Chapter 10

ANN BASED SOFTWARE DEVELOPMENT AND APPLICATION

10.1 GENERAL REMARKS

In a Back propagation Neural Network (BPN), signals flow bidirectionally, but in only one direction at a time. During training, there are two types of signals present in the network. During the first half-cycle, modulated output signals flow from input to output and during the second half-cycle, error signals flow from output layer to input layer. In the production mode, only the feedforward, modulated output signal is utilized. Thus the BPNN simulator consists basically of two algorithms.

The first algorithm takes weighted input to find hidden node output which in turn are weighted and sent to the output node as an input and calculates the actual output. Difference of actual output and predicted output is termed as an error and is moved in backward direction to find errors at the hidden nodes. Based on this error the weights are modified and these modified weights are then used to again find the output. The process is repeated until the network converges. The second algorithm is for testing of network which just calculates the actual output by taking the final weights obtained after training.

In the present chapter a software is developed based on BPNN to check the possibility of combining it with other soft computing tools for optimization purpose in the next chapter. Use of BPNN is illustrated by implementing it for predicting confined compressive strength and corresponding strain of the circular concrete column.

10.2 COMPUTER IMPLEMENTATION

The VB program for BPNN simulator contains various forms and modules developed in pre-, main- and post-processors of the program. The forms are front end tools which prompt the user to enter input data and inform about ongoing process and show instantaneous cycle number and root mean squared error. The modules are back end tools which perform the actions such as reading data, carrying out all the calculations and creating output files. The function of three processors is explained below.

The **Pre-processor** helps the user define primary data such as number of input vectors, number of output vectors, number of training patterns, learning rate and number of training cycles and define network topology by specifying number of hidden layers and number of nodes in each hidden layer.

The program developed is menu driven which provides facility to easily define the network parameters which are required to train and test the network. Various menus and operation of program for training and testing the neural network are outlined in the following section.

The main form of the BPNN program contains four main menus as shown in Fig. 10.1. The operation of the simulator starts with clicking *Define* menu which contains two sub-menus, (i) *Net Parameters for Training* and (ii) *Net Parameter for Testing.* On clicking first sub-menu the data entry form for network parameters for training appears. This form prompts to enter number of training patterns, number of input vectors, output vectors and hidden vectors, number of hidden layers, learning rate and number of training cycles as shown in Fig. 10.2. After this, simulator prompts to select the option for supplying the weight matrices. User can create the file containing weight matrices and give this file name to the simulator or he can direct simulator to generate the weight matrices randomly as shown in Fig. 10.3. The second submenu is clicked just before testing operation starts. The *Train* and *Test* menus transfer the control to main-processor for carrying out training and testing respectively.

The **Main-processor** of the BPNN is comprised of two modules: (i) Module for training and (ii) Module for testing. Using the input data supplied by preprocessor the network is trained until convergence criteria is satisfied and final weight matrix is stored in the file specified by the user which can be used later for testing purpose.

In the module for training, the **ReadPatrain** subroutine reads the input patterns, **ReadWt** subroutine reads weight matrices and **RandomWt** subroutine generates weight matrices randomly. The **NeTrain** subroutine of the *Train* module carries out actual training process. During this, the algorithm of BPNN calculates the error between actual output and target output and adjusts the weight matrices to minimize the error. Figure 10.4 shows training process in the flow chart form indicating some of the calculations. Provision is made in the software to display convergence graph during the training process.

