

STATUS OF THESIS

Title of thesis

PROPOSED METHODOLOGY FOR OPTIMIZING THE TRAINING PARAMETERS OF A MULTILAYER FEED-FORWARD ARTIFICIAL NEURAL NETWORKS USING A GENETIC ALGORITHM

I OSMAN AHMED ABDALLA,

hereby allow my thesis to be placed at the information Resource Center (IRC) of Universiti Teknologi PETRONAS (UTP) with the following conditions:

- 1. The thesis becomes the property of UTP
- 2. The IRC of UTP may make copies of the thesis for academic purposes only.
- 3. This thesis is classified as

Confidential

Non-confidential

If the thesis is confidential, please state the reason:

---

---

The contents of the thesis will remain confidential for \_\_\_\_\_ years.

Remarks on disclosure:

---

---

Signature of Author

Permanent address:

University of Gezira, Wadmadani,  
Sudan , P.O. Box 20.

Date: 28/1/2011

**Dr. Mohamed Nordin Zakaria**  
Senior Lecturer  
Computer & Information Science Department  
Universiti Teknologi PETRONAS  
Bandar Seri Iskandar, 31750 Tronoh  
Perak Darul Ridzuan, MALAYSIA

Endorsed by

Signature of Supervisor

Dr. Mohd Nordin Bin Zakaria

Date: 28/1/2011

UNIVERSITI TEKNOLOGI PETRONAS

PROPOSED METHODOLOGY FOR OPTIMIZING THE TRAINING  
PARAMETERS OF A MULTILAYER FEED-FORWARD ARTIFICIAL NEURAL  
NETWORKS USING A GENETIC ALGORITHM

By

OSMAN AHMED ABDALLA

The undersigned certify that they have read, and recommend to the Postgraduate  
Studies Programme for acceptance this thesis for the fulfillment of the requirements  
for the degree stated.

Dr. Mohamed Nordin Zakaria  
Senior Lecturer  
Computer & Information Science Department  
Universiti Teknologi PETRONAS  
Bandar Seri Iskandar, 31750 Tronoh  
Perak Darul Ridzuan, MALAYSIA

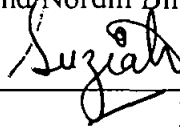
Signature:



Main Supervisor:

Dr. Mohd Nordin Bin Zakaria

Signature:

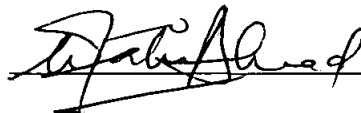


Dr. Suziah Sulaiman  
Senior Lecturer  
Computer & Information Sciences Department  
Universiti Teknologi PETRONAS

Co-Supervisor:

Dr. Suziah Bint Sulaiman

Signature:



Assoc. Prof. Dr. Wan Fatimah Birwan Aniffah  
Senior Lecturer  
Computer & Information Sciences Department  
Universiti Teknologi PETRONAS  
Bandar Seri Iskandar, 31750 Tronoh,  
Perak Darul Ridzuan, MALAYSIA.

Co-Supervisor:

Assoc. Prof. Dr. Wan Fatimah Birwan Aniffah

Signature:



Dr Mohd Fadzil B Hassan  
Head  
Computer & Information Sciences Department  
Universiti Teknologi PETRONAS

Head of Department:

Dr. Mohd Fadzil Bin Hassan

Date:

31/1/2011

PROPOSED METHODOLOGY FOR OPTIMIZING THE TRAINING  
PARAMETERS OF A MULTILAYER FEED-FORWARD ARTIFICIAL NEURAL  
NETWORKS USING A GENETIC ALGORITHM

By

OSMAN AHMED ABDALLA

A Thesis

Submitted to the Postgraduate Studies Programme  
as a Requirement for the Degree of

DOCTOR OF PHILOSOPHY

COMPUTER AND INFORMATION SCIENCES DEPARTMENT

UNIVERSITI TEKNOLOGI PETRONAS

BANDAR SRI ISKANDAR

PERAK

JANUARY 2011

DECLARATION OF THESIS

Title of thesis

PROPOSED METHODOLOGY FOR OPTIMIZING THE  
TRAINING PARAMETERS OF A MULTILAYER FEED-  
FORWARD ARTIFICIAL NEURAL NETWORKS USING A  
GENETIC ALGORITHM

I OSMAN AHMED ABDALLA,

hereby declare that the thesis is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at UTP or other institutions.



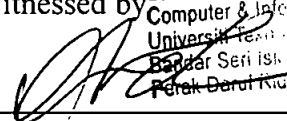
Signature of Author

Permanent address:

University of Gezira, Wadmadani,  
Sudan, P.O. Box 20.

Date: 28/1/2011

Witnessed by **Dr. Mohamed Nordin Zakaria**  
Senior Lecturer  
Computer & Information Science Depart  
Universiti Teknikal  
Bandar Seri Iskandar, 32010, Perak  
Perak Darul Ridzuan, MALAYSIA



Signature of Supervisor

Dr. Mohd Nordin Bin Zakaria

Date: 28/1/2011

## ABSTRACT

An artificial neural network (ANN), or shortly "neural network" (NN), is a powerful mathematical or computational model that is inspired by the structure and/or functional characteristics of biological neural networks. Despite the fact that ANN has been developing rapidly for many years, there are still some challenges concerning the development of an ANN model that performs effectively for the problem at hand. ANN can be categorized into three main types: single layer, recurrent network and multilayer feed-forward network. In multilayer feed-forward ANN, the actual performance is highly dependent on the selection of architecture and training parameters. However, a systematic method for optimizing these parameters is still an active research area. This work focuses on multilayer feed-forward ANNs due to their generalization capability, simplicity from the viewpoint of structure, and ease of mathematical analysis. Even though, several rules for the optimization of multilayer feed-forward ANN parameters are available in the literature, most networks are still calibrated via a trial-and-error procedure, which depends mainly on the type of problem, and past experience and intuition of the expert. To overcome these limitations, there have been attempts to use genetic algorithm (GA) to optimize some of these parameters. However most, if not all, of the existing approaches are focused partially on the part of architecture and training parameters. On the contrary, the GA-ANN approach presented here has covered most aspects of multilayer feed-forward ANN in a more comprehensive way. This research focuses on the use of binary-encoded genetic algorithm (GA) to implement efficient search strategies for the optimal architecture and training parameters of a multilayer feed-forward ANN. Particularly, GA is utilized to determine the optimal number of hidden layers, number of neurons in each hidden layer, type of training algorithm, type of activation function of hidden and output neurons, initial weight, learning rate, momentum term, and epoch size of a multilayer feed-forward ANN. In this thesis, the approach has been analyzed and algorithms that simulate the new approach have been mapped out. The

approach has been tested in three actual operations, in addition to standard XOR problem; where the results have shown the applicability of the proposed approach in those applications. The proposed method is considered novel as it has proven that GA-based method can be comprehensively utilized to determine multilayer feed-forward ANN architecture and training parameters. This method is more effective and gives a more precise performance than existing approaches, in addition to being less human dependent. It also has a better generalization capability and training stability. In summary, the main contributions of this research are: demonstrates the strength of genetic algorithm (GA), auto designing of multilayer feed-forward ANN, and demonstrates the hybridization capability of GA with multilayer feed-forward ANN.

## ABSTRAK

Rangkaian neural buatan (ANN) atau secara ringkasnya 'rangkaian neural' (NN) ialah sebuah model matematikal atau komputasi berkeupayaan tinggi yang diilhamkan dari ciri-ciri struktur dan/atau fungsi rangkaian neural biologi. Walau pun ANN telah membangun dengan pesat sejak beberapa tahun, namun masih terdapat beberapa masalah berkaitan pembinaan model ANN yang dan tepat dalam menangani sesuatu masalah. ANN boleh dikategorikan kepada tiga jenis utama: satu lapisan, berulang dan rangkaian suap ke depan berbilang lapisan. Dalam ANN suap ke depan berbilang lapisan, prestasi sebenar amat bergantung kepada pemilihan seni bina dan parameter latihan. Walau bagaimanapun, kaedah sistematik untuk menentukan parameter ini masih merupakan satu bidang penyelidikan yang aktif. Kajian ini memberi fokus kepada ANN suap ke depan berbilang lapisan berdasarkan kepada keupayaan pengitlakan ANN, ringkas dari segi struktur, dan senang untuk dianalisa secara matematik. Walau pun, terdapat beberapa peraturan untuk menentukan parameter ANN suap ke depan berbilang lapisan di dalam penerbitan, kebanyakan rangkaian masih ditentukur menggunakan kaedah cuba-cuba, yang bergantung kepada jenis masalah dan pengalaman serta gerak hati pakar. Bagi mengatasi kekurangan-kekurangan ini, terdapat beberapa percubaan untuk menggunakan algoritma genetik bagi menentukan sebahagian dari parameter-parameter tersebut. Namun, kebanyakan dari kaedah-kaedah yang dicadangkan tersebut, jika tidak semuanya, lebih tertumpu kepada seni bina dan parameter latihan. Sebaliknya, kaedah GA-ANN yang diperkenalkan di sini merangkumi kebanyakan aspek-aspek ANN suap ke depan berbilang lapisan dalam satu cara yang lebih komprehensif. Penyelidikan ini memberi fokus kepada penggunaan algoritma genetik terkod perduaan (GA) bagi melaksanakan strategi pencarian yang efisien untuk mendapatkan seni bina optima dan parameter latihan bagi rangkaian ANN suap ke depan berbilang lapisan. Khususnya, GA digunakan untuk menentukan bilangan lapisan tersembunyi optima,

bilangan neuron di dalam setiap lapisan tersembunyi, jenis algoritma latihan, jenis fungsi pengaktifan neuron tersembunyi dan neuron output, beban awal, kadar pembelajaran, faktor momentum, dan saiz epok rangkaian ANN suap ke depan berbilang lapisan. Di dalam tesis ini, kaedah ini telah dianalisa dan algoritma-algoritma bagi mensimulasi kaedah baru ini telah dikenal pasti. Kaedah ini telah diuji dalam tiga keadaan operasi sebenar, termasuk juga masalah XOR standard, di mana keputusan-keputusannya telah menunjukkan kebolegunaan kaedah yang dicadangkan ini di dalam aplikasi-aplikasi tersebut. Kaedah yang dicadangkan dianggap baru kerana ia telah membuktikan bahawa kaedah berasaskan GA boleh digunakan secara komprehensif untuk menentukan seni bina ANN suap ke depan berbilang lapisan dan parameter untuk latihan. Kaedah ini lebih berkesan dan memberi prestasi yang lebih tepat berbanding kaedah-kaedah sedia ada, selain dari kurang bergantung kepada manusia. Ia juga mempunyai keupayaan pengitlakan dan kestabilan latihan yang lebih baik. Secara ringkasnya, sumbangan-sumbangan utama hasil penyelidikan ini ialah: membuktikan kekuatan algoritma genetik (GA), rekaan automatik ANN suap ke depan berbilang lapisan, dan membuktikan keupayaan penghibridan GA dan ANN suap ke depan berbilang lapisan.



In compliance with the terms of the Copyright Act 1987 and the IP Policy of the university, the copyright of this thesis has been reassigned by the author to the legal entity of the university,  
Institute of Technology PETRONAS Sdn Bhd.

Due acknowledgement shall always be made of the use of any material contained in, or derived from, this thesis.

© OSMAN AHMED ABDALLA, 2011  
Institute of Technology PETRONAS Sdn Bhd  
All rights reserved.

## ACKNOWLEDGEMENTS

Firstly, I would like to thank ALLAH SWT, whom with His willing giving me the opportunity to complete this thesis.

Secondly, I would like to express my deep and sincere gratitude to my supervisor Dr. Mohd Nordin Zakaria for his support, patient, valuable advice, and guidance from the very early phase of this research as well as giving me extraordinary experiences throughout the work.

Deepest gratitude are also due to the members of the supervisory committee, Dr Suziah Sulaiman and Dr Wan Fatimah without whose knowledge and assistance this work would not have been successful.

Special thanks also to all graduate friends, especially; Yssir Abdelgadir, Mahamat Issa, Asim Abdalla, and Hassan Yousif for sharing the literature and invaluable assistance.

The author would also like to convey thanks to the UTP and Computer and information Sciences Department Staff for providing the financial means and laboratory facilities.

The author wishes to express his love and gratitude to his beloved families; for their understanding & endless love, through the duration of his work.

## DEDICATION

*I humbly dedicate this thesis to my brother the late Omer Ahmed, my wonderful parents, who have raised me to be the person I am today, and my wife, Mai, and my children, Rayan and Ahmed; whose encouragement, support and understanding graciously made it possible for me during all these years of study.*

## TABLE OF CONTENTS

ABSTRACT.....	v
ABSTRAK.....	vii
ACKNOWLEDGEMENTS.....	x
DEDICATION.....	xi
TABLE OF CONTENTS.....	xii
LIST OF FIGURES.....	xviii
LIST OF TABLES.....	xxi
<b>INTRODUCTION.....</b>	<b>1</b>
1.1 Introduction.....	1
1.2 Background.....	1
1.3 Research problem.....	4
1.4 Research questions.....	8
1.5 Research objectives.....	9
1.6 Scope of the thesis.....	9
1.6.1 GA for multilayer feed-forward ANN architecture and training optimization... 10	10
1.6.2 GA-ANN approach evaluation.....	10
1.6.3 GA-ANN approach validation.....	11
1.7 Research significance.....	11
1.8 Thesis outline.....	11
1.9 Summary.....	13

<b>LITERATURE REVIEW</b> .....	14
2.1 Introduction.....	14
2.2 Artificial neural networks (ANN).....	14
2.2.1 Network architectures .....	16
2.2.1.1 Single-layer neural networks .....	17
2.2.1.2 Multilayer feed-forward ANN .....	17
2.2.1.3 Recurrent neural networks (RNNs) .....	19
2.2.2 Back-propagation algorithm (BP).....	21
2.2.2.1 Feed forward process .....	22
2.2.2.2 Back-propagation of error process.....	23
2.2.2.3 Update of connection weights process.....	24
2.2.2.4 The BP training algorithm .....	25
2.2.3 Network training, validation and testing.....	26
2.2.4 Applications of ANN .....	28
2.2.4.1 Engineering .....	28
2.2.4.2 Science .....	29
2.2.4.3 Manufacturing.....	30
2.2.4.4 Marketing, finance and accounting.....	30
2.2.4.5 Health and medicine .....	31
2.2.4.6 General applications.....	31
2.2.5 ANN advantage.....	32
2.3 Genetic algorithm (GA).....	32
2.3.1 GA operators.....	34
2.3.1.1 Selection.....	34
2.3.1.2 Crossover .....	34
2.3.1.3 Mutation.....	35
2.3.2 GA applications .....	36
2.3.3 GA advantage.....	36
2.4 GA in ANN design.....	37
2.4.1 GA to optimize input variable of ANN.....	38
2.4.2 GA to optimize the structure and training process of ANN.....	40
2.4.3 GA to optimize the connection weights of ANN.....	42
2.4.4 Comparison between state of the art approaches and the proposed approach....	43

2.5 Multilayer feed-forward ANN data preprocessing, architecture and training parameters .....	45
2.5.1 Data preprocessing .....	47
2.5.1.1 Missing values .....	48
2.5.1.2 Data outlier.....	49
2.5.1.3 Data normalization.....	50
2.5.1.4 Dataset partitioning.....	51
2.5.2 Architecture.....	52
2.5.2.1 Number of hidden layers and number of neurons in hidden layers .....	52
2.5.3 Activation functions.....	54
2.5.4 BP multilayer feed-forward ANN.....	57
2.5.4.1 Steepest Descent (SD).....	58
2.5.4.2 Conjugate Gradient (CG).....	59
2.5.4.3 Quasi-Newton (QN).....	60
2.5.4.4 Levenberg Marquardt (LM).....	61
2.5.5 Initial weights.....	62
2.5.6 Learning rate .....	62
2.5.7 Momentum term.....	64
2.5.8 Epoch size .....	65
2.6 Summary .....	66
<b>RESEARCH METHODOLOGY .....</b>	<b>67</b>
3.1 Introduction.....	67
3.2 Research tools .....	67
3.2.1 MATLAB2009a with neural networks ToolBox6.0.....	67
3.3 Research activities .....	69
3.3.1 Collection of experimental data .....	70
3.3.1.1 Data preprocessing.....	71
3.3.1.2 Missing data removal.....	71
3.3.1.3 Outlier detection and removal.....	71
3.3.1.4 Data normalization.....	74
3.3.1.5 Data partitioning .....	74
3.3.2 Multilayer feed-forward back-propagation ANN .....	75

3.3.3 GA method.....	75
3.3.3.1 GA encoding scheme .....	76
3.3.3.2 Mapping multilayer feed-forward back-propagation ANN training and design parameters into GA binary encoding .....	77
3.3.3.3 Representation of the minimization algorithm of the training function .....	78
3.3.3.4 Representation of the network structure .....	80
3.3.3.5 Representation of the activation function .....	82
3.3.3.6 Representation of the initial weight .....	83
3.3.3.7 Representation of the learning rate .....	85
3.3.3.8 Representation of the momentum term.....	86
3.3.3.9 Representation of the number of epochs.....	87
3.3.3.10 Generation of initial population .....	87
3.3.3.11 Multilayer feed-forward ANN prediction.....	87
3.3.3.12 Fitness evaluation.....	88
3.3.3.13 Termination criteria .....	88
3.3.3.14 Creation of a new population.....	88
3.4 Performance evaluation .....	88
3.5 Summary .....	89
<b>GA-ANN DEVELOPMENT APPROACH.....</b>	<b>91</b>
4.1 Introduction.....	91
4.2 General approach .....	91
4.3 Applications .....	92
4.3.1 Debutanizer of CRU .....	92
4.3.1.1 Input and output variables.....	93
4.3.1.2 Network architecture.....	94
4.3.2 Gas turbine .....	95
4.3.2.1 Input and output variables.....	95
4.3.2.2 Network architecture.....	96
4.3.3 Drilling process.....	97
4.3.3.1 Input and output variables.....	98
4.3.3.2 Network architecture.....	99
4.3.4 XOR problem.....	99

4.3.4.1 Input and output variables.....	99
4.3.4.2 Network architecture.....	100
4.4 The algorithm.....	100
4.5 Summary.....	102
<b>RESULTS AND DISCUSSION.....</b>	<b>104</b>
5.1 Introduction.....	104
5.2 GA method.....	104
5.2.1 Prediction of $iC_5$ and $nC_5$ using GA-ANN approach.....	105
5.2.2 Prediction of flank wear using GA-ANN approach.....	108
5.2.3 Prediction of net power and $T_4$ using GA-ANN approach.....	110
5.2.4 Prediction of XOR output using GA-ANN approach.....	112
5.3 Benchmarking.....	114
5.3.1 Genetic algorithms to select architecture of a feed-forward artificial neural network (Jasmina and Ramazan 2001).....	115
5.3.2 Genetic algorithms to find the number of hidden layer nodes of a feed-forward artificial neural network (Torres et al. 2005).....	115
5.3.3 A combination of genetic algorithm and artificial neural network (Makoto et al. 2006).....	116
5.3.4 Feed-forward neural networks designed and parameterized by genetic algorithms (Ferentinos 2005).....	116
5.3.5 Design of neural networks using genetic algorithm (Manojit and Deoki 2006).....	117
5.4 Comparative study.....	117
5.4.1 Optimality.....	118
5.4.2 Generalization capability and stability of multilayer feed-forward ANN.....	119
5.4.3 Prediction effectiveness.....	121
5.4.4 Simulation time.....	124
5.5 Summary.....	125
<b>CONCLUSION AND RECOMMENDATION.....</b>	<b>126</b>
6.1 Conclusion.....	126
6.2 Contributions.....	129



6.3 Recommendations for future work .....	130
6.3.1 Evolving the network structure .....	130
6.3.2 Selection of minimization and activation function .....	130
6.3.3 Selection of GA parameters .....	130
<b>REFERENCES</b> .....	<b>132</b>

## LIST OF FIGURES

Figure 1.1 Structure of neural network of n input, n hidden, and n output layers .....	2
Figure 1.2 Effect of hidden layer size on network generalization .....	5
Figure 2.1 Structure of biological neuron (Caetano 2006) .....	15
Figure 2.2 Model of Computing Neuron (Haykin 1994) .....	16
Figure 2.3 Principles of ANN algorithms (Demuth and Beale 2005) .....	16
Figure 2.4 Atypical Single layer neural network (Demuth and Beale 2005) .....	17
Figure 2.5 A schematic diagram of multi-layered feed-forward neural network with back-propagation training algorithm (Demuth and Beale 2005) .....	18
Figure 2.6 Elman recurrent networks (Elman 1990) .....	20
Figure 2.7 Jordan recurrent networks (Jordan 1986) .....	21
Figure 2.8 Schematic diagram of error back propagation (Imdat and Yasar 2009) ....	25
Figure 2.9 Flowchart of the back-propagation training algorithm process (Imdat and Yasar 2009) .....	26
Figure 2.10 GA procedure .....	33
Figure 2.11 Crossover operation .....	35
Figure 2.12 Mutation operator .....	36
Figure 2.13 Search speed according to (Kitano 1990) .....	38
Figure 2.14 A hard-limit activation function .....	56
Figure 2.15 A threshold-logic activation function .....	56
Figure 2.16 Continuous activation functions (a) the sigmoid (b) the hyperbolic tangent .....	56
Figure 2.17 The Gaussian activation functions .....	57
Figure 3.1 The research methodology steps .....	70
Figure 3.2 Score plot of the first two principal components of a PCA study for 402 observations from debutanizer .....	72
Figure 3.3 Score plot of the first two principal components of a PCA study for 40 gas turbine observations .....	73

Figure 3.4 Score plot of the first two principal components of a PCA study for 64 flank wear observations .....	73
Figure 3.5 Binary representations of multilayer feed-forward ANN architecture and training parameters.....	78
Figure 3.6 Training algorithm sub-string.....	79
Figure 3.7 Network structure sub-string.....	82
Figure 3.8 Sigmoid, tanh and linear activation function.....	82
Figure 3.9 Activation function sub-string.....	83
Figure 3.10 Initial weights sub-string.....	83
Figure 3.11 Learning rate sub-string.....	85
Figure 3.12 Momentum term sub-string.....	86
Figure 3.13 Epoch numbers sub-string.....	87
Figure 4.1 Debutanizer CRU Units at Melaka Refinery.....	93
Figure 4.2 A schematic of the multilayer feed-forward ANN architecture for debutanizer.....	95
Figure 4.3 TAURUS 60 gas turbine .....	95
Figure 4.4 A schematic of the multilayer feed-forward ANN architecture for gas turbine.....	97
Figure 4.5 A radial drilling machine (Batliboi Limited, BR618 model).....	97
Figure 4.6 A schematic of the multilayer feed-forward ANN architecture for flank wear.....	99
Figure 4.7 A schematic of the multilayer feed-forward ANN architecture for XOR 100	
Figure 4.8 Sequence diagram of the algorithm.....	102
Figure 5.1 GA method performance for several crossover and mutation probabilities .....	106
Figure 5.2 GA method performance for several population size and two values of generation size .....	106
Figure 5.3 Best MSE found during GA process for the $iC_5$ and $nC_5$ prediction model .....	107
Figure 5.4 Average MSE of the entire population during GA process for the $iC_5$ and $nC_5$ prediction model .....	107
Figure 5.5 Best MSE found during GA process for the flank wear prediction model .....	109

Figure 5.6 Average MSE of the entire population during GA method process.....	109
Figure 5.7 Best MSE found during GA process for the net power and $T_4$ prediction model.....	111
Figure 5.8 Average MSE of the entire population during GA process for net power and $T_4$ .....	111
Figure 5.9 Best MSE found during GA process for the XOR output prediction model .....	113
Figure 5.10 Average MSE of the entire population during GA process for the XOR output prediction model .....	113
Figure 5.11 GA structure of Jasmina and Ramazan approach.....	115
Figure 5.12 GA structure of Torres et al. approach .....	115
Figure 5.13 GA structure of Makoto et al. approach .....	116
Figure 5.14 GA structure of Ferentinos approach .....	117
Figure 5.15 GA structure of Manojit and Deoki approach .....	117
Figure 5.16 Comparison of proposed approach with other approaches in terms of prediction effectiveness (MSE).....	122
Figure 5.5.17 Comparison of proposed approach with other approaches in terms of prediction effectiveness (RMSE).....	122
Figure 5.18 Comparison of proposed approach with other approaches in terms of prediction performance (R).....	123
Figure 5.19 Comparison of proposed approach with other approaches in terms of prediction performance ( $R^2$ ) .....	124
Figure 5.5.20 Comparison of proposed approach with other approaches in terms of simulation time.....	125

## LIST OF TABLES

Table 2.1 The state of art of ANN design using GA (1/2).....	44
Table 2.2 The state of art of ANN design using GA (2/2).....	45
Table 2.3 The list of most common parameters in designing a multilayer feed-forward ANN (Kaastra and Boyd 1996).....	47
Table 2.4 Common approaches to replace missing data.....	49
Table 2.5 Common methods to detect outlier data .....	50
Table 2.6 Common approaches for data partitioning.....	52
Table 2.7 Some common methods to select the number of hidden neurons.....	53
Table 2.8 Common activation functions of hidden and output layers .....	55
Table 2.9 Common methods to select the range of initial weights.....	62
Table 2.10 The common methods to select learning rate .....	63
Table 2.11 The common methods of selecting momentum term.....	64
Table 3.1 Range of multilayer feed-forward ANN architecture and training parameters .....	78
Table 3.2 Binary encoding of training algorithm of the multilayer feed-forward ANN .....	80
Table 3.3 Binary encoding of multilayer feed-forward ANN structure.....	81
Table 3.4 Binary encoding of activation functions of hidden and output neurons of the multilayer feed-forward ANN.....	83
Table 3.5 Binary encoding of initial weights value of the multilayer feed-forward ANN (1/2) .....	84
Table 3.6 Binary encoding of initial weights value of the multilayer feed-forward ANN (2/2) .....	84
Table 3.7 Binary encoding of learning rate value of the multilayer feed-forward ANN .....	85
Table 3.8 Binary encoding of momentum term value of the multilayer feed-forward ANN .....	86
Table 3.9 Binary encoding numbers epoch of the multilayer feed-forward ANN .....	87

Table 4.1 Statistical analysis of CRU debutanizer dataset .....	94
Table 4.2 Statistical analysis of gas turbine dataset.....	96
Table 4.3 Statistical analysis of drilling machine dataset .....	98
Table 4.4 XOR mapping .....	100
Table 5.1 GA parameters .....	105
Table 5.2 Required architecture and training parameters of multilayer feed-forward ANN.....	118
Table 5.3 GA specification of proposed and other approaches .....	119
Table 5.4 GA features of proposed and other approaches .....	119
Table 5.5 Data pre-processing techniques .....	120
Table 5.6 Necessary features for generalization capability and stability.....	121
Table 5.7 Results of the proposed GA-ANN approach and other approaches in terms of prediction effectiveness .....	121
Table 5.8 Results of the proposed GA-ANN approach and other approaches in terms of prediction performance.....	123
Table 5.9 Simulation time of proposed GA-ANN approach and other approaches ..	124

# Chapter 1

## INTRODUCTION

### 1.1 Introduction

The aim of this work is to propose a methodology for optimizing the architecture and training parameters of a multilayer feed-forward artificial neural network (ANN). It is based on a genetic algorithm (GA). The current chapter describes the overall research background and provides brief information on ANN and GA. It outlines the problem statements together with the relevant research questions. The scope of work and contributions of the thesis are discussed as well.

### 1.2 Background

The ANN has been used to emulate the human decision and prediction abilities. ANN is found to be more flexible and suitable than other modeling methods (Zhang et al. 1998). ANN is based on the neural architectures and function of the human brain (Haykin 1994), and is described as a group of simple processing units, *known as neurons (nodes)*, which are arranged in parallel layers that are connected to each other by weighted connections. By virtue of hidden layers of neurons that lie between the input and output layers of the network, and the nonlinear activation functions that are used to translate nodal input to output, ANN provides linear and nonlinear modeling without the requirement of preliminary information and assumption as to the relationship between input and output variables. This provides ANN an advantage over other statistical and conventional prediction methods such as logistic regression and numerical methods, in which nonlinear interactions between variables must be modeled in explicit functional form (Tu 1996).

Based on their architectures, ANN can be categorized into three main types: single layer, recurrent and multilayer feed-forward network. This work focuses on multilayer feed-forward (ANN) due to their generalization capability, simplicity from a structure viewpoint, and ease of mathematical analysis.

ANN trained with feed-forward back-propagation BP algorithm has been extensively applied with great success in various disciplines, such as automotives (Majors et al. 2002), banking (Arzum and Yalcin 2007), electronics (Bor-Ren and Hoft 2003), finance (Xiaotian et al. 2008), industry (Cheginia et al. 2008), telecommunication (Perambur and Preechayasomboon 2002), oil and gas (Fred et al. 2000), and robotics (Huang et al. 2008) as well as others.

In designing a multilayer feed-forward ANN, the most common parameters that should be selected by researchers are as follows (Kaastra and Boyd 1996):

- Data preprocessing parameters including frequency of data (i.e., daily, weekly, monthly, quarterly type of technical data, fundamental), method of data sampling, and method of data scaling (i.e., minimum/maximum, mean/standard deviation).
- Training parameters including learning rate, momentum term, epoch size, weight initialization, size of training, validation, testing sets, and type of training algorithm.
- Design parameters including number of input neurons, number of hidden layer, number of neurons in each hidden layer, number of output neurons, and type of activation function for the hidden and output layers.

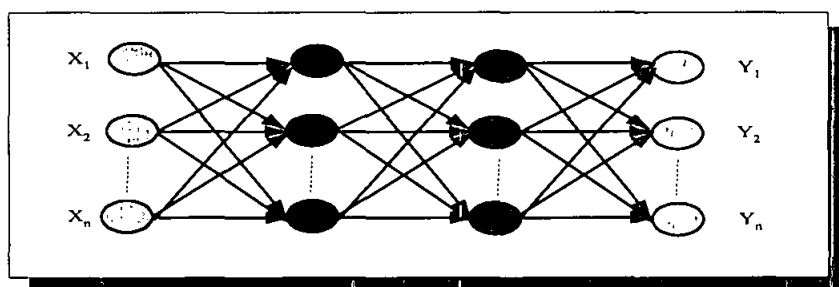


Figure 1.1 Structure of neural network of  $n$  input,  $n$  hidden, and  $n$  output layers



Despite their popularity, a multilayer feed-forward ANN, in particular, has a drawback that is its performance efficiency, convergence and its accuracy may vary depending on the parameters used to design and to train them (Sungzoon and James 1993). The networks can be configured with different number of hidden layers, different number of neurons in each layer, and can be trained with different values of learning rate and momentum term that govern the changing of connection weights, as new cases are learned. Different network structures, different type of activation function for hidden and output neurons, and different type of training algorithms may have some effect on network performance. Therefore, it is an extremely difficult task to choose an appropriate method for determining the optimal network design and training parameters.

A promising method, however, GA based on the Darwin Theory of Evolution is a combinatorial optimization and powerful technique that searches for an optimal/near value of a complex objective function by simulation of the natural evolutionary process. GA can be used to search for optimal ANN configurations. In fact, GA has been successfully used in a wide range of problem areas (Goldberg 1989).

In brief, GA consists of three main operators: selection, crossover, and mutation. GA algorithm begins with a set of potential solutions to the problem being examined; this solution which is represented by chromosomes in GA is called the population.

Crossover operation is used to obtain a new solution by combining two different chromosomes (Parents) to generate new better offspring (Childs); while a new solution that results by altering existing members of the population is called mutation.

Although there are many existing GA-based attempts to optimize multilayer feed-forward ANN parameters, these attempts are still partially focused on some parameters; for example (Taeksoo and Ingoo 2000; Blanco et al. 2001; Jasmina and Ramazan 2001; Makoto et al. 2006; Sedki et al. 2009) applied GA method to evolve the connection weights of multilayer feed-forward ANN. Another GA method was used by (Hyun-jung and Kyung-shik 2007) to optimize both connection weights and the number of hidden neurons, while (Ferentinos 2005) considered a binary-encoded GA to determine the optimum multilayer feed-forward ANN structure, training algorithm, and activation function. In this research, GA is applied in a manner more comprehensive compared to the existing ones. The proposed GA-ANN approach has

been adopted to search for optimal structure, training algorithm, activation function, initial weight, learning rate, momentum term and epoch size.

Further, a comprehensive benchmarking has been carried out to assure the significance of the present work in accordance to the following performance metrics:

- Optimality
- Generalization capability and training stability of multilayer feed-forward ANN
- Prediction effectiveness
- Simulation time

### **1.3 Research problem**

Multilayer feed-forward ANN, in particular, has been utilized for modeling and prediction purpose due to nonlinearity, time variability, and difficulty in inferring input-output mapping (Fred, James et al. 2000; Kaynak et al. 2003; Patrick 2007). Current literatures on multilayer feed-forward ANN show that the determination of optimal architecture and training parameters are the major obstacles for their accuracy, and effectiveness. The success of multilayer feed-forward ANN performance for the purpose of prediction and modeling in science and engineering is tremendously affected by these factors: network architecture such as, the number of hidden layers and number of neurons in each layer, training algorithms (i.e. steepest descent, quasi-Newton, conjugate gradient, Levenberg-Marquardt, resilient back propagation algorithm, etc), activation functions, initial weight (i.e. Log-sigmoid, Softmax, Linear, etc), learning rate, momentum term, and epoch size. In ANN design and training processes, determination of the best parameters is an extremely important task.

The hidden layer(s) provide the network its capability to generalize. In theory, a neural network with single hidden layer with a sufficient number of hidden neurons is capable of approximating any continuous function. In practice, neural networks with single and infrequently two hidden layers are extensively used and have performed very well. The determination of appropriate number of hidden layers and neurons in each hidden layer is one of the extremely critical tasks in designing multilayer feed-

forward ANN. As shown in Figure 1.2, the use of huge number of hidden layers and neurons in each hidden layer may cause over-fitting, which means loss of generalization capability of the network. Over-fitting happens when a prediction model has too few degrees of freedom. In other words, it has relatively few observations in relation to its parameters, and therefore it is able to memorize individual points rather than learn the general patterns (Kaastra and Boyd 1996). In contrast, a network with too few hidden neurons may cause under-fitting which would cause the network as incapable to differentiate complex patterns leading to only a linear estimate of the actual trend (Figure 1.2).

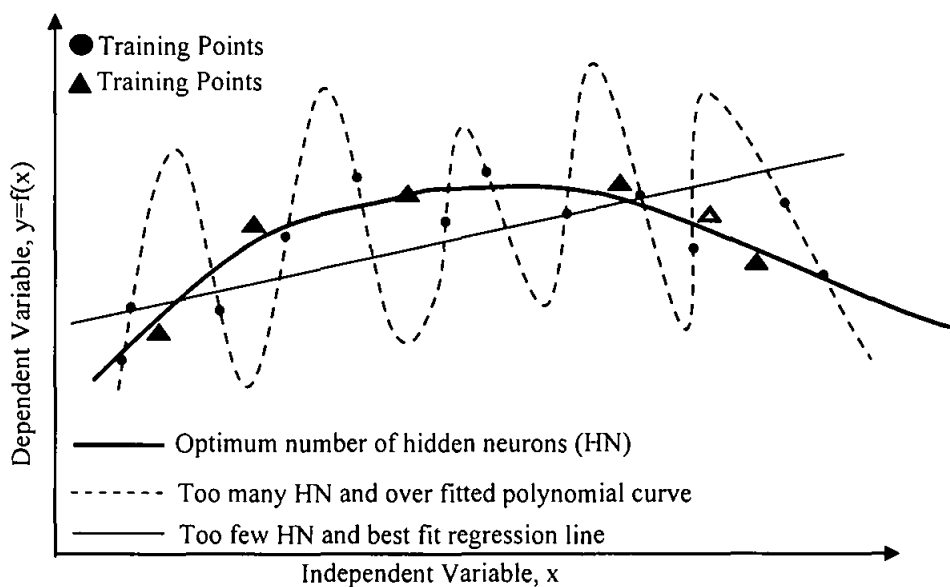


Figure 1.2 Effect of hidden layer size on network generalization

The application of multilayer feed-forward ANN in modeling nonlinear processes has a central drawback which is the lack of an accurate method to choose the most appropriate type of activation function and training algorithm. These tasks are usually based on a “trial and error” procedure performed by the developer of the model.

The activation (transfer) function is required to transform the weighted sum of all signals impinging onto a neuron so as to define its firing intensity. Most applications utilizing back-propagation ANN employ a sigmoid function, which possesses the distinctive properties of continuity and differentiability on  $(-\infty, \infty)$ . The advantage of choosing a particular transfer function over another is still not well theoretically understood (Hassoun 1995). In addition, the use of different types of activation functions will result in different performances of the network.

The basic training methodology that has been widely used is back-propagation training algorithm (Rumelhart et al. 1986). This algorithm has several modifications according to the multidimensional minimization algorithm that it uses to minimize the error during training process. In the literature, four different types of minimization algorithms were considered: steepest descent, quasi-Newton, conjugate gradient and Levenberg-Marquardt algorithm. Their main difference is the way of approximating the inverse of the Hessian matrix. The steepest descent and the conjugate gradient algorithms replace the inverse of the Hessian with the identity matrix, while the other two algorithms try to approximate it with different methods. Especially in the case of engineering applications, even different minimization algorithms of some specific training algorithm, like the back-propagation training algorithm (Rumelhart et al. 1986) that is most widely used, can result in perceptibly different performances.

Learning rate and momentum term are commonly used in connection weight changing during the training process, and to reduce the likelihood of search instability (Zupan and Gasteiger 1993; Haykin 1994). The connection weight change is determined by the use of the modified delta rule (Zupan and Gasteiger 1993), which can be written as:

$$\Delta w_{ji}(t + 1) = \alpha \cdot \Delta w_{ji}(t) + \eta \cdot \alpha \cdot \frac{\partial E}{\partial w_{ji}} \quad (1.1)$$

Where the learning rate  $\eta$  is limited to the range  $0 \leq \eta < 1$ ,  $\alpha$  is momentum term,  $t$  is the iteration of learning,  $w_{ij}$  is an adjustable parameter known as synaptic connection weight, and  $E$  is equal to:

$$\frac{1}{2} \sum_{k=1}^c e_k^2$$

In which  $e_k$  is the error between calculated output,  $o_k$ , and desired output,  $t_k$ , and  $e_k$  is determined as follows:

$$e_k = t_k - o_k \quad (1.2)$$

As implied in Eq. 1.1,  $\alpha$  accelerates the weight updates when there is a need to reduce  $\eta$  to avoid oscillation. A high  $\alpha$  value will decrease the danger of the network being stuck in local minima, however it increases the danger of overshooting the solution as does a high  $\eta$  value. In contrast, a small  $\alpha$  value leads to slow training speed.

Moreover, a high learning rate,  $\eta$ , will accelerate training speed (because of the large step) by updating the connection weight vector,  $w$ , significantly from one iteration to another. But, this may cause the search to oscillate on the error surface and never converge, consequently increasing the danger of overshooting a near-optimal  $w$ . While a small  $\eta$  value drives the search steadily in the direction of the global minimum, even though slowly.

Weight initialization has effect on network convergence (Li et al. 1993; Schmidt et al. 1993). Normally, network weights are initialized uniformly in a relatively small range with zero-mean random numbers (Rumelhart, Hinton et al. 1986). Nevertheless, a small range can guide to very small error gradients which may slow down the initial training process. The small number is very essential to decrease the likelihood of premature neurons saturation (Lee et al. 1991).

