Chapter - V
Chapter - V

Application of Genetic Algorithm for Optimization of Load Forecasting Parameters

This part of the thesis presents a Genetic Algorithm (GA) solution to the optimization of the number of rules in the inferencing layer of a fuzzy-neural network. The optimized number of rules in turn optimize the network structure. The performance of the network is validated with extensive simulation results using practical data.

Here, we address the problem of load forecasting by optimizing a fuzzy neural network that was modeled in chapter IV. The number of rules in the network are optimized using an algorithm called, Basic Genetic Algorithm (BGA). The network performs satisfactorily starting from an initial set of random rules, but this gets optimized after a genetic search for a perfect set of rules using BGA.

5.1 Introduction

Genetic Algorithms (GA) is a search algorithm based on the mechanics of natural genetics and natural selection [5.1]. It combines the adaptive nature of natural genetics or the evolution procedures of organs with functional optimizations. By simulating the "the survival of the fittest" of Darwinian evolution among chromosome structure, the optimal chromosome (solution) is searched by randomized information exchange.
Recently, genetic algorithms have been widely applied to different optimization problem [5.1]-[5.4]. Unlike the conventional algorithms which optimize the problems from a single direction, the genetic algorithms for the optimal solutions from a multiple directions. This allows the genetic-based method to have a better chance of finding the optimal or near-optimal solutions. Besides due to the simplicity in programming, the genetic algorithms can be considered to replace the gradient descent method [5.5]-[5.8] to automatically optimize both the parameters and the structure in the fuzzy system.

In order to search for a given problem, a fitness function is defined to evaluate the performance for each chromosome. The well known roulette wheel selection criterion [5.6] is adopted here to decide whether a chromosome can survive or not in the next generation. The survival chromosomes are then put into a mating pool for the crossover and mutation operations. Once a pair of strings have been selected for crossover, a randomly selected site is assigned into the to-be-crossed strings. One sub-string then exchanges part of its string (from the crossover site to the end of the string) with the other’s. The newly-crossed strings join the rest of the chromosomes to form a new population. The mutation operation follows the crossover to offer a chance for each bit to flip. This completes the evolution of a new generation.

The evolution continues till certain preset criteria are met. So, in every generation, a new set of artificial chromosomes is created using bits and pieces of the fittest of the old ones. Though it is randomized, GAs don’t follow a simple random walk. It efficiently exploits historical information to speculate on new search points with expected improved performance.

5.2 Optimization of Fuzzy-Neural network Using GA

The objective of the present approach is to study the optimization of a Fuzzy-Neural-Network (FNN) using Basic Genetic Algorithm (BGA), which combines self-organizing capability of neural networks and fuzzy logic reasoning attributes along with the evolutionary concept of BGA in getting a fit candidate to an optimization problem.

Before we can apply the BGA to the optimization of the model, we want to give the outline of the FNN structure already given in chapter IV to validate its usefulness here.
The network modeling starts with a random set of weights, so an arbitrary set of fuzzy sets are initially involved. Then the training algorithm used was the error back propagation (BP) technique to update the weights and the associated parameters of fuzzy membership function. Here, we used the back propagation algorithm instead of GA. This is because, GA is computationally intensive when the dimension of the set of parameters is large.

Similarly, the network is also initialized with a sufficiently large number of rule-nodes. Here, we like to optimize this rule set using Basic Genetic Algorithm (BGA). This approach provides a model with a reduced dimension and helps in the better implementation of the forecasting model.

5.3 Overview of the FNN Forecasting Model

One of the salient aspects of Fuzzy Inference System (FIS) is the determination of the knowledge base (KB) which consists of the following subsystems:

- Mechanism for developing membership functions.
- Fuzzy reasoning mechanism.
- Number of rules and the rule base.

The network consisting of input, fuzzification, inferencing, defuzzification layers are shown in Fig. 5.1. The network has $N$ number of input variables with $N$ neurons in the input layer, $R$ number of rules with $R$ neurons for inferencing, thus number of neurons in the fuzzification layer is $N \times R$.

