Computational issues in process optimisation using historical data.

Nawi, Nazri Mohd

CIVIL AND COMPUTATIONAL ENGINEERING CENTRE
SWANSEA UNIVERSITY

COMPUTATIONAL ISSUES IN PROCESS OPTIMISATION USING HISTORICAL DATA

NAZRI MOHD NAWI

B.Sc., M.Sc. (Malaysia)

THESIS SUBMITTED TO THE SWANSEA UNIVERSITY IN FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

AUGUST 2007

SUMMARY

This thesis presents a new generic approach to improve the computational efficiency of neural-network-training algorithms and investigates the applicability of its '*learning from examples*' featured in improving the performance of a current intelligent diagnostic system. The contribution of this thesis is summarised in the following two points:

- For the first time in the literature, it has been shown that significant improvements in the computational efficiency of neural-network algorithms can be achieved using the proposed methodology based on using adaptive-gain variation.
- The capabilities of the current Knowledge Hyper-surface method (Meghana R. Ransing, 2002) are enhanced to overcome its existing limitations in modelling an exponential increase in the shape of the hyper-surface.

Neural-network techniques, particularly back-propagation algorithms, have been widely used as a tool for discovering a mapping function between a known set of input and output examples. Neural networks learn from the known example set by adjusting its internal parameters, referred to as weights, using an optimisation procedure based on the 'least square fit principle'. The optimisation procedure normally involves thousands of iterations to converge to an acceptable solution. Hence, improving the computational efficiency of a neural-network algorithm is an active area of research. Various options for improving the computational efficiency of neural networks have been reviewed in this thesis. It has been shown in the existing literature that the variation of the gain parameter improves the learning efficiency of the gradient-descent method. However, it can be concluded from previous researchers' claims that the adaptive-gain variation improved the learning rate and hence the efficiency. It was discovered in this thesis that the gain variation has no influence on the learning rate; however, it actually influences the search direction. This made it possible to develop a novel approach that modifies the gradient-search direction by introducing the adaptive-gain variation. The proposed method is robust and has been shown that it can easily be implemented in all commonly used gradient-based optimisation algorithms. It has also been shown that it significantly improves the computational efficiency as compared to existing neural-network training algorithms. Computer simulations on a number of benchmark problems are used throughout to illustrate the improvement proposed in this thesis.

In a foundry a large amount of data is generated within the foundry every time a casting is poured. Furthermore, with the increased number of computing tools and power there is a need to develop an efficient, intelligent diagnostic tool that can learn from the historical data to gain further insight into cause and effect relationships. In this study the performance of the current Knowledge Hyper-surface method was reviewed and the mathematical formulation of the current Knowledge Hyper-surface method was analysed to identify its limitations. An enhancement is proposed by introducing mid-points in the existing shape formulation. It is shown that the mid-points' shape function can successfully constrain the shape of decision hyper-surface to become more realistic with an acceptable result in a multi-dimensional case. This is a novel and original approach and is of direct relevance to the foundry industry.

DECLARATION

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed ..... (candidate)

Date 18-11-2017

STATEMENT 1

This thesis is the result of my own investigations, except where otherwise stated. Where correction services have been used, the extend and nature of the correction is clearly marked in a footnote(s). Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed ..... (candidate)

Date 18-11-2017

STATEMENT 2

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loans, and for the title and summary to be made available to outside organisations.

Signed ..... (candidate)

Date 18-11-2017

CONTENTS

CHAPTER 3 : ENHANCED LEARNING ALGORITHM FOR BACK PROPAGATION NETWORK

CHAPTER 4 : RESULTS AND VALIDATION ON BENCHMARK PROBLEMS

CHAPTER 5 : ENHANCEMENT TO THE METHOD PROPOSED FOR CONSTRUCTING OPTIMAL KNOWLEDGE HYPER-SURFACE

LIST OF FIGURES

Appendix A

LIST OF TABLES

ACKNOWLEDGEMENTS

Working as a Ph.D. student in Swansea University was a magnificent as well as challenging experience to me. In all these years, many people were instrumental directly or indirectly in shaping up my academic career. It was hardly possible for me to thrive in my doctoral work without the precious support of these personalities. Here is a small tribute to all those people.

My sincere thanks first and foremost go to my first supervisor, Dr. Rajesh S. Ransing, for his patience and understanding in guiding me throughout this PhD research. I really appreciate the enormous time that Dr. Rajesh spent with me to discuss my research in detail, from the first discussion up to the last modifications to the text of this thesis. I learned a tremendous amount about how to concentrate on setting up and solve various scientific questions. His company and assurance at the time of crisis would be remembered lifelong.

From those others who helped me the most in Swansea University, I would like to thanks my second supervisor, Prof. David Gethin and the officers in Post Graduate Office, School of Engineering for helping me in various ways to clarify the things related to my academic works in time with excellent corporation and guidance. Special acknowledgment is also given to Malaysian Public Service Department and Universiti Tun Hussein Onn Malaysia (UTHM) for funding this research.

I express my gratitude to Dr. Meghana for her kind assistance and support with all the convenience I needed. I am sincerely grateful to my colleague, Dr. Peter Dyson for correcting my English. I am grateful to group at Civil and Computational Engineering Centre for sharing their knowledge and life experience with me.

I thank my parents for their continuous prayer, patience, support and love whenever I need it over these years. I also would like to dedicate this work to my two young children, Aina and Hafiz for their silent prayer for my work at the time when they needed my company most. They are always the source of motivation behind me.

Finally, I would like to express my deepest appreciation to my beloved wife, Zawati Harun, for her presence in my life. We have been each others best friends and the strongest supporters all these times. Without her loving support and understanding I would never have completed my present work. Her patience and encouragement was the drive for me.

ABBREVIATIONS AND NOMENCLATURE

BP	Back propagation
DOE	Design of experiment
OA	Orthogonal array
NN	Neural network
RSM	Response surface method
MLP	Multi layer perceptron
PS	Premature saturation
$g^{(n)}$	gradient of error at step n
η	Learning rate
$x_1,...,x_i,...,x_l$	Set of input signals
$h_1,...,h_j,...,h_m$	Set of output signals of hidden nodes in the first hidden layer (suffice j and m are used for hidden nodes)
$o_1,...,o_k,...,o_n$	Set of output signals of output nodes in the output layer (suffice k and n are used for output nodes)
a_k	Activation function of k^{th} unit
α	Momentum term
E	Error function
f	The squashing or activation function of the a processing unit
w_{ij}	Weight of the link form unit i to unit j
w_{jk}	Weight of the link form unit j to unit k
$d_{(n)}$	Search direction at step n
θ_j	Bias for the j^{th} unit
c_j	Gain of the j^{th} unit
$a_{net,j}$	Net input activation function for the j^{th} unit
β	Momentum coefficient in conjugate gradient
CT	Convergence tolerance
Nt	Number of free parameters (weights and bias)

H	Hessian matrix
$\Delta \bullet (n)$	Weight, bias or gain correction of $\bullet$ for n^{th} training iteration
$\partial \bullet$	Partial derivative of $\bullet$
N	The number of design variables
$x_i,...,x_j$	Set of design variables
$c \quad (c = 0 to \quad 1)$	Output belief value in cause
w_j	Weight variable associated with the j^{th} reference point
$\xi^j (j = 1 to \quad p)$	Belief values for '$_p$' effects
$z_i (i = 1 to \quad m)$	Function of $\xi^j (j = 1 to \quad p)$
MSE	Mean squared error
QN	Quasi-Newton
LM	Levenberg-Marquardt
GDM/AG	Gradient descent with momentum and Adaptive Gain
CGFR/AG	Conjugate gradient - Fletcher-Reeves method with Adaptive Gain
CGPR/AG	Conjugate gradient - Polak-Ribiere method with Adaptive Gain
BFGS/AG	Broyden-Fletcher-Goldfarb-Shanno method with Adaptive Gain
DFP/AG	Davidon-Fletcher-Powell with Adaptive Gain

GLOSSARY OF TERMS

Activation function	A mathematical function applied to a node's activation that computes the signal strength it outputs to subsequent nodes.
Adaptive gain value	Gain value that is adjusted according to an algorithm during training to minimize error.
Architecture	Description of the number of the layers in a neural network, each layer's transfer function, the number of neurons per layer, and the connections between layers.
Artificial Neural network	A computer simulation of the human brain which consists of at least one neuron and a set of synapses. Neurons have an activation level and a transfer function.
Back-propagation	A supervised learning algorithm which uses data with associated target output to train an artificial neural network.
Benchmark	A test that measures the performance of a system or a method on a well-defined task or set of tasks.
Belief value	A value that representing the strength of the occurrence of effect or cause.
BFGS Quasi-Newton method	Variation of Newton's optimization algorithm, in which an approximation of the Hessian matrix is obtained from gradients computed at each iteration of the algorithm.

Bias	Neuron parameter that is summed with the neuron's weighted inputs and passed through the neuron's transfer function to generate the neuron's output.
Conjugate gradient algorithm	In the conjugate gradient algorithms, a search is performed along conjugate directions, which produces generally faster convergence than a search along the steepest descent directions.
Epoch	One iteration through the neural network training algorithm (presentation of the entire training set once to the network).
Feed-forward network	Layered network in which each layer only receives inputs from previous layers.
Fletcher-Reeves update	Method for computing a set of conjugate directions. These directions are used as search directions as part of a conjugate gradient optimization procedure.
Function approximation	Task performed by a network trained to respond to inputs with an approximation of a desired function.
Gain value	Training parameter that controls the steepness of activation function during learning.
Generalisation performance	The ability of a trained network to correctly classify on a set of unseen data which is similar to but not the same as the training data set by finding their similarities with training data set patterns.
Golden section search	Linear search that does not require the calculation of the slope. The interval containing the minimum of the

	performance is subdivided at each iteration of the search, and one subdivision is eliminated at each iteration.
Gradient descent	Process of making changes to weights and biases, where the changes are proportional to the derivatives of network error with respect to those weights and biases. This is done to minimize network error.
Heuristic	A method that serves as an aid to problem solving. It is sometimes defined as any 'rule of thumb'.
Learning	Process by which weights and biases are adjusted to achieve some desired network behavior.
Learning rate	Training parameter that controls the size of weight and bias changes during learning.
Momentum	A constant that is often used to make it less likely for a back-propagation network to get caught in a shallow minimum.
MSE	Performance function that calculates the average squared error between the network outputs O_K and the target outputs t_K.
Network topology	Ways to arrange nodes in a network.
One Dimensional (1D)	A plotted graph that shows only one defect is connected to the a cause
Optimisation	A process that finds a best, or optimal, solution for a selected model.
Over fitting	Case in which the error on the training set is driven to a

	very small value, but when new (unseen) data is presented to the network, the error is large.
Performance	Behavior of a network or a method.
Perceptron	The basic processing element used in neural networks without hidden layer. A simple analog circuit with weighted inputs and a nonlinear decision element such as a hard limiter, threshold logic or sigmoid nonlinearity.
Polak-Ribiére update	Method for computing a set of conjugate directions. These directions are used as search directions as part of a conjugate gradient optimization procedure.
Premature saturation	The situation where the instantaneous sum of differences between network output and target value is almost unchanged for some period of time.
Quasi-Newton algorithm	Class of optimization algorithm based on Newton's method. An approximate Hessian matrix is computed at each iteration of the algorithm based on the gradients.
Search direction	The choice of direction where error function decreases most quickly.
Sigmoid activation function	Squashing function of the form shown below that maps the input to the interval [0, 1].
Two Dimensional (2D)	A plotted graph that shows two defect are connected to the a cause

Validation	The process of testing the models with a data set different from the training data set.
Weight	The strength of connection between two nodes.

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Every day foundries manufacture a large number of castings. Every time a casting is produced, a large amount of data is generated involving process-parameter values and one or more indicators on whether the casting is defective or not. This data is encoded for each type of defect, for each day, week and month of the casting process and is available for all casting components.

The rejection data for a given casting and time frame, normally indicates a pattern, which has normally few defects occurring at significantly high proportions and some occurring at significantly low proportions. Therefore, the diagnostic casting problem was defined as recognising patterns in the casting rejection data and identifying a corresponding combination of causes. It was observed that a combination of defects generally occurs as a result of a combination of causes (Meghana R. Ransing, 2002).

The cause and effect relationship is generally complex and highly interlinked for many manufacturing processes. Identification of the degree of influence of a cause on the occurrence of a defect is one of the most difficult tasks in a diagnostic process and the highly interlinked causal relationship further complicates the problem. Furthermore, many foundries monitor their process closely and record vast amounts

of data. There is a need to develop a computational tool that can learn from the historical data and provide options to minimise production of defective components.

1.2 AN OVERVIEW OF THE OPTIMISING CASTING PROCESS USING HISTORICAL DATA

The cause and effect relationship in a casting process is complex and non-linear. Furthermore, a large number of parameters are needed to be coordinated with each other in an optimal way to minimise the occurrence of defective castings. This has led to the necessity of developing computer-based optimisation techniques. An optimisation process is a computational technique that determines an optimal value for process parameters such that the magnitude of one or more response variables of the process is minimised. It also ensures that the process operates within established limits or constraints (B. Lally *et al.*, 1991). Casting process optimisation has facilitated foundrymen in making right choices, but it still remains a challenging area that has drawn the attention of many researchers during the last two decades

Recent studies have used the response surface method (RSM) to optimise parameters in the casting process (J. Grum and J.M. Slabe, 2004; Theodore T. Allen and Liyang Yu, 2002). The computational efficiency of the RSM approach significantly reduces as the number of process parameters increase (David L. Rodriguez, 2003). This is mainly because RSM techniques show the same limitations as showed by polynomial-regression techniques; the number of unknowns in the system increases exponentially with the number of parameters.

In contrast, Taguchi's robust design method provides a process engineer with a systematic and efficient approach for conducting experimentation to determine near optimum settings of design parameters for performance and cost (A. Bendall, 1988; R.N. Kackar, 1985; S.M. Phadke, 1989). The robust design method uses orthogonal arrays (OA) to study the parameter space, usually containing a large number of decision parameters, with a small number of experiments. To this date, a quite significant amount of research and development work has been done in order to

optimise parameters of the casting process by using the Taguchi method (G.P. Syrcos, 2003; Y.V. Kamat and M.V. Rao, 1994; H. Singh and P. Kumar, 2005; V.D. Tsoukalas *et al.*, 2004).

Recently, the artificial-neural networks (ANN), or simply neural-networks (NN), technique has gained more popularity in learning cause and effect analysis in casting processes (D. Barschdorff *et al.*, 1997; H. Lin *et al.*, 1995; M. Perzyk and A. Kochanski, 2003; E.E. Martinez *et al.*, 1994; Prasad K. Yarlagadda, 2000; H.C. Zang and S.H. Huang, 1995). ANN consists of interconnected cells, called neurons, and simulates the behaviour of the biological neural network in a human brain (R.L. Wilson and R. Sharda, 1994). Neural-networks' techniques are able to adapt, learn from examples and are generally used to model complex relationships between inputs and outputs or to classify data finding common patterns (K. Funahashi, 1989). This ability makes the field of diagnosis a potential application for neural networks.

A new approach, which is of direct relevance to the manufacturing industry, was proposed by Ransing (Meghana R. Ransing, 2002). The proposed method used Lagrange Interpolation polynomials to explore how the degree of influence of each cause on the occurrence of a defect or a combination of defects can be quantified based on past diagnostic examples. For some selected data sets the method showed superior extrapolation abilities as a result of the networks' ability to constrain the shape of the resulting multi-dimensional hyper-surface to the known variation in the belief values in causes and effects. Furthermore, the proposed method had reduced the number of unknowns to an acceptable number which improved computational efficiency as compared to the RSM approach. This work was also compared with neural-network techniques.

The thesis also proposed initial work on using one of the internal parameters of neural networks (i.e. gain) in improving its computation efficiency.

The objective of the work presented in this thesis is to continue the research proposed by Ransing (2002) in order to: (i) overcome the limitation of the current Knowledge

Hyper-surface method; and (ii) to develop a new and robust methodology for improving computational efficiency of neural networks using the gain value.

1.3 RESEARCH CHALLENGES

Based on the research objective presented in Section 1.2, the following research challenges were identified:

1) A neural-network training algorithm, particularly back propagation (BP) algorithm, has been widely known as a tool for mapping non-linear relationships between input and output examples. Many variations of this algorithm have been proposed by previous researchers to increase the BP training efficiency and one of the approaches is to adjust the slope of activation function. Previous researchers have claimed that changing the gain value in a BP algorithm is equivalent to changing the learning-rate value. Efficient methods are currently available to decide an optimal learning-rate value at every iteration of the optimisation process, and hence this research direction was probably not taken forward. It was discovered during this research that the gain variation does not influence the learning rate but it actually affects the search direction. The challenge in this research direction was to prove the finding, develop a mathematical formulation, and validate it on a number of benchmark problems.
2) The diagnosis of defective castings has always been a centre of attention in the manufacturing industry. An intelligent diagnosis system should be able to diagnose effectively the causal representation and also justify its diagnosis. A previous method, known as the Knowledge Hyper-surface method, proposed by Ransing (2002), had shown that the belief value of the occurrence of cause with respect to the change in the belief value in the occurrence of effect can be modelled by linear, quadratic or cubic relationships. However, the methodology was unable to model exponential increase/decrease in belief values in cause and effect relationships. A challenge in this research direction was to propose a strategy that is computationally efficient and able to model

the exponential increase/decrease in belief values in cause and effects relationships without introducing the side-effects of 'over fitting'.

1.4 SCOPE OF WORK AND RESEARCH CONTRIBUTIONS

A summary of research achievements is outlined below:

- Detailed review of various methods that influence the computational efficiency of neural networks.
- Study the effect of the newly proposed method that combines adaptive gain variation with adaptive learning rate and analyses the performance of the proposed method on back propagation training algorithms.
- Discovery that the introduction of gain variation actually improves the search direction and not the learning rate as presumed by previous researchers. As a result, for the first time in the literature, it has been demonstrated that the adaptive gain variation can be used with a number of gradient-based optimisation methods.
- The transcription of the algorithm to computer codes was achieved by building on MATLAB programming language.
- Validate the performance of the proposed method with the conventional algorithm and neural-network toolbox method on a number of benchmark problems.
- Discover some of the practical limitations of the current Knowledge Hyper-surface method and bring about enhancements by constructing a midpoints method on the existing version to achieve better results.
- Compare the performance of the integration system with the current version of Knowledge Hyper-surface method on real casting data.

The major research contributions are summarised as follows:

- A novel approach for improving the training efficiency of BP neural-network algorithms has been proposed with respect to gain variation. It was discovered in this thesis that adaptive gain variation actually improves the gradient search

direction instead of the learning rate as claimed by previous researchers. A coupled algorithm has been proposed that adaptively adjusts the learning rate and gain variation in order to speed up the BP neural-network training process.

- For the first time, the proposed method has been successfully implemented into other well-known optimisation methods. This was done with an objective of improving the computational efficiency of the neural-networks training process. The robustness of the proposed method has been validated against a number of commonly used optimisation methods by using a variety of benchmark problems.
- Enhancements were implemented into the current Knowledge Hyper-surface method to overcome limitations posed by the existing version and by constructing a midpoints method.

The research output during the entire course of study period was documented and a number of publications, as a result, originated or are forthcoming and are listed next.

1.5 LIST OF PUBLICATIONS

The following publications were produced during the course of the research-study period.

- N. M. Nawi, M. R. Ransing, and R. S. Ransing: "An improved Conjugate Gradient based learning algorithm for back propagation neural networks", *International Journal of Computational Intelligence,* March 2007, Vol. 4, No. 1, pp. 46-55.

- R. S. Ransing, N. M. Nawi and M. R. Ransing: "A new method to improve the gradient based search direction to enhance the computational efficiency of back propagation based Neural Network algorithms: Part I: Theory", *Submitted to IEEE Transaction on Neural Networks, 2007.*

- R. S. Ransing, N. M. Nawi and M. R. Ransing: "A new method to improve the gradient based search direction to enhance the computational efficiency of

back propagation based Neural Network algorithms: Part II: Validation on benchmark problems and discussion of results", *Submitted to IEEE Transaction on Neural Networks, 2007.*

- N. M. Nawi, M. R. Ransing, and R. S. Ransing: "Improving the gradient based search direction to enhance training efficiency of back propagation based neural network algorithms", *Proceedings of the 26th International Conference of Innovative Techniques and Applications of Artificial Intelligent (SGAI'06)*, Cambridge, UK, 11th-13th December, 2006, pp. 45-58.

- N. M. Nawi, M. R. Ransing, and R. S. Ransing: "A new efficient search direction for conjugate gradient training methods", *Proceedings of the 3rd International Conference Artificial Intelligence in Engineering and Technology (ICAIET'06)*, Sabah, Malaysia, 22nd-24th November, 2006, pp. 176-181.

- N. M. Nawi, M. R. Ransing, R. S. Ransing: "An improved learning algorithm based on the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method for back propagation neural networks", *Proceedings of the 6th International Conference on Intelligent Systems Design and Applications (ISDA'06)*, Jinan, China, 16th-18th October, 2006, vol. 1, pp. 152-157.

- N. M. Nawi, M. R. Ransing, and R. S. Ransing: "An improved learning algorithm based on the Conjugate Gradient method for back propagation neural networks", *Proceedings of the 14th International Conference on Computational and Information Sciences (ICIS '06)*, Prague, Czech Republic, 25th-27th August, 2006, vol. 14, pp. 211-215.

1.6 OUTLINE OF THE THESIS

The thesis is subdivided into six chapters, including the introduction and conclusion chapters. The following is the synopsis of each chapter.

- *Chapter One: Introduction.* Apart from providing an outline of the thesis, this chapter contains an overview of the background to research work, objectives, scope of the research, and research contributions made during the period of study.
- *Chapter Two: Review of efficient learning methods for back propagation networks.* The back propagation (BP) algorithm is one of the best known and widely used learning algorithms for neural networks. However, its convergence rate can be very slow. Researchers have tried to improve its computational efficiency by using adaptive learning rate values, momentum term, gain tuning of activation functions, network topology and different learning algorithms. This chapter reviews the research contribution made by various researchers to improve the training efficiency of neural networks. One of the modifications is to change the gain parameter used in the activation function. This chapter demonstrates some of the misconceptions claimed by the previous literature on using the gain value and a detailed description of the method proposed by Ransing (2002) is given. At the end of this chapter, some of the advantages posed by the current method are outlined. This chapter lays a foundation for introducing a novel and innovative algorithm for improving the learning efficiency as described in Chapter Three.
- *Chapter Three: Enhanced learning algorithm for back propagation network.* This chapter extends the work on using the adaptive-gain variation as proposed in Chapter Two. It was discovered that the gain variation influences the search direction used in an optimisation process. Since most of the gradient-based optimisation algorithms employed during the training process of BP networks use the negative gradient of error as a gradient-based search direction. An improved and efficient algorithm has been presented that adaptively modified the gradient-based search direction by using the gain parameter used in the activation function. The implementation of the proposed method into other optimisation methods is presented. The proposed method is programmed in MATLAB programming language and is tested for its correctness on a simple sine curve approximation function. The results of the proposed method are then compared to facilitate further testing and validation in the next chapter.

- *Chapter Four: Results and validation on benchmark problems.* The new method developed in Chapter Three is further validated for its efficiency and accuracy on a variety of benchmark problems. The performance of the proposed method is tested in two ways: (a) the speed of convergence measured in number of iterations and CPU time; and (b) the classification accuracy on testing data from the benchmark problems. The benchmark problems used to verify the proposed algorithm are taken from the open literature (Lutz Prechelt, 1994). The results are then discussed for their interpretation and implementation through various optimisation methods.
- *Chapter Five: Improved method for constructing optimal Knowledge Hyper-surface.* A detailed description of the current Knowledge Hyper-surface method proposed by Ransing (2002) is given. Then the limitations posed by the current Knowledge Hyper-surface method are outlined. The proposed enhancements are implemented in the method to overcome the limitations by constructing midpoints between each primary weight along each dimension. The new improved algorithm is then tested on real-casting data.
- *Chapter Six: Conclusion and future work.* The novel research contributions are summarised and recommendations are made for further continuation of work.

CHAPTER 2

REVIEW OF EFFICIENT LEARNING METHODS FOR BACK PROPAGATION NETWORKS

CHAPTER LAYOUT

In this chapter, the back propagation (BP) algorithm, which is one of the best known and widely used learning algorithms for neural networks, is reviewed in detail. The second section of this chapter highlights the limitations of the conventional BP-training algorithm. In particular, two major issues are identified which are namely convergence to a local minima and long-learning time. The next section then discusses some improvements and contributions suggested by various researchers to overcome the limitations. The two major areas of improvement that have been identified in the literature are: (a) firstly, the use of heuristic-based techniques that modify network parameters such as learning rate value, momentum term, activation function, and topology optimisation; and (b) the integration of (a) with second-order optimisation techniques for minimising the error. In heuristic based networks, the gain value is one of the less commonly used parameters for improving learning efficiency. The next section of this chapter studies the relevant literature on gain parameter in detail and demonstrates some of the improvements proposed by previous researchers using the gain value for improving computational efficiency. Then a detailed description of the method proposed by Ransing (2002) is given and some of the advantages and disadvantages posed by the current method are outlined in the next section. This lays the foundation for the next chapter that describes a new and robust methodology to further improve the learning efficiency of the method proposed by Ransing (2002).