The selection of the training epoch size can affect the design and performance of the final network. For a given ANN architecture, the error in training, validation and test dataset are monitored for each training epoch. Extended training period can result in a network that can only serve as a look-up table, a phenomenon called overtraining or memorization (Zupan and Gasteiger 1993; Wythoff 1993). In theory, a high epoch size can result in near-zero error on predicting the training data (called recall), but small epoch size may produce ANN that is incapable of representing the data.

The successful application of multilayer feed-forward ANN in predicting and modeling problems is extremely influenced by the selection of these parameters. However, the selection of these parameters depends mainly on the type of problem, and the past experience and intuition of the expert. In other words, there are no perfectly clear methods or theoretical background as how to determine these parameters. This research looks into determining multilayer feed-forward ANN

architecture and training parameters, and improving the efficiency and accuracy of the network performance by adopting GA.

#### **1.4 Research questions**

The research problem specified in Section 1.3 has led to a formulation of the following research questions for this thesis:

*Question 1: What data preprocessing technique can enhance the designing of multilayer feed-forward ANN and increases the generalization ability prior to the use of GA?*

This question leads to the data preprocessing techniques presented in Chapter 3 and Chapter 4. The chapters describe the importance of data representation to the network performance and their effect on the network designing success.

*Question 2: Which multilayer feed-forward ANN architecture and training parameters that need to be optimized by using GA and why?*

This question is addressed in Chapter 3. In this research, GA is adopted in a novel way to determine optimal multilayer feed-forward ANN architecture and training parameters due to their significant effect on network performance efficiency and accuracy.

*Question 3: What is the GA structure or encoding scheme that maximizes its use as an effective tool for the optimization of multilayer feed-forward ANN architecture and training parameters and why?*

This question is addressed in Chapter 3. In this research, binary-encoded GA is adopted in a novel way to determine optimum multilayer feed-forward ANN architecture and training parameters.

*Question 4: How can GA be used to determine the best multilayer feed-forward ANN architecture and training parameters that fall into existing works?*

This question is addressed and presented conceptually in Chapter 3 and in further details in Chapter 4. The main idea is to develop an approach that focuses on every aspect of multilayer feed-forward ANN, such as, training algorithm, activation function, initial weight, learning rate, momentum term and epoch size.

*Question 5: What application domains can take advantage of the new approach presented in this thesis and for what standard and real world uses can they be applied to?*

This question is addressed in Chapter 4.

## **1.5 Research objectives**

The main objective of this work is to present a methodology for optimizing the architecture and training parameters of a multilayer feed-forward artificial neural network (ANN) and to contribute to existing work on multilayer feed-forward ANN designing and training performance. It is based on the employment of a binary-encoded genetic algorithm (GA).

The specific research objectives of this thesis are as follows:

- To propose an algorithm for auto designing of multilayer feed-forward ANN.
- To design a GA-based method that has the ability to cover most multilayer feed-forward ANN aspects for both architecture and training parameters.
- To investigate the effect of using GA on multilayer feed-forward artificial neural networks performance.

## **1.6 Scope of the thesis**

This thesis focuses on the following issues:

### **1.6.1 GA for multilayer feed-forward ANN architecture and training optimization**

Based on network architecture there are a number of different ANN types. In this work, multilayer feed-forward with back-propagation ANN is utilized due to their simplicity from the viewpoint of structure, ease of mathematical analysis and generalization capabilities. Further, there are several optimization methods that can be applied for multilayer feed-forward ANN optimization, but in this work GA is adopted to determine the best architecture and training parameters. The GA is selected due to their significant advantages such as:

- It operates on a coded form of the problem's parameters, not the parameters themselves.
- GA is capable of searching in very large solution spaces efficiently by providing a lower computational cost, because they use probabilistic transition methods instead of deterministic ones.

There are two main GA encoding schemes, which are direct and indirect. Among these types, this work provides a binary GA encoded sometimes called indirect encoding, which looks more biologically reasonable than the direct encoding one because genetic information in real chromosomes cannot specify the whole nervous system directly and independently (Yao 1993).

### **1.6.2 GA-ANN approach evaluation**

To evaluate the proposed GA-ANN approach, datasets from four different domains were obtained for experimentation. The datasets used include: a dataset from Universiti Teknologi PETRONAS GDC plant (TAURUS 60 gas turbine single-shaft generator set) collected for the period Jan. to Feb. 2008. Another dataset was collected from PETRONAS Penapisan (Melaka) Sdn Bhd from Jan. to Feb. 2007. The third dataset is a published experimental dataset of flank wear for drilling process (Panda et al. 2008). Lastly, a dataset of standard XOR problem has been selected to benchmark the proposed approach.



### **1.6.3 GA-ANN approach validation**

To validate the proposed approach, this work provides a comprehensive comparative study against relevant outstanding approaches in terms of optimality, generalization capability and training stability of multilayer feed-forward ANN, prediction effectiveness, and simulation time.

### **1.7 Research significance**

Most, if not all, of the existing approaches to optimize the multilayer feed-forward ANN based on GA have been partially focused on architecture and training parameters. In contrary, the GA-ANN approach presented in this thesis has covered most aspects of multilayer feed-forward ANN in a more comprehensive way and it has been applied practically to a real world and standard applications. The research significantly contributes to demonstrate the strength of genetic algorithm (GA). The proposed research method is considered novel in the sense that it proves that GA-based method can be comprehensively utilized to determine multilayer feed-forward ANN architecture and training parameters such as number of hidden layers, number of neurons in hidden layer, training algorithm, activation function, initial weight, learning rate, momentum term and epoch size.

### **1.8 Thesis outline**

This thesis is written in a manner that follows the steps that were taken during the design and development of the ANN using GA-ANN approach. The thesis outlines are as the followings:

Chapter 1 outlines the basic concept of the ANN and GA design and operators, explains the research problem and objectives, identifies the relevant research questions and defines the scope of the thesis. The chapter, thereafter, provides a discussion on the outline and contributions of the thesis.

Chapter 2 introduces the general survey that covers the concept of ANN, GA, and GA-ANN. For this, a more comprehensive definition and description of ANN and back propagation algorithm is presented, followed by a discussion of the basic

elements, architecture and operations of ANN. The chapter then goes on to discuss various ANN applications that have been developed, as well as a summary of their advantages. A history of GA method for optimization purpose is then covered, followed by a discussion of basic operators, applications and their advantages. The related works of optimization of ANN using GA is described, including how they can be used to design neural networks using genetic algorithm. Then the chapter provides an extensive detailed survey related to every aspect of multilayer feed-forward ANN model development such as, data-preprocessing techniques, type of training algorithm, type of activation function, initial weight, learning rate, momentum term, and epoch size. For this, the recommended rules that have been used to determine the best parameters are presented.

Chapter 3 presents the methodology that has been used in this research. The first section of this chapter discusses the procedure involved in this research and focuses on the design methodology of an GA-ANN approach. The second section explains in detail every stage involved in designing the GA-ANN approach using the four different datasets, followed by a description of the parameters, functions, and techniques that are required in back-propagation training algorithm. The last section of this chapter explains how the experiment and analysis are accomplished in order to investigate the efficiency and accuracy of BP learning algorithm, and the prediction performance of the GA-ANN approach by using binary-encoded GA technique.

Chapter 4 describes the implementation of the new approach (GA-ANN) to determine optimum multilayer feed-forward ANN architecture and training parameters. The GA-ANN approach proposed in this chapter has been designed to be usable in any kind of science and engineering modeling domains, and is not limited to petroleum and energy domains only. To implement the GA-ANN approach, a large hierarchy of MATLAB codes was created, executed and they are presented in a format similar to algorithms.

Chapter 5 analyzes and discusses the results of the simulation and modeling of the GA-ANN obtained using MATLAB software. Further, this chapter proposes a benchmark of new GA-ANN approach and provides a comprehensive comparative study between GA-ANN approach and four relevant outstanding approaches in terms of generalization capability and training stability, prediction effectiveness, accuracy, optimality and simulation time. The findings indicated that the training algorithm,

activation function, number of hidden layer, number of neuron per hidden layer, learning rate, momentum term, and epoch size have a significant effect on the multilayer feed-forward ANN performance.

Chapter 6 presents the conclusions for this work, contributions, as well as some recommendations for potential future research directions that can be extended based on this research work.

## **1.9 Summary**

This chapter introduces the problem of optimizing the architecture and training parameters of a multilayer feed-forward ANN using GA. The chapter presents a brief introduction to ANN and GA. Known methods and algorithms in this domain are briefly introduced and presented. The problem of this research is clearly stated in this chapter; the objectives and scope of this thesis are thoroughly explained. The chapter ends with a description of the thesis contributions and outline.

## Chapter 2

### LITERATURE REVIEW

#### **2.1 Introduction**

In this chapter a general survey of the state of the art in ANN and GA research that are relevant to this thesis is presented. The intention is to provide a basic understanding of the subject matter. The chapter begins with a comprehensive definition and description of ANN, followed by a discussion of basic element, architecture and operations of ANN. The chapter then goes on to discuss various ANN applications that have been developed. A history of GA method for optimization purpose is then covered. This is followed by a discussion of basic operators, applications and the advantages of GA. Related works in the optimization of ANN using GA are described, including how they can be used to determine ANN architecture and training parameters. It is claimed in this chapter that there is a need to have a new approach that covers every aspect of ANN to improve prediction performance efficiency and accuracy of the network.

#### **2.2 Artificial neural networks (ANN)**

ANN is one of the major branches of artificial intelligence. Essentially ANN is a powerful and general technique that has been used to emulate the structure and biological functionality of the natural human brain.

The basic processing element of ANN is the neuron. A biological neuron receives inputs from other sources, unites them, then generally performs a nonlinear process on the outcome, and finally outputs the result. In particular, the natural biological

neurons have the same four fundamental elements (Figure 2.1). These are dendrites, soma, axon, and synapses.

Dendrites are hair-like extensions of the soma that proceed like input channels. Dendrites receive their input during the synapses of other neurons. The soma (or cell body) processes these inputs and then turns that processed value into an output that is sent out to other neurons through the axon and the synapses.

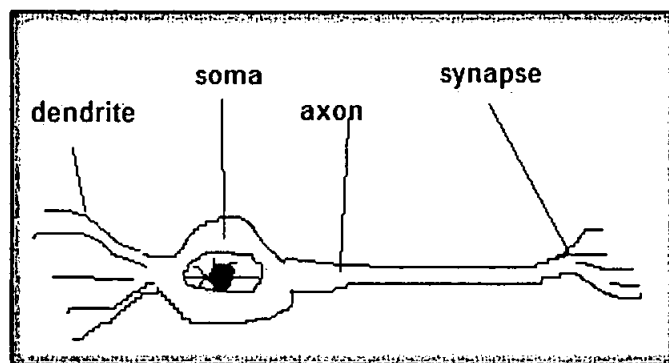


Figure 2.1 Structure of biological neuron (Caetano 2006)

The artificial neurons mimic the four fundamental functions of the biological neurons in human brain. In Figure 2.2, the input signals are represented by the symbol  $x(n)$ . Each of these inputs is multiplied by their corresponding connection weights, and these connection weights are represented by  $w(n)$ . Then the outcome products are simply summed up, fed throughout an activation function to produce a result and the output is represented by the symbol  $y(n)$ .

In recent years, ANN has gained more and more popularity for prediction and optimization purpose because of their wide range of applicability, their strong capability to map complicated and nonlinear problems, their high accuracy for learning, and for their high robustness.

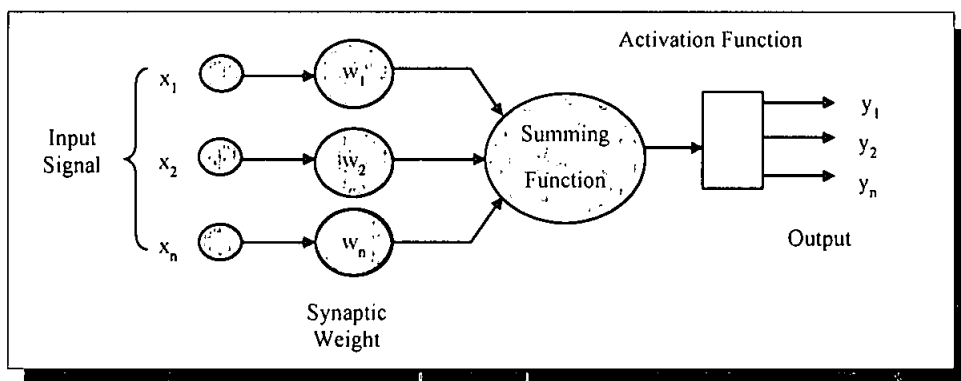


Figure 2.2 Model of Computing Neuron (Haykin 1994)

The basic principle of ANN modeling technique is represented schematically in Figure 2.3. ANN tries to map the relationship between input pattern and the corresponding output pattern during training process. The connection weights of network are adjusted, based on a comparison of the actual output and the target output, until the network output matches the target (Demuth and Beale 2005).

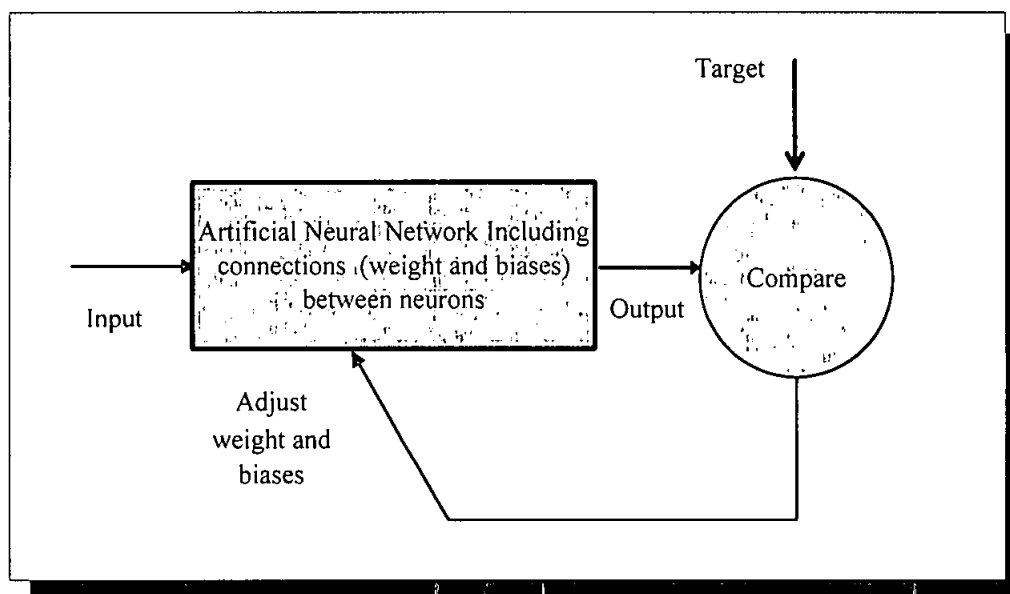


Figure 2.3 Principles of ANN algorithms (Demuth and Beale 2005)

### 2.2.1 Network architectures

ANNs can be categorized into three main types based on network architectures, namely, single-layer, multilayer and recurrent neural networks. The single layered networks have very limited application. Recurrent networks are most popular particularly with control systems. Multi-layered networks have been successfully used in several applications and have achieved satisfactory performances.

### 2.2.1.1 Single-layer neural networks

A single-layer neural network which was first proposed by Haykin (Haykin 1994) consists of input layer neurons that are interconnected to corresponding output layer neurons. In this case, the input layer is not counted as a layer since no computation is performed in this layer.

For single layer networks, we can define the following matrices:

$$X := [x(1), \dots, x(m)]' \in R \quad (2.1)$$

Where  $X$  is defined as the input matrix with  $m$  elements, while:

$$Y := [y(1), \dots, y(n)]' \in R \quad (2.2)$$

Where  $Y$  is the matrix of corresponding outputs with  $n$  elements, and  $w_{ij}$  represents the connection weights. A typical single layer neural network is shown in Figure 2.4

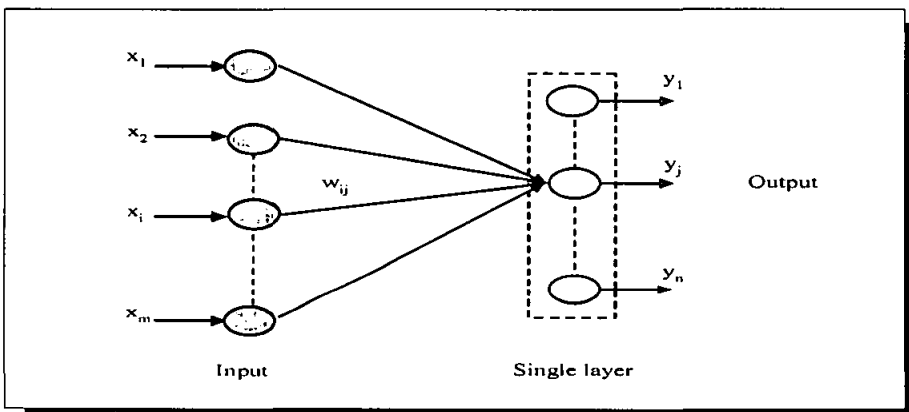


Figure 2.4 Atypical Single layer neural network (Demuth and Beale 2005)

### 2.2.1.2 Multilayer feed-forward ANN

Multilayer feed-forward ANN was first established by (Hinton, Rumelhart, Williams 1986). Among all suitable network architectures, feed-forward networks are

the most commonly used algorithms, primarily due to their simplicity from the viewpoint of structure and ease of mathematical analysis; FFNN has been applied successfully to various application domains. These are namely prediction, controlling, system modeling and identification, signal processing and pattern classification (Bilski 2005).

Generally, multilayer feed-forward ANN architecture as shown in Figure 2.5 demonstrates an arrangement of interconnected nodes called neurons by sets of connection weights organized into three groups called layers, i.e., input, hidden, and output layers containing  $M$ ,  $K$ , and  $N$  numbers of processing neurons (hidden neurons), respectively. The input layer neurons accept external information into the Multilayer feed-forward ANN model. Each hidden layer is a group of neurons that receive their input from the previous layer (input or other hidden layer), then perform some mathematical processing (combination and transfer) and feed an outcome to the nearest neurons in the next layer (output or other hidden layer). The third layer represents the output neurons that process the transferred values from the last hidden layer.

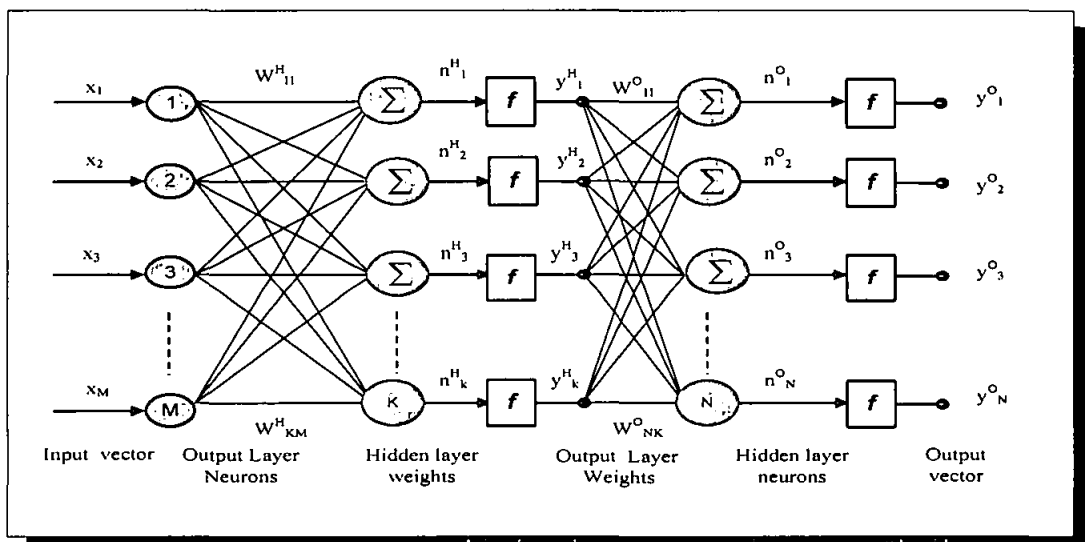


Figure 2.5 A schematic diagram of multi-layered feed-forward neural network with back-propagation training algorithm (Demuth and Beale 2005)

From Figure 2.5, the connection weights between input and hidden neurons, and between hidden and output neurons are represented by  $W^H$  and  $W^O$ , respectively. Whereas,  $y^H$  and  $y^O$  indicate the outputs vector from hidden and output layers,



respectively. The weighted input ( $W$ ) is the argument of the activation function  $f$ , which performs the scalar process of output  $y$ . Then the activation function net input is a summing function ( $n^H$  or  $n^O$ ) which is the summation of the weighted input ( $W^H$  or  $W^O$ ).

In practice, most ANN researchers rely on the application of multilayer feed-forward ANN because of their many characteristics, among them, they do not need a user-defined problem solving algorithm (as is the case with conventional programming language), but as a substitute they “learn” from example or pattern, a great deal like human brain. Therefore, the FFNN has an inherent generalization capability. This means that they can recognize and respond to patterns that are similar but not identical to the ones with which they have been trained (G.-C. Vosniakos P.G. Benardos 2007).

### 2.2.1.3 Recurrent neural networks (RNNs)

Recurrent neural networks (RNNs) such as Hopfield network (Hopfield 1982) and (Hopfield 1984) are usually used to model a variety of nonlinear dynamical behaviors (Siegelman and Sontag 1995), and can be a powerful solution for nonlinear problems (Seidl and Lorenz 1991). In recent years, a large number of research activities have studied the capabilities and limitations of RNNs applied to subjects associated with pattern recognition and control. On the other hand, the use of RNNs is not as extensive as multilayer feed-forward ANN because of their complexity through developing learning algorithms.

In practical, multilayer feed-forward ANN can be used to model static nonlinear systems, and can have either single or multilayer network architecture. However, recurrent networks (RNNs) are dynamic networks, and their architectures are fundamentally different from the static ones, because they include feedback (Mandic and Chambers 2001).

Moreover, RNNs are a neural network in which the output of some neurons is fed back as an input to some other neurons. The internal feedback allows the network to

temporally memorize the behaviors of some neurons as states. Not only by the inputs, but also by the previous states, the network can output in a context-dependent manner.

Two major types of recurrent neural networks are presented here as examples, Elman Recurrent Neural Network (ERNN) (Elman 1990) and Jordan Recurrent Neural Network (Jordan 1986). Elman RNN is a network, which in principle is set up as a normal feed-forward network. This means that all neurons in one layer are connected with all neurons in the next layer. An exception is the context layer, which is a special case of hidden layer. The architecture of an Elman RNN is shown in Figure 2.6. The neurons in the context layer (context neurons) hold a copy of the output of the hidden neurons. The output of each hidden neuron is copied into a specific neuron in the context layer. The value of the context neuron is used as an extra input signal for all the neurons in the hidden layer, one time step later. Therefore, the Elman network has an explicit memory of one time lag (Elman 1990).

In Jordan recurrent networks the output feeds back into the hidden layer with a time delay. The output of the previous periods becomes input in the current period as illustrated in Figure 2.7. Consequently, the output of the existing period takes the history of past outputs, which contains past values of inputs.

RNN, however, necessitates complex computational processes that can only be achieved by more powerful software. Similar to regular feed-forward neural networks, RNN can be trained with gradient descent back-propagation and optimization methods such as conjugate gradient, quasi-Newton, and Levenberg–Marquardt.

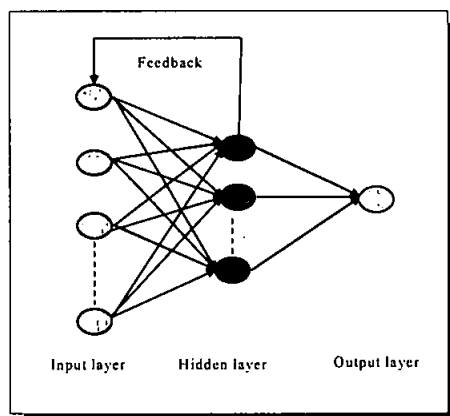


Figure 2.6 Elman recurrent networks (Elman 1990)

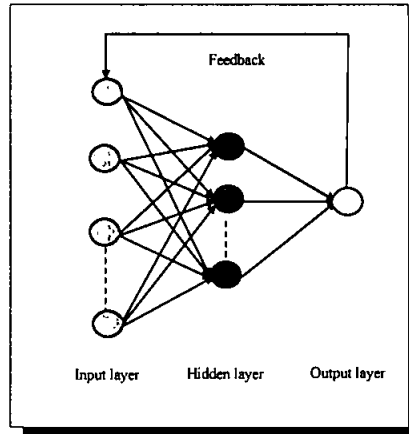


Figure 2.7 Jordan recurrent networks (Jordan 1986)

### 2.2.2 Back-propagation algorithm (BP)

The major performance characteristics of ANN are capability to learn, corresponding operation and distributed memory, finally leading to error tolerance. According to these advantages, Back-propagation (BP) is a recognized representative of all gradient descent algorithms, which is a commonly used technique for neural network learning in various areas of application.

A BP is the best-known ANN that uses a supervised learning technique and feed-forward architecture that maps the complex and nonlinear relationship between inputs (independent variables) and outputs (dependent variables) through the training process with the given training datasets (Jung and Hong 2007).

The input pattern is presented at the network and the feed forward phase is first executed. Hence, an error between the actual output from the BP network and the desired output, which is given from the training data set, are computed. These errors are sent back in reverse direction and used to determine connection weight changes in the BNN according to the back-propagation of errors rule. The training of the BP network consists of three main processes, which are, feed forward of the input data, back-propagation of error, and updating of the connection weights. These three main processes are detailed in the next section.

### 2.2.2.1 Feed forward process

In the multilayer feed forward back-propagation ANN, each neuron in the layer receives an input value from the training dataset, and then each of these input variables are multiplied by corresponding connection weight values. The sums of weighted inputs are computed and its activation function is applied, that scales the output to a fixed range of values, i.e., (0,1) or (-1,1). Then the output value is sent to all neurons in the next layer (hidden or output layer) (Laurene 1994). Steps for the feed-foreword process can be expressed as follows (Laurene 1994):

Step 1: Each input neuron  $x_i$  ( $i=1, 2,..n$ ), receives its input value from training pattern.

Step 2: Each hidden neuron  $h_j$  ( $j=1, 2,..p$ ), receives its input value from input neurons:

(i) Sum the weighted input  $net_{h_j}$  as follows:

$$net_{h_j} = (v_{0j} + \sum_{i=1}^n x_i v_{ij}) \quad (2.3)$$

Where  $v_{0j}$  is bias on hidden layer  $j$  ( $j = 1, \dots, p$ ),  $X$  is the input training vector:  $X = \{x_i, i = 1, \dots, n\}$ , and  $V_i$  is the input-to-hidden weights vector:  $V_i = \{v_{ij}, j = 1, \dots, p\}$ .

(ii) Passé it to the activation function to calculate output values as follows:

$$h_j = f(net_{h_j}) \quad (2.4)$$

Where  $f$  is the binary sigmoid activation function:  $f(x) = 1/(1 + e^{-x})$ .

Step 3: Each output neuron  $y_k$ , has been received its input from hidden neuron:

(i) Sum the weighted input as follows:

$$net_{y_k} = (w_{ok} + \sum_{j=1}^p h_j w_{jk}) \quad (2.5)$$

Where  $w_{0k}$  is the bias on output layer  $k(k = 1, \dots, m)$ ,  $h_j$  is the hidden neuron output, and  $W_j$  is the hidden-to-output weights vector:  $W_j = \{w_{jk}, k = 1, \dots, m\}$ .

(ii) Pass it through activation function to calculate output values  $y_k$  as follows:

$$y_k = f(\text{net}_{y_k}) \quad (2.6)$$

Where  $\hat{f}(x)$  is the derivative of the sigmoid activation function:  $\hat{f}(x) = f(x)[1-f(x)]$ .

### 2.2.2.2 Back-propagation of error process

The back-propagation error is the difference between the actual network output and the corresponding desired output. During the back-propagation of the errors calculation process, the learning rate ( $\partial$ ) determines the amount of the connection weight changes. Steps for the back-propagation of errors can be expressed as follows (Laurene 1994):

Step 1: For each output neuron  $y_k(k = 1, \dots, m)$ :

(i) Compute the error term  $\delta_k$  as follows:

$$\delta_k = (\text{target}_k - y_k) f'(\text{net}_{y_k}) \quad (2.7)$$

Where Target is the output target vector:  $\text{Target} = \{\text{target}_k, k = 1, \dots, m\}$  and  $y_k$  is actual output and  $\hat{f}(x)$  is the derivative of the sigmoid activation function:  $\hat{f}(x) = f(x)[1-f(x)]$ .

(ii) Compute the weight correction term  $\Delta w_{jk}$  ( $j = 1, \dots, p$ ) and  $\Delta w_{0k}$  as follows :

$$\Delta w_{jk} = \partial \delta_k h_j, \Delta w_{0k} = \partial \delta_k \quad (2.8)$$

Where  $\delta_k$  is the error term at output layer  $y_k$  and used to change weight  $w_{jk}$ , and  $\partial$  is the learning rate that defines the amount weights are changed during training cycle.

Step 2: For each hidden neuron  $h_j$  ( $j = 1, \dots, p$ ):

(i) Compute the error term  $\delta_j$  as follows:

$$\delta_j = \left( \sum_{k=1}^m \delta_k w_{jk} \right) f'(net_{h_j}) \quad (2.11)$$

Where  $\delta_j$  is the error term at hidden layer  $h_j$  and used to change weight  $v_{ij}$ , and  $\hat{f}(x)$  is the derivative of the sigmoid activation function:  $\hat{f}(x) = f(x)[1-f(x)]$ .

(ii) Compute the weight correction term  $\Delta v_{ij}$  and  $\Delta v_{0j}$  as follows:

$$\Delta v_{ij} = \partial \delta_{jxi}, \Delta v_{0j} = \partial \delta \quad (2.12)$$

Where  $V_i$  is the input-to-hidden weights vector:  $V_i = \{v_{ij}, j=1, \dots, p\}$ ,  $\delta_j$  is the error term at hidden layer  $h_j$ , and  $v_{0j}$  is the bias on hidden layer  $j$  ( $j = 1, \dots, p$ ).

### 2.2.2.3 Update of connection weights process

Learning occurs when the network weights are updated by the generalized delta rule. All connection weights are changed in each layer. Steps for the weights updating can be described as follows (Laurene 1994):

Step 1: Each output neuron  $y_k$  ( $k = 1, \dots, m$ ):

(i) Changes its bias and weights as follows:

$$(2.13)$$

Where  $W_j$  ( $j=1, 2, \dots, p$ ) is the hidden-to-output weights vector:  $W_j = \{w_{jk}, k = 1, \dots, m\}$ .

Step 2: Each hidden neuron  $h_j$  ( $j = 1, \dots, p$ ):

(i) Changes its bias and weights as follows:

$$v_{ij}(new) = v_{ij}(old) + \Delta v_{ij} \quad (2.14)$$

Where  $V_i$  ( $i=1, 2, \dots, n$ ) is the input-to-hidden weights vector:  $V_i = \{v_{ij}, j=1, \dots, p\}$ .

The *learning* or *training* of ANN is a process of adjusting the connection weights. Figure 2.8 shows the schematic diagram of error back propagation. The error for each neuron is the squared difference between the desired output and the actual output.

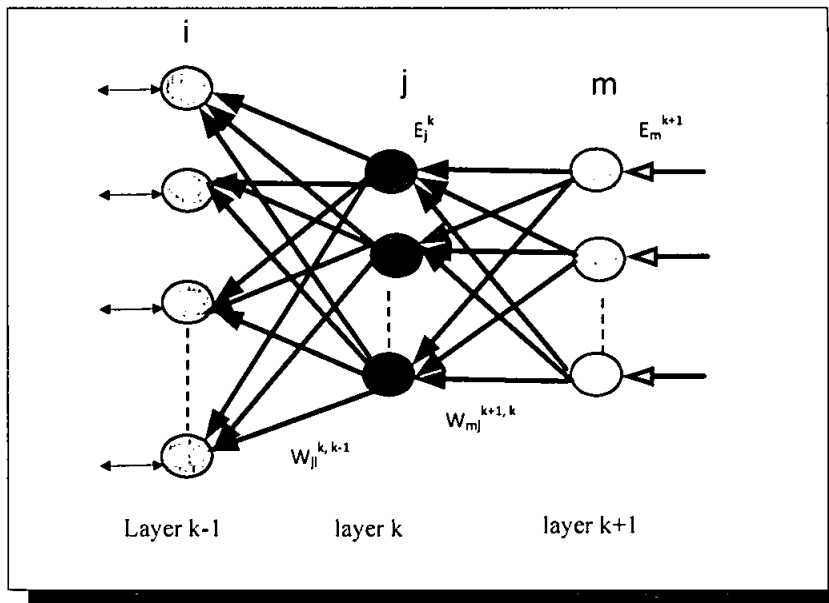


Figure 2.8 Schematic diagram of error back propagation (Imdat and Yasar 2009)

#### 2.2.2.4 The BP training algorithm

The BP training algorithm processes have been employed in this study is presented below:

- Step 1: The weights have been initialized.
- Step 2: Steps 2–3 have been repeated until termination condition is false.
- Step 3: For each training dataset pair (input and desired output vector):
  - (i) Feed-forward has been processed.
  - (ii) Back-propagation of error has been processed.
  - (iii) The weights have been updated.
- Step 4: The termination condition has been tested.

The flowchart of the back-propagation training algorithm process is shown in Figure 2.9.

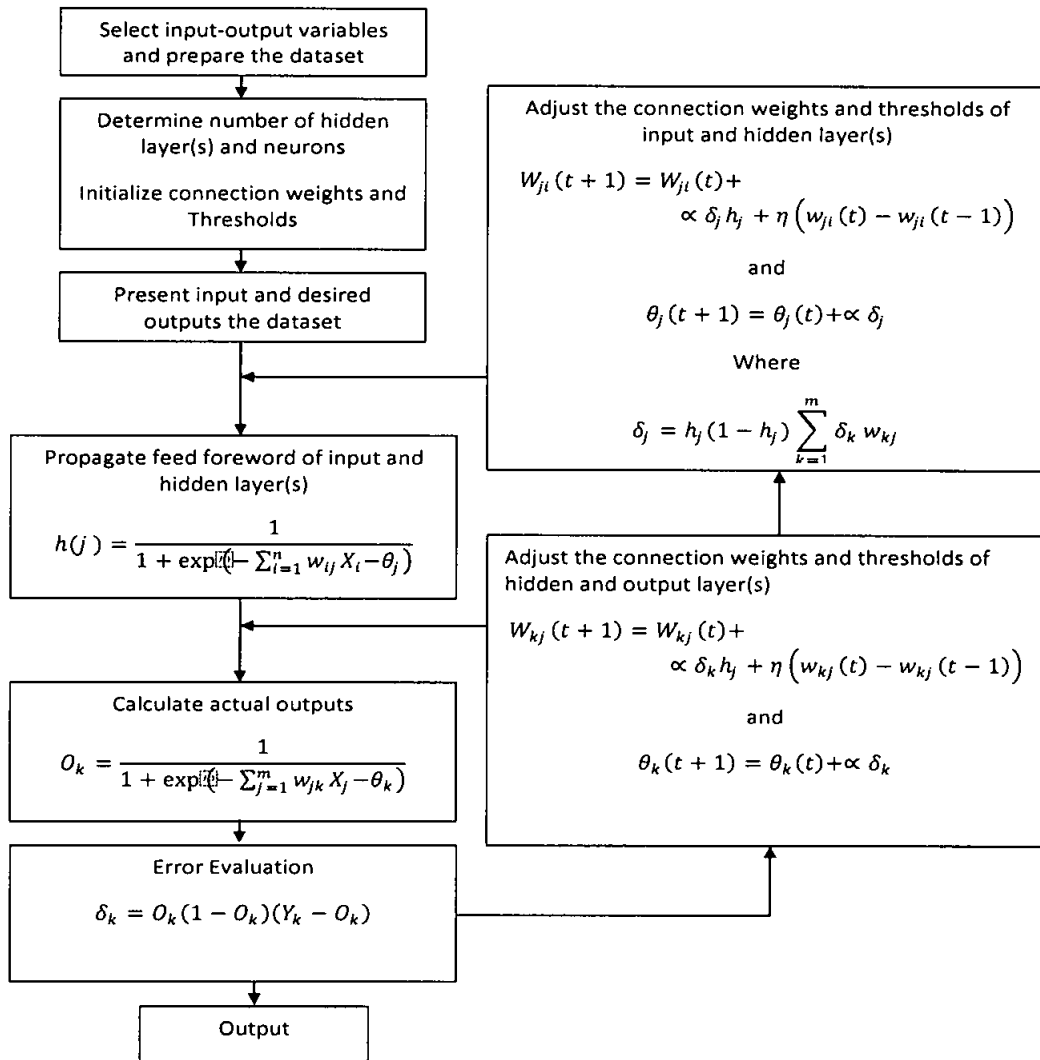


Figure 2.9 Flowchart of the back-propagation training algorithm process (Imdat and Yasar 2009)

### 2.2.3 Network training, validation and testing

The success of the ANN model development is basically dependent on three important steps: the learning or training process, testing and validation process. During the training process several activities are performed, a set of input-output patterns is repeated to the ANN, propagating the input patterns through the architecture, connection weights of all interconnected neurons are adjusted until the differences between actual outputs and the desired outputs are minimized to be in permissible limit. After an acceptable training, the values of the connection weights represent the state of knowledge of the ANN.



*Training or learning* paradigm concerns about what information is fed to ANN. Learning paradigm in ANN is often classified into three basic types based on whether outputs are provided or not, namely supervised, unsupervised or self-organization, and reinforcement learning or graded.

In type of supervised learning, a *supervisor* or sometimes called *teacher* provides the network with a pair of input and target pattern called training pair or set. A network computes the outputs for a given set of inputs and compares these computed outputs with target outputs (Adiel 2001). The difference between the calculated output of the network and the target output serves as an error evaluation, and is used in correcting synaptic connection weights. This connection weights are adjusted regularly, through updating them at each step during the learning process, therefore the error between the network output and corresponding desired output is minimized. A good example of supervised training type is back-propagation network, and is the most popular training method in ANN, according to the literature.

In an unsupervised learning, sometimes called self-organization, the training set is composed of only inputs through learning process, therefore no target output or desired response is provided. In this case, the error information cannot be used to improve performance of the network because the desired response is unknown. Thus, the network adjusts the connection weights in a way that similar inputs yield similar outputs. The purpose of unsupervised learning is to learn to set together patterns that are similar for a given training set. Some good examples of the unsupervised learning are Kohonen's self-organizing feature maps and Hebbian learning (Kohonen 1988b).

Reinforcement learning and error correction learning are alike, in which connection weights are reinforced for appropriately achieved actions and punished for poorly achieved actions.

The difference among the two types of learning is that error correction learning uses more specific error information by utilizing the error values at each output unit, whereas, reinforcement learning type uses non-specific error information to determine the performance of the network. In error correction learning, an entire vector of values is used for error correction, whereas only one value is used to describe the

performance of the output layer's through reinforcement learning type. This type of learning is ideal in some areas such as prediction and control where specific error information is not available (Barto 1992; Adiel 2001).

In validation phase, the ANN is subjected to input data unseen during the training process, and adjustments are made to make the ANN model more reliable and robust. Besides, it is used to examine network generalization and to halt training when generalization stops improving. Generalization refers to the neural network building reasonable outputs for inputs not encountered through training (Haykin 1994).