The inputs to the model are chosen as follows:

$$X = [Y_a(k - \Delta + 1) ... Y_a(k - n\Delta + 1) ... T_a(k + 1) ... T_a(k - \Delta + 1) ... T_a(k - n\Delta + 1)]$$

and, the output $Y = Y_a(k)$. where, $Y_a(k)$ represents the load

- $T_a(k)$ is the temperature at kth hour and
- $\Delta$ is the time step ahead for which forecasting is desired.
The past loads are taken to improve upon the prediction capabilities; the notion being similar to that of auto-regression. The temperatures are included to reflect weather sensitivity of the load. It should be noted that \((k+1)\)th element of the input vector is the \textit{apriori} information of the temperature of the hour at which load forecasting is to be done.

\[ \text{Input Variables} \]

Input to the fuzzification layer is a weighted version of input variables. Output of each neuron in the fuzzification layer is a fuzzy membership corresponding to a particular input variables. The defuzzification layer (4th) has the connecting weights to the output from the inferencing layer, and these weights signify the strength of each rule in the output of the model.

\[ \text{Figure 5.1 Load forecasting Model} \]

Input to the fuzzification layer is a weighted version of input variables. Output of each neuron in the fuzzification layer is a fuzzy membership corresponding to a particular input variables. The defuzzification layer (4th) has the connecting weights to the output from the inferencing layer, and these weights signify the strength of each rule in the output of the model.

5.4 Training and Model Adjustment

5.4.1 Training Procedures

The proposed forecasting model is trained to minimize an objective function. The performance index \((PI)\) to be minimized is the \textit{Mean-Square-Error} (MSE) given by,

\[ PI = (y_d(k) - y(k))^2 \]

where, \(y_d(k)\) is the desired output and \(y(k)\) is the model output. The model parameters of the proposed FIS are updated using the notion of error back-propagation.
We try to optimize the performance index $PI = (y_d(k) - y(k))^2$. The $W_j$, $W_0$ and $y$ are updated till some stopping criterion is reached.

The training is continued until the following stopping criteria is reached.

$$\sum_{k=i}^{i-50} PI_k \leq \varepsilon \quad \text{or,} \quad k \geq I_{\max}.$$ 

where, $\varepsilon > 0$ and $I_{\max}$ is the maximum number of iterations allowed. In our present implementation $I_{\max} = 2400$.

If after completion of training the performance of the model is not found to be satisfactory then, the weights are reinitialized, the number of rules are increased and the above mentioned procedure is repeated. Thus, this procedure initiates a linear search for the best number of rules in a FNN structure.

5.4.2 Model Optimization

After achieving a suitable level of performance over the entire training range, the following optimization procedure is followed to optimize the size of the network:

i. The rule nodes which produce an output close to zero over the entire training set, are searched. Obviously, these nodes do not contribute significantly to the model output. Hence, these rule-nodes can be safely omitted without affecting the model performance.

ii. Again, inputs for which the fuzzy membership is unity or close to unity over the entire training set, do not play any role in the actual model. These input variables can be traced down and are omitted from the network.

The model is tested without any retraining and if the performance is still satisfactory without the above mentioned rule-nodes and inputs, then we obtain a model with reduced size. This makes the network computationally more efficient. However, if the performance of the network is not found suitable then the network has to be retrained with the reduced set of rules and inputs.
However, this process involves inferencing done by a human expert and the process requires repeated training and retraining with different set of rules. To eliminate this we introduce here the BGA for the optimization of the number rules. This at the same time reduces the dimension of the model and helps in real time implementation.

5.5 The Genetic Algorithm (GA) Approach

Here, we have introduced the concept of optimization to the selection of the number of rules in a fuzzy-neural network. We choose the rule for the optimization because, it is a singular variable in the present network. GA are computationally intensive and not suitable for optimization of large number of parameters. So, optimization of weights, number of inputs and other parameters in the present network would take much time, in a PC based forecasting model to be practically useful.

So, determination of the rules is treated here as an optimization problem. This algorithm exploits the concept of natural evolutionary process to reach the solution after repeated crossover, mutation and selection procedures. The components of GA are:

(i). Creation of a population P candidate solutions which are encoded in binary strings are generated at random or representation of candidate solutions by chromosomes.