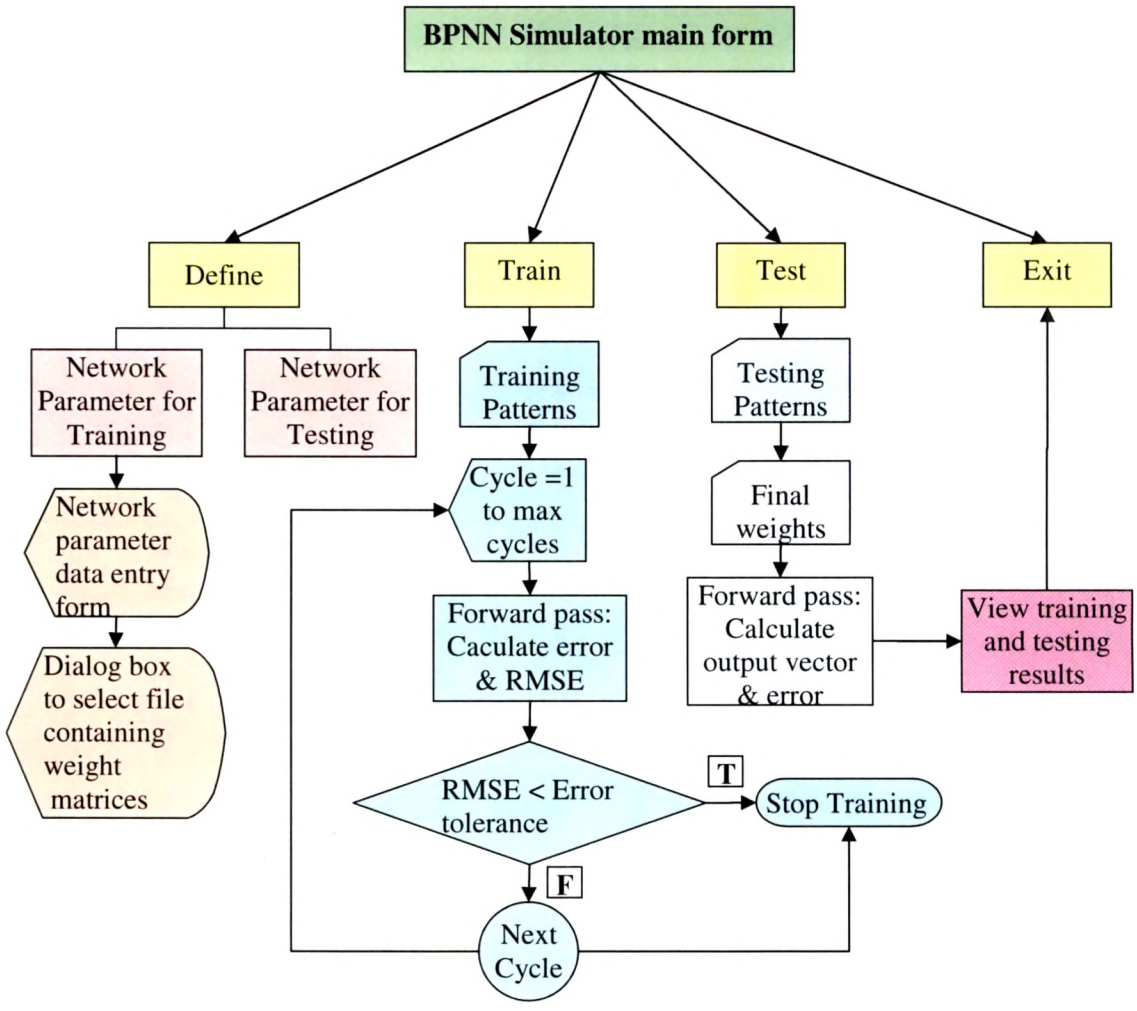

Fig. 10.1 Operation of BPNN Simulator

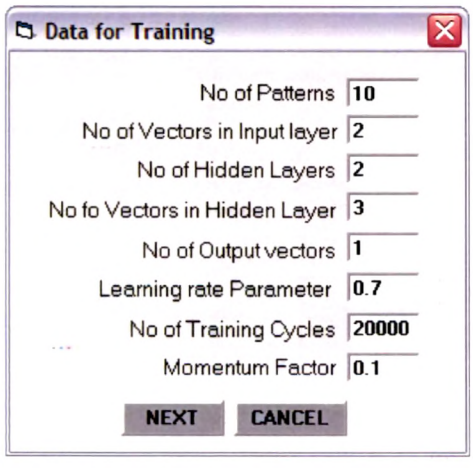

Fig. 10.2 Input Data Form

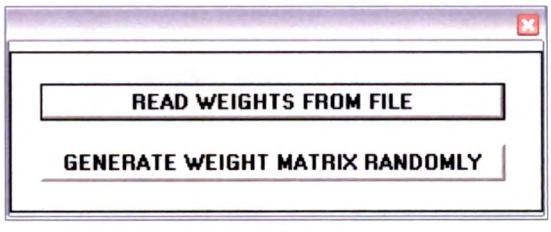

Fig. 10.3 Option Form

Module for testing reads the final weight matrix obtained after training and testing patterns and the resulting output values are stored in the specified file. The algorithm used for testing

is depicted in the flow chart form in Fig. 10.5. The post-processor is developed to view the predicted output for the testing data set.