In general, mean square error (MSE), mean absolute error (MAE), and sum square error (SSE) which are computed between the target output and the actual network output are typical used as the fitting criteria to measure the model validity

Testing phase has no effect on training, and it provides an independent evaluation of network performance during and after the training phase.

#### **2.2.4 Applications of ANN**

ANNs are being widely used to model complex real-world problems. This could be accredited to the fact that these networks attempt to emulate the abilities of human brains (Mukta and Usha 2009).

The ANN applications are not limited to a specific application area as it extends across a broad variety of domains. These applications have been organized into various categories, for example, engineering, science, manufacturing, marketing, finance and accounting, health and medicine, pattern recognition and other general applications.

##### *2.2.4.1 Engineering*

ANNs are powerful computational techniques that have resulted from the work in the area of engineering. (Rafiq et al. 2001) presented a practical example of a reinforced concrete slab design using three types of ANNs mainly, multi-layer perceptron

(MLP), radial basis network (RBF) and normalized RBF (NRBF). The results show that the MLP and the NRBF networks performed equally well. The RBF demonstrated a poorer performance. The NRBF has the benefit of a faster training. (Serkan et al. 2002) developed a new approach of ANN to select optimum bit using real rock bit data for several wells in a carbonate field. Their new approach of the neural networks provided satisfactory results. (Molga 2003) investigated new aspects of neural modeling of chemical reactors, the performance of two neural models: global neural model (GNMs) and a hybrid neural model (HNM) were compared. He found that GNMs have superior interpolation capabilities, but quite weak capability to knowledge generalization, but HNM models have better capabilities to generalize knowledge.

#### 2.2.4.2 *Science*

Much of the research on neural networks have focused on science problems, with special attention to animal, environmental, agricultural, etc. (Fernández et al. 2006) focused on applications to intelligent data analysis in the field of animal science using two traditional neural networks: time series prediction and clustering. The results showed that the quality and accuracy of the ANN models was essential to increase farm returns. (Faria P. M. Ferreira, Ruano 2002) addressed neural network models in greenhouse air temperature prediction for environmental control. Several on-line training and on-line learning feed-forward methods for prediction were used and had achieved satisfactory performance and acceptable prediction errors, but the results of a new algorithm based on a Levenberg–Marquardt method achieved the best results in terms of parameter convergence, error performance, and with smaller computational costs. (Morimoto et al. 1997) presented a new control technique, as well as neural networks and genetic algorithms, for realizing the optimal control of the fruit-storage process. The results recommended that the storage process should be treated as a dynamic process, and the proposed approach had been useful for the optimization of such a control process.

#### 2.2.4.3 *Manufacturing*

ANNs have been applied successfully to several manufacturing areas. (Maguire, McGinnity 2009) reported the application of neural networks techniques to improve the downstream performance prediction within the manufacturing environment. The authors used the manufacture of hard disc drives as a case study. Neural networks techniques were used to control most of the hard drive processes, i.e., read and write head to the recording media and the general manufacture of the hard disc drive. The results showed that satisfactory downstream prediction accuracy can be achieved using neural networks approach. (Magdy and Houshang 2009) presented an application of neural network to fault-tolerant control (FTC) of automated sequential manufacturing systems (ASMS) theme to sensor faults. The result highlighted that the proposed application was effective for a real simple plant, in identifying the sensor faults, and reducing the complexity of the hardware redundancy.

#### 2.2.4.4 *Marketing, finance and accounting*

ANNs have been used by various researchers and by the author for modeling and predictions in the field of marketing, finance and accounting. (Shuliang 2000) developed a hybrid intelligent system for developing marketing strategy, artificial neural networks (ANNs), expert system, and fuzzy logic is coupled to support the procedure of marketing strategy development. The empirical experiments show that the hybrid intelligent system is helpful and useful for developing marketing strategy.

Neural networks have been focused on finance problems, with special attention to bankruptcy prediction and credit scoring. (Chih-Fong and Jhen-Wei 2008) investigated the performance of two types of neural network classifier: a single classifier and multiple classifiers using three different datasets for the bankruptcy prediction and credit scoring problems. The results implied that the neural network single classifier was more suitable than the multiple for the bankruptcy prediction and credit scoring domains.

(Chrysovalantis et al. 2007) proposed an application of artificial neural network for the identification of qualified audit opinions. The results expressed the highly descriptive power of the ANN model in highlighting qualifications in audit reports.

#### *2.2.4.5 Health and medicine*

ANNs have been applied successfully to model various aspects of health and medicine. (Sheppard et al. 1999) described the use of neural networks approach to predict patients who risk developing cytomegalovirus disease after renal transplantation. The results showed significant improvement on current methods and supported the principle that the neural networks with back propagation training algorithm may respond well to analysis of renal transplantation data. (Yu-Chuan et al. 2000) developed and compared three different mathematical models, mainly logistic regression model, a multi-layer perceptron (MLP) neural network and a radial-basis-function (RBF) neural network for surgical decisions on traumatic brain injury patients. The results showed the possibility of neural networks as the mechanism for traumatic brain injury (TBI) decision support system based on clinical databases. The results also suggested that neural networks may be a better solution for difficult, nonlinear medical decision support systems than traditional statistical techniques such as logistic regression.

(Reeti et al. 2006) presented neural networks prediction model for Alzheimer's disease using longitudinal data collected through multiple clinic visits. The results showed that the new model can incorporate this type of longitudinal information.

#### *2.2.4.6 General applications*

In recent years, ANN technique has been applied successfully for general applications, such as prediction (Hee-Yeal and Sung-Yang 1997), controlling (Takayuki et al. 2007), recognition (Steven et al. 1995), last but not least in classification (Ganesh and Abdesselam 2003).

### **2.2.5 ANN advantage**

ANN is the main branch of artificial intelligence which is mostly useful in situations for which rules are either not known, or they are extremely difficult to determine. Some of the significant advantages of ANN can be listed as (Chen et al. 2003; Stefan 2003; Zhang and Qi 2005):

- ANNs can predicate a huge class of functions with a high degree of precision, because ANN provides a parallel processing of the information from the data.
- ANN can learn and generalize by examples, thus ANN can construct significant solutions to problems before they are tested for their “inference” capability on unknown situation of the problem. They can, consequently, identify new object previously untrained.
- ANN can give results for problems that do not have an algorithmic solution; moreover, ANN provides solutions for which an algorithmic solution is too difficult to be found.
- ANN offers generalization capability; therefore, they can correctly process information that mostly looks like the original training data.
- ANN demonstrates mapping capabilities, that is, it can map input pattern to the corresponding output pattern.
- ANN can produce information in parallel form with high speed, and in a distributed manner.

### **2.3 Genetic algorithm (GA)**

The GA, which was first presented by Professor John Holland in the early 1970s (Holland 1975), is becoming a powerful tool for optimizing functions. GA is a global search method based on Darwinian biological evolution principle. The GA differs from more conventional optimization methods since it involves a search from a population of solutions, and not from a single point; and it can avoid the convergence to suboptimal solutions.

GA is a structured probabilistic algorithm, the initial population is randomly generated and gradually evolves towards a population that is expected to contain the

better solutions by applying so-called ‘genetic operators’, which are selection, crossover and mutation. GA is modeled based on genetic processes that are biologically natural. In particular, genetic operators are used to produce offspring in the next generation that differ from their parents, however, still maintaining characteristics of the parents.

In general, the mechanics of GA consists of the following basic stages: solutions encoding, determination of fitness function, population initialization, and finally perform GA operators including selection, crossover and mutation. (Figure 2.5)

Encoding denotes representing the potential solutions, which contain a set of decision variables, as strings of codes. Various formats can be used to encode the decision variables, i.e., binary, real number, letter, etc. However the binary is the most popular encoding form. By definition, the coded variables are called genes, and the actual values (i.e., either 0 or 1) of genes are called alleles. For example, if a gene represents colors, then its alleles are blue, yellow, and green and so on. A chromosome is a solution that is created by connecting the pieces of genes into a string of fixed length. Figure 2.10 shows the basic step of GA procedure.

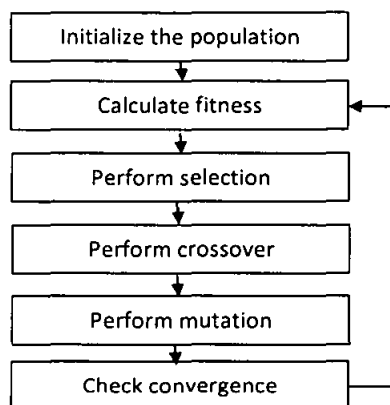


Figure 2.10 GA procedure

Following from the solutions that have been encoded, the objective function is required to determine the best solution. The objective function is the value of an individual solution that can be accomplished through computer simulation or lookup table. This stage launches a relationship between the variables and the function. The goal can be either to maximize or minimize the objective function.

After the solutions are encoded and the objective function is defined, we now can proceed to the evolution process. First, an initial population of coded solutions or chromosome is created randomly or through pre-learned knowledge. Then, the population undergoes an iterative process of evaluation and reproduction. Evaluation assigns each chromosome with a fitness value according to the objective function. After evaluation, chromosomes of the new generation are created by applying reproduction operators on the old generation.

### **2.3.1 GA operators**

Selection, crossover and mutation are the most commonly used GA operators (Ali et al. 2010; Jiangfeng et al. 2010). Generally, the chromosomes in the population are encoded as strings of real or binary form. This implementation of chromosomes is called encoding scheme.

#### *2.3.1.1 Selection*

Selected chromosomes are subjected to mutation and crossover. With a given probability, the chromosomes are to be selected from the current population, and cloned chromosomes are created. The selection process of chromosomes is based on their fitness relative to the current population i.e., the stronger chromosomes will have a higher probability of being copied (Taho et al. 2006). In this study, the fitness sometimes called objective is a function of the ANN model's response. Then the selected chromosomes are subjected to mutation and crossover.

#### *2.3.1.2 Crossover*

Crossover is the major genetic operator for producing new chromosomes, which combines the two characteristics of parent information to form offspring. The purpose of crossover operator is to generate new chromosomes that are definitely different from their parents, but contain some of their parents' characteristics (Mousavi et al. 2006). In other words, two good quality strings (parents) share their good qualities to



generate better strings (Childs) than before. The new strings (offspring) may be bad, which will die off in the next generation. In the crossover operator, the selection of two strings is performed randomly but proportional to their fitness values in the mating pool, and then an arbitrary position along the two strings is selected, further than which the crossover takes place. The part of the chromosome string between the selected position and end position is then exchanged, therefore creating two offspring. The crossover methods, which are one-point or two-point crossover, are commonly used (Nan et al. 2003; Nicolás et al. 2006). The crossover operator is show in Figure 2.11.

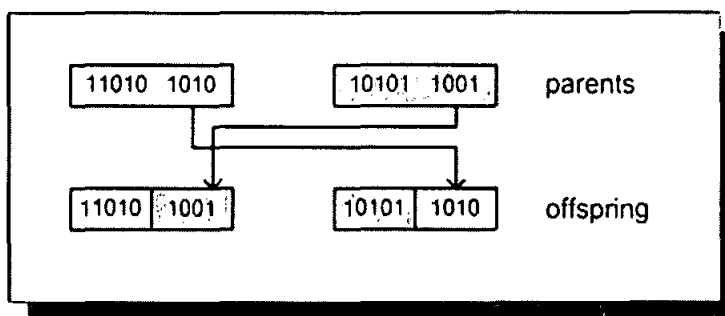


Figure 2.11 Crossover operation

### 2.3.1.3 Mutation

During genetic generation process, sometimes a desired bit misses at a particular position in the string. This bit may be critical to produce good quality offspring. The selection and crossover operators may not create the critical bit at the particular position; however, the mutation operator will take care of this trouble. The most common way of mutation is to take a bit from a chromosome and alter it with some predetermined probability (Figure 2.12). As mutation rate is too small in natural evolution, the probability at which the mutation operator is applied is set to a very low value (Xueqiang et al. 2007).

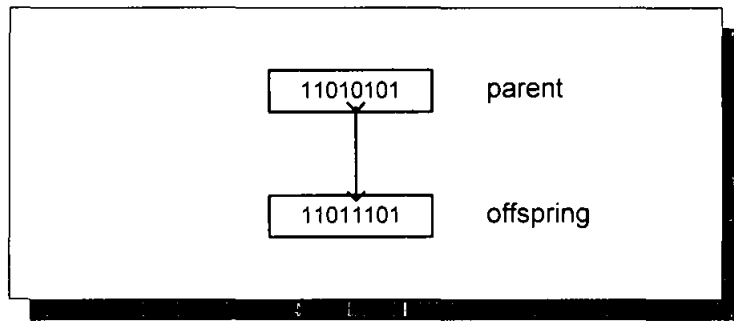


Figure 2.12 Mutation operator

### 2.3.2 GA applications

Since its introduction, GA has been adopted in several areas. In the area of engineering design, (Afshin et al. 2009) used GA to optimize core configuration design of research reactors; (Onwubolu and Mutingi 2001) applied GA to cellular manufacturing systems, and (Ali and Ali 2008) introduced GA to steam turbine model. In the area of science, (Srinivasaa et al. 2007) optimized data mining applications using GA; (Pezzellaa et al. 2008) presented flexible GA for job-shop scheduling problem. In the area of finance and accounting, (Kyung-Shik and Yong-Joo 2002) applied GA in bankruptcy prediction modeling. In the area of health and medicine, (Shital and K. Andrew 2007) applied GA with data mining to further enhance the classification accuracy of cancer detection; (Jinn-Yi and Wen-Shan 2007) used GA to improve the quality care of a hospital emergency department. In general applications, (Peter and Mirzaei 2003) presented GA to improve the accuracy of image reconstruction; (Wei et al. 2005) proposed two methods based on a GA and a nonlinear algorithm for standardization of image characteristics in digital mammography; (Romero and Carter 2001) used GA to search for reservoir characterization.

### 2.3.3 GA advantage

The important advantages of adopting GA instead of traditional optimization methods are listed below (Ferentinos 2005; Benardos and Vosniakos 2007):

- The GA operates on a coded form of the problem's parameters, not the parameters themselves.

- GA is capable of searching in very large solution spaces efficiently by providing a lower computational cost, because they use probabilistic transition methods instead of deterministic ones.
- GA is easily hybridized (or work cooperatively) with other optimization methods.
- GA does not require a deep mathematical knowledge about the problem at hand, since GA is automated, therefore it necessitates much less human attempt than trial-and-error.
- GA can optimize numerous objective functions (multi-objective optimization), thus it provides a list of solutions, not a simple solution.
- GA can effectively find a global minimum, yet on a very complex and complicated objective function.
- GA is a powerful tool in applications of feed-forward neural networks.

## **2.4 GA in ANN design**

Due to the great potential of hybridizing GA and ANN, this area of research has attracted the interests of many researchers. There are several studies on how GA can be used to determine the best parameters of ANNs, optimize its structure and to select optimal process parameters from science to business and engineering domains, although there is still no general method to design ANN. In this research, we will mainly concern with the optimization of ANN designing and training parameters in order to determine optimum ANN parameters through Genetic Algorithm.

As a fact, experiments have demonstrated that the combination of ANN-GA outperforms GA and ANN from the point of view of satisfactory solution. Although the very long run of back-propagation of ANN gives higher precision solutions(Kitano 1990). Figure 2.13 shows the search speed of back-propagation of ANN, GA, and combination of ANN-GA in term of the error and computation costs (cycle).

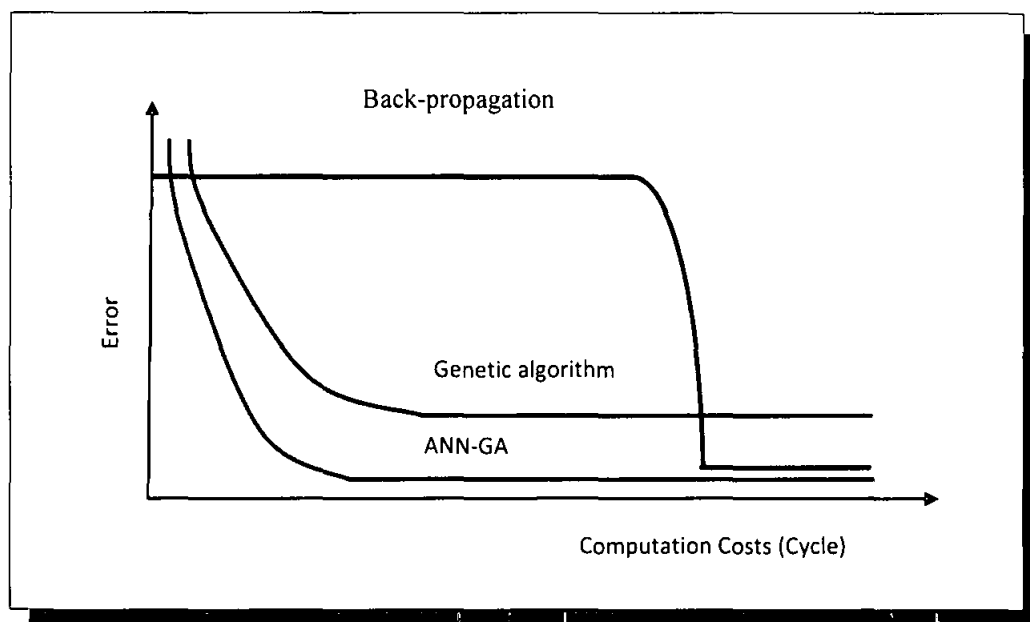


Figure 2.13 Search speed according to (Kitano 1990)

#### 2.4.1 GA to optimize input variable of ANN

Several studies have made to optimize input variables of ANN. (Mok et al. 2001) developed NN-GA approach consists of two parts: an ANN prediction and a GA part, the author found that the time to generate initial process parameters can be significantly reduced. (Somnath et al. 2004) integrated ANN with support vector regression and genetic algorithms (ANN-GA, SVR-GA) to obtain the optimal values of process input variables that minimize a specified objective function, the author reported that the author found that a significant improvement had been achieved while comparing the results of ANN-GA with SVR-GA. Feed-forward ANN approach have been used to investigate the effects of the input variables on the performance of the reactor (Istadi and Nor Aishah 2007). The GA method was applied to optimize the input space of the ANN model, to achieve the best results, the input and output variables were selected carefully, multi input and multi output (MIMO) ANN model that contains 4 inputs and 5 output variables was chosen to conduct the experiments. The results of the ANN model showed that the correlation coefficient ( $R$ ) had been increased, while the mean square error had been reduced with increased number of neurons in the hidden layer. A combination of ANN and GA approach has been

developed to optimize the release of emission from the palm oil mill (Ahmad et al. 2004), the results showed that data of any operation condition can be analyzed, optimized and predicted by the developed GA-ANN approach. (Cook et al. 2000) developed a hybrid NN-GA system to determine the process parameter values that are needed for different conditions. NN model was constructed to predict the value of a critical strength parameter (internal bond) in a particleboard manufacturing process, based on process operating parameters and conditions. While a GA method was applied to the trained neural network model in order to determine the optimum process parameter values that would produce the desired strength for the parameter for known operating conditions. The results showed that the ANN-GA system allowed the manufacturer to rapidly determine the values of critical process parameters required to attain acceptable levels of board strength, based on current operating conditions and the stage of manufacturing. (Hasan et al. 2005) created a feed forward ANN predictive model for warpage value. The ANN model was validated for capability of predictive task and then coupled with an effective GA in order to determine the optimum process parameter values. The results indicated that the initial model of the bus ceiling lamp base of the warpage was minimized significantly by 46.5% through using genetic algorithm. (Nasseri et al. 2008) integrated a feed-forward ANN with GA to overcome the complexity in modeling of rainfall forecasting. GA was applied to detect effective inputs and the best input combination(s) for intelligence prediction. Results of the study demonstrated the ANN network with the input parameter selection, when integrated with GA, performed better than a similar study of using ANN only. (Adineh et al. 2008) presented a suitable method for attaining the optimum operational parameters in a FAF CO<sub>2</sub> laser, leading to maximum laser output power by coupling ANN and GA. First, a number of experimental data were used as the input variables for the ANN model. Then the best-trained network was linked to the GA as a fitness function to determine the optimum operational parameters. The result demonstrated that the computed laser power was maximized by 33%, and the measured value was maximized by 21% in an experiment as compared with a non-optimized case.

#### **2.4.2 GA to optimize the structure and training process of ANN**

This part discusses the state of the art of research carried out in the optimization of ANN structure and training process using GA. (Ferentinos 2005) developed two feed-forward neural networks, first for a fault detection system model and second for a predictive modeling system, while GA was adopted to determine optimum NN parameters. A back-propagation algorithm for the training of the NN was used as minimization algorithm. Mainly, GA was used to optimize the selection of the minimization algorithm used by the back propagation algorithm, the number of neurons per hidden layer, and the types of activation functions of the hidden layer and of the output layer. The results showed that a satisfactory ANN performance had been achieved by using the fault detection and predictive models. In addition, the GA system had successfully replaced the problematic trial-and-error method that was used usually in this task. To solve the time consuming problem of training process, while most networks are calibrated using the trial-and-error procedure, (Goloka 2006) presented GA method to search for the optimal geometry and values of internal parameter of a multilayer feed-forward ANN with back-propagation algorithm (BPNN), and a radial basis function network (RBFN). Furthermore, to examine the efficiency of the combined GA-ANN method, a published experimental dataset of cross-flow membrane filtration was used. The results showed that the GA-ANN method predicted the permeate flux decline more accurately than the ANN combined with a trial-and-error procedure. It was also noted that the efficiency of the ANN model prediction was significantly affected by the sample size included in the training dataset; either the largest or smallest sample in the training and validation data set affected the performance of the ANN model. Thus, the prediction of the ANN model using small dataset will deviate significantly if the ANN model had not been optimized and well trained. Therefore, the data division strategy for training, validation and testing phases is a very important issue. A new radial basis function (RBF) network training based on a specially designed GA as a new method for extracting valuable process information from input and output data has been proposed (Haralambos et al. 2004). GA was used mainly to optimize ANN parameters, i.e., structure of the network, the weights and the centers of hidden neurons. To verify the effectiveness of the method, the developed model was illustrated through two

different datasets. Firstly, a simulated data from a Continuous Stirred Tank Reactor (CSTR). The results of the new method in this case study had shown that for the prediction of the first output, the best network structure was found to have 28 hidden neurons, while the best network structure for the prediction of second output consisted of 36 hidden neurons. Results from the second dataset from an industrial Kamyr digester showed that the new RBF achieved the best network structure with 18 neurons and the training time was 18 minutes. (Zhiye et al. 2003) proposed a structural modular neural network, by integrating the BP neurons and the RBF neurons at the hidden layer to build a better input–output mapping. GA method was applied to search for the best number of hidden neurons that makes the structural modular neural network less likely to be trapped in local minima than the conventional gradient-based search algorithms. The preliminary results showed that the proposed structural modular NN approach was more accurate than using the BP and RBF network, or alone. (Delgado and Pegalaja 2005) presented GA which was able to determine the optimal size of RNN. The developed solutions demonstrated that the near/optimal size of RNN with the lowest number of neurons and higher error, or solutions with a higher number of neurons and lowest error were obtained. Thus, depending on the problem, a solution with certain characteristics can be chosen. A new method for the auto-design of NN based on GA has been developed (Boozarjomehry and Svrcek 2001). Mainly GA was applied to optimize structure of NN: connection weights and number of neurons in hidden layer. For performance evaluation, the auto-design of NN was tested against three standard benchmarks that are commonly used by researchers in the field of artificial neural networks. The results obtained by the method were much simpler, yet more accurate than those designed by conventional method. A methodology for determining the best neural networks architecture (i.e. number of hidden layer and number of neurons per layer) based on GA in order to improve ANN's performance for training process and generalization ability, as well as to improve its complexity (Benardos and Vosniakos 2007). The results illustrated that the approach performed better than ANN based on human expert, besides offering several advantages in comparison to similar approaches found in the literature.

### 2.4.3 GA to optimize the connection weights of ANN

Actually, there are numerous researches focused on optimization the connections weight of ANN using GA for example, (Blanco, Delgado et al. 2001) presented a real coded genetic algorithm to train RNN. In particular, GA is utilized to optimize the weights of the network. The results showed that the RNN trained with real coded GA method has a low complexity time and are able to search not only in depth, like the real-time recurrent learning algorithm, but also in width. (Taeksoo and Ingoo 2000) developed a hybrid back-propagation ANN and genetic algorithm to find optimal signal multi-resolution method of the decomposed univariate time series for the daily Korean won/US dollar exchange-rate forecasting, real coded GA was provided to optimize every connection weight in the NNs. The experimental results showed that the forecasting performance of the ANN was enhanced through using GA. Furthermore, a significantly better generalization ability of NNs had been achieved. A hybrid approach of ANN for time series properties, i.e. the time delay neural networks (TDNNs) and the adaptive time delay neural networks (ATNNs), amongst the GA for stock market prediction tasks (Hyun-jung and Kyung-shik 2007). For an efficient search, GA was applied to search for the optimal number of time delays and network architectural factors in ATNN and TDNN, which are the connection of hidden and output neurons, and the number of hidden neurons. The results demonstrated that a higher accuracy had been provided by the proposed integrated approach than standard ATNN, TDNN and the RNN. (Sedki, Ouazar et al. 2009) investigated the effectiveness of real coded GA that evolved from back-propagation NN for rainfall-runoff forecasting in Morocco. GA was employed to search for optimal or approximate optimal connection weights and thresholds for the back-propagation network. The results indicated that the predictive performance of the developed hybrid model was better than that of the conventional BP network. (Kyoung-jae and Ingoo 2000) proposed a GA approach to determine ANN connection weights and their thresholds for input feature discretization to predict the stock price index. Experimental results showed that the GA approach in the feature discretization model performed better than the other conventional models. (Krishna 2009) used a genetic algorithm procedure to determine the connection weights that contributed to the minimization of error between desired output and actual output of the neural network



for surface roughness in electric discharge machining. The results observed that the error, when the network was optimized via genetic algorithm, had been reduced to less than 2% from more than 5%. A hybrid ANN and GA to initialize and optimize the connection weights of back-propagation ANN in order to improve the network performance (Shanthi et al. 2009). The presented model has been applied in the medical field for predicting stroke disease. The results showed improvement in the prediction accuracy of ANN performance. (Zhengjun et al. 2004) discussed the key advantage of the GA optimized NN. In this case, real coded GA method and back propagation NN are applied. The GA operators were carefully intended to optimize the connection weights of the NN, aiming to avoid premature convergence and permutation problems. The experiment results showed that the coupled genetic algorithm and neural network outperformed gradient descent-based NN. (Yang et al. 2009) incorporated a GA based on back propagation neural network (IGABP) to determine connection weights of BPNN automatically, and to provide an efficient GA with reduced computation time in order to enhance the BPNN training capacity. The result indicated that the IGABP model could effectively overcome the inadequacies of the traditional model, besides the efficiency and forecasting performance were significantly improved.

#### **2.4.4 Comparison between state of the art approaches and the proposed approach**

The literature review has demonstrated the lack and limitation of an integrated method that takes into account, the type of minimization functions of the training algorithm, the type of activation functions for hidden and output neurons, initial weight, learning rate, momentum term and epoch numbers, which, particularly have impact on ANN prediction performance in terms of efficiency and accuracy. Therefore, there is a need for a novel approach that covers most ANN aspect in order to improve multilayer feed-forward performance. A comparison between the state of the art approaches and proposed approach are listed as follows:

- Similar to state of art approaches the proposed GA-ANN approach is developed to optimize the architecture and training parameters of multilayer feed-forward ANN. It based on binary-encoded GA method.
- Unlike state of art approaches which used to optimize parte of multilayer feed-forward ANN, the proposed GA-ANN comprehensively covered most aspect of multilayer feed-forward ANN.
- The state of art approaches examined with real or standard datasets (not both). However the proposed approach is applied to three real operation datasets and XOR standard problem.
- The proposed approach presents fully automatic designing of multilayer feed-forward ANN. The state of art approaches proposed semi or partial automatic designing of multilayer feed-forward ANN.
- In literature the statistical analysis is used to verify the efficiency of ANN prediction model. Some researchers found the mean square and root mean square errors the best parameters to distinct between observed and predicted output. In this research the statistical parameters: mean square error, root mean square error, correlation coefficient and determination coefficient are used to distinguish between observed and predicted output.

Moreover, the following Tables show the comparison between the state of the art approaches.

Table 2.1 The stae of art of ANN deisng using GA (1/2)

Approach	Multilayer feed-forward ANN architecture and training parameters								
	Input	Hidden Layer	Number of neurons	weight	Training algorithm	Activation function	Learning rate	Momentum term	Epoch number
Mokl et al. (2001)	√	x	x	x	x	x	x	x	x
Somnath et al. (2003)	√	x	x	x	x	x	x	x	x
Istadia and Nor Aishah(2007)	√	x	x	x	x	x	x	x	x
Fertinous (2005)	x	√	√	x	√	√	x	x	x
Goloka and Chittaranjan (2006)	x	√	√	x	x	x	x	x	x

Table 2.2 The state of art of ANN design using GA (2/2)

approach	Input	Hidden Layer	Number of neurons	weight	Training algorithm	Activation function	Learning rate	Momentum term	Epoch number
Haralambos et al. (2004)	X	√	X	√	X	X	X	X	X
Blanco et al. (2001)	X	X	√	X	X	X	X	X	X
Aeksoo and Ingoo (2000)	X	X	√	X	X	X	X	X	X
Hyun-jung and Kyung-shik (2007)	X	√	√	X	X	X	X	X	X
Ahmad et al. (2004)	√	X	X	X	X	X	X	X	X
Cook et al. (2000)	√	X	X	X	X	X	X	X	X
Hasan et al. (2005)	√	X	X	X	X	X	X	X	X
Nan et al. (2003)	X		√	X	X	X	X	X	X
Delgado and Pegalajar (2005)	X	√	√	X	X	X	X	X	X
Boozarjomehry and Svrcek (2001)	X	X	√	√	X	X	X	X	X
Sedki et al. (2009)	X	X	X	√	X	X	X	X	X
Kyoung-jae and Ingoo (2000)	X	X	X	√	X	X	X	X	X
Rao (2009)	X	X	X	√	X	X	X	X	X
Shanthi et al. (2009)	X	X	X	√	X	X	X	X	X
Koc et al. (2007)	√	X	X	X	X	X	X	X	X
Nasseri et al. (2008)	√	X	X	X	X	X	X	X	X
Adineh (2008)	√	X	X	X	X	X	X	X	X
Benardosa and Vosniakos (2007)	X	√	√	X	X	X	X	X	X
Zhengjun et al. (2004)	X	X	X	√	X	X	X	X	X
Sedki et al. (2009)	X	X	X	√	X	X	X	X	X
Yang (2009)	X	X	X	√	X	X	X	X	X

## 2.5 Multilayer feed-forward ANN data preprocessing, architecture and training parameters

Currently to the best of our knowledge, there is no clear method or theoretical background to determining the best multilayer feed-forward ANN architecture and training parameters. The implementation of multilayer feed-forward ANN models requires the solutions of two complex optimization tasks, which are the architecture and the training process parameters. The two tasks are closely correlated. Since the significance of a candidate multilayer feed-forward ANN architecture can only be assessed on the trained parameters, the accuracy and reliability of the training process

affect the outcome of the multilayer feed-forward ANN design process. Alternatively, the selection of architecture has a considerable impact on the multilayer feed-forward ANN performance efficiency and learning capabilities.

The determination of the best multilayer feed-forward ANN architecture and training is performed by using a trial and error procedure, which can be relatively inefficient, human-dependent and less precise. The determination algorithm usually involves two methods: destructive and constructive (Nabhan and Zomaya 1994; Rychetsky et al. 1998; Parekh et al. 2000). In the former, one starts with a large network size and removes unnecessary neurons and connection weights until 'optimum' architecture is obtained based on prediction performance and processing time. Meanwhile, the latter is initialized with a quite simple network, and neurons and connection weights are added to minimize the error, which is the difference between actual output and desired output.

Neural training process is considered successful only if the model can perform well on test data for which the model has not been trained. This ability of network is called generalizability. Given a large network structure, training iterations has the potential to successively improve performance of the network on training data e.g. by "memorizing" training set; however, the resulting network may perform poorly on test data (unseen data). This phenomenon is called "overtraining".

This section provides a detailed survey of the data preprocessing, multilayer feed-forward ANN architecture and training parameters, which are removal of missing value, data outlier detection, data normalization, data portioning, number of hidden layers, number of neurons in hidden layer, training algorithm, activation function, learning rate, momentum term, and epoch numbers (Table 2.3). This is followed by related methods that are recommended by previous researchers to determine the best parameters relevant to this thesis.

Table 2.3 The list of most common parameters in designing a multilayer feed-forward ANN (Kaastra and Boyd 1996)

Parameter	Elements
Data Preprocessing	Frequency of Data – Daily, Weekly, Monthly, Quarterly. Type of Data-Technical, Fundamental. Method of Data Sampling. Method of Data Scaling- Minimum/Maximum, Mean/Standard Deviation.
Architecture	Number of Input Neurons Number of Hidden Layer Number of Neurons in Each Hidden Layer Number of Output Neurons Activation Function for Hidden and Output neurons.
Training	Training Algorithm Weight Initialization Learning Rate Momentum Term Epoch Size Size of Training, Validation, Testing Sets

### 2.5.1 Data preprocessing

One of the most important issues in the success of any ANN design is data preprocessing. The quality, reliability, repeatability, availability, and relevance of the data used to construct and run the ANN model are critical to its success. Even a primitive model can perform well if the training data has been processed in such a way that it clearly discloses the important information. Alternatively, even the best model cannot perform well if the necessary training data are presented in a complex and confusing way.

ANN has been shown to be able to process data from a broad range of resources. They are, though, only able to process the data in a specified form. Moreover, the techniques of data presentation to the network influence the training process of the

network. Consequently, a certain amount of data processing is required before presenting the training dataset to the network.

It may sometimes be essential to remove some of the data outliers for better and smoother network generalization capabilities. Statistical analysis shows that 95% of a normally distributed data lie within 2 standard deviations and 99% within three standard deviations (Swingler 1996). The process of removing data outside these ranges will greatly improve network training, provided that these data are great and uncharacteristic of the problem area (Rafiq, Bugmann et al. 2001).

#### 2.5.1.1 *Missing values*

Missing data may be one sample or sets of samples; there is/are one or more variable(s) (i.e. measurements) that contain a value which does not reflect the real situation of the measured physical quantity. The influenced variables typically have values like  $\pm\infty$ , 0 or any other constant value.

Since most learning and statistical method can be significantly affected by missing values, the data with missing values should be treated before modeling.

The simplest method to deal with missing values is to remove them in the original data set; however, removing missing values can affect in loss of a great deal of valuable information (Jiu-sun and Chuan-hou 2009). In addition, when ANN model is run, missing of some input variables may lead to malfunction and bad results. In the majority of cases, variables in the datasets are dependent upon each other. Therefore, the missing values can be determined through a mapping of relationship among the variables.

There are different approaches to replace missing values. Table 2.3 lists some examples of approaches to replace missing values.

Table 2.4 Common approaches to replace missing data

No.	Reference(s)	Approach involved
1	Commonly applied in practical scenarios	Replacing the missing values with the mean values of the affected variable.
2	(Scheffer 2002)	Skipping the data samples consisting of variable or variables with the missing values.
3	(Walczak and Massart 2001a)	Maximum-likelihood multivariate approach to missing values replacement.

### 2.5.1.2 Data outlier

Outlier detection belongs to the extremely important task in data preprocessing. The outliers are typical, infrequent observations which deviate significantly from the majority of observations, or do not appear to follow the statistical characteristics of the rest of the data. ANN is noise tolerant. Though, there is a limit to this tolerance. If there are infrequent outliers far outside the range of typical values for a variable, they may cause a bias in the training process. The best approach to such outliers is to remove or convert them.

Outlier detection as a branch of the data pre-processing remains extremely significant for the ANN development because undetected outliers have negative impact on the performance of the network.

Various approaches have been proposed for outlier detection. The estimation of location (e.g., the mean) and scatter (e.g., variance/covariance) are the two most important statistical elements for data analysis in the presence of outliers (Rousseeuw and Leroy 1987). However, most approaches suppose that the data are from a normal distribution, and recognize observations which are considered 'unlikely' based on the mean and standard deviation. Table 2.4 shows the common approaches to detect outlier data.

Table 2.5 Common methods to detect outlier data

No.	Reference(s)	Method involved
1	(Pearson 2002; Lin et al. 2007)	$3\sigma$ outlier detection algorithm
2	(Egan and Morgan 1998)	Resampling by half-means (RHM)
3	(Davies and Gather 1993)	Hampel identifier
4	(Egan and Morgan 1998)	Smallest half volume (SHV)
5	(Jolliffe 2002; Warne et al. 2004)	Jolliffe parameter based on PCA

In a three SIGMA method ( $3\sigma$ ), all data that fall out of the range of  $\mu(x) \pm 3\sigma(x)$  are labeled as outlier, where  $\mu(x)$  is the mean value and  $\sigma(x)$  is the standard deviation of the variable  $x$ . The Hampel identifier method, which is in contrast to the  $3\sigma$  method, substitutes the outlier-sensitive mean and standard deviation estimates with the outlier resistant median and median absolute deviation from the median (MAD) (Pearson 2002) to compute the limits. The principal component analysis (PCA) is a popular multivariate analysis method that is used to detect outlier data.

### 2.5.1.3 Data normalization

The normalization of dataset is a greatly significant task. This typically makes the training algorithm faster and numerically stable. This step is needed to transform the data into a suitable form for the network inputs. The approach that is usually used for scaling network inputs and outputs is to normalize the mean and standard deviation of the training dataset. Normalizing the inputs and outputs mean that they will have a zero mean and unity standard deviation if sorted and plotted against their frequencies. This step too, involves the selection of the most relevant data that the network can obtain. If the input and output data are not of the same order of magnitude, some variables may appear more considerably large than they actually do. Since the activation functions are bounded between 0 and 1, or -1 and 1 etc, the input and output data are normalized to the same range as that of the activation function. Furthermore, normalization of inputs leads to avoidance of numerical overflows due to very large or very small weights (Richard et al. 1998).



In ANN configuration, the restrictions on the design variables are pre-defined, such that the optimum point and all other intermediate iterates are sure to fall inside the restrictions. These pre-defined restrictions are used to normalize the input and output variables as follows:

$$x_i = \frac{x_i - x_{min_i}}{x_{max_i} - x_{min_i}} \quad (2.15)$$

$$y_k = \frac{y_k - y_{min_k}}{y_{max_k} - y_{min_k}} \quad (2.16)$$

Where,  $x_{min_i}$  and  $x_{max_i}$  are the lower and upper bounds for the  $i$ th input variables, respectively. Similarly,  $y_{min_k}$  and  $y_{max_k}$  are the lower and upper bounds for the  $k$ th output variables.

#### 2.5.1.4 Dataset partitioning

Partitioning process is used to classify the dataset into three different sets: training or learning set, validation set, and test or over-fitting test set. The number of training set used in a network has a significant influence on the learning process. By definition, usually the *training set*, which is the biggest is used to develop and adjust the connection weights and biases in a network; the *validation set* is used to ensure the generalization capability of the developed network through the training process, and the test set is used to examine the final network performance and for calibration, which prevents overtraining the networks. The critical concerns should be to ensure that: the training set contains enough data to represent the whole data (i.e. all possible minimum and maximum values in the training set), appropriate data distribution to sufficiently cover the entire range of data, and there is no unnecessary similarity between data in different datasets. Different partitioning ratios have been used by previous researcher for example (2:1:1, 3:1:1, and 4:1:1). However, the ratio of 4:1:1 (suggested by (Haykin 1994)) yielded better training and testing results.