(ii). The fitness function for the evaluation of the degree of fitness of the solutions. Here the fitness $f_{\text{max}}(P)$ corresponding to each candidate solution is computed.

(iii). The crossover operator as a mechanism for generating new candidate solutions (child chromosomes) from selected old candidate solutions (parent chromosomes). Here, offspring are generated by applying two point crossover operator and mutation operators.

(iv). The mutation operator which operates on chromosomes to introduced new information in the chromosomes.

(v). The above process is repeated until the stopping criterion is reached. The algorithm is stopped when more than half the population have equal and high fitness.
Figure 5.2 shows the general flow chart of the proposed genetic approach for optimization of rules in a fuzzy-neural network. The detail process is shown in the Fig. 5.3.
5.5.1 Parameter Selection (selection mechanism)

Like other stochastic methods, the GA has a number of parameters that must be selected. These include - size of population, crossover probability and probability of mutation.

Usually, a relatively small size of population, high crossover probability, and low mutation probability is recommended. Here, we have taken the following values in our problem.
5.5.2 Encoding and Decoding (representation)

Implementation of a problem in a GA starts from the parameter encoding (i.e. representation of the problem). The encoding must be carefully designed to utilize the GA's ability to efficiently transfer information between chromosome strings and objective function of problem. So, the initial population of $k$ parent vectors $P_i$, $i = 1, 2, \ldots, k$ is generated randomly from a reasonable range in each dimension. Typically, the distribution of initial trials is uniform. This population of chromosomes is represented in Fig. 5.4 shown below:

\[\begin{array}{cccccccc}
1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
\end{array}\]

\[\begin{array}{cccc}
1 & 0 & 1 & 0 \\
1 & 1 & 0 \\
1 & 1 & 0 \\
\end{array}\]

Figure 5.4 Representation of Fuzzy rules in Fuzzy-Neural Network

In our problem we have tried to optimize the structure of fuzzy-neural network meant for the forecasting of electric load. Specifically, we want to optimize the number of fuzzy rules in the network. So, in adopting GA to the optimization problem that this section is dealing with, each chromosome represents a candidate solution consisting of numbers of rules in the fuzzy neural network. Each element in a chromosome mentions the number of rules that the network can have, which is coded using a binary integer number coding method.

In the generation of new candidate solutions we introduce some genetic operations like: fitness function, crossover and mutation. In the next section the procedure followed has been explained.
5.5.3 The Fitness function and Parent Selection

Implementation of a problem in a genetic algorithm is realized within the fitness function. The proposed approach uses the training error as its basis. The constraint equation is written as:

$$\varepsilon = (y_d(k) - y(k))$$

where, \(\varepsilon\) is the training error, \(y_d(k)\) is the desired output and \(y(k)\) is the model output.

Then, the converging rule is obtained, when error \(\varepsilon\) decreases to within a specified tolerance. In order to emphasize the "best" chromosomes and speed up convergence of the iteration procedure, fitness is normalized into the range between 0 and 1. In this problem fitness is defined as:

$$f_{\text{fitness}}(P) = \frac{1}{1 + \varepsilon}$$

where, \(\varepsilon\) represent the error at the output node of the net resulting from a particular candidate solution.

Then, we used the following selection mechanism for selection of the offspring for the next generation.

1. If the average fitness of the offspring (children) are more than that of the parents, then offsprings are considered as parents for the next generation.
2. Otherwise take parents as offsprings.
3. If the fitness of the offspring is more, offsprings replace the parents at the same position.
4. Thus out of \(2P\) population \(P\) candidate solutions are considered for the next generation by using the selection mechanism.

The selection mechanism
5.5.4 Crossover

Crossover is an extremely important component of the GA. It is a structured recombination operation. This is similar to students exchanging information in a group discussion. This operator is applied with a certain probability. When applied, the parent genotypes are combined (exchange bits) to form two new genotypes that inherit solution characteristics from both parents. In the opposite case, the offspring are identical replications of their parents.