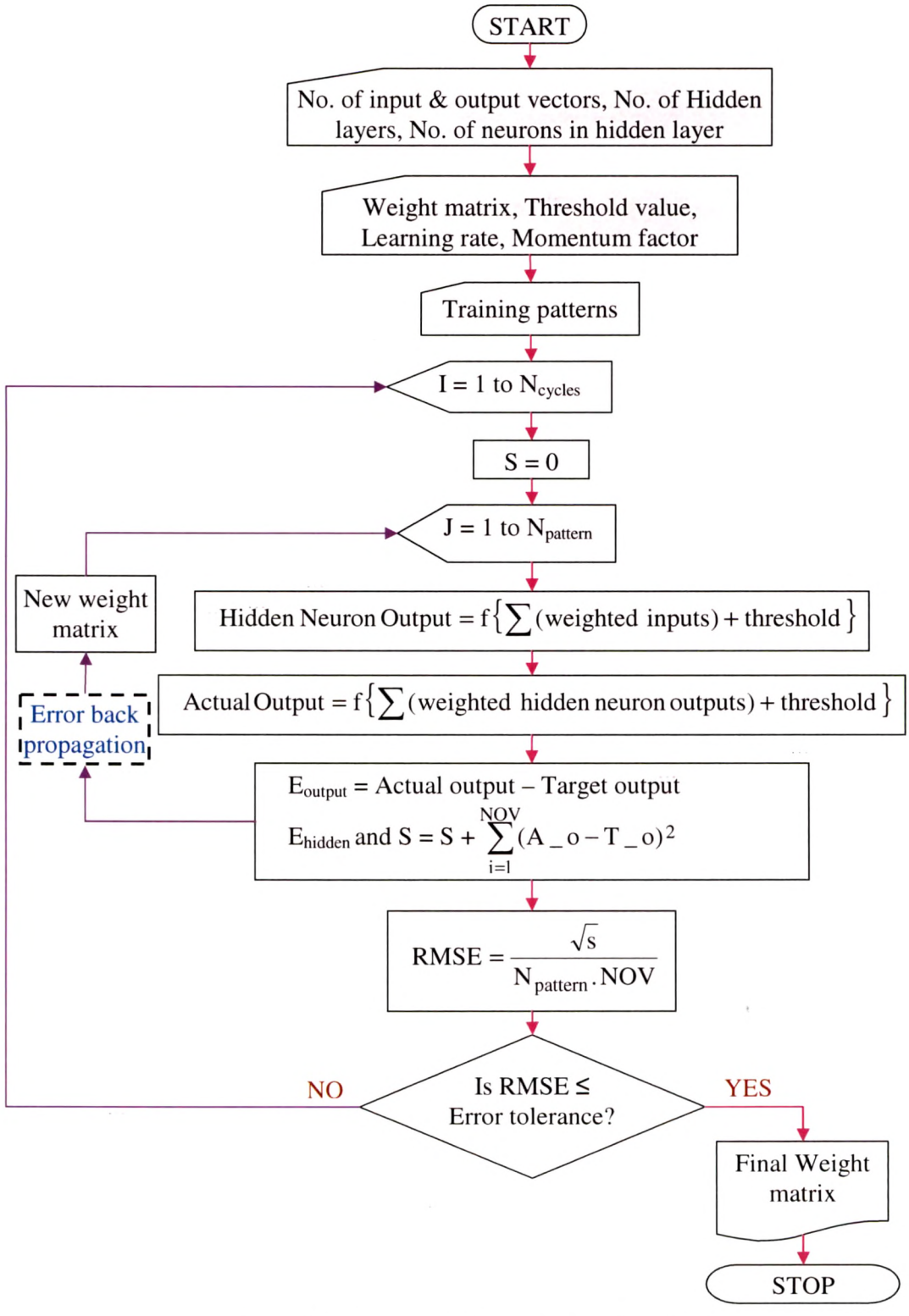

Fig. 10.4 Flowchart for Training Algorithm

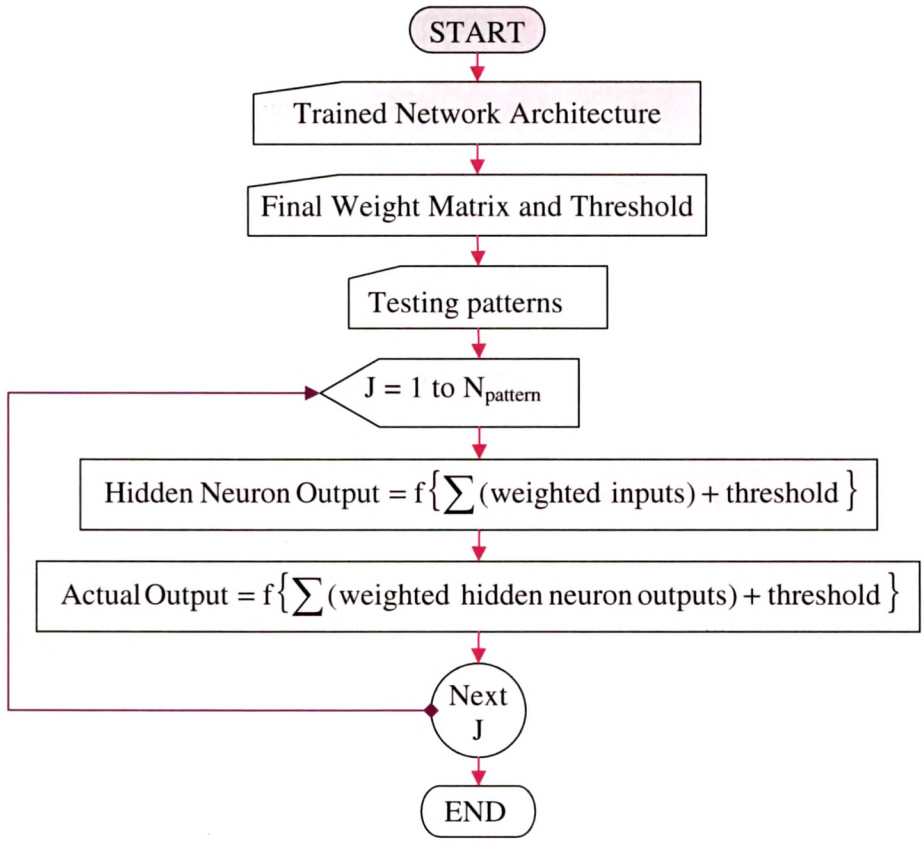

Fig. 10.5 Flowchart for Testing Algorithm

The **Post-processor** allows user to view final weight matrices which are found after training and to view the output for new input values which are supplied through a separate file.

10.3 CIRCULAR COLUMN EXAMPLE

BPNN simulator is used here to predict the confined compressive strength and corresponding strain of circular reinforced columns using available data from past experiments. Since the behaviour of confined concrete is influenced by various factors with a complex and sometimes unknown interrelationship and the experimental data available are "noisy" and limited [99], ANNs may be appropriately used as an alternative tool to the problem of predicting the compressive strength and corresponding strain of confined concrete columns.

Confined concrete may be described as concrete which when subjected to uniaxial compression is prevented from lateral "swelling', due to presence of transverse reinforcement in the form of closed ties (hoops) or spirals and longitudinal reinforcement bars. Confinement in concrete is effective only from the instant when internal cracking causes an increase in volume resulting into what is termed as passive confinement, which results in the

enhancement of the compressive strength of concrete and increase in ductility. In the experimental study carried out by Andres and Kazuhiko [99] there were a total of 38 columns considered. The 15 column specimens had 500 mm diameter and 1500 mm height. The core diameter of the column was 438 mm. All columns consisted of both lateral and longitudinal bars with varying sizes and spacing. Seven columns had 200 mm diameter and 600 mm height, and provided with both lateral ties and longitudinal bars (10 bars with 6.35 mm diameter). The 16 columns had 300 mm diameter and 900 mm height. Sixteen longitudinal bars with diameter of 9.53 mm were provided.