There is no accurate rule on the optimum size of the three sets of data, although several authors have suggested that the training set must be the largest (Kaastra and Boyd 1996; Zhang, Patuwo et al. 1998), however the risk of memorization becomes

possible. Table 2.6 provides a number of different suggestions for the required number of training, validation and testing set.

Table 2.6 Common approaches for data partitioning

No.	Reference(s)	Approach involved
1	(Looney 1996; Swingler 1996)	$20\% \leq \text{testing set} \leq 25\%$ and $75\% \leq \text{training set} \leq 80$
2	(Nelson and Illingworth 1990)	$20\% \leq \text{testing set} \leq 30\%$ and $70\% \leq \text{training set} \leq 80$
3	(Zhang et al. 2001)	70%, 15% and 15% for training set, validation set and testing set of data

## 2.5.2 Architecture

Multilayer feed-forward ANN architectures are defined as a structure that includes the number of input neurons, number of output neurons, number of hidden layers, number of neuron per layer(s) and number of output neurons. The hidden layer(s) provide the network generalization capabilities. This research has applied three layers feed-forward BP network: one or two hidden layer(s), input and output layer. Based on the literature review, ANN with one or two hidden layer(s) has been found as sufficient and has performed very well (Kaastra and Boyd 1996; Zhang, Patuwo et al. 1998).

### 2.5.2.1 Number of hidden layers and number of neurons in hidden layers

A “hidden layer” exists as a process between the input neurons and the corresponding output neurons within a network. Hidden and numbers of neuron per hidden layer(s) are the ones that bring out the defining properties within the data, and assist to establish the nonlinear relationship between the input and output. The number of hidden layer(s) and neuron(s) in the hidden layer(s) is determined mostly by trial and error procedure. Determining the optimal number of hidden and neurons in hidden layer(s) is a considerable task. The size of hidden layers of ANN has direct impact on its complexity. Generally one hidden layer may be sufficient for the majority of problems, but in some problems, the application of ANN with two hidden layers can be satisfactory.

In practical, networks with too many hidden layers will be more inclined to have: memorization ability rather than generalization, increased training time and the danger of over-fitting, which tend to lead to poor out-of-sample prediction performance. Over-fitting happens when a prediction model has too few degrees of freedom. In other words, it has comparatively few observations in relation to its parameters, and therefore it is able to memorize individual points rather than learn the general patterns (Kaastra and Boyd 1996). Nevertheless, the networks with too few hidden layers are preferred, because generalization is more achievable this way even though they may not have enough power to represent and learn the data, and have much less-fitting problem.

The increment in the number of hidden layer and neurons per layer are depending on the computational resources available. There is no general method to determine an optimum number of hidden neurons per layer(s), but several researchers have provided some guidance in order to arrive at some kind of optimal structure of an ANN model, as showing in Table 2.7 and 2.8.

Table 2.7 Some common methods to select the number of hidden neurons (1/2)

No.	Reference(s)	Method Involved
1	(Masters 1993)	Number of hidden neurons $= \sqrt{\text{number of the input} * \text{number of the output}}$
2	(Masters 1993)	For network with two hidden layers, $n$ neurons for first hidden layer and $2n + 1$ neurons for second hidden layer, sufficient for $n$ inputs
3	(Kusiak 2000)	Number of the input < number of hidden neurons < Number of the output
4	(Baum and Haussler 1988)	Number of hidden neurons = number of data point in the training set * error / (number of the input + number of the output layer)
5	(Baily and Thompson 1990)	Number of hidden neurons = 75% of the number of the input.
6	(Katz 1992)	$1.5 * \text{the number of the input} < \text{number of hidden neurons} < 3 * \text{the number of the input.}$

Table 2.8 Some common methods to select the number of hidden neurons (2/2)

No.	Reference(s)	Method Involved
7	(Zhang 1993)	Number of hidden neurons $= \sqrt{\text{number of the input} * \text{number of the output} + a}$ and number of hidden neurons = total number of data points/ total number of weights Where a is varying from 1 to 10
8	(Chen and Yang 2002)	Number of hidden neurons = (number of the input + number of the output)/2 + $\sqrt{\text{number of training sample}}$
9	(Lawrence 1994)	Number of hidden neurons = 1/2 * number of the input + number of the output

The methods 1-9 as showing in Table 2.6 and 2.8, which are dependent on the size of input neurons and output neurons are not logical, since the factors that influence the networks structure are the number of samples in a training set, the noise size of samples, and the complex degree of function or classification to learn, and so on.

### 2.5.3 Activation functions

Activation functions are mathematical formulas that determine the output of processing neurons. It is an important element of the network structure, and has a significant impact on the network performance. They are also called threshold, transformation, squashing, or transfer functions. The activation function of the fundamental element of ANN has two sub-functions: the combination function and the activation function. The combination function generally uses the “standard weighted sum” (the summation of the input variables multiplied by their corresponding weights that have been assigned to those variables) to calculate a value to be passed on to the activation function. The activation function applies either a linear or a nonlinear transformation to the value passed to it by the combination function. The “hidden layer” then employs this activation function to pass data to the output neurons (John et al. 2006). Based on the problem types, the activation functions can take any form and may be linear or nonlinear, and can take on any value between 0 and 1, or between -1 and 1, depending on the particular function selected.

Most of the current ANN models use the sigmoid (S-shaped) function because of their nonlinearity network abilities, which plays a significant function in the performance of the ANN (Kong and Martin 1995 ), but other functions such as the tangent hyperbolic, logistic, step, ramping, arc tan, and linear have also been proposed. The purpose of the activation function is to prevent outputs from reaching very large values, which can ‘paralyze’ ANN and thereby inhibiting training, and to reduce the network storage capacities.

The selection of best activation functions may strongly influence performance and complexity of ANN (Duch and Jankowski 2001). The common activation function types and their mathematical formulas are shown in Table 2.9.

Table 2.9 Common activation functions of hidden and output layers

No.	Activation function	Formula
1	Hard limit	$f(x) = \begin{cases} 0, & \text{if } x < 0 \\ 1, & \text{if } x \geq 0 \end{cases}$
2	Symmetrical hard limit	$f(x) = \begin{cases} -1, & \text{if } x < 0 \\ 1, & \text{if } x \geq 0 \end{cases}$
3	Saturating linear	$f(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } 0 \leq x \leq 1 \\ 1, & \text{if } x > 1 \end{cases}$
4	Symmetrical Saturating linear	$f(x) = \begin{cases} -1, & \text{if } x < 0 \\ x, & \text{if } 0 \leq x \leq 1 \\ 1, & \text{if } x > 1 \end{cases}$
5	Positive linear	$f(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases}$
6	Linear	$f(x) = x$
7	Sigmoid	$f(x) = \frac{1}{1 + e^{-x}}$
8	Hyperbolic tangent	$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
9	Triangular basis	$f(x) = \begin{cases} 0, & \text{if } x < -1 \\ 1 + x, & \text{if } -1 \leq x < 0 \\ 1, & \text{if } x = 0 \\ 1 - x, & \text{if } 0 < x \leq 1 \\ 0, & \text{if } x > 1 \end{cases}$
10	Gaussian	$f(x) = e^{-x^2/2}$

Some examples of typical activation functions are illustrated in Figures 2.14 to 2.17.

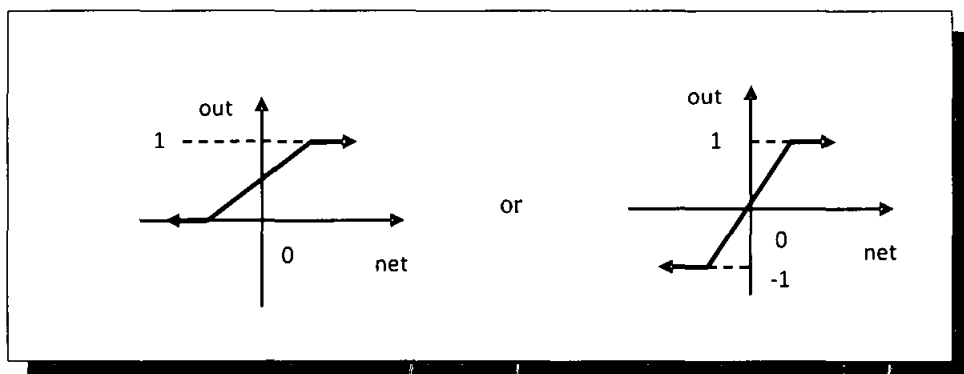


Figure 2.14 A hard-limit activation function

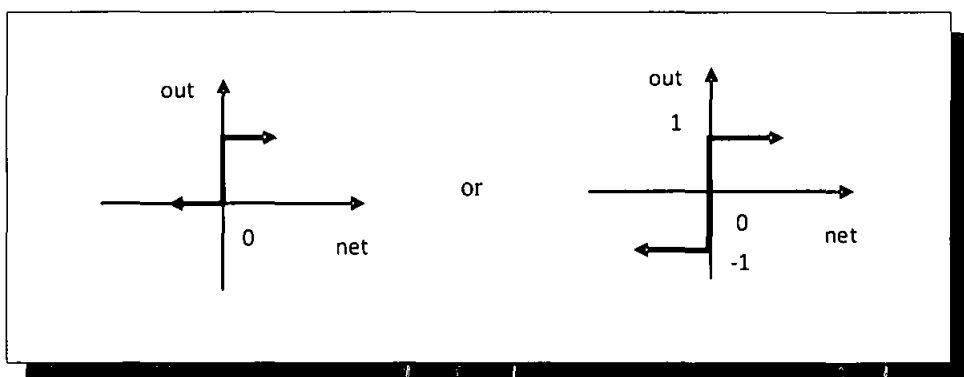


Figure 2.15 A threshold-logic activation function

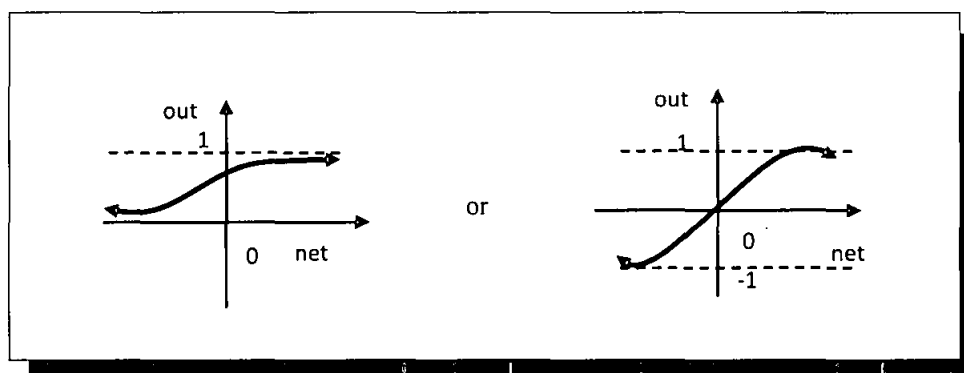


Figure 2.16 Continuous activation functions (a) the sigmoid (b) the hyperbolic tangent

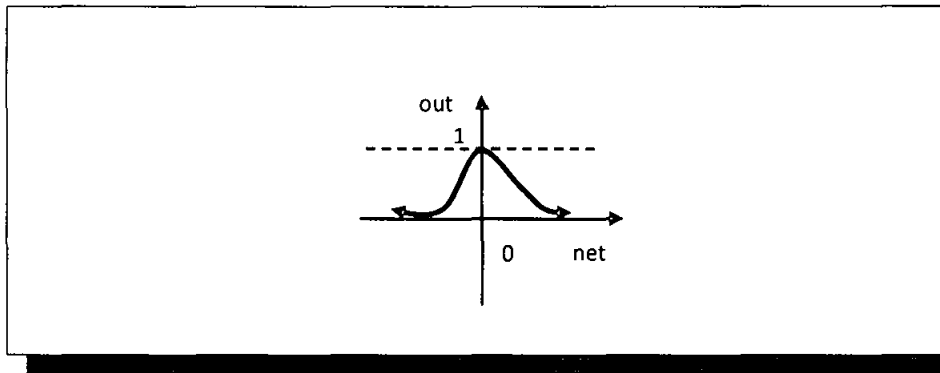


Figure 2.17 The Gaussian activation functions

#### 2.5.4 BP multilayer feed-forward ANN

Recently, ANN is used as a powerful tool in variant type of nonlinear problem solution. The training algorithm is the basic element needed to design ANN for a problem solution. It is required to prepare a set of data, which represents the problem in the forms of inputs and outputs variables. Throughout the training process, the connection weights and biases of the network are adjusted to reduce the error in order to attain a high network performance in the solution. In addition, during the training error, i.e., mean squared error is calculated between actual outputs and target outputs. There are several training algorithms that have been used in ANN applications. It is very hard to identify which training algorithm will be the fastest for a given problem. Because it depends on numerous issues, including the number of data points in the training dataset, the complexity of the problem, the error goal, the number of weights and biases in the network, and whether the network is being used for example pattern recognition or function approximation.

In this research, four different BP training algorithms of multilayer feed-forward ANN are considered: Steepest-Descent, Conjugate-Gradient, Quasi-Newton, and Levenberg-Marquardt algorithms (Lahiri and Ghanta 2008). Their main difference is the method of approximating the inverse of the Hessian matrix. The conjugate gradient and the steepest descent algorithms replace the inverse of the Hessian with the identity matrix, whereas the other two algorithms try to approximate it with different methods.

The simplest implementation of back-propagation learning changes the network connection weights and biases in the direction in which the performance function decreases most rapidly, i.e., the negative of the gradient ( $-g_k$ ). The  $(k+1)$ th iteration of this algorithm, as described in (Demuth and Beale 2004), can be written as:

$$w_{k+1} = w_k - \alpha_k g_k \quad (2.17)$$

Where  $w_k$  is a vector of current connection weights and biases,  $g_k$  is the current gradient, and  $\alpha$  is the learning rate.

#### 2.5.4.1 Steepest Descent (SD)

Steepest descent (SD) is a generalization version of the Least Mean Square (LMS) algorithm. In fact, steepest descent corresponds to the LMS algorithm when applied on a single-layer network. However, the characteristics of SD are quite different when applied to multi-layer network type.

In the SD algorithm  $M_k = I$ , where  $I$  is the identity matrix. In the process of error parameter minimization, SD algorithm takes the negative direction of the gradient ( $-g_k$ ), i.e. the direction of the SD of the error function. The SD algorithm can be written conveniently in matrix notation as:

$$w_{k+1}^m = w_k^m - \alpha s^m (a^{m-1})^T \quad (2.18)$$

$$b_{k+1}^m = b_k^m - \alpha s^m \quad (2.19)$$

$$s^m = \frac{\partial \hat{F}}{\partial n^m} \quad (2.20)$$

Where  $m = 0, 1, \dots, M - 1$ , and  $M$  is the number of layers in the network,  $\alpha$  is the learning rate,  $s^m$  is the sensitivity of layer  $m$ , and  $a$  is the output.



In the process of error parameter minimization, according to SD method, the variation of connection weights and bias is decided from the product of learning rate and the negative gradient, but it can easily fall into a local minimum instead of a global minimum. If the learning rate increases the speed of convergence, the network becomes unstable; and if the value of learning rate is too small, it will converge slowly. Therefore, if a momentum term is added to connection weights that should improve the convergence performance.

#### 2.5.4.2 Conjugate Gradient (CG)

Conjugate gradient (CG) method (Yu and Chen 1997; Kamarthi and Pittner 1999) is a class of extremely significant methods for the process of error parameter minimization, in particular when the dimension is large (Hertz et al. 1991). They are considered as conjugate direction or gradient deflection methods, which lie between the method of SD and Newton's method. Their primary benefit is that they do not need the storage of any matrices like in quasi-Newton methods, or as in Newton's method.

CG method is introduced to converge faster than the SD method. CG initializes by searching in the steepest descent direction (negative of the gradient) on the first iteration.

$$p_0 = -g_0 \quad (2.21)$$

A line search is then performed to determine the optimal distance to move along the current search direction:

$$w_{k+1} = w_k + \alpha p_k \quad (2.22)$$

Now, the next search direction is determined, therefore it is conjugated to previous search directions. The general process for determining the new search direction is to join the new steepest descent direction of the previous search direction:

$$p_k = -g_k + \beta_k p_{k-1} \quad (2.23)$$

Where  $p_k = w_{k+1} - w_k$  and  $g_k = g_{k+1} - g_k$ . The variants of CG are well-known by the method, in which the constant  $\beta_k$  is calculated. For the Fletcher–Reeves update as illustrated in (Chong and Stanislaw 2004; Demuth and Beale 2004), the process for computation is:

$$\beta_k = \frac{g_k^T g_k}{g_{k-1}^T g_{k-1}} \quad (2.24)$$

This is the ratio of the squared norm of the current gradient to the squared norm of the previous gradient.

#### 2.5.4.3 Quasi-Newton (QN)

The Newton's method is an alternative to the CG method, particularly for fast optimization. The basic step of this algorithm is:

$$w_{k+1} = w_k - A_k^{-1} g_k \quad (2.25)$$

Where  $A_k^{-1}$  is the second derivatives (Hessian matrix) of the performance index at the current values of the connection weights and bias on the network. The Newton's algorithm convergence speed is generally faster than CG descent methods.

Unfortunately, it is difficult and costly to calculate the Hessian matrix for feed-forward ANN. There is a part of algorithms that is based on Newton's method, but which doesn't need calculation of second derivatives. These are named quasi-Newton (QN) methods, which updates an approximate Hessian matrix at each epoch of the algorithm (Dennis and Schnabel 1983; Battiti 1992).

The update is calculated as a function of the gradient by using the following Eq. (2.26), as given in (Chong and Stanislaw 2004).

$$A_{k+1} = A_k + \left( 1 + \frac{\Delta g_k^T \Delta g_k}{\Delta g_k^T \Delta w_k} \right) \frac{\Delta w_k \Delta w_k^T}{\Delta w_k^T \Delta g_k} - \frac{A_k \Delta g_k \Delta w_k^T + (A_k \Delta g_k \Delta w_k^T)^T}{\Delta g_k^T \Delta w_k} \quad (2.26)$$

#### 2.5.4.4 Levenberg Marquardt (LM)

The LM algorithm (Hagan and Menhaj 1994; Demuth and Beale 2003) is established to be fastest training algorithm; however, it needs further memory than the BP rule. LM is similar to QN method, both are based on approaching second-order training speeds without having to compute the Hessian matrix (Hagan and Menhaj 1994; Hagan et al. 1996). The great advantage of this training method is its convergence about minimum and it gives more accurate results, whereas the disadvantage is the requirement of more memory than the traditional back propagation method (Karkoub and Elkamel).

In the LM algorithm, the inverse of the Hessian is approximated by the quantity:

$$H = j^T j \quad (2.27)$$

and the gradient can be calculated as:

$$g = j^T e \quad (2.28)$$

Where J is the Jacobian's matrix that contains first derivatives of the network errors with respect to the connection weights and biases, and e is a vector of network errors. The Jacobian's matrix can be calculated through a standard back-propagation algorithm that is too less complex than computing the Hessian matrix. The LM employs this approximation to the Hessian matrix in the following Newton-like update:

$$w_{k+1} = w_k - [J^T J + \mu I]^{-1} J^T e \quad (2.29)$$

This leads to a conclusion that activation function is a major element of the network structure, and has a significant impact on the ANN performance and their

training capabilities. Often, the activation function is assigned by the network designer based on either past experiences, or trial and error approach.

### 2.5.5 Initial weights

The BP algorithm is quite sensitive to connection weights initialization (Kolen and Pollack 1991). It should be initialized to some small, non-zero random values. The appropriate initialization of the connection weights has significant impact on the speed of convergence. Starting with inappropriate connection weights, may lead to getting stuck in local minima or a slow training progress. Changing the number of hidden neurons or training parameters will often avoid the problem, or alternatively, start with a different set of initial connection weights. Different ranges to set the initial weights have been suggested by several researchers. Table 2.10 provides an example of suggested ranges.

Table 2.10 Common methods to select the range of initial weights

No.	Reference(s)	Method involved
1	(Jocelyn and Robert 1991; Looney 1997)	$-0.05 \leq \text{connection weights} \leq 0.5$
2	(Gallahger and Downs 1997; Kavzoglu 2001)	$-0.25 \leq \text{connection weights} \leq 0.25$
3	(Paola 1994; Staufer and Fischer 1997)	$-1 \leq \text{connection weights} \leq 1$

### 2.5.6 Learning rate

Learning rate is defined as the amount of changes in the network connection weights (step size) throughout the training process. The effectiveness and convergence of the error back-propagation learning algorithm depends on the value of learning rate, and it works by adding a proportion of the previous weight changes to the current weight changes (Rumelhart, Hinton et al. 1986; Getiner et al. 1998; Necat and Rasit 2004), where weight changes are calculated as:

$$\Delta w_{ji}(t + 1) = \alpha \cdot \Delta w_{ji}(t) + \eta \cdot \alpha \cdot \frac{\partial E}{\partial w_{ji}} \quad (2.30)$$

Where the learning rate  $\eta$  is limited to the range  $0 \leq \eta < 1$ ,  $\alpha$  is momentum term,  $t$  is the iteration of learning,  $w_{ij}$  is an adjustable parameter known as synaptic connection weight, and  $E$  is equal to:

$$\frac{1}{2} \sum_{j=1}^c e_k^2$$

In which  $e_k$  is the error between calculated output,  $o_k$ , and desired output,  $t_k$ , and  $e_k$  is determined as follows:

$$e_k = t_k - o_k \tag{2.31}$$

Based on the results by other researcher that have been extracted from the literature, the optimum value of learning rate depends significantly on the class of the problem being solved, and on the network architecture; and there is no single learning rate value suitable for different training problems.

Basically, a smaller value of learning rate ensures a true gradient descent, but this may lead to an increase in the total number of learning steps that need to be made to reach the optimum solution. On the contrary, a high value increases the speed of convergence; the network becomes unstable and may converge to local minima instead of global minima in the error space. Several researchers have given guidance as how to arrive at some kind of optimal value of learning rate as shown in Table 2.11.

Table 2.11 The common methods to select learning rate

No.	Reference(s)	Method involved
1	(Uros, 2003)	$0.01 \leq \text{learning rate} \leq 0.2$
2	(Attoh-Okine 1999)	$0.001 \leq \text{learning rate} \leq 0.005$
3	(Refenes et al. 1994)	learning rate = 0.2
4	(Rumelhart, Hinton et al. 1986)	$0.05 \leq \text{learning rate} \leq 0.5$

### 2.5.7 Momentum term

In order to overcome the local minima and slow convergence problems during training process, the momentum term is introduced and it works by adding a proportion of the previous weight changes to the current weight changes (Rumelhart, Hinton et al. 1986; Getiner, Hasiloglu et al. 1998; Necat and Rasit 2004). Where weight changes are calculated as:

$$\Delta w_{ji}(t + 1) = \alpha \cdot \Delta w_{ji}(t) + \eta \cdot \alpha \cdot \frac{\partial E}{\partial w_{ji}} \quad (2.32)$$

Where the momentum term,  $\alpha$ , is limited to the range  $0 \leq \alpha < 1$ . Since, generally every BP algorithm used for training ANN initializes the starting connection weights randomly, there is a high chance that the starting point is situated in a local valley (Nii 1999).

To develop an ANN model, the selection of appropriate momentum term is a very important task as this parameter is concerned with the speed of convergence and stability of the network (Biswajit et al. 2009). In order to have a better convergence speed and to escape local minima to some extent, a momentum term is recognized in ANN; however, a large momentum value leads to oscillation, whereas a small value leads to slow convergence. Some rules to select the momentum term have been given by previous researchers as listed on Table 2.11.

Table 2.12 The common methods of selecting momentum term

No.	Reference(s)	Method involved
1	(Uros and Franci 2003)	$\leq$ momentum term $\leq 0.005$
2	(Randall and Jatinder 2000)	$0.1 \leq$ Momentum Term $\leq 0.9$
3	(Clarence and Gerhard 1993)	$0.5 \leq$ Momentum Term $\leq 0.7$
4	(Freisleben 1992)	Momentum Term = 0.7
5	(Nii 1999)	$0.5 \leq$ Momentum Term $\leq 0.9$
6	(Refenes, Zapranis et al. 1994)	$0.3 <$ Momentum Term $\leq 0.5$
7	(Wythhoff 1993)	$0.4 \leq$ Momentum Term $\leq 0.9$
8	(Fu 1995; Hassoun 1995)	$\leq$ Momentum Term $\leq 1.0$
9	(Hertz, Krogh et al. 1991; Henseler 1995)	Momentum Term $\leq 1.0$

### 2.5.8 Epoch size

During the training process, calculated outputs are compared with the desired output values, and as a backward pass, the difference between desired outputs and calculated outputs is used to adjust the connection weights of the network in order to reduce the level of error. This is an iterative process, which is repeated until an acceptable error level is achieved (indicating that the network has successfully learned the data) or a maximum number of epochs have passed (indicating that the network has been unsuccessful to learn the data in the number of epochs it has selected). The time the network processes each adjustment between the connection weights and biases (both a forward and a backward pass), is called an *epoch* or *iteration*. The network is trained in this way and the error is reduced by each epoch until an acceptable error level is obtained.

The selection of epoch number is extremely important for the network development. This parameter has a strong effect on the ANN model prediction. The number of training epochs is one of the major drawbacks of using feed-forward ANN; there is no ideal technique to find a suitable epoch number. Increasing the size of training epoch may increase the over-fitting problem (Lae et al. 1999), and decreasing the epoch number will increase the number of neurons in the hidden layer. Moreover, a suitable size of training epoch has to be assigned to overcome the problems of data over-fitting and under-fitting. Several previous researchers have provided some method to determine the epoch size, for example the user defined error level, early stopping and using test dataset.

The problem with user defined error method is the difficulty to choose the appropriate error level. Usually the error drops after a certain number of epochs, where it levels off and does not get much smaller; though at this point the network may be over-trained. In the second method, which is early stopping, the training data are held out from the training process but instead, are used to test the network performance; in this method the error goes down on the training data as the training proceeds. The error also initially goes down on the holdout data, but then the error level rises again as the model becomes over-trained. One of the disadvantages of this method is that it requires more data, which often is not available; and it does not give

an assurance that the minimum error found is a global minimum rather than a local minimum. The last method is using test dataset to determine when to stop the training process, in other words, it is not an independent test of the model.

## **2.6 Summary**

In this chapter the current state of the art in ANN and GA domains has been discussed. Then the chapter covers the most common data preprocessing techniques to designing multilayer feed- forward for example missing values removal, data outlier detection and removal, data normalization, data partitioning. Further, the chapter provides a comprehensive detailed survey about the multilayer feed-forward ANN architecture and training parameters, and their effects on it performance efficiency, convergence and accuracy, these parameters including the number of hidden layers, number of neurons in hidden layers, training algorithm, activation function, learning rate, momentum term, epoch size. Some methods to determine appropriate network architecture and training parameters recommended by previous researchers have also been explored and compared. Based on the recommended methods, a new approach is required to overcome the drawback of trial-and-error procedure and to fill up existing attempts gap.



## Chapter 3

### RESEARCH METHODOLOGY

#### **3.1 Introduction**

This chapter describes the methodology that has been used in this research. The first part of this chapter discusses the procedures involved in this research and focuses on the design of proposed GA-ANN approach. The second part explains in detail every step involved in designing the GA-ANN approach using different four datasets. This is followed by a description of the parameters, functions, and techniques that are required in multilayer feed-forward ANN learning algorithm. The last part of this chapter explains how the experiment and analysis have been accomplished in order to investigate the efficiency of proposed GA-ANN, and to predict the performance of the GA-ANN approach by using GA binary encoding technique.

#### **3.2 Research tools**

This research has been carried out using MATLAB2009a with Neural Network ToolBox6.0 for process optimization and modeling to provide data for regional analyses.

##### **3.2.1 MATLAB2009a with neural networks ToolBox6.0**

MATLAB stands for "MATrix LABoratory" is computation software developed by Math Works Inc. with excellent computation and visualization capabilities (H.B. Demuth and M. T. Beale 2004). MATLAB is becoming a powerful programming language due to its interactive computational environment.

MATLAB neural network toolbox has been utilized to develop various ANN applications because of its simplicity, and flexibility that allows the user to quantitatively and graphically monitor the network training process, and analyze the results.

The ANN toolbox allows modeling of the problem using feed forward back-propagation, radial basis and recurrent ANN with a wide range of activation functions, learning techniques, network architectures, performance optimization and performance functions.

In MATLAB software, a feed-forward back propagation network is produced by the following syntax and takes several arguments:

```
net = newff(P,T, [S1 S2...S(N-1)], {TF1 TF2...TFN1}, BTF, BLF, PF)
```

The description of this syntax as follows:

*newff* Create feed-forward back-propagation network

*P* R x Q1 matrix of Q1 sample with R element input vectors

*T* SN x Q2 matrix of Q2 sample with SN element output vectors

*Si* Size of *i*th layer, for N-1 layers, default = [ ], (Output layer size SN is determined from T)

*TFi* Transfer function of *i*th layer (Default = tangent sigmoid 'tansig' for hidden layers and pure linear 'purelin' for output layer.)

*BPF* Back-propagation network training function (default = Levenberg-Marquardt 'trainlm')

*BLF* Back-propagation weight/bias learning function (default = Gradient descent 'leamgdm')

*PF* Performance function (Default = mean square error 'mse')

In back-propagation training process stops when any of the following situations occurs:

- (1) The maximum size of epoch is met.
- (2) The maximum amount of time is exceeded.
- (3) Performance is minimized to the goal.
- (4) The performance gradient falls below min\_grad.
- (5) Validation performance has increased more than max\_fail times since the last time it decreased (when using validation).

All these termination criteria are determined by user.

### **3.3 Research activities**

The research activity started with a review of earlier researches that are related to parameterization and design of artificial neural network using genetic algorithm.

The purpose of this study is to adopt GA to determine the number of hidden layers, the number of neurons in the hidden layers, training algorithm, activation function, initial weight, learning rate, the momentum term and epoch size of multilayer feed-forward ANN. The steps in the research methodology include collection of experimental data, data preprocessing, optimizing multilayer feed-forward ANN architecture and training parameters using GA, training, validation, and testing phases of the multilayer feed-forward ANN , and finally comparison and analysis step. All of the abovementioned steps are shown in Figure 3.1.

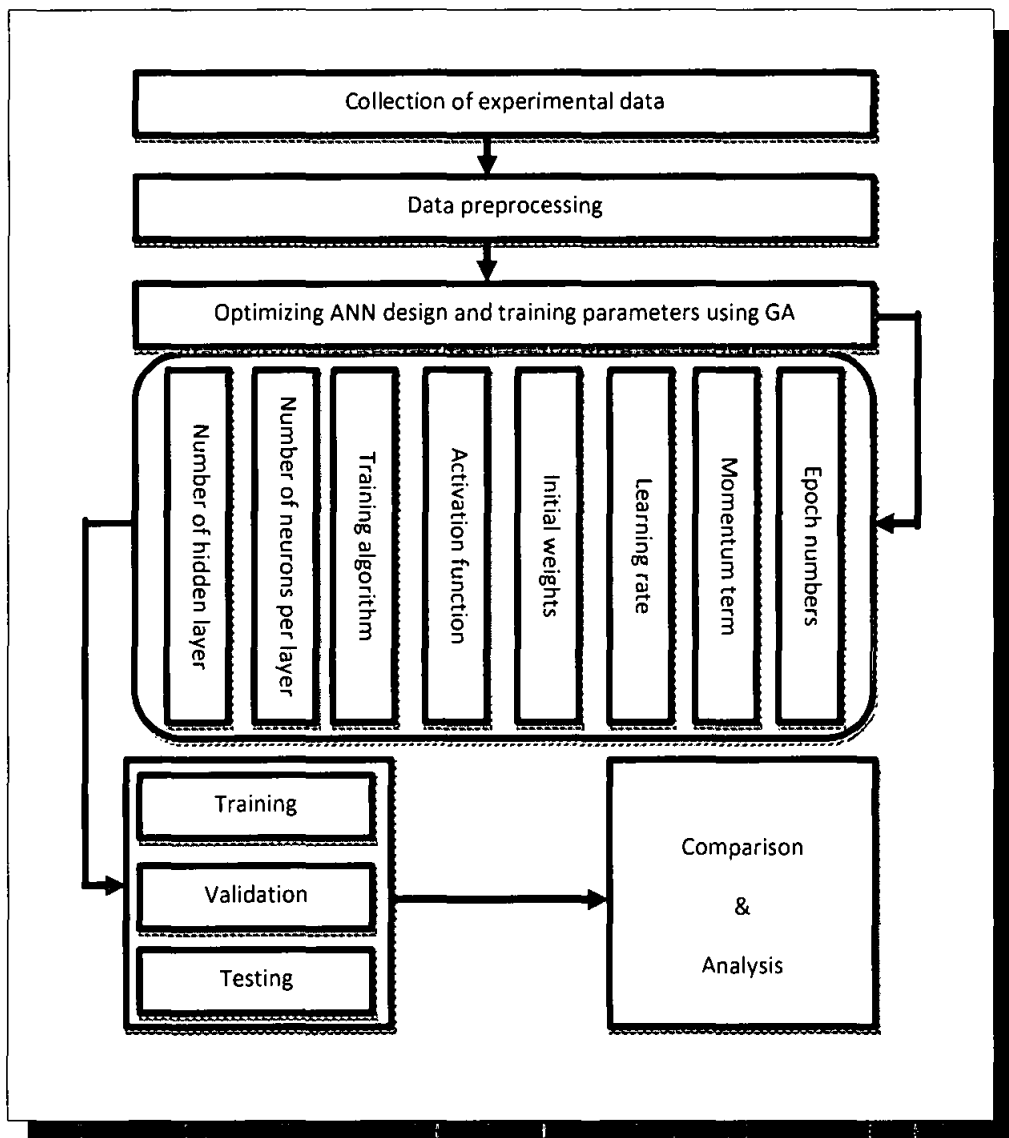


Figure 3.1 The research methodology steps

### 3.3.1 Collection of experimental data

Four different datasets were collected and used to obtain GA-ANN approach. One of them is from Universiti Teknologi PETRONAS GDC plant (TAURUS 60 gas turbine single-shaft generator set) collected during Jan. to Feb. 2008 period. Another dataset was collected from PETRONAS Penapisan (Melaka) Sdn Bhd from Jan. to Feb. 2007. The third dataset is a published experimental dataset of flank wear for drilling process (Panda, Chakraborty et al. 2008). Lastly, standard XOR problem dataset was used to ensure the applicability of propose GA-ANN approach.

### 3.3.1.1 *Data preprocessing*

Multilayer feed-forward ANN has shown the capability to process data from a wide range of sources. They are, however, only capable to process the data in a certain format. In addition, the ways of data representation to the networks have an effect on the designing of successful network. Consequently, a certain quantity of data processing is necessary before presenting the training data to the network. Data preprocessing refers to: missing values removal, outlier detection, normalization of data to zero-mean and unit variance, and data partitioning.

### 3.3.1.2 *Missing data removal*

In this step, all data were crosschecked visually and statistically to ensure accuracy and validity of the input data. A simple MATLAB algorithm was created and used to remove the missing and non-number values. The following steps were used to identify and remove the missing data values:

*Step1: Load the data set.*

*Step2: Identify Not-a-Number in the data using isnan function.*

*Step3: Returns an array of the same size as dataset containing logical 1 (true), where the members of dataset are Not-a-Number and logical 0 (false), where they are not*

*Step4: Remove any rows containing Not-a-Number from the data.*

### 3.3.1.3 *Outlier detection and removal*

The detection and removal of data outside the range of a typical value for variable processing remains extremely significant for the neural network development because undetected outliers have negative impact on the performance of the ANN models. In this study, principal component analysis (PCA) is used to detect the outlier data.

- Principal component analysis (PCA)

Principal component analysis (PCA) (Eriksson et al. 2001; Bao et al. 2007) is a general multivariate statistical technique that has been adapted to various practical problems, including engineering, science, and econometrics.

An outlier can be detected simply by using a score plot of a PCA of the data (Eriksson, E. Johansson et al. 2001). Figure 3.2 demonstrates the score plot of the first two principal components ( $t[1]$ ,  $t[2]$ ) for all 402 observations that have been collected from the debutanizer unit. In this figure, 9 observations can be recognized as clear outlier values, and hence they have been removed from the bulk of data.

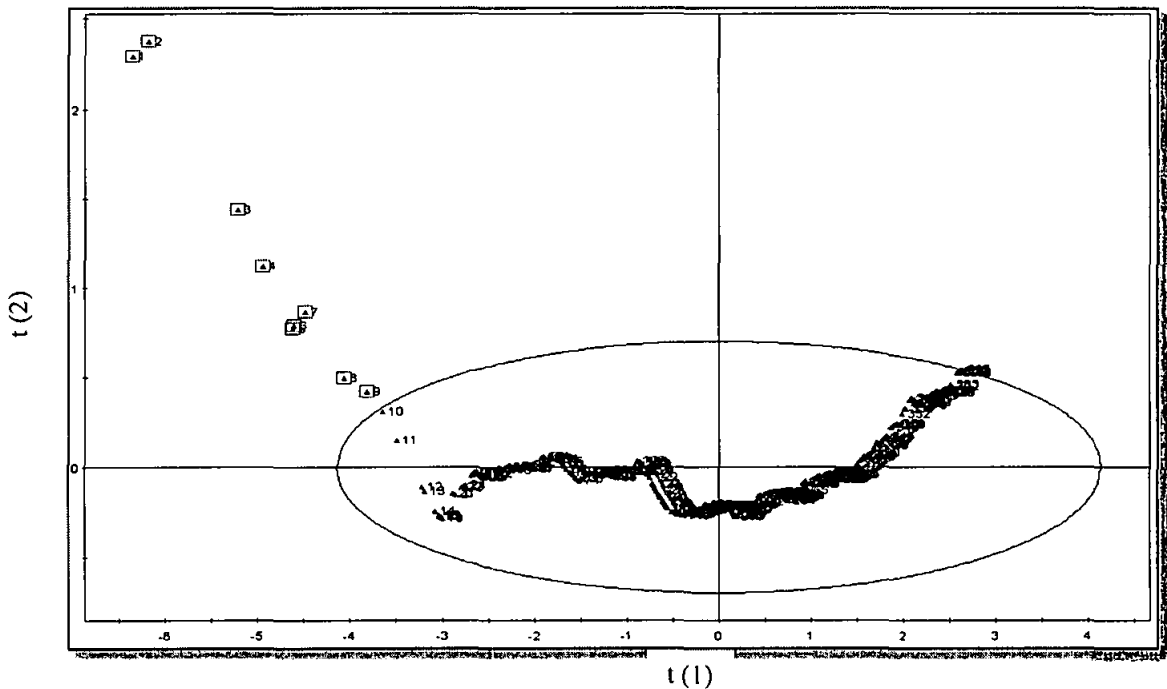


Figure 3.2 Score plot of the first two principal components of a PCA study for 402 observations from debutanizer

Figure 3.3 demonstrates the score plot of the first two principal components ( $t[1]$ ,  $t[2]$ ) for all 40 observations that have been collected from the gas turbine single shaft. In this figure, no observations can be recognized as clear outlier values, and hence all the data have been used to develop the ANN model.

