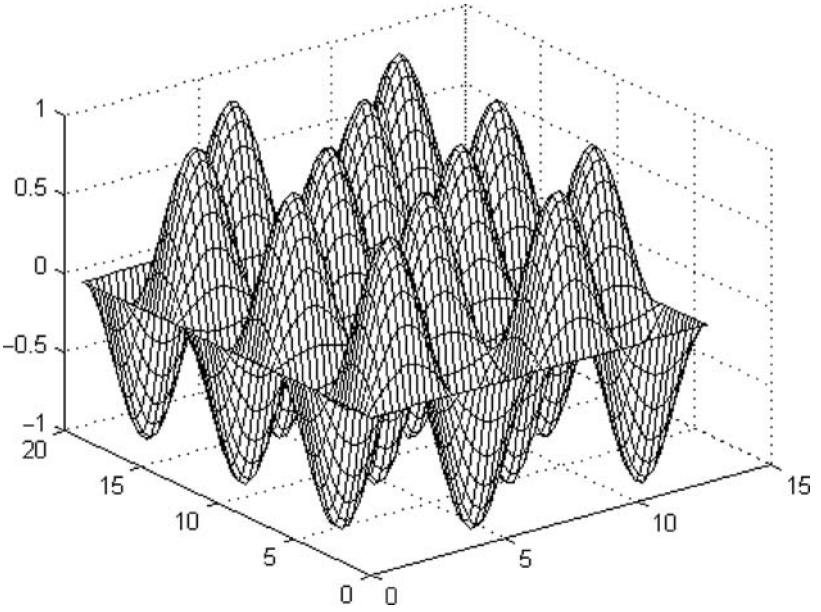


Fig. 5.19. The Sin(x1) Sin(x2) problem

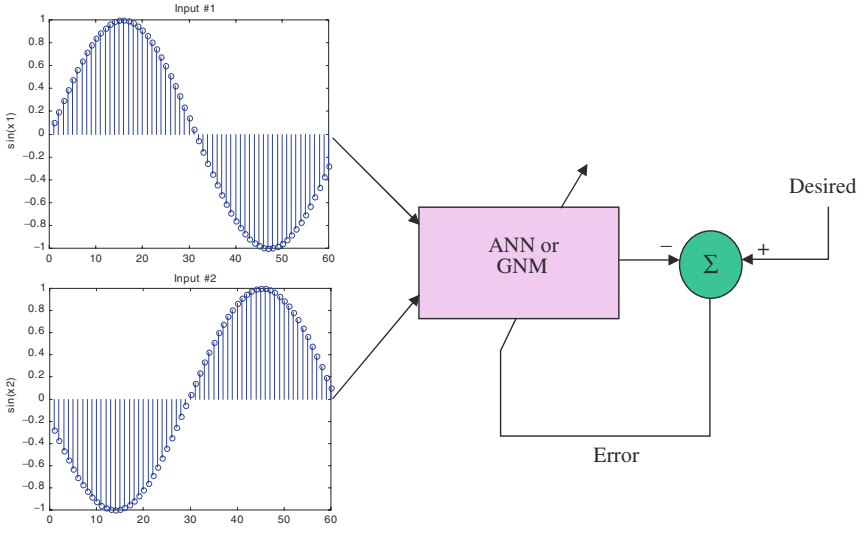


Table 5.10. Training performance of ANN AND GN model for SIN (x1) * SIN (x2) problem

Models	Structure	Training epochs
ANN	4-4-4-1	50,870
GN model	Single neuron	764

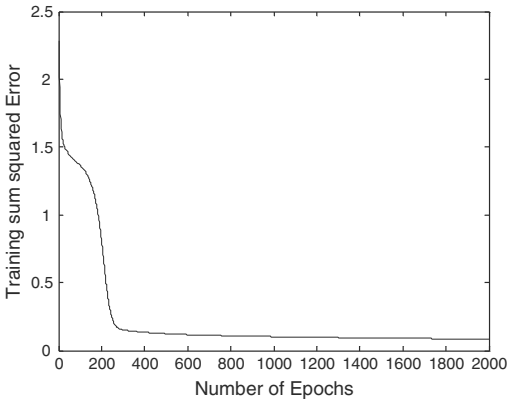


Fig. 5.20. Training results of GN model

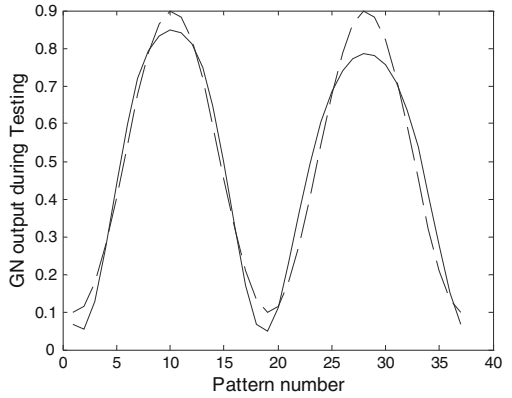


Fig. 5.21. Testing results of GN model

Table 5.11. Testing performance of ANN and GNM SIN (X1) * SIN (X2) problem

Models	Errors (After 10,000 training epochs)		
	RMS error	MAX error	MIN error
ANN	0.36632	0.71788	−0.85541
GN model	0.04011	0.04442	−0.06601

particular output value corresponding to each set of bit. The training set for coding problem is given below in Table 5.12. The training patterns are normalized in the range 0.1–0.9 and then presented to ANN and GN model for training. Once the model is trained then it can be used for testing. The training and testing results are shown in Figs. 5.22–5.23 and Tables 5.13 and 5.14.

Table 5.12. Training patterns for coding problem

Input pattern	Output
1 1 1 1	1.5
1 1 1 0	1.4
1 1 0 1	1.3
1 1 0 0	1.2
1 0 1 1	1.1
1 0 1 0	1.0
1 0 0 1	0.9
1 0 0 0	0.8
0 1 1 1	0.7
0 1 1 0	0.6
0 1 0 1	0.5
0 1 0 0	0.4
0 0 1 1	0.3
0 0 1 0	0.2
0 0 0 1	0.1

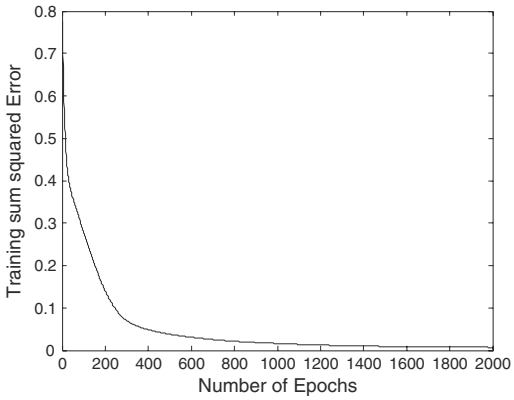


Fig. 5.22. Performance of GN model during training

For the GN model following points are important to discuss:

1. The GN input must be normalized in the appropriate range. Most of the time 0.1–0.9 normalization range works very well. If the input is slightly outside to the normalization range then there is a margin of 0.1 on both sides, so the GN model could give appropriate results. If normalization range is between 0 and 1, then for 0 inputs the output of product part of GN model is zero. Hence this normalization range in not suitable for GN model.
2. Learning rate and momentum factor should be decided in such a way that GN model learns faster and give stable response. One can start with very low value of learning rate and higher value of momentum factor. The

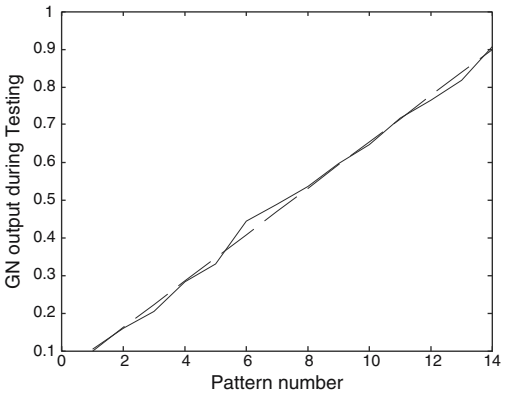


Fig. 5.23. Performance of GN model during testing

Table 5.13. Training performance of ANN and GN model for coding problem

Models	Structure	Epochs
ANN	4-4-1	38,430
GN model	Single neuron	689

Table 5.14. Testing performance of ANN and GN model for coding problem

Model	RMS error	MAX error	MIN error
ANN	0.36543	0.79301	−0.55296
GN model	0.00079	0.0018	−0.00111

typical values are – Learning rate = 0.001 and momentum factor = 0.1 to start with GN model. Then it could be increased.