The ANN model is considered with seven input nodes, two output nodes and with varying number of nodes in the hidden layer. Only one hidden layer is used. The experimental data for 38 columns are grouped randomly into training and test data. Twenty-Nine data pairs are used as training data and the remaining nine data pairs are used as test data. Table 10.1 shows training sets used. A BPNN Simulator is used here with learning parameter as 0.01, error tolerance as 0.001, noise factor as 0.01, momentum factor as 0.1 and topology as 7 – 4 – 2 (Fig. 10.6). Training with 29 training patterns after 500,000 cycles gave RMSE of 0.0024 and took training time of 42 minutes on Pentium IV computer system. Fig.10.7 shows the convergence graph displayed during training. Table 10.2 shows testing results and percentage error between desired (D) and predicted (P) output.

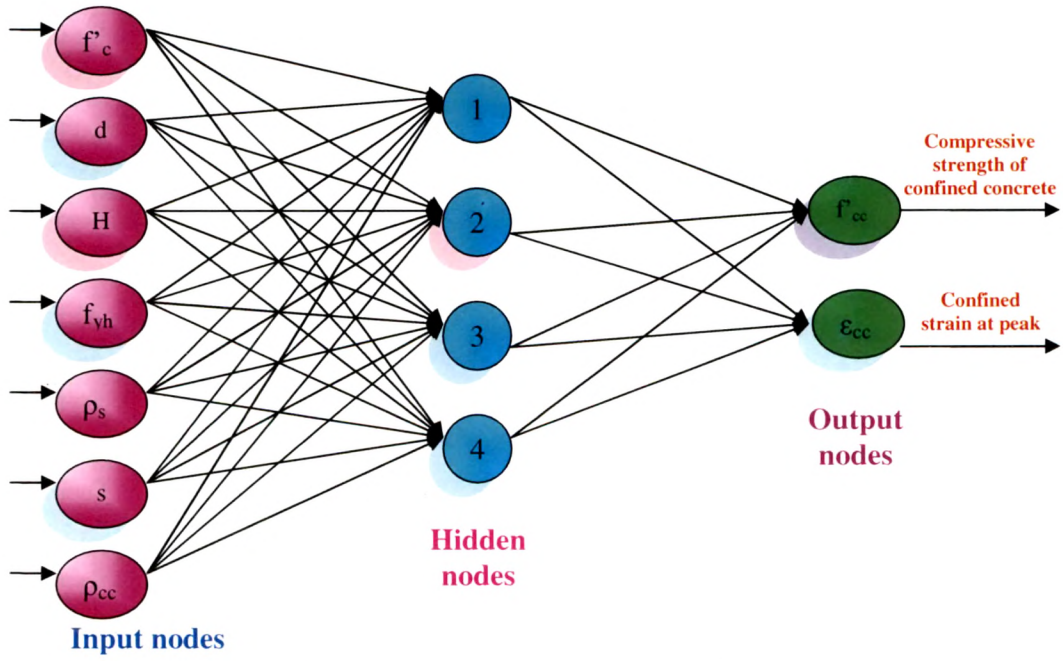

Fig. 10.6 ANN Topology 7 – 4 – 2

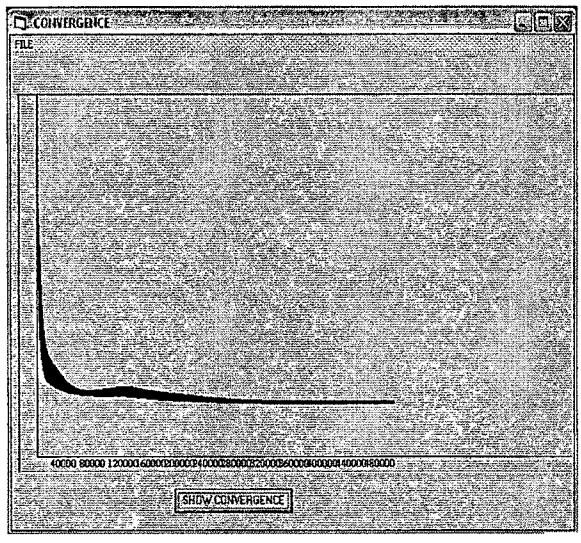

Fig. 10.7 Convergence Graph

Table 10.1 Training Data Sets for Circular Concrete Column Problem

INPUTS							OUTPUTS	
f'_c	d	H	f_{yh}	ρ_s	s	ρ_{cc}	f'_{cc}	ε_{cc}
28	438	1500	310	2	52	1.60	38	0.80
31	438	1500	340	2	52	1.60	48	0.42
33	438	1500	340	2	52	1.60	47	0.58
28	438	1500	340	1.5	69	1.60	46	0.50
28	438	1500	320	0.6	119	1.59	36	0.33
28	438	1500	320	2	36	1.59	47	0.65
28	438	1500	307	2	93	1.63	46	0.58
31	438	1500	340	2	52	3.27	52	0.57
27	438	1500	340	2	52	3.30	49	0.58
27	438	1500	340	2	52	3.20	50	0.64
27	438	1500	340	2	52	4.80	54	0.45
31	438	1500	340	2	52	3.20	52	0.56
29.8	185	600	376	1.14	60	1.18	29.7	0.58
29.8	185	600	376	1.14	60	1.18	34.4	0.49
29.8	185	600	376	1.71	40	1.18	35.9	0.59
29.8	185	600	376	1.14	120	1.18	36.0	0.40
29.8	185	600	376	1.71	80	1.18	36.1	0.57
19.45	280	900	363	2.26	20	1.85	35.4	1.30
19.45	280	900	363	1.51	30	1.85	29.7	0.80
19.45	280	900	363	1.31	40	1.85	27	0.62
19.45	280	900	363	0.57	80	1.85	22.8	0.42
19.45	280	900	363	0.38	120	1.85	19.8	0.31
19.45	280	900	363	0.28	160	1.85	19.3	0.24
19.45	280	900	363	2.26	40	1.85	33.8	1.38
19.45	280	900	363	1.51	60	1.85	27.8	0.71
19.45	280	900	363	0.75	120	1.85	22.3	0.37
19.45	280	900	363	0.57	160	1.85	20.1	0.34