2.1 INTRODUCTION

The most popular artificial neural-networks' (ANN) architecture is called multilayer perceptrons (MLP) because of its similarity to perceptron networks with more than one layer. The MLP refer to the network consisting of a set of sensory units (source nodes) that constitute the input layer, one or more hidden layers of computation nodes, and an output layer of computation nodes. Nodes or neurons in any layer of the network are connected to all neurons in the previous layer. The input signal propagates through the network in a forward direction, from left to right and on a layer-by-layer basis. In Figure 2.1, a detailed schema of MLP with a single hidden layer is given.

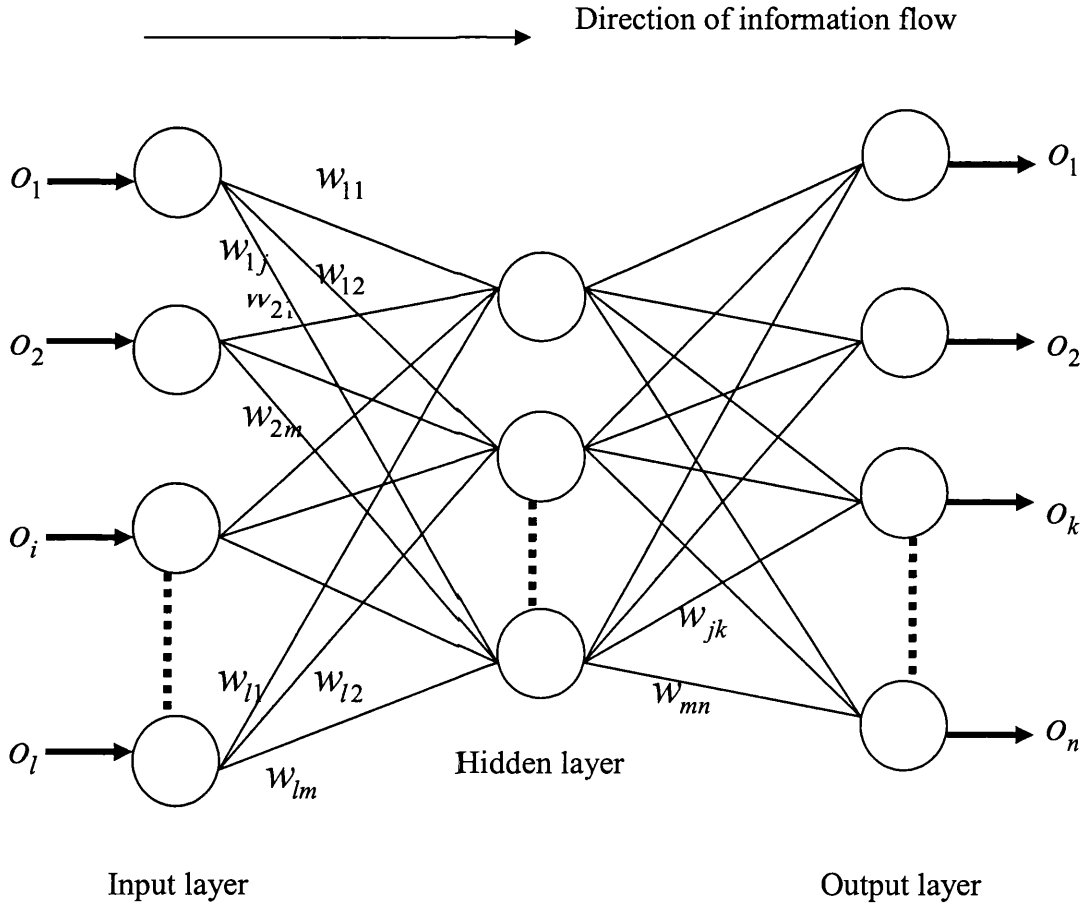


Figure 2.1: Multilayer Perceptrons (MLP).

The training of MLP in neural networks is also known as supervised learning processes and can be interpreted as an example of an optimisation method. The objective of a learning process is to find a weight vector w^* which minimises the

difference between the actual output o_k and the desired output t_k or can be defined as error function $E(w)$.

$$E = \frac{1}{2}\sum_{k=1}^{n}(t_k - o_k)^2 \tag{2.1}$$

where:

n : number of output nodes in the output layer.

t_k : desired output of the k^{th} output unit.

o_k : network output of the k^{th} output unit.

The error function in a one dimensional weight space can be visualised as shown in Figure 2.2.

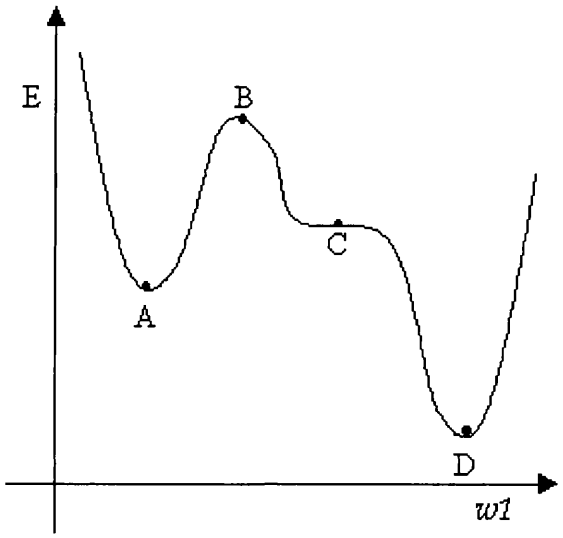


Figure 2.2: Schematic error function for a single parameter w, showing four stationary points, at which $\nabla E(w) = 0$. Point A is a local minimum, point B is a local maximum, point C is a saddle point, and D is the global minimum.

For networks with more than one layer of adaptive weights, the error function is a non-linear function of weights and may have many minima, which satisfy the following equation:

$$\nabla E(w) = 0 \tag{2.2}$$

Where $\nabla E(w)$ denotes the gradient of E with respect to weights. The point at which the value of the error function is smallest (point D in Figure 2.2) is called the global minimum while all other minima are called local minima. There may also be other points, which satisfy conditions (Equation 2.2) such as local maximum (point B, Figure 2.2) or saddle point (point C, Figure 2.2).

2.2 BACK PROPAGATION ALGORITHM (SUPERVISED LEARNING)

Multilayer perceptrons (MLP) training is an iterative process which involves, at each iteration or epoch, the calculation of the network outputs for (one or more) patterns in the training set, and the adjustment of the network weights according to the difference between the actual network output and the desired output. Given a suitable network architecture and training algorithm, the network weights will be progressively adjusted to the point where the network output is acceptably close to the desired output for each pattern in the training set. The most widely used training algorithm for updating the MLP weights during the training process is known as the back propagation (BP) algorithm. The BP algorithm has been independently derived by several researchers working in different fields. Werbos (P.J. Werbos, 1974) discovered the BP algorithm while working on his doctoral thesis in statistics and called it *the dynamical feedback algorithm*. Parker (D. Parker, 1985) rediscovered the BP algorithm in 1982 and called it *the learning logic algorithm*. Finally, in 1986, Rumelhart, Hinton and Williams (D.E. Rumelhart *et al.*, 1986) rediscovered the algorithm and the technique became widely known. Even today, the vast majority of MLP research uses a version of BP algorithm. For this reason, BP can be viewed as the benchmark against which all other training methods are judged.

The BP algorithm implements the steepest descent (gradient-descent) method which is the most venerable, but also one of the least effective, classical optimisation strategies. There are many different versions of the basic BP algorithm, and new

modifications are regularly published in neural-network journals (see Section 2.4). However, this section begins by considering the traditional implementation of the BP algorithm (as presented in Rumelhart and colleagues (1986)) known as batch or off-line BP. The procedure for supervised error-back-propagation is as follows:

Step 1 Start the cycle by presenting input patterns to the neural network.

Step 2 Specify desired outputs for each input pattern.

Step 3 The input pattern is then propagated forward through the network, layer-by-layer until the output layer.

Step 4 A set of output produced is considered as the actual response of the network. Steps **1, 2, 3** and **4** constitute the **'Forward propagation phase'** in that the signal propagates from nodes in the input layer to nodes in the output layer.

Step 5 Error is calculated by comparing the network output with the desired output by using Equation 2.1.

Step 6 The error signal (E) is propagated backwards through the network and is used to adjust the weights. The weights in the links connecting to output nodes (w_{jk}) are then modified based on the gradient descent method as follows:

$$\Delta w_{jk} = \eta(-\frac{\partial E}{\partial w_{jk}}) \tag{2.3}$$

$$= \eta \delta_k o_j$$

where:

o_j : Output of the j^{th} hidden node.

The error is propagated backwards to compute the error specifically, at the hidden nodes:

$$\Delta w_{ij} = \eta(-\frac{\partial E}{\partial w_{ij}}) \tag{2.4}$$

$$= \eta \delta_j o_i$$

where:

o_i : output of i^{th} input node (which is the same as the output value)

η : step length (learning rate)

i, j, k : subscripts i, j and k correspond to input, hidden and output nodes, respectively.

w_{jk} : weight on the link from node j to k.

w_{ij} : weight on the link from unit i to j.

δ_k : $o_k(1-o_k)(t_k-o_k)$ for output nodes.

δ_j : $o_j(1-o_j)\sum_k \delta_k w_{jk}$ for hidden nodes.

In this way, the error is propagated backwards to modify weights so as to minimise the error. Steps **5** and **6** above are referred to as the **'Backward propagation phase'**.

Step 7 Go back to Step **1** until a satisfactory configuration is found.

A detailed derivation of the BP gradient calculation is given by Rumelhart and colleagues (1986).

2.3 LIMITATIONS OF THE BACK PROPAGATION TRAINING ALGORITHM

The traditional BP algorithm has proved satisfactory when applied to many training tasks, but despite many successful applications the BP algorithm has several important limitations. Since the BP algorithm uses the gradient descent method to update weights, one of the limitations of this method is that it is not guaranteed to find the global minimum of the error function (refer to Figure 2.2). The gradient-descent method may easily get trapped in a local minima especially for non-linearly separable problems (Marco Gori and Alberto Tesi, 1992) such as the XOR problem (E.K. Blum, 1989). Being trapped into a local minima is one of the reasons that may lead BP to fail in finding the global optimal solution.

The gradient-descent method is an iterative procedure for obtaining the values of parameters that minimise an objective function $E(\mathrm{w})$. Geometrically, the objective

function specifies an error surface defined over the weight space (R. P. Lippman, 1987). During each iteration or at the end of each epoch, the weight vector is iteratively changed from a randomly chosen magnitude and direction along the negative gradient of the error function in which the error function decreases most rapidly as follows:

$$w^{(n+1)} = w^{(n)} + \Delta w^{(n)} \tag{2.5}$$

where:

n : the iteration step.

$\Delta w^{(n)}$: $\eta^{(n)} d^{(n)}$

$d^{(n)}$: $-\nabla \frac{\partial E}{\partial w}$ is the search direction in which an objective (error) function $E(\mathrm{w})$ is reduced.

$\eta^{(n)}$: step length or the learning rate.

For a sufficiently small learning-rate value, the error $E(\mathrm{w})$ is expected to decrease at each successive step/pattern, eventually leading to a weight vector at which the condition (Equation 2.2) is satisfied. The selection of an optimal learning rate value is important to achieve faster convergence. An incorrect choice of the learning rate can result in a slow convergence.

Even though the gradient descent method can be an efficient method for obtaining the weight values that minimise an error measure, error surfaces frequently possess properties that make this procedure too slow to converge. There are various reasons for this slow rate of convergence. They involve the magnitude and the direction components of the gradient vector. When the error surface is fairly flat along a weight dimension, the derivative of the weight is small in magnitude. Thus, the value of the weight is adjusted by a small amount and many steps are required to achieve a significant reduction in error. Alternatively, where the error surface is highly curved along a weight dimension, the derivative of the weight is large in magnitude. Thus, the value of the weight is adjusted by a large value which may overshoot the

minimum of the error surface along that weight dimension. Another reason for the slow rate of convergence of the gradient-descent method is that the direction of the negative gradient vector may not point directly towards the minimum of the error surface.

2.4 IMPROVING THE BACK PROPAGATION TRAINING EFFICIENCY USING OPTIMISATION METHODS

The problem of improving the learning efficiency and convergence rate of the BP algorithm has been investigated by a number of researchers. Several acceleration techniques have been proposed as modifications to the original BP algorithm. The research has fallen roughly into two categories:

(a) Heuristic techniques which include variations of the learning rate, use of a momentum term, gain tuning of the activation function, and use of topology optimisation methods.

(b) Second-order optimisation techniques for minimising the error.

2.4.1 Heuristic techniques

A detailed survey of BP improvements lies outside the scope of this chapter, as the remaining sections in this chapter examine some of the most significant and popular modifications to the BP algorithm. Based on this first category, various acceleration techniques have been proposed.

(a) ***The Bold driver method.*** One of the main issues with the traditional BP algorithm is the fixed learning rate η. It is very common for neural-network researchers that in finding the optimal learning rate (i.e. the one that brings about the greatest reduction in the network error value E) is likely to vary not only from task to task, but also for different regions of a single error surface. Evidently, a strategy for adapting η as training proceeds is highly desirable. The bold driver is one of the simple and effective strategies for adapting the BP learning rate η. The strategy involves incrementally

increasing or decreasing η at each training epoch depending on whether the algorithm is making progress or not (T.P. Vogl *et al.*, 1988). Unlike a traditional fixed-learning rate BP, the bold driver method can be implemented as a strict descent algorithm which means that no increase in E is allowed during any training epoch. This is achieved simply by maintaining the network weights at the same location such as by setting $w_{n+1} = w_n$ whenever $E(w_n - \eta_n g_n) > E(w_n)$, provided that the gradient $g(w_n)$ is non-zero, it is guaranteed that the bold driver method will eventually reduce η to a small enough value to bring about a reduction in E along the negative gradient from location w_n. Vogl and colleagues (1988) proved in their paper that the bold driver method frequently outperformed both batch BPs with fixed learning rates (even when η is set to its optimal fixed value), and the gradient descent algorithm, a classical implementation of batch BP that sets the learning rate optimally at each epoch using a one-dimensional line-search procedure. As long as the optimal learning rate η does not change rapidly as training proceeds, the bold driver method will tend to set η to a near optimal value much of the time, furthermore the computational cost of adapting η is minimal as compared to the classical gradient descent method.

(b) ***BP with momentum***. Rumelhart and colleagues (1986) proposed a simple heuristic strategy for speeding up the BP training method which involves incorporating a momentum term in the generalised delta rule (Equation 2.5) as follows:

$$\Delta w^{(n)} = \eta g^{(n)} + \alpha \Delta w^{(n-1)} \tag{2.6}$$

where the user-defined parameter α is set in the range $0 \leq \alpha < 1$. Note that if the momentum effects are turned off ($\alpha = 0$) then the update rule given by Equation 2.6 is equivalent to the standard BP update of Equation 2.5. Many researchers have shown that the addition of a momentum term can significantly speed up the BP training algorithm. One potential drawback with the update in Equation 2.6 is that, if α is increased, it may be necessary to reduce the

η in order to maintain network stability in preventing excessive weight changes. To counter this problem, Widrow and Lehr (B. Widrow and M. A. Lehr, 1990) proposed a modified version of Equation 2.6 which incorporates the factor $(1-\alpha)$ as follows:

$$\Delta w^{(n)} = -(1-\alpha)\eta g^{(n)} + \alpha \Delta w^{(n-1)} \tag{2.7}$$

However, this approach has its own potential drawback, as if α is set to a comparatively large value, the weight update $\Delta w^{(n)}$ in Equation 2.7 will tend to be dominated by gradient information from previous epochs so the update in Equation 2.7 frequently proves less effective, as compared to Equation 2.6, in practice. The comparison between batch BP with momentum and a class of classical optimisation algorithm known as conjugate-gradient methods (see Chapter Three) discovered that the update rule for batch BP with a momentum term can be viewed as an approximation to the conjugate-gradient update, with the important difference that BP with momentum sets η and α to fixed heuristic values, whereas the conjugate-gradient methods automatically set η and α to near optimal values at each iteration (M.F. Moller, 1993).

(c) ***Delta-Bar-Delta Rule***. Jacobs (1988) noted from his research that if consecutive changes of a weight $w_{ij}^{(n+1)}$ and $w_{ij}^{(n)}$ possess opposite signs, then the weight value is oscillating, hence the learning rate for that weight should be decremented. Similarly, if the consecutive derivatives of a weight possess the same sign then the learning rate for that weight should be increased. Jacobs (1988) has introduced a Delta-Bar-Delta modification, which consists of a weight update rule and a learning rate update rule. The Delta-Bar-Delta algorithm controls the learning rates by observing the sign changes of an exponentially averaged gradient (Jacobs R. A., 1988). The weight update rule is similar to the original gradient-descent algorithm with the exception that each weight possesses its own learning rate parameter. The Delta-Bar-Delta rule increments the learning rates linearly, but decrements them exponentially. Incrementing linearly prevents the individual learning rates from becoming too large. Decrementing exponentially ensures that the learning rates are always positive and allows them to be decreased rapidly. The disadvantage of this method is that the designers have to determine three new parameters, and like the

conventional BP, convergence rates are slow. In addition, a large number of trial runs are required before arriving at the right choice of parameter values. Most of the researchers have pointed out that a constant learning rate is not suitable for a complex error surface. Other researchers (D. R. Hush and J. M. Salas, 1988; T.P. Vogl *et al.*, 1988) proposed learning rate adaptations, while Weir (1991) considered the problem of choosing an optimum learning-rate value and established a method for self-determination of the adaptive learning-rate value for every epoch. Most of these techniques can be considered as the variations of line search methods.

(d) ***Starting with appropriate weights***. It has been shown that the BP method is sensitive to initial weights (J.F. Kolen and J.B. Pollack, 1991). Weights are usually initialised with small random values. However, starting with incorrect weight values is one reason for getting trapped in local minima or leading to a slow learning progress. For example, initial weight values which are too large can cause **'Premature Saturation (PS)'** (B.W. Lee and B.J. Sheu, 1993). The learning progress can be accelerated by initialising weights in such a way that all hidden units are scattered uniformly in the input pattern space (D. Nguyen and B. Widrow, 1990).

(e) ***Improving the error function***. Since the sigmoid derivative which appears in the error function of the original BP method has a bell shape, it sometimes causes slow learning progress when output of a unit is near '0' or '1'. To remove it from the error signal, van Ooyen and Nienhuis (1992) and Krzyzak and colleagues (1990) have employed an entropy-like error function by making the error gradients for poorly classified patterns of significantly higher values than calculated mean squared error (MSE) functions. Therefore adjustment allows the network to progress in flat regions of the weights space. Oh (1997) proposed a modified error function by reducing the probability that output nodes take the extreme values of sigmoid function. Chandra and Singh (2004) proposed a new activation function for sigmoidal feed forward neural network training. Lee and colleagues (1999) analysed the cause of **'Premature Saturation (PS)'** in output layer, which is caused by the use of the gradient-descent method. PS will greatly slow down the learning speed of the BP algorithm. They proposed an Error Saturation Prevention (ESP) function to prevent the nodes in the output layer from the PS condition. They also applied to the hidden nodes in hidden layers to adjust the learning term. Again Oh and Lee (1995) proposed another

improved error function of the error back propagation (EBP) algorithm for MLPs by allowing the output nodes of the MLP to generate an appropriate error signal according to the situation of the output nodes. When some output nodes of MLP are incorrectly saturated, the strong error signal of the output nodes updates the associated weights so that they can escape the incorrectly saturated state. This can accelerate the learning speed.

(f) ***Improving activation function***. One of the main reasons for the slow convergence of BP algorithms is the derivative of the activation function that leads to the occurrence of PS of the network output units. When the actual output o_{pk} (where o_{pk} is the actual output of the k^{th} output neuron for the $p-th$ pattern) is approaching either extreme values of the sigmoidal function, that is either 0 or 1, the derivative of the activation function having the factor $o_{pk}(1-o_{pk})$ will become extremely small, and the BP error signal may vanish. This will lead the algorithm to be trapped into a 'flat spot'. Consequently the learning process and weight adjustment of the algorithm will be very slow or even suppressed. That is why BP usually requires tens to a thousand iterations to leave the flat spot, and causing the slow convergence of the algorithm. Ng and colleagues (2003) proposed a modification to the derivative of the activation function so as to improve the convergence of the learning process by preventing the error signal dropping to a very small value by magnifying the derivative term $o_{pk}(1-o_{pk})$, especially when the value of o_{pk} approaches 0 or 1, by using the power factor so that the derivative of the activation function will not be too small and improve the convergence of the algorithm. Chandra and Singh (2004) proposed an algorithm that adapts the activation function itself. The choice of the final activation function is done dependent on the data set used for training and the initial weight condition. The results demonstrated that the proposed algorithm could be an order of magnitude faster than the BP algorithm.

(g) ***Adjusting the steepness of the sigmoid function.*** As Hush and colleagues (1992) have pointed out, because of the sigmoid's non-linearity, error surfaces tend to have many flat areas as well as steep regions. If one such flat area with a high error is

encountered, no significant decrease in the error occurs for some period, after which the error decreases again then this will lead to PS condition. Since considerable time is often needed to traverse such an area, PS retards the learning process. The basic remedy is to adjust the sigmoid's steepness (A. Rezgui and N. Tepedelenlioglu, 1990; K. Yamada *et al.*, 1989). The adaptation of gain parameters of the activation functions has been shown to prevent the network from becoming trapped in a local minimum caused by the neuron saturation in the hidden layer (X.G. Wang *et al.*, 2004). The gain term controls the steepness of the activation function. It has been shown recently that a BP algorithm using a gain variation term in an activation function converges faster than the standard BP algorithm as will be discussed further in the next section.

2.4.2 Second-order optimisation methods

The second category of research in improving the training efficiency of BP algorithms has focused on the use of the second-order method. Several researchers have proposed the use of second-order gradient techniques such as conjugate gradient and Quasi-Newton (QN) methods, instead of the simple gradient-descent technique. For instance, Fahlman (1988) claimed that a set of first-order partial derivatives collected at a single point only tell very little about how large a step one can safely take in weight space. But if something about higher order derivatives (the curvature of the error function) is known, one can presumably achieve better performance. The momentum term and Delta-Bar-Delta techniques which were discussed in Section 2.4.1 are an ad-hoc variation of this strategy. Other approaches make explicit use of the second derivative of the error with respect to each weight. Fahlman developed 'quickprop', a variation of the BP with momentum that utilises both the second-order method, which is loosely based on Newton's method, as well as other heuristic methods, for the purpose of improving the convergence rate of the original BP algorithm.

The use of second derivatives has been proposed to increase the convergence speed in several works (R. Battiti., 1992; W.L. Buntine and A.S. Weigend, 1993). It has been demonstrated (Y. LeCun *et al.*, 1991) that these methods are more efficient, in terms of learning speed, than the methods based only on the gradient-descent technique. In fact, second-order methods are among the fastest learning algorithms. Some of the

most relevant examples of this type of methods are the Quasi-Newton (QN), Levenberg-Marquardt (LM) (D.W. Marquardt, 1963; K. Levenberg, 1944; M.T. Hagan and M. Menhaj, 1994), and the conjugate-gradient algorithms (E.M.L. Beale., 1972). The Quasi-Newton methods use a local quadratic approximation of the error function, like Newton's method, but they employ an approximation of the inverse of the Hessian matrix to update the weights, thus getting the lowest computational cost. The two most common updating procedures are the Davidon-Fletcher-Powell (DFP) and Broyden-Fletcher-Goldfarb-Shanno (BFGS) (J. E. Dennis and R. B. Schnabel, 1983). The Levenberg-Marquardt method combines, in the same weight updating rule, both the gradient and the Gauss-Newton approximation of the Hessian of the error function. The influence of each term is determined by an adaptive parameter, which is automatically updated. Regarding the conjugate-gradient methods, they use at each iteration of the algorithm, different search directions in a way that the component of the gradient is parallel to the previous search direction. Several algorithms based on conjugate directions were proposed such as the Fletcher-Reeves (Adrian J. Sheperd, 1997; R. Fletcher and C.M. Reeves, 1964), Polak-Ribiere (C.M. Bishop, 1995; R. Fletcher and C.M. Reeves, 1964), Powell-Beale (M.J.D. Powell, 1977) and scaled conjugate-gradient algorithms (M.F. Moller, 1993).

2.5 SUPERVISED LEARNING USING ADAPTIVE GAIN VARIATION

Among various attempts to enhance the learning efficiency of BP algorithms (gradient-descent method) that have been mentioned in Section 2.4.1, those using the gain value are among the easiest to implement. The gain value controls the steepness of the activation function. As shown in Equation 2.8, for a j^{th} node, the weighted sum of inputs is passed through a sigmoid activation function to generate the nodal output as follows:

$$o_j = \frac{1}{(1+e^{-c_j a_{net,j}})} \tag{2.8}$$

and $a_{net,j} = \left[\sum_{i=1}^{I} w_{ij} o_i\right] + \theta_j$ where:

o_i : output signal from the i^{th} unit.

w_{ij} : weight of the link from unit i to unit j.

o_j : output signal of the j^{th} unit.