3. For better generalization capability of GN model little amount of noise may be included in the training data (e.g. 0–5%).

D. 3-D Surfaces for different types of neurons

A simple matlab program is written to draw the 3-D error surfaces for single input single output system for all four type of neurons (i.e. Σ – neuron, Π – neuron, summation type generalized neuron and product type generalized neuron) as shown in Fig. 5.24–5.27. It is seen that for the simple Σ – neuron the error surface is just a inclined plane. Hence, it can never map the non-linear function. On the other hand, in Π – neuron the 3-D surface is a curved one and it can handle non-linear problems, but the error surface suddenly changes. In case for summation or product type generalized neuron the error surfaces are curved and learn quickly.

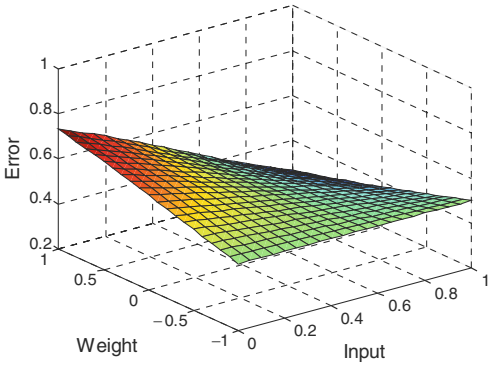


Fig. 5.24. 3-D surface for conventional Σ – neuron

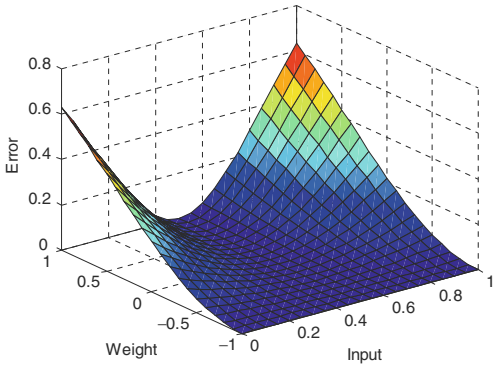


Fig. 5.25. 3-D surface for conventional Π – neuron

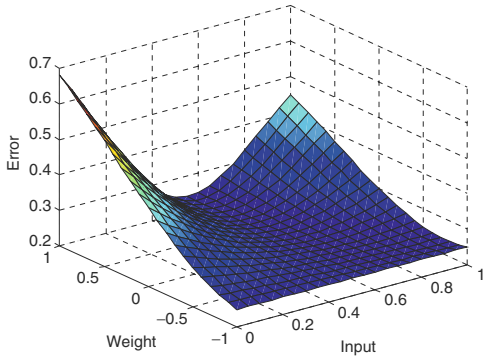


Fig. 5.26. 3-D surface for summation type GN model

% Matlab program for surface generation for different types of Neuron Models

```
clear all;
```