19.45	280	900	363	2.26	60	1.85	34.1	1.06
19.45	280	900	363	1.13	120	1.85	22.4	0.48

Table 10.2 Testing Results for Circular Concrete Column Problem

INPUTS							OUTPUTS		
f'_c	d	H	f_{yh}	ρ_s	s	ρ_{cc}	*Type of Output*	f'_{cc}	ε_{cc}
28	438	1500	340	2.5	41	1.60	**D**	51	0.73
							P	46.16	0.77
							E	9.49	-5.48
28	438	1500	340	1	103	1.60	**D**	40	0.40
							P	45.80	0.38
							E	-14.5	5
31	438	1500	340	2	52	2.34	**D**	52	0.54
							P	49.40	0.52
							E	5	3.70
29.8	185	600	376	0.57	120	1.18	**D**	29.6	0.37
							P	22.98	0.37
							E	22.36	0
29.8	185	600	376	0.57	240	1.18	**D**	31.1	0.32
							P	19.67	0.25
							E	36.75	21.87
19.45	280	900	363	0.75	60	1.85	**D**	24	0.47
							P	24.56	0.46
							E	-2.33	2.12
19.45	280	900	363	1.31	80	1.85	**D**	25.4	0.58
							P	27.73	0.54
							E	-9.17	6.89
19.45	280	900	363	1.70	80	1.85	**D**	26.7	0.79
							P	30.42	0.67
							E	-13.93	15.19
19.45	280	900	363	0.85	160	1.85	**D**	20.3	0.33
							P	22.79	0.33
							E	-12.26	0

10.4 CLOSING REMARKS

ANNs are tools which are used to develop relationships between inputs and known outputs and then to use it for predicting output for unknown input. In the present chapter, ANN was used for simulation of structural analysis and design process. As the BPNN is very well defined mathematical simulation model its programming does not need any complicated algorithm. Rather selection of training and testing data set, selection of inputs for the desired output is more important and requires thorough knowledge of the problem at hand. Moreover, the selection of number of hidden layers and hidden neurons plays an important role in the convergence speed of the network. Since there is no mathematical theory available to precisely decide network architecture, intensive trial error process is required to decide not only the network architecture but also other ANN parameters like learning rate, momentum factor etc.

The percentage error obtained in testing results of confined compressive strength of concrete column is comparatively more as the training data sets contain many manual and instrumental error as they are out comes of experimental study.