Evolutionary Design of Radial Basis Function Neural Network for Data Modelling

Alparslan Kaplan

Submitted to the
Institute of Graduate Studies and Research
in partial fulfilment of the requirements for the Degree of

Master of Science
in
Computer Engineering

Eastern Mediterranean University
December 2012
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Elvan Yılmaz
Director

I certify that this thesis satisfies the requirements as a thesis for the degree of Master of Science in Computer Engineering.

Assoc. Prof. Dr. Muhammed Salamah
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Computer Engineering.

Asst. Prof. Dr. Adnan Acan
Supervisor

Examining Committee

1. Assoc. Prof. Dr. Muhammed Salamah

2. Asst. Prof.Dr. Adnan Acan

3. Asst. Prof. Dr. Arif Akkeleş

ABSTRACT

In this thesis, implementation of Radial Basis a Function Neural Network (RBFNN) using genetic algorithm is described. The developed algorithm is used to model a certain dataset by training a RBFNN using some part of it, and then testing the performance of this RBFNN using the rest of data. The objective function of the proposed algorithm is to minimize the error between the computed output by the model and the target output given in the dataset.

The genetic algorithm used in this thesis is an evolutionary algorithm that uses natural evolutionary process for selection and reproduction. An individual is constructed from the RBFNN parameters, which are hidden units, centers, weights, widths and bias associated with hidden units and output of RBFNN. Therefore, the fitness values are also assigned to all chromosomes as a result of getting the difference between the target output and the computed output by the RBFNN, in which a Gaussian function was used as an activation function.

In experimental results, different tests were conducted in order to see the performance and correctness of the developed model. Since the number of hidden units plays an important role as well as weights, the intervals of weight values were adjusted accordingly and the number of hidden units was changed for different tests. As a result of conducted experiments, it is observed that the developed algorithm is successful in obtaining good results by minimizing the error.

Keywords: Evolutionary algorithms, Radial Basis Functions, Data Modeling.

ÖZ

Bu tezde Radyal Tabanlı Fonksiyonlar Ağı (RTFA), genetik algoritması kullanılarak tasarımlanmıştır. Geliştirilen algoritma modellenmesi hedeflenen belirli bir veri kümesinin bir kısmını öğretme geri kalan kısmını test için kullanmaktadır. Geliştirilen algoritmanın amaç fonksiyonu bulunan model ile verilen hedef çıktı arasındaki hatayı en aza indirmektir.

Bu tezde kullanılan genetik algoritma bir evrimsel algoritmadır. Bu genetik algoritmadaki bireyler RTFA parametrelerinden oluşturulmuştur: gizli birimler, merkezler, ağırlıklar, genişlikleri ve sapma. Bireylere verilen uygunluk değeri olarak, bulunan model ile verilen hedef çıktı arasındaki fark kullanılmaktadır. RTFA'nın gizli birimlerinde aktivasyon işleri olarak Gauss işleri kullanılmıştır.

Deney sonuçlarında, geliştirilen modelin performans ve doğruluğunu kontrol etme amaçlı farklı testler uygulanmıştır. Deneysel sonuçları elde ederken RTFA parametrelerinden en önemli iki parametre olan gizli birimlerin sayısı ve ağırlıkların değerleri değiştirilmiş ve sonuçlar gözlemlenmiştir. Deneysel sonuçlar neticesinde geliştirilen uygulamanın, iyi sonuçlar bulma ve hatayı en aza indirmede başarılı bir yöntem olduğu tespit edilmiştir.

Anahtar Kelimeler: Evrimsel algoritmalar, Radyal Tabanlı Fonksiyonlar Ağı, Veri Modelleme.

ACKNOWLEDGEMENTS

First of all, I would like to thank my dear parents–Latif Kaplan and Ayten Kaplan for their patience and support. My brothers' suggestions and help were invaluable for me throughout the time I was writing my thesis. Their beleive in me made me strong in accomplishing this thesis successfully.

My sincere thanks go to my supervisor Asst. Prof. Dr. Adnan Acan for his direction, assistance and guidance. His recommendations and suggestions have been invaluable for the thesis.

I am also thankful to Zhavat Sherinov for his advises and help throughout the thesis. And I also thank Cem Ayas for his support and motivation.

I am particularly indepted to my wife Gülşah Kaplan for her patience and being always next to me, who used to give confidence to me throughout the thesis.

TABLE OF CONTENTS

ABSTRACT iii

ÖZ iv

ACKNOWLEDGEMENTS v

LIST OF FIGURES viii

LIST OF TABLES x

1 INTRODUCTION 1

2 THE DATA MODELING PROBLEM IN INDUSTRY 3

3 GENETIC ALGORITHMS 6

3.1 Genetic Operators 8

3.1.1 Selection 9

3.1.2 Crossover 10

3.1.3 Mutation 12

3.1.4 Reordering 14

4 RADIAL BASIS FUNCTION NEURAL NETWORK 15

4.1 General Formula of RBF 17

4.2 Types of RBFNN 18

4.4 Training Algorithms of RBFNN 19

4.5 Least Squares 20

4.6 RBF Properties 20

4.7 RBFNN Improvements 20

5 THE DEVELOPED MODEL 22

5.1 Genetic Operators 24

5.1.1 Constraints 24

5.1.2 Arithmetic Crossover.. 25

5.2 Mutation 25

5.3 Chromosome Representation and Data Structure 25

5.4 Implemented Objective Function 28

5.5 Cross Validation.. 30

5.5.1 RBFNN Parameters 30

5.5.2 K-Fold Implementation 31

5.6 Local Search 32

5.6.1 Pattern Search 32

5.7 Normalization... 34

5.8 Algorithm Description... 34

6 EXPERIMENTAL RESULTS.. 39

7.CONCLUSION... 46

REFERENCES.. 48

LIST OF FIGURES

Figure 3.1: Genetic Algorithm Flowchart 8

Figure 3.2: One - Point Crossover 11

Figure 3.3: Two–Point Crossover 12

Figure 3.4: Uniform Crossover 12

Figure 3.5: Swap Mutation 13

Figure 3.6: Point Mutation 14

Figure 3.7: Mutation Operator 14

Figure 4.1: RBFNN Structure 17

Figure 4.2: Gaussian RBF 18

Figure 4.3: Neuron Influence 19

Figure 5.1: RBFNN Design Procedure 23

Figure 5.2: Chromosome Representation 26

Figure 5.3: General Skeleton of RBFNN Combining with GA 28

Figure 5.4: K-fold Cross Validation 31

Figure 5.5: Algorithm Main Flowchart 36

Figure 5.6: Evolution Process 37

Figure 5.7: Implementation of Test Part 38

Figure 6.1: Training results with 3 hidden units 40

Figure 6.2: Test results with 3 hidden units 40

Figure 6.3: Training results with 5 hidden units 41

Figure 6.4: Test results with 5 hidden units 41

Figure 6.5: Training results with 10 hidden units 42

Figure 6.6: Test results with 10 hidden units 42

Figure 6.7: Training results with 15 hidden units 43

Figure 6.8: Test results with 15 hidden units 43

Figure 6.9: Comparison of results 44

LIST OF TABLES

Table 3.1: Relation between tournament size and selection intensity 9

Table 3.2: Selection probability and fitness value 10

Chapter 1

INTRODUCTION

Among the set of search and optimization techniques, the development of Evolutionary Algorithms (EA) have been very important in the last decade. EAs are a set of modern meta heuristics used successfully in many applications with great complexity. EAs are used in many fields, since they are very useful for optimization and searching tasks. Therefore, they are well known algorithms in the field of artificial intelligence.

There are different approaches and methods of applying EAs for different problems. However, there is a main idea behind them, which is very similar to natural evolution in terms of generating populations or in other words reproduction. There are many benefits of using Evolutionary Computation (EC) techniques due to their implementation flexibility for various problems.

Most of the present implementations of EA come from any of these four basic types: Genetic Algorithms (GA), Evolutionary Programming (EP) and Evolutionary Strategies (ES) and Genetic Programming.

Genetic Algorithms imitate the process of natural evolution in terms of reproduction of population from a generation to generation. The main aim in genetic algorithms is to find the best solution in a given search space during the evolution process. The evolution process includes the most important activities for reproduction purposes, which are selection, crossover and mutation. Some of the most famous ones are as follows: Travelling Salesman Problem (TSP), Vehicle Routing Problem (VRP) and Scheduling Problems (SP), and Difficult Real-Valued Optimization Problems.

In this thesis, a genetic algorithm was used in order to implement a Radial Basis Function Neural Networks (RBFNN). In general, Radial Basis Functions are of the following types: the Gaussian RBF and a Multiquadric RBF. The calculation of the output by the Radial Basis Function network is done through the computation of local parameters from input to the output. RBFNN is used to solve modeling and classification problems, since it is fast in terms of learning speed and approximations. The RBFNN can be considered as a three-layer feed-forward neural network with a simple architecture. The type of activation function of RBFNN in this thesis is a Gaussian function.

The implementation of RBFNN using a GA is done in such a way that GA controls the parameters of RBFNN. These are hidden units, centers, weights and widths.

The organization of other chapters in this thesis is as follows. Chapter 2 presents the general description of data modeling. In Chapter 3, the genetic algorithms are explained along with different variants of selection, crossover and mutation operators. Chapter 4 presents the description and evolutionary design of RBFNN. Chapter 5 focuses on the proposed work. The experimental results along with discussions are found in Chapter 6. Finally, the conclusion and discussion of possible future work is presented in Chapter 7.

Chapter 2

THE DATA MODELING PROBLEM IN INDUSTRY

One of the most important technologies developed in industries is the pulp and paper manufacturing technology, which combines engineering with industrial training. In the pulp and paper manufacturing technology there are several factors to be considered for a good and profitable development, which are effective and organized factory systems that are people-oriented and economic. Engineers in this field focus their work in areas such as research, development and quality control. Moreover, they are responsible for the whole pulp and paper making process to ensure its quality. Another issue with the pulp and paper making process is that it breaks fast if recycling was done by improper addition of other materials such as water or other liquid and inaccurate procedure. It is the engineer's job to make sure that the whole process of the pulp and paper operation meets all requirements. There are several problems that technologists face with while implementing such operations. For example, the accurate measurements of products' quality and process performance are very important issues for manufacturing industries. In addition, this requires real-time for making necessary measurements for good control of products, which is usually a problem in most of the cases, since such results as process variability and unscheduled downtime may occur. One of the examples of such problems is the control of the black liquor concentration system in the pulp and paper manufacturing industries. Therefore, different models are proposed for implementing this kind of tasks using software systems. In this thesis, the radial basis functions neural network

(RBFNN) based model was designed to control the black liquor solid content (BLSC).

Wood chips are used for producing pulp during paper making process. Wood chips and “white liquor” are combined and cooked in a digester, which is called “black liquor”. Through evaporation, by heating with steam, the black liquor is concentrated up to the point when it is around 66% of solids, and then it is burned in the recovery boiler. After burning of the black liquor steam and a residue of molten chemicals are produced. Then addition of water and lime is performed for the residue of molten chemicals to be processed and turned back to fresh white liquor for future use. After other several stages such as evaporation, the black liquor solid content’s process ends. There are several important variables that the solid content of the black liquor depends on, i.e. pressures, temperatures, flow rates, etc [21].

