CHAPTER-3

SOFTWARE IMPLEMENTATION

3.1 General

The realization of an artificial neural network and genetic algorithm was an important consideration for successful implementation of the problem selected for the present investigation. While selecting an implementation strategy, the case in selection of network configuration, training time required and the recall facility are the main considerations. The selection of a network configuration for a particular application is based on a study of performance of different network configurations. Therefore a network simulation tool is needed to speed up the process. Further this process is time consuming and often takes nearly 50% of the time required for development of the network. This in turn, demands a fast computational device for implementation. The hardware implementation offers increased speed for computations once a network configuration is selected and the weight and the threshold vectors are obtained. Artificial neural networks and genetic algorithms can be implemented as hardware in the form of electrical circuits consisting of operational amplifiers and resistors (Hopfield 1986). Recently neural networks are also being implemented using VLSI (Very Large Scale Integration) technology. (Castillo et al., 1991). The availability of such hardware is however, rather scant due to its high cost and machine requirements. However the number of attempts required for arriving at a particular configuration of the network is often too large especially when the input/output patterns are continuous valued. Therefore a software implementation is desired for the selection of network configuration and training.
The software implementation of the neural network and genetic algorithm is common for application purposes. A noticeable advantage of software simulation of feed forward network and genetic algorithm is that many different combinations of the network configuration can be studied quickly before arriving at the finally acceptable configuration. The selection of an optimal configuration for the network is important from computational time and memory requirements at training and application stage. As there are no fixed guidelines available for the selection of the network configuration at present, this is often achieved by watching the convergence of many different configurations. However, the size of the network influences the generalization capabilities also. With a neural network simulation tool, it is possible to watch the performance of different network configurations for a fixed number of training cycles. The optimal configuration of the network can then be selected from such a study of different configurations. A software simulation tool also helps in revising the configuration of the network while the numerical experiments for selecting an optimal configuration are in progress. The revision of the configuration is in the form of:

1. Revision of number of nodes in input/output layers,
2. Revision of numbered nodes of intermediate (hidden) layers,
3. Revision of input or output vectors,
4. Revision of scaling factors for the input or output nodes,
5. Revision of weight vector and threshold of neurons,
6. Revision of Learning Coefficient,
7. Revision of Momentum Factor and Tolerance,
8. Revision of Threshold Value and Sigmoidal Gain,
9. Revision of Iterations.

The training of the network is often a time taking process and sometimes, it may take weeks to train a network. It has been observed that the rate of learning is high for a binary type input or output pattern that is when the input or output vectors consists of either 0 and +1 or -1 and +1 as elements. On the other hand, when continuous valued inputs of output patterns are used, the training time is drastically increased. Different network parameters as well as the elements of input or output vectors are often changed to enhance the learning ability and the learning speed. The network parameters that may be changed during the training include momentum factor and the learning rate. The elements input or output vectors may also need modifications to facilitate fast learning. These changes, however, are often problem oriented and depend entirely on the problem complexity. The revision of scale factors for the input or output nodes are also often required. The scaled target value for a particular output node with a continuous value function is an important factor, which affects the training process. The numerical in terms of available significant digits for weight changes to occur may govern the learning rate. Therefore, it is desirable to have a facility to change the scaling factors during the training phase. A neural network simulator with a built-in facility for incorporating such changes is thus important. This helps in making the processing of arriving at the optimal configuration of the network quick and easy.

3.2 Software implementation:

Neural networks can be implemented as software simulations on serial processing personal computer, workstation and mainframe or on modern parallel processing super computer. The implementation of ANN on a parallel machine offers advantage of massively parallel processing in the form of fast computations. The development and training time required is also drastically reduced. Further, the inherent property of ANN
in the form of parallel computing algorithm can be best exploited. On the other hand, the software implementation of ANN on a serial processing personal computer is becoming more popular because of the availability and the low cost of these machines. In the present work, it is proposed to develop simple BPN models and hybrid neural network models for the structural design of R.C.C elements. It is proposed to implement these networks through software. Accordingly, separate simulators have been developed in C++ to implement the neural networks as mentioned below.