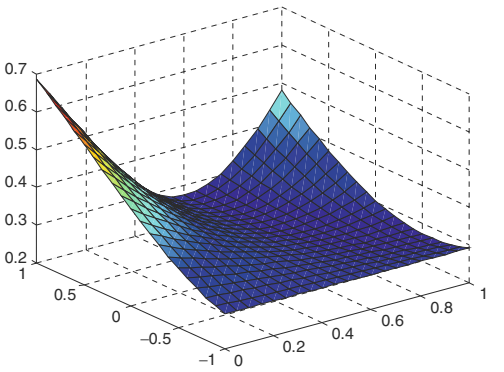


Fig. 5.27. 3-D surface for product type GN model

```
w1=[-1:.1:1];                    % Weight varaition in the range -1 to+1.
x=[0:.05:1];                     % Input variation from 0 to 1
 for i=1:21                       % Loop
 net_s=x(i)*w1;  % Calculating weighted sum for the neuron output
 net_pi=x(i)*w1;                  % Calculating the product of input
                                    and weight
 sumout=1./(1+exp(-net_s));       % Output of summation part
 piout=exp(-net_pi.^2);           % Output of product part
 ysum(i,:)=sumout;                % Output of summation neuron
 ypi(i,:)=piout;                  % output of product neuron
 ySGNM(i,:)=((sumout+piout)./2);      % output of summation type
                                       generalized neuron
 yPGNM(i,:)=(sqrt(sumout.*piout));    % output of product type
                                       generalized neuron
end
figure(1)
surf(x,w1,1-ysum);                   % Ploting the 3-D surface
xlabel('Input'); ylabel('Weight'); zlabel('Error');
title('Standard Summation Neuron');
figure(2)
surf(x,w1,1-ypi); xlabel('Input');ylabel('Weight'); zlabel('Error');
title('Standard Product Neuron');
figure(3)
surf(x,w1,1-ySGNM); xlabel('Input');ylabel('Weight'); zlabel('Error');
title('Summation type Generalized Neuron');
figure(4)
surf(x,w1,1-yPGNM);
surf(x,w1,1-ySGNM); xlabel('Input');ylabel('Weight'); zlabel('Error');
title('Product type Generalized Neuron');
```

5.6 Generalization of GN model

There are many GN models proposed based on the flexibility at both the aggregation and activation function level to cope with the non-linearity involved in the type of applications dealt with. The neuron can use "n" number of aggregation and "m" number of activation functions. The final output of the neuron is a function of output of all activation functions as shown in Fig. 5.28–5.29.

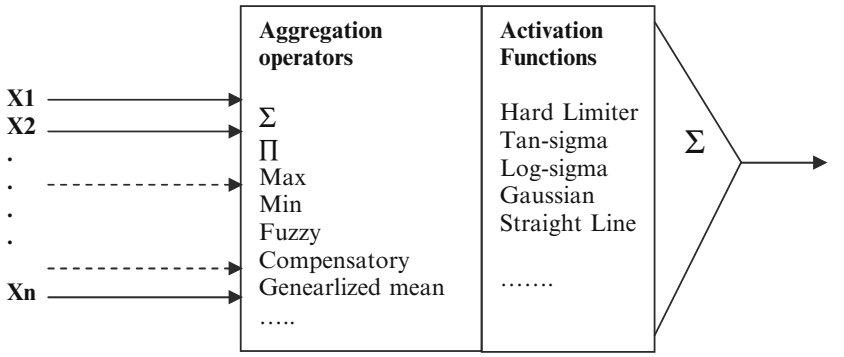


Fig. 5.28. Generalization of GN model

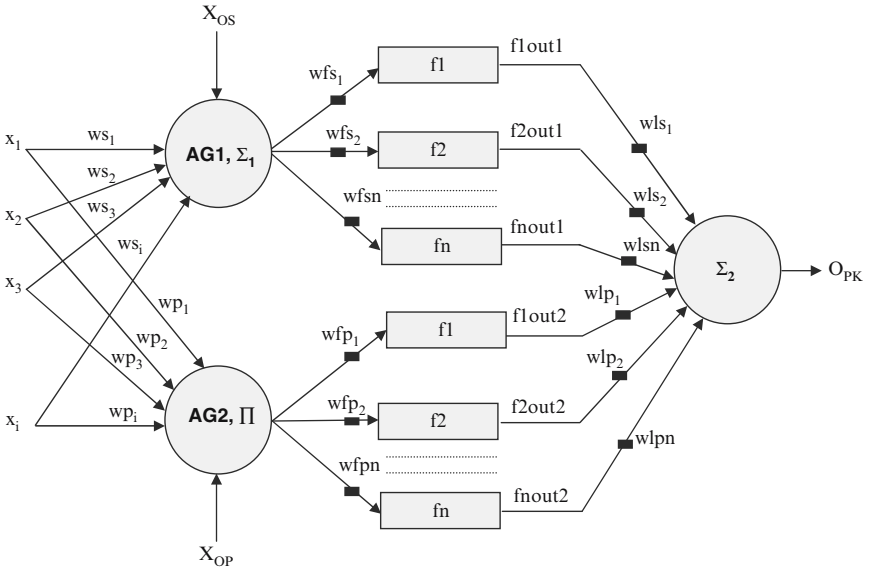


Fig. 5.29. Detailed diagram of generalization of GN model

5.6.1 GN Model-1

In this model of generalized neuron two aggregation functions (Σ and Π) and two aggregation functions (Sigmoidal and Gaussian) have been considered. Finally, the outputs are summed up to get the neuron output. The output of new neuron can be mathematically written as:

$$Opk = f1out1 * W1s1 + f1out2 * W1p1 + f2out1 * W1s2 + f2out2 * W1p2 \tag{2.22}$$

where,

$$W1p1 = (1 - W1s1),$$
$$W1p2 = (1 - W1s2)$$

Outputs of sigmoid activation functions are

$$f1out1 = \frac{1}{1 + e^{(-sumsigma^* Wfs1)}}$$
$$f1out2 = \frac{1}{1 + e^{(-product^* Wfp1)}}$$