The proposed RBFNN model, using regression analysis, makes accurate estimation of the BLSC that can lead to variability and cost reduction and a more stable operation with limited unscheduled shutdowns. Advantage of using RBFNN is that it uses established linear regression techniques, which allow reaching to the global minimum and fast convergence with specified hidden neurons and other RBFNN parameters.

The dataset used in this thesis is obtained from a Canadian pulp and paper company. The dataset contains historical sensor measurements over 10 years. There are a lot of outliers and missing values in the original dataset. However, the company provided us a cleaned dataset. That is obtained from the original one by removing outliers and interpolating the missing ones.

Starting in 19th century, the pulp and paper industry became widespread all over the Canada and was serving the needs of the people. As the technology was developed, the more there was a need for the paper industry. The Great Lakes-St Lawrence and the Maritimes were filled with new mills due to the need of the paper.

The cotton and linen rags were known for many centuries to be the mostly used resources for paper manufacture. Later in 1840s it was realized that the paper manufacture can be done using the vast forest wood. Ground wood, is used primarily for cheap papers, which is prepared by grinding the wood.

The modern papermaking began in 1864, when the first chemical wood-pulp mill was built at Windsor Mills. Preparation of chemical pulp is done from wood chips boiled under pressure with chemicals with usage of soda in order to leave mostly cellulose fiber. The wood pulp is washed, bleached, blended and then poured over a wire screen, after which a fine layer of fiber is produced [22].

The BLSC dataset used in this thesis contains 19 variables. Each sample contains 18 explanatory variables and one response variable. The data set used in this thesis is also clear with regards to distinguishing the explanatory variables from the response variable. This is due to the fact that 18 variables in each sample are the input variables, which are going to ‘explain’ the remaining one variable – response variable, which is the output.

Chapter 3

GENETIC ALGORITHMS

Genetic Algorithms (GAs) are search heuristics, which involve the process of natural evolution. In GAs, potential solutions are repeatedly generated in a particular number of times for optimization or search purposes. GAs are very useful to apply in a problems when very small amount of information is known. GAs are generally the adaptation of random search, which is used in optimization problems, whereas, there still exists a direction for search due to constraints. GAs can be applied for single objective optimization and for different problems with more than one objective function. Usually, GAs contain several components which are:

i. **Individual**: any possible solution generated by GA, which is also called a *chromosome*.
ii. **Population**: set of individuals present together.
iii. **Search Space**: all possible solutions to the problem.
iv. **Fitness**: the value assigned to an individual based on the quality of the individual.
v. **Objective Function**: a function that assigns fitness value to the individuals.
vi. **Parent**: member of a current generation selected for reproduction
vii. **Offspring**: generated child (individual) from parent(s) as a result of crossover or mutation, which is a member of the next generation.

viii. **Crossover**: using two individuals, interchange their genes with each other and thus generating two new individuals.

ix. **Mutation**: changing a random gene in an individual.

x. **Selection**: selecting individuals (parents) for creating the next generation.

xi. **Generation**: iteratively created new populations.

The following is a general workflow of a GA: first, initial population is generated randomly and the fitness value for each individual in the current population is computed and saved based on the objective function and constraints if exist. After that, the selection operator is applied based on the fitness values of individuals for further reproduction. The probability of an individual being selected is directly proportional to its the fitness value. As a result, those individuals possessing higher fitness values will have more probability to be selected than the other ones. Then, the application of crossover and mutation operators is committed on the selected individuals (parents) for reproduction and generating new population. This whole process is repeated for a number of particular generations, which is specified by the user. Figure 3.1 illustrates the GA process.

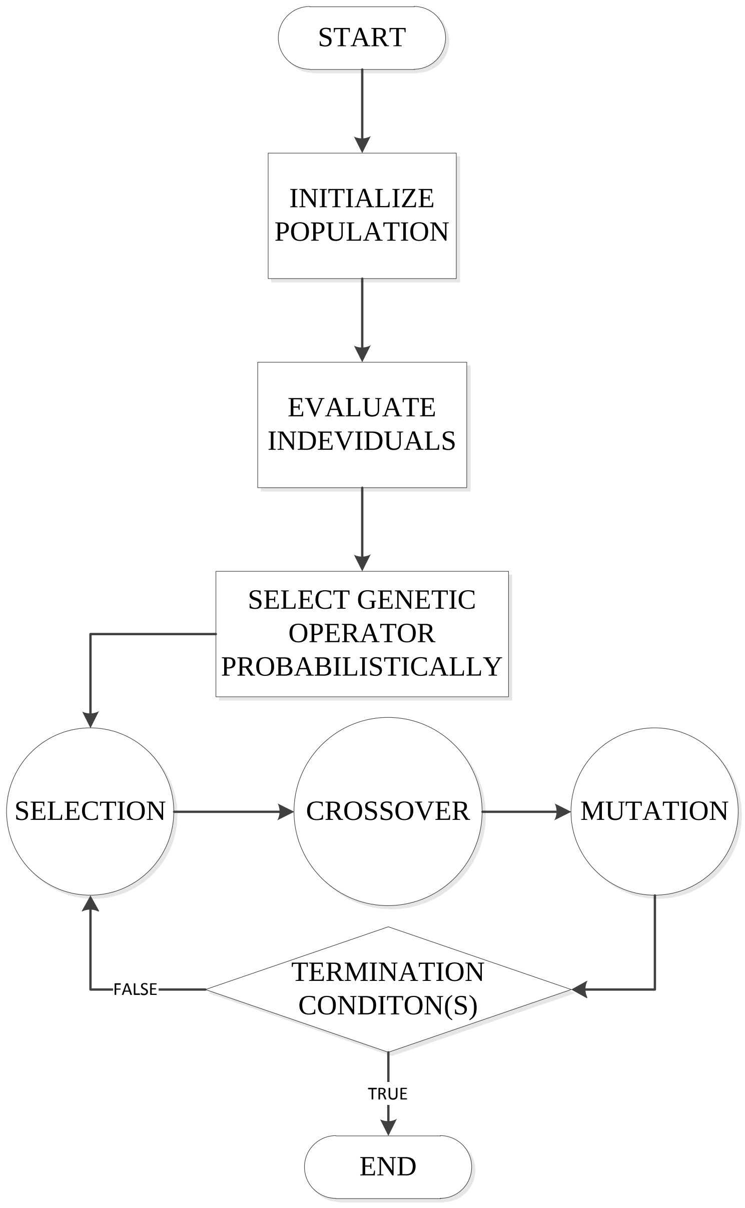


Figure 3.1: Genetic Algorithm Flowchart

3.1 Genetic Operators

In GAs, several operators are involved during the evolution process, which are *selection*, *crossover* and *mutation*. Each of these operators has different variants of implementation and carries different aims, which fulfill the evolution process. This section describes in detail each of these operators and their different approaches for implementation.

3.1.1 Selection

Selecting individuals for reproduction is one of the most important steps in evolution process, since they are the reason for getting better offspring and thus obtaining best results. Therefore, the main idea of selection operators is to make a decision and give preference when selecting among candidate individuals those, which are better than the other. There are many different approaches for selecting parents for reproduction, so some of them are mentioned below.

3.1.1.1 Tournament Selection

In tournament selection a number *Tour* of individuals is chosen randomly from the population and the best individual from this group is selected as parent. This process is repeated as often as individuals must be chosen. These selected parents produce uniform at random offspring. The parameter for tournament selection is the tournament size *Tour*. *Tour* takes values ranging from 2 to the number of individuals in the population. Table 3.1 shows implementation of tournament selection with *Tour* size equal to 7. In this table, there are 7 candidate parents that were randomly chosen from the population, and each has a fitness value, where less fitness indicates a better quality of the individual, since we consider minimizing the objective function. As a result from the given data in below table, parent 3 and parent 7 are the best among the all selected candidate parents.

Table 3.1: Relation between tournament size and selection intensity

Candidate Parent	1	2	3	4	5	6	7
Fitness (Quality)	23	11	5	32	17	28	9

3.1.1.2 Roulette Wheel Selection

Roulette Wheel selection is one of the traditional GA selection techniques. The Roulette Wheel selection method sums all the fitness's of all individuals and then it calculates the probability of selection for each individual. Then, an array is built containing cumulative probabilities of individuals. So, the probability of a chromosome (individual) to be selected is defined below:

Individual *i* is chosen according to the following selection probability: $\frac{f(i)}{\sum_i^N f(i)}$ [11],

where N is the number of individuals in the population.

Table 3.2: Selection probability and fitness value

Number of individual	1	2	3	4	5	6	7	8	9	10	11
Fitness Value	2.0	1.8	1.6	1.4	1.2	1.0	0.8	0.6	0.4	0.2	0.0
Selection Probability	0.18	0.16	0.15	0.13	0.11	0.09	0.07	0.06	0.03	0.02	0.0

3.1.2 Crossover

Crossover is a recombination of two different chromosomes and generating two better chromosomes. In crossover operation, recombination process creates different individuals in the next generations by combining genes from two individuals of the previous generation. Crossover operator exchanges the genes of current selected chromosomes and generates two new offspring as a result. Some of the different types of crossover operators are described in the next subsections.

3.1.2.1 Crossover Probability

In the GAs, crossover operation is the main method of reproduction. Therefore, crossover probability indicates how often crossover will be performed. If crossover probability is high, it will destroy the good individuals, because of extreme usage of crossover operator. Otherwise, offspring will be exact copies of parents. Moreover, if crossover probability is 100%, then no offspring will be generated by mutation, which is explained in the next section.

3.1.2.2 One-Point Crossover

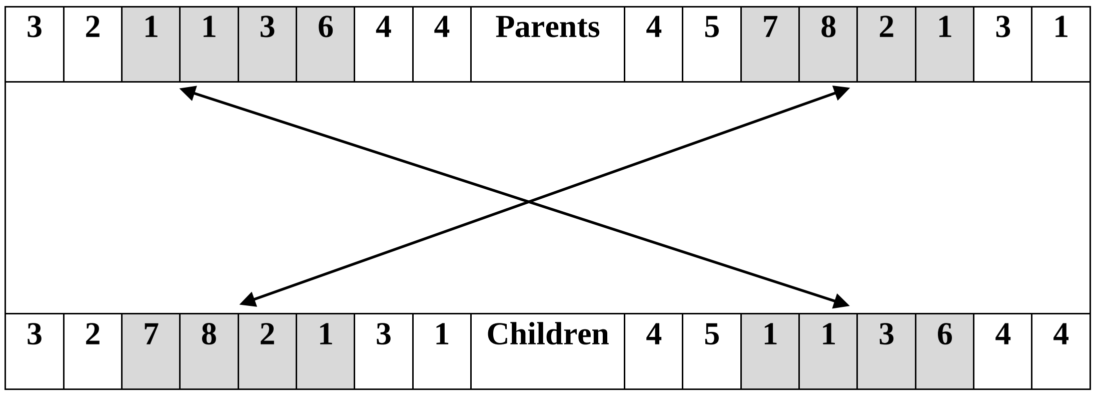
One-point crossover is one of the simple types of crossover operators, though it is a base for many other crossover types. In one-point crossover operator, a common crossover point in the parent chromosome is selected and then the swapping of the corresponding sub trees is performed between two individuals. There are two steps in implementation of one-point crossover operation: a point is generated randomly between 1 and the size of the chromosome, and then the first part of parent 1 is concatenated with the second part of parent 2 and given to child 1. The same is implemented for child 2, the first part of parent 2 is concatenated with the second part of parent 1. Figure 3.2 illustrates the implementation of one-point crossover.