Simulator-I: To implement simple BPNs-NNSBPN (Neural Network Simulator for Back Propagation Network)

Simulator-II: To implement GA based BPN- NSSGABPN (Neural Network Simulator for Genetic Algorithm based Back Propagation Network)

The features of these two simulators are presented in the following sections.

3.2.1 Neural Network Simulator for Back propagation Networks (NNSBPN)

In the present work, neural networks have been implemented as software simulations. NNSBPN has been developed for this purpose in C++. This has facilities for speedy development and training of BP neural networks. The software offers a great deal of flexibility in development and training of a network for the selected application. With the help of NNSBPN, it is possible to create, revise and obtain revised weight matrix and threshold while training. The program has been coded in C++ programming language. Extensive use of graphics capabilities of C language has been made to develop a user friendly interface. The program simulates feed forward networks. The standard back propagation algorithm (BP) has been used for error minimization. Using the BP Algorithm, the error at the output nodes is back propagated along the steepest descent
direction over all the nodes within the network. The program can run under DOS/Windows environment.

The *minimum Hardware requirements* include:

1. Pentium IV Machine:
2. A 64MB RAM,
3. A 5GB Hard disk,
4. A CGA, EGA or VGA monitor.

The main features of the simulator are:

1. Menu driven user friendly interface
2. Facility to define/revise a network configuration
3. Facility to train the network
4. Facility to create/revise inference data
5. Facility to create/read file containing training input/output data
6. Facility to generate random weight vector
7. Facility to store scale factors separately.
8. Facility to generate scaled target training set
9. Facility to view the inference data, along with the error.

A brief description of the various modules used in the program is given below:

3.2.1.1 *Definition/Revision of a network configuration*

A network is fully described by NNSBPN using:
1. Number of input nodes

2. Number of output nodes

3. Number of hidden layers

4. Number of nodes in each hidden layer

5. Name of the file containing training set data

6. Name of the file containing weights and thresholds

7. Number of training examples in the case of learning

This option is available in training and inference files. The user can specify the desired network configuration by selecting the proper option available in these files. The parameters selected by the user to define a network are displayed on the screen. The user can make changes in the configuration by again selecting a desired option from the menu. The network configuration is stored by NNSBPN in a separate configuration file called *.con. This file is opened by NNSBPN for training a network or for obtaining output. This option facilitates development of many alternative configurations for using the same training data. However, when the size of the network is increased, it must be assured that the relevant weight and threshold vectors are also revised to account for the increased number of connection links and number of thresholds. The user can also change the name of the file containing the training set examples in a normalized form. This facilitates the use of different training set data files containing training examples which are prepared by using different scale factors. This study helps in arriving at an optimum network by watching the performance of different configurations for a fixed number of cycles. Different combinations of training set data and weight vectors are also possible. The increase and decrease in the size of the hidden layers does not create any memory
problems as dynamic memory allocation facility offered by 'C++' has been used to allocate the memory space for weight, threshold and other vectors. This module returns the control to the submenu.

3.2.1.2 Training of a network

For a given configuration of the network, NNSBPN uses the BP algorithm with default sigmoidal nodal function for training. The training algorithm presented in Chapter 2 is implemented. The training of NNSBPN involves selection of proper values of thresholds. The selection of proper threshold for a node or processing unit (PE) depends on the total weighed sum coming to it as input. However, as the network is initialized with the random numbers, the initial values of threshold of different neurons are often too low. On the other hand, very high values of the thresholds may take the output to extreme values i.e. 0 and +1. A proper initial threshold helps in speeding up the training process. Therefore, prior to starting the training epochs, this module provides an option for modification of threshold of different nodes in each layer. This facility greatly helps in setting the initial threshold for neurons. The user may wish to set the threshold values or skip this option. Once the initial values are set, NNSBPN starts the training cycles. The current cycle number is displayed on the screen. After reading the configuration file, proper memory is allocated for different vectors. An important feature of the module is that it allows the user to change the learning rate and momentum factors. These parameters play an important role in learning phase. Once the prescribed number of training epochs is over, the modified weights and thresholds are stored on a permanent storage device. The initially existing data file is replaced. This module returns the control to main menu.
3.2.1.3 Output from a trained network