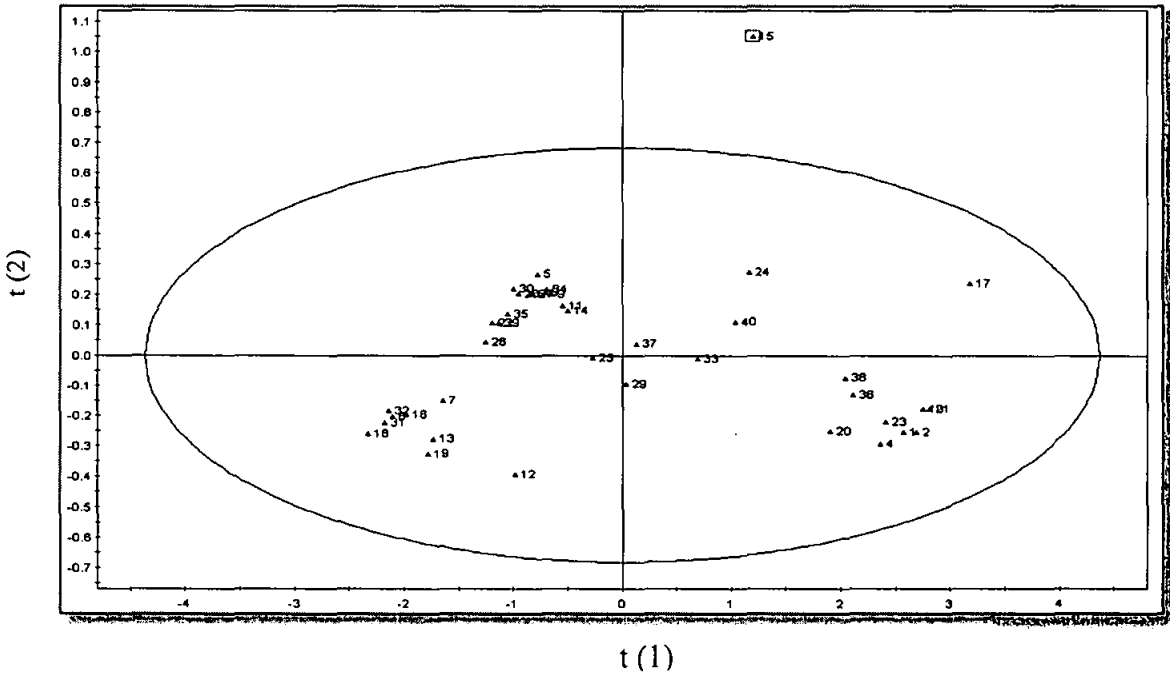


Figure 3.3 Score plot of the first two principal components of a PCA study for 40 gas turbine observations

Figure 3.4 demonstrates the score plot of the first two principal components ( $t[1]$ ,  $t[2]$ ) for all 64 observations that have been collected from the drilling process unit. In this figure, only one observation can be recognized as a clear outlier value, and hence it has been removed from the bulk of data.

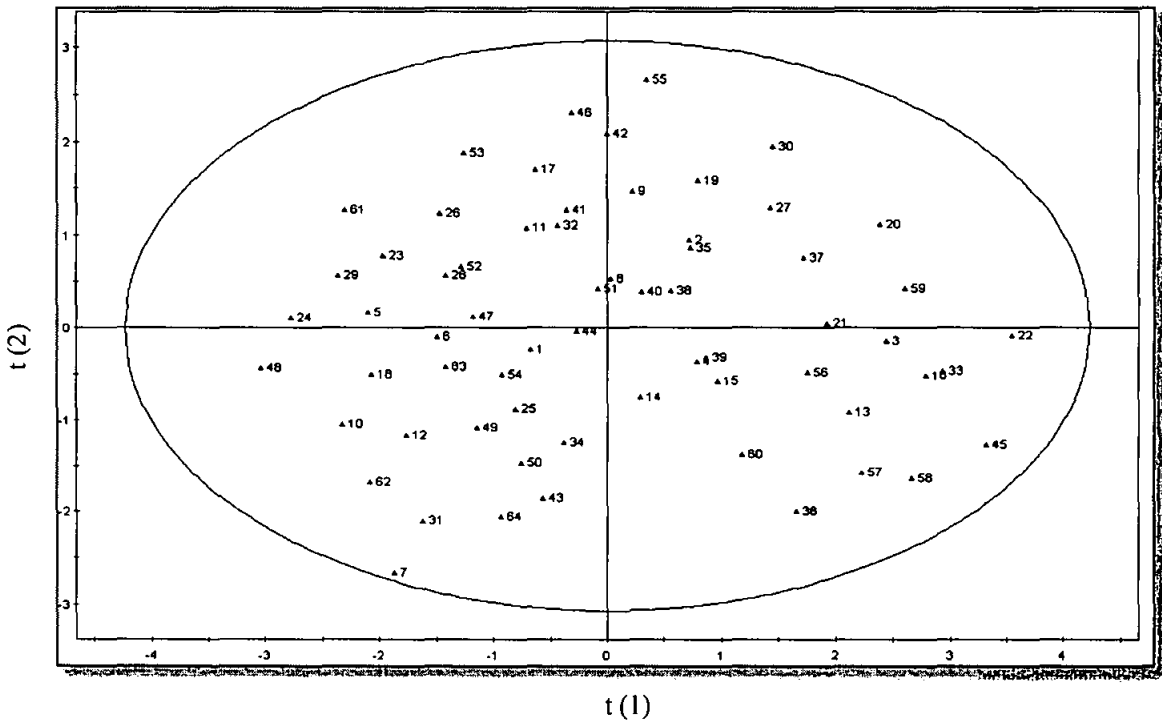


Figure 3.4 Score plot of the first two principal components of a PCA study for 64 flank wear observations

#### 3.3.1.4 Data normalization

The training, validation and testing dataset were scaled to the range of (0–1) using the modified MATLAB functions ‘premnmx’ and ‘trammx’. The following equation was used for the purpose:

$$x_{ni} = \frac{x_i - x_{min}}{x_{max} - x_{min}} \quad (3.1)$$

Where  $x_i$  is the real-world input value,  $x_{ni}$  is the normalized input value of the real-world input value  $x_i$ ,  $x_{min}$  and  $x_{max}$  are the corresponding minimum and maximum values of the un-normalized dataset.

The network predicted values, which were in the range of (0–1), were transformed to real-world values using the modified MATLAB function ‘postmnmx’. The equation below was used for the purpose:

$$x_i = x_{ni}(x_{max} - x_{min}) + x_{min} \quad (3.2)$$

#### 3.3.1.5 Data partitioning

In order to achieve the best performance of multilayer feed-forward ANN model, the experimental dataset were further randomly partitioned into three different sets: training, validation, and testing. The training set is used to train the multilayer feed-forward ANN by adjusting the connection weights and biases in a network. The validation set dictates that when the error remains constant for a predefined epoch numbers or begins to increase quickly, the training process should stop since the multilayer feed-forward ANN has started to over-fit the data; which means that the error of the network is driven to a small value for the training set but will become large when a new input data is presented. The testing set is used after finishing the training to examine the generalization capability of the multilayer feed-forward ANN model. In this study, the most popular ratio which is 4-1-1 has been used to partition the data.



### 3.3.2 Multilayer feed-forward back-propagation ANN

Four multilayer feed-forward back-propagation ANN models in the fields of petroleum, energy and standard XOR problem have been developed. The first multilayer feed-forward ANN model is used to predict iso-pentane ( $iC_5$ ) and normal pentane ( $nC_5$ ) of debutanizer CRU. The input layer contains three input neurons and represents the input variables, which are temperature, reflux flow, and flow rate. The output layer contains two neurons ( $iC_5$  and  $nC_5$ ). The second multilayer feed-forward ANN model is used to predict net power and turbine outlet temperature ( $T_4$ ) of gas turbine single shaft. The inputs of the model are ambient conditions ( $T_1$  and  $P_1$ ) and turbine inlet temperature ( $T_3$ ), while the outputs are net power and  $T_4$ .

The third multilayer feed-forward ANN model is used to predict flank wear of drilling process. Seven neurons are used to represent input variables, i.e. diameter, speed, feed, thrust, torque, feed vibration, and radial vibration, whereas flank wear is the output variable.

Lastly multilayer feed-forward ANN model is applied to predict the output of XOR problem. This model has two input neurons and one output neurons.

### 3.3.3 GA method

The combination of multilayer feed-forward ANN and GA technique is a powerful method for modeling and optimization purposes. In this work, the GA is applied to obtain the optimal multilayer feed-forward back-propagation ANN architecture and training parameters.

There are three major reasons for the implementation of a GA instead of a conventional optimization technique in this approach as listed below (Karr 1995):

- The GA works and focuses directly on a coded form of the problem's parameter set not on the parameters themselves.
- The GA process is started from a group of points of the solution space (initial population) not from a single point, therefore, reducing the chance of converging to local optima.

- The sampling process is conducted using the genetic operators (i.e., selection, crossover, and mutation) which are stochastic rules and not deterministic rules.

### 3.3.3.1 *GA encoding scheme*

The genetic algorithm has two major schemes for encoding the candidate solutions, namely direct encoding or ‘strong-specification’ (Miller et al. 1989) and indirect encoding or ‘weak representation’ (Yao 1999). Direct encoding scheme represents and encodes every ANN connection weights and neurons; by just looking at a bit string, it can encode the corresponding network structure. A bit of 1 corresponds to a connection, while a bit of 0 corresponds to lack of connection. Following this encoding scheme, chromosomes are easy to decode, but it requires very big binary strings. The scheme does not scale well to represent large ANN structures because it also needs very big string. These limitations involving feed-forward ANN and in addition the strings are of variable length.

The indirect encoding scheme is used to represent specific correspondence of specific binary strings to specific network architectures, (Ferentinos 2005; Marco and Hefin 2009) which have been pre-defined by the user. The property of the indirect encoding scheme looks more biologically reasonable than the direct encoding, one because genetic information in real chromosomes cannot specify the whole nervous system directly and independently (Yao 1993).

Furthermore, there is evidence that the direct encoding scheme is more appropriate for precise and deterministic handling of only small ANN (Ferentinos 2005; Marco and Hefin 2009).

Based on the results extracted from some applications of GA encoding in the works of previous researchers, the author recommends that indirect encoding scheme should be explored further.

### 3.3.3.2 Mapping multilayer feed-forward back-propagation ANN training and design parameters into GA binary encoding

In this work, multilayer feed-forward back-propagation ANN architecture and training parameters are represented by weak specification encoding scheme, which is the way of encoding several possible chromosomes of multilayer feed-forward ANN into specific genotypes, in the form of a simple binary string comprising a series of *0th* and *1th* in order to decode the decision variables of a specific problem. A chromosome, in this research, consists of the, training algorithm, number of hidden layers and neurons in each hidden layer, activation functions of the hidden and output neurons, initial weight, the learning rate, momentum term, and epoch size. A genotype is a sequence of bits (0 or 1) with a specific constant length. Each genotype corresponds to a unique chromosome.

Based upon the specified ranges for each multilayer feed-forward ANN architecture and training parameters in Table 3.1, in this study, string with a total of 42 bits of binary code is used. Appropriate gene (sub-string) is then allocated to each multilayer feed-forward ANN architecture and training parameters. For example, 2 bit sub-string (sub-string 1 in Figure 3.5) is used to represent the minimization algorithm of back propagation. The topology setting, i.e., the number of hidden layers, the number of neurons in the first layer, and the number of neurons in the second layer are represented by 6 bits of sub-string (sub-string 2 in Figure 3.5). Activation functions are represented by 2 bits of sub-string (sub-string 3 in Figure 3.5). The range of initial connection weight values, which are a real number, ranges from 0 to 1.0. In binary code, these values are represented using a 10 bit sub-string (sub-string 4 in Figure 3.5). Likewise, an 8 bit sub-string is used to represent learning rate value, which is also a real number (sub-string 5 in Figure 3.5). Whereas, the momentum term ranges from 0 to 1.0 and is represented by 10 bit sub-string (sub-string 6 in Figure 3.5). Finally the epoch numbers is represented by 4 bit sub-string (sub-string 7 in Figure 3.5).

Table 3.1 Range of multilayer feed-forward ANN architecture and training parameters

Name of multilayer feed-forward ANN parameters	Range
Training algorithm	$1 \leq \text{integer number} \leq 4$
Number of hidden layers	$1 \leq \text{integer number} \leq 2$
Number of neurons in the first layer	$3 \leq \text{integer number} \leq 30$
Number of neurons in the second layer	$3 \leq \text{integer number} \leq 8$
Activation function	$1 \leq \text{integer number} \leq 4$
Initial weight	$0 \leq \text{real number} < 1$
Learning rate	$0 < \text{real number} < 1$
Momentum term	$0 \leq \text{real number} < 1$
Epoch size	$200 \leq \text{integer} \leq 1600$

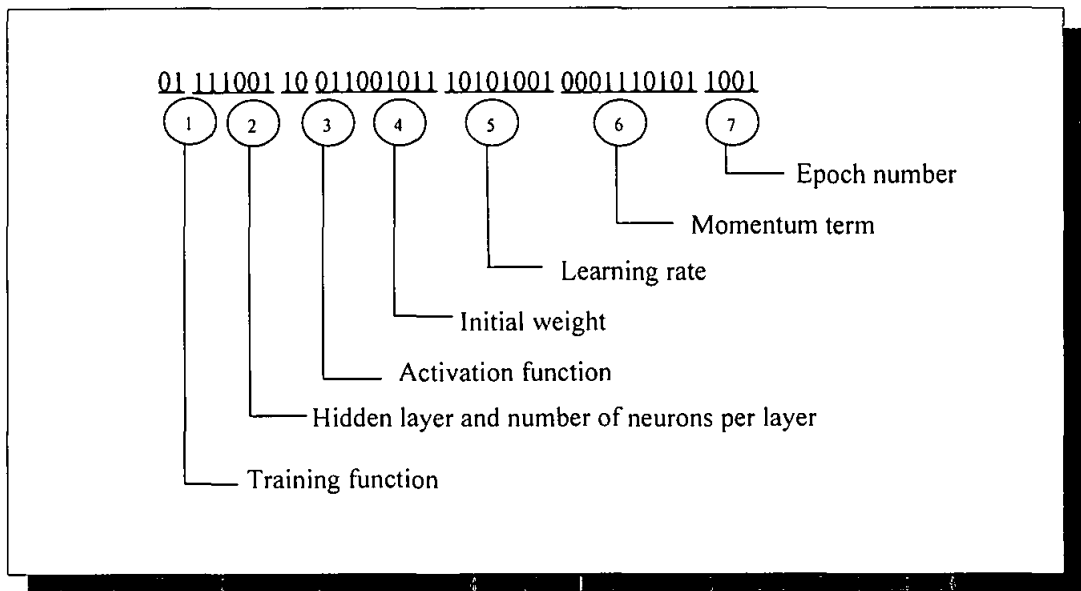


Figure 3.5 Binary representations of multilayer feed-forward ANN architecture and training parameters

### 3.3.3.3 Representation of the minimization algorithm of the training function

*Training or learning* a network consists of adjusting its connection weights using a training algorithm. Once the connection weights and biases are initialized, the

network is ready for training process. The network can be trained for predictive purpose as it is the case in the present study or can be trained for other tasks.

Most of all the training algorithms use the gradient of the performance function (i.e., MSE) to determine how to adjust the connection weights to minimize performance. A technique called back-propagation is used to determine the gradient. This scheme involves performing calculations backward through the network, and the connection weights are shifted in the direction of the negative gradient.

The back propagation ANN has been extremely successful in solving several problems for the purpose of adjusting the connection weights of the networks during the training process (Holger and Graeme 1998). There are several training algorithms used in neural network applications. It is extremely difficult to forecast which of these training algorithms will be the appropriate one for any problem (Koker et al. 2007). Generally, it depends on various issues: the architecture of the networks, i.e., the number of hidden layers, connection weights and biases in the network, aimed error at the training process, and application area, for example, classification or function approximation, or pattern recognition problem. Besides, the data structure and consistency of the training set are also important issues that have an effect on the network accuracy and performance. In this study, four different training algorithms have been considered in the proposed GA encoding: quasi-Newton, steepest descent, conjugate gradient, and Levenberg–Marquardt algorithms. The location of training algorithm sub-string is shown in Figure 3.6.

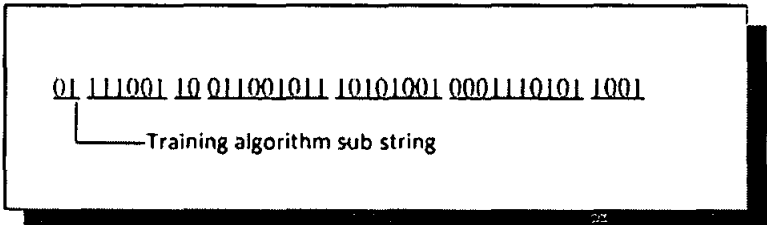


Figure 3.6 Training algorithm sub-string

Table 3.2 Binary encoding of training algorithm of the multilayer feed-forward ANN

The Encoded	The Decoded	Gene Size (bits)	Bit order in Chromosome
00	Steepest Descent	2	1-2
01	Quasi-Newton		
10	Levenberg–Marquardt		
11	Conjugate Gradient		

#### 3.3.3.4 Representation of the network structure

Since determining the best multilayer feed-forward ANN structure (i.e., number of hidden layers and number of neurons per layer) is the goal of this methodology, in reality, setting the number of hidden layers is problem dependent. The amount of hidden layers corresponds to the complexity of the problem to be solved, and the probable solution could be either a single or two hidden layer. Whereas, the selection of hidden neuron size to be contained in a given hidden layer is often a compromise between training time and accuracy of training. A larger number of hidden neurons results in a longer training time, while fewer hidden neurons provide faster training process at the cost of generalization performance. This calls for a method which would be able to quantify the differences that result from each examined ANN architecture. Thus, the GA method has been adopted to determine the best ANN architecture.

This GA method favours smaller multilayer feed-forward ANN structure by applying a binary encoding scheme. Advantages of smaller architectures include:

- Avoid over fitting of training data.
- Increase the generalization capability of multilayer feed-forward ANN because the network is just as complex as it needs to be.
- Increase the speed of training process because there are smaller numbers of hidden layers, neurons and connection weight factors to be calculated.

In this study, the 64 potential network structures of one-hidden-layer (1-HL) and two-hidden-layer (2-HL) networks are represented by the next 6 bit entries of the

string (Table 3.3). The location of network architecture sub-string is shown in Figure 3.7.

Table 3.3 Binary encoding of multilayer feed-forward ANN structure

The Encoded	The Decoded	The Encoded	The Decoded	The Encoded	The Decoded	Gene Size (bits)	Bit order in Chromosome
000000	1-HL-3	010101	1-HL-24	101011	2-HL-5/6	6	8
000001	1-HL-4	010110	1-HL-25	101100	2-HL-5/7		
000010	1-HL-5	010111	1-HL-26	101101	2-HL-5/8		
000011	1-HL-6	011000	1-HL-27	101110	2-HL-6/3		
000100	1-HL-7	011001	1-HL-28	101111	2-HL-6/4		
000101	1-HL-8	011010	1-HL-30	110000	2-HL-6/5		
000110	1-HL-9	011011	2-HL-3/3	110001	2-HL-6/6		
000111	1-HL-10	011100	2-HL-3/4	110010	2-HL-6/7		
001000	1-HL-11	011101	2-HL-3/5	110011	2-HL-6/8		
001001	1-HL-12	011110	2-HL-3/6	110100	2-HL-7/3		
001010	1-HL-13	011111	2-HL-3/7	110101	2-HL-7/4		
001011	1-HL-14	100000	2-HL-3/8	110110	2-HL-7/5		
001100	1-HL-15	100001	2-HL-4/3	110111	2-HL-7/6		
001101	1-HL-16	100010	2-HL-4/4	111000	2-HL-7/7		
001110	1-HL-17	100011	2-HL-4/5	111001	2-HL-7/8		
001111	1-HL-18	100100	2-HL-4/6	11010	2-HL-8/3		
010000	1-HL-19	100101	2-HL-4/7	111011	2-HL-8/4		
010001	1-HL-20	100110	2-HL-4/8	111100	2-HL-8/5		
010010	1-HL-21	100111	2-HL-5/3	111101	2-HL-8/6		
010011	1-HL-22	101000	2-HL-5/4	111110	2-HL-8/7		
010100	1-HL-23	101001	2-HL-5/5	111111	2-HL-8/8		

*1-HL-x: one-hidden-layer with x hidden neurons; 2-HL-x/y: two-hidden-layer with x hidden neurons in the first hidden layer and y hidden neurons in the second hidden layer.*

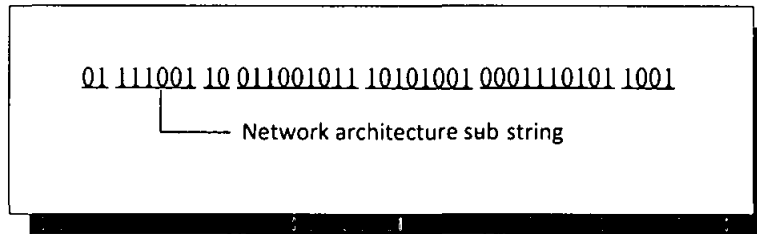


Figure 3.7 Network structure sub-string

### 3.3.3.5 Representation of the activation function

In the present study, three available activation functions i.e. logistic function (*logsig*), hyperbolic tangent-sigmoid (*tansig*) and linear functions (*purelin*) have been considered. The function logistic sigmoid generates outputs between 0 and 1 as the neuron's net input goes from negative to positive infinity, whereas, the activation function tangent-sigmoid generates outputs between  $-1$  and  $1$  for the same range of input data. Figure 3.8 shows the three types of activation functions that are used in this study, whereas the location of the activation functions sub-string is shown in Figure 3.9.

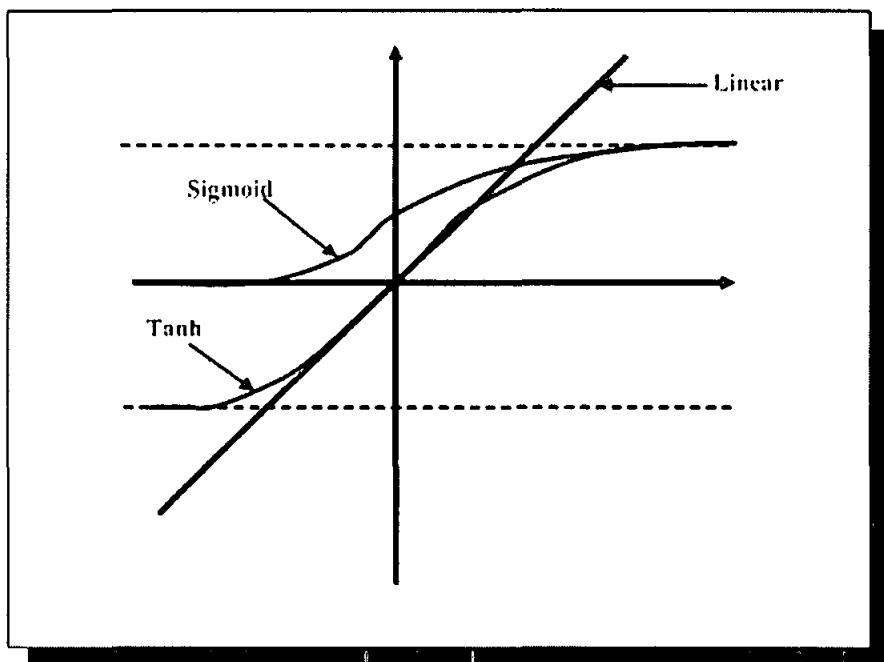


Figure 3.8 Sigmoid, tanh and linear activation function



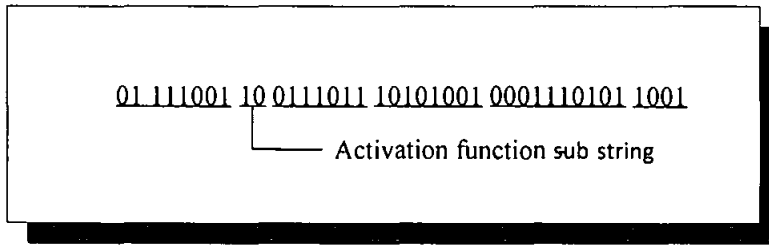


Figure 3.9 Activation function sub-string

The 9<sup>th</sup> to 10<sup>th</sup> genes have been used to represent the type of activation function of hidden and output neurons (Table 3.4).

Table 3.4 Binary encoding of activation functions of hidden and output neurons of the multilayer feed-forward ANN

The Encoded	The Decoded		Gene Size (bits)	Bit order in Chromosome
	Hidden neuron's Activation function	Output neuron's Activation function		
00	Logistic	Logistic	2	9-10
01	Logistic	Pure Linear		
10	Tangent sigmoid	Pure Linear		
11	Tangent sigmoid	Logistic		

### 3.3.3.6 Representation of the initial weight

The 11th to 20th genes represent the discrete value of initial weight with which the network was trained (Table 3.5 and 3.6). The location of the initial weight sub-string is shown in Figure 3.10.

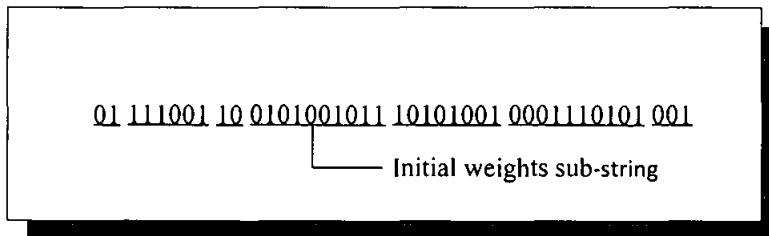


Figure 3.10 Initial weights sub-string

Table 3.5 Binary encoding of initial weights value of the multilayer feed-forward ANN (1/2)

The Encoded	The Decoded	The Encoded	The Decoded	The Encoded	The Decoded	Gene Size (bits)	Bit order in Chromosome
000000000	0.000341	0000010101	0.023632	0000101011	0.049632	11	11-20
000000001	0.001198	0000010110	0.024855	0000101100	0.05034		
000000010	0.001301	0000010111	0.025151	0000101101	0.050646		
000000011	0.001419	0000011000	0.026107	0000101110	0.051314		
000000100	0.003394	0000011001	0.029332	0000101111	0.051436		
000000101	0.004580	0000011010	0.030270	0000110000	0.051448		
000000110	0.005834	0000011011	0.030385	0000110001	0.052192		
000000111	0.007349	0000011100	0.032073	0000110010	0.053863		

Table 3.6 Binary encoding of initial weights value of the multilayer feed-forward ANN (2/2)

The Encoded	The Decoded	The Encoded	The Decoded	The Encoded	The Decoded	Gene Size (bits)	Bit order in Chromosome
0000001000	0.008648	0000011101	0.032940	0000110011	0.053978	11	11-20
0000001001	0.009333	0000011110	0.033179	0000110100	0.055953		
0000001010	0.009759	0000011111	0.034866	0000110101	0.055976		
0000001011	0.010979	0000100000	0.035423	0000110110	0.056343		
0000001100	0.011681	0000100001	0.036114	0000110111	0.056933		
0000001101	0.014362	0000100010	0.036382	0000111000	0.057340		
0000001110	0.015645	0000100011	0.036426	0000111001	0.057654		
0000001111	0.016675	0000100100	0.039184	0000011010	0.059031		
0000010000	0.017173	0000100101	0.043390	0000111011	0.059095		
0000010001	0.018613	0000100110	0.047078	.....	.....		
0000010010	0.019621	0000100111	0.047787	.....	.....		
0000010011	0.019765	0000101000	0.048739	1111111110	0.999329		
0000010100	0.020618	0000101001	0.049213	1111111111	0.999478		

### 3.3.3.7 Representation of the learning rate

The 21th to 28th genes represent the discrete value of learning rate with which the network was trained (Table 3.7). The location of the learning rate sub-string is shown in Figure 3.11.

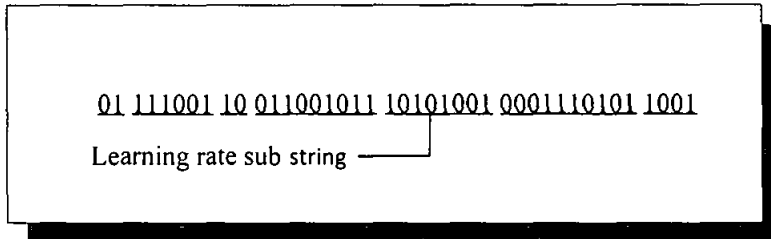


Figure 3.11 Learning rate sub-string

Table 3.7 Binary encoding of learning rate value of the multilayer feed-forward ANN

The Encoded	The Decoded	The Encoded	The Decoded	The Encoded	The Decoded	Gene Size (bits)	Bit order in Chromosome
00000000	0.009802252	00010101	0.09427839	00101011	0.189206843	8	21-28
00000001	0.018177534	00010110	0.09908965	00101100	0.19324533		
00000010	0.019257477	00010111	0.099095282	00101101	0.195476764		
00000011	0.021555887	00011000	0.101533889	00101110	0.210145637		
00000100	0.025857471	00011001	0.106941659	00101111	0.217732068		
00000101	0.029991950	00011010	0.107888905	00110000	0.217801594		
00000110	0.031922630	00011011	0.121658454	00110001	0.224277071		
00000111	0.041819864	00011100	0.122020518	00110010	0.227712826		
00001000	0.042297798	00011101	0.124774041	00110011	0.230383067		
00001001	0.042659856	00011110	0.125654587	00110100	0.231237816		
00001010	0.044165572	00011111	0.133503860	00110101	0.236444933		
00001011	0.046191556	00100000	0.137546595	00110110	0.240904997		
00001100	0.047401462	00100001	0.137762893	00110111	0.244165287		
00001101	0.047554673	00100010	0.137868992	00111000	0.248628960		
00001110	0.054616615	00100011	0.138601716	00111001	0.257846170		
00001111	0.054791790	00100100	0.138724636	0011010	0.268438821		
00010000	0.066946258	00100101	0.152234013	00111011	0.269054732		
00010001	0.070684335	00100110	0.177123754	.....	.....		
00010010	0.082592727	00100111	0.178982479	.....	.....		
00010011	0.083873508	00101000	0.182141076	11111110	0.996156111		
00010100	0.087077220	00101001	0.182227506	11111111	0.99756035		

### 3.3.3.8 Representation of the momentum term

The 29th to 38th genes represent the discrete value of momentum term with which the network was trained (Table 3.8). The location of the momentum term sub-string is shown in Figure 3.12.

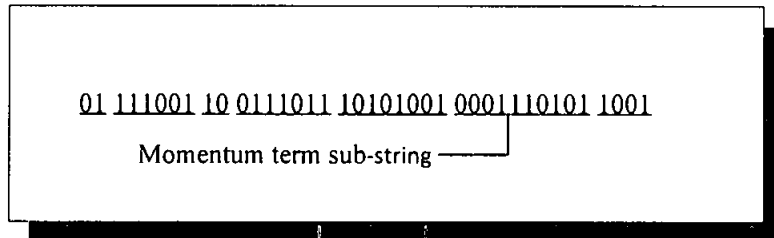


Figure 3.12 Momentum term sub-string

Table 3.8 Binary encoding of momentum term value of the multilayer feed-forward ANN

The Encoded	The Decoded	The Encoded	The Decoded	The Encoded	The Decoded	Gene Size (bits)	Bit order in Chromosome
000000000	0.000341	0000010101	0.023632	0000101011	0.049632	10	29-38
000000001	0.001198	0000010110	0.024855	0000101100	0.05034		
000000010	0.001301	0000010111	0.025151	0000101101	0.050646		
000000011	0.001419	0000011000	0.026107	0000101110	0.051314		
000000100	0.003394	0000011001	0.029332	0000101111	0.051436		
000000101	0.004580	0000011010	0.030270	0000110000	0.051448		
000000110	0.005834	0000011011	0.030385	0000110001	0.052192		
000000111	0.007349	0000011100	0.032073	0000110010	0.053863		
0000001000	0.008648	0000011101	0.032940	0000110011	0.053978		
0000001001	0.009333	0000011110	0.033179	0000110100	0.055953		
0000001010	0.009759	0000011111	0.034866	0000110101	0.055976		
0000001011	0.010979	0000100000	0.035423	0000110110	0.056343		
0000001100	0.011681	0000100001	0.036114	0000110111	0.056933		
0000001101	0.014362	0000100010	0.036382	0000111000	0.057340		
0000001110	0.015645	0000100011	0.036426	0000111001	0.057654		
0000001111	0.016675	0000100100	0.039184	0000011010	0.059031		
0000010000	0.017173	0000100101	0.043390	0000111011	0.059095		
0000010001	0.018613	0000100110	0.047078	.....	.....		
0000010010	0.019621	0000100111	0.047787	.....	.....		
0000010011	0.019765	0000101000	0.048739	1111111110	0.999329		
0000010100	0.020618	0000101001	0.049213	1111111111	0.999478		

### 3.3.3.9 Representation of the number of epochs

The 39th to 42nd genes represent the discrete value of epochs with which the network was trained (Table 3.9). The location of the momentum term sub-string is shown in Figure 3.13.

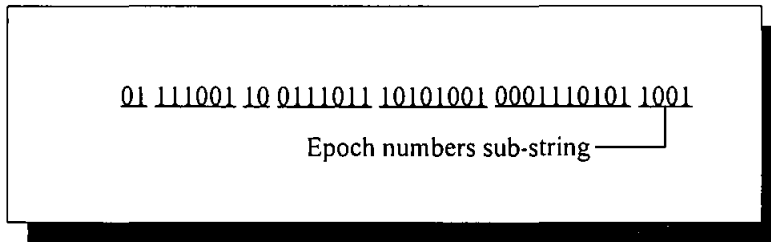


Figure 3.13 Epoch numbers sub-string

Table 3.9 Binary encoding numbers epoch of the multilayer feed-forward ANN

The Encoded	The Decoded	The Encoded	The Decoded	Gene Size (bits)	Bit order in Chromosome
0000	100	1000	900	4	39-42
0001	200	1001	1000		
0010	300	1010	1100		
0011	400	1011	1200		
0100	500	1100	1300		
0101	600	1101	1400		
0110	700	1110	1500		
0111	800	1111	1600		

### 3.3.3.10 Generation of initial population

Proposed GA-ANN approach was started with an initial population of elements which contains a predefined number of chromosomes (strings). To obtain the GA-ANN approach, the GA optimization parameters such as population size, generation numbers, mutation rate, and crossover rate were taken different values.

### 3.3.3.11 Multilayer feed-forward ANN prediction

After the initial population has been generated, the population was fed into a trained multilayer feed-forward ANN for prediction purpose. The input of the network was a

set of architecture and training process parameters generated by the GA optimization part.

#### 3.3.3.12 *Fitness evaluation*

A fitness function was recognized to evaluate the fitness of individual population. In our case, the fitness of each individual string is the negative of mean square error (MSE). To keep the values as positive Eq. 3.3 (Ferentinos 2005) was used:

$$fitness = 10^6 - \frac{1}{n} \cdot \sum_{i=1}^k (y_{obs} - y_i^o)^2 \quad (3.3)$$

Where  $y_i^o$  is the desired output and  $y_{obs}$  is the observed output.

#### 3.3.3.13 *Termination criteria*

The evaluation processes will continue until some termination criteria are applied. In this case, the maximum number of generation is defined as the termination criteria in GA optimization.

#### 3.3.3.14 *Creation of a new population*

In this approach, new population is generated by applying reproduction operator (selection) and recombination operators (crossover and mutation).

### **3.4 Performance evaluation**

An important issue in multilayer feed-forward ANN application is the selection of the performance evaluation function. Most applications use the mean square error (MSE) to evaluate the efficiency of ANN performance as given by:

$$MSE = \frac{1}{n} \cdot \sum_{i=1}^k (y_{obs} - y_i^o)^2 \quad (3.4)$$

Where  $n$  denotes the size of data patterns,  $y_{obs}$  is observed output of  $k$ th pattern, and  $y_i^o$  is desired output of  $k$ th pattern.

Also, in order to examine how close the efficiency is to the actual output, the root mean squared error (RMSE), correlation coefficient (R), and coefficient of determination ( $R^2$ ) were employed:

$$RMSE = \sqrt{\frac{1}{n} \cdot \sum_{i=1}^k (y_{obs} - y_i^o)^2} \quad (3.5)$$

$$R = \frac{\sum_{k=1}^n (y_{obs} - \bar{y}_{obs})(y_i^o - \bar{y}_i^o)}{\sqrt{\sum_{k=1}^n (y_{obs} - \bar{y}_{obs})^2 \sum_{k=1}^n (y_i^o - \bar{y}_i^o)^2}} \quad (3.6)$$

$$R^2 = \left[ \frac{\sum_{k=1}^n (y_{obs} - \bar{y}_{obs})(y_i^o - \bar{y}_i^o)}{\sqrt{\sum_{k=1}^n (y_{obs} - \bar{y}_{obs})^2 \sum_{k=1}^n (y_i^o - \bar{y}_i^o)^2}} \right]^2 \quad (3.7)$$

Where  $n$  is equal to total number of training dataset.

### 3.5 Summary

All of the research methodology steps that have been discussed in the previous sections are significant in the determination of multilayer feed-forward ANN architecture and training parameters using GA method. At the beginning, MATLAB 2009a software with neural networks toolbox has been chosen for developing the new approach. Then, four different data sets were collected to examine the efficiency and accuracy of the new approach. A comprehensive data preprocessing techniques have been provided, this include removal of missing value, detection and removal of data outlier, data partitioning, and data normalization. Afterward, the multilayer feed-

forward ANN architecture and training parameters are represented using binary GA encoding. At the end, an evaluation criterion of the new approach performance has been discussed.



## Chapter 4

### GA-ANN DEVELOPMENT APPROACH

#### **4.1 Introduction**

This chapter presents the algorithm that has been designed and developed to improve multilayer feed-forward ANN architecture and training parameters. Initially, the general approach has been explored. Then the chapter describes the four different applications that have been used to examine the new approach. The new GA-ANN approach described in this chapter is designed to be usable for any kinds of science and engineering modeling domains. To implement the GA-ANN approach, a large hierarchy of MATLAB code has been created, executed and these are presented in a format similar to algorithm.

#### **4.2 General approach**

The application of multilayer feed-forward ANN in modeling nonlinear processes has a central drawback: the lack of proper method to select the most appropriate network architecture, activation functions of hidden and output neurons and the parameters of the training algorithm. These tasks are usually based on a “trial and error” method performed by the developer of the model. Thus, optimality, generalization capability, training stability, prediction effectiveness and accuracy are not guaranteed, as the explored space is just a small piece of the whole search space and the type of search is random. To overcome the problems associated with human network design and training parameterization, an automatic method, based on the evolutionary features of the GA, is developed. GA evolves several network designs with different activation functions and several parameters of the training algorithm so that the best possible combination is finally chosen.

In the vast majority of approaches in the literature, the parameters of training algorithm such as learning rate, momentum term and epoch size are not considered in the encoding of the genetic algorithm method. In addition, some possible *a priori* knowledge of the system characteristics and possible general intuitions about the expected architecture of the multilayer feed-forward, were not taken into account. Such an *a priori* knowledge can drastically limit the huge search space of the problem of multilayer feed-forward ANN design, and more dimensions of the problem, like the parameters of training algorithm or the types of activation functions of hidden and output neurons, can also be encoded into the GA without making the encoding very complex and difficult to be optimized.

### **4.3 Applications**

Four different datasets were collected and used to obtain new GA-ANN approach. One of them is from Universiti Teknologi PETRONAS GDC plant (TAURUS 60 gas turbine single-shaft generator set) collected during Jan. to Feb. 2008 period. Another dataset was collected from PETRONAS Penapisan (Melaka) Sdn Bhd from Jan. to Feb. 2007. The third dataset is a published experimental dataset of flank wear for drilling process (S.S. Panda, D. Chakraborty et al. 2008). Lastly, standard XOR problem dataset was used to ensure the applicability of propose GA-ANN approach. Moreover, these datasets were partitioned into three different sets: training, validation and testing in the ratio of 4:1:1 (suggested by Haykin (1999))

#### **4.3.1 Debutanizer of CRU**

Debutanizer process (E. Almeida et al. 2000) in CRU is an important step in oil refining process; it is usually utilized to fractionate the straight-run naphtha from the Crude Distillation Unit, thus essentially removing propane ( $C_3$ ) and butanes ( $C_4$ ). The debutanizer column has two parts, the first part is overhead that is used to feed the LPG, which is a light saturated paraffinic hydrocarbon derived from the refining process of crude petroleum oil (Howard 2000; James and Glenn 2001; William 2004). LPG consists mainly of either  $C_3$  or  $C_4$ , or a combination of them, and some other hydrocarbons. They are mainly liquid under pressure, for transportation and storage

purposes. The second (bottom) part is used to feed hydrocarbon components for refining process.

The debutanizer column at PETRONAS Penapisan (Melaka) Sdn Bhd, shown in Figure 4.1, processes the light straight-run naphtha that comes from the product separator bottom through P-1301, and the liquid from outlet of the debutanizer overhead is sent to the LPG unit; while the bottom of C-1301 is sent to reformat tank as a reformatted product.

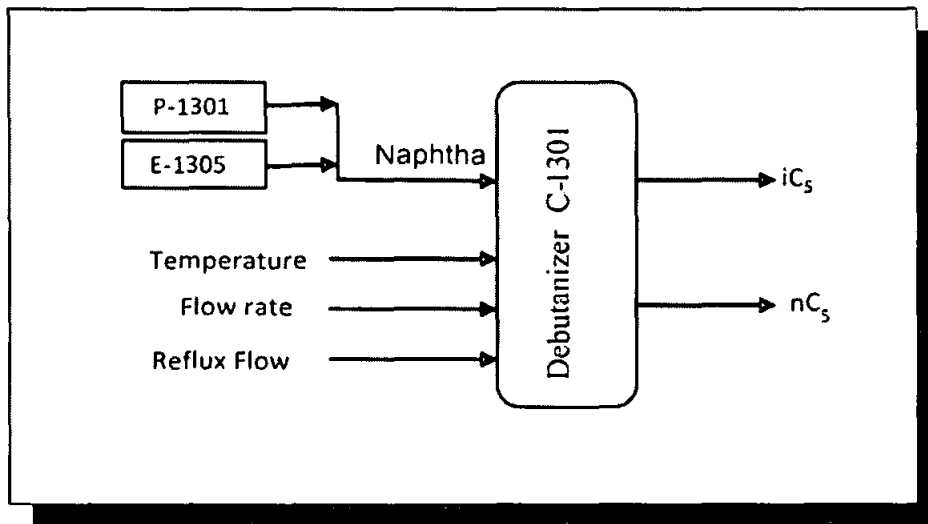


Figure 4.1 Debutanizer CRU Units at Melaka Refinery

#### 4.3.1.1 *Input and output variables*

One of the most important steps in developing a satisfactory multilayer feed-forward ANN prediction model is the selection of appropriate input variables; since these variables determine the structure of the multilayer feed-forward ANN model, and have an impact on the weighted coefficient and performance of the model. The model, called debutanizer, consisting of three input variables is constructed to predict iso-pentane ( $iC_5$ ) and normal-pentane ( $nC_5$ ).

The experimental data were collected from PETRONAS Penapisan (Melaka) Sdn Bhd from Jan. to Feb. 2007. The statistics of the data are presented in Table 4.1. The inputs for training are temperature, flow rate, reflux flow and the amount of iso-pentane ( $iC_5$ ) and normal pentane ( $nC_5$ ). The experimental data values reported are obtained from the mean of 400 observations collected from the CRU unit. The value

of temperature varies in the range of 183.57–195.67 °C, flow rate varies from 59.48 to 93.83 m<sup>3</sup>/hr, and reflux flow varies from 25.11 to 36.00 m<sup>3</sup>/hr. The value of iso-pentane (iC<sub>5</sub>) varies in the range from 2.27 to 3.28%, and normal pentane (nC<sub>5</sub>) varies from 1.34 to 2.06%.

Table 4.1 Statistical analysis of CRU debutanizer dataset

Variables	Training data (320 set)			Validation data (40 set)			testing data (40 set)		
	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean
Temperature °C	183.57	195.67	191.38	184.01	195.55	190.53	183.84	194.94	190.99
Flow rate m <sup>3</sup> /hr	59.48	93.83	88.37	81.60	93.69	88.15	72.41	93.20	87.66
Reflux Flow m <sup>3</sup> /hr	25.11	36.00	31.36	27.00	35.96	31.11	25.90	35.10	31.06
iC <sub>5</sub> %	2.27	3.28	2.70	2.31	3.20	2.68	2.29	3.11	2.66
nC <sub>5</sub> %	1.34	2.06	1.61	1.36	2.00	1.60	1.35	1.91	1.59

#### 4.3.1.2 Network architecture

The type of ANN that has been used to predict iso-pentane (iC<sub>5</sub>) and normal pentane (nC<sub>5</sub>) is the feed-forward with back-propagation training algorithm. This network type can solve the complicated function of desired approximation task. The back-propagation algorithm (BP) is one of the major types of multilayer feed-forward ANN that has gained widespread use. It contains three kinds of layers: input, hidden and output layer. All of the data information in BP flows in one way, “feed-forward”. The neurons of one layer are linked with the neurons of the next layer, there is no feedback. In this work, a fully linked BP was used where all neurons of two successive layers are linked with each other. Accordingly, the network consists of three neurons in the input layer representing the temperature, reflux flow and flow rate. The output layer consists of two neurons to represent iC<sub>5</sub> and nC<sub>5</sub>. Figure 4.2 shows a schematic diagram of the architecture of the feed-forward with back-propagation training algorithm for the debutanizer.

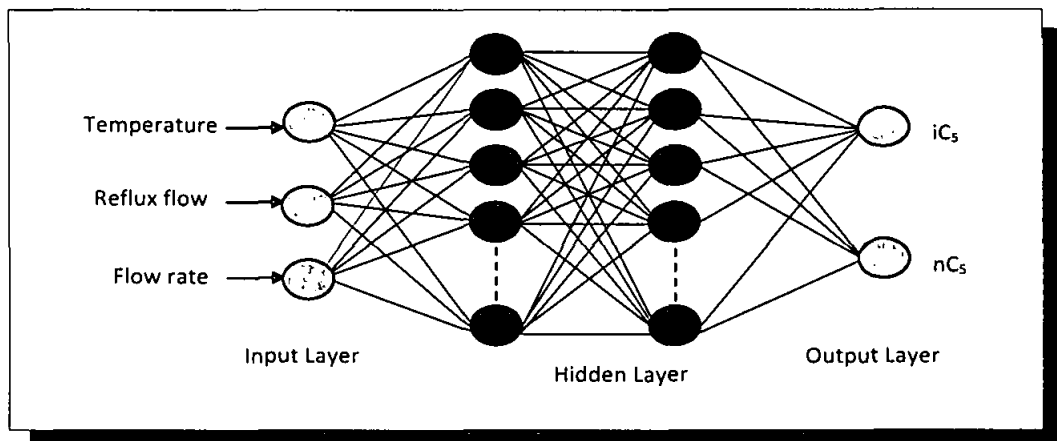


Figure 4.2 A schematic of the multilayer feed-forward ANN architecture for debutanizer

### 4.3.2 Gas turbine

A TAURUS 60 gas turbine is shown in Figure 4.3. The air is compressed in the engine compressor from state 1 to state 2. The heat added in the combustor brings the cycle from 2 to 3, and the hot gas is then expanded from 3 to 4.

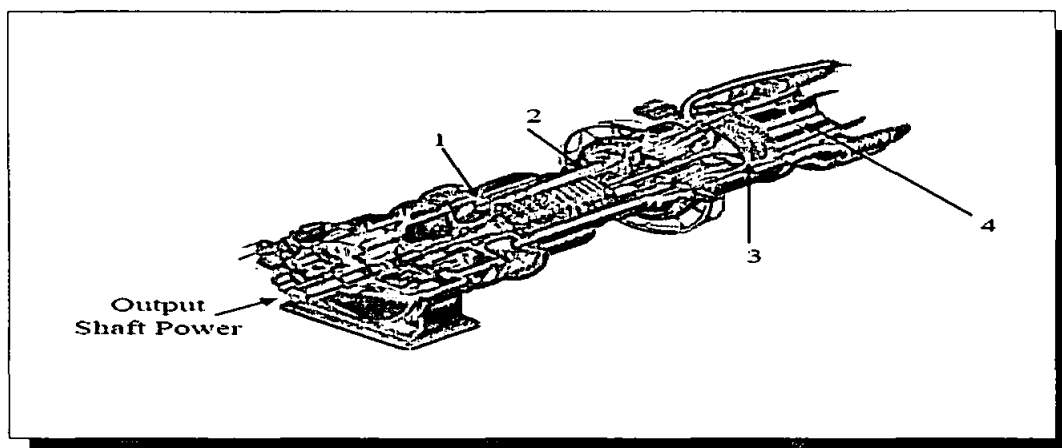


Figure 4.3 TAURUS 60 gas turbine

#### 4.3.2.1 Input and output variables

The dataset from actual operation at different load of gas turbine from Universiti Teknologi PETRONAS GDC plant (TAURUS 60 gas turbine single-shaft generator set) were collected during the period from Jan. to Feb. 2008. The statistics of the data are presented in Table 4.2. The inputs for training are ambient conditions ( $T_1$  and  $P_1$ ) and turbine inlet temperature ( $T_3$ ), while the outputs are net power and turbine outlet

temperature ( $T_4$ ). The experimental data values reported are obtained from the mean of 60 observations collected from the GDC plant. The value of  $T_1$  varies in the range from 299.5–308 °C, pressure from 647.7 to 922.2 KPag, and  $T_3$  varies from 572.5 to 581 °C. The value of power varies in the range from 2110 to 3236KW, and  $T_4$  varies from 500.74 to 2.06 °C.

The dataset was randomly divided into three separate datasets; training dataset consisting of a total of 48 dataset, validation dataset consisting of a total of 6 set and testing dataset consisting of 6 set.

Table 4.2 Statistical analysis of gas turbine dataset

Variables	Training data (48 set)			Validation data(6 set )			Testing data (6 set )		
	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean
Inlet temperature( $T_1$ )	299.5	308	303.21	299.7	304.2	301.26	301	306.7	304.4
Outlet temperature( $T_3$ )	572.5	581	575.98	572.7	577.2	574.26	301	306.7	304.4
Pressure ( $P_1$ )	647.7	922.2	767.57	656.4	796.9	718.16	722.7	852.6	804.46
Net power	2110	3236	2763.57	2210	2864	2540.4	2637	3119	2914.8
$T_4$	500.74	909.43	664.77	505.09	676.38	603.29	646.35	683.30	672.89

#### 4.3.2.2 Network architecture

Similar to the debutanizer model, a fully linked BP was used where all neurons of two successive layers are linked with each other. Accordingly, the network consists of three neurons in the input layer representing the inlet temperature ( $T_1$ ), outlet temperature ( $T_3$ ), and pressure. The output layer consists of two neurons to represent net power and  $T_4$ . Figure 4.4 shows a schematic diagram of feed-forward back-propagation network for gas turbine.

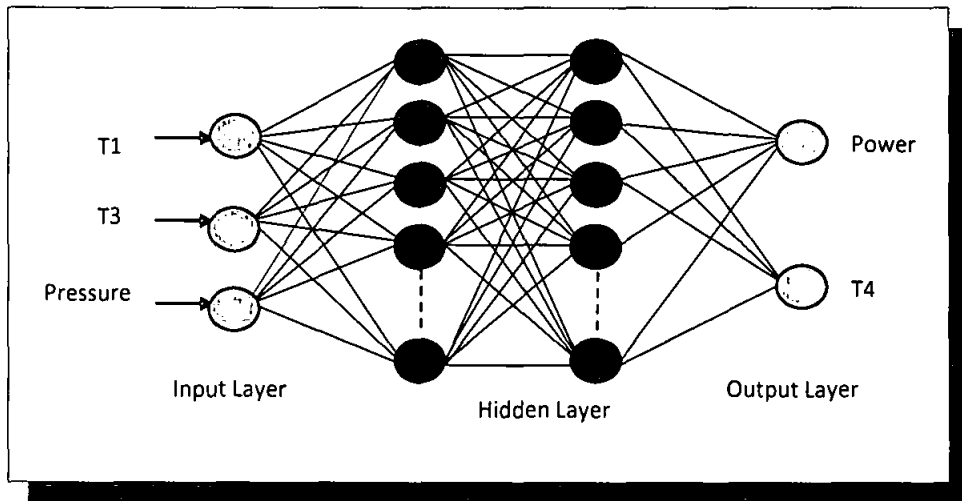


Figure 4.4 A schematic of the multilayer feed-forward ANN architecture for gas turbine

### 4.3.3 Drilling process

Drill wear is a significant factor that directly affects the whole surface quality and tool life of the drill (Toshiyuki and Jun 2004; Panda, Chakraborty et al. 2008). It is particularly important for decision making in tool condition monitoring. The prediction of tool wear before the tool causes any damage on the machined surface is extremely valuable in order to avoid loss of product, damage to the machine tool and associated loss in productivity (Tugrul and Abhijit 2002). Different types of intelligent systems have been successfully applied such as neural networks. Artificial neural networks have such a high learning ability that they have been applied to the classifications of more complicated conditions. Figure 4.5 shows a radial drilling machine.

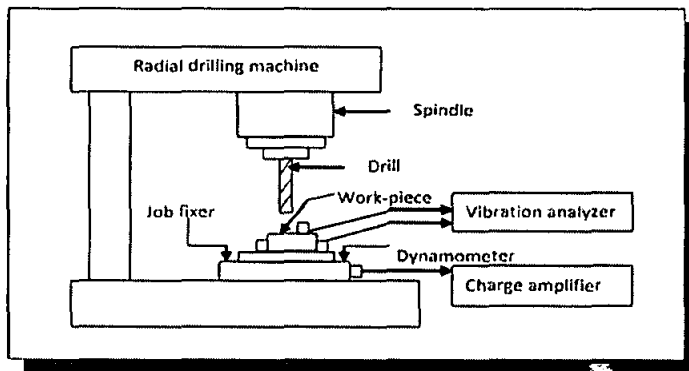


Figure 4.5 A radial drilling machine (Batliboi Limited, BR618 model)

#### 4.3.3.1 Input and output variables

A published experimental dataset of flank wear for drilling process (Panda, Chakraborty et al. 2008) collected from a radial drilling machine (Batliboi Limited, BR618 model) is used. The statistics of the data are presented in Table 4.3. The inputs for training are drill diameter, spindle speed, feed rate, thrust force, torque, feed vibration, and radial vibration while the output is flank wear. The experimental data values reported are obtained from the mean of 64 observations. The value of drill diameter varies in the range of 9–12 mm, spindle speed from 250 to 500 rpm, feed rate varies from 0.13 to 0.36 mm/rev, thrust force from 1088.1 to 3323.1 N, torque from 10.67 to 33.11 Nm, feed vibration from 15.46 to 61.51 m/s<sup>2</sup>, and radial vibration from 16.21 to 62.36 m/s<sup>2</sup>. The value of flank wear varies in the range of 0.06 to 0.24 mm.

The dataset was randomly divided into three separate datasets, training dataset includes a total of 50 data, validation dataset includes a total of 7 data and testing dataset includes 7 data.

Table 4.3 Statistical analysis of drilling machine dataset

Variables	Training data (50 set)			Validation data (7 set )			Testing data (7 set )		
	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean
Drill diameter	9	12	10.63	9	12	10.25	9	12	10.23
Spindle speed	250	500	364.87	250	500	365	250	500	361.15
Feed rate	0.13	0.36	0.23	0.13	0.36	0.22	0.13	0.36	0.24
Thrust force	1150.9	3311.2	2036.24	1088.1	3323.1	1817.91	1185.2	3284.2	2041.73
Torque	11.06	33.11	20.50	10.67	33.08	18.43	11.43	32.95	20.4
Feed vibration	17.21	57.11	35.51	18.63	55.46	33.94	15.46	61.51	33.77
Radial vibration	18.1	59.49	36.98	19.52	57.43	35.45	16.21	62.36	35.03
Flank wear	0.07	0.24	0.14	0.09	0.18	0.14	0.06	0.2	0.14



### 4.3.3.2 Network architecture

A diagram of feed-forward with back propagation training algorithm for drilling process is illustrated in Figure 4.6. The network has three layers, an input layer with seven input variables, a hidden layer and an output layer which gives the flank wear. The input variables are drill diameter, spindle speed, feed rate, thrust force, torque, feed vibration, and radial vibration, whereas flank wear is the output variable. The output neuron is flank wear.

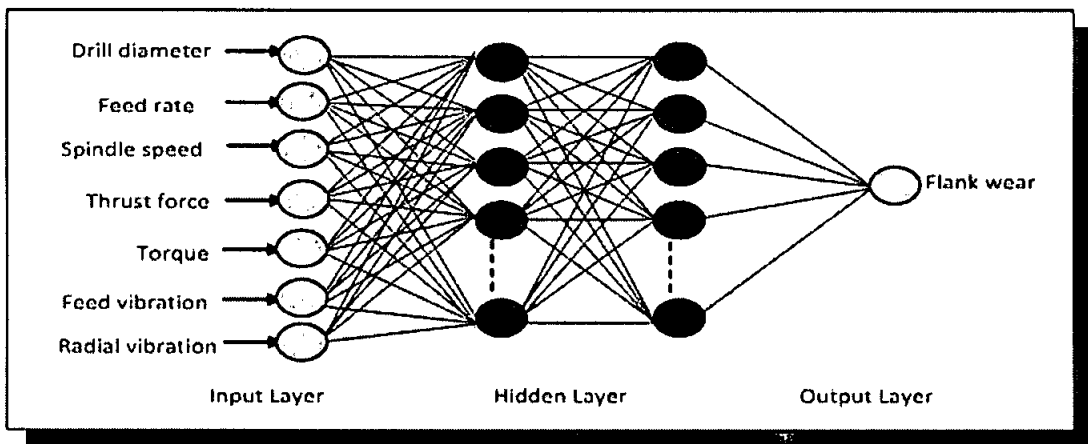


Figure 4.6 A schematic of the multilayer feed-forward ANN architecture for flank wear

### 4.3.4 XOR problem

Exclusive OR (XOR) is standard problem; the data consist of two binary input variables and a single binary output variable and has historically been considered as a benchmarking problem for the initial testing of different ANN models performance. XOR is a simple nonlinear function that cannot be emulated with a network with just an output layer.

#### 4.3.4.1 Input and output variables

The dataset of XOR problem consists of four training samples. If the two inputs are identical, the output is 0. If the inputs are different ( $\{0, 1\}$  or  $\{1, 0\}$ ), the output is 1. The truth table of this problem is shown in Table 4.4.

Table 4.4 XOR mapping

Input1	Input2	Output= $x \oplus y$
1	1	0
1	0	1
0	1	1
0	0	0

#### 4.3.4.2 Network architecture

A diagram of feed-forward with back propagation training algorithm for XOR problem is illustrated in Figure 4.7. The network has three layers, an input layer with two input variables, a hidden layer and an output layer which gives the output.

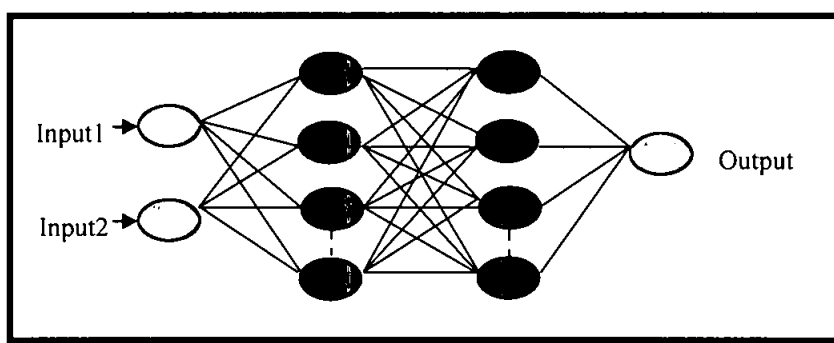


Figure 4.7 A schematic of the multilayer feed-forward ANN architecture for XOR

#### 4.4 The algorithm

The algorithm of the proposed GA-ANN consists of three main parts: the “user level” part, the “genetic algorithm” part and the “training” part. The first part deals with the input/output procedures, even as the other two parts, which have several sub-sections, interconnect with each other and with the first part to accomplish the desired process. The algorithm is shown sequentially in Figure 4.8. Each box in Figure 4.8 represents a separate function and the names of these functions are shown at the top of each box.

The links between functions are shown with the appropriate arrows. An explanation of the algorithm and the symbols involved follows:

- In the beginning, the user gives some parameters, which are input training set (P), the output training set (T), the number of generation of the GA (N), the

size of population ( $M$ ), and finally the crossover and mutation probabilities ( $P_c$ ) and ( $P_m$ ). This information is then passed on to another function called GA.

- The GA randomly generates an initial population of  $m$  individual chromosomes ( $X_{\text{initial}}$ ). Each of which represents specific network architecture, activation function, training algorithm, initial weight, learning rate, momentum term and epoch size.
- Repeat the next steps until the maximum number of generation ( $N$ ) is reached.
  - The population of binary string is passes to decoding function, where each string of the binary values is decoded into explicit information about the multilayer feed-forward ANN architecture and training parameters.
  - The fitness of each individual string is calculated.
  - According to this fitness value, the GA (function 'selection') selects the new group of string, which will continue as parents in the next generation of the GA ( $X_{\text{parents}}$ ):
  - The strings are then subjected to crossover operation with a given probability,  $P_c$ .
  - Perform mutation with a given probability,  $P_m$ .
  - Then the final population is formed ( $X_{\text{new}}$ ).
  - Decode each chromosome in the new generation. Train each network and compute the new MSE values after the training of each new chromosome.
- Finally the best string, that is the string that gives the minimum fitness, is returned to the user, together with its corresponding minimum MSE (best MSE) and some other information useful for statistical analysis.

The details about the GA parameters that were used (population sizes, probabilities of crossover and mutation, etc.) as well as the result of the applications, are presented in Chapter 5. The approach was developed in MATLAB with Neural networks Toolbox.

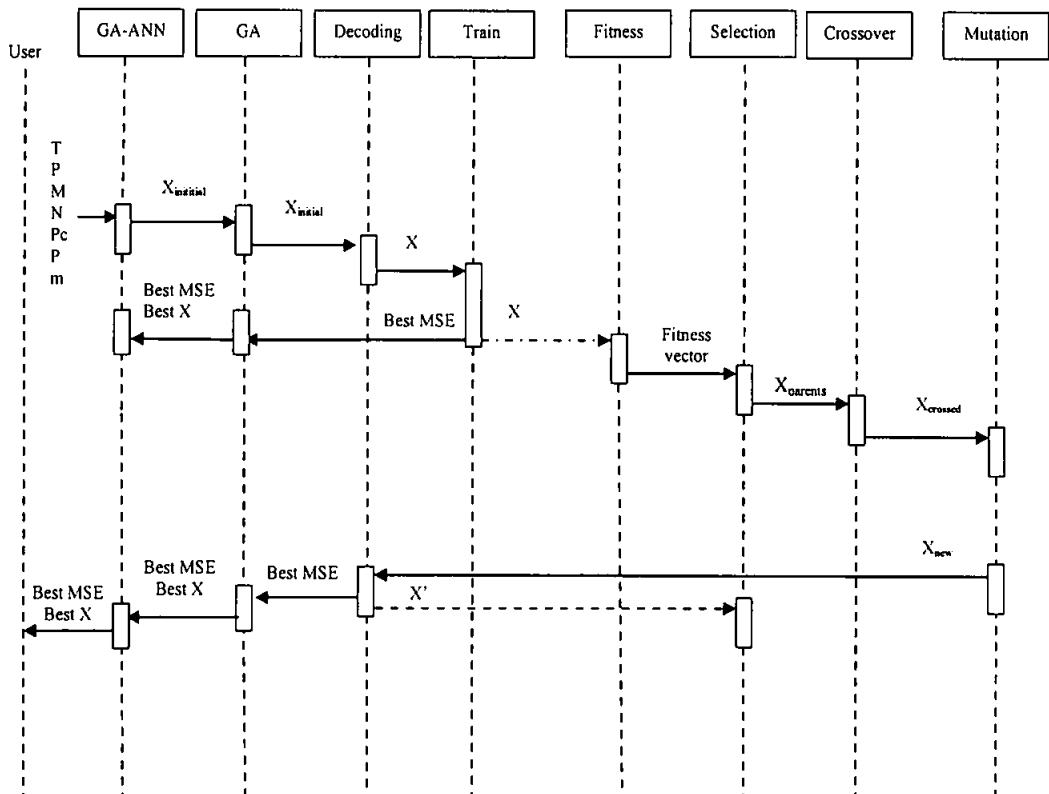


Figure 4.8 Sequence diagram of the algorithm

#### 4.5 Summary

The algorithm for improving multilayer feed-forward ANN performance has been developed. Therefore, the multilayer feed-forward ANN architecture and training parameters including, training algorithm, number of hidden layers, number of neurons in hidden layer, activation function, initial weight, learning rate, momentum term and epoch size are adopted using GA.

The new GA-ANN approach described in this chapter has been designed to be usable for any kinds of science and engineering modeling domains and is not limited to petroleum and energy domain alone. To implement the GA-ANN approach, a large hierarchy of MATLAB code has been created and executed, and these are presented in a format similar to algorithm.

The networks that have been developed as presented in sections 4.1.2, 4.2.2, 4.3.2 and 4.4.2 have an architecture that can handle almost all four applications. These networks have been used to predict iso-pentane ( $iC_5$ ) and normal pentane ( $nC_5$ ) for the CRU debutanizer, net power and  $T_4$  for gas turbine single shaft, flank wear for drilling process, and the output of XOR problem.

## Chapter 5

### RESULTS AND DISCUSSION

#### 5.1 Introduction

In this study, four different experimental datasets have been used to design and simulate the new approach. For the debutanizer of CRU unit the numbers of datasets for training, cross-validation and testing network are 320, 40 and 40, respectively. For the gas turbine the numbers of data sets for training, cross-validation and testing network are 40, 10 and 10, respectively. For the drilling process the numbers of data sets for training, cross-validation and testing network are 41, 12 and 12, respectively. Finally, for XOR problem the numbers of training networks are 4 datasets. The results of proposed GA-ANN, together with some additional information about its generalization capability and stability of multilayer feed-forward, prediction efficiency, accuracy, optimality and simulation time, are presented in this chapter. For judging the new GA method against the existing ones that are proposed to work in multilayer feed-forward ANN, likes most previous works the simulation has used as a verification method.

#### 5.2 GA method

The training sets for the prediction of  $iC_5$ ,  $nC_5$ , flank wear, net power and  $T_4$ , and finally the output of XOR problem with data collected as described in Chapter 5 were fed into the GA method.

The mechanism of GA method has been presented in Chapter 4. GA is a probabilistic adaptive optimization algorithm, thus, the entire optimization process must be repeated a number of times, and beginning from different random initial populations of possible solution each time. The fundamental parameters of GA that

must be explored are the number of individual in population ( $m$ ), the number of generations ( $n$ ), the probability of crossover ( $P_c$ ), and the probability of mutation ( $P_m$ ). In this work, a number of experiments have been conducted to determine the best possible ANN design and training parameters.

The initial populations of each run of the GA method, that is the initial values of the 42-bit binary strings as they are described in Chapter 3, were formed randomly and evaluated in each generation. One-point crossover which is one of the most common types of crossover is used (Goldberg, 1989a, b, c). Table 5.1 shows the optimal GA parameters that were used to achieve the results.

Table 5.1 GA parameters

Parameter	Optimal value
Population size	20
Maximum number of generation	30
Crossover probability $P_c$	0.95
Mutation probability $P_m$	0.05

### 5.2.1 Prediction of $iC_5$ and $nC_5$ using GA-ANN approach

Figure 5.1 shows the best mean squared errors (MSEs) attained by the GA-ANN approach as a function of the crossover probability for three different values of mutation probability. The best performance was achieved for  $P_c = 0.95$  and  $P_m = 0.05$ .

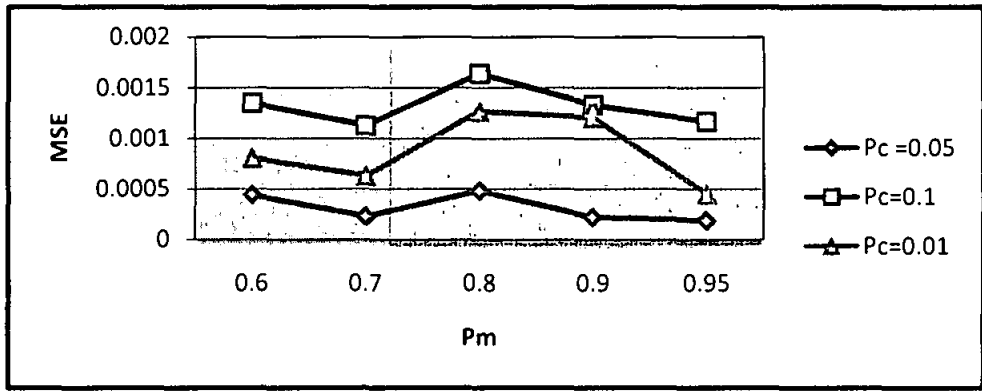


Figure 5.1 GA method performance for several crossover and mutation probabilities

In this study, the number of individual between 10 and 20 in the population were examined, and the best MSE values are shown in Figure 5.2. The best performance of GA method was achieved with the population size of 20. Further, two different values of generation numbers were explored, 20 and 30, and the results after 30 generations are generally better.

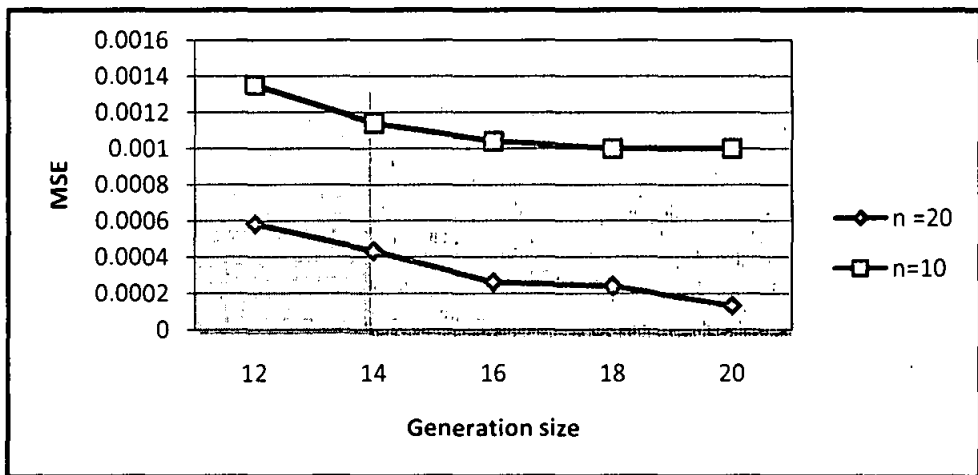


Figure 5.2 GA method performance for several population size and two values of generation size

Figure 5.3 shows the best MSE values found after each generation during the GA method process. In this run, the best MSE was found after 26 generations. The corresponding average MSE values of the entire population after each generation is shown in Figure 5.4.



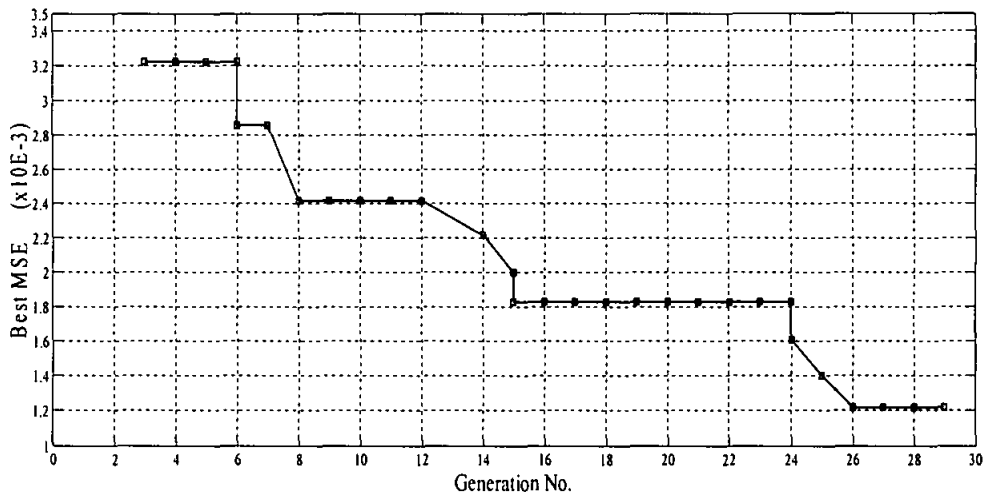


Figure 5.3 Best MSE found during GA process for the  $iC_5$  and  $nC_5$  prediction model

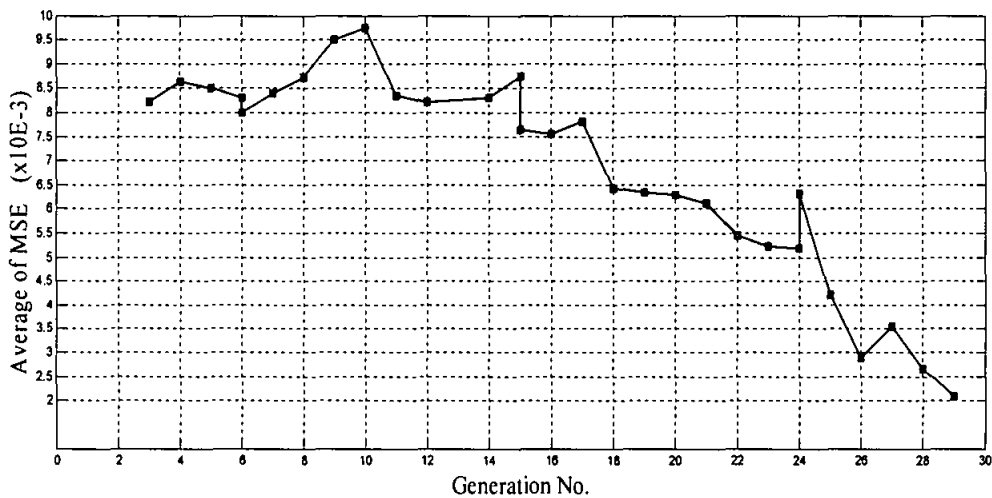


Figure 5.4 Average MSE of the entire population during GA process for the  $iC_5$  and  $nC_5$  prediction model

□ The first best solution found is the string:

**10 001001 01 1000000001 00000100 0000111110 0001**

Which are interpreted as a single hidden layer network with 15 neurons, logistic sigmoid activation function in hidden neurons and pure linear function in output neurons, trained with the Levenberg-Marquardt back-propagation algorithm that used a 0.519716 value as initial weight, by learning rate of 0.025857471, with a value for momentum term of 0.061401 after 200 epochs. This solution gave a MSE value of 0.0019 for  $iC_5$  and 0.0002 for  $nC_5$ .

□ The second best solution found is the string :

**10 000100 10 1100011011 00001100 0001100000 0101**

Which is interpreted as a single hidden layer with 7 neurons, tangent sigmoid activation function in hidden neurons, and tangent sigmoid in output neurons, trained with Levenberg-Marquardt back-propagation algorithm that used a 0.784855 value as initial weight, by learning rate of 0.047401462, with a value for momentum term of 0.095949 after 600 epochs. This solution gave MSE value of 0.0019 for  $iC_5$  and 0.00027 for  $nC_5$ .

□ The string :

**10 000100 10 1110110011 00000000 0000110101 0101**

is found as the third best solution, which was interpreted as one hidden layer with 7 neurons, tangent sigmoid activation function in hidden neurons and tangent sigmoid in output neurons, trained with Levenberg-Marquardt back-propagation algorithm that used a 0.932469 value as initial weight, by learning rate of 0.009802252, with a value for momentum term of 0.5 after 600 epochs. This solution gave MSE value of 0.0019 for  $iC_5$  and 0.00028 for  $nC_5$ .

### 5.2.2 Prediction of flank wear using GA-ANN approach

Figure 5.5 shows the best MSE values found after each generation during GA method process. In this run, the best MSE was found after 29 generations. The corresponding average MSE values of the entire population after each generation are shown in Figure 5.6.

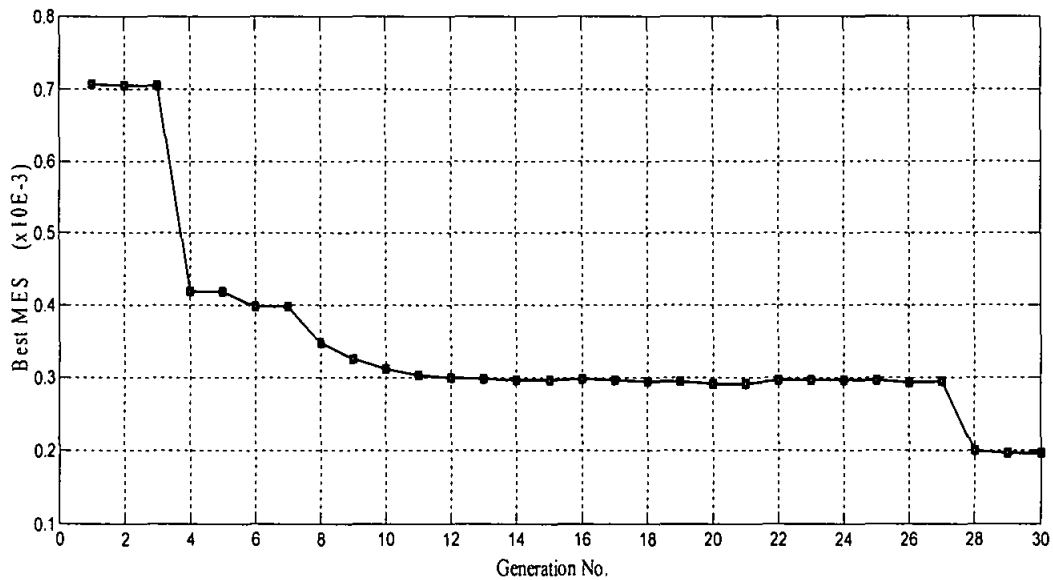


Figure 5.5 Best MSE found during GA process for the flank wear prediction model

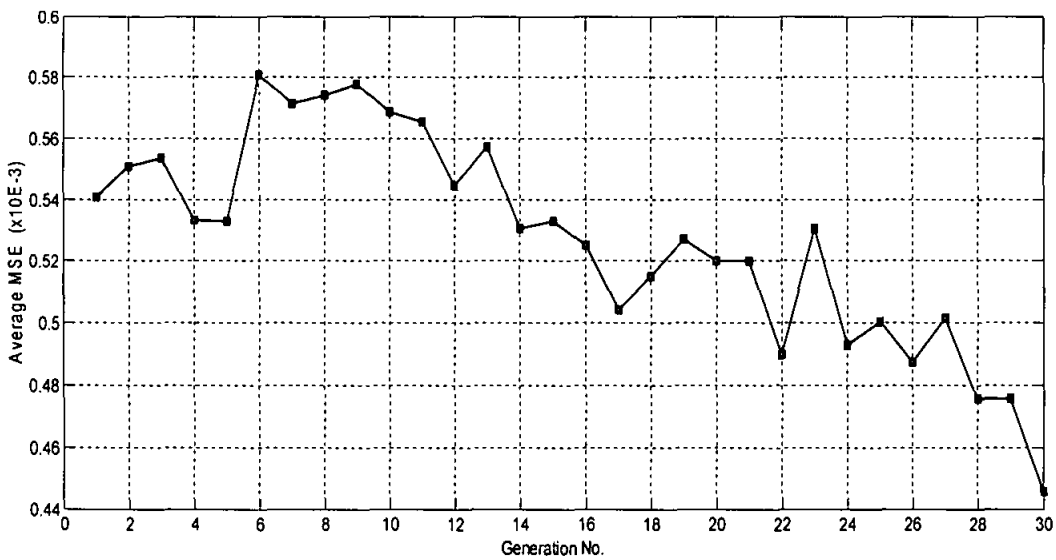


Figure 5.6 Average MSE of the entire population during GA method process

□ The best solution found is the string:

**00 101000 10 1100111100 00000011 0000001000 0001**

Which is interpreted as a one hidden layer NN with 17 neurons, tangent sigmoid activation function in hidden neurons, and tangent sigmoid function in output neurons, trained with the Steepest Descent back-propagation algorithm that used a 0.519716 value as initial weight, by learning rate of 0.025857471 with a value for

momentum term of 0.061401 after 200 epochs. This solution gave a value of 0.000019 for the MSE of flank wear.

□ The second best solution found is the string:

**11 111001 01 1100100101 00001001 0000101000 0111**

Which is interpreted as a two hidden layer with 7 neurons in the first hidden layer and 8 neurons in the second hidden layer, logistic sigmoid activation function in hidden neurons, and pure linear function in output neurons, trained with Conjugate-Gradient back-propagation algorithm that used a 0.796258 value as initial weight, by learning rate of 0.042659856, with a value for momentum term of 0.048739 after 800 epochs. This solution gave a value of 0.000019 for the MSE of flank wear.

□ The string:

**10 110001 01 1010011101 00000001 0001001111 0111**

is found as the third best solution, which is interpreted as a two hidden layer NN with 6 neurons in the first hidden layer and 7 neurons in the second hidden layer, logistic sigmoid activation function in hidden neurons, and pure linear function in output neurons, trained with Levenberg-Marquardt back-propagation algorithm that used a 0.796258 value as initial weight, by learning rate of 0.5, with a value for momentum term of 0.2 after 800 epochs. This solution gave a MSE value of 0.000020 for flank wear.

### 5.2.3 Prediction of net power and $T_4$ using GA-ANN approach

Figure 5.7 shows the best MSE values found after each generation during GA method process. In this run, the best MSE was found after 28 generations. The corresponding average MSE values of the entire population after each generation are shown in Figure 5.8.

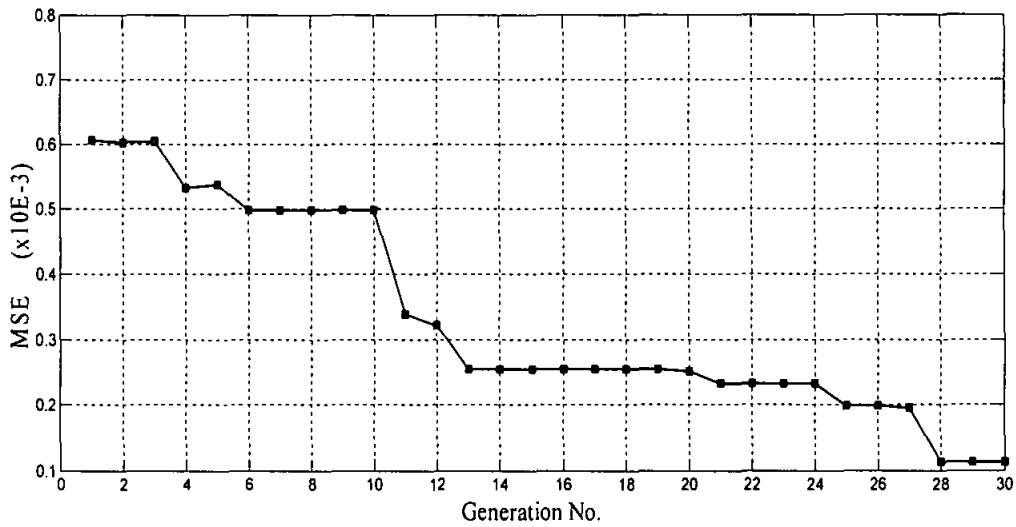


Figure 5.7 Best MSE found during GA process for the net power and  $T_4$  prediction model

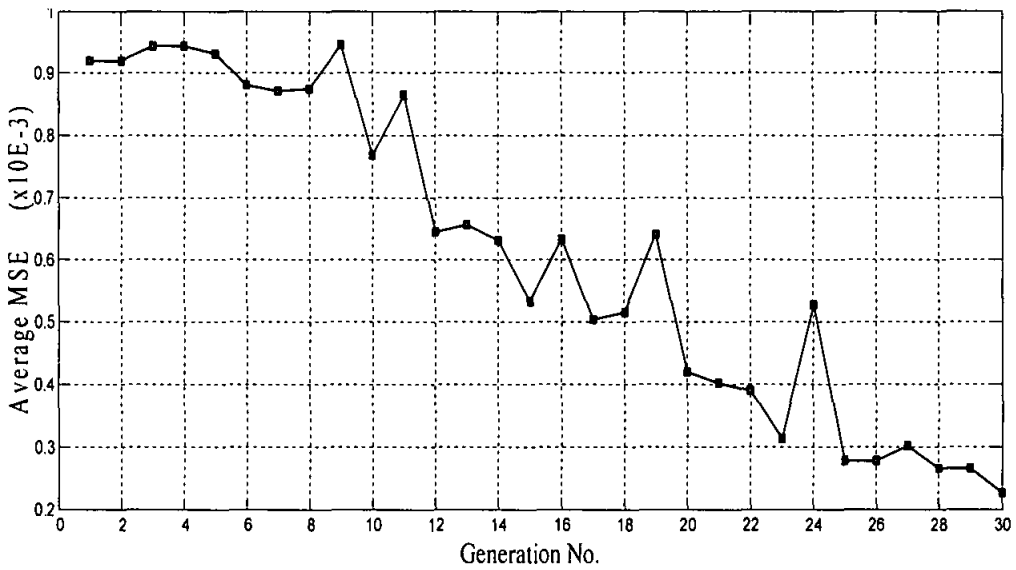


Figure 5.8 Average MSE of the entire population during GA process for net power and  $T_4$

□ The first best solution found is the string:

00 101000 10 1111011010 00000101 0000000011 0011

Which is interpreted as a two hidden layer NN with 5 neurons in the first hidden layer and 3 neurons in the second hidden layer, tangent sigmoid activation function in hidden neurons, and tangent sigmoid function output neurons, trained with the Steepest Decent back-propagation algorithm that used a 0.963612 value as initial weight, by learning rate of 0.02999195, with a value for momentum term of 0.001419

after 400 epochs. This solution gave a MSE value of 0.0001 for net power and 0.0003 for  $T_4$ .

□ The second best solution found is the string:

10 110111 00 1111111001 000001011 0001000000 0001

Which is interpreted as a two hidden layer with 7 neurons in the first hidden layer and 6 neurons in the second hidden layer, logistic sigmoid activation function in hidden neurons, and logistic sigmoid function in output neurons, trained with Levenberg-Marquardt back-propagation algorithm that used a 0.994243 value as initial weight, by learning rate of 0.046191556, with a value for momentum term of 0.06616 after 200 epochs. This solution gave a MSE value of 0.0003 for net power and 0.0001 for  $T_4$ .

□ The third best solution found is the string:

10 110001 01 111100001 000000010 0001011110 0101

Which was interoperated as a two hidden layer NN with 6 neurons in the first hidden layer and 6 neurons in the second hidden layer, logistic sigmoid activation function in hidden neurons, and pure linear function in output neurons, trained with Levenberg-Marquardt back-propagation algorithm that used a 0.483295 value as initial weight, by learning rate of 0.019257477, with a value for momentum rate of 0.094629 after 600 epochs. This solution gave a MSE value of 0.0001 for net power and 0.0001 for  $T_4$ .

#### 5.2.4 Prediction of XOR output using GA-ANN approach

Figure 5.9 shows the best MSE values found after each generation during GA method process. In this run, the best MSE was found after 13 generations. The corresponding average MSE values of the entire population after each generation are shown in Figure 5.10.

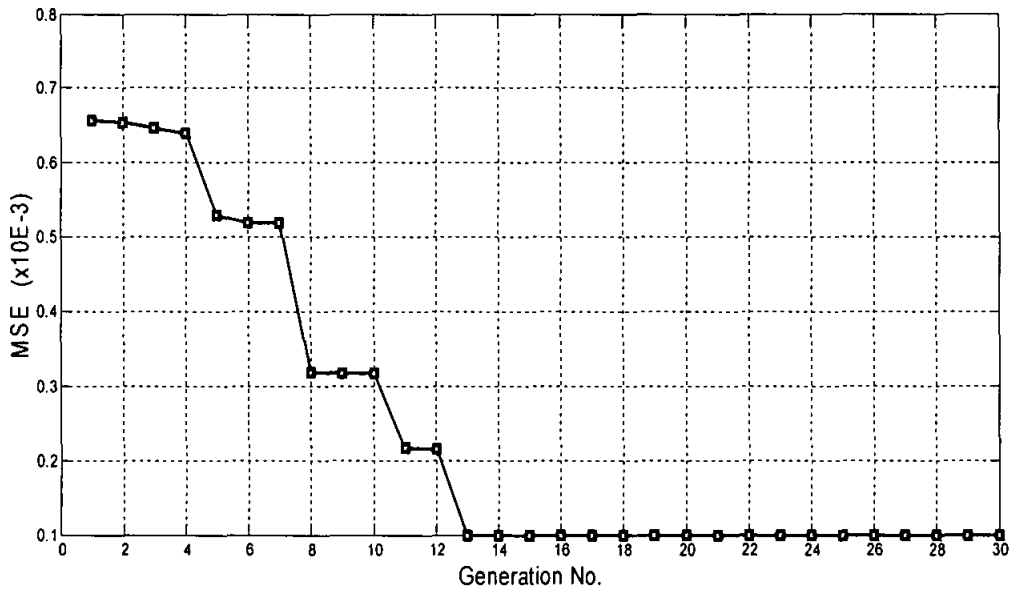


Figure 5.9 Best MSE found during GA process for the XOR output prediction model

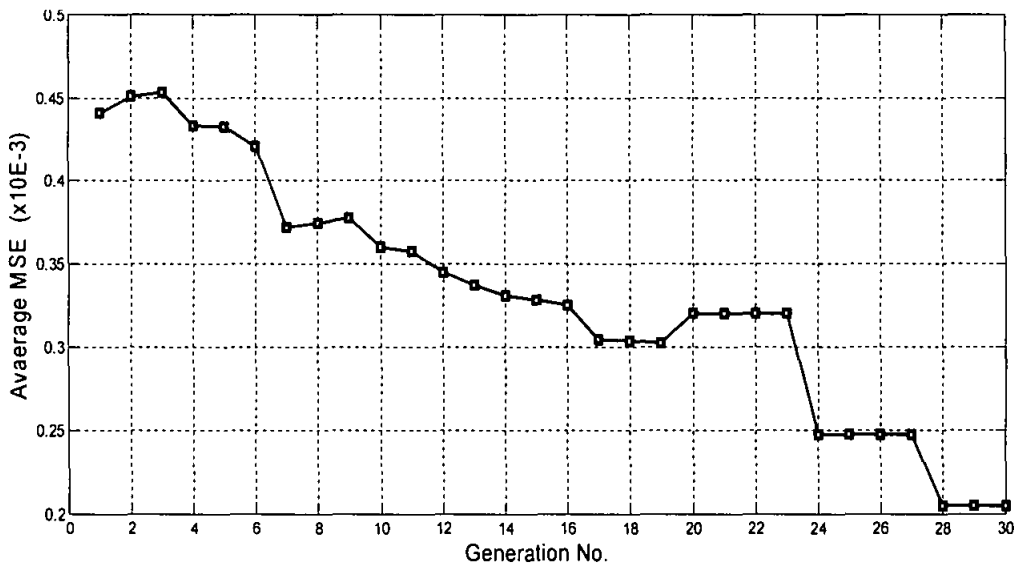


Figure 5.10 Average MSE of the entire population during GA process for the XOR output prediction model

□ The first best solution found is the string:

**00 111100 10 0000011010 00000110 0000110001 0011**

Which is interpreted as a two hidden layer NN with 8 neurons in the first hidden layer and 5 neurons in the second hidden layer, tangent sigmoid activation function in first hidden neurons, and tangent sigmoid function in output neurons, trained with the Steepest Decent back-propagation algorithm that used a 0.059031 value as initial weight, by learning rate of 0.031922630, with a value for momentum term of

0.052192 after 400 epochs. This solution gave a MSE value of 0.0002 for XOR output.

❑ The second best solution found is the string:

**10 001101 11 0000110010 00111001 0001000000 0001**

Which is interpreted as a single hidden layer with 17 neurons, tangent sigmoid activation function in hidden neurons, and pure linear function in output neurons, trained with Levenberg-Marquardt back-propagation algorithm that used a 0.053863 value as initial weight, by learning rate of 0.257846170, with a value for momentum term of 0.024855 after 200 epochs. This solution gave a MSE value of 0.0004 for XOR output.

❑ The third best solution found is the string:

**11 100111 01 00000101001 000000010 0000100010 0010**

Which was interoperated as a two hidden layer NN with 5 neurons in the first hidden layer and 3 neurons in the second hidden layer, logistic sigmoid activation function in hidden neurons, and pure linear function in output neurons, trained with Conjugate Gradient back-propagation algorithm that used a 0.020618 value as initial weight, by learning rate of 0.036382, with a value for momentum rate of 0.094629 after 300 epochs. This solution gave a MSE value of 0.0004 for XOR output.

### 5.3 Benchmarking

The proposed GA-ANN is compared with most outstanding approaches that have been proposed for optimizing multilayer feed-forward ANN parameters using GA. The comparison performance metrics are as follows:

- Optimality
- Generalization capability and stability of multilayer feed-forward ANN
- Prediction effectiveness
- Simulation time



### 5.3.1 Genetic algorithms to select architecture of a feed-forward artificial neural network (Jasmina and Ramazan 2001)

In the approach of Jasmina and Ramazan (2001), a genetic algorithm has been applied to select the optimum architecture of feed-forward ANN, mainly to evolve the number of hidden layers and connection weights. The multilayer feed-forward parameters haven't been comprehensively utilized. In this approach, the number of hidden neurons, type of activation function, learning rate, momentum term and epoch size have not been considered. Figure 5.11 shows the GA structure of Jasmina and Ramazan approach.

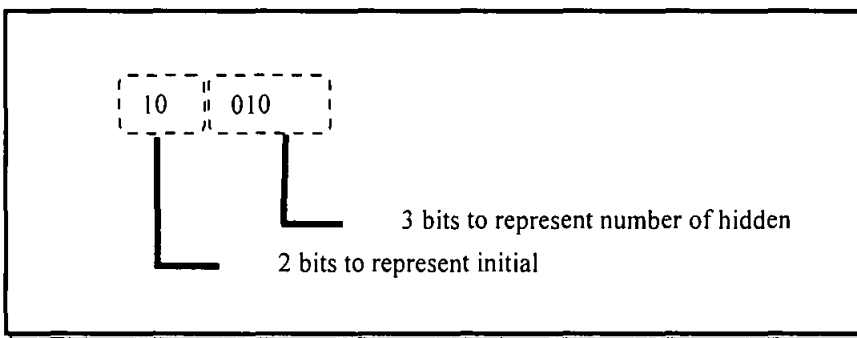


Figure 5.11 GA structure of Jasmina and Ramazan approach

### 5.3.2 Genetic algorithms to find the number of hidden layer nodes of a feed-forward artificial neural network (Torres et al. 2005)

The multilayer feed-forward ANN parameter that has been optimized in Torres et al. 2005 work is the number of hidden layer neurons. In this approach, number of hidden layers, type of activation function, initial weight, learning rate, momentum term and epoch size have not been considered either. The GA structure of Torres et al. approach is shown Figure 5.12.

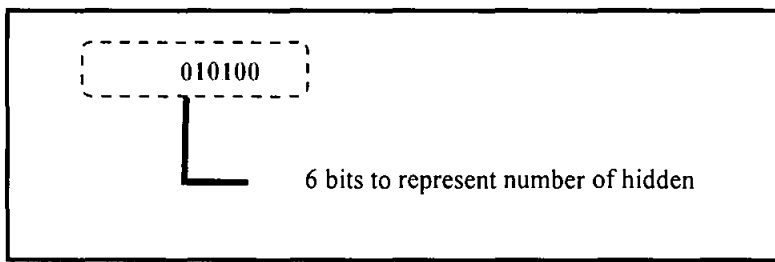


Figure 5.12 GA structure of Torres et al. approach

### 5.3.3 A combination of genetic algorithm and artificial neural network (Makoto et al. 2006)

The Makoto et al. 2006 work is focused on the optimization of weight connection of feed-forward ANN using genetic algorithm. In this work, the number of hidden layers, number of neurons in hidden layer, type of activation function, learning rate, momentum term and epoch size have not been considered as well. Figure 5.13 shows the GA structure of Makoto et al. approach.

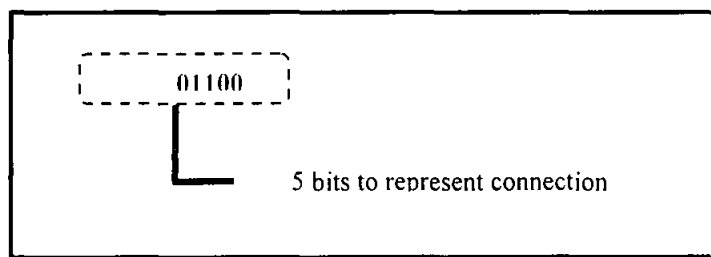


Figure 5.13 GA structure of Makoto et al. approach

### 5.3.4 Feed-forward neural networks designed and parameterized by genetic algorithms (Ferentinos 2005)

The Ferentinos 2005 work is considered as one of the most recent approaches that have been applied to the selection of the following multilayer feed-forward ANN parameters using genetic algorithm:

- Minimization algorithm used by the back-propagation training algorithm.
- The architecture of the ANN.
- The types of activation functions of the hidden neurons and the output neurons.

There are some multilayer feed-forward parameters that have not been considered in this approach such as initial weight, learning rate, momentum term and epoch size. The GA structure of Ferentinos approach is shown in Figure 5.14.

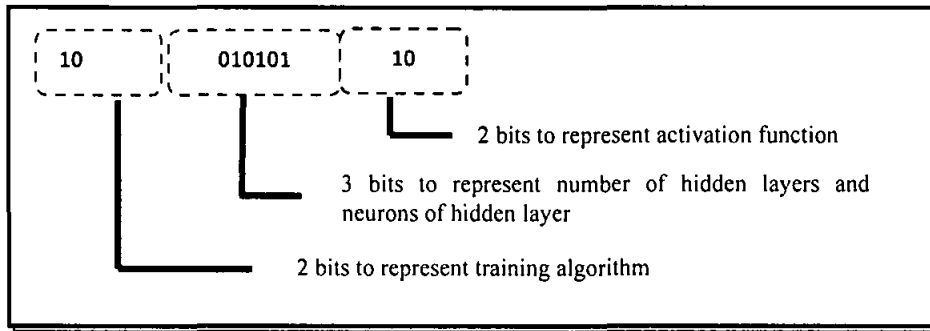


Figure 5.14 GA structure of Ferentinos approach

### 5.3.5 Design of neural networks using genetic algorithm (Manojit and Deoki 2006)

A methodology based on genetic algorithm (GA) has been developed by Manojit and Deoki (2006) to overcome the problem of designing ANN by a trial and error procedure. This involves optimally determining the number of hidden layers, number of neurons in each hidden layer, and activation functions. In their work, training algorithm, learning rate, momentum term, initial weight and epoch size have not been considered. Figure 5.15 shows the GA structure of Manojit and Deoki approach.

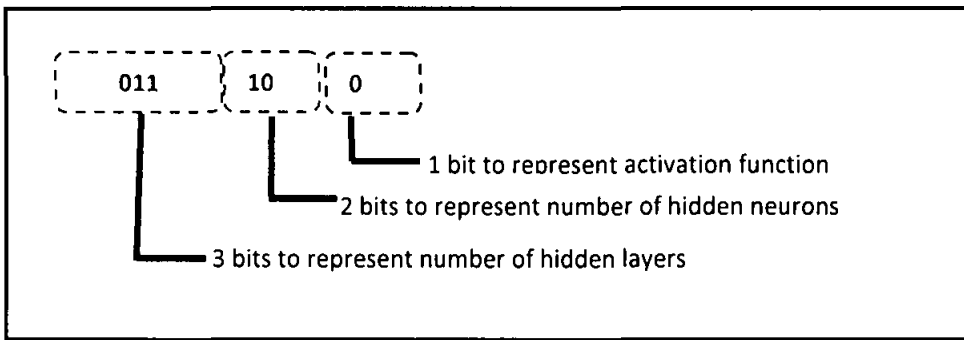


Figure 5.15 GA structure of Manojit and Deoki approach

### 5.4 Comparative study

The dataset of the four applications have been used to evaluate the prediction performance.

### 5.4.1 Optimality

In order to assess the optimality degree of the proposed approach with the other existing approaches of multilayer feed-forward ANN using GA, Table 5.2 shows the parameters of multilayer feed-forward ANN that have been covered in the proposed approach and five other approaches. Further, Table 5.2 shows that the proposed approach has provided a comprehensive optimization for architecture and training parameters of multilayer feed-forward ANN which means less human dependent, and higher precision and effectiveness have been achieved.

Table 5.2 Required architecture and training parameters of multilayer feed-forward ANN

Approach	Hidden layers	Hidden neurons	Training algorithm	Activation function	Initial weight	Learning rate	Momentum term	Epoch size
Jasmina and Ramazan (2001)	√	×	×	×	√	×	×	×
Torres et al. (2005)	√	√	×	×	×	×	×	×
Makoto et al. (2006)	×	×	×	×	√	×	×	×
Ferentinos (2005)	√	√	√	√	×	×	×	×
Manojit and Deoki (2006)	√	√	×	√	×	×	×	×
Proposed GA-ANN (2010)	√	√	√	√	√	√	√	√

Necessary GA specifications are the subject of comparison in Table 5.3; GA features that are applied for comprehensive optimum method are the subject of comparison in Table 5.4.

Table 5.3 GA specification of proposed and other approaches

Approach	Number of individuals	Encoding type	Chromosome length	Selection method	Mutation method	Crossover method
Jasmina and Ramazan (2001)	2	Indirect	7	Tournament	Not mentioned	One point
Torres et al. (2005)	2	Indirect	6	Roulette	Not mentioned	One point
Makoto et al. (2006)	1	Indirect	5	Not mentioned	Not mentioned	Not mentioned
Ferentinos (2005)	4	Indirect	10	Proportional selection	Occasional random alteration	One-point
Manojit and Deoki (2006)	3	Indirect	6	Tournament	Occasional random alteration	One-point
Proposed GA-ANN (2010)	8	Indirect	42	Proportional selection	Occasional random alteration	One-point

Table 5.4 GA features of proposed and other approaches

Approach	Population size	Number of generation	Pc	Pm	Fitness function
Jasmina and Ramazan (2001)	50	30	0.6	0.0033	MSE
Torres et al. (2005)	Not mentioned	Not mentioned	Not mentioned	Not mentioned	Not mentioned
Makoto et al. (2006)	Not mentioned	40	0.7	0.01	R <sup>2</sup>
Ferentinos (2005)	20	30	0.9	0.05	MSE
Manojit and Deoki (2006)	50	100	0.4	0.08	SSE
Proposed GA-ANN (2010)	20	10-30	0.05, 0.01, and 0.1	0.5-0.95	MSE

#### 5.4.2 Generalization capability and stability of multilayer feed-forward ANN

Data processing techniques have the greatest effect on the ANN generalization and performance. In the proposed approach, a comprehensive data pre-processing techniques are provided to improve the generalization capability and performance of multilayer feed-forward ANN. These techniques include missing value removal, data outlier detection and removal, data normalization, and data partitioning. The data pre-processing techniques are listed in Table 5.5, including the technique employed in our approach.

Table 5.5 Data pre-processing techniques

Approach	Missing value removal	Data outlier removal	Data normalization	Data partitioning
Jasmina and Ramazan (2001)	×	×	√	×
Torres et al. (2005)	×	×	√	×
Makoto et al. (2006)	×	×	√	×
Ferentinos (2005)	×	×	√	√
Manojit and Deoki (2006)	×	×	√	√
Proposed GA-ANN (2010)	√	√	√	√

The number of hidden layers and neurons per layer influence the generalization capability and reliability of the ANN. The more hidden layers the ANNs possess, the more precise the ANNs are. However, the ANN operation calculation rate is slower. Therefore, the number of hidden layers is reasonably designed in accordance with the computation speed and the performance precision. The number of neurons in hidden layers is related to the precision. Theoretically, the more number of neurons in the hidden layers, the lesser would be the errors in the system, which will result in an interminable computation.

The epoch size affects multilayer feed-forward ANN generalization capability. The use of small epoch size may produce ANN that is incapable of representing the data, whereas, a large epoch may enable ANN to memorize the data, but will demonstrate bad generalization to untrained data.

In addition, a smaller value of learning rate may slow down the training speed. Though, a high value of learning rate increases the training speed but the network becomes unstable. ANN features that are necessary for generalization capability and stability are the subject of comparison in Table 5.6.

Table 5.6 Necessary features for generalization capability and stability

Approach	ANN features				
	Epoch size	Learning rate	Momentum term	Hidden layers	Hidden neurons
Jasmina and Ramazan (2001)	Trial-and-error	Not mentioned	Not mentioned	Single hidden layer	Adopted using GA/1-8
Torres et al. (2005)	Trial-and-error	Not mentioned	Not mentioned	Single hidden layer	Adopted using GA
Makoto et al. (2006)	1000	0.1	0.8	Adopted using GA	Adopted using GA
Ferentinos (2005)	1000	On-line adjustable learning rate	Not mentioned	Adopted using GA/single & two	Adopted using GA/30 & 8
Manojit and Deoki (2006)	Trial-and-error	0.05	0.5	Adopted using GA/single, two & three	Adopted using GA/10, 10 & 10
Proposed GA-ANN (2010)	Adopted using GA	Adopted using GA	Adopted using GA	Adopted using GA/single & two	Adopted using GA/30 & 8

### 5.4.3 Prediction effectiveness

The performance of the proposed GA-ANN approach is compared to the existing approaches in terms of prediction effectiveness; the results are summarized in Table 5.7.

Table 5.7 Results of the proposed GA-ANN approach and other approaches in terms of prediction effectiveness

Output	Jasmina and Ramazan (2001)		Torres et al. (2005)		Makoto et al. (2006)		Ferentinos (2005)		Manojit and Deoki (2006)		Proposed GA-ANN (2010)	
	MSE	RMSE	MSE	RMSE	MSE	RMSE	MSE	RMSE	MSE	RMSE	MSE	RMSE
iC <sub>5</sub>	0.042	0.205	0.113	0.336	0.301	0.549	0.032	0.179	0.221	0.470	0.002	0.045
nC <sub>5</sub>	0.077	0.277	0.215	0.464	0.419	0.647	0.036	0.190	0.197	0.444	0.0002	0.014
Flank wear	0.013	0.114	0.032	0.179	0.012	0.110	0.007	0.084	0.058	0.241	0.00002	0.004
Net power	0.483	0.695	0.313	0.559	0.442	0.665	0.037	0.192	0.33	0.574	0.0001	0.010
T4	0.325	0.570	0.295	0.543	0.256	0.506	0.131	0.362	0.114	0.338	0.0001	0.010
XOR output	0.039	0.197	0.006	0.077	0.016	0.126	0.001	0.032	0.077	0.277	0.0002	0.014

In this work, mean square error (MSE), and root mean square error (RMSE) have been utilized to evaluate the prediction efficiency of the proposed approach and other approaches. In general, if the values of MSE and RMSE are close to zero, that means perfect prediction efficiency has been achieved.

From Table 5.7, it is obvious that the proposed GA-ANN approach is capable of judging a good solution that will give effective prediction performance of the multilayer feed-forward ANN.

A comparison of the results produced by the proposed GA-ANN approach and by the other approaches in terms of effectiveness clearly indicates that the proposed GA-ANN approach has achieved a higher degree of success. Figure 5.16 and Figure 5.17 show the performance comparison of the proposed approach and other approaches in terms of MSE and RMSE.

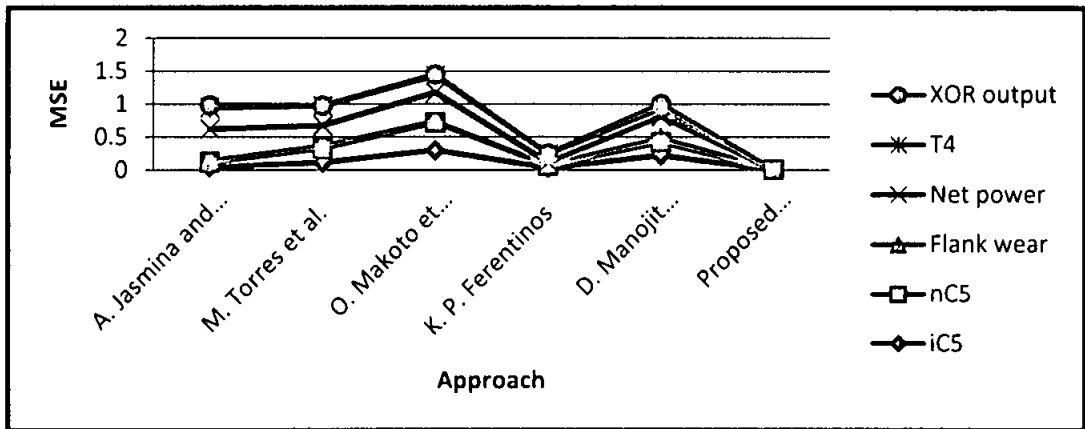


Figure 5.16 Comparison of proposed approach with other approaches in terms of prediction effectiveness (MSE)

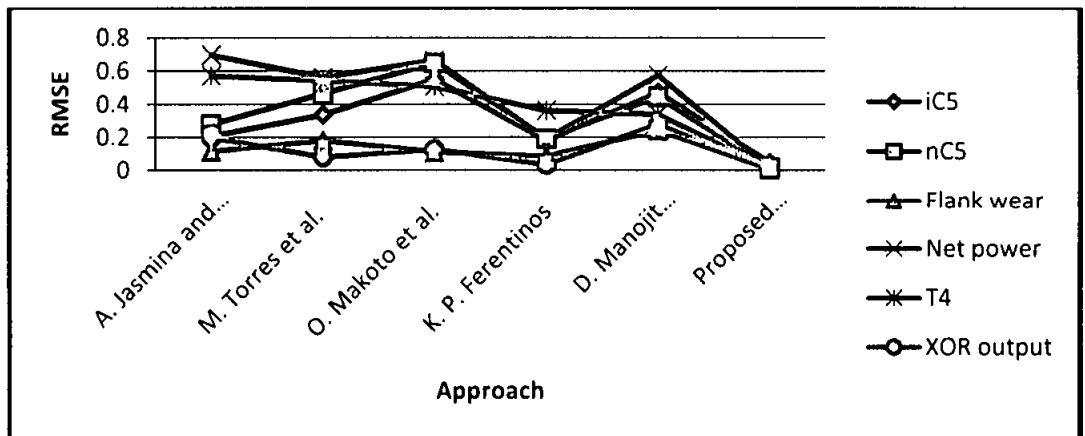


Figure 5.17 Comparison of proposed approach with other approaches in terms of prediction effectiveness (RMSE)

Besides that, the coefficient correlation and determination coefficient of the proposed GA-ANN approach and the other five approaches in order to measure the performance of prediction are summarized in Table 5.8.



Table 5.8 Results of the proposed GA-ANN approach and other approaches in terms of prediction performance

Approach	Jasmina and Ramazan (2001)		Torres et al. (2005)		Makoto et al. (2006)		Ferentinos (2005)		Manojit and Deoki (2006)		Proposed GA-ANN (2010)	
	R	R <sup>2</sup>	R	R <sup>2</sup>	R	R <sup>2</sup>	R	R <sup>2</sup>	R	R <sup>2</sup>	R	R <sup>2</sup>
iC <sub>s</sub>	0.931	0.867	0.943	0.889	0.933	0.870	0.965	0.931	0.956	0.914	0.998	0.996
nC <sub>s</sub>	0.922	0.850	0.947	0.897	0.934	0.872	0.954	0.910	0.963	0.927	0.998	0.996
Flank wear	0.96	0.922	0.955	0.912	0.961	0.924	0.97	0.941	0.963	0.927	0.995	0.990
Net power	0.946	0.895	0.921	0.848	0.903	0.815	0.952	0.906	0.942	0.887	0.994	0.988
T <sub>4</sub>	0.938	0.880	0.947	0.897	0.91	0.828	0.955	0.912	0.955	0.912	0.989	0.978
XOR output	0.976	0.953	0.966	0.933	0.95	0.903	0.973	0.947	0.971	0.943	0.995	0.990

In practical, correlation coefficient (R) and determination coefficient (R<sup>2</sup>) are utilized to measure the performance of prediction model; if the values of R and R<sup>2</sup> are close to one that means high prediction performance has been achieved.

From Table 5.8, it is obvious that the proposed GA-ANN approach is capable of judging a good solution that will give a highly prediction performance of the multilayer feed-forward ANN.

A comparison of the results produced by the proposed GA-ANN approach and by the other approaches in terms of effectiveness clearly indicates that the proposed GA-ANN approach has achieved a higher degree of success. Figure 5.18 and Figure 5.19 show the comparison of performance of the proposed approach and the other five approaches.

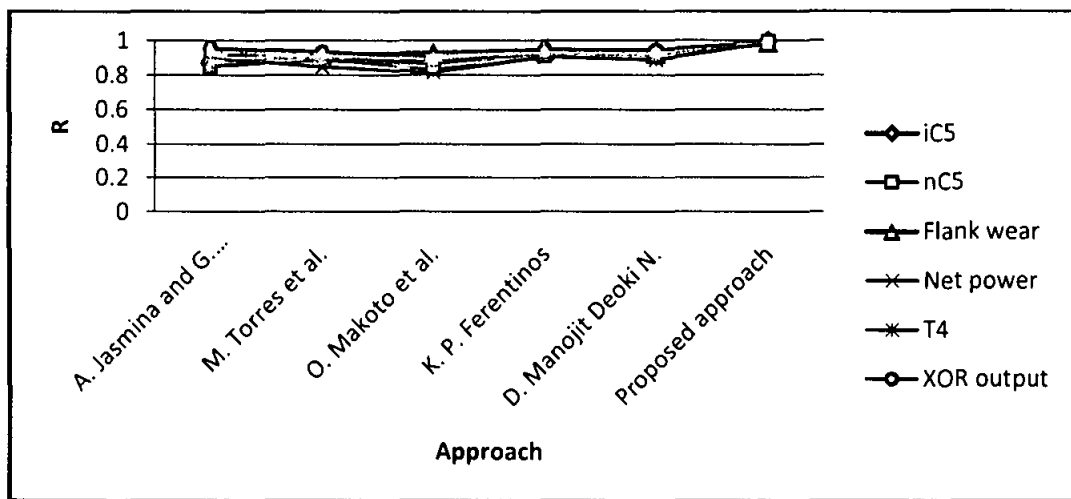


Figure 5.18 Comparison of proposed approach with other approaches in terms of prediction performance (R)

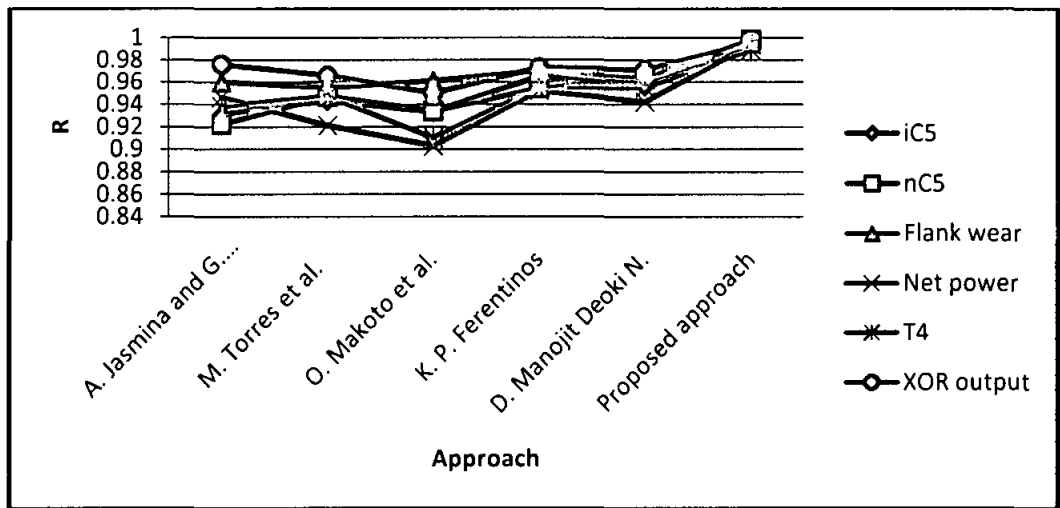


Figure 5.19 Comparison of proposed approach with other approaches in terms of prediction performance ( $R^2$ )

#### 5.4.4 Simulation time

MATLAB 2009a with 6.0.2 Neural Networks Toolbox has been used to simulate the whole approaches. The simulation time of the proposed GA-ANN approach and the other approaches are summarized in Table 5.9.

Table 5.9 Simulation time of proposed GA-ANN approach and other approaches

Approach	Jasmina and Ramazan (2001)	Torres et al. (2005)	Makoto et al. (2006)	Ferentinos (2005)	Manojit and Deoki Saraf (2006)	Proposed GA-ANN (2010)
Feature						
Optimized parameters	No of hidden layers and weights	No of hidden neurons	Connection weights	No of hidden layers , neurons, activation function and training algorithm	No of hidden layers, neurons, and activation function	No of hidden layers , neurons, activation function, training algorithm, initial weight, learning rate, momentum term and epoch size
No. of optimized parameters	2	1	1	3	3	8
GA structure	5	6	5	10	6	42
Application	Simulation Time					
Debutanizer	16:58	17:36	10:11	21:33	19:20	19:23
Drilling process	14:39	15:24	12:46	15:57	13:17	15:44
Gas turbine	11:42	13:51	13:49	16:21	14:50	14:20
XOR	2:26	3:31	3:16	3:00	2:45	3:10

Although the proposed approach has comprehensively been covered every aspect of multilayer feed-forward neural networks, which means the use of long space solution, this approach has given acceptable simulation time compared to other approaches that use short space solution and utilize a few multilayer feed-forward neural network parameters.

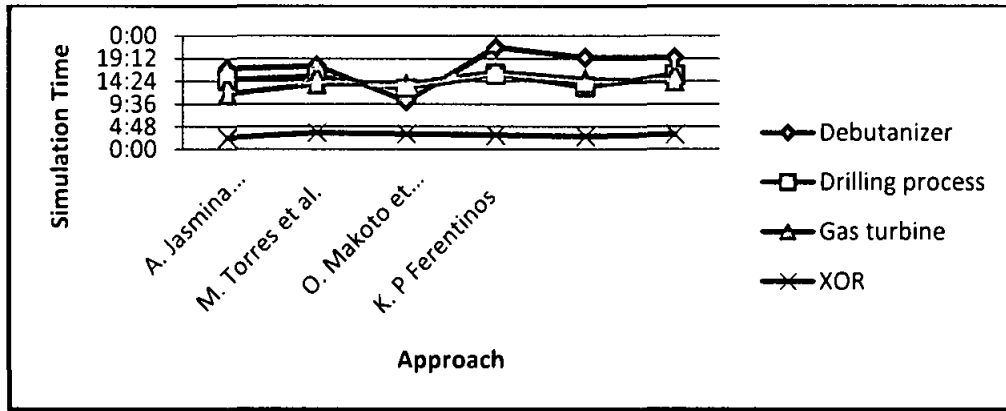


Figure 5.20 Comparison of proposed approach with other approaches in terms of simulation time

A comparison of the results achieved by the proposed GA-ANN approach and by the other approaches in terms of simulation time clearly indicates that the proposed GA-ANN approach has achieved acceptable degree of success. Figure 5.20 compares the simulation time of the proposed approach and other approaches.

### 5.5 Summary

The results of the proposed GA-ANN approach that has been used to predict the outputs of four different datasets described in chapter 4 have been presented in this chapter, including a discussion of the developed GA-ANN approach. A direct comparison between five outstanding approaches and the proposed GA-ANN has been conducted; the results have given a clear indication of the degree of success of the proposed GA-ANN approach in terms of generalization capability and training stability, optimality, performance prediction effectiveness, prediction accuracy and simulation time. In this work, MATLAB 2009a software with Neural Networks Toolbox was employed to develop the GA-ANN approach.

## Chapter 6

### CONCLUSION AND RECOMMENDATION

This chapter summarizes the findings from this thesis, contribution and provides some recommendations for further work.

#### **6.1 Conclusion**

The main goal of this work is to propose a methodology for determining the optimum architecture and training parameters of multilayer feed-forward artificial neural network (ANN) and to contribute to existing work on multilayer feed-forward ANN designing and training performance, based on a genetic algorithm (GA).

Despite the popularity of multilayer feed-forward ANN, there are still some challenges concerning the development of multilayer feed-forward ANN model that is able to perform effectively and accurately. In other words, the desired performance is depended on the selection of appropriate architecture and training parameters. These parameters include the number of hidden layers, number of hidden neurons, type of training algorithm, type of activation function, initial weight, learning rate, momentum term and epoch size.

There are existing attempts based on GA to optimize multilayer feed-forward ANN parameters, but, these attempts still focus only on parts of the parameters. For example (Blanco et al. 2001; Taeksoo and Han 2000; Sedki et al. 2009) applied GA method to evolve the connection weights of multilayer feed-forward ANN. Another GA method was used by (Hyun-jung and Shin 2007) to optimize both connection weights and number of hidden neurons, while (Ferentinos 2005) considered a binary-encoded GA to determine the optimum multilayer feed-forward ANN structure, training algorithm, and activation function.

In this work, the GA-based approach has been tested on four different datasets. One of them is from Universiti Teknologi PETRONAS GDC plant (TAURUS 60 gas turbine single-shaft generator set) collected during Jan. to Feb. 2008 period. Another dataset was collected from PETRONAS Penapisan (Melaka) Sdn Bhd from Jan. to Feb. 2007. The third dataset is a published experimental dataset of flank wear for drilling process (Panda et al. 2008). Lastly, standard XOR problem dataset was used to ensure the applicability of the proposed GA-ANN approach.

The data preprocessing is one of the most important issues in the success of any multilayer feed-forward ANN design. In this research a comprehensive data preprocessing techniques have been provided. These techniques include missing value removal, data outlier detection and removal, data normalization, and data partitioning.

Then the GA method was applied to obtain the optimal multilayer feed-forward ANN architecture and training parameters. There are three major reasons for the implementation of a GA instead of a conventional optimization technique in this approach (Karr 1995):

- The GA works and focuses directly on a coded form of the problem's parameter set and not on the parameters themselves.
- The GA process is started from a group of points of the solution space (initial population) and not from a single point, therefore, reducing the chance of converging to local optima.
- The sampling process is conducted using the genetic operators (i.e., selection, crossover, and mutation) which are stochastic rules and not deterministic rules.

Based on the approaches extracted from some applications of GA encoding in the works of previous researchers, indirect encoding scheme has been applied, which is the way of encoding several possible chromosomes of multilayer feed-forward ANN into specific genotypes, in the form of a simple binary string comprised of a series of 0 and 1 in order to encode the decision variables of a specific problem. A chromosome, in this research, consists of the training algorithm, number of hidden layers and neurons in each hidden layer, activation functions of the hidden and output neurons, initial weight, learning rate, momentum term, and epoch size. A genotype is

a sequence of bits (0 or 1) with a specific constant length. Each genotype corresponds to a unique chromosome.

Several algorithms for improving multilayer feed-forward ANN performance have been developed. These algorithms have been implemented using MATLAB 2009a and Neural networks Toolbox.

The results of the proposed approach have been compared with the most outstanding approaches that have been proposed for optimizing multilayer feed-forward ANN parameters using GA. The comparison performance metrics are as follows:

- Optimality
- Generalization capability and stability of multilayer feed-forward ANN
- Prediction effectiveness
- Simulation time

As far as the application of the new method for ANN combination based on GA is concerned, the following conclusions can be extracted:

- GA encoding and optimization can be successfully adopted to a combinatorial problem such as the decision of what is the best multilayer feed-forward ANN structure, training algorithm, activation functions, initial weight, learning rate, momentum term, and epoch size for a specific model.
- Compared to the existing approaches, the proposed approach give better results in terms of prediction effectiveness and accuracy. For example, the proposed approach gave results of MSE value in range of (0.00004 to 0.0002) for all applications, whereas the range of best MSE value by the other approaches is (0.001 to 0.036). In terms of prediction accuracy, the proposed approach gave results of R value in the range of (0.989 to 0.998) for all applications, whereas the range of best R value given by the other approaches is (0.815 to 0.976).
- In most cases, the proposed approach should be more desirable because it is an automated technique, compared to the other approaches.

- For some cases, the results of the proposed approach could give useful information and insights for further, more successful applications of the other approaches.

Nevertheless, the proposed approach can be extended to different minimization algorithms of the back-propagation, or network architectures different from that presented here, more potential activation function types, different values of the initial weight and epoch size.

## **6.2 Contributions**

In contrary to the previous works that have been partially designated for some parameters of multilayer feed-forward ANN, a new GA-ANN approach that covers most of those required parameters have been presented. The research considerably contributes to demonstrating the strength of genetic algorithm (GA) and the predetermined objectives have already been achieved. The proposed research method is considered novel in the sense that it proves that GA-based method can be comprehensively utilized to determine multilayer feed-forward ANN architecture and training parameters such as number of hidden layers, number of neurons in hidden layer, training algorithm, activation function, initial weight, learning rate, momentum term and epoch size.

Specifically, the proposed approach has contributed to the followings:

- Auto design of multilayer feed-forward ANN, because the proposed approach optimizes most of architecture and training parameters.
- Effective and precise performance, besides, less human dependence than existing approaches, proposed approach achieved low sum square error and high value of determination coefficient compared to existing approaches.
- Generalization capability and training stability of multilayer feed-forward ANN, this work focused on the multilayer feed-forward parameters that have a great affect on these performance metrics.

- Demonstrates the hybridization capability of GA with multilayer feed-forward ANN.

### **6.3 Recommendations for future work**

With respect to this thesis, the following three areas are recommended for future work.

#### **6.3.1 Evolving the network structure**

Based on theory, the number of hidden layers and the number of neurons in hidden layer have a significant effect on network performance (Torres et al. 2005). Therefore, the proposed GA method can be extended to determine extra number of hidden layers with even extra number of neurons in the hidden layer of multilayer feed-forward ANN.

#### **6.3.2 Selection of minimization and activation function**

Although evolving adaptive characteristics using GA-based method has shown multilayer feed-forward ANN robustness, we believe, and from the results obtained from this research, it can be further extended to evolve more minimization functions of back-propagation algorithm and activation function types. Because the use of different activation function may achieve totally different multilayer feed-forward ANN performance, and different minimization function of back propagation algorithm may give quite different multilayer feed-forward ANN performance (Ferentinos 2005).

#### **6.3.3 Selection of GA parameters**

Although the GA method has been successful at improving multilayer feed-forward ANN architecture and training parameters, there is still a big challenge to select the appropriate GA design parameters in order to reduce the dependency on human ; for example the selection of population size, number of generations , and probability of



crossover and mutation. These parameters have a significant effect on the GA performance. Appropriate selection method is required towards fully auto design of GA (Ferentinos 2005).

## REFERENCES

- A. Ali, S. Morteza and A. Mona (2010). "An integrated artificial neural network algorithm for performance assessment and optimization of decision making units." Expert systems with applications.
- A. Blanco, M. Delgado and M. C. Pegalajar (2001). "A real-coded genetic algorithm for training recurrent neural networks." Neural networks 14(1): 93-105.
- A. Chen, M.T. Leung and D. Hazem (2003). "Application of previous termneural networksnext term to an emerging financial market: Forecasting and trading the Taiwan stock index." Computers and operations research 30: 901-923.
- A. Ganesh and B. Abdesselam (2003). "A generalized feedforward neural network architecture for classification and regression." Neural networks 16(5-6): 561-568.
- A. Jasmina and G. Ramazan (2001). "Using genetic algorithms to select architecture ofa feedforward articial neural network." Physica A: statistical mechanics and its applications 289(3-4).
- A. Necat and K. Rasit (2004). "Neural network approach to prediction of bending strength and hardening behaviour of particulate reinforced (Al-Si-Mg)-aluminium matrix composites." Materials & design 25(7): 595-602.
- A. Sedki, D. Ouazar and E. El Mazoudi (2009). "Evolving neural network using real coded genetic algorithm for daily rainfall-runoff forecasting." Expert systems with applications 36(3): 4523-4527.
- L.P. Maguire A.B. Johnston, T.M. McGinnity (2009). "Downstream performance prediction for a manufacturing system using neural networks and six-sigma improvement techniques." Robotics and computer-integrated manufacturing 25(3): 513-521.
- A.B. Koc, P.H. Heinemann and G.R. Ziegler (2007). "Optimization of whole milk powder processing variables with neural networks and genetic algorithms." Food and bioproducts processing 85(4): 336-343.
- A.L. Ahmad, I.A. Azid, A.R. Yusof and K.N. Seetharamu (2004). "Emission control in palm oil mills using artificial neural network and genetic algorithm." Computers & chemical engineering 28(12): 2709-2715.

- A.N. Refenes, A. Zapranis and G. Francis (1994). "Stock performance modeling using neural networks: A comparative study with regression models." Neural networks 7(2): 375-388.
- I. Adiel (2001). training and optimization of product units neural networks. Pretoria, the university of Pretoria, master thesis.
- Nii O. Attoh-Okine (1999). "Analysis of learning rate and momentum term in backpropagation neural network algorithm trained to predict pavement performance." Advances in engineering software 30(4): 291-302.
- B. Lin, B. Recke, J. Knudsen and S. B. Jrgensen (2007). "A systematic approach for soft sensor development." Computers and chemical engineering 31(5): 419-425.
- B. Walczak and D. L. Massart (2001a). "Dealing with missing data, Part I." Chemometrics and intelligent laboratory systems 58(1): 15-27.
- A. Barto (1992). Reinforcement learning and adaptive critic methods. New York, Van Nostrand Reinhold.
- R. Battiti (1992). "First and second-order methods for learning: Between steepest descent and Newton's method." Neural computing 4(141-166).
- H. Demuth and M. Beale (2005). Neural network toolbox user's guide, The mathworks Inc.
- BG. C, etiner, AS. Has\_ilog˘lu, M. Bayramog˘lu and R. Ko˘ker (1998). "Comparative study of texture classification using various wavelet schemes." Second international symposium on IMS, Turkey: 709-716.
- J. Bilski (2005). "the UD RLS algorithm for training feedforward neural networks." Int. J. Appl. Math. Comput. Sci. 15(1): 115-123.
- C. Ali and G. Ali (2008). "Steam turbine model." Simulation modelling practice and theory 16(9): 1145-1162.
- C. Arzum E. and K. Yalcin (2007). "Evaluating and forecasting banking crises through neural network models: an application for Turkish banking sector." Expert systems with applications 33(4): 809-815.
- C. E. Romero and J. N. Carter (2001). "Using genetic algorithms for reservoir characterization,." Journal of petroleum science and engineering 31(2-4): 113-123.
- C. Fernández, E. Soria, J.D. Martín and A.J. Serrano (2006). "Neural networks for animal science applications: two case studies." Expert aystems with Applications 31(2): 444-450.

- C. Marco and R. Hefin (2009). "Evolutionary artificial neural network design and training for wood veneer classification." Engineering applications of artificial intelligence 22: 732-741.
- C. Sungzoon and R. James (1993). "Multiple disorder diagnosis with adaptive competitive neural networks." Artificial intelligence in medicine 5(6): 469-487.
- S. Caetano (2006). Nature reviews immunology 6: 476-483.
- D. Mandic and J. Chambers (2001). Recurrent neural networks for prediction. learning algorithms, architectures and stability, John Wiley and Sons.
- Chen MC. and Yang T. (2002). "Design of manufacturing systems by a hybrid approach with neural network metamodelling and stochastic local search." Int J Prod Res 40: 71-92.
- D. Baily and D.M. Thompson (1990). "Developing neural network applications." AI expert: 33-41.
- D. E. Rumelhart, G. E. Hinton and R. J. Williams (1986). "Learning internal representation by error propagation." Parallel distributed processing 1, MIT Press: 318-362.
- D. F. Cook, C. T. Ragsdale and R. L. Major (2000). "Combining a neural network with a genetic algorithm for process parameter optimization." Engineering applications of artificial intelligence 13(4): 391-396.
- D. Shanthi, G. Sahoo and N. Saravanan (2009). "Evolving connection weights of artificial neural networks using genetic algorithm with application to the prediction of stroke disease." International journal of soft computing 4(2): 95-102.
- D. Sheppard, D. McPhee, C. Darke, B. Shrethra, R. Moore, A. Jurewitz and A. Gray (1999). "Predicting cytomegalovirus disease after renal transplantation: an artificial neural network approach." International journal of medical informatics 54(1): 55-76.
- D.E. Rumelhart, G.E. Hinton and R.J. Williams (1986). "Learning internal representations by error propagation." Parallel distributed process 1: 318-362.
- G.E. Hinton D.E. Rumelhart, R.J. Williams (1986). "Learning internal representations by error propagation." Parallel distributed process 1: 318-362.
- E. Almeida, M.A.Rodrigues and D. Odloak (2000). "Robust predictive control of a gasoline debutanizer column." Brazilian journal of chemical engineering.
- E. B. Baum and D. Haussler (1988). "Neural computation I." 151-160.
- J. Elman (1990). "Finding structure in time." Cognitive science 14: 179-211.

- F. Fred, G. James and L. Juliet (2000), Predicating temperature profiles in producing oil wells using artificial neural networks.
- F. Pezzellaa, G. Morgantia and G. Ciaschetti (2008). "A genetic algorithm for the Flexible Job-shop Scheduling Problem." Computers & operations research 35(10): 3202-3212.
- R. Howard F. (2000). Handbook of commercial catalysts: Heterogeneous catalysts, CRC Press.
- K.P. Ferentinos (2005). "Biological engineering applications of feedforward neural networks designed and parameterized by genetic algorithms." Neural networks 18(7): 934-950.
- B. Freisleben (1992). Stock market prediction with backpropagation networks. industrial engineering applications of artificial intelligence and expert system, 5th International conference. Paderborn, Germany: 451-460.
- L. Fu (1995). Neural networks in computer intelligence. New York, McGraw-Hill.
- G Cheginia, J. Khazaeia, B. Ghobadianb and A. Goudarzi (2008). "Prediction of process and product parameters in an orange juice spray dryer using artificial neural networks." Journal of Food Engineering 84(4): 534-543.
- G.-P. Nicolás, O.-B. Domingo and H.-M. César (2006). "An alternative approach for neural network evolution with a genetic algorithm: crossover by combinatorial optimization." Neural Netw. 19(4): 514-528.
- G. C. Onwubolu and M. Mutingi (2001). "A genetic algorithm approach to cellular manufacturing systems." Computers & industrial engineering 39(1-2): 125-144.
- G. Chrysovalantis, P. Fotios and D. Michael (2007). "Probabilistic neural networks for the identification of qualified audit opinions." Expert systems with applications 32(1): 114-124.
- G. James and E. Glenn (2001). Petroleum refining: technology and economics, Handwork, CRC Press.
- G. Li, H. Alnuweiri and W. Wu (1993). "Acceleration of backpropagation through initial weight pre-training with Delta rule." Proceedings of an international joint conference on neural networks: 580-585.
- G. P. Zhang, B. E. Patuwo and M. Y. Hu (2001). "A simulation study of artificial neural networks for nonlinear time-series forecasting." Computers & operations research 28(4): 381-396.
- G. Zhang, B.E. Patuwo and M.Y. Hu (1998). "Forecasting with artificial neural networks: the state of the art." International journal of forecasting 14: 35-62.

- G.F. Miller, P.M. Todd, S.U. Hegde and V.A. Fairfax (1989). "Designing neural networks using genetic algorithms." In: J.D. Schaffer, Editor, Proceedings of the third international conference on genetic algorithms, Morgan Kaufmann, : 379-384.
- G.P. Zhang and G.M. Qi (2005). "Neural network forecasting for seasonal and trend time series." European journal of operational research 160: 501-514.
- D. Goldberg (1989). Genetic algorithms in search, optimization, and machine learning. Reading, Addison-Wesley.
- H. Afshin, D. Hadi, B. Ahmad and S. Kamran (2009). "Optimization of the core configuration design using a hybrid artificial intelligence algorithm for research reactors." Nuclear engineering and design 239 (12): 2786-2799.
- H. Demuth and M. Beale (2005). Neural network toolbox user's guide, The mathworks Inc.
- H.B. Demuth and M. Beale (2004). Neural network toolbox for use with MATLAB, user's guide. Natick, MA, USA, The mathworks Inc.
- M.H. Hassoun (1995). Fundamentals of artificial neural networks. Cambridge, Mass, MIT Press.
- S. Haykin (1994). Neural networks: A comprehensive foundation. New York MacMillan.
- Hee-Yeal Yu and Sung-Yang Bang (1997). "An improved time series prediction by applying the layer-by-layer learning method to FIR neural networks." Neural networks 10(9): 1717-1729.
- J. Henseler (1995). Artificial neural networks, an introduction to ANN theory and practice. Berlin, Springer.
- J.H. Holland (1975). Adaptation in natural and artificial systems, University of Michigan Press, Ann Arbor.
- J.J. Hopfield (1982). "Neural networks and physical systems with emergent collective computational abilities." Proceedings of the national academy of sciences of the United States of America 79(8): 2254-2258.
- J.J. Hopfield (1984). "Neurons with graded response have collective computational properties like those of two-state neurons." Proc. Natl. Acad. Sci 81: 3088-3092.
- I. Istadi and A. Nor Aishah (2007). "Modelling and optimization of catalytic-dielectric barrier discharge plasma reactor for methane and carbon dioxide conversion using hybrid artificial neural network-genetic algorithm technique." Chemical engineering science 62(23): 6568-6581.

- I. Kaastra and M. Boyd (1996). "Designing a neural network for forecasting financial and economic time series." Neurocomputing 10(3): 215-236.
- J. F. Kolen and J. B. Pollack (1991). "Back propagation is sensitive to initial conditions." In R. P. Lippmann, J. E. Moody, & D. S. Tpuretzky (Eds.), Advances in neural Information processing systems 3 860-867.
- J. Hertz, A. Krogh and Rg. Palmer (1991). Introduction to the theory of neural computation. Reading, Mass: Addison-Wesley.
- J. Kong and G. Martin (1995 ). A backpropagation neural network for sales forecasting. Proc. of ICNN, IEEE International conference on neural networks: 1007-1011
- J. Nan, Z. Zhiye and R. Liqun (2003). "Design of structural modular neural networks with genetic algorithm." Advances in engineering software 34(1): 17-24.
- J. Zupan and J. Gasteiger (1993). Neural Networks for chemists. an introduction. VCH, Weinheim.
- J.E. Dennis and R.B. Schnabel (1983). Numerical methods for unconstraint optimization and nonlinear equations. Prentice Hall, Englewood Cliffs (NJ)
- I. T. Jolliffe (2002). Principal component analysis, Springer.
- M. Jordan (1986). "Attractor dynamics and parallelism in a connectionist sequential machine." The eighth annual conference of the cognitive science society: 531-546.
- K. Hasan, O. Babur and E. Tuncay (2005). "Warpage optimization of a bus ceiling lamp base using neural network model and genetic algorithm." Journal of materials processing technology 169(2): 314-319.
- K. Hyun-jung and S. Kyung-shik (2007). "A hybrid approach based on neural networks and genetic algorithms for detecting temporal patterns in stock markets." Applied soft computing 7(2): 569-576.
- K. John, L. Mike and B. Marvin (2006). "Effects of the neural network s-Sigmoid function on KDD in the presence of imprecise data." Computers & operations research 33(11): 3136-3149.
- K. Kyoung-jae and H. Ingoo (2000). "Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index." Expert systems with applications 19(2): 125-132.
- K. Peter and S. Mirzaei (2003). "Validation of a parallel genetic algorithm for image reconstruction from projections." Journal of parallel and distributed computing 63(3): 356-359.

- K. Takayuki, N. Hiroyuki and I. Tohru (2007). "A packet routing method for complex networks by a stochastic neural network." Physica A: statistical mechanics and its applications 376: 658-672.
- K. Warne, G. Prasad, S. Rezvani and L. Maguire (2004). "Statistical and computational intelligence techniques for inferential model development: A comparative evaluation and a novel proposition for fusion." Engineering applications of artificial intelligence 17(8): 871-885.
- K.G. Srinivasaa, K.R. Venugopala and L.M. Patnaik (2007). "self-adaptive migration model genetic algorithm for data mining applications." Information sciences 177(20): 4295-4313.
- K.P. Chong and H. Stanislaw (2004). An introduction to optimization. Singapore, Wiley.
- C. Karr (1995). "Adaptive control of an exothermic chemical reaction system using fuzzy logic and mgenetic algorithms." In: L.R. Medsker, Editor, Hybrid intelligent systems, Kluwer, New York.
- J.O. Katz (1992). "Developing neural network forecasters for trading." Technical analysis of stocks and commodities: 58-70.
- T. Kavzoglu (2001). An investigation of the design and use of feedforward artificial neural networks in classification of remotely sensed images, University of Nottingham, UK. PhD thesis.
- H. Kitano (1990). "Empirical studies on the speed of convergence of neural network training using genetic algorithms." Proceedings of the eighth national conference on artificial intelligence AAAI/MIT Press, Boston, MA: 789-795.
- T. Kohonen (1988b). "Representation of sensory information in self-organizing feature maps, and the relation of these maps to distributed memory networks." Computer simulation in brain science.
- A. Kusiak (2000). Computational intelligence in design and Manufacturing. New York, John Wiley & Sons.
- L. Bao, R. Bodil, K. Jørgen and J. Sten (2007). "A systematic approach for soft sensor development." Computers & chemical engineering 31(5-6): 419-425.
- L. Bor-Ren and R. Hoft (2003). "Neural networks and fuzzy logic in power electronics." Control engineering practice 2(1): 113-121.
- L. Davies and U. Gather (1993). "The identification of multiple outliers." Journal of the american statistical association 88(423): 782-792.
- L. Eriksson, E. Johansson, N. Kettaneh-Wold and S. Wold (2001). Multi- and megavariate data analysis principles and applications, Umetrics academy, Sweden.



- L. Xueqiang, C. Xiaoguang, W. Wenfu and P. Guilan (2007). "A neural network for predicting moisture content of grain drying process using genetic algorithm." Food control 18(8): 928-933
- L. Yu-Chuan, L. Li, Ch. Wen-Ta and J. Wen-Shan (2000). "Neural network modeling for surgical decisions on traumatic brain injury patients." International journal of medical Informatics 57(1): 1-9.
- L. Zhengjun, L. Aixia, W. Changyao and N. Zheng (2004). "Evolving neural network using real coded genetic algorithm (GA) for multispectral image classification." Future generation computer systems 20(7): 1119-1129.
- F. Laurene (1994). Fundamentals of neural networks, Prentice-Hall.
- J. Lawrence (1994). "Introduction to neural networks: design, theory, and applications." 6th ed. Nevada City, CA: California Scientific Software.
- C.G. Looney (1996). "Advances in feed-forward neural networks: demystifying knowledge acquiring black boxes." IEEE Trans Knowl Data Eng 8(2): 211-226.
- CG. Looney (1997). Pattern recognition using neural networks, theory and algorithms for engineers and scientists. New York, Oxford university Press.
- M. Delgado and M.C. Pegalaja (2005). "A multiobjective genetic algorithm for obtaining the optimal size of a recurrent neural network for grammatical inference " Pattern recognition 38(9): 1444-1456.
- M. Gallahger and T. Downs (1997). "Visualisation of learning in neural networks using principal component analysis." Proceedings of international conference on computational intelligence and multimedia applications, Australia: 327-331.
- M. Holger and D. Graeme (1998). "The effect of internal parameters and geometry on the performance of back-propagation neural networks: an empirical study." Environmental modelling and software 13(2): 193-209.
- M. Karkoub and A. Elkamel "Modelling pressure distribution in a rectangular gas bearing using neural networks." Tribol Int 30(2): 139-150.
- M. Magdy and D. Houshang (2009). "Neural network based design of fault-tolerant controllers for automated sequential manufacturing systems." Mechatronics 19(5): 705-714.
- M. Majors, J. Stori and C. Dong-il (2002). "Neural network control of automotive fuel-injection systems." Control systems magazine IEEE 14(3): 31-36.
- M. Nasserri, K. Asghari and M.J. Abedini (2008). "Optimized scenario for rainfall forecasting using genetic algorithm coupled with artificial neural network." Expert systems with applications 35(3): 1415-1421.

- M. Nelson and W.T. Illingworth (1990). A practical guide to neural nets. Reading, Mass, Addison-Wesley.
- M. Rychetsky, S. Ortmann and M. Glesner (1998). "Correlation and regression based neuron pruning strategies." In: Fuzzy-neuro-systems '98, Fifth international workshop, Munich D, Germany.
- M. Y. Rafiq, G. Bugmann and D. J. Easterbrook (2001). "Neural network design for engineering applications." Computers & structures 79(17): 1541-1552.
- M. Yang, W. Yun-jia and Ch. Yuan-ping (2009). "An incorporate genetic algorithm based back propagation neural network model for coal and gas outburst intensity prediction." Procedia earth and planetary science 1(1): 1285-1292.
- M.T. Demuth and H.B. Beale (2003). Neural network design, Vikas publishing house.
- M.T. Hagan, H.B. Demuth and M.H. Beale (1996). Neural network design. Boston (MA) PWS publishing.
- M.T. Hagan and M.B. Menhaj (1994). "Training feed forward networks with the Marquardt algorithm." IEEE Trans neural networks 5(6): 989-993.
- T. Masters (1993). Practical neural network recipes in C++. San Diego, California, Academic Press, .
- E. J. Molga (2003). "Neural network approach to support modelling of chemical reactors: problems, resolutions, criteria of application." Chemical engineering and processing 42(8-9): 675-695.
- N. Huang, K.K. Tan and T.H. Lee (2008). "Adaptive neural network algorithm for control design of rigid-link electrically driven robots." Neurocomputing 71(4-6): 885-894.
- N. Perambur S. and A. Preechayasomboon (2002). "Development of a neuroinference engine for ADSL modem applications in telecommunications using an ANN with fast computational ability." Neurocomputing 48(1-4): 423-441.
- N. Somnath, B. Yogesh, L. Jayaram, U. Sridevi, B. S. Rao, T. Sanjeev and K. Bhaskar (2004). "Hybrid process modeling and optimization strategies integrating neural networks/support vector regression and genetic algorithms: study of benzene isopropylation on Hbeta catalyst." Chemical engineering journal 97(2-3): 115-129.
- Stefan P. Niculescu (2003). "Artificial neural networks and genetic algorithms in QSAR." Journal of molecular structure: THEOCHEM 622(1-2): 71-83.
- A.-O. Nii (1999). "Analysis of learning rate and momentum term in backpropagation neural network algorithm trained to predict pavement performance." Advances in engineering software 30(4): 291-302.

- O. Kaynak, E. Alpaydin, E. Oja and L. Xu (2003). Artificial neural networks and neural information processing. Istanbul, Springer.
- O. Makoto, H. Takashi, M. Jun-Ichi, H. Ryuichi and F. Yasumi (2006). "Comparisons of gap-filling methods for carbon flux dataset: A combination of a genetic algorithm and an artificial neural network." Ecological modelling 198(473-486).
- O. Toshiyuki and S. Jun (2004). "Monitoring of flank wear of coated tools in high speed machining with a neural network ART2." International journal of machine tools and manufacture 44(12-13): 1311-1318.
- Ö. Tugrul and N. Abhijit (2002). "Prediction of flank wear by using back propagation neural network modeling when cutting hardened H-13 steel with chamfered and honed CBN tools." International journal of machine tools and manufacture 42(2): 287-297.
- P. J. Rousseeuw and A. M. Leroy (1987). Robust regression and outlier detection. New York, Wiley.
- P. Jung and K. Hong (2007). "Prediction of fatigue life for spot welds using back-propagation neural networks." Materials & design 28(10): 2577-2584.
- E. A. Faria P. M. Ferreira, A. E. Ruano (2002). "Neural network models in greenhouse air temperature prediction." Neurocomputing 43(1-4): 51-75.
- P. Mukta and K. Usha (2009). "Neural networks and statistical techniques: A review of applications." Expert systems with applications 36(1): 2-17.
- P. Staufer and M.M. Fischer (1997). Spectral pattern recognition by two-layer perceptron: effects of training set size, Neurocomputation in remote sensing data analysis, Springer.
- P.G. Benardos and G.-C. Vosniakos (2007). "Optimizing feedforward artificial neural network architecture." Engineering applications of artificial intelligence 20(3): 365-382.
- G.-C. Vosniakos P.G. Benardos (2007). "Optimizing feedforward artificial neural network architecture." Engineering applications of artificial intelligence 20(3): 365-382.
- JD. Paola (1994). Neural network classification of multispectral imagery. Tucson, University of Arizona. MSc thesis.
- H. Patrick (2007). "Application of artificial neural networks to the prediction of sewing performance of fabrics." International journal of clothing and technology 19(5): 291-318.
- R. K. Pearson (2002). "Outliers in process modeling and identification." IEEE transactions on control systems technology 10(1): 55-63.

- Q. Wei, S. Ravi, S. Xiaoshan, S. Xuejun and C. Robert (2005). "Standardization for image characteristics in telemammography using genetic and nonlinear algorithms." Computers in biology and medicine 35(3): 183-196.
- R. B. Boozarjomehry and W. Y. Svrcek (2001). "Automatic design of neural network structures." Computers & chemical engineering 25(7-8): 1075-1088.
- R. Koker, A. Necat and A. Demir (2007). "Neural network based prediction of mechanical properties of particulate reinforced metal matrix composites using various training algorithms." Materials & design 28(2): 616-627.
- R. Lae, S. Lek and J. Moreau (1999). "Predicting fish yield of African lakes using neural networks." Ecological modelling 120: 325-335.
- R. Parekh, J. H. Yang and V. Honavar (2000). "Constructive neural-network learning algorithms for pattern classification." IEEE transactions on neural networks 11(2): 436-451.
- R. Steven, C. John, M. Curtis, G. James, F. Ken, B. Tom, R. Dennis, K. Matthew and O. Mark (1995). "Neural networks for automatic target recognition." Neural networks 8(7-8): 1153-1184.
- S. Biswajit, S. Aritra, D. Sirshendu and P. Sunando (2009). "Prediction of permeate flux during electric field enhanced cross-flow ultrafiltration- A neural network approach." Separation and purification technology 65: 260-268.
- S. Haralambos, A. Alex, M. Stefanos and B. George (2004). "A new algorithm for developing dynamic radial basis function neural network models based on genetic algorithms." Computers & chemical engineering 28(1-2): 209-217.
- S. Jocelyn and D. Robert (1991). "Creating artificial neural network that generalize." Neural networks 4(1): 67-79.
- S. Kyung-Shik and L. Yong-Joo (2002). "A genetic algorithm application in bankruptcy prediction modeling." Expert systems with applications 23(3): 321-328.
- S. Randall and G. Jatinder (2000). "Comparative evaluation of genetic algorithm and backpropagation for training neural networks." Information sciences 129(1-4): 45-59.
- S. Shital and K. Andrew (2007). "Cancer gene search with data-mining and genetic algorithms." Computers in biology and medicine 37(2): 251-261.
- S. Taeksoo and H. Ingoo (2000). "Optimal signal multi-resolution by genetic algorithms to support artificial neural networks for exchange-rate forecasting." Expert systems with applications 18(4): 257-269.
- S. V. Kamarthi and S. Pittner (1999). "Accelerating neural network training using weight extrapolations." Neural networks 12( 9): 1285-1299.

- S.H. Mousavi, A. Bahramia, H.R. Madaah and A. Shafyeib (2006). "Using genetic algorithm and artificial neural network analyses to design an Al-Si casting alloy of minimum porosity " Materials & design 27(7): 605-609
- S.K. Lahiri and K.C. Ghanta (2008). "Development of an artificial neural network correlation for prediction of hold-up of slurry transport in pipelines." Chemical engineering science 63(6): 1497-1509.
- S.L. Mok, C.K. Kwong and W.S. Lau (2001). "A hybrid neural network and genetic algorithm approach to the determination of initial process parameters for injection Moulding." The international journal of advanced manufacturing technology 18(6): 404-409.
- S.S. Panda, D. Chakraborty and S.K. Pal (2008). "Flank wear prediction in drilling using back propagation neural network and radial basis function network." Applied soft computing 8(2): 858-871.
- J. Scheffer (2002). "Dealing with missing data." Research letters in the information and mathematical sciences 3(1): 153-160.
- D. Seidl, Lorenz, R., (1991). "A structure by which a recurrent neural network can approximate a nonlinear dynamic system." Proceedings of the international joint conference on neural networks1991 II: 709-714.
- L. Shuliang (2000). "The development of a hybrid intelligent system for developing marketing strategy." Decision support systems 27(4): 395-409.
- H.T. Siegelman and E.D. Sontag (1995). "On computational power of neural networks." J. Comput. Syst. Sci 50(1): 132-150.
- K. Swingler (1996). Applying neural networks a practical guide. New York, Academic Press.
- T. Chih-Fong and W. Jhen-Wei (2008). "Using neural network ensembles for bankruptcy prediction and credit scoring." Expert systems with applications 34(4): 2639-2649.
- T. Clarence and W. Gerhard (1993). "A Study of the parameters of a backpropagation stock price prediction model." IEEE Trans neural networks.
- T. Imdat and I. Yasar (2009). "Prediction of convection heat transfer in converging-diverging tube for laminar air flowing using back-propagation neural network." International communications in heat and mass transfer 36(6): 614-617.
- T. Morimoto, J. De Baerdemaeker and Y. Hashimoto (1997). "An intelligent approach for optimal control of fruit-storage process using neural networks and genetic algorithms." Computers and electronics in agriculture 18(2-3): 205-224.

- T. Reeti, A. Sudeshna and A. K. Jeffrey (2006). "Neural networks for longitudinal studies in Alzheimer's disease." Artificial intelligence in medicine 36( 3): 245-255.
- T.M. Nabhan and A.Y. Zomaya (1994). "Toward generating neural network structures for function approximation." Neural networks 7(1): 89-99.
- J. Tu (1996). "Advantages and disadvantages of using artificial neural networks versus logistic regression in predicting medical outcomes." J Clin Epidemiol 49: 1225-1231.
- V.R. Adineh, C. Aghanajafi, G.H. Dehghan and S. Jelvani (2008). "Optimization of the operational parameters in a fast axial flow CW CO<sub>2</sub> laser using artificial neural networks and genetic algorithms." Optics & laser technology 40(8): 1000-1007.
- W. Duch and N. Jankowski (2001). "Transfer functions: hidden possibilities for better neural networks." Proceedings of the ninth European symposium on artificial neural networks (ESANN), Brugge: 81-94.
- W. Richard, G. Meirion and N. S. Haitham (1998). "Dynamic ultrafiltration of proteins - A neural network approach." Journal of membrane Science 146(2): 225-235.
- W. Schmidt, S. Raudys, M. Kraaijveld, M. Skurikhina and R. Duin (1993). Initialization, backpropagation and generalization of feed-forward classifiers. In: Proceeding of the IEEE international conference on neural networks.
- W.J. Egan and S.L. Morgan (1998). "Outlier detection in multivariate analytical chemical data,." Analytical chemistry 70: 2372-2379.
- L. William (2004). Petroleum refining in nontechnical language, PennWell Books.
- BJ. Wythoff (1993). Backpropagation neural networks: a tutorial, Chemometr Intell Lab Syst.
- B.J. Wythoff (1993). "Backpropagation neural networks: a tutorial." Chemometr. Intell. Lab. Syst. 18: 115-155.
- X.-H. Yu and G.-A. Chen (1997). "Efficient backpropagation learning using optimal learning rate and momentum." Neural networks 10(3): 517-527.
- Y. Jinn-Yi and L. Wen-Shan (2007). "Using simulation technique and genetic algorithm to improve the quality care of a hospital emergency department." Expert systems with applications 32(4): 1073-1083.
- Y. Lee, S.H. Oh and M. Kim (1991). The effect of initial weights on premature saturation in backpropagation learning. In: Proceedings of an international joint conference on neural networks, Seattle, WA.

- Y. Serkan, D. Cem and A. Serhat (2002). "Application of artificial neural networks to optimum bit selection." Computers & geosciences 28(2): 261-269.
- Y. Taho, L. Huan-Chang and C. Meng-Lun (2006). "Metamodeling approach in solving the machine parameters optimization problem using neural network and genetic algorithms: A case study." Robotics and computer-integrated manufacturing 22(4): 322-331.
- X. Yao (1993). "A review of evolutionary artificial neural networks." 8 (International journal of intelligent systems ): 539-567.
- X. Yao (1999). "Evolving artificial neural networks." Proceedings of the IEEE 87(9): 1423-1447.
- Z. Jiu-sun and G. Chuan-hou (2009). "Improvement of identification of blast furnace ironmaking process by outlier detection and missing value imputation,." Journal of process control 19(9): 1519-1528.
- Z. Uros and C. Franci (2003). "Optimization of cutting conditions during cutting by using neural networks." Robotics and computer integrated manufacturing 19 189-199.
- Z. Xiaotian, X. Hong and L. Huaizu (2008). "Predicting stock index increments by neural networks: The role of trading volume under different horizons." Expert systems with applications 34(4): 3043-3054.
- L. Zhang (1993). Model and its application of artificial neural networks. Shanghai, Fudan University press.