Parents:	3	2	1	1	3	6	4	4
	4	5	7	8	2	1	3	1
					Crossover Point			
Children:	3	2	1	1	2	1	3	1
	4	5	7	8	3	6	4	4

Figure 3.2: One - Point Crossover

3.1.2.3 Two-Point Crossover

In addition to single point crossover, many different crossover types have been introduced by involving more than one cut point, there is an advantage of having more crossover points, since the problem space may be searched in deep. In two-point crossover, two crossover points are chosen and the genes between these points are exchanged between two mated parents as shown in Figure 3.3.



3	**2**	**1**	**1**	**3**	**6**	**4**	**4**	**Parents**	**4**	**5**	**7**	**8**	**2**	**1**	**3**	**1**
3	**2**	**7**	**8**	**2**	**1**	**3**	**1**	**Children**	**4**	**5**	**1**	**1**	**3**	**6**	**4**	**4**

Figure 3.3: Two–Point Crossover

3.1.2.4 Uniform Crossover

Uniform crossover is a different type of crossover, which uses an extra temporary parameter called mask, which is created from bits randomly. Therefore, the genes between the chromosomes are distributed to new offspring according to the mask. If mask contains bit 1, then the gene is taken from parent 1 and given to child 1, otherwise it is given to child 2 and place in the same location. The same is implemented for parent 2. The Uniform crossover is illustrated in Figure 3.4.

					Mask:	1	0	1	1	0					
Parent1:	0	1	0	1	1	Parent2:					1	1	1	0	0
Child1:	0	1	0	1	0	Child2:					1	1	1	0	1

Figure 3.4: Uniform Crossover

3.1.3 Mutation

Another important genetic operator is the mutation operator, which is also very important for reproduction process. The difference between mutation and crossover operations is that mutation does not involve Exchange of information between chromosomes, since it is applied on a single individual. So mutation is done within the same chromosome without introducing new genes from other individuals. Mutation enables algorithm to escape local minimum. There are various types of mutation operators among which some are mentioned below.

3.1.3.1 Mutation Probability

The mutation probability is the important parameter of GAs, since as in the case of crossover probability it plays a vital role, which decides whether mutation operation will be performed much. When the mutation probability is very high or 100%, then there is a risk of getting stuck at local minimum or local maximum. However, it should not be very low either, since the mutation is important to implement from time to time during the evolution process in order to compensate the loss of diversity during crossover operations.

3.1.3.2 Swap Mutation

One of the most used and simple mutation types is a swap mutation, since the operation, which is done during swap is just to replace the positions of two genes in a chromosome. As a result of swapping the order in a chromosome is destroyed, which improves the solution. Swap mutation is shown in Figure 3.5.

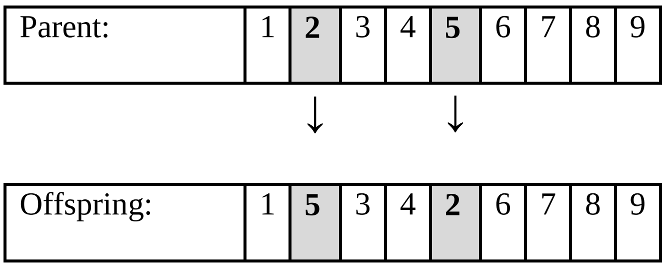


Figure 3.5: Swap Mutation

3.1.3.3 Point Mutation

In point mutation a random point in a chromosome is chosen and its element is replaced by the complement of it or given another value, which results in a different chromosome. Point mutation is shown in Figure 3.6.

Parent	0	1	0	1	1	0	1	1	0
				↓					
Offspring	0	1	0	0	1	0	1	1	0

Figure 3.6: Point Mutation

3.1.4 Reordering

Reordering is also a very effective mutation type, since in most of the problems it gives good results from mutation operator. Reordering is a simple operation, which selects two points within the chromosome and then reorders all the genes in that interval. Reordering is shown in Figure 3.7.

Before	0	1	0	1	1	0	1	0	1	1	0	1	0	0
After	0	1	0	1	0	1	0	1	1	1	0	1	0	0

Figure 3.7: Mutation Operator

Chapter 4

RADIAL BASIS FUNCTION NEURAL NETWORK

A radial basis function (RBF) network is a special type of neural network in which the activation function is a radial basis, which is a real-valued function, where its value depends only on the distance from the origin. Examples of some of the applications are as follows: interpolation, system identification, approximation, curve fitting, modeling and classification problems. Because of compact topology and faster training speed the radial basis function neural network (RBFNN) is different from other neural Networks [16]. Moreover, another thing that makes RBF neural networks different from others is that in most of the cases it reaches the global minimum of error surface during training [17]. As a result, RBFNN have attracted considerable attention in the field of science and engineering. RBFNN's main advantage is that after the number of hidden neurons, centers and radii have been set the optimal biases and weights, it can be efficiently computed for a certain set of desired output [14].

RBFNN is a type of feed forward neural network consisting of three layers – the input layer, the hidden layer and the output layer (see Figure 4.1 for its structure). The first layer (input layer) is responsible for accepting inputs, i.e. data sample and pass them to the each of the hidden units in the second layer. The hidden units in the second layer accept these inputs and using the activation function defined by the user process them independently from each other and signal them to the output layer. The output layer, which is the RBF real valued output using summation mechanism sums all the hidden unit results by multiplying each of them by the relative weights. Thus this process is done so many times as it is required in order to design a model.

When using RBFNN, the hidden unit activation function must be specified such as the number of processing units. Moreover, in order to find the parameters, the particular rules for modeling a given task and a training algorithm should also be specified. The purpose of network training is to find RBFNN weights. If there exists a set of input-output pairs, called training set, the network parameters are optimized in order to fit the network outputs to the given inputs. Mean squared error is used for the cost function, which is used to evaluate the fitness. The RBF network can be used with data, which is similar in structure to the data that was used for training the model [12].

Determination of the number of neurons in the hidden layer in RBFNN is very important. This is because all the neurons have radial basis function, which has as many dimensions as the number of inputs. A few number of hidden neurons in the hidden layer may cause the RBFNN receive the data incorrectly. However, usage of many neurons in RBF network may cause overlearning and thus bad prediction. Determination of center locations is also a very important issue in RBF, since the positions of centers have a considerable affect on the performance. All neurons have activation functions in the hidden layer and the Gaussian function is mainly used as an activation function. Then, since the weights are also another important factor in RBF network, they also must have a range of values to be selected for the weights, which are used between hidden and the output layers. The spread (radius) of the RBF function is generally different for different samples, and the bias values which are also added as they have a small affect. The following data model in Figure 4.1 represents the RBFNN structure with all three layers.

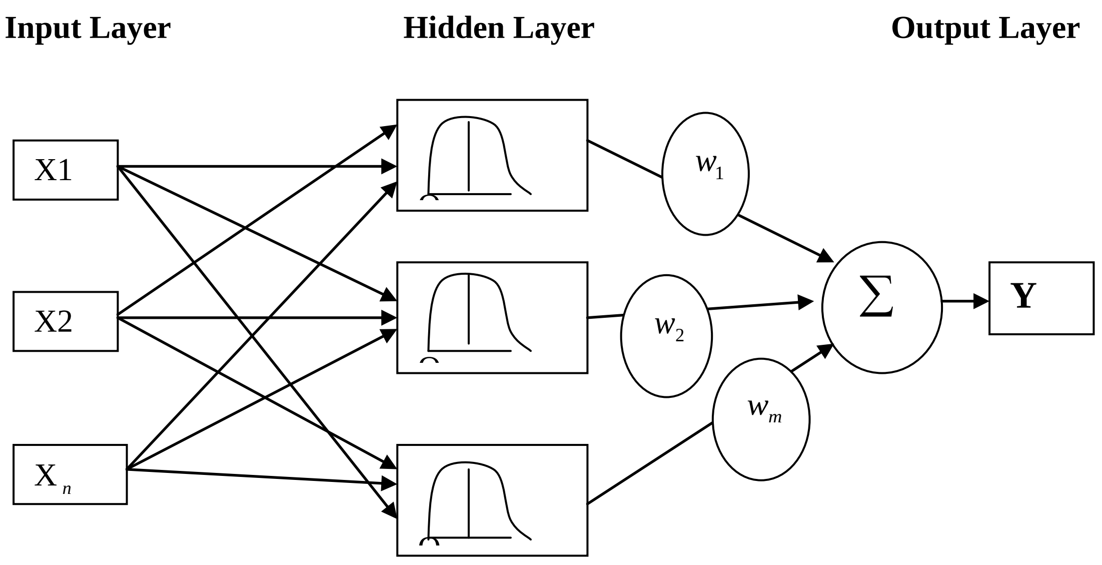


Figure 4.1: RBFNN Structure

All inputs are connected to neurons without weights (w) but neurons are connected to output through weights. Every neuron has one weight and output is calculated as follows:

y(x)= $\sum_{1}^{N} w_i \Phi(\|x - x_i\|)$, where *y(x)* represents the sum of radial basis functions, *w_i* is some weight, *x* is the sample input and *x_i* is the center of the input.

4.1 General Formula of RBF

The most general formula for any RBF is

h(x) = $\phi((x-c)^{\mathrm{T}} R^{-1}(x-c))$,

where ϕ is the function used (Gaussian, Multi-quadratic, Cubic, etc.), c is the center and R is the metric. The term $(x-c)^{\mathrm{T}} R^{-1}(x-c)$ is the distance between the input x and the center c in the metric defined by R. There are several common types of functions used and Euclidean is often used as metric. In this case, R= r^2I for some scalar radius r and the above equation simplifies to:

h(x) = $\phi(\frac{(x-c)^{\mathrm{T}}(x-c)}{r^2})$ [15].

4.2 Types of RBFNN

There are many basis functions used in RBFNN. However, in this thesis the Gaussian function was used, whose definition and description is given below:

- **Gaussian Function:**

The Gaussian function is the most preferred activation function, which has a spread parameter that controls the behavior of the function. During the training process of RBF the spread parameter is also optimized for each hidden neuron. The following is the Gaussian function:

$$\Phi(r) = \exp(\frac{\left\| x - c_j \right\|^2}{2\sigma_j^{\ 2}})$$

Where width parameter σ >0 and j = 1, 2… ***m***, x is an input vector, $\Phi(r)$ is the output of hidden layer nodes, c_j is the center of Gaussian function, $\left\| x - c_j \right\|$ is the Euclidean distance between x and the center c_j, σ is the width (radius) of Gaussian function, ***m*** is the number of hidden nodes.