This option provides facility to evaluate the trained progress and/or to obtain output for the unseen or new examples. The module requires the name of the configuration file. The input is read from the data file as per the specification in the configuration file. NNSBPN requires the new input values written in the same file format as used for the training of the concerned configuration file. The scale factors for the new values also must be the same as those used while training of the network. For a particular network, the scaling factors are stored in a separate file called *.scl which can be framed for the new input values. This option is invoked for either evaluating the progress of training or for watching the training progress, the average mean square for all the training examples is also calculated and is written at the end of the output file. It becomes easy to monitor the training process by watching the variation of mean square error over all the training examples. However in some cases, the average mean square error over all the nodes may not give a very comprehensive idea about learning. Especially, when the error surface has many ups and downs. A graphical study of performance of a particular output node is essential in such cases. Therefore, this module provides facility to watch the variation of the output for a particular node. The option to output the error for a particular node only has been provided. When this option is chosen, the output file shows the desired and predicted values of output for the selected node only. The mean square error for each node per example is calculated by the program and is mentioned in the output file before the respective output nodes.

3.2.1.4 Creation of training input/output vectors

The training input/output vectors are stored as data files on the disc. This data file is used for creating a scaled target vector for the network using the file containing the
scale factors for different input/output nodes. The user can create this data file using NNSBPN through this option. This module creates a file into which the unscaled target output and corresponding input vectors are stored. The training data can be entered through the keyboard. It is often required to read a file created by other analysis program or software as training set examples. This facility is offered by NNSBPN. The program can read a previously created data file as target input/output vector values. The format of such a data file should be such that it gives the target input for all the nodes first and then the corresponding target values for the output vector. The data so entered / read is stored in a format as required for future use by NNSBPN. A new data file containing the information about the input/output nodes and the number of training examples is created each time when this option is invoked. The program reads data for all input/output nodes for every example. This data file does not contain the scaled target values. This facilitates to use different scaling factors for the same data. The new file is stored as *.dat.

3.2.1.5 Generation of random weight vector

The weight and the threshold for different connection links and the nodes are often set to some small random numbers to initialize the network. This is facilitated by NNSBPN through this option. This module generates a random weight vector to initialize the network. The program asks for the information about the number of layers and the sizes of each layer. The total number of connection links and the thresholds required are automatically calculated. A random weight is assigned to each connection link. The thresholds are also assigned randomly. While the training of the network is in progress, the connection weights are updated automatically depending on the learning rate, momentum factor and the error at the respective nodes. On the other hand, the randomly initialized values of the various thresholds are either adjusted by using the BP algorithm
or they are also modified by the training supervisor in the training module. The values generated in this module are between 0 to +1. The small random values for the initialization have two advantages. First, it does not necessitate very high values for the thresholds of the nodes in upper layers. Secondly, the chances of network paralysis due the high weights are reduced. However, this does not help in overcoming the local minima problems if it arises. The main advantage of initiating the network with the random numbers is that it breaks the symmetry of a symmetric network. Thus a symmetric network can also be trained using this module.

3.2.1.6 Storage/Modification of scaling factors separately

The training of a network is very sensitive to the error at the output nodes for a software simulation. The use of a single scale factor for widely ranging input/output patterns may cause the “Premature Saturation” condition. Therefore, a facility is provided by NNSBPN in this module. This module creates a data file containing the scaling factors. The file is stored as *.scl. It is designed to store the scaling factors separately to facilitate the use of same training set with different scaling factors. This is generally needed if it is found that the scaling factors for the output nodes are too big or too small.

When the output values have large differences or when the range of output is large, it is often required to try many combinations of the scaling factors. This is mainly because:

1. If the output required for the training set examples has large variations, then the network may get trapped in local minima in some cases.