In the generation of new candidate solutions, the 2-point crossover (Fig. 5.5) method [5.8] is adopted in the present work. In 2-point crossover, the points are randomly decided as in single-point crossover. The elements in between the 2 selected points are swapped between two parent chromosomes to form two child chromosomes. The crossover operation is initiated when a random number generated between 0 and 1, rand[0,1], is less than a preset value of the probability of crossover, here this is taken with a value of 0.6.

![Figure 5.5 Two-point crossover](image)

Figure 5.5 Two-point crossover

The crossover scheme used in a multi-point crossover version is shown in Fig. 5.6. Here, multiple points are randomly generated as in single-point or two-point crossover. Similarly, the elements in between the selected points are swapped between two parent chromosomes to form two child chromosomes. The crossover operation is initiated when a
random number generated is less than value of the probability of crossover. But in our problem we preferred to use the two-point crossover instead of multi-point crossover as it reduces the performance of GA.

![Multi-point crossover](image)

**Figure 5.6 multi-point crossover**

### 5.5.5 Mutation

Although, reproduction and crossover effectively search and recombine existing chromosomes, they do not create any new genetic material in the population. Mutation is capable of overcoming this shortcoming. It is an occasional (with small probability) random

![Mutation Operator](image)

**Figure 5.7 Mutation**
alteration of a chromosome position as shown in the figure 5.7. This provides background variation and occasionally introduces beneficial materials into the population.

So, mutation is applied to chromosome that is formed after a crossover and the value of an arbitrarily selected element in the chromosome is changed. There are two type mutations: single point mutation and multiple point mutation. But, usually single point mutation is applied, as multiple point mutation may alter the character of a chromosome too much. Mutation is initiated only if the random number is less than the pre-specified value ie. 0.01 of the probability of mutation.

To reduce the memory requirement for storing the s chromosomes, the BGA is modified such that - new child chromosomes replace chosen chromosomes in the current generation instead of being stored in the next generation [5.9]. This means that the population of generation of chromosomes consists of parent and child chromosomes. The modified BGA can be termed as the Incremental Genetic Algorithm (IGA). The above modification is indicated by the dotted-line-path in the flowchart of Fig. 5.3.

The above procedure is repeated until P new genotypes are produced which are considered as the new generation of solutions. The new generation totally replaces the parents.

5.6 Simulation Results and Evaluation

Here, each string of the entire population represented the rule of the fuzzy neural network. After the optimization process, the best rule in the population over each generation
is illustrated in Fig. 5.8. After 31 generations, the value of rule in each generation converge towards the best one. The figure below shows the number of rules settling to an optimal value. In our goal of finding the number of rules through GA, we found 16 rules are necessary for the optimum performance of the model.

However, we had got the same number by the previous procedure, that initiates a linear search for the best number of rules in a FNN structure. However, this process involved inferencing done by a human expert and required repeated training and retraining with different set of rules. There, after achieving a suitable level of performance over the entire training range, the rule nodes which produce an output close to zero are searched and these rule-nodes were omitted. Similarly, inputs for which the fuzzy membership is unity or close to unity over the entire training set are traced down and are omitted from the network.

Thereafter, the model is tested without any retraining and if the performance is still satisfactory without the above mentioned rule-nodes and inputs, we obtain a model with reduced size. However, if the performance of the network is not found suitable then the network has to be retrained with the reduced set of rules and inputs. This takes a lot of time to develop a model. Here, GA can came to rescue and save a lot of time.

As the number of rule set obtained remains the same in both the cases, the forecasting results obtained using GA optimization can be referred to the results obtained by FNN model in chapter IV.

5.7 Conclusion

Though the selection process is randomized, GA doesn’t follow a simple random walk. It efficiently exploits historical information to speculate on new search points with expected improved performance. This processes is utilized here to give us the optimized number of rules for the network. It also allows the genetic-based method to have a better chance of finding the optimal or near-optimal solutions, than the processes that initiates a linear search. Besides, GA based optimization processes helps in reducing a considerable amount of time in building a model.
5.9 References