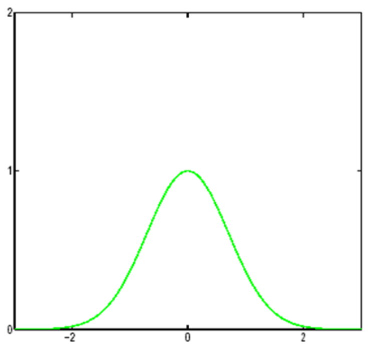


Figure 4.2: Gaussian RBF

Above figure illustrates the Gaussian RBF with center $c = 0$ and radius $r = 1$, which monotonically decreases with distance from the center.

4.3 How RBFNN Works

An RBF network has hidden neurons, which have as many dimensions as there are predictor variables or also may be called sample attributes. The radial basis function is applied to the distance for calculating the weight for each neuron, where distance is the Euclidean distance computed from the point being evaluated to the center of each neuron. The radial basis function is named as such, because the radius distance is the argument of the function.

The further a neuron is from the point being evaluated, the less influence it has.

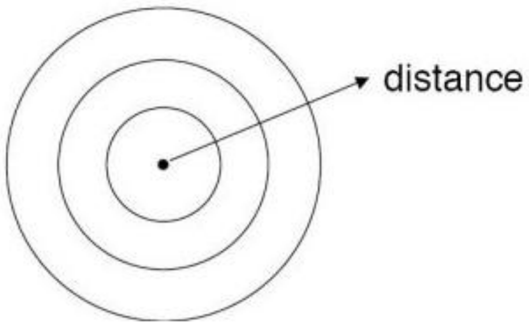


Figure 4.3: Neuron Influence

4.4 Training Algorithms of RBFNN

A training algorithm's purpose is to find optimal parameters for the specified network structure and dataset. There are two categories of training algorithms: supervised and unsupervised. However, in RBFNN, supervised methods are used more often. There are data samples in supervised method with the same number of attributes, called training set, and in addition to this the corresponding network outputs are also known.

4.5 Least Squares

In supervised learning the least squares principle becomes an easy optimization task. If the model is

$$f(x) = \sum_{j=1}^{m} w_j h_j(x)$$

and the training set is $\{x_i, \hat{y}_i\}_{i=1}^{p}$, then the least squares implementation is to minimize the sum-squared-error (S)

$$S = \sum_{i=1}^{p} (\hat{y}_i - f(x_i))^2 .$$

When a weight value is added to the sum- squared-error, then the following cost function is minimized as follows:

$$C = \sum_{i=1}^{p} (\hat{y}_i - f(x_i))^2 + \sum_{j=1}^{m} \lambda_j w_j^{\ 2} \quad ,$$

where the $\{\lambda_j\}_{j=1}^{m}$ are regularization parameters [15].

4.6 RBF Properties

The Gaussian Function is formed so that φ (*r*) →0 as r →∞. Moreover, when considering in detail we may notice that the Linear Function φ (*r*) = *r* = $\left\| x - x^{p} \right\|$ is still non-linear in terms of ***x***. In one dimension, this comes to a linear interpolating function, which in return represents a simple form of interpolation function.

4.7 RBFNN Improvements

Although below mentioned techniques will make analyzing and optimizing the network much more difficult, RBFNN basic structure can be improved in several ways as follows [18]:

- The number ***M*** of hidden units need not equal the number ***N*** of training data samples. Thus, usually it is better to have ***M*** much less than ***N***.
- The centers of the hidden units do not need to be defined as the training data input samples. They can be determined by a training algorithm instead.
- The basis functions need not all have the same width parameter σ, since a training algorithm can also determine them.
- We can introduce bias parameters into the linear sum of activations at the output layer. These will compensate for the difference between the average value over the data set of the basis function activations and the corresponding average value of the targets.

Chapter 5

THE DEVELOPED MODEL

This chapter discusses the proposed work and the developed model using RBFNN techniques and genetic algorithms. The developed model consists of implementation of RBFNN using GA approach. This thesis is concentrated on a single objective function, which is the Root Mean Square Error (RMSE), and therefore the objective function was minimized. There are two vectors involved in the objective function, which are the found vector output by the RBFNN and the target vector output given in the dataset. Applying the RMSE on these vectors we get the difference between the target output and the found output by the algorithm, thus minimizing this difference, such that the error. Moreover, the k-fold implementation was adopted in this thesis for better results and explained in detail in this chapter. Another essential part of the developed model in this thesis is the integration of local search into the algorithm. By introducing local search into the algorithm, the model could produce better results and could minimize error more. Normalization of the dataset also took place in the developed model, since the purpose of normalization was to eliminate the effects of certain sample’s large influences. In the development of this model the method of evolutionary computations was successfully adopted for finding the optimal parameters of RBFNN. Therefore, GA was used to find optimal parameters for the RBFNN. In other words, GA was responsible for generating hidden units, centers, widths and weights of RBFNN. In order to implement the combination of GA with RBFNN, various methods have been applied to different parts of the whole model. The main target was to improve and find optimal parameters of RBFNN in order to get minimum error when comparing found output with the target output in an attempt that the difference between them would be ensured to be very close to zero.

When using GA for optimizing the parameters of the hidden layer (the centers μ and widths σ) of the RBFNN model, there is also one more factor that is important for GA to take care of: number of hidden neurons, and centers and radii related to them. The number of hidden units in RBFNN is very important as it has a big influence on the results obtained. Very few hidden neurons may not achieve good results because of lack of knowledge. However, using too many hidden neurons may result in overloading the model with extra knowledge and information. Therefore, finding golden middle for the number of hidden neurons to be used is very important issue in RBFNN.

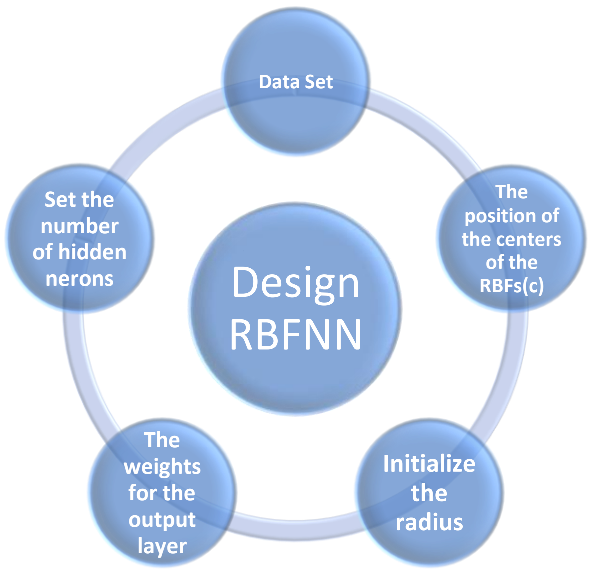

Data Set
The position of the centers of the RBFs(c)
Initialize the radius
The weights for the output layer
Set the number of hidden nerons
Design RBFNN

Figure 5.1: RBFNN Design Procedure

5.1 Genetic Operators

In this algorithm genetic operators that are used are different for control genes (hidden units) and other parameter genes (center, weights and width). For control genes, single point crossover is used and arithmetic crossover is used for other parameter genes. As a purpose of selection operator Roulette Wheel Selection was used.

5.1.1 Constraints

When applying single point crossover for control genes, it may be ended up with no hidden neurons after crossover is done as illustrated in the following example, where cross point is between 3 and 4:

Parent1:	0	1	1	0	0	Parent2:	0	0	0	1	1
Offspring1:	0	0	0	0	0	Offspring2:	0	1	1	1	1

As shown in the above example, there are 5 hidden neurons in total. However, not all hidden neurons are active since 1 indicates that the hidden neuron is active and 0 indicates that it is inactive. Thus, after applying single point crossover to two parents, one offspring is obtained, which has all of its hidden neurons inactive, i.e. offspring 1. Every chromosome must contain at least one hidden neuron to be active. Therefore, when applying single point crossover, the constraint is checked for at least one active hidden neuron to present in each chromosome and that is done by randomly selecting one of the hidden neurons and making it active, i.e. making its 0 value 1.

For the rest of the parameters of RBFNN, this constraint is not checked since all values are between 0 and 1. Moreover, the crossover that is applied for other parameters is arithmetic crossover, which is explored in depth in the following section.

5.1.2 Arithmetic Crossover

In arithmetic crossover, one Formula is used assuming that two parents p1 and p2, the two new offspring c1 and c2 are:

$c1 = k\,(p2) + (1-k)\,p1 \xrightarrow{\hspace{4cm}}$ Offspring 1

$c2 = k\,(p1) + (1-k)\,p2 \xrightarrow{\hspace{4cm}}$ Offspring 2

while *k* is a random number between 0 and 1.

5.2 Mutation

The mutation operation used for control genes and parameter genes are different as in the case of crossover operations. In the case of control genes, the bits were changed from ‘1’ to ‘0’ or from ‘0’ to ‘1’.For parameter genes, uniform mutation method was used. Assuming that K is the individual selected according to random probability to mutate, then the result is:

$$K^{I} = P_{min} + r \times (P_{max} - P_{min})$$

P_{max} and P_{min} are the maximum and minimum values of K respectively.

5.3 Chromosome Representation and Data Structure

In this algorithm, a chromosome is represented as follows. If we consider hidden units of size *m* and number of inputs in dataset *n*, then in the chromosome the first *m* elements are the control genes (hidden units). Next, from m+1 to n*m a chromosome elements contain the centers, so the total number of centers is equal to the number of hidden units, whereas the length of each center is equal to the number of inputs in the dataset, i.e. the sample size. Therefore, each hidden unit contains only one center, but with the length equal to the number of inputs in the dataset. After positions of centers, a chromosome contains the weights of size *m*, because each hidden unit requires one weight and in addition to weights a bias is placed at the end of all weights. The last *m* positions, also can be named as genes in a chromosome, are occupied by the widths. When the chromosome is initialized, both weights and widths are randomly created between -1 and 1. The figure 5.2 illustrates the representation of chromosome.

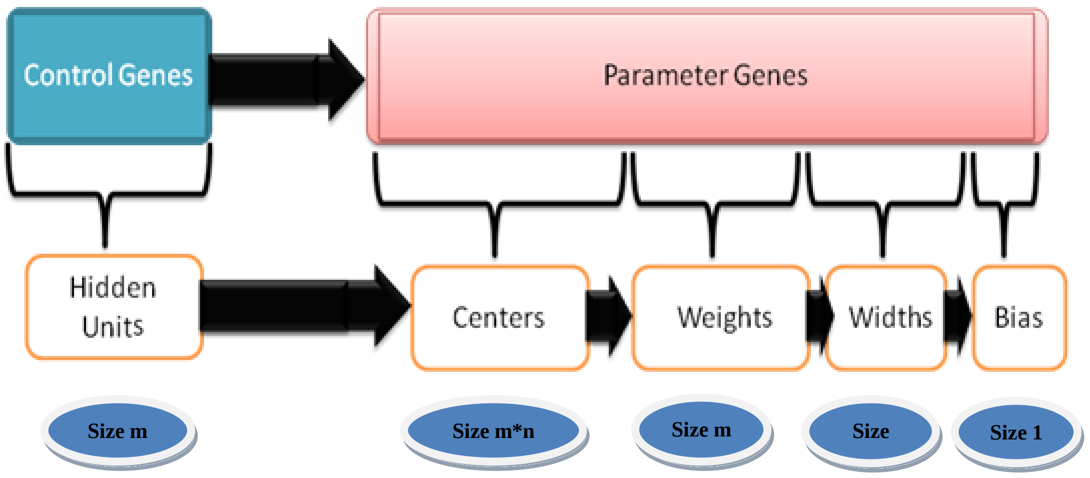


Figure 5.2: Chromosome Representation

Where control genes are our hidden nodes, initialization of hidden nodes is done by generating 0 or 1 randomly and assigning it to hidden unit, where 0 stands for inactive hidden unit and the number 1 means the hidden unit is active. As shown in Figure 5.2 above, the length of chromosome depends on the total number of hidden units. Thus, even if we have 1 active hidden unit length of our chromosome does not become shorter, which means that search space remains big all the time independently on the number of hidden units used. Although search complexity remains the same for any number of active hidden units, the number of active hidden units plays a big role in getting good results. Therefore, hidden nodes' size is important, as it will be discussed in the experimental results chapter.

Centers are the other important genes in our chromosome, which are initialized by using predefined MATLAB function called ***fcm***, which has two parameters. These are our dataset samples and number of centers. When initialized, centers are randomly selected from the dataset. After that, genetic operators are used on every generation in order to get better results. So, ***fcm*** MATLAB function is used only once when centers are initialized.

Weights are important parameters in order to find good solutions. We use weights for all hidden nodes. Every hidden node has one weight and all weights must be greater than -1 and smaller than 1. In the same way, genetic operators are applied to the weights with constraints that the lower and upper bounds of weights are -1 and 1 respectively.

Below is the main algorithm structure used in this thesis.

- Load training, validation and test sets.
- Create initial population. Evaluate and assign fitness to every individual.
- Instantiate operators and stop condition testers.
- Instantiate the evolutionary algorithm with the requıred *RBFNN* parameters.
- If stop condition is not reached, do the following:
    a) Select individuals (parents) from current population.
    b) Apply operators to these parents; evaluate and assign them a fitness.
    c) Combine newly generated population with the old one and apply elitism.
- Compare generated output by training the (*RBFNN*) algorithm and the target output.
- Use the test data set to obtain the generalization power of best solution found by the (*RBFNN*) algorithm.

Figure 5.3: General Skeleton of RBFNN Combining with GA

5.4 Implemented Objective Function

The objective function in this thesis is the Root Mean Square Error, which uses the output found by RBFNN and the target output given in the dataset. So, it calculates the difference between forecast and corresponding observed values are each squared and then averaged over the sample. Finally, the square root of the average is taken. Since the errors are squared before they are averaged. The following is the RMSE Formula:

$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - t_i)^2}{n}}$, where y_i is the found output by RBF, t_i is the target output and n is the number of samples in the dataset.

In this algorithm, three parameters were used in order to calculate the Root mean square error. These are new output, target output and sample size. When the new output is calculated, RBFNN is used.

There is a function that is used to calculate the output in Radial Basis Functions (RBF). This function is given by:

$$y_i(x) = \sum_{j=1}^{n} w_{ij}\phi(x - x_j)$$

where y_i is the output, $\phi\ (x - x_j)$ is the activity of the hidden node *j*, with a RBF function centered on the vector x_j, x is the target input vector and w_{ij} are the hidden layer weights from the RBF nodes. However, in this thesis a different function for the activity of the hidden unit was used, which is a Gaussian Radial Basis Function. The following is the Gaussian function:

$$\phi(x - x_j) = \exp\left(-\frac{\sum_{k=1}^{K}(x_k - c_{jk})^2}{2\sigma^2}\right)$$

where x_k is a sample from the dataset inputs, *K* is the sample size (or length), c is the centers and σ is the width of the Gaussian and the Euclidean distance is used for calculating the distance between input samples and centers. Lastly, the output of RBFNN is found which is the summation of all active hidden unit results with addition of bias value.

Thus, when the output is found for RBFNN, the error is calculated using RMSE formula in order to get chromosome’s fitness value. Based on fitness value the best chromosome is chosen. Less fitness value is better when chromosome is considered since it is aimed at minimizing the error.

5.5 Cross Validation

In this algorithm there are two phases – training and testing phases. Training phase is the first part of the whole process in the developed algorithm that is the most important one. It is responsible for training the algorithm and generating the model that will be used for testing. In order to implement training part of the algorithm different methods were attempted and finally k-fold method was decided to be used as it promises improvement and effective solution to the given problem.

5.5.1 RBFNN Parameters

In training part of the algorithm, all the parameters of RBFNN are affected and modified through training process in order to achieve the best results when comparing obtained results with the actual data output, i.e. target output. Therefore, all RBFNN parameters are modified so that the error is minimized in the best possible way. Among the most important parameters that are affected by the training process are the weights, centers and the widths. These are the most critical parameters that change behavior of a whole developed model. The training process modifies hidden units also, whereas they do not affect the process as much as other parameters, since hidden units mainly affect the algorithm model by the number of active hidden units, not their positions.

For all of these parameters in training process different values with different intervals were attempted in order to find out the best values for them. For example, weights were attempted to be between 0 and 1, and also between -1 and 1. As a result, interval of -1 and 1 was better for weights than between 0 and 1, since better results could be achieved. Since the centers were initially generated by the predefined MATLAB function *fcm*, their interval could not be set or modified to examine the results for different intervals.

5.5.2 K-Fold Implementation

Cross validation technique was achieved by using the *k-fold* cross validation method for implementation of training process and testing the developed model. General idea behind k-fold is that the whole dataset is divided into several blocks and after that one of the partitions is used for testing and the rest, i.e. *k*-1 is used for training process. This is repeated *k* times and every time the partition for testing is changing, thus attempting all partitions for testing. For instance, if *k* is equal to 10, then say partition 1 is used for testing and partitions from 2-10 are used for training. Therefore, in the next round for testing partition 2 will be used and for training all partitions other than the second one. The following Figure 5.4 illustrates *k-fold* cross validation technique.

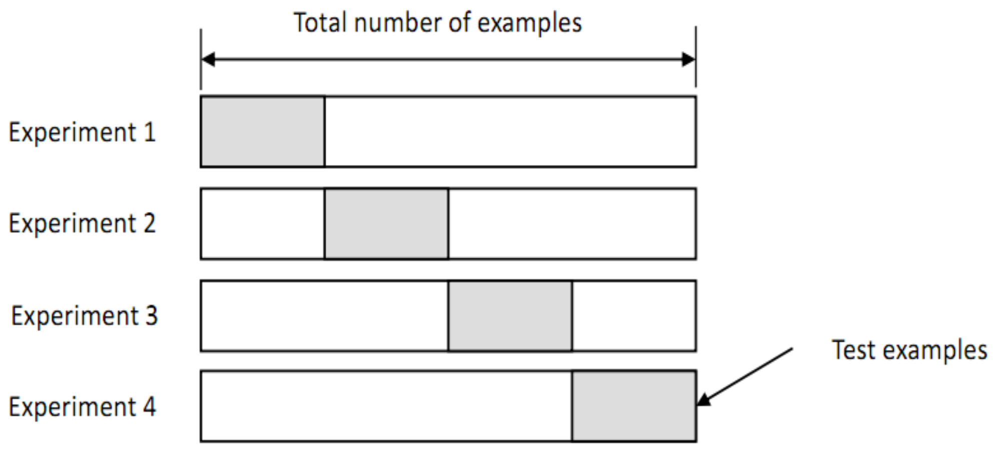


Figure 5.4: K-fold Cross Validation

In general, in *k-fold* cross validation *k* is given 10. However, in this algorithm *k* is decided to have a value of 5 since the datasets, which were used in this thesis for training and testing the algorithm did not contain number of samples more than 1500.

5.6 Local Search

In this GA that was combined with RBFNN a local search technique was also adopted in order to find better results and to make the study more optimized. Local search techniques are widely used in evolutionary optimization algorithm and hard computational problems and thus hybridizing them and improving their performance and getting best results faster. The advantage of using local search is that it enables algorithm to get the best results among the given candidate solutions, i.e. search space, since for finding the optimal solution a global search may not be enough. Pattern search was decided to implement the role of local search in this thesis.

5.6.1 Pattern Search

In order to implement local search in developed algorithm a predefined MATLAB function called *pattern search* was used, which finds the minimum of a function using pattern search. Let us consider in more detail the following fragment of MATLAB code: *x = patternsearch(@fun, x0),* where it returns the local minimum and assigns it to *x*. The local minimum is found according to the MATLAB function, *fun*, that computes the values of the objective function *f(x)*. The second parameter*x0* is an initial point for the pattern search algorithm. The syntax of above fragment of code is using *@fun,* so function *patternsearch* accepts the objective function as a function handle of the form *@fun* and this function *fun* accepts an input, which is vector, and returns a scalar function value.

In the developed algorithm, a more complex form of this function was implemented, i.e. it had more parameters and *options* along with those parameters. The following fragment of code demonstrates implementation of *patternsearch* function used with more parameters and options:

```
[X,E] = patternsearch(@obj,chromosome,[],[],[],[],LB,UB,[],options);
```

In this case, *patternsearch* returns two parameters, where *X* is the chromosome with optimized RBFNN parameters and *E* is the error. The first parameter, as it was discussed above, *@obj* is the objective function to be evaluated while *patternsearch* is searching for the optimal solution. The second parameter *chromosome* is the initial chromosome passed to the *patternsearch* function by the genetic algorithm, which contains the RBFNN parameters. The next parameters *LB* and *UB* are passed to this function for lower and upper bound of the RBFNN parameters contained in the chromosome respectively. All other parameters that are passed as empty brackets [], indicate that the default parameters should take place since we are not interested in them.

The last parameter, as shown in above fragment of code, is *options*. This is an extra parameter that enables us to change one of many default settings of *patternsearch* function. So in this study, *maximum function evaluations* parameter was changed using *options* as follows:

```
options = psoptimset('MaxFunEvals',MaxEvals);
```

The *psoptimset* is another MATLAB function that allows performing required operation. Thus, *psoptimset* function sets *MaxFunEvals* parameter to the variable called *MaxEvals*. The default value of *MaxFunEvals* is *2000*numberOfVariables,* where *numberOfVariables* is the sample size from the dataset. The dataset used in these experimental results had 18 inputs, i.e. the sample size is 18. What remained was to calculate the value for *numberOfVariables*. Several different values were used for the number of hidden units. Assuming that there are 5 hidden units in RBFNN, then the length of our chromosome becomes as follows (number of genes/variables): 5 hidden units, 5*18 centers, 5 weights, 1 bias and 5 widths. Therefore, in total there are 106 variables in a chromosome. Thus, the value for *numberOfVariables* is 2000*106 = 212000. However, this value was incremented a bit more and decided to initialize *MaxEvals* variable with 250000.

5.7 Normalization

In order to achieve better results and improve the developed model, normalization was used in proposed algorithm. The whole dataset was normalized before in both training and testing parts of the algorithm. The normalization formula that was used in the algorithm is as follows:

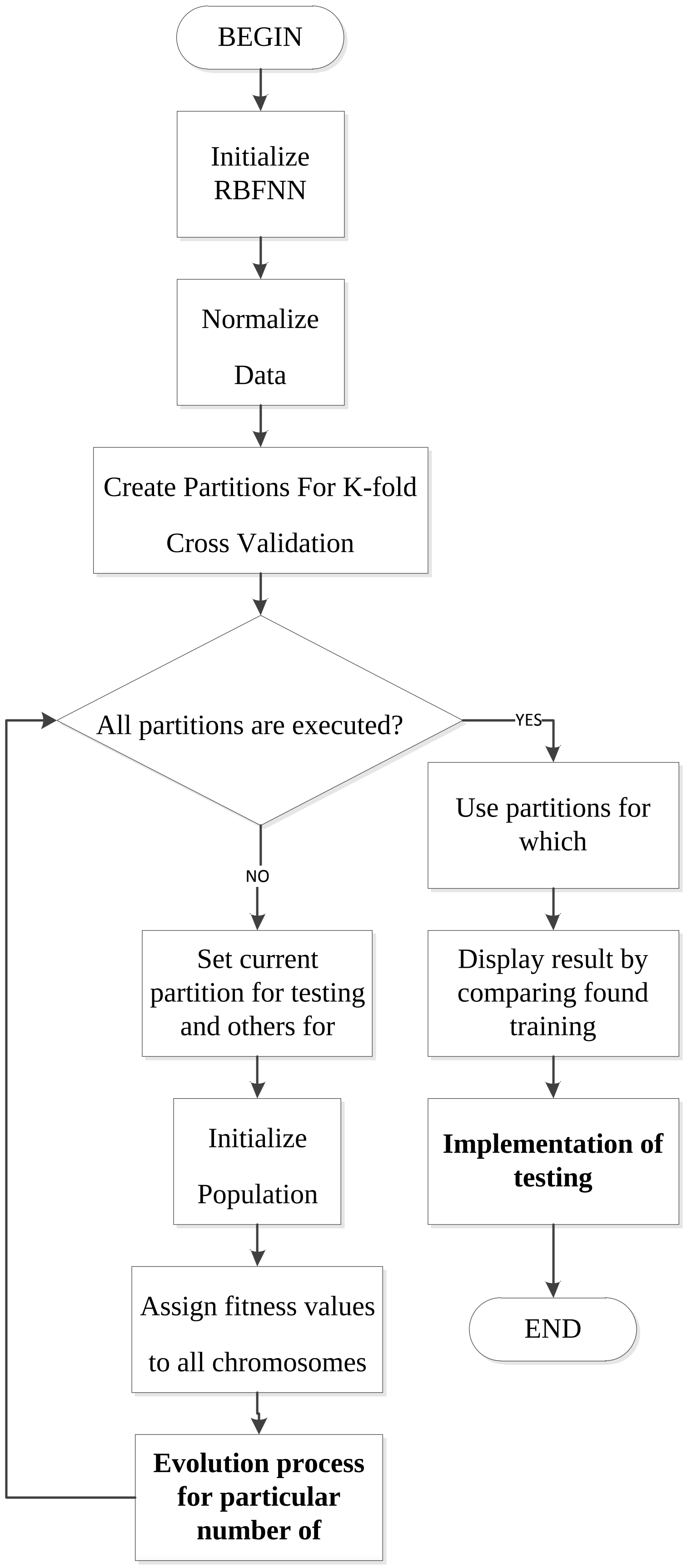
$$X_{Norm} = \frac{X - X_{\min} + d_1}{X_{\max} - X_{\min} + d_2}$$

Where *X* is the original dataset, d_1 = 0.05 and d_2 =0.1

5.8 Algorithm Description

The developed algorithm consists of several modules, which construct a powerful model. Some of these are evolution process and a local search.

The following flowcharts describe the developed algorithm in detail. All of these flowcharts are interconnected and thus implement the model. Figure 5.5 is the representation of algorithm main flowchart, which includes local search and testing algorithm in it. Figure 5.6 is the illustration of the evolution process, which is contained in the main algorithm. And finally, Figure 5.7 is the demonstration of the implementation of the testing part of the main algorithm.



BEGIN
Initialize RBFNN
Normalize Data
Create Partitions For K-fold Cross Validation
All partitions are executed?
YES
NO
Use partitions for which
Set current partition for testing and others for
Display result by comparing found training
Initialize Population
Implementation of testing
Assign fitness values to all chromosomes
END
Evolution process for particular number of

Figure 5.5: Algorithm Main Flowchart

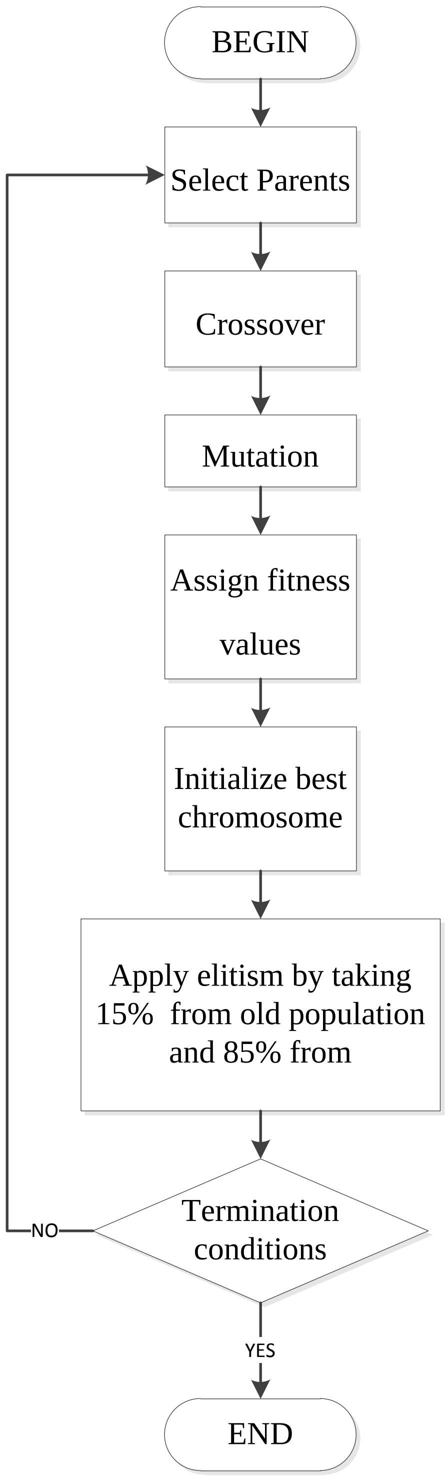


Figure 5.6: Evolution Process

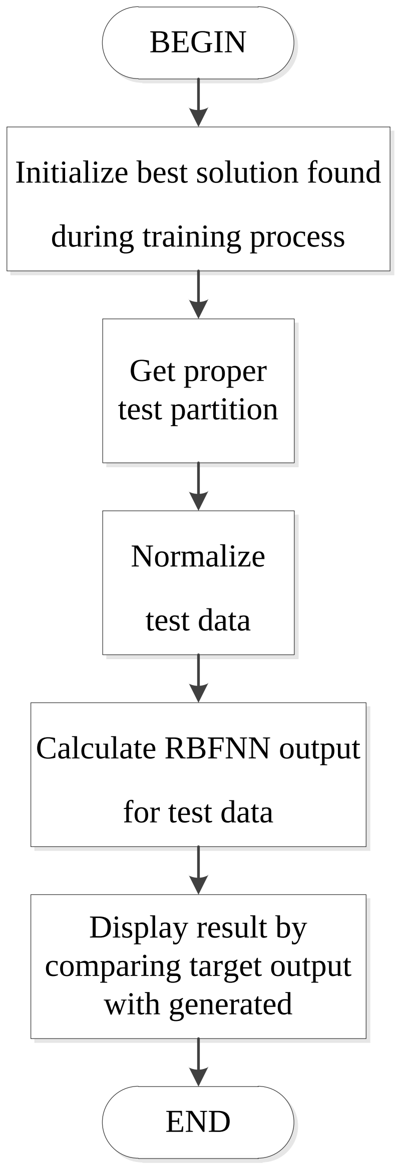

BEGIN
Initialize best solution found during training process
Get proper test partition
Normalize test data
Calculate RBFNN output for test data
Display result by comparing target output with generated
END

Figure 5.7: Implementation of Test Part

Chapter 6

EXPERIMENTAL RESULTS

Several experimental results were obtained through different values of the number of hidden units, since other parameters, such as weight or width interval was decided to be fixed for all experiments. Moreover, the population size was also decided to be fixed and has value of 100, though generally population size affects the results. In order to see how the number of hidden units affects the results, number of iterations was also fixed. However, the only parameter in the local search, i.e. pattern search, was changed, which is *maximum function evaluations,* for which a value of 250000 was assigned.

All experimental results were conducted on the same BLSC dataset, which consists of 1452 observations or also known as samples and the number of variables of each sample is 18 and 19th variable is the target output of the sample.

The following Figure 6.1 is the training data result with 3 hidden units.

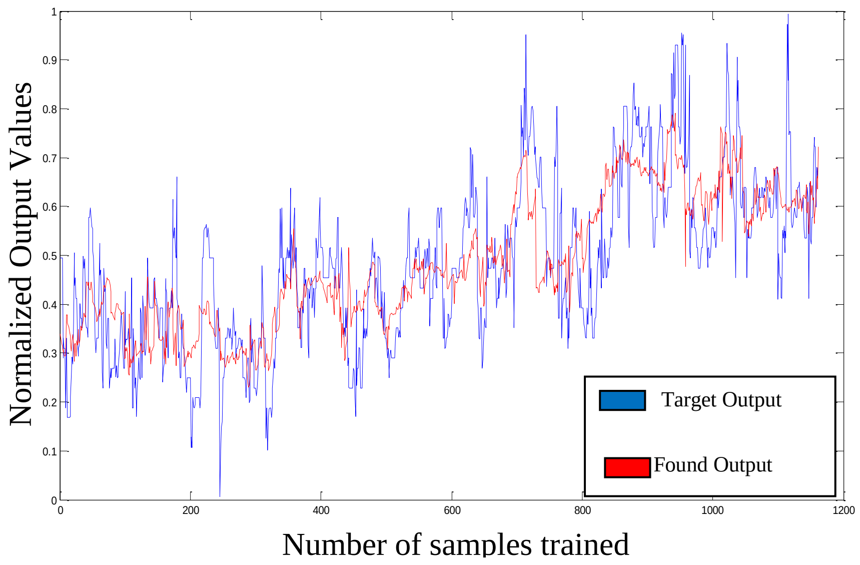


Figure 6.1: Training results with 3 hidden units

The error could be minimized down to 0.0956 for the above figure with 3 hidden units.

The following Figure 6.2 demonstrates the test data results for the same data set with the same number of hidden units.

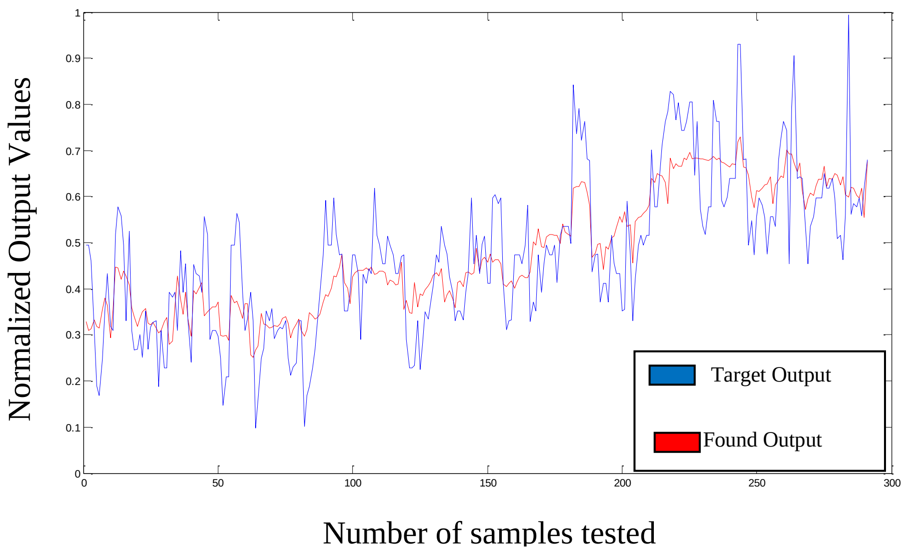


Figure 6.2: Test results with 3 hidden units

The error could be minimized down to 0.1049 for the above figure with 3 hidden units.

The following Figure 6.3 is the display of training data results with 5 hidden units.

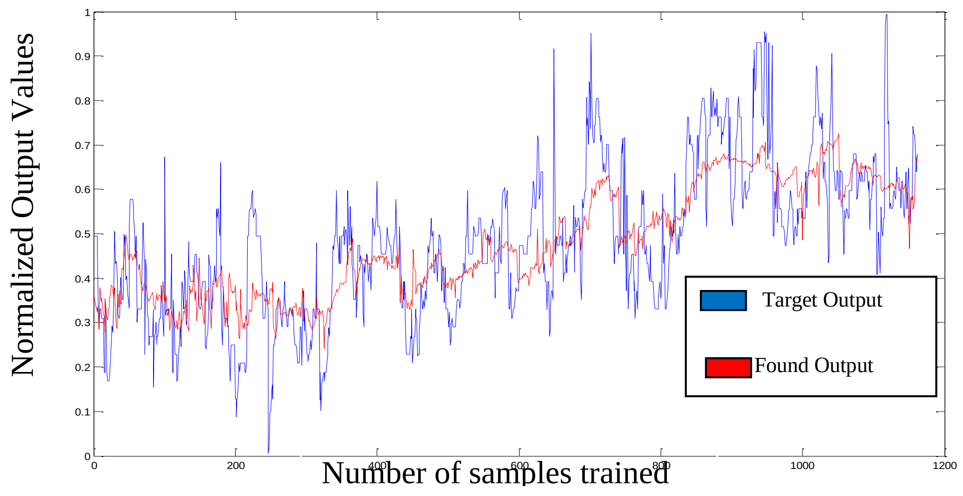


Figure 6.3: Training results with 5 hidden units

The error could be minimized down to 0.0996 for the above figure with 5 hidden units.

The following Figure 6.4 exhibits the test data results for the above Figure 6.3 with the same number of hidden units.

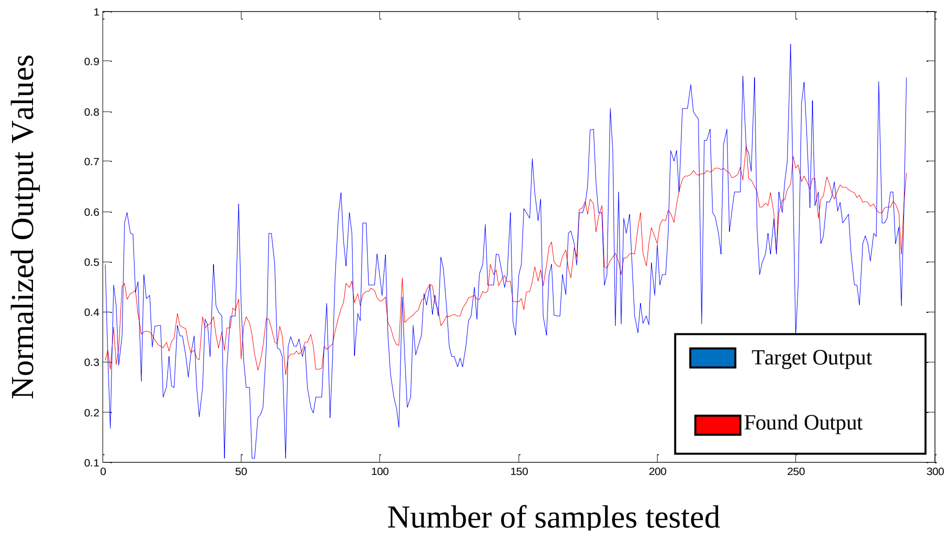


Figure 6.4: Test results with 5 hidden units

The error could be minimized down to 0.1070 for the above figure with 5 hidden units.

The following Figure 6.5 shows the training data results with 10 hidden units.

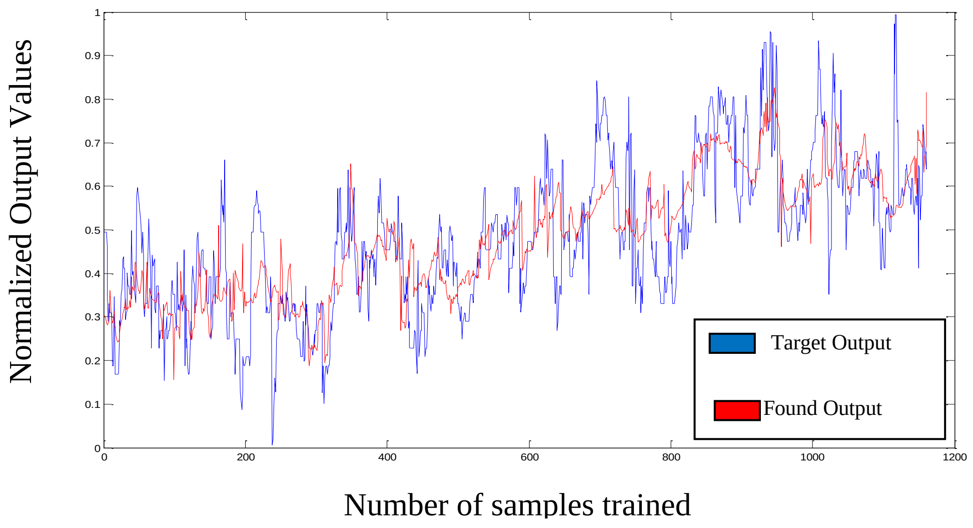


Figure 6.5: Training results with 10 hidden units

The error could be minimized down to 0.1025 for the above figure with 10 hidden units.

The following Figure 6.6 shows the test data results for the above Figure 6.5 with the same number of hidden units.

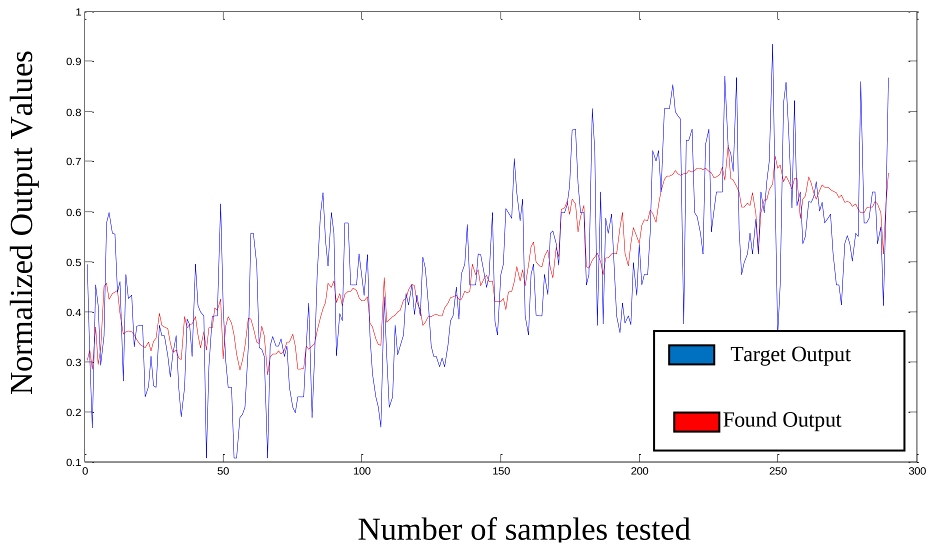


Figure 6.6: Test results with 10 hidden units

The error could be minimized down to 0.1115 for the above figure with 10 hidden units.

The following Figure 6.7 is the training result with 15 hidden units.

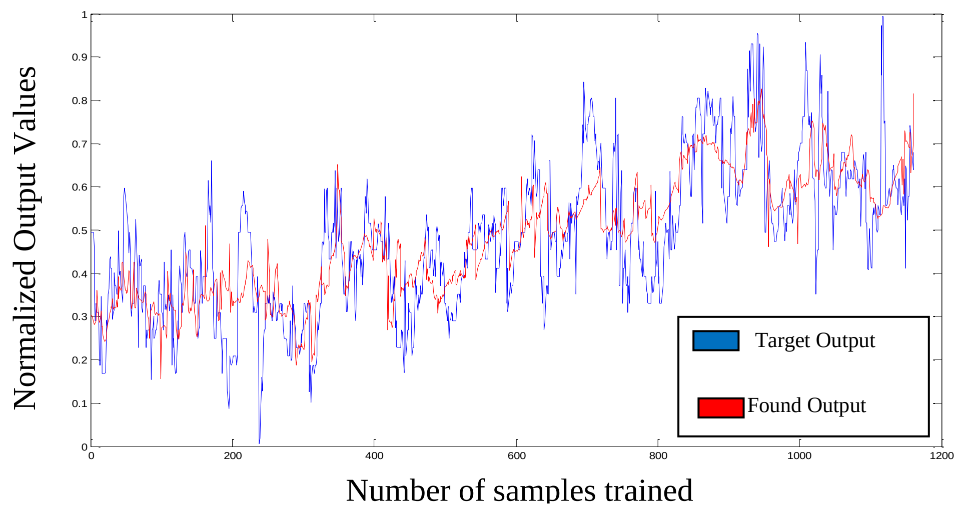


Figure 6.7: Training results with 15 hidden units

The error could be minimized down to 0.1040 for the above figure with 15 hidden units.

The following Figure 6.8 displays the test data results for the above Figure 6.7 training result with the same number of hidden units.

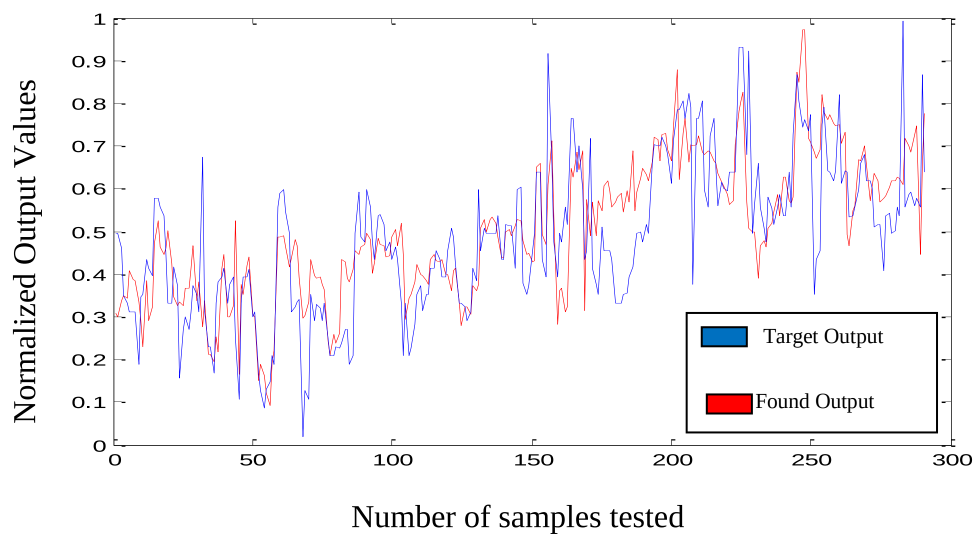
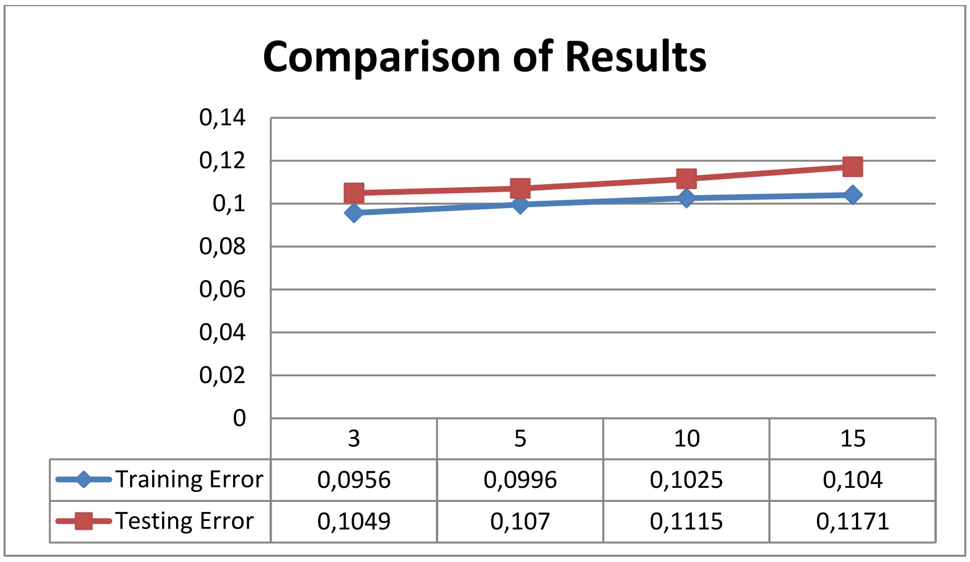


Figure 6.8: Test results with 15 hidden units

The error could be minimized down to 0.1171 for the above figure with 15 hidden units.

As shown from the above experimental results, as the number of hidden units increase, the magnitude RMSE error also increases slightly. This is due to increase in number of variables to be optimized and GAs are very sensitive to this. Their performance degrades with increasing number of variables to be optimized. Moreover, as was mentioned in previous chapters, very small number of hidden units may not be enough in order to reach the best results. And too many hidden units may cause overlearning in the model. So there should be a middle point for selecting the number of hidden units. However, in this thesis experimental results show that when less number of hidden units are used, then the better results are obtained. Therefore, we may see that for training results when we have 15 and 3 hidden units, the error could be minimized down to 0.1040 and 0.0956 respectively.

The following Figure 6.9 illustrates all the results conducted in this thesis and demonstrates the influence of the number of hidden units used.



	3	5	10	15
◆ Training Error	0,0956	0,0996	0,1025	0,104
■ Testing Error	0,1049	0,107	0,1115	0,1171

Figure 6.9: Comparison of results

The dataset that is used in this thesis was used for other modeling methods such as Partial Least Squares (PLS), Neural networks with feedforward backpropagation and 2 layers, Sugeno-type fuzzy logic and Anfis. The following Table 6.1 illustrates the results of above mentioned modeling methods by Radu Platon in [22]. In most cases, our proposed model performs better than these well-known approaches.

Table 6.1 Different Modeling Method Results [22]

Modeling method	PLS	Sugeno-type fuzzy logic	Neural networks	Anfis
Training Error	0.689	0.196	0.157	0.163
Testing Error	0.627	0.331	0.401	0.480

When comparing results obtained in this thesis using RBFNN modeling method and results in above Table 6.1, we can see a significant difference in minimization of the training and validation errors by modeling method used in our work.

Chapter 7

CONCLUSION

This thesis presents a solution to the data modeling problem. The developed model in this thesis is able to model a data based on previous experience and optimized parameters of RBFNN. The dataset of BLSC was used for development of the model. The combination of a genetic algorithm with RBFNN is a good approach to the given task as seen from the performance of the developed model. A hybridized genetic algorithm with an integrated local search could easily optimize the RBFNN parameters for a single objective problem, which was to minimize the error, i.e. making difference between target output and the found output to be zero.

RBFNN parameters played very important roles in optimizing this model, since a genetic algorithm used all those parameters in its chromosome. There were many conducted experiments of developed model with different variations of RBFNN parameters, for example; different numbers of hidden units were used to see how it reflects the results and also different intervals for such parameters as weights and width was used, which showed great difference in improvement of results.

The Multi-objective version of developed model can be a future task to be implemented for the proposed algorithm in this thesis. Optimizing more than one objective function is very popular for the last decades. And therefore it will be a primary concern for further improvement, since the current model consists of the powerful integration of genetic algorithm with RBFNN, which in addition uses local search, thus becoming a hybridized version of genetic algorithm.

REFERENCES

[1] Parras-Gutierrez, E. , del Jesus, M.J. , Rivas, V.M. , Merelo, J.J. , “Parameter Estimation for Radial Basis Function Neural Network Design by Means of Two Symbiotic Algorithms”, Advanced Engineering Computing and Applications in Sciences, 2008. ADVCOMP '08, Sept. 29 2008-Oct. 4 2008, 164–169.

[2] Alberto Guillen, Ignacio Rojas, Jesus Gonzalez, Hector Pomades, L. J. Herrera and Francisco Fernandez, “Multiobjective RBFNNs Designer for Function Approximation: An Application for Mineral Reduction”, Advances in Natural Computation, 2006, Volume 4221/2006, 511-520, DOI: 10.1007/11881070_71.

[3] N. Tang, “Application of RBF neural network based on adaptive hierarchical genetic algorithm in soft sensor modeling”, Natural Computation (ICNC), 2011 Seventh International Conference, 26-28 July 2011, 83–86.

[4] Qi Zhi-dong, Zhu Xin-jian, Cao Guang-yi, “Temperature modeling of DMFC based on RBFNN and neural fuzzy control study”, Journal of System Simulation, 2007, 126-137.

[5] B. Burdsall, and C. Giraud-Carrier, “GA-RBF: A Self Optimising RBF Network”, ICANNGA’97, Springerverlag, 1997, 348-351.

[6] C. Harpham et al, “A review of genetic algorithms applied to training radial basis function Networks”, Neural Computing & Applications, 2004, 193-201.

[7] V.M. Rivas, J.J. Merelo, P.A. Castillo, M.G. Arenas, J.G. Castellanos, “Evolving RBF neural Networks for time series forecasting with Ev RBF”, Information Sciences, 2004, 207-220.

[8] R. Schwaiger, and H. A.Mayer, “Genetic Algorithms to create training data sets for artifical neural networks”, “Proceedings of the 3NWGA”, 1997.

[9] B. A. Whitehead, T. D. Choate, “Cooperative-Competitive Genetic Evolution of Radial Basis Function Centers and Widths for Time Series Prediction”, IEEE Transactions on Neural Networks 7, 1996, 869-880.

[10] Li-juan Xie, Xing-qian Ye, Dong-hong Liu, Yi-bin Ying, “Application of principal component-radial basis function neural networks (PC-RBFNN) for the detection of water-adulterated bayberry juice by near-infrared spectroscopy”, J ZhejiangUnivciB, 2008, 982–989.

[11] R.Sivaraj, Dr. T. Ravichandran, “A Review of Selection Methods in Genetic Algorithms”, International Journal of Engineering Science and Technology (IJEST), 2011, 0975-5462.

[12] A.G. Bors, "Introduction of the Radial Basis Function (RBF) Networks", Online Symposium for Electronics Engineers, 2001, 1-7.

[13] S. S. Fayaed, A. El-Shafie, O. Jaafar, “Performance of artificial neural network and regression techniques for simulation model in reservoir inter-relationships”, International Journal of Physical Sciences, 2011, 7738-7748.

[14] Victor M Rivas, Maribel G Arenas, Juan J Merelo, Alberto Prieto, EvRBF: evolving RBF neural networks for classification problems, “AIC'07 Proceedings of the 7th Conference on 7th WSEAS International Conference on Applied Informatics and Communications”, 2007, Pages 98-103.

[15] M. Orr., Introduction to radial basis function networks, Technical report, Institute for Adaptive and Neural Computation, 1996.

[16] Nawaf Hamadneh, Saratha Sathasivam,Ong Hong Choon, Higher Order Logic Programming in Radial Basis Function Neural Network", Applied Mathematical Sciences, Vol.6," 2012, no.3, 115–127.

[17] Xiaojun Yao, Xiaoyun Zhang, Ruisheng Zhang, Mancang Liu, Zhide Hu, Botao Fan, Prediction of gas chromatographic retention indices by the use of radial basis function neural networks, "Department of Chemistry, Lanzhou Uni_ersity, Lanzhou 730000, China Uni_ersite´ Paris 7 -Denis Diderot, ITODYS 1, Rue Guy de la Brosse, 75005 Paris, France", Talanta 57 (2002) 297–306.

[18] Krzanowski, W.J., Statistical Modelling, Arnold, London, 1998, p.107.

[19] Hazewinkel, Michiel, ed., "Regression analysis", Encyclopedia of Mathematics, Springer, (2001) ISBN 978-1-55608-010-4.

[20] Dodge, Y., The Oxford Dictionary of Statistical Terms, 2003, OUP. ISBN 0-19-920613-9.

[21] M.Amazouz, R.Platon, "Soft-sensors for real-time monitoring and control of a black liquor concentration process", CanmetEnergy, Natural Resources Canada, Varennes(QC) J3X IS6 Canada.

[22] R.Platon, "Soft Sensor Development", Concordia University, Montreal, Quebec, Canada, (2009).