2. If the required output for training examples is too small, the network may not improve the performance due to the less floating point significant digits available on some machines.
3. If many of the required output values are close to +1, the network may get saturated i.e. it virtually stops the learning even without suffering network paralysis which is caused due to high weights values.

NNSBPN, therefore, offers the facility to store the scaling factors separately.

Thus, more than one scaling factor set can be used with the same training data.

3.2.1.7 Generation of scaled target training vector

As mentioned in the previous section, the user may be interested in using the same training set vector with different scaling factors. This is achieved using NNSBPN by providing facility to generate the scaled target vector. This module uses a training set which is already stored in a file called *.dat on the disc for creating the scaled target vector. The modulus requires the names of the training set data file and the file containing the scale factors. These two files are read to generate the scaled target vector. This is stored in a separate data file which is called *.res. The user can generate more than one *.res file by changing the name of the file containing scale factor (*.scl).

The stepwise procedure for implementing the software (NNSBPN) is presented below.

1. Edit a file TEST.DAT containing input and corresponding output in the order shown below.

   1) No of hidden layers
   2) No of input neurons
   3) No of hidden neurons
   4) No of output neurons
   5) Momentum factor
6) Tolerance
7) Learning coefficient
8) Threshold value
9) Sigmoidal gain
10) No of data sets
11) No of iterations

II. For each data set give the following:
12) No of input neurons (say L)
13) Since it is column vector give 1
14) L values (normalized) must be given one by one on each row
15) No of output neurons (say N)
16) Since it is column vector give 1
17) N values (normalized) must be given one by one on each row

III. Edit a file TEST.INF which compares network output and given output as shown below

First nine data exactly same as TEST.DAT
No of data sets to be inferred
No of input neurons - L
Give 1
Give L values (normalized) one by one on each row
No of output neurons - N
Give 1
Give N values (normalized) one by one on each row
Then run the program NNSBPN.CPP. It displays

1) TRAINING
2) INFRINGEMENT
3) EXIT

First the network is to be trained so give option 1.

The option for input file name will be displayed. Give TEST without any extension.

The iteration number and error will be displayed at the end of training.

Then again it displays

1) TRAINING
2) INFRINGEMENT
3) EXIT

Now option 2 is invoked for inference to compare the network predicted values and actual calculated values.

To exit from the program give option 3.

After training phase, three files will be created. Namely

TEST.WGT: This contains weights of synapses connecting input, hidden and output layers.

TEST.RST: This file contains the data of inferred results compared with actual values.

TEST.OUT: The output of the program.

3.3 Simulator-II for development of GA/BPN model (NNSGABPN)

NNSGABPN is the software developed to train hybrid neural network i.e., combination of genetic algorithm and back propagation neural network. The software facilitates to derive the weights by using genetic algorithm and later training is accomplished by back propagation algorithm. This simulator has been developed by using the code presented by Rajasekaran and Pai (2003).
The step wise procedure for implementation of this software is presented below.

**Step-1: Preparation of input and inference data files**

Two files are to be prepared. Namely

1. **Input data file**
   
   Name: some name_inp.dat
   
   Format:
   
   - Input data set size
   - Input 1   Output 1
   - Inputs 2   Outputs 2
   
   Use of the file: Input data file is used for training the given data set.

2. **Infer data file**
   
   Name: some name_inf.dat
   
   Format:
   
   - Infer data set size
   - Input 1   Expected output 1
   - Input 2   Expected output 2
   
   Use of the file: Inference file is to test the performance of the network towards prediction of new problems.

**Step-2: Execution of NNSGABPN**

The following steps are adopted to obtain the weights for a given input and output data sets.
1. Get the input data file ready (As explained in the Step-1)

2. Run the EXE file GANNWT.EXE

   It displays two options START (0) and REBUILD (1). At the beginning
   START(0) is opted for training the neural network to obtain the weights.

   **Step-3:** Enter the following

   i) Enter the input data file name
   
   ii) Seed for random number generation (Max:32767):
   
   iii) Number of input neurons:
   
   iv) Number of hidden neurons:
   
   v) Number of output neurons:
   
   vi) Population size

   After training, the output of the network is stored in the file GAOUT.DAT

3.3.1 **The Other executable files called during execution**

   I. RANWT09.EXE- This EXE file generates random weights positive or negative in
to two different files (WIH09.DAT, WHO09.DAT) for input-hidden (I-H) and
hidden output (H-O) layers of the network.

   II. JOINWT.EXE-This EXE file combines I-H weights in WIH09.DAT and H-O
weights in WHO09.DAT as a single GENETIC.DAT. Weights for each
chromosome of the population appear on a single line to facilitate the application
of genetic operators.

   III. CONWT.EXE- This executable file converts weights to be in the range -5 to +5.
Accessing data (Chromosomes representing the weights) existing in
GENETIC.DAT. Final weights are in WIH5.DAT and WHO5.DAT.
IV. FITGEN.EXE-This EXE file reads weights form WIH5.DAT and WHO5.DAT and calculates error in OUT.DAT and Inverse error in FITNESS.DAT.

V. MATEPOOL.EXE- This EXE file reads the fitness values from FITNESS.DAT and creates FIT.DAT the fitness values of the best fit chromosomes.

VI. CROSSPT.EXE-This executable file applies double cross over function on FIT.DAT

VII. REPROD.EXE: This executable file builds new generation of chromosomes to create new weights.

This process repeats for each cycle and generates new set of weights. The process terminates after the fixed number of cycles are completed.

3.3.2 Output of the hybrid network model

1. Open file FITNESS.DAT to check for the convergence of fitness values. If convergence has not been attained run GANNWT.EXE calling the REBUILD (1) option. REBUILD begins generation from the last formed population of chromosomes and their acquired fitness.

2. Open WIH5.DAT and WHO5.DAT to view the weights determined by the GA based network. WIH5.DAT records the Input-hidden layer weights and WHO5.DAT records the Hidden-Output layer weights. The weights recorded by each chromosome in the population can be found. However depending on the percentage of convergence, that many chromosomes would record the same weights.
3. If convergence has been attained, Run GAINF.EXE to infer the output of the testing data set. The outputs given in infer data file is used by the program to compare it with the Computed output by the Neural Network.

4. Check computed outputs displayed on the screen as well as at the end of the inference session in the file GAOUT.DAT. The expected output is followed by the computed output for each of the testing data set.

3.3.2.1 Other intermediary data files

The program generates many intermediary data files some of which may not be of direct use to the user. However, FITNESSHIST.DAT, MATEHIST.DAT, ERRHIST.DAT are some archive files which record the history of fitness values of the population, the population of chromosomes and the error respectively.

3.4 Important remarks

- Population size is determined as:

  Population size = Number of weights to be determined * 5. Here 5 is the number of digits allotted for each gene in the real coded chromosome. Hence 5 is the gene size.

- Total length of the chromosome is restricted to 250

- Population size should be an even number

- At the time of execution, enter a large number for the seed of the random number generator (max 32767)

- Choose Start (0) or Rebuild (1) depending on whether the existing generations are not to be considered or to be considered respectively
• FITNESS.DAT registers the fitness function values of the current population. Check off and on after every epoch to know convergence.

• The configuration of the Neural Network viz., Number of input neurons, hidden and output neurons are interactively obtained by GANNWT.EXE

3.5 Summary:

1. The artificial neural networks can be implemented as hardware as well as software simulations. The hardware implementation using the VLSI technology and the electrical circuits is desirable from the computational time requirements. On the other hand, the software implementation of neural network is popular because of the flexibility and ease of selecting the network configuration offered by such simulation.

2. In the present work, the neural networks are implemented through software. For this purpose two simulators have been developed.

Simulator -I

Simulator-II

Simulator-I is used to implement simple BP Networks and simulator-II is used to implement GA based BP Networks. The various features of these two simulators have been presented in this chapter. Development of network models for structural designs of different R.C.C. elements will be presented in next chapters.