$a_{net,j}$: net input activation function for the j^{th} unit.

θ_j : bias for the j^{th} unit.

c_j : gain describing the slope of the activation function for the j^{th} unit.

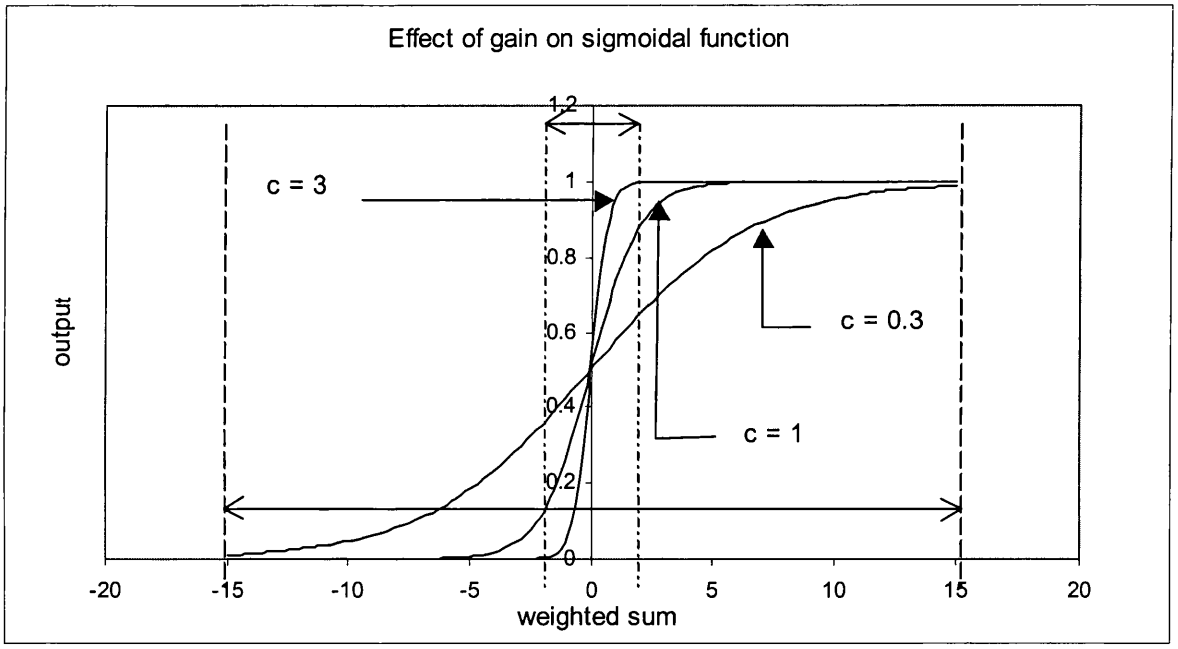


Figure 2.3: Sigmoid-activation function with different slopes.

The value of the gain parameter, c, directly influences the slope of the activation function. For large gain values ($c_j >> 1$), the activation function approaches a '*step function*' whereas for small gain values ($0 < c_j << 1$), the output values change from zero to unity over a large range of the weighted sum of the input values and the sigmoid function approximates a *'linear function'* as shown in Figure 2.3.

It has been recently shown that a BP algorithm using adaptive gain variation in an activation function converges faster than the standard BP algorithm. The early research on adapting the gain value was conducted by Kruschke and Movelland (1991). They explored the benefits of adaptive gains in BP networks and showed that gradient descent with respect to gain greatly increases learning speed, and concluded that adaptive gain only has a catalytic effect in the learning process by modifying the magnitude, not the direction, of the weight change. However it was found out that the algorithms that employ the gain parameter suffered from increased instability, and frequently fail to converge within a finite time because of an inappropriate choice for the initial weights. Tai-Hoon Cho and colleagues (1991), then proposed an automatic weight reinitialisation solution with a larger initial gain value (around 2 or 3) on BP algorithms to converge much faster and are more stable.

Later, Thimm and colleagues (1996) had proved that changing the gain value of the activation function is equivalent to changing the learning rate and the weights and claimed that the idea simplified the BP learning rule by eliminating one of its parameters. In order to support the argument, Figure 2.4 shows a summary of network performance when the learning rate was fixed with a constant value and at the same time varying the gain value. The horizontal axis shows a variety of gain values that were used to control the steepness of the sigmoid function. For each gain value, the maximum generalisation accuracy is plotted (using the vertical scale on the right), along with the number of training epochs required to reach that level of accuracy (using the vertical scale on the left). As can be seen from Figure 2.4, the generalisation accuracy gets better as the gain value gets bigger, and the training time also improves dramatically. At a gain value of about 0.005, the accuracy is dramatically decreased and training time (in epochs) is at its maximum value. It is often possible to get slightly higher accuracy by using an even bigger gain value than the 'fastest' one, as is the case here, where accuracy is improved by using a gain value of 2, at the expense of increased training time. Beyond that, however, smaller gain value requires linearly more training time with no significant improvement in accuracy.

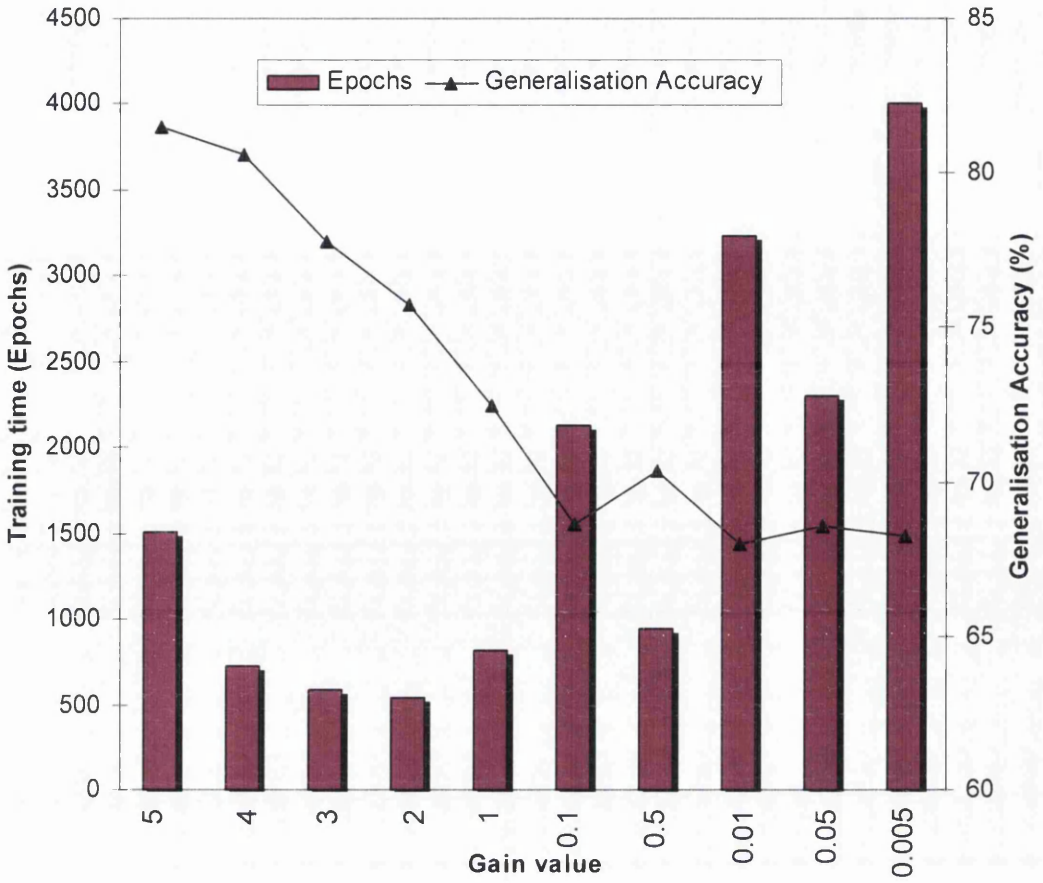


Figure 2.4: Training time (in epochs) and maximum hold-out set phoneme generalisation accuracy for each gain value. The bars indicate the number of epochs needed to reach the maximum generalisation accuracy, and the line indicates what the maximum accuracy was for each gain value.

Research also showed that the BP learning algorithm with adaptive gain value and combined with a dynamic learning rate optimisation method can achieve the goal of fast convergence (Murphy Hot and Hiroaki Kurokawa, 1998). As can be seen from Figure 2.5 that for the tests carried out, the behaviour of the networks is fairly controlled when learning rate values of 0.1 and 0.5 were used. The results obtained when a learning rate value of 0.1 was used were clearly better than those obtained using the smaller learning rate values. This is mainly due to the increased number of iterations required in reaching the target error. There was a large variation in the results obtained when a learning rate of 0.05 was used, indicating that the oscillations in the MSE prediction error were large.

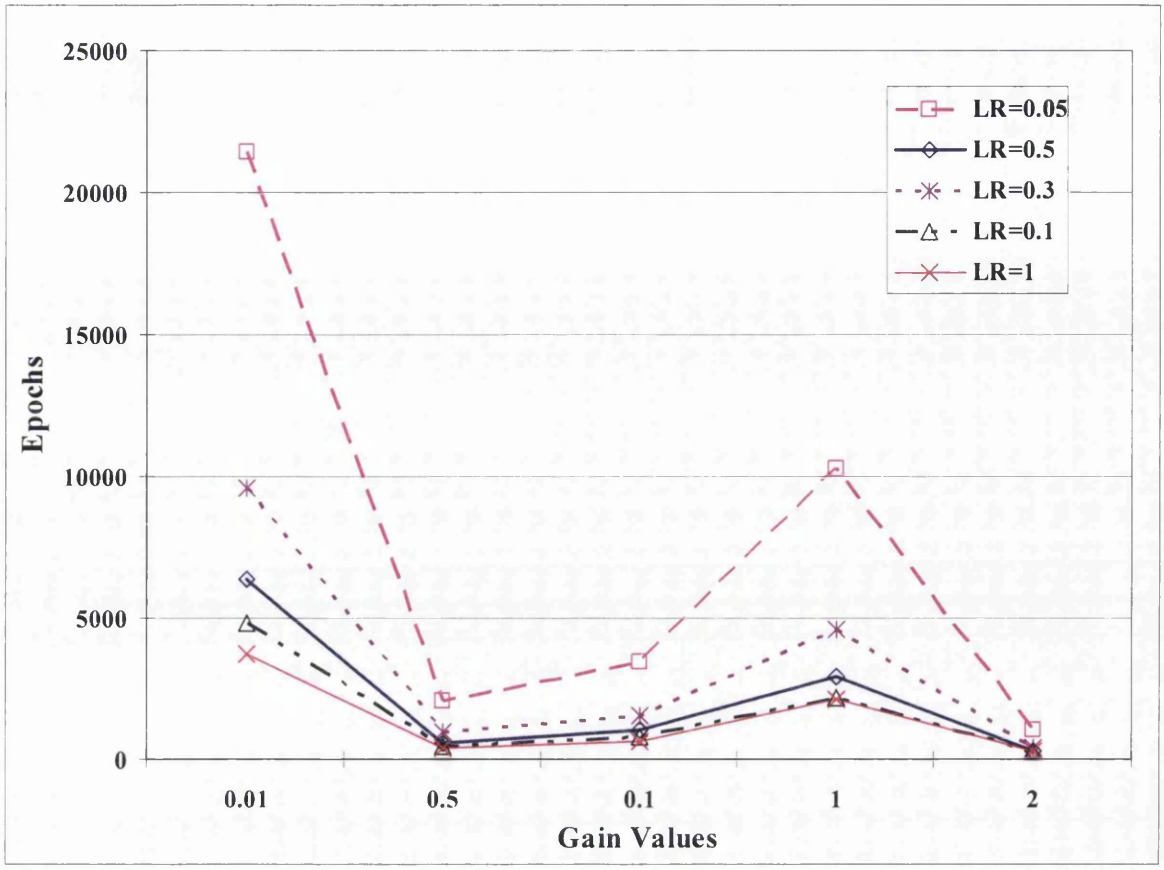


Figure 2.5: Best results obtained when various gain values were used at a varied learning rate.

Danilo and colleagues (1999) provided a good analysis on the relationship between the learning rate η and the gain value c in the hyperbolic tangent activation function for a feed forward NN, trained by BP and it showed that such relationships reduced the number of parameters in the non-linear optimisation task. Again, Eom and Jung (2003) proposed a method using fuzzy logic for automatically tuning the gain parameter of the activation function and demonstrated that changing the gain of the activation function is equivalent to changing the learning rate, the weights and the biases. Kandil and colleagues (2005) improved further the use of variable gain of the log-sigmoid function by optimising the gain value by using the Stretched Particle Swarm Optimisation (SPSO) technique. The proposed algorithm is trained and compared with the popular training (Levenberg-Marquardt back propagation) method, on application examples and showed an increase in the speed of convergence for the training and learning phase.

It has been shown in the above literature that the variation in the gain value does improve the BP learning efficiency. It can be concluded from this literature review that changing the gain value is equivalent to changing the learning-rate value. Those claimed had simplified the BP learning rule by eliminating one of its parameters (i.e. gain) and as a result, it appears that the researchers had begun to lose their interest in studying the effect of gain in improving training efficiency.

Until in 2002, when Ransing (2002) discovered that for the BP algorithm gain is not equivalent to change in the learning rate value, but its effect is like having a global learning rate value and an additional localised learning rate contribution for every node. A coupled algorithm that changed the gain value adaptively for each node was presented in this work.

The next section will discuss in detail the previous research carried out by Ransing (2002) to illustrate the effect of adaptive gain variation in increasing the training efficiency of the gradient-descent method.

2.6 AN INNOVATIVE METHOD TO ENHANCE BACK PROPAGATION TRAINING ALGORITHM BY RANSING (Meghana R. Ransing, 2002)

The analysis provided by Ransing (2002) studying the effect of adaptive gain variation on BP network training showed that adaptive gain variation has a significant impact on gradient-descent training speed. For the first time, Ransing had proposed a new method that modified the standard BP by coupling the weight, bias and gain update expressions in the standard BP algorithm (refer to Equations 2.11 and 2.12). The algorithm had been proposed for sequential as well as batch training. The weight and gain update expressions for output as well as the hidden nodes are shown below.

The weight update expression for the links connecting to output nodes is:

$$\Delta w_{jk} = \eta(t_k - o_k)o_k(1-o_k)c_k o_j \tag{2.9}$$

The gain update expression for the output node is:

$$\Delta c_k(n+1) = \eta(t_k - o_k)o_k(1-o_k)(\sum w_{jk}o_j) \tag{2.10}$$

The weight update expression for the links connecting to hidden nodes is:

$$\Delta w_{ij} = \eta\left[\sum_k c_k w_{jk} o_k(1-o_k)(t_k - o_k)\right] c_j o_j(1-o_j) \tag{2.11}$$

Similarly, the gain update expression for the links connecting hidden nodes is:

$$\Delta c_j(n+1) = \eta\left[-\sum_k c_k w_{jk} o_k(1-o_k)(t_k - o_k)\right] o_j(1-o_j)\left[\sum_j w_{ij} o_i\right] \tag{2.12}$$

It is evident from the literature that coupling the expressions for updating weight and gain (Equations 2.11 and 2.12), for sequential as well as batch training, is an innovative and original approach. The following iterative and coupled algorithm had been proposed by Ransing (2002) for batch training. Weight, bias and gain values were calculated and updated after the presentation of all the training example pairs as shown in Table 2.1 were presented to the network. The current algorithm used the following terms.

For a given epoch:

Update the weight and bias values after the presentation of the entire example set using the previously converged gain value. (1)

Use the weight and bias values calculated in Step 1 to calculate the new gain value. (2)

Repeat Steps 1 and 2 by using the gain value calculated in Step 2 in Step 1 until the difference in consecutive weight, bias and gain values becomes less than the predefined value.

The speed of convergence achieved using the proposed current method was demonstrated by using the sine curve problem. Consider a single input-output layer network with one hidden layer having five hidden nodes. The training data set was created by using the two following cases:

(a) Case 1: the training datasets was created by using the function: $y = \sin(pi * x)\ where\ \ x \in [0,1]$.

(b) Case 2: the same training datasets as created in Case 1 by using the function: $y = \sin(pi * x)\ where\ \ x \in [0,1]$ but with by adding an approximate twenty per cent random Gaussian noise.

2.6.1 Case 1: The sin(x) problem without noise

The first training dataset was created by using the function: $y = \sin(pi * x)\ where\ \ x \in [0,1]$. The training required the network to approximate the function for a sample of fifty-two input points chosen uniformly as illustrated in Figure 2.6 (circles). These data points were also shown as tabulated in Table 2.1. Before training the dataset was further divided into a training set of examples and a validation set of examples. Only the training set of examples was used to adjust the network weights until the stopping criteria was satisfied. During each epoch, the gain, weight and bias values for all hidden nodes and output nodes converged to achieve an MSE of 0.001 for gain, weight and bias values, respectively. The terminated number of epochs in reaching the target error is shown in Figure 2.8.

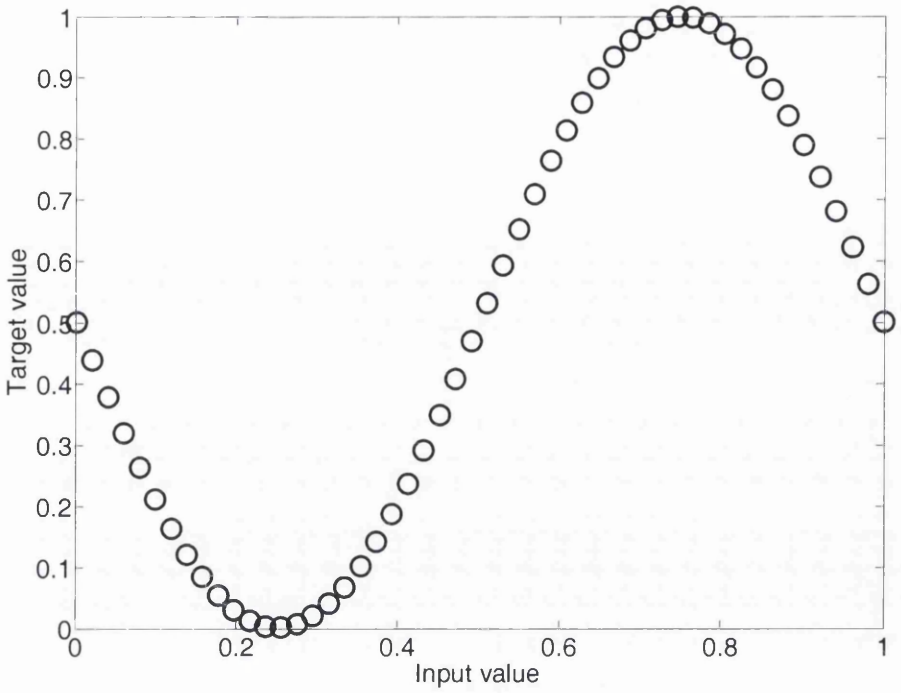


Figure 2.6: The sine curve data points used in the training data set as tabulated in Table 2.1(data points taken from Ransing (2002)).

The error versus number of epochs required to achieve the target error by the method proposed by Ransing (2002) is plotted in Figure 2.8 (red solid curve). The dotted curve represents the same graph for the network trained using a constant unit gain. It can be seen from Figure 2.8 that the method proposed by Ransing with adaptive gain has consistently outperformed the standard gradient-descent method. The current method took 6,017 epochs to learn the target function whereas the standard gradient-descent method took 14,098 epochs which is almost twice bigger.

The gain values for the five hidden nodes at the end of training are 1.0004, 1.0322, 1.0727, 1.1107 and 1.3493, respectively. The gain value for the output node at the end of the training is 1.3602. The speed of convergence for the current proposed method was high because the modified gain values are greater than unity.

Training data		
No.	**Input**	**Target**
1	0.002	0.501
2	0.021	0.439
3	0.041	0.379
4	0.060	0.320
5	0.080	0.264
6	0.099	0.212
7	0.119	0.164
8	0.139	0.121
9	0.158	0.084
10	0.178	0.054
11	0.197	0.030
12	0.217	0.013
13	0.236	0.003
14	0.256	0.002
15	0.276	0.007
16	0.295	0.020
17	0.315	0.041
18	0.334	0.068
19	0.354	0.102
20	0.374	0.142
21	0.393	0.188
22	0.413	0.238
23	0.432	0.292
24	0.452	0.349
25	0.471	0.409
26	0.491	0.470
27	0.511	0.532
28	0.530	0.593
29	0.550	0.652
30	0.569	0.710
31	0.589	0.764
32	0.608	0.814
33	0.628	0.859
34	0.648	0.899
35	0.667	0.933
36	0.687	0.961
37	0.706	0.981
38	0.726	0.994
39	0.745	1.000
40	0.765	0.998
41	0.785	0.989
42	0.804	0.972
43	0.824	0.948
44	0.843	0.917
45	0.863	0.880
46	0.883	0.837
47	0.902	0.789
48	0.922	0.737
49	0.941	0.681
50	0.961	0.623
51	0.980	0.562
52	1.000	0.501

Table 2.1: The training data set used for Case 1 (data points taken from Ransing (2002)).

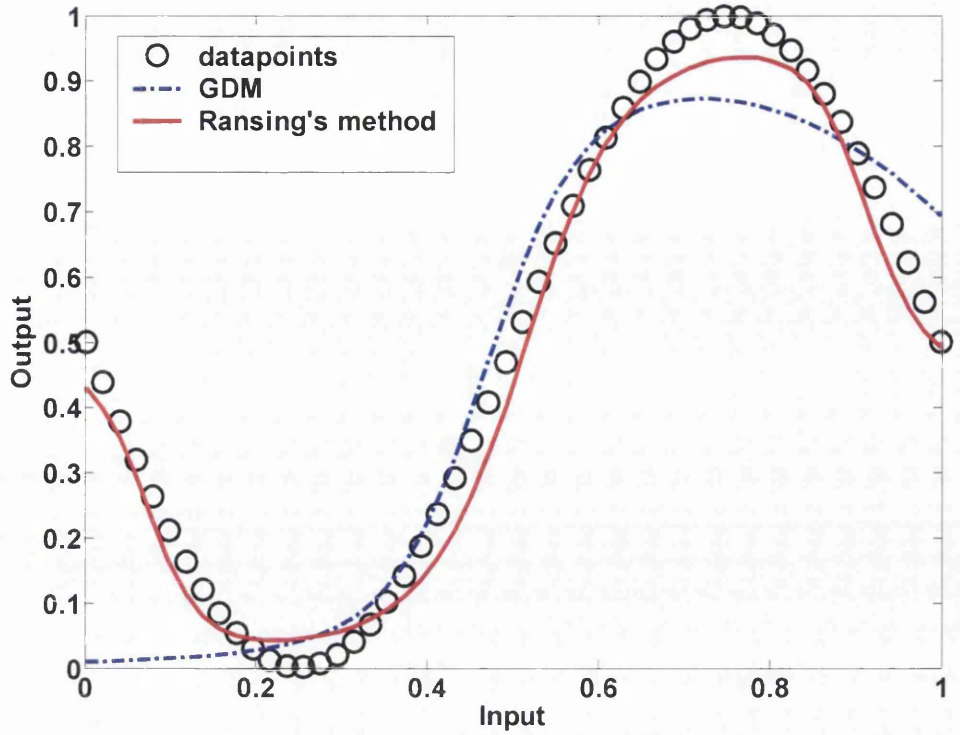


Figure 2.7: Output of neural network trained to learn a sine curve using the current proposed method (source adopted from Ransing, 2002).

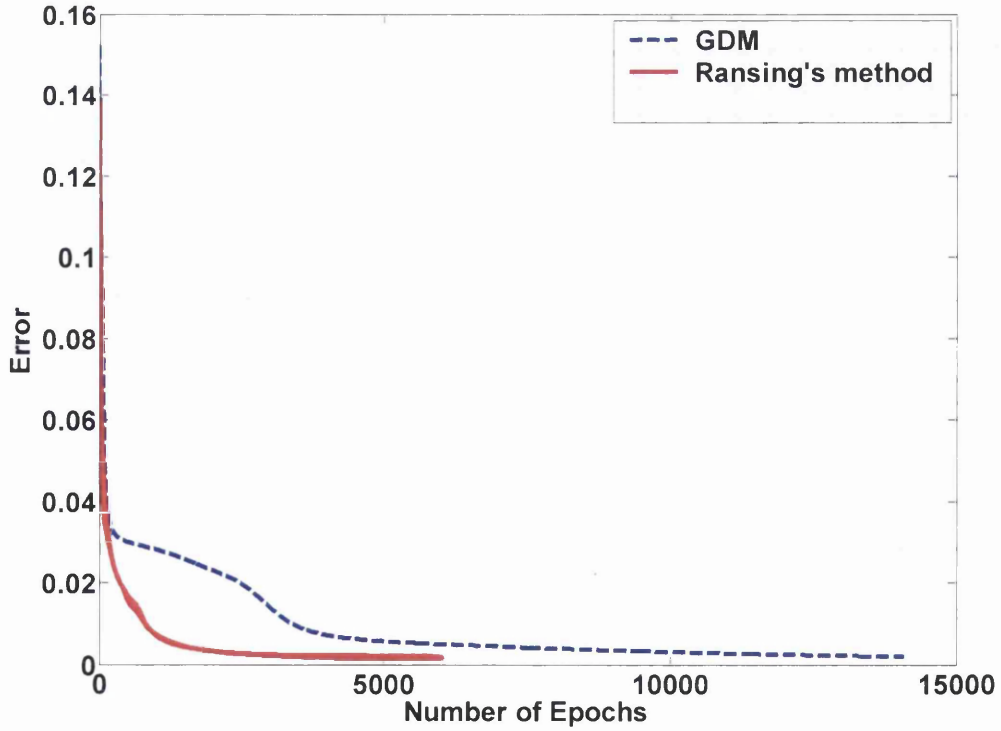


Figure 2.8: Error versus number of epochs required to achieve the target error of 0.001(source adopted from Ransing (2002)).