Outputs of Gaussian activation functions are

$$f2out1 = e^{-(sumsigma^* Wfs2)^2}$$
$$f2out2 = e^{-(product^* Wfp2)^2}$$

where

$$Wfs2 = (1 - Wfs1), \;\; Wfp2 = (1 - Wfp1)$$

5.6.2 GN Model-2

In GN model-2 three activation functions sigmoid, Gaussian and straight line have been tried with two aggregation functions "Σ" and "Π". The outputs of functions used in this case of the model are given below.

Outputs of activation function for Σ part are f1out1, f2out1 are same as in Case-1.

$$f3out1 = K^* sumsigma, \quad \text{"straight line function"}$$

Output of activation functions for Π part are f1out2, f2out2 are same as in Case-1.

$$f3out2 = K^* product \quad \text{'straight line function'}$$
$$\text{K} = \text{slope of straight line}$$

Output of the neuron is

$$Opk = f1out1 * W1s1 + f1out2 * W1p1 + f2out1 * W1s2 + f2out2 * W1p2 + f3out1 * W1s3 + f3out2 * W1p3 \tag{2.23}$$

5.6.3 GN Model-3

In this case of new GN model-3 also three activation functions (sigmoid, Gaussian and straight line) have been tried with two aggregation functions Σ and Π. Three output weights (wl1, wl2, wl3) are independent for activation functions, the other three dependent output weights are taken as given below:

$$\text{wlp1} = 1 - \text{wls1}, \quad \text{wlp2} = 1 - \text{wls2}, \quad \text{wlp3} = 1 - \text{wls3}.$$

Output of the neuron is

$$\begin{aligned} Opk = {} & f1out1 * W1s1 + f1out2 * (1 - W1s1) + f2out1 * W1s2 \\ & + f2out2 * (1 - W1s2) + f3out1 * W1s3 + f3out2 * (1 - W1s3) \end{aligned} \tag{2.24}$$

5.6.4 GN Model-4

In this case of new GN model-4 four activation functions namely sigmoid, Gaussian, straight line and sinusoidal have been tried with two aggregation functions Σ and Π. Output of the neuron is

$$\begin{aligned} Opk = {} & f1out1 * W1s1 + f1out2 * Wlp1 + f2out1 * W1s2 + f2out2 * Wlp2 \\ & + f3out1 * W1s3 + f3out2 * Wlp3 + f4out1 * W1s4 + f4out2 * Wlp4 \end{aligned} \tag{2.25}$$

where $W1p1 = (1 - W1s1)$, $W1p2 = (1 - W1s2)$, $W1p3 = (1 - W1s3)$, $W1p4 = (1 - W1s4)$

The above mentioned GN models have been tested for different benchmarks problems and compared with ANN with the parameters shown in Table 5.15. The ANN (4-4-4-1) and GNM both have been trained using gradient descent back-propagation learning algorithm for 0.002 error tolerance with same training parameters. The results are as given in Tables 5.16 and 5.17.

3-D surfaces for GN models 1–4 are given in Figs. 5.30–5.33.

Table 5.15. Neural network parameters for ANN and GNM

Learning rate	–	0.0001
Momentum	–	0.9
Gain scale factor	–	1.0
Tolerance	–	0.002
All initial weights	–	0.95

Table 5.16. Training epochs of ANN AND GN models when 5% noise is included in data

Problems	ANN	GNM-1	GNM-2	GNM-3	GNM-4
Ex-OR	150,680	50,435	3,452	7,967	3,470
4-bit parity	255,430	120,423	9,998	26,335	10,122
Mackey-Glass time series	30,340	183	131	150	162
Character recognition	70,945	3,037	2,358	268	241
Sin(x1) Sin(x2)	50,870	764	242	315	229
Coding problem	38,−30	689	7,478	421	223

Table 5.17. Testing performance (rms error) of ANN and GNM For benchmark problem with 5% noise in testing data

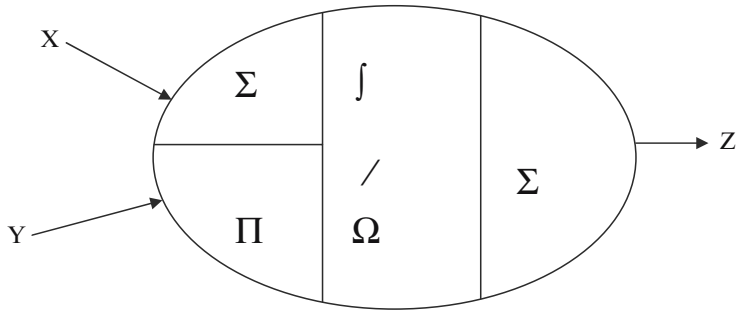
Problems	ANN	GNM-1	GNM-2	GNM-3	GNM-4
Ex-OR	0.63673	0.48671	0.00261	0.00627	0.00292
4-bit parity	0.46543	0.12257	3.02×10^{-10}	1.18×10^{-8}	1.09×10^{-9}
Mackey-Glass time series	0.27301	0.24426	0.02058	0.02654	0.02426
Character recognition	0.39375	0.13432	0.15084	0.03376	0.03201
Sin(x1) Sin(x2)	0.36632	0.04011	0.01841	0.03941	0.00613
Coding problem	0.36543	0.00079	3.74×10^{-6}	9.91×10^{-2}	2.21×10^{-12}

5.7 Discussion on Benchmark Testing

Training performances of ANN and GN Models during training on various benchmark problems are discussed in this chapter. Convergence in GN model is much faster as compared to ANN. For Ex-OR, 4-bit Parity and Mackey–Glass problems GN Model-2 requires only 3,452, 9,998 and 131 epochs, however, ANN requires 150,680, 255,430 and 30,340 epochs, respectively, to achieve same tolerance (0.002) in output. In character recognition, sin (x1)$*$ sin (x2) and coding problems GN Model - 4 requires 241, 229 and 223 epochs only, which are much less as compared to epochs 70,945, 50,870 and 38,430 required by ANN to achieve same tolerance level in output.

This shows that GN models have better training performance than feed-forward commonly used ANN. It is also observed that the performance of GN model not only depends on type of problem but also depends on type and number of activation and aggregation functions used. Apart from this, the above model gives good performance over 5% noise in testing input data as shown earlier.

In the GN model structural complexity is very less as compared to ANN. Comparison of structural complexity associated with ANN and GN Model is represented in Table 5.18. Four layered ANN with 13 neurons and 52 interconnections with 13 number of biases uses for modeling of benchmark problems, however GN Model - 4 uses only 1 neuron with 24 number of interconnections and 2 biases. Further, ANN uses 13 activation functions for all its neurons;



(a) GNM Case – 2

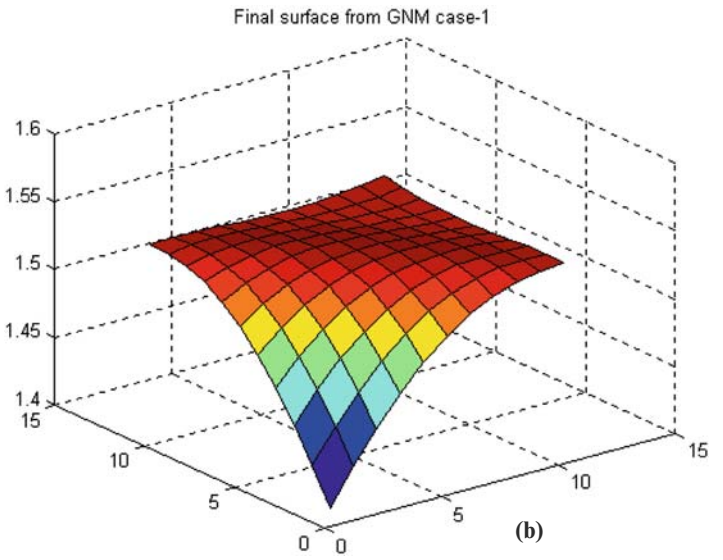


Fig. 5.30. (**a**) GNM Case-2. (**b**) Final output surface obtained from GNM-1

however, GN model -4 uses maximum eight activation functions. This shows that ANN requires more complex structure as compared to GN model to model a problem.

The computational complexity of ANN and GN models are also represented in Table 5.19. In one stroke, ANN requires 91 total number of operations, however GN Model requires only 31 operations as in the GNM case-4. It means structural complexity as well as computational complexity involved both are reduced in GN model as compared to ANN. Further, the computation time required on PC – Pentium III in one stroke is also less, i.e. 125.5 ms in case of GN model; however it is 808 ms for ANN.

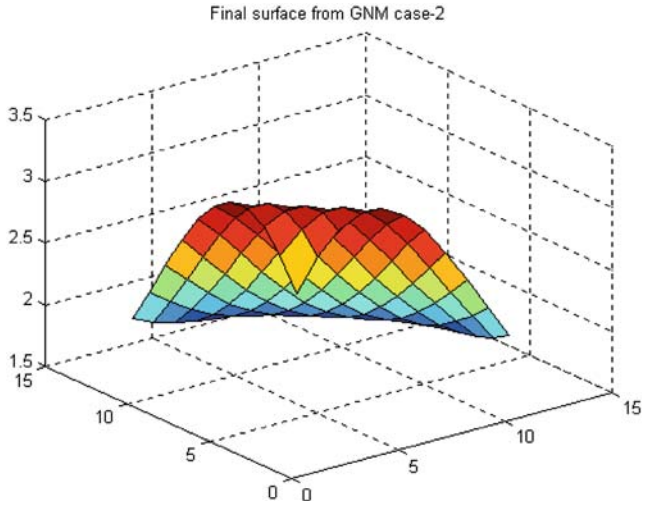
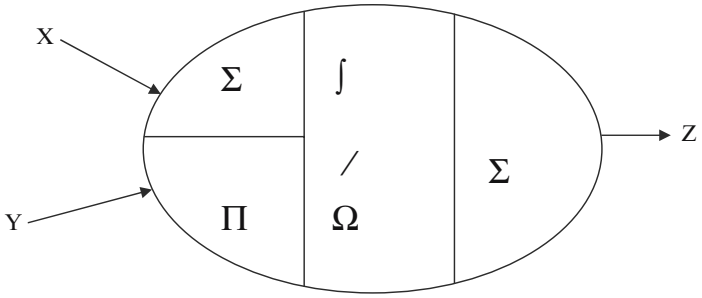


Fig. 5.31. Final output surface obtained from GNM-2



(a) GNM Case – 3

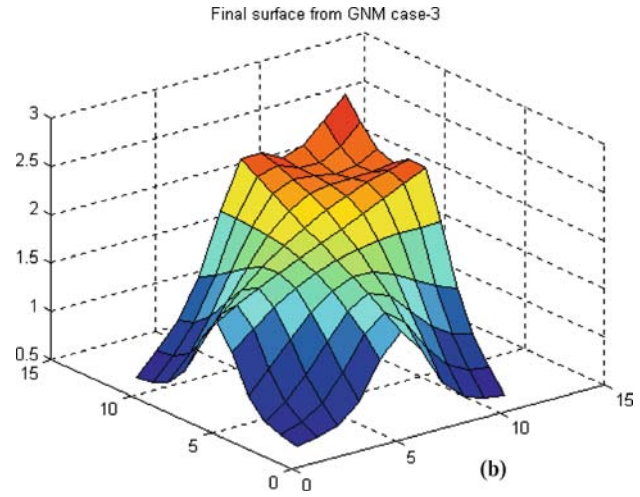
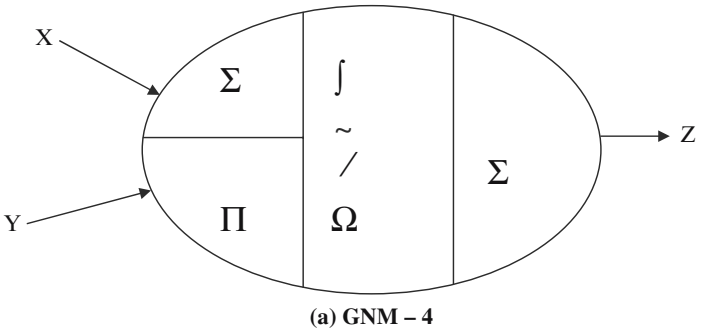


Fig. 5.32. (**a**) GNM Case-3 (**b**) Final output surface obtained from GNM-3



(a) GNM – 4

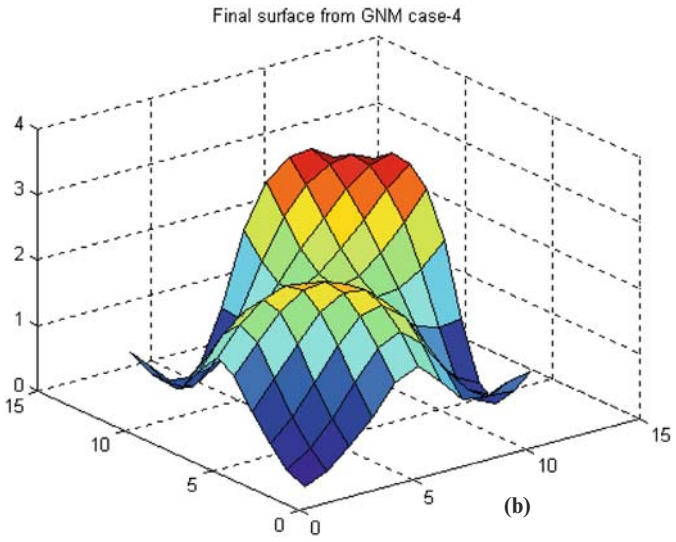


Fig. 5.33. (**a**) GNM-4 (**b**) Final output surface obtained from GNM-4

Table 5.18. Comparison of network complexity involved in ANN and GNM

Components	ANN	GN model-1	GN model-2	GN model-3	GN model-4
Number of neurons used	13	1	1	1	1
Number of layers in network	4	1	1	1	1
Number of interconnections in network	52	16	20	20	24
Number of biases	13	02	02	02	02
Number of aggregation functions	13	03	03	03	03
Number of activation functions used	13	4	6	6	8

Table 5.19. Comparison of ANN and GN model in one stroke for Mackey–Glass problem

Operations	ANN	GN Model-1	GN Model-2	GN Model-3	GN Model-4
No. of summations	13	02	02	02	02
No. of product	52	17	21	21	25
No. of divisions	13	02	02	02	02
No. of exponential functions computed	13	04	04	04	04
No. of sin functions computed	–	–	–	–	02
Total number of operations	91	25	29	29	31
Time consumed in one stroke (ms)	808	116.3	120.3	121.5	125.5

5.8 Summary

1. The training time of neural network models is a function of type and number of aggregation activation functions used and configuration among different functions. More number of configurations is possible in GN model as compared to ANN because of large number of aggregation and activation functions used, which is helpful to reduce the training time.
2. Among all models GN Model-4 suits best for character recognition, $\text{Sin}(x1) * \text{Sin}(x2)$ and coding problems taking minimum training epochs, however for Ex-OR, Parity–4 and Mackey-Glass time series problem GN Model-2 requires minimum training epochs to reach same tolerance level. It shows that GN Model has flexibility to select proper configuration of itself according to problem in hand.
3. The results reveal that convergence capability of GN Model is very good for all benchmark problems.
4. The requirement of the total number of neurons and hidden layers is reduced drastically in case of the GN models.
5. The GN model exhibits much superior property both in terms of convergence time during training as well as prediction error during testing.
6. The performance of generalized neuron model is better compared to ANN with noisy data also.
7. The structural complexity as well as computational complexity in GN Model is reduced as compared to ANN.
8. The computation time in seconds has also been reduced in GN Model as compared to ANN.

5.9 Exercises

1. Explain various factors on which ANN performance depends
2. Write a matlab program for one hidden layer ANN and study the effect of variation of hidden neurons.
3. What do you mean by over fitting of neural network? How this problem may be overcome?
4. Use generalized mean as aggregation function of GN and sigmoidal as activation function. Write a Matlab programfor this GN and train it with back-propagation algorithm.
5. Test the above developed GN for benchmark problems.
6. Write a Matlab program for GN training with adaptive backpropagation learning by varying learning rate and momentum factor.
7. Study the effect of noise on training and testing performance of GN.
8. Write step by step solution for training of GN with Gaussian as aggregation function.
9. Let us consider a function $f(x) = x^{2*}e^{-5X}$

Deteremine 100 training and 25 testing patterns for ANN and GN. Compare the performance of ANN and GN while training and testing. Also compare the performance of both if 5% random noise is added in training.