2.6.2 Case 2: The sin(x) problem with twenty per cent random Gaussian noise

The second training dataset was created by using the same data created from the function $y = \sin(pi * x)$ *where* $x \in [0,1]$ but the data points were further added with an approximate twenty per cent random Gaussian noise as illustrated in Figure 2.9 (circles). These data points were also put into the table as shown in Table 2.2. The training data was also further divided into a training set of examples used to adjust the network weights and a validation set of examples used to estimate network performance during training as required by the stopping criteria.

The network was trained using a constant learning rate value of 0.3 to achieve a target error value within one per cent, using the standard gradient descent training algorithm in a batch mode with coupled and adaptive changes in weight, bias and gain values.

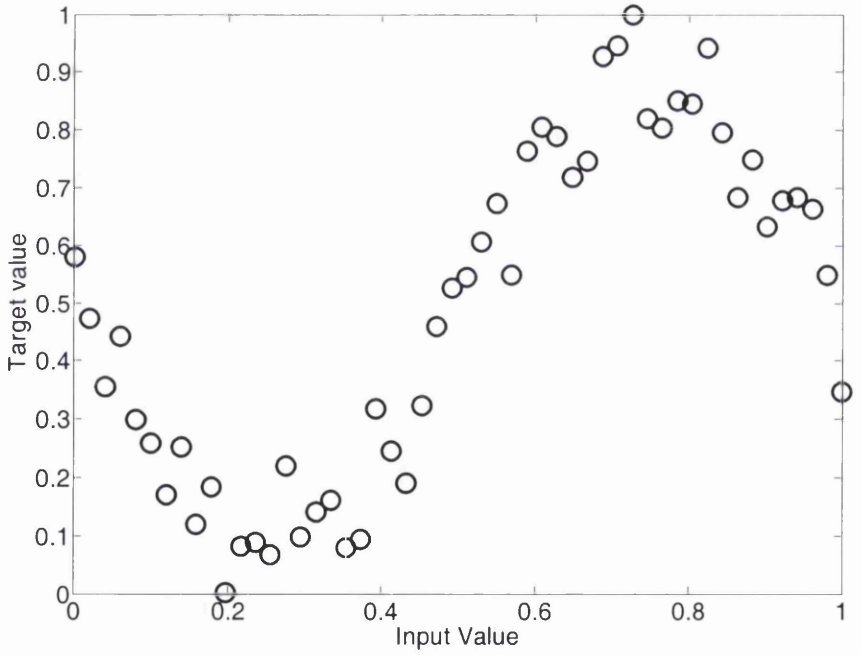


Figure 2.9: The sine curve data points with twenty per cent random Gaussian noise as tabulated in Table 2.2 (data points taken from Ransing (2002)).

The error value versus number of epochs required to achieve the target error was plotted in Figure 2.11 (solid red curve). The dashed curve represents the same graph for the network trained using a constant unit gain value. The results clearly showed that the current method proposed by Ransing (2002) outperformed the standard algorithm with constant gain by only taking 905 epochs as compared to 3,296 epochs in learning the target function. The results showed that the speed of convergence of the current method was improved as compared to the standard gradient-descent method because the modified gain values were greater than unity. The gain values for the five hidden nodes at the end of training were 2.0, 1.0511, 1.0692, 1.0551 and 1.0320. The gain value for the output nodes at the end of training was 1.9858.

Training data		
No.	**Input**	**Target**
1	0.002	0.579
2	0.021	0.473
3	0.041	0.354
4	0.060	0.442
5	0.080	0.298
6	0.099	0.259
7	0.119	0.171
8	0.139	0.253
9	0.158	0.120
10	0.178	0.183
11	0.197	0.002
12	0.217	0.082
13	0.236	0.090
14	0.256	0.068
15	0.276	0.220
16	0.295	0.098
17	0.315	0.141
18	0.334	0.161
19	0.354	0.080
20	0.374	0.095
21	0.393	0.317
22	0.413	0.245
23	0.432	0.191
24	0.452	0.323
25	0.471	0.459
26	0.491	0.526
27	0.511	0.544
28	0.530	0.606
29	0.550	0.673
30	0.569	0.549
31	0.589	0.764
32	0.608	0.805
33	0.628	0.789
34	0.648	0.719
35	0.667	0.746
36	0.687	0.928
37	0.706	0.945
38	0.726	1.000
39	0.745	0.820
40	0.765	0.804
41	0.785	0.851
42	0.804	0.846
43	0.824	0.941
44	0.843	0.796
45	0.863	0.684
46	0.883	0.749
47	0.902	0.633
48	0.922	0.679
49	0.941	0.683
50	0.961	0.663
51	0.980	0.549
52	1.000	0.348

Table 2.2: The training data set used for Case 2 (data points taken from Ransing (2002))

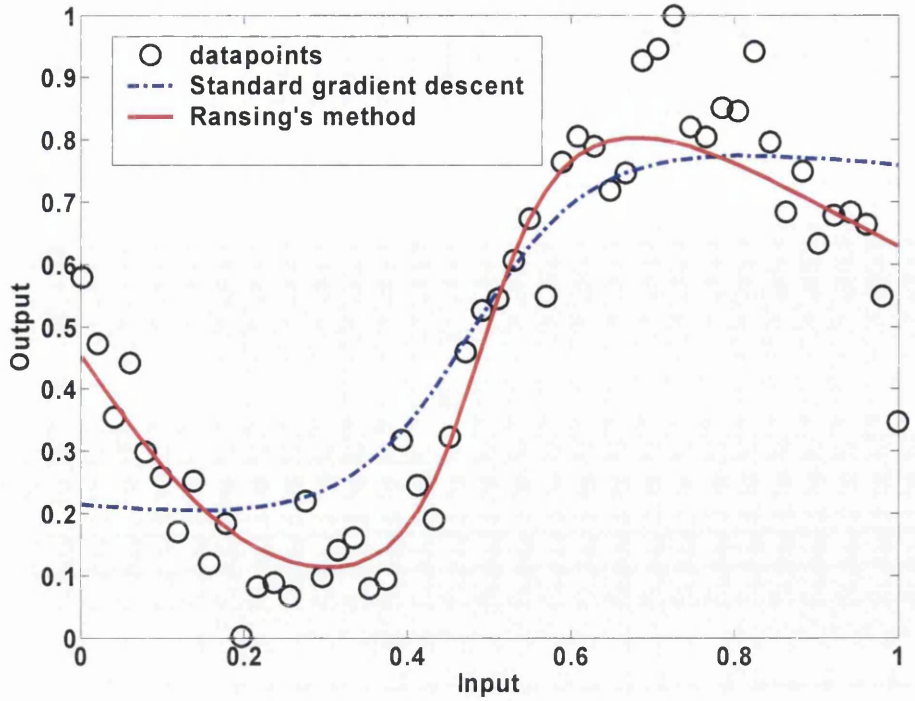


Figure 2.10: Output of neural network trained to learn a sine curve with twenty per cent random Gaussian noise in batch mode using the coupled algorithm.

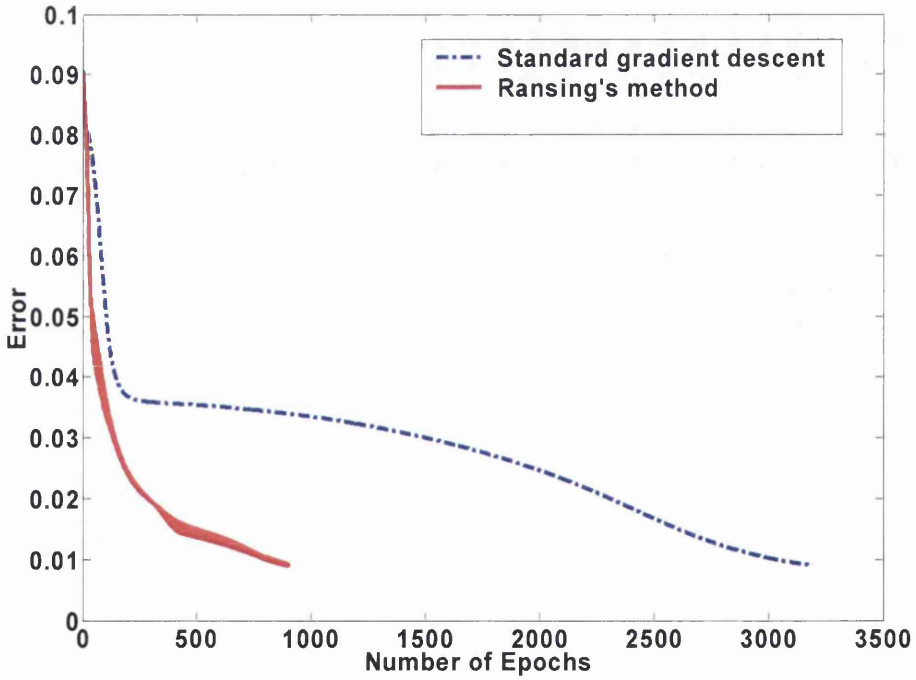


Figure 2.11: Error versus number of epochs required to achieve the target error value of 0.01.

2.6.3 Advantages of the current method

The results obtained from the case studies clearly showed that the method proposed by Ransing (2002) substantially improved the learning speed and some of the contributions of the current method were summarised as follow:

- Ransing (2002) showed in her work that it is easy to introduce an adaptive gain value into a gradient-descent method as agreed by previous researchers but not for other optimisation methods.
- Previous researchers eliminated the gain variation value or they used a constant gain value in the training process since they claimed that varying the gain value was equivalent to changing the learning rate value. Whereas, the current method discovered that gain variation contributed like an adaptive learning rate for individual nodes.
- An innovative algorithm that coupled the weight, bias and gain update expressions was proposed and the algorithm demonstrated significantly improvement in sequential as well as batch-learning modes.

Even though, the analysis results, as shown in Section 2.6, demonstrated that the current proposed method significantly increased the learning speed and outperformed the standard algorithm with constant gain in learning the target function. However, the next section identifies some limitations of the current method.

2.6.4 Limitations of the current method

There were two major limitations that have been identified in the current method:

- During training, only weights, bias and gain update expressions were coupled and update adaptively. Whereas, the learning-rate value was kept constant until the end of training.
- The current method had significantly increased the training speed by improving the learning-rate value. However, the implementation of the current method was restricted only for the gradient-descent method.

2.7 CONCLUSION

While the back propagation (BP) algorithm is used widely in the majority of practical neural-network applications and performed relatively well, the algorithm still suffers from several problems such as in sensitivity to initial conditions and slow convergence. Consequently, in the past few years, a number of research studies have been attempted to improve and overcome problems associated with BP algorithms. This chapter reviewed some of the major improvements and contributions suggested by various researchers to overcome those limitations. Two major directions for improvements have been identified which are: (a) firstly, the use of heuristic techniques such ideas as variation of the learning-rate value, use of the momentum term, gain tuning of activation function; and (b) secondly the integration of (a) with second-order optimisation techniques for minimising the error value.

It has been shown in the literature that the variation of the gain parameter does improve the learning efficiency. The relevant literature on adaptive-gain variation has also been reviewed. It can be concluded that early researchers claimed that the adaptive-gain variation improved the learning rate. The work done by Ransing (2002) concluded that the adaptive-learning rate can be achieved for each neural network node by varying the gain value for each node. A coupled algorithm for the efficient calculation of the adaptive-gain value was proposed. A detailed description and the performance of the current proposed method are given. The advantages and disadvantages posed by the current proposed method have been identified.

The next chapter will take this discussion forward and explain how improvements have been implemented in the method proposed by Ransing (2002) in order to overcome some restrictions posed by the method.

CHAPTER 3

ENHANCED LEARNING ALGORITHM FOR BACK PROPAGATION NETWORK

CHAPTER LAYOUT

A novel approach for improving the training efficiency of BP neural networks algorithms is presented in this chapter. This chapter carries on with the investigation on improvements to the BP algorithm proposed by earlier researchers, particularly the work presented by Ransing (2002), on using the adaptive-gain variation in improving the training efficiency (Section 2.6 of Chapter Two). It was discovered during this work that the training efficiency of the gradient-descent formulation with adaptive gain was not improved as a result of having an adaptive-learning rate for each node as proposed by Ransing (2002). It was, however, because the adaptive-gain change improved the search direction. This chapter introduces a novel technique of integrating the adaptive-learning rate method coupled with an improved search direction for improving the computational efficiency of neural networks. As a result of this new understanding, it was possible for the first time to implement the proposed technique into various second-order optimisation methods. The mathematical formulation is described in the following sections. The penultimate section of this chapter then tests the software for its correctness on the data generated using a simple sine curve. The conclusions are drawn from the research presented in this chapter to facilitate further testing and validation that is described in Chapter Four.

3.1 INTRODUCTION

The work proposed by Ransing (2002) assumed that the learning efficiency was improved because the gain variation contributed like an adaptive-learning rate for individual nodes. The supervised training procedure performed search through a weight space in a succession of steps as follows:

$$w^{(n+1)} = w^{(n)} + \Delta w^{(n)} \tag{3.1}$$

where:

$\Delta w^{(n)} = -\eta^{(n)} d^{(n)}$

n : is the iteration step

$\eta^{(n)}$: learning rate value

$d^{(n)}$: search direction.

At each (n) step, $\Delta w^{(n)}$ is chosen to reduce an objective (error) function $E(w)$. A widely accepted choice for error function $E(w)$ is the mean-of-squares error (Equation 2.1 in Chapter Two). The efficiency of training algorithms is determined by the way in which the learning rate and the search direction is calculated (C. de Groot and D. Würtz, 1994). Equation 3.1 describes the procedure for updating the weight vector using the learning-rate value (η) and the search direction ($d^{(n)}$). The influence of gain as claimed by previous researchers is shown diagrammatically in Figure 3.1.

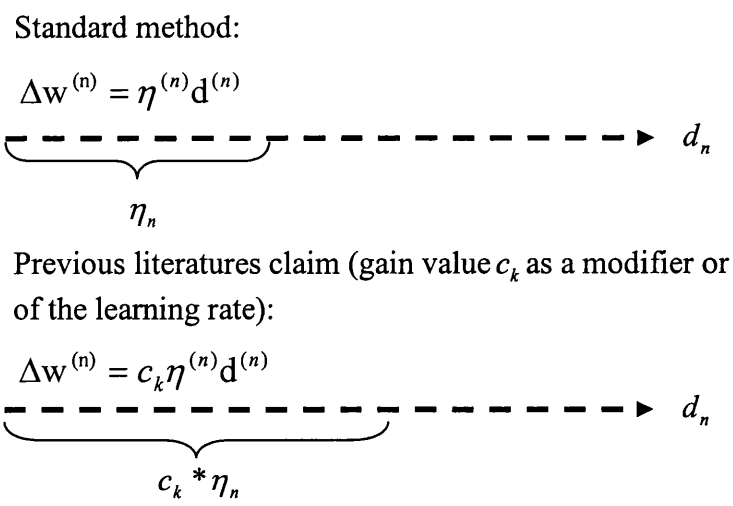


Figure 3.1: The effect of gain variation on gradient-descent method claimed by previous researchers.

In this chapter, however, it was discovered that the gain variation actually improves the search direction ($d^{(n)}$) rather than the learning rate ($\eta^{(n)}$). The next section describes this new understanding in detail and compares its findings with the results calculated by Ransing (2002).

The MATLAB implementation of the newly proposed method is then tested for its correctness. This will be illustrated further by comparing the simulation results of the proposed method with the previous version on a non-linear mapping function given by a function $y = \sin(pi * x)$ *where* $x \in [0,1]$. The mathematical formulation of implementing the proposed method into various second-order methods is described in Section 3.3. The correctness of the mathematical formulation is tested again on the data generated using the sine curve in Section 3.4. The chapter is concluded in Section 3.5.

3.2 THE PROPOSED METHOD BY IMPROVING THE CURRENT METHOD

In this section a significant improvement on the method introduced by Ransing (2002) on improving the training efficiency of BP neural-network algorithms is presented. The proposed algorithm uses the same gain-update expressions for output as well as hidden nodes as derived in Section 2.6 and for more detailed calculation please refer to Ransing's work.

The significant difference between the proposed method with that proposed by Ransing (2002) is in the understanding of how adaptive-gain variation improves the training efficiency.

Ransing's method assumed that the network had two types of learning rates: (i) the global-learning rate: and (ii) the local-learning rate for each node. At step (n) Ransing's method used a constant value for the global-learning rate and calculated the local-learning rate with respect to gain variation for each node by using Equation 3.2:

$$\Delta w_{ij}^{(n)} = -\eta^{(n)}\left[\eta_{\substack{for\\each\\node}}(c_j^L(n))\right]\frac{\partial E}{\partial w_{ij}^{(n)}} \text{ (Meghana R. Ransing, 2002)} \qquad (3.2)$$

As shown in Figure 3.2 it is proposed in this chapter that the adaptive-gain variation altered the initial search direction ($\mathrm{d}^{(n)}$) and not the learning rate. This clarity in understanding allowed a coupling of this theory with existing methods of using adaptive-learning rate values. It will also be shown in Section 3.3 that this modification of initial search direction ($\mathrm{d}^{(n)}$) can be implemented in various second-order optimisation methods to yield significant improvements in the computational speed.

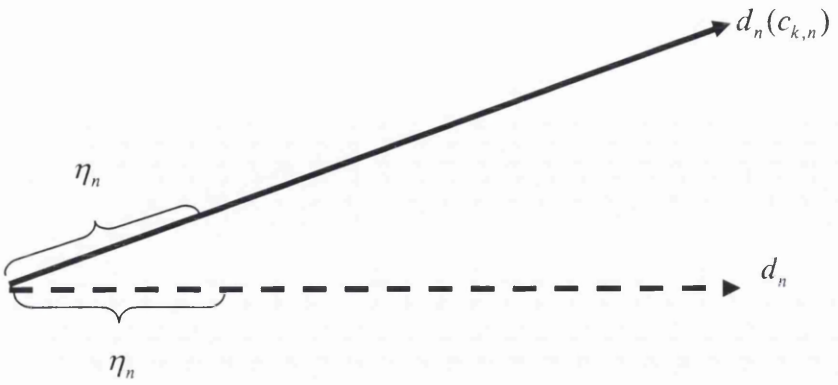


Figure 3.2: The real effect of gain variation on the gradient-descent method in improving search direction.

With the newly proposed method, the learning rule is determined by calculating the function of gain into the gradient of error with respect to weight and gain as shown in Equation 3.3:

$$\Delta w_{ij}^{(n)} = -\eta^{(n)} \frac{\partial E}{\partial w_{ij}^{(n)}}(c_j^L(n)) \tag{3.3}$$

In this research a widely used Golden Section line search method (Curtis F. Gerald and Patrick O. Wheatley, 2004) is chosen to obtain the optimised learning rate ($\eta^{(n)}$).

The complete comparison between the proposed algorithm and the current method on the BP algorithm is shown in Table 3.1. The flowchart illustrating the steps involved in the proposed algorithm is shown in Figure 3.3.

The following iterative algorithm is proposed for changing the gradient-based search direction by adaptively changing the gain value. The learning rate is adaptively changed by using the Golden Section method.

Step 1 *Initialise the weight vector with random values and the vector of gain values with unit values.*

Step 2 *Calculate the gradient of error with respect to the weights and gradient or error with respects to gain.*

Step 3 *Adaptively calculate the learning rate by using the Golden Section search method.*

Step 4 *Use the gradient weight vector and gradient of gain calculated in Step **2** to calculate the new weight vector and vector of new gain values for use in the next epoch.*

Step 5 *Repeat the following Steps **2, 3** and **4** on an epoch-by-epoch basis until the given error minimisation criteria are satisfied.*

By using the proposed method, the gradient is re-evaluated optimally at each step to produce a better choice for the search direction (d^n). The proposed method modifies the gradient-search direction (d^n) adaptively with the gain parameter for each node. Furthermore, the learning rate (η) is also determined optimally by using the Golden Section method.

Ransing's method	The Proposed method
1. Initialise all weights to small random numbers	
2. Until satisfied, DO	
3. For each training pattern, DO	
3.1 Input the training pattern to the network and compute the network output using sigmoid activation function with gain term $o_j = \frac{1}{(1+e^{-c_j a_{net,j}})}$	3.1 Input the training pattern to the network and compute the network output using sigmoid activation function with gain term $o_j = \frac{1}{(1+e^{-c_j a_{net,j}})}$
3.2 Calculate the gradient of Error *w.r.t* weight and gain $\frac{\partial E}{\partial w_{ij}^L} = (\sum_k \delta_k^{L+1} w_{k,j}^{L+1}) f'(c_j^L net_j^L) c_j^L$ $\frac{\partial E}{\partial c_j^L} = (\sum_k \delta_k^{L+1} w_{k,j}^{L+1}) f'(c_j^L net_j^L) net_j^L$	3.2 Calculate the gradient of Error *w.r.t* weight and gain $\frac{\partial E}{\partial w_{ij}^L} = (\sum_k \delta_k^{L+1} w_{k,j}^{L+1}) f'(c_j^L net_j^L) c_j^L$ $\frac{\partial E}{\partial c_j^L} = (\sum_k \delta_k^{L+1} w_{k,j}^{L+1}) f'(c_j^L net_j^L) net_j^L$
3.3 Use constant value for learning rate value.	**3.3 At step n calculate learning rate by using Golden section search.** $E(w_n + \eta_n^* d_n) = \min_{\lambda \geq 0} E(w_n + \eta_n d_n)$
3.4 At step n calculate learning rule for weights and gain $\Delta w_{ij}^{(n)} = -\eta^{(n)} \eta_{\substack{for \\ each \\ node}} (c_j^L(n)) \frac{\partial E}{\partial w_{ij}^{(n)}}$ $\Delta c_j^L(n) = \eta \delta_j^L \frac{net_j^L}{c_j^L(n)}$	**3.4 At step n calculate learning rule for weights and gain** $\Delta w_{ij}^{(n)} = -\eta^{(n)} \frac{\partial E}{\partial w_{ij}^{(n)}} (c_j^L(n))$ $\Delta c_j^L(n) = \eta \delta_j^L \frac{net_j^L}{c_j^L(n)}$
3.5 Update each network weight $w^{(n+1)} = w^{(n)} + \Delta w_{ij}^{(n)}$ $c_j^L(n+1) = c_j^L(n) + \Delta c_j^L(n)$	3.5 Update each network weight $w^{(n+1)} = w^{(n)} + \Delta w_{ij}^{(n)}$ $c_j^L(n+1) = c_j^L(n) + \Delta c_j^L(n)$
END DO	

Table 3.1: Comparison between the proposed algorithm and the method proposed by Ransing (2002).

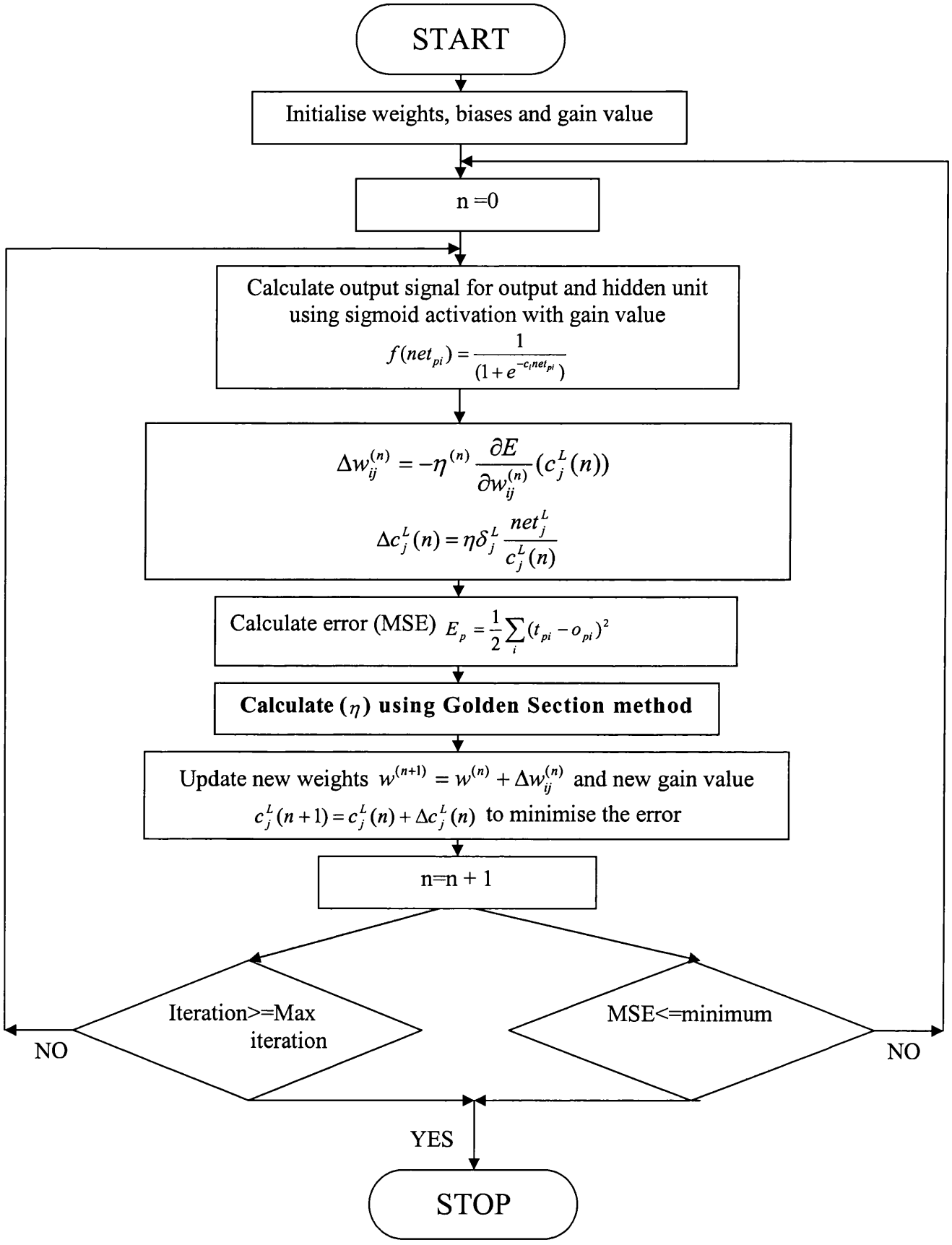


Figure 3.3: Flowchart for the proposed method.

3.2.1 Verification of the proposed method on simple data sets:

(a) Case 1

The speed of convergence achieved using the proposed method on the gradient-descent method is demonstrated by using the same datasets created by Ransing (2002) in Section 2.6.1 (refer to Table 2.1). The output of the proposed method (black continuous curve) is shown against the training data points (circles) in Figure 3.4. Whereas, Figure 3.3 demonstrated the error versus number of epochs required to achieve the target error 0.01.

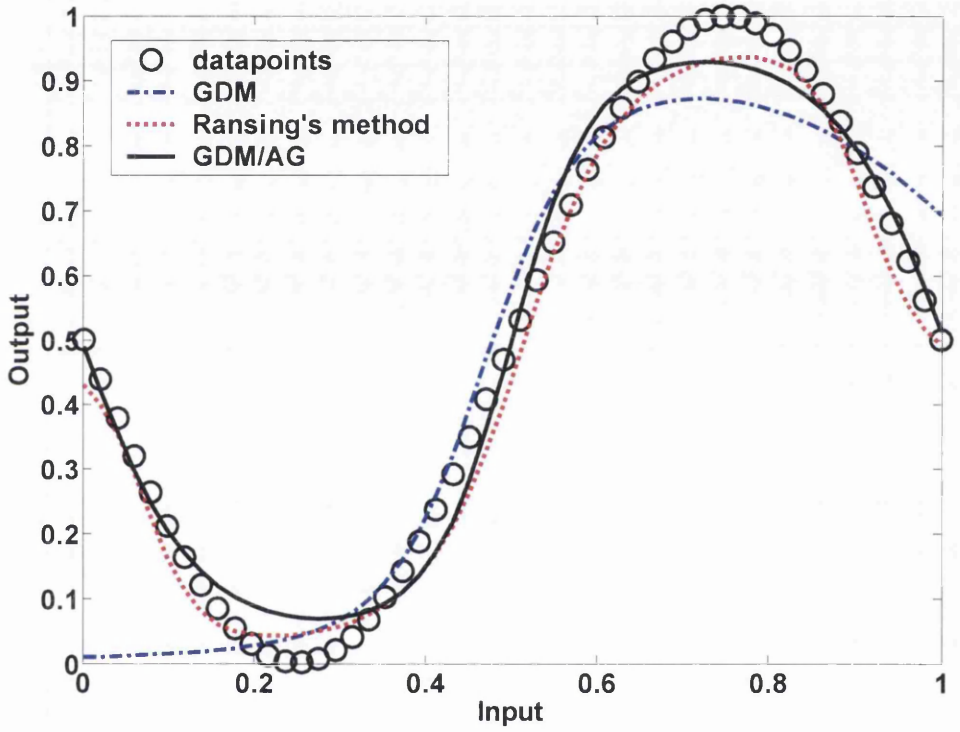


Figure 3.4: Output of the proposed network (GDM/AG) trained to learn a sine curve corresponds to Figure 2.7.

As shown in Figure 3.5, that the proposed network (GDM/AG) outperformed both methods including the method proposed by Ransing (2002), the proposed method significantly reduced the number of epochs in order to reach the target error without losing the generalisation accuracy.

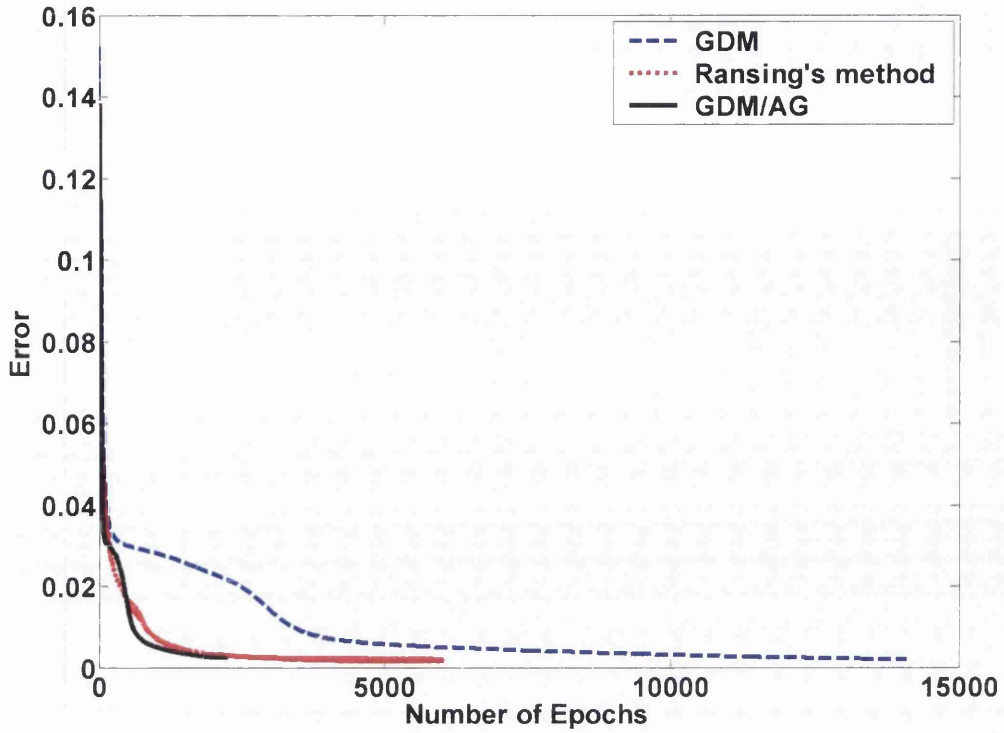


Figure 3.5: Error versus number of epochs required by the proposed method (GDM/AG) to achieve the target error of 0.001 using the gradient-descent method. This figure corresponds to Figure 2.8.

(b) Case 2

The dataset created for Case 2 is also taken from Ransing (2002). The network was required to approximate the function determined by a sample of fifty-two input points shown in Figure 3.6 (circles). The output of the proposed method (black continuous curve) is shown against the training data points (circles) in Figure 3.6. Figure 3.7 computes the speed of convergence. The number of epochs required by the proposed method (black continuous curve) are compared with the method proposed by Ransing (2002) (red solid curve) and the standard gradient-descent method (blue dot continuous curve). It is clear that the proposed method (GDM/AG) outperformed both methods including that proposed by Ransing. Even though the speed of convergence can be improved by using the method proposed by Ransing (2002), but it still required more iterations as compared to the proposed method. The proposed method only took 625 epochs in reaching the target function as compared with 905 epochs required by Ransing's method and 3,295 epochs for standard gradient-descent

(GDM) method. It shows that the speed of convergence for the proposed method is high due to the modification of the gradient-search direction by adaptively modifying gain values together with varying the learning-rate value. At the end of the training the values of gain of the proposed method for the five hidden nodes at the end of training are 2.0, 0.8020, 0.5474, 0.9116 and 1.6337. The value of gain for the output node at the end of training is 0.5318.

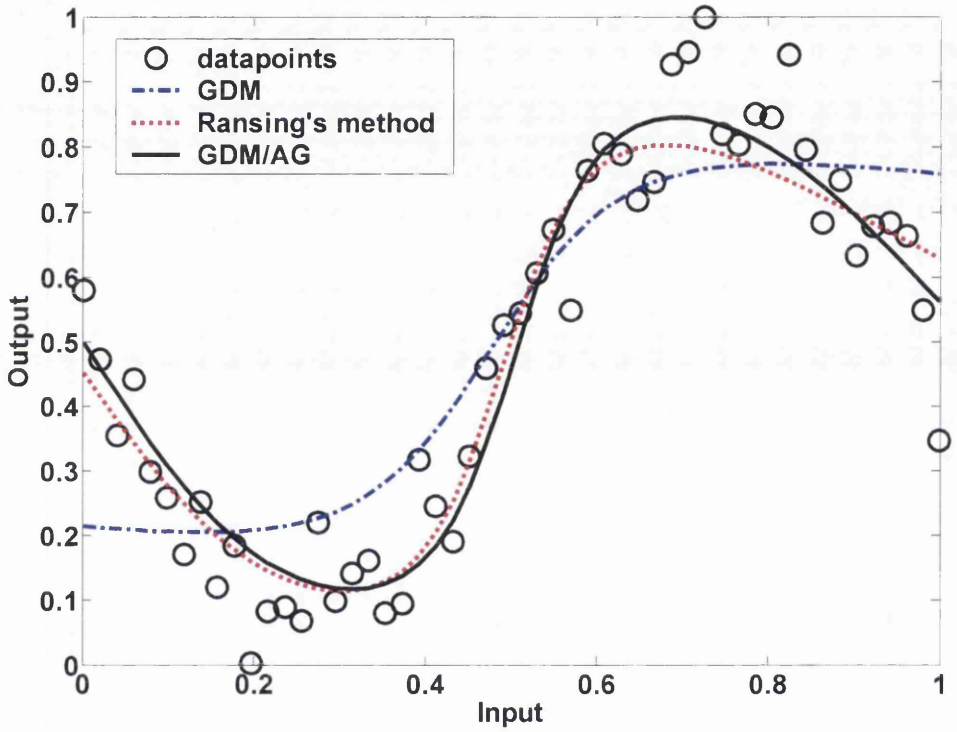


Figure 3.6: Output of the proposed network (GDM/AG) trained to learn a sine curve with twenty per cent random Gaussian noise using the proposed method. This figure corresponds to Figure 2.10.

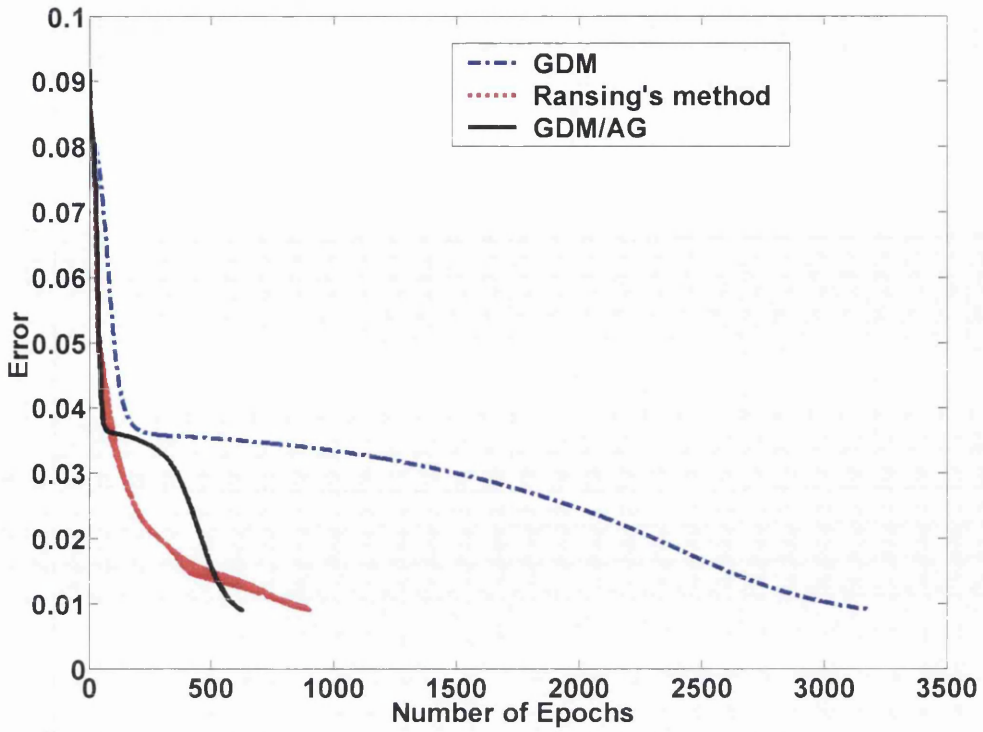


Figure 3.7: Error versus number of epochs required by the proposed method (GDM/AG) to achieve the target error of 0.01 using the gradient-descent method. This figure corresponds to Figure 2.11.

In general, both sets of results (i.e. for cases (a) and (b)) clearly showed that the proposed method implemented with the gradient-descent method significantly improved the training speed. The speed of convergence was high due to the effect of introducing gain variations and learning rates at each training epoch.

Since most of the optimisation techniques used the gradient information to calculate their search direction, the next section demonstrates the implementation of the proposed method in modifying the search direction using adaptive-gain variation with various optimisation techniques.

3.3 THE IMPLEMENTATION OF THE PROPOSED METHOD WITH VARIOUS OPTIMISATION TECHNIQUES

In this section, the proposed method is implemented into other well-known optimisation methods. The proposed optimisation algorithm calculates the learning rate adaptively by using the Golden Section method (Curtis F. Gerald and Patrick O. Wheatley, 2004) and calculates the search direction optimally by using gain variation.

3.3.1 Conjugate-gradient method with adaptive-gain variation

One of the remarkable properties of the conjugate-gradient method is its ability to generate, in a very economical fashion, a set of vectors with a property known as 'conjugacy' (C.M. Bishop, 1995). The most widely used conjugate-gradient algorithms are given by Fletcher and Powell (R. Fletcher and M.J.D. Powell, 1963) and the Fletcher-Reeves (R. Fletcher and R.M. Reeves, 1964) ones. Both of these procedures generate conjugate directions of search and therefore aim to minimise a positive definite quadratic function of n variables in n steps.

The search direction at each iteration is determined by updating the weight vector as given in Equation 3.1, where: $d_{(n)} = g_{(n)} + \beta_{(n)} d_{(n-1)}$. The scalar β_n is to be determined by the requirement that d_n and d_{n+1} must fulfill the conjugacy property (C.M. Bishop, 1995). The calculation procedure for scalar β_n determines different types of conjugate-gradient methods. The well-known formulae for β_n are those of Fletcher-Reeves (Fletcher R. and Reeves R. M., 1964) and Polak-Ribiere (E. Polak, 1971) and are given by:

$$\beta_n = \frac{g_{n+1}^T g_{n+1}}{g_n^T g_n} \quad \text{(Fletcher-Reeves (Fletcher R. and Reeves R. M., 1964))} \qquad (3.4)$$

$$\beta_n = \frac{g_{n+1}^T [g_{n+1} - g_n]}{g_n^T g_n} \quad \text{(Polak-Ribiere (Polak E., 1971))} \qquad (3.5)$$

The proposed method that is implemented with the conjugate-gradient method is referred as CGFR/AG for Fletcher-Reeves' implementation and CGPR/AG for Polak-Ribiere's implementation. The method begins the minimisation process with an initial estimate w_0 and an initial search direction as:

$$d_0 = -\nabla E(w_0) = -g_0 \tag{3.6}$$

Then, for every epoch the search direction at $(n+1)^{th}$ iteration is calculated as:

$$d_{n+1} = -\frac{\delta E}{\delta w_{n+1}}(c_{i,n+1}) + \beta_n(c_{i,n}) d_n(c_{i,n}) \tag{3.7}$$

The proposed algorithm for the conjugate-gradient method is summarised as follows:

Step 1 Initialise the weight vector randomly, the gradient vector g_0 to zero and gain value to one. Let the first search direction $d_0 = g_0$. Set $\beta_0 = 0$, $epoch = 1$ and $n = 1$. Let Nt be the number of weight parameters. Select a convergence tolerance (CT).

Step 2 At step n, evaluate gradient vector $g_n(c_n)$ with respect to gain vector c_n and calculate gain vector.

Step 3 Evaluate $E(w_n)$. IF $E(w_n) < CT$ then STOP training ELSE go to Step **4**.

Step 4 Calculate a new search direction: $d_n = -g_n(c_n) + \beta_{n-1} d_{n-1}$.

Step 5 For the first iteration, check if $n > 1$ THEN with the function of gain, update β_{n+1} by modifying the following formula:

$$\beta_{n+1} = \frac{g_{n+1}^T(c_{n+1}) g_{n+1}(c_{n+1})}{g_n^T(c_n) g_n(c_n)} \text{ for CGFR/AG or}$$

$$\beta_n = \frac{g_{n+1}^T(c_{n+1})[g_{n+1}(c_{n+1}) - g_n(c_n)]}{g_n^T(c_n) g_n(c_n)} \text{ for CGPR/AG}$$

ELSE go to Step **6**.

Step 6 IF $[(epoch+1)/Nt] = 0$ THEN 'restart' the gradient vector with $d_n = -g_{n-1}(c_{n-1})$, ELSE go to Step **7**.

Step 7 Calculate the optimal value for learning rate η_n^* by using the Golden Section line search technique such as:

$$E(w_n + \eta_n^* d_n) = \min_{\lambda \geq 0} E(w_n + \eta_n d_n)$$

Step 8 Update w_n: $w_{n+1} = w_n + \eta_n^* d_n$

Step 9 Evaluate new gradient vector $g_{n+1}(c_{n+1})$ with respect to gain value c_{n+1}.

Step 10 Calculate new search direction: $d_{n+1} = -g_{n+1}(c_{n+1}) + \beta_n(c_n) d_n$

Step 11 Set $n = n+1$ and go to Step **2**.

3.3.2 Quasi-Newton methods with adaptive-gain variation

Quasi-Newton methods are also among the most popular algorithms used in unconstrained optimisation techniques. For an error function $E(w)$, a typical iteration of the Quasi-Newton method is given as in Equation 3.3, where the search direction $d_{(n)}$ is defined by solving a system of equations:

$$d_{(n)} = -[H_{(n)}]\nabla E(w_{(n)}) \tag{3.8}$$

where: $\nabla E(w) = g$ and H is the Hessian matrix which is adjusted from iteration to iteration such that the direction $d_{(n)}$ approximates the Newton direction. Quasi-Newton methods achieve fast convergence and also do not require direct computation of second-order derivatives. Two commonly used implementations of Quasi-Newton methods are Broyden-Fletcher-Goldfarb-Shanno (BFGS) formulation (Adrian J. Sheperd, 1997) and Davidon-Fletcher-Powell (DFP) formulation (F. Fnaiech *et al.*, 1994). The basic difference between those two formulations is in the way the inverse Hessian is constructed. This is illustrated in the following formula:

$$\nabla_n = \left(1 + \frac{y_n^T H_n y_n}{s_n^T y_n}\right)\frac{s_n s_n^T}{s_n^T y_n} - \frac{s_n y_n^T H_n}{s_n^T y_n} \text{ (BFGS (Adrian J. Sheperd, 1997))} \tag{3.9}$$

$$\nabla_n = \left(1 + \frac{s_n^T H_n s_n}{y_n^T s_n}\right)\frac{y_n y_n^T}{y_n^T s_n} - \frac{y_n s_n^T H_n + H_n s_n y_n^T}{y_n^T s_n} \text{ (DFP (F. Fnaiech } et\ al.\text{, 1994))} \tag{3.10}$$

The vectors s_n and y_n are determined as in Equation 3.11 and Equation 3.12, respectively.

The proposed method uses the gradient vector that is a function of the gain parameter ($g_n(c_n)$) as described in Equation 3.7. Such a modification has been implemented in both the BFGS and DFP formulations and referred as BFGS/AG and DFP/AG. The proposed algorithms for Quasi-Newton is summarised as follows:

Step 1 Initialise the weight vector $w(0)$ along with a positive definite of Hessian matrix $H(0) = I$. Select a convergence tolerance CT.

Step 2 Compute the search direction d_n with respect to gain variation by solving $d_n = -H_n g_n(c_n)$.

Step 3 Search the optimal value for η_n^* by using a Golden Section line search technique such as:

$$E(w_n + \eta_n^* d_n) = \min_{\lambda \geq 0} E(w_n + \eta_n d_n).$$

Step 4 Update w_n: $w_{n+1} = w_n + \eta_n^* d_n$.

Step 5 Compute:

$$s_n = w_{n+1} - w_n \tag{3.11}$$

$$y_n = g_{n+1}(c_{n+1}) - g_n(c_n) \tag{3.12}$$

Construct the inverse Hessian by modifying the following formula:

$\nabla_n = \left(1 + \frac{y_n^T(c_n) H_n y_n(c_n)}{s_n^T y_n(c_n)}\right) \frac{s_n s_n^T}{s_n^T y_n(c_n)} - \frac{s_n y_n^T(c_n) H_n}{s_n^T y_n(c_n)}$ for BFGS/AG or

$\nabla_n = \left(1 + \frac{s_n^T H_n s_n}{y_n^T(c_n) s_n}\right) \frac{y_n(c_n) y_n^T(c_n)}{y_n^T(c_n) s_n} - \frac{y_n(c_n) s_n^T H_n + H_n s_n y_n^T(c_n)}{y_n^T(c_n) s_n}$ for DFP/AG.

Step 6 Update the inverse matrix H_n: $H_{n+1} = H_n + \nabla_n$.

Step 7 Compute the error function value. $E(w_n)$

Step 8 If $E(w_n) > CT$ go to Step **2**, else stop training.

3.4 COMPARISON OF THE PROPOSED TRAINING METHOD WITH THE EQUIVALENT STANDARD METHODS ON A SIMPLE DATA SET

This section is dedicated to presenting an analysis of the performance of the proposed training method coupled with other optimisation methods when applied to a continuous approximation function. The proposed method described in Section 3.4 has been implemented by using the MATLAB programming language (refer to Appendices I.1 to I.6). In training those networks, the main objective of this section is to ensure that the proposed programming code learns and trains correctly on a sample data set. Further validation of the proposed method will be undertaken in the next chapter.

3.4.1 Experimental setup

The critical issue in training neural networks is to measure the generalisation performance of the network by comparing the network output with the known output on a data set that is not in the training data. The standard neural network architectures, such as the fully-connected multi-layer perceptrons, are prone to overfitting (Stuart Geman *et al.*, 1992). As the number of hidden nodes increase, the number of unknowns, or degree of freedom in the network, increases and the risk of overfitting to the data also increases. On the other hand, with too few nodes the network will not be flexible enough to adapt to the true input-output relationship. Therefore it is important to get the number of hidden nodes approximately right before proceeding with the training.

3.4.1.1 Early stopping

Overfitting as mentioned in Section 3.4.1 may be prevented in three main ways: (a) by limiting the number of hidden nodes; (b) by adding a penalty term to the objective function for large weights; or (c) by limiting the amount of training using early stopping (Murray Smith, 1993). Although all three methods are potentially useful,

early stopping is used in this section to prevent overfitting, as it is the most popular, simple to understand, and easy to implement.

In this technique, as shown in Figure 3.8, data is divided into training and validation data sets. The network is trained using only the training data and at predetermined training intervals (e.g., after every twenty epochs), the error of the network is determined in both the training and the validation data sets, and the connection weight configuration of the network is saved. Training continues with the error in the training set usually declining with additional training. However, the training is stopped as soon as the corresponding error in the validation set is higher than it was the last time it was checked. The network weight configuration with the least error in the test set is considered the best network and is used for future predictions in external validation sets.

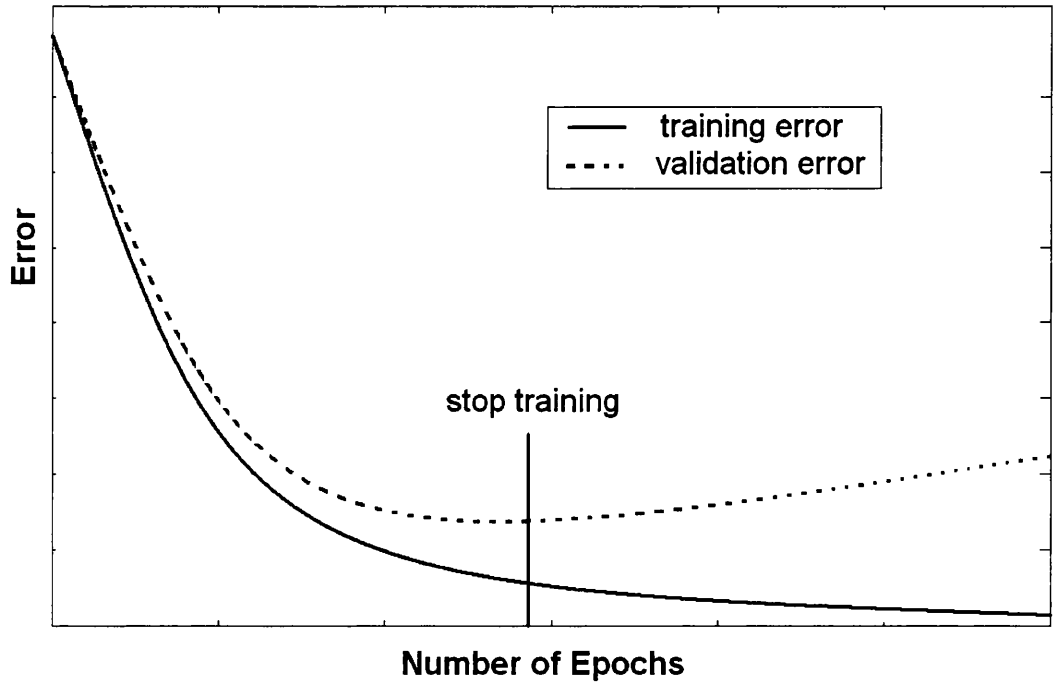


Figure 3.8: Idealised training and validation error curve.

3.4.1.2 Initial training conditions

During the training process in neural networks, the weight, bias and gain values are updated after each epoch. An epoch is said to be complete after the presentation of a training example. A mean squared-error (MSE) value is calculated after the presentation of all training examples and compared with the target error. Training is

done on an epoch-by-epoch basis until the MSE value falls below the desired target-error value or by stopping the training after the stopping criteria as mentioned in Section 3.4.1.1. has been satisfied.

The selection of initial weights and bias are important parameters in the training process. If the initial weights are very small, the back-propagated error is so small that practically no change takes place for some weights, and therefore more iterations are necessary to decrease the error (D.E. Rumelhart *et al.*, 1986). In the worst case the error remains constant and the learning stops in an undesired local minimum (Y. Lee *et al.*, 1993). On the other hand, large values of weights speed up learning, but they can lead to saturation and to flat regions of the error surface where training is considerably slow (G.D. Magoulas *et al.*, 1996). Thus, in order to evaluate the performance of the algorithms better, the experiments were conducted using the same initial weight and bias vectors that have been randomly chosen from a uniform distribution between [0,1].

3.4.1.3 Training cases

The speed of convergence achieved using the proposed algorithm is demonstrated in the following example which corresponds to the data set created by Ransing (2002) in the second chapter of her thesis. Consider a single input-output two-layer network with one hidden layer having five hidden nodes as can be seen in Figure 3.9. There are two training data sets that have been used in this section and those training data sets are created by using the function $y = \sin(pi * x)$ *where* $x \in$ [0,1] and by adding an approximate twenty per cent random Gaussian noise. The network is trained using 0.3 as the learning rate and 0.4 as the momentum value to achieve a one per cent target error. A sigmoid activation function was used in order to generate an output value between '0' and '1' at each node.

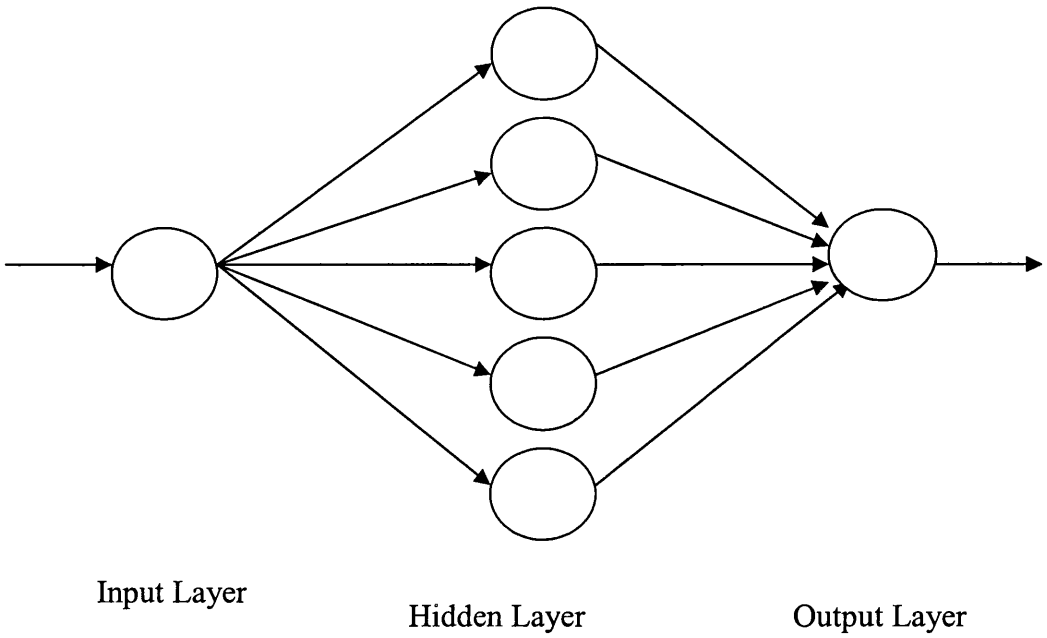


Figure 3.9: Fully-connected MLP network with 1-5-1 architecture.

3.4.1.4 Training algorithms

All the proposed methods have been implemented by using MATLAB programming language. The network is trained using eight training algorithms with coupled and adaptive changes in weight, bias and gain values. The network is trained with an adaptive gain for all output as well as hidden nodes. Those eight training algorithms are:

1) The standard conjugate gradient-Fletcher-Reeves (CGFR) (Fletcher R.and Reeves R. M., 1964).
2) The newly proposed conjugate gradient-Fletcher-Reeves method with adaptive gain (CGFR/AG)
3) The standard conjugate gradient-Polak-Ribiere (CGPR) (Polak E., 1971).
4) The newly-proposed conjugate gradient-Fletcher-Reeves method with adaptive gain (CGPR/AG)
5) The standard Broyden-Fletcher-Goldfarb-Shanno (BFGS) (Adrian J. Sheperd, 1997).
6) The newly-proposed Broyden-Fletcher-Goldfarb-Shanno method with adaptive gain (BFGS/AG).
7) The standard Davidon-Fletcher-powell (DFP) (F. Fnaiech *et al.*, 1994).
8) The newly-proposed Davidon-Fletcher-Powell with adaptive gain (DFP/AG)

3.4.2 Experiment results

(a) Case 1: The sin(x) problem without noise

The speed of convergence achieved by all training algorithms as mentioned in Section 3.6.1.4 is demonstrated using the same data sets as illustrated in Section 2.6.1. The training procedure and conditions are kept the same as proposed by Ransing (2002). During each epoch, the gain, weight and bias values for all hidden nodes and output nodes converged to achieve an MSE of 0.001 for gain, weight and bias values, respectively. The terminated number of epochs and average running time for all algorithms are included in Table 3.2.

Methods	No of epochs in which the convergence was achieved	CPU time (seconds)
Standard CGFR	292	21.88
CGFR/AG	257	19.34
Standard CGPR	365	23.64
CGPR/AG	315	27.72
Standard BFGS	1400	89.23
BFGS/AG	760	76.55
Standard DFP	2546	175.67
DFP/AG	1932	121.21

Table 3.2: Summary of simulation results.

It can be seen from Table 3.2 that all proposed methods with adaptive gain have consistently outperformed other standard algorithms in terms of number of epochs and CPU time required for the convergence. For the purpose of comparison, only the performance of the conjugate gradient (Fletcher-Reeves) formulation is presented and discussed in this section whereas the performance of other formulations is discussed in Appendices I.1 to I.16.

Figures 3.10 and 3.11 illustrate the performance of the proposed method implemented into conjugate-gradient (Fletcher-Reeves) formulation. As can be seen from Figure

3.11, the proposed method (CGFR/AG) took 257 epochs to learn the target function as compared to the standard algorithm which took about 292 epochs.

The gain values for the five hidden nodes at the end of training are 1.0232, 1.0002, 1.1743, 0.5107, and 1.0343, respectively. The gain value for the output node at the end of the training is 1.6232. As shown in Table 3.2, the speed of convergence for the proposed method (CGFR/AG) is fast because the modified gain values improved the gradient-search direction as compared to the standard algorithm with unity gain value.

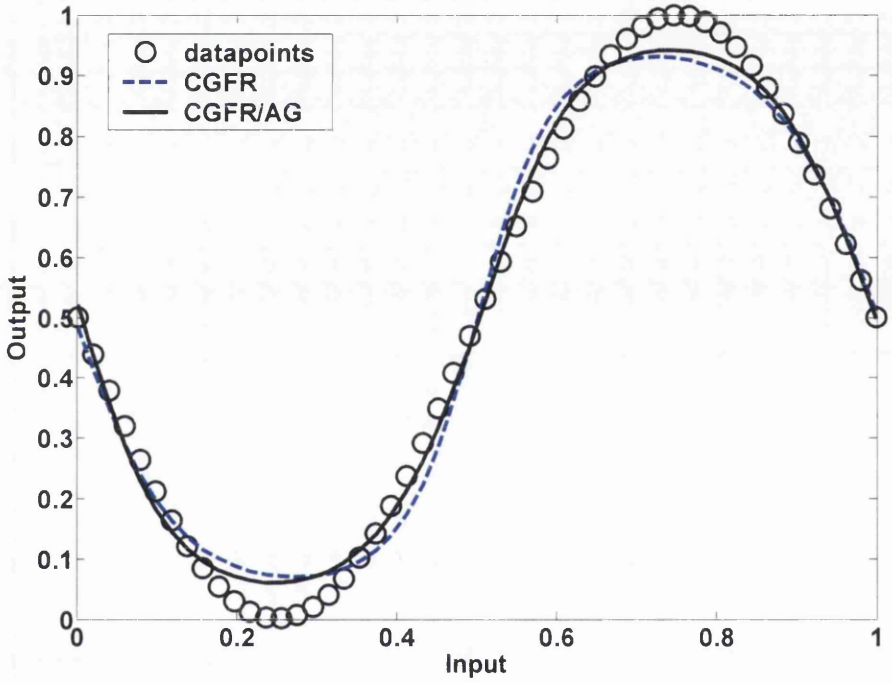


Figure 3.10: Output of neural networks trained to learn a sine curve using the proposed conjugate-gradient method with Fletcher-Reeves' formulation.

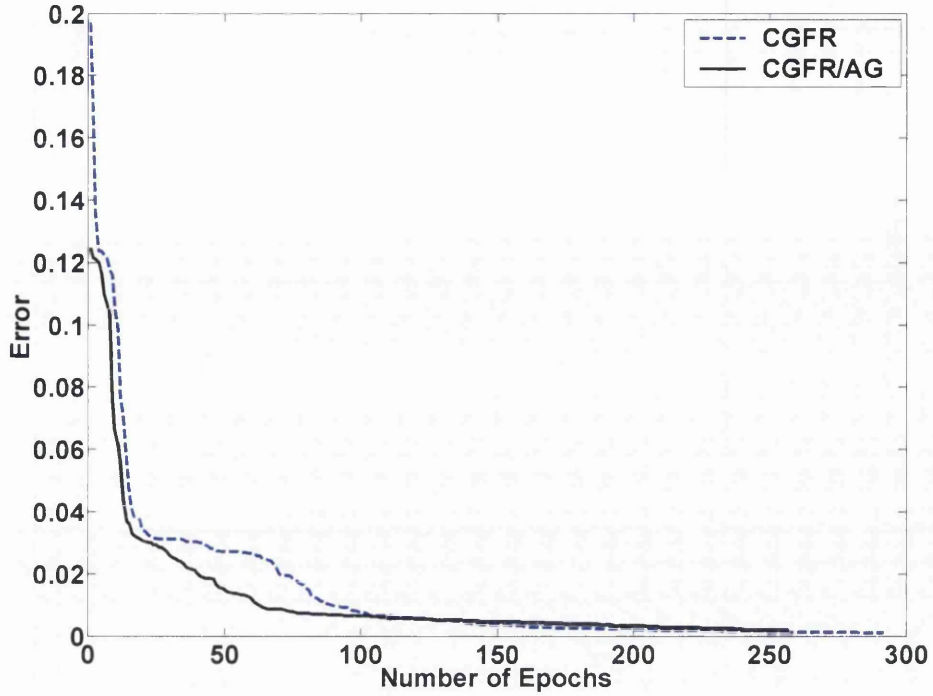


Figure 3.11: Error versus number of epochs required to achieve the target error of 0.001 for the conjugate-gradient method with Fletcher-Reeves' formulation.

(b) Case 2: The sin(x) problem with noise

As mentioned in the previous section, the second training data set is also created by using the same function used by Ransing. The data point values were altered using the Gaussian noise. The same network architecture is used for this experiment's data and all parameters were kept the same as for the first experiment, except for the target-error value. At each epoch, the gain, weight and bias values for all hidden nodes and output nodes converged to achieve a target error of 0.01 for gain, weight and bias values, respectively. The terminated number of epochs and average running time for all algorithms are included in Table 3.3.

Methods	Number of iteration in which the convergence was achieved	CPU time (seconds)
Standard CGFR	111	7.59
CGFR/AG	54	3.36
Standard CGPR	167	11.75
CGPR/AG	111	7.51
Standard BFGS	450	34.73
BFGS/AG	316	22.48
Standard DFP	659	41
DFP/AG	429	27

Table 3.3: Summary of simulation results with twenty per cent random Gaussian noise.

In achieving the results in Table 3.3 the following discoveries were made: first, the standard gradient descent was not able to learn the noisy data properly. As a result it took a longer time and number of epochs to achieve the target-error value. Second, all proposed methods exhibited good performance in reaching the target-error value in terms of the number of epochs and CPU time required for convergence as compared with other standard algorithms.

The proposed method implemented with the conjugate gradient (Polak-Ribiere) formulation is chosen for the purpose of comparison. As can be seen from Figure 3.12, both algorithms have predicted the same mapping approximating function. However, the proposed method (CGPR/AG) significantly improved the training speed by reducing the number of epochs from 167 to 111 (Figure 3.13). The speed of convergence for the proposed method (CGPR/AG) is reduced due to the modified gain values that further improved the gradient-search direction as compared to the standard algorithm.

The performance comparison of other optimisation methods coupled with the proposed method is shown in Appendix A.I.

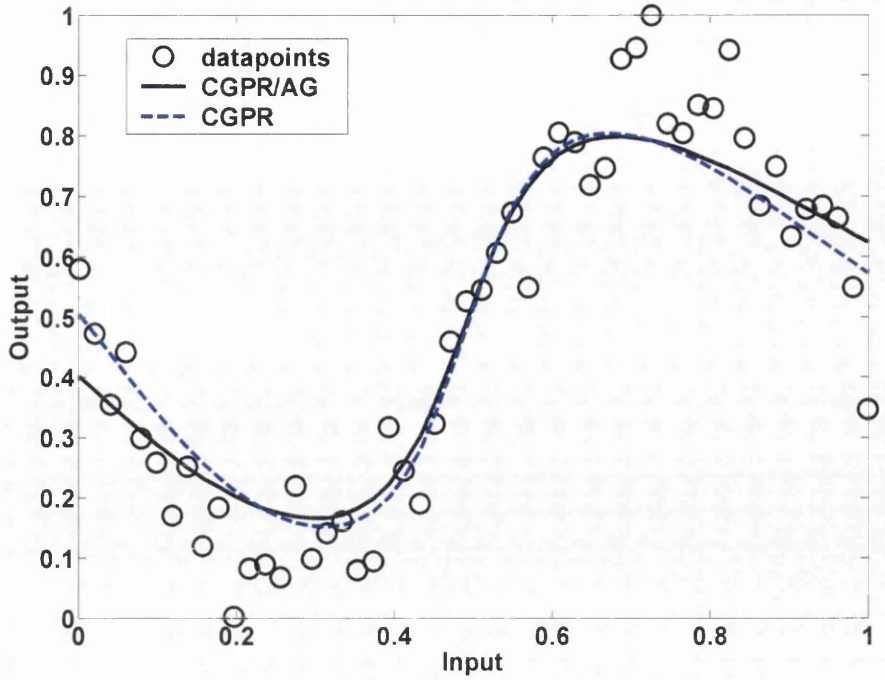


Figure 3.12: Output of neural networks trained to learn a sine curve with twenty per cent random Gaussian noise using the proposed method with Polak-Ribiere formulation.

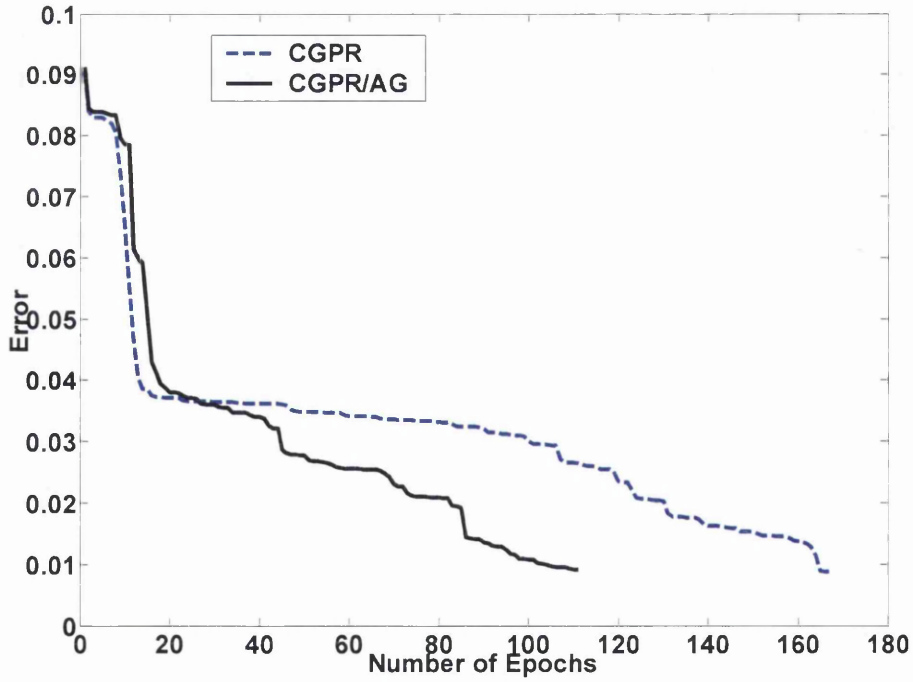


Figure 3.13: Error versus number of epochs required to achieve the target error of 0.01 using the conjugate-gradient method with Polak-Ribiere formulation.

3.5 CONCLUSION

The back propagation algorithm, which is frequently used in the neural-network training process, has often been criticised for its stability and convergence problems. Many researchers have devoted their efforts in improving its efficiency ranging from employing various optimisation methods for proposing new algorithms. Previous literature showed that changing the 'gain' value adaptively for each node can also improve the network training speed. The previous research concluded that adaptive gain has a catalytic effect in the learning process by magnifying the magnitude of the learning rate.

The research presented in this chapter proposes that for the standard gradient-descent algorithm, the adaptive-gain variation does not influence the learning rate as perceived in the previous research but modifies the initial search direction. A mathematical formulation, based on this insight, was developed for various optimisation techniques.

The proposed method has been successfully implemented into the MATLAB programming language and the experimental results have shown that the proposed programming code was trained correctly and significantly reduced both the number of epochs and the CPU time in reaching the target error as compared to other standard algorithms. Furthermore, the proposed method has been shown to be more generic and easy to implement into other commonly used gradient-based optimisation algorithms. The proposed method provides an important lead into the gradient-based optimisation techniques for improving training efficiency.

CHAPTER 4

RESULTS AND VALIDATION ON BENCHMARK PROBLEMS

CHAPTER LAYOUT

This chapter undertakes further tests to validate the efficiency and accuracy of the newly-developed method discussed in Chapter Three. The benchmark problems used to verify the proposed algorithm are taken from the open literature (Lutz Prechelt, 1994). The following two criteria are used to assess the computational efficiency and accuracy of the proposed method in comparison with the benchmark problem: (a) the speed of convergence measured in the number of epochs and the CPU time; and (b) the classification accuracy on testing data from the benchmark problems. The results are discussed and conclusions drawn from the research presented in the last section of this chapter.

4.1 INTRODUCTION

Chapter Three suggested that a simple modification to the gradient-based search direction can also substantially improve the training efficiency of almost all major optimisation methods. It was discovered that if the gradient-based search direction is locally modified by a gain value used in the activation function of the corresponding node, significant improvements in the convergence rates can be achieved irrespective of the optimisation algorithm used. The correctness of the proposed method using the MATLAB programming language was demonstrated by comparing the learning speed on the data generated from a sine function.

In this chapter, the robustness of the proposed algorithm is further validated and it is illustrated by comparing convergence rates for gradient descent, conjugate-gradient and Quasi-Newton methods on many benchmark examples. The remains of the chapter is organised as follows: Section Two discusses the preliminaries of the simulation; and in Section Three, the robustness of the proposed algorithm is shown by comparing convergence rates for gradient descent, conjugate gradient and Quasi-Newton methods on many benchmark examples. The chapter is concluded in the final section along with a short discussion on further research.

4.2 PRELIMINARIES

The performance analysis used in this research focuses on two criteria: (a) the speed of convergence measured in number of iterations and CPU time; and (b) the classification accuracy on testing data from the benchmark problems. The benchmark problems used to verify the proposed algorithm are taken from the open literature (Lutz Prechelt, 1994). To perform the experiments the data has to be arranged into a collection of training and testing sets. The algorithm is trained on the training set and its performance is measured on the corresponding testing set (early stopping) as mentioned in Section 3.4.1.1. In this case two-thirds of the examples in each category were randomly placed in the training set, and the remaining examples formed the

testing set. To reduce statistical fluctuations, the results are averaged over several simulations on the same training and testing sets.

Six classification problems have been tested including Thyroid, Wisconsin breast cancer, Diabetes, IRIS classification problem and Glass classification (Lutz Prechelt, 1994). The simulations were carried out on a Pentium IV with a 3 GHz processor, 1 GB RAM and using MATLAB version 6.5.0 (R13).

On each problem, the following fifteen algorithms were analysed and simulated:

1) The standard gradient-descent method with the momentum term (*traingdm*) from 'MATLAB Neural Network Toolbox version 4.0.1'.
2) The standard gradient-descent method with the momentum term (GDM).
3) The modified gradient-descent method proposed by Ransing (2002).
4) The gradient-descent method with the momentum term and adaptive-gain variation (GDM/AG) method.
5) The standard conjugate gradient-Fletcher-Reeves (*traincgf*) method from the 'MATLAB Neural Network Toolbox version 4.0.1'.
6) The standard conjugate gradient-Fletcher-Reeves (CGFR) method.
7) The conjugate gradient-Fletcher-Reeves method with adaptive-gain variation (CGFR/AG) method.
8) The standard conjugate gradient-Polak-Ribiere (*traincgp*) method from the 'MATLAB Neural Network Toolbox version 4.0.1'.
9) The standard conjugate gradient- Polak-Ribiere (CGPR) method.
10) The conjugate gradient-Polak-Ribiere method with adaptive-gain variation (CGPR/AG) method.
11) The standard Broyden-Fletcher-Goldfarb-Shanno (*trainbfg*) method from the 'MATLAB Neural Network Toolbox version 4.0.1'.
12) The standard Broyden-Fletcher-Goldfarb-Shanno (BFGS) method.
13) The Broyden-Fletcher-Goldfarb-Shanno method with adaptive-gain variation (BFGS/AG) method.
14) The standard Davidon-Fletcher-Powell (DFP) method.
15) The standard Davidon-Fletcher-Powell method with adaptive-gain variation (DFP/AG) method.

To compare the performance of the proposed algorithm with respect to other standard optimisation algorithms from the MATLAB neural-network toolbox, network parameters such as network size and architecture (number of nodes, hidden layers, etc.), and the values for the initial weights and gain parameters were kept the same. For all problems the neural network had one hidden layer with five hidden nodes and the sigmoid activation function was used for all nodes. This architecture of the neural network was finalised after an initial study with a different number of hidden nodes.

Prior to training, the weights are initialised to small random values. The reason to initialise weights with small values is to prevent saturation (where one or more hidden node is highly active or inactive for all patterns and therefore insensitive to the training process) and random to break symmetry (Adrian J. Sheperd, 1997). If, on the other hand, the initial weights are too small, training will tend to start very slowly. In order to reduce statistical fluctuation, all algorithms were tested using the same initial weights that were initialised randomly from range [0, 1] and received the input patterns for training in the same sequence.

For the gradient-descent algorithm, the learning rate value was 0.3 and the momentum term value was 0.4. The initial value used for the gain parameter was set to one. The values were used for comparison purpose only and there was no particular reason for the choice of these values.

For each simulation run, the numerical data is shown in two files: (1) the summary results file; and (2) a detailed description of the successful algorithm. The number of iterations required to achieve convergence for each simulation result were noted to calculate the mean, the standard deviation and the number of failure cases. The cases that failed to converge are obviously excluded from the calculations of the mean and standard deviation but are reported as failure cases.

For each simulation run, the generalisation accuracy (the accuracy of the network in classifying unknown data) of all algorithms is also calculated based on the formulation proposed by Watkins (Dave Watkins, 1997). Watkins determined generalisation accuracy by calculating the inverse of a distance measure of simulation results from the real answers, expressed as a percentage of the limits of the range:

$$Accuracy(\%) = \frac{1 - \left| t_k - o_k^L \right|}{UB - LB} * 100 \qquad (4.1)$$

where UB and LB represent the upper bound and the lower bound. Both are defined based on the type of activation function that is used during the simulations, in this case all simulations are run using a sigmoid activation function. Hence UB is defined as one and LB is defined as zero. The final accuracy is determined by taking the mean of all simulation runs.

4.3 VERIFICATION ON BENCHMARK PROBLEMS

For each problem, one hundred simulation runs were obtained, each with a different randomly chosen set of initial weights. For each simulation run, the number of iterations required for convergence is reported. For an experiment of one hundred simulation runs, the mean of the number of iterations, the standard deviation, and the number of failures are collected. A failure case occurs when the network exceeds the maximum iteration limit or the stopping criteria as mentioned in Section 3.6.1.1 is met; each simulation is run for up to one thousand iterations except for the back-propagation algorithm based on the gradient-descent method as it failed to converge within the specific iteration limit and needed at least ten thousand iterations to converge; otherwise, the training procedure is halted and the run is reported as a failure case. Convergence is achieved when the output of the network achieves the above mentioned stopping criteria.

4.3.1 Performance comparison setup

To simplify the process of verification, this section describes the summary of some procedures and criteria that have been used to compare the results of all training algorithms on selected benchmarks taken from the open literature (Lutz Prechelt, 1994). The list of benchmarks tested is as follows:

- Thyroid classification problem

- Wisconsin Breast Cancer classifications problem
- Diabetes classification problem
- IRIS classification
- Seven-bit parity problem
- Glass-classification problem

To provide comparable and interpretable results, each problem will be presented with the same table as shown in Table 4.1 together with a figure that illustrates the comparison of average CPU time and number of epochs required for convergence. Each table only presents the summary of the performance for all training algorithms based on certain criteria. The detailed information is presented in tables in Appendix AIII provided at the end of this thesis.

	Mean no. of epochs	**CPU time(s) per Epoch**	**Total CPU time(s) to converge**	**SD**	**Accuracy (%)**	**Fails**
traingdm						
GDM						
Ransing's method						
GDM/AG						
traincgf						
CGFR						
CGFR/AG						
traincgp						
CGPR						
CGPR/AG						
trainbfg						
BFGS						
BFGS/AG						
DFP						
DFP/AG						

Table 4.1: Sample table showing characteristics used to compare the performance of all algorithms.

For the purpose of comparison, the following notations together with their explanation specify characteristics that have been used for each problem:

Mean number of epochs – the ratio of sum of total number of epochs with the number of simulation runs.

CPU time (s) per epochs – the total CPU time (in seconds) divided by the mean number of epochs.

Total CPU time (s) to converge – the sum of all CPU times (in seconds) divided by the number of simulation runs.

Standard deviation (SD) – the value that describes how close the results are around the mean number of epochs in one hundred simulation runs. When the result values are very close to each other, the SD value is small. When the result values are spread apart it has a relatively large SD value.

Accuracy – the value that was used by all training algorithms to determine the network performance when classifying unknown datasets (testing data) for each simulation run. It was calculated based on the formulation proposed by Watkins (Dave Watkins, 1997).

Number of failures – the value that indicates the number of runs that failed to converge within the specified MSE value or the training was stopped when the stopping criteria (refer Section 3.6.1.1) are met.

4.3.1.1 Thyroid classification problem

(ftp://ftp.ics.uci.edu/pub/machine-learning-databases/thyroid-disease)

This data set was created based on the 'artificial neural-network' version of the 'thyroid disease' problem. The data set is designed so that a neural-network model can diagnose thyroid hyper- or hypo-function based on patient query data and patient examination data. The model decides whether the patient's thyroid has over-function, normal-function, or under-function. Some 7,200 observations are used and the data is partitioned into training and validation subsets, and permutated in three ways. The selected architecture of the Feed-forward Neural-network is 21-5-3. The target error is set to 0.05 and the maximum epochs to one thousand.

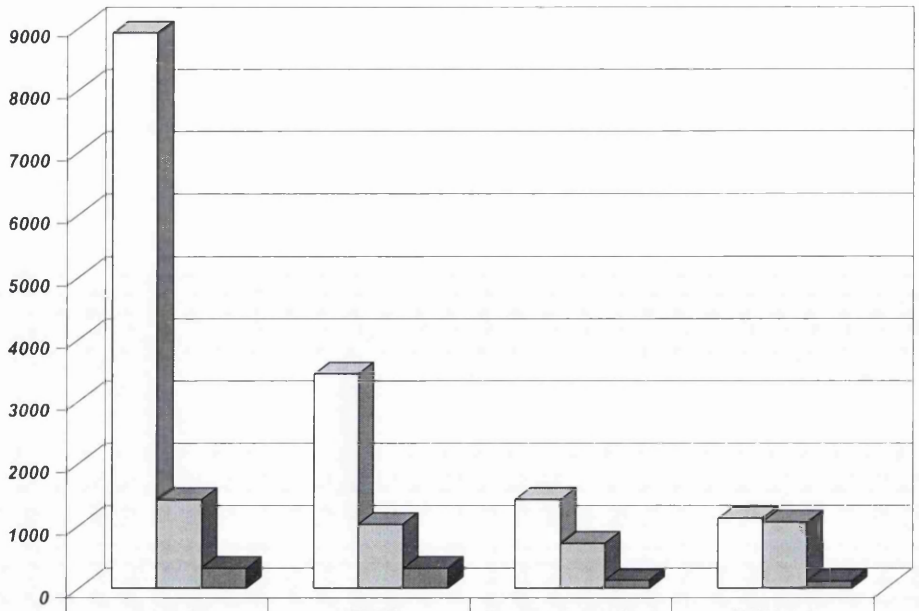
Table 4.2 provides the summary of all algorithms' performances for the Thyroid classification problem. It shows that all gradient-based algorithms coupled with the

proposed method outperform other algorithms in terms of the CPU time and number of iterations required for convergence.

	Thyroid classification problem (target error=0.05)					
	Mean no. of epochs	**CPU time(s) per Epoch**	**Total CPU time(s) to converge**	**SD**	**Accuracy (%)**	**Fails**
traingdm	8925	3.72×10^{-2}	332.03	1.70×10^{2}	87.04	16
GDM	3441	9.20×10^{-2}	316.49	1.02×10^{3}	88.17	7
Ransing's method	1413	8.87×10^{-2}	125.30	7.14×10^{2}	88.41	4
GDM/AG	1114	1.00×10^{-1}	111.63	7.44×10^{1}	88.66	4
traincgf	49	2.03×10^{-1}	10.02	5.43×10^{1}	91.03	6
CGFR	15	4.27×10^{-1}	6.48	5.89×10^{0}	91.33	4
CGFR/AG	11	4.30×10^{-1}	4.84	4.22×10^{0}	90.97	3
traincgp	**34**	**2.20×10^{-1}**	**7.4593**	**6.59×10^{1}**	**91.64**	**9**
CGPR	**13**	**3.49×10^{-1}**	**4.6151**	**3.28×10^{0}**	**89.85**	**6**
CGPR/AG	**10**	**2.64×10^{-1}**	**2.5802**	**3.53×10^{0}**	**90.37**	**3**
trainbfg	63	1.40×10^{-1}	8.7439	7.86×10^{1}	88.56	4
BFGS	35	2.74×10^{-1}	9.6520	3.68×10^{0}	90.04	1
BFGS/AG	21	2.52×10^{-1}	5.2977	4.24×10^{0}	89.62	1
DFP	107	2.79×10^{-1}	29.93	8.38×10^{1}	90.03	5
DFP/AG	53	2.39×10^{-1}	12.6739	8.47×10^{1}	89.47	3

Table 4.2: Summary of algorithms' performances for the Thyroid classification problem.

Figure 4.1 displays the performance of various gradient-descent methods on the Thyroid classification problem. Note that the proposed method and the method that was introduced by Ransing (2002) had drastically reduced the number of epochs required for convergence as well as the CPU time over one hundred simulation runs as compared to the neural-network toolbox. However, it can be seen that the performance of the method proposed in this thesis is still better as compared to the one proposed by Ransing. Furthermore, in reaching the target error, the proposed method had reduced the number of failure cases from sixteen when using a neural-network toolbox to four, even though both algorithms gave most similar results on generalisation accuracy. Almost all algorithms have given similar results on the generalisation accuracy.

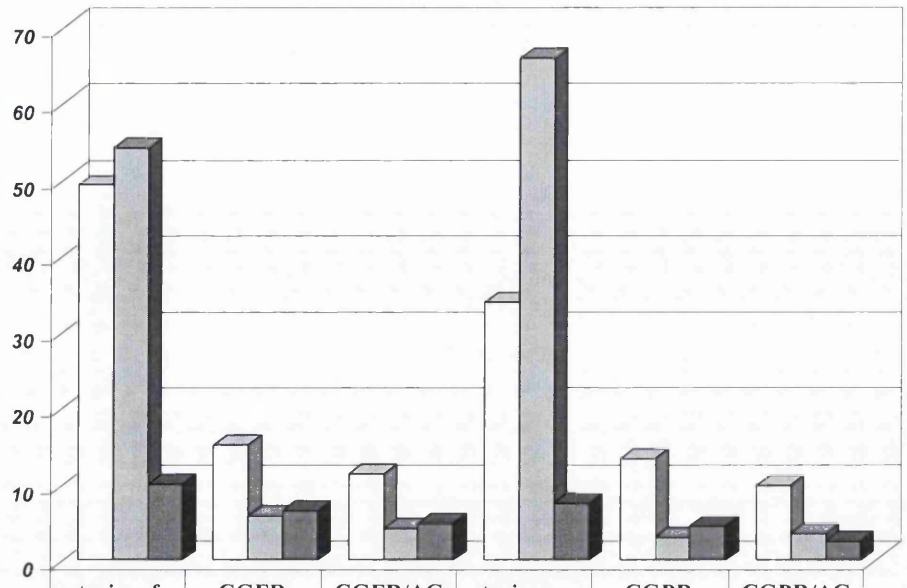


	traingdm	GDM	Ransing's method	GDM/AG
Mean no. of Epochs	8925	3441	1413	1114
Standard deviation	1419.85	1015.82	713.64	1066.93
CPU time (seconds)	332.03	316.49	125.30	111.63

Figure 4.1: Comparison of average CPU time and number of epochs required for convergence using the gradient-descent method for the Thyroid classification problem.

The proposed method has also shown superior performance when integrated with the conjugate-gradient formulation. As can be seen in Figure 4.2, in order to reach the target error of 0.05, the mean value for epoch calculated over one hundred simulation runs was ten for the proposed method (CGPR/AG) as opposed to the standard CGPR method where the mean value for epochs was thirteen. This is an improvement ratio of nearly 3.4, similarly the improvement ratio is almost 1.6477 for the convergence time.

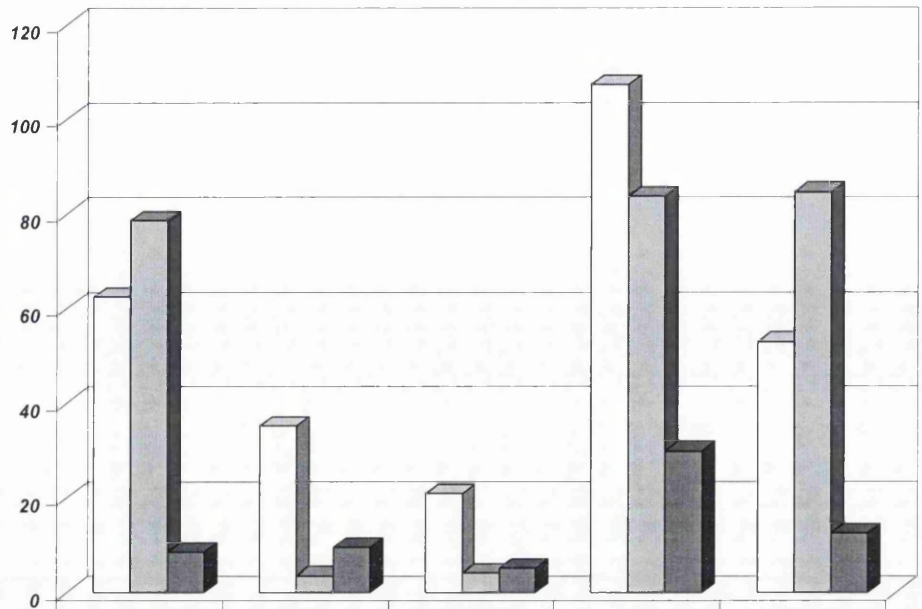
The proposed algorithm (CGPR/AG) also shows better results as compared to the neural-network toolbox (*traincgp*) even though the proposed method had three failure cases, but it is considered better as compared to the neural-network toolbox with nine failure cases. This makes the CGPR/AG algorithm a better choice for this problem since it had only three failure cases for the one hundred different simulation runs. (Refer to Appendix II.1 for a detailed calculation procedure for evaluating the CGPR/AG performance).



	traincgf	CGFR	CGFR/AG	traincgp	CGPR	CGPR/AG
Mean no. of Epochs	49	15	11	34	13	10
Standard deviation	54.26	5.89	4.22	66	2.99	3.53
CPU time (seconds)	10.02	6.48	4.84	7.46	4.62	2.58

Figure 4.2: Comparison of average CPU time and number of epochs required for convergence using the conjugate-gradient method for the Thyroid classification problem.

Figure 4.3 displays a summary of the Thyroid classification problem for second-order methods using Quasi-Newton formulation. The adaptive gain implementation in the proposed method in Quasi-Newton formulation has also increased the computational efficiency as compared to the traditional Quasi-Newton-based optimisation methods.



	trainbfg	BFGS	BFGS/AG	DFP	DFP/AG
Mean no. of Epochs	63	35	21	107	53
Standard deviation	78.64	3.68	4.24	83.84	84.71
CPU time (seconds)	8.74	9.65	5.30	29.93	12.67

Figure 4.3: Comparison of average CPU times and number of epochs required for convergence using the Quasi-Newton method for the Thyroid classification problem.

4.3.1.2 Wisconsin breast cancer classification problem

(ftp://ftp.ics.uci.edu/pub/machine-learning-databases/breast-cancer-wisconsin)

In this test the task is to predict malignancy from nine continuous clinical variables: clump thickness; uniformity of cell size; uniformity of cell shape; marginal adhesion; single epithelial cell size; bare nuclei; bland chromatin; normal nucleoli; and mitoses. The database consists of 699 patients, of which sixteen were eliminated due to missing values. Of the remaining cases, 239 were classified as having a malignancy.

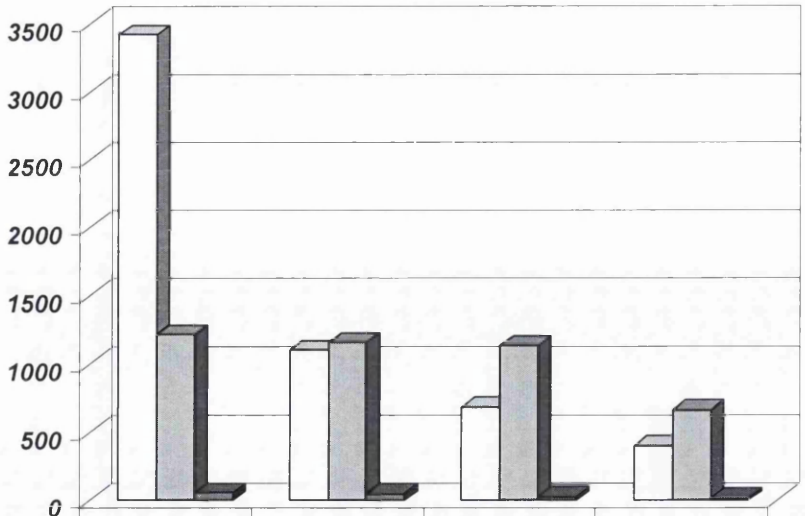
Dr. William H. Wolberg (O.L. Mangasarian and W.H. Wolberg, 1990) applied a multi-surface method of pattern separation, training on 246 of the 369 inputs available at that time, and obtained a 96 per cent test set predictive accuracy. The selected architecture of the Feed-forward Neural-network is 9-5-2. The target error is set as to 0.02 and the maximum epochs to a thousand.

With the cancer classification problem, the disparity between the convergence rate of first- and second-order methods is clearly illustrated in Table 4.3. The results clearly show that algorithms which implement the proposed method exhibit a very good average performance in order to reach target error.

	Cancer classification problem (target error=0.02)					
	Mean no. of epochs	**CPU time(s) per Epoch**	**Total CPU time(s) to converge**	**SD**	**Accuracy (%)**	**Fails**
traingdm	3419	$1.60\text{x}10^{-2}$	54.59	$1.22\text{x}10^{3}$	88.31	14
GDM	1105	$4.71\text{x}10^{-2}$	52.05	$1.16\ \text{x}10^{3}$	88.13	4
Ransing's method	690	$3.87\ \text{x}10^{-2}$	26.68	$1.14\ \text{x}10^{3}$	88.61	4
GDM/AG	405	$4.45\text{x}10^{-2}$	18.02	$6.64\ \text{x}10^{2}$	88.92	3
traincgf	71	$5.36\ \text{x}10^{-2}$	3.78	$5.35\ \text{x}10^{1}$	90.08	4
CGFR	65	$5.09\ \text{x}10^{-2}$	3.32	$4.03\ \text{x}10^{1}$	90.16	3
CGFR/AG	39	$3.98\ \text{x}10^{-2}$	1.55	$2.45\ \text{x}10^{1}$	90.37	2
traincgp	**29**	$\mathbf{9.83\text{x}10^{-2}}$	**2.82**	$\mathbf{4.07\ \text{x}10^{1}}$	**90.49**	**3**
CGPR	**33**	$\mathbf{4.77\text{x}10^{-2}}$	**1.56**	$\mathbf{1.51\ \text{x}10^{1}}$	**89.93**	**2**
CGPR/AG	**24**	$\mathbf{4.57\text{x}10^{-2}}$	**1.082**	$\mathbf{1.16\ \text{x}10^{1}}$	**90.38**	**2**
trainbfg	35	$7.00\text{x}10^{-2}$	2.46	$2.46\ \text{x}10^{1}$	88.92	2
BFGS	32	$5.37\text{x}10^{-2}$	1.72	$5.39\ \text{x}10^{0}$	88.88	1
BFGS/AG	29	$5.48\text{x}10^{-2}$	1.58	$7.66\ \text{x}10^{0}$	89.13	1
DFP	219	$8.31\text{x}10^{-2}$	18.16	$8.12\ \text{x}10^{1}$	88.77	3
DFP/AG	147	$8.00\text{x}10^{-2}$	11.79	$1.08\ \text{x}10^{2}$	88.34	2

Table 4.3: Summary of algorithms' performance for the cancer problem.

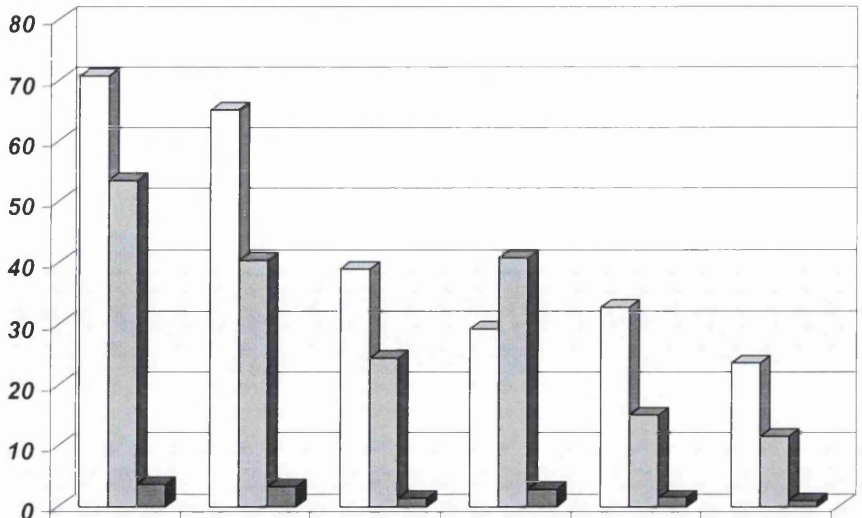
To underline the point, the performance of gradient-descent method is compared first in Figure 4.4. The proposed method (GDM/AG) took only 405 iterations to converge over one hundred simulation runs as compared to the neural-network toolbox (*traingdm*) with 3,419 iterations which is an improvement ratio of nearly 8.4. It shows that the proposed algorithm still outperforms other algorithms including the method proposed by Ransing (2002). Furthermore, the number of failure cases also indicated that the training efficiency of the gradient-descent method improved drastically by using the proposed method.



	traingdm	GDM	Ransing's method	GDM/AG
Mean on. of Epochs	3419	1105	690	405
Standard deviation	1215.62	1162	1137.59	663.53
CPU time (seconds)	54.59	52.05	26.68	18.02

Figure 4.4: Comparison of average CPU time and number of epochs required for convergence using the gradient-descent method for the cancer classification problem.

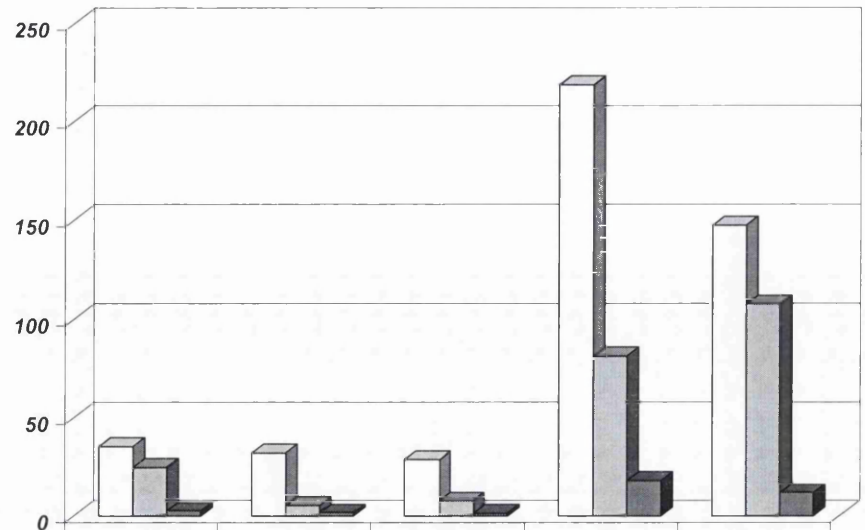
Figure 4.5 demonstrates the performance of the proposed method when implemented into the conjugate-gradient formulation. It shows that the proposed algorithm, particularly CGPR/AG, outperforms other algorithms as shown by the low mean value for epochs that was required for convergence. Yet for this problem CGPR/AG outperformed CGFR/AG with a mean of twenty-four, the lowest value standard deviation and two failure cases. Even though the number of failures is similar to the neural-network toolbox and the standard methods, the proposed method has performed better as compared to the standard method in terms of number of iterations and CPU time. Appendix II.2 demonstrates a detailed explanation of CGPR/AG performance.



	traincgf	CGFR	CGFR/AG	traincgp	CGPR	CGPR/AG
Mean no. of Epochs	71	65	39	29	33	24
Standard deviation	53.54	40.33	24.54	41	15.11	11.58
CPU time (seconds)	3.78	3.32	1.55	2.88	1.56	1.08

Figure 4.5: Comparison of average CPU time and number of epochs required for convergence using the conjugate-gradient method for the cancer classification problem.

Figure 4.6 illustrates the results for the cancer classification problem with various optimisation methods based on the Quasi-Newton formulation. As can be seen from the figure, the proposed algorithms (i.e. DFP/AG and BFGS/AG) show an improvement and still outperform other standard and neural-network algorithms as shown by a low mean value for epochs required for convergence.



	trainbfg	BFGS	BFGS/AG	DFP	DFP/AG
Mean on. Of Epochs	35	32	29	219	147
Standard deviation	25	5.39	7.66	81.23	107.69
CPU time (seconds)	2.46	1.72	1.58	18.16	11.79

Figure 4.6: Comparison of average CPU time and number of epochs required for convergence using the Quasi-Newton method for the cancer classification problem.

4.3.1.3 Diabetes classification problem

(ftp://ftp.ics.uci.edu/pub/machine-learning-databases/diabetes)

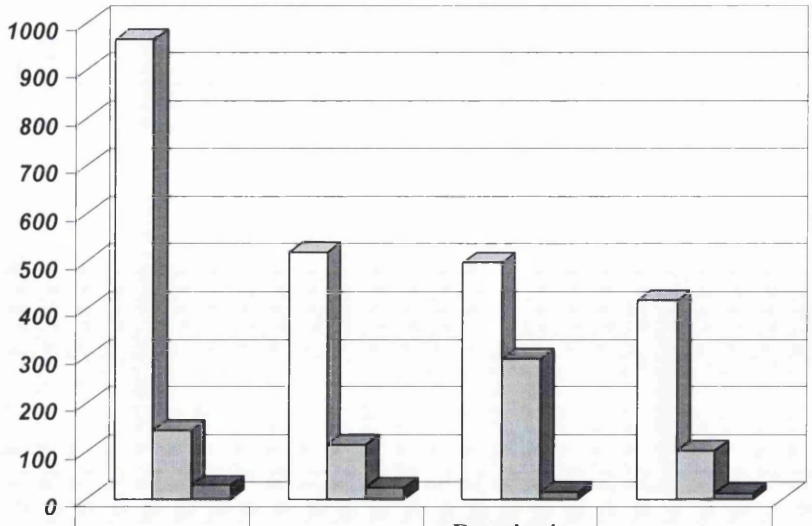
This dataset was created based on the 'Pima Indians diabetes' problem dataset from the UCI repository of machine-learning databases. The task is to decide whether a patient is diabetic or not, based on eight clinical variables, all continuous: age; diabetes pedigree function; body mass index; two-hour serum insulin level; triceps skin fold thickness; diastolic blood pressure; plasma glucose concentration; and number of pregnancies. Of the 768 patients, 268 are diabetic. The selected architecture of the Feed-forward Neural-network is 8-5-2. The target error is set to 0.01 and the maximum epochs to one thousand.

It is worth noticing in Table 4.4 that the performance of the proposed method with all gradient-based methods is substantially faster as compared to other standard methods.

	Diabetes classification problem (target error=0.01)					
	Mean no. of epochs	**CPU time(s) per Epoch**	**Total CPU time(s) to converge**	**SD**	**Accuracy (%)**	**Fails**
traingdm	965	3.14×10^{-2}	30.36	1.45×10^{2}	93.86	13
GDM	520	5.00×10^{-2}	25.97	1.14×10^{2}	89.09	5
Ransing's method	499	3.28×10^{-2}	16.36	2.96×10^{2}	90.78	4
GDM/AG	417	3.54×10^{-2}	14.76	1.02×10^{2}	89.10	4
traincgf	**98**	$\mathbf{4.11 \times 10^{-2}}$	**4.03**	$\mathbf{8.70 \times 10^{1}}$	**91.50**	**5**
CGFR	**51**	$\mathbf{5.16 \times 10^{-2}}$	**2.61**	$\mathbf{1.70 \times 10^{1}}$	**89.97**	**3**
CGFR/AG	**40**	$\mathbf{5.05 \times 10^{-2}}$	**2.00**	$\mathbf{1.59 \times 10^{1}}$	**90.70**	**4**
traincgp	46	7.12×10^{-2}	3.27	5.28×10^{1}	92.09	7
CGPR	54	4.73×10^{-2}	2.54	1.99×10^{1}	91.66	5
CGPR/AG	46	4.94×10^{-2}	2.26	1.59×10^{1}	91.24	4
trainbfg	106	3.90×10^{-2}	4.12	7.45×10^{1}	89.16	3
BFGS	95	4.93×10^{-2}	4.67	1.91×10^{1}	88.20	4
BFGS/AG	82	4.90×10^{-2}	4.01	1.89×10^{1}	87.61	3
DFP	401	6.04×10^{-2}	24.24	1.74×10^{2}	92.39	0
DFP/AG	309	6.06×10^{-2}	18.74	1.83×10^{2}	92.07	0

Table 4.4: Summary of algorithm performance for the diabetes problem.

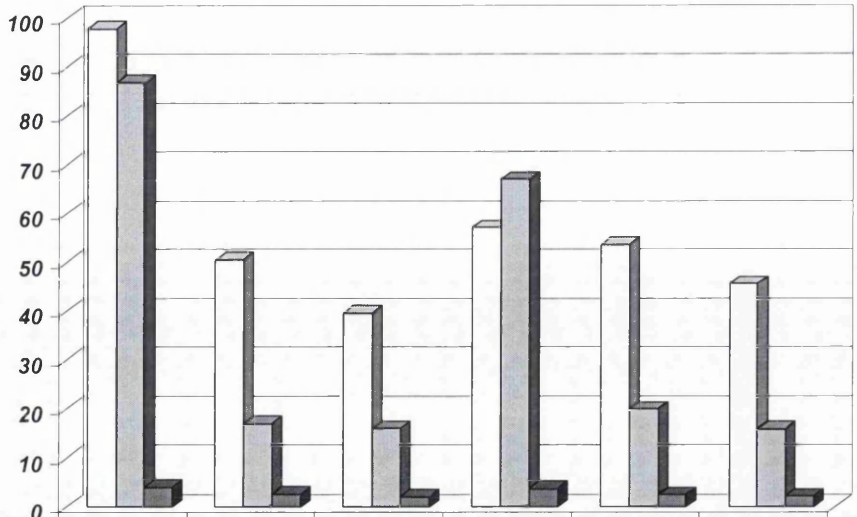
Figure 4.7 clearly shows that the proposed method had improved the training efficiency of gradient-descent methods by reducing the number of iterations and CPU time. It shows that the proposed algorithm (GDM/AG) is almost twice as fast as compared to the neural-network toolbox (*traingdm*) in achieving the target error of 0.01. Nevertheless with only four failure cases as compared to thirteen failure cases for the neural-network toolbox (*traingdm*), Figure 4.7 also shows that the proposed method (GDM/AG) gives better results with a lower number of epochs as compared to the method proposed by Ransing (2002). Furthermore, the proposed algorithm (GDM/AG) with the lowest standard deviation value yields more consistent results to reach the target error as compared to the neural-network toolbox.



	traingdm	GDM	Ransing's method	GDM/AG
Mean no. of Epochs	965	520	499	417
Standard deviation	145.33	114.14	295.54	102.23
CPU time (seconds)	30.36	25.97	16.36	14.76

Figure 4.7: Comparison of average CPU times and number of epochs required for convergence using the gradient-descent method for the diabetes classification problem.

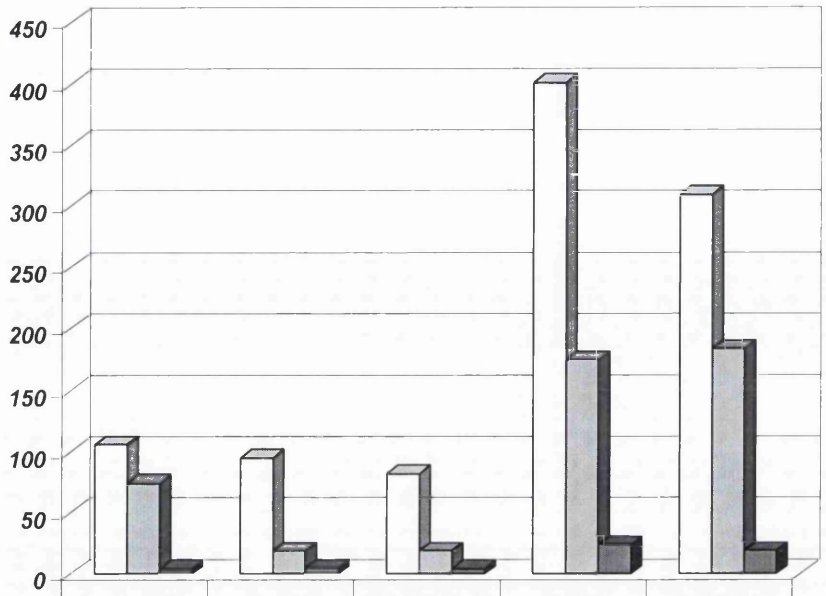
Figure 4.8 demonstrates the performance of the proposed method with the conjugate-gradient method. It shows that the proposed CGFR/AG algorithm took only forty epochs to reach the target error compared to CGFR at about fifty-one epochs and neural-network toolbox (*traincgf)* which needs about ninety-eight epochs to converge. The proposed algorithm also outperforms other algorithms in terms of the total CPU time to converge. The consistent performance of the proposed CGFR/AG algorithm is further shown by its lowest standard-deviation value as compared to other standard algorithms. The detailed explanation of the proposed CGFR/AG algorithm performance can be seen in Appendix II.3.



	traincgf	CGFR	CGFR/AG	traincgp	CGPR	CGPR/AG
Mean no. of Epochs	98	51	40	57	54	46
Standard deviation	86.99	17.03	15.94	67.06	19.90	15.90
CPU time (seconds)	4.03	2.61	2.01	3.73	2.54	2.26

Figure 4.8: Comparison of average CPU time and number of epochs required for convergence using the conjugate-gradient method for the diabetes classification problem.

Figure 4.9 shows that the proposed method implemented with the Quasi-Newton optimisation formulation had significantly improved the training efficiency in terms of number of iterations and CPU time. A longer learning time is required for the diabetes problem than the previous two problems, particularly for the quasi-Newton methods. The mean convergence ranged for Quasi-Newton methods are from eighty to four hundred iterations. Even though all implementations of the proposed methods have outperformed other algorithms, for this problem, the proposed BFGS/AG algorithm produced the best results with eighty-two iterations. The neural-network toolbox algorithm (*trainbfg*) took 1.29 seconds longer to learn than the proposed BFGS/AG algorithm. In addition, the proposed method performs more consistent results to reach the target error as compared to the neural-network toolbox as shown by the value of standard deviation in Figure 4.9.



	trainbfg	BFGS	BFGS/AG	DFP	DFP/AG
Mean no. of Epochs	106	95	82	401	309
Standard deviation	74.46	19.08	18.90	174.36	183.30
CPU time (seconds)	4.12	4.67	4.01	24.24	18.74

Figure 4.9: Comparison of average CPU time and number of epochs required for convergence using the Quasi-Newton method for the diabetes classification problem.

4.3.1.4 IRIS classification problem

(ftp://ftp.ics.uci.edu/pub/machine-learning-databases/iris/iris.data)

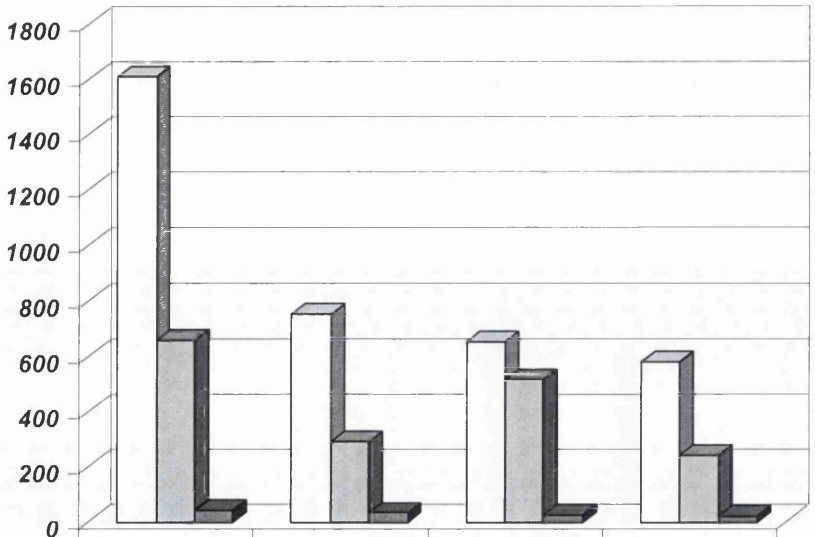
This classical classification data set was made famous by Fisher (R.A. Fisher, 1936), who used it to illustrate principles of discriminant analysis. This is perhaps the best-known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. The selected architecture of the Feed-forward Neural-network is 4-5-3 with a target error set as 0.05 and the maximum epochs to one thousand.

Table 4.5 shows that using the proposed method with second-order methods presents substantial improvements over the first-order methods in terms of the CPU time and number of iterations. The proposed formulation still outperforms other algorithms in terms of CPU time and number of epochs.

	IRIS classification problem (target error=0.05)					
	Mean no. of epochs	**CPU time(s) per Epoch**	**Total CPU time(s) to converge**	**SD**	**Accuracy (%)**	**Fails**
traingdm	1609	2.69×10^{-2}	43.30	6.58×10^{2}	94.01	15
GDM	754	3.89×10^{-2}	29.28	2.94×10^{2}	94.33	4
Ransing's method	653	4.09×10^{-2}	26.70	5.18×10^{2}	95.35	3
GDM/AG	581	3.69×10^{-2}	21.42	2.43×10^{2}	94.45	3
traincgf	70	5.45×10^{-2}	3.83	7.41×10^{1}	96.52	6
CGFR	39	4.91×10^{-2}	1.93	3.25×10^{1}	96.65	2
CGFR/AG	29	4.93×10^{-2}	1.42	7.00×10^{0}	96.07	2
traincgp	**39**	**7.67×10^{-2}**	**2.99**	**6.31×10^{1}**	**96.28**	**10**
CGPR	**28**	**4.54×10^{-2}**	**1.25**	**1.63×10^{1}**	**97.77**	**4**
CGPR/AG	**23**	**4.55×10^{-2}**	**1.06**	**1.19×10^{1}**	**97.73**	**3**
trainbfg	51	5.20×10^{-2}	2.64	2.54×10^{1}	93.26	2
BFGS	63	4.76×10^{-2}	3.01	4.61×10^{0}	95.85	1
BFGS/AG	40	4.57×10^{-2}	1.83	4.26×10^{0}	95.90	0
DFP	599	6.45×10^{-2}	38.67	2.58×10^{2}	94.55	5
DFP/AG	534	5.90×10^{-2}	31.49	4.39×10^{2}	94.31	3

Table 4.5: Summary of algorithm performances for the IRIS problem.

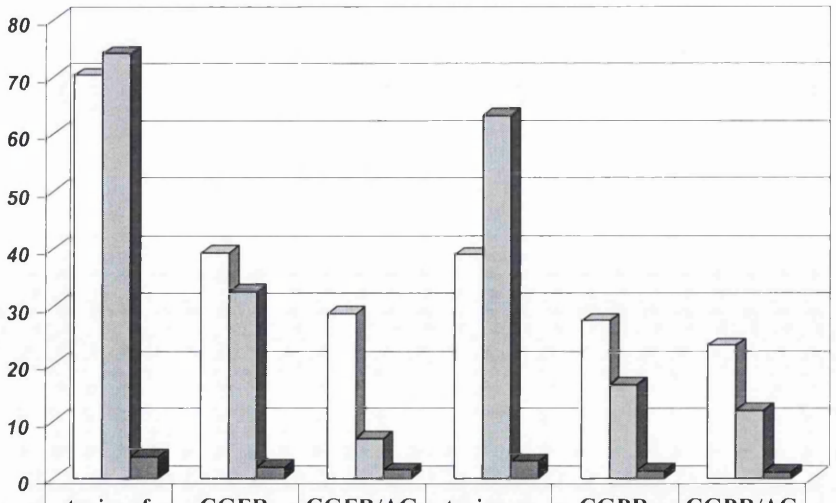
Figure 4.10 illustrates the performance of the gradient-descent method coupled with the proposed method. The proposed method clearly shows a better result in terms of number of iterations required to converge. The proposed method is three times faster as compared to the neural-network toolbox ('*traingdm*'). Moreover, the proposed algorithm (GDM/AG) gave more consistent results with a lower value of standard deviation as compared to other algorithms.



	traingdm	GDM	Ransing's method	GDM/AG
Mean no. of Epochs	1609	754	653	581
Standard deviation	658.22	294.28	518.24	242.57
CPU time (seconds)	43.30	34.59	26.70	21.42

Figure 4.10: Comparison of average CPU times and number of epochs required for convergence using the gradient-descent method for the IRIS classification problem.

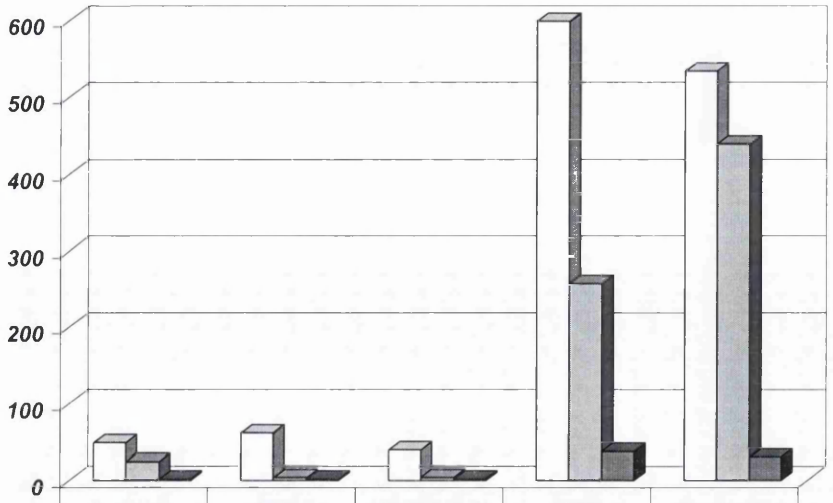
As for second-order methods with conjugate-gradient formulation, the proposed algorithm as illustrated in Figure 4.11 shows better results as compared to other standard algorithms. Even though the proposed algorithm (CGFR/AG) had only two failure cases as compared to CGPR/AG with three failure cases, yet CGPR/AG is faster. The proposed algorithm (CGPR/AG) needs only twenty-three epochs to converge as compared to CGFR/AG with twenty-nine epochs. This makes the proposed algorithm (CGPR/AG) a better choice for this problem. As far as the consistency of the performance is concerned, it is clear that the neural-network toolbox implementation failed to converge at certain trials and showed a higher standard deviation value which indicates that the neural-network toolbox implementation are unstable as compared to the proposed method (Figure 4.11). The detailed explanation on the proposed algorithm (CGPR/AG) performance is given in Appendix II.4.



	traincgf	CGFR	CGFR/AG	traincgp	CGPR	CGPR/AG
Mean no. of Epochs	70	39	29	39	28	23
Standard deviation	74.08	32.51	7.00	63.10	16.33	11.97
CPU time (seconds)	3.83	1.93	1.42	2.99	1.25	1.06

Figure 4.11: Comparison of average CPU times and number of epochs required for convergence using the conjugate-gradient method for the IRIS classification problem.

The performance of the proposed method with Quasi–Newton formulation is shown in Figure 4.12. The proposed algorithm (BFGS/AG) outperforms neural-network toolbox '*trainbfg*' with an improvement ratio of nearly 1.4, for the total CPU time. Furthermore, the proposed algorithm (BFGS/AG) needs only forty epochs to reach the target error as compared to standard BFGS with sixty-three epochs and neural-network toolbox '*trainbfg*' with fifty-one epochs. The proposed method significantly reduced the number of epochs required for the convergence.



	trainbfg	BFGS	BFGS/AG	DFP	DFP/AG
Mean no. of Epochs	51	63	40	599	534
Standard deviation	25.41	4.61	4.26	257.76	438.62
CPU time (seconds)	2.64	3.01	1.83	38.67	31.49

Figure 4.12: Comparison of average CPU times and number of epochs required for convergence using the Quasi-Newton method for the IRIS classification problem.

4.3.1.5 Seven-bit parity problem

(Training data: http://homepages.cae.wisc.edu/~ece539/data/parity7r)

(Testing data: http://homepages.cae.wisc.edu/~ece539/data/parity7t)

N-bit parity is the name given to a set of binary problems that are widely used in benchmark-training tests for neural-network algorithms. The N-bit parity training set consist of 2^N training pairs, with each training pair comprising an N-length input vector and a single binary target value. The 2^N input vectors represent all possible combinations of N binary numbers. Basically if a given input vector contains an odd number of ones, the corresponding target value is 1, otherwise the target value is 0. The parity problem is one of the most popular initial testing tasks and very demanding classification problem for neural networks to solve. This is because the target-output changes whenever a single bit in the input vector changes and this makes generalisation difficult and learning does not always converge easily (Erik Hjelmas and P.W. Munro, 1999). The selected architecture of the Feed-forward Neural-network is 7-5-1. The target error has been set to 0.05.

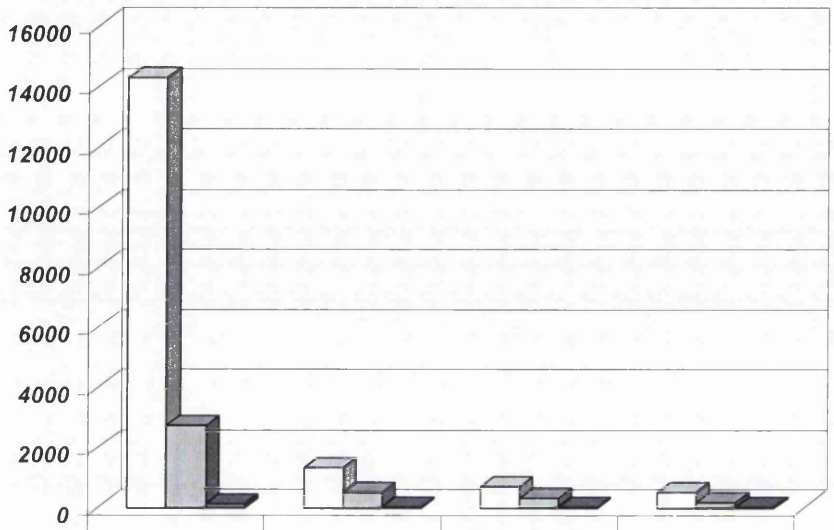
It is very important to recognise that there is a fundamental distinction between the causes of training failure recorded in Table 4.6. Convergence to a stationary point (typically a local minimum) is the main cause of training failure cases with first- or second-order methods (i.e. *traingdm*, GDM, DFP and DFP/AG). However, the high failure rate for first-order methods when applied to this problem is the result of a failure to converge to any stationary point within the allowed number of training epochs, in this case maximum epochs for the first-order method is twenty thousand. The implementations of the proposed method into first- and second-order methods greatly decreases the number of failure cases to converge and also decreases the number of epochs required as well as CPU time. The detailed explanation of the proposed BFGS/AG algorithm performance is demonstrated in Appendix II.5.

	7 bit parity problem (target error=0.05)					
	Mean no. of epochs	**CPU time(s) per Epoch**	**Total CPU time(s) to converge**	**SD**	**Accuracy (%)**	**Fails**
Traingdm	14272	9.81×10^{-3}	140.02	2.72×10^{3}	87.62	12
GDM	1347	3.80×10^{-2}	51.15	5.09×10^{2}	87.54	7
Ransing's method	717	3.35×10^{-2}	24.04	3.07×10^{2}	88.59	3
GDM/AG	537	3.99×10^{-2}	21.42	1.83×10^{2}	88.75	4
Traincgf	283	3.87×10^{-2}	10.97	3.32×10^{2}	90.98	5
CGFR	148	7.27×10^{-2}	10.74	1.07×10^{2}	91.12	3
CGFR/AG	114	7.11×10^{-2}	8.11	6.53×10^{1}	90.34	2
Traincgp	143	3.46×10^{-2}	4.94	1.41×10^{2}	90.19	3
CGPR	129	6.15×10^{-2}	7.92	3.32×10^{1}	91.23	2
CGPR/AG	92	5.28×10^{-2}	4.86	1.37×10^{1}	90.57	1
Trainbfg	**166**	$\mathbf{2.70 \times 10^{-2}}$	**4.50**	$\mathbf{8.69 \times 10^{1}}$	**88.74**	**3**
BFGS	**93**	$\mathbf{4.31 \times 10^{-2}}$	**4.02**	$\mathbf{3.83 \times 10^{0}}$	**90.43**	**2**
BFGS/AG	**85**	$\mathbf{4.39 \times 10^{-2}}$	**3.73**	$\mathbf{3.01 \times 10^{0}}$	**90.66**	**1**
DFP	525	5.97×10^{-2}	31.35	1.32×10^{2}	90.11	10
DFP/AG	326	5.97×10^{-2}	19.48	1.19×10^{2}	90.15	10

Table 4.6: Summary of algorithm performance for the Seven-bit parity problem.

Figure 4.13 displays a summary of the Seven-bit parity classification problem for gradient-descent methods. It shows that training with the neural-network toolbox (*traingdm*) took 14,272 iterations to reach the target error with sixteen failure cases. The proposed GDM/AG reduces the number of iterations almost nine times with only

four failure cases and clearly shows that the proposed method outperforms the neural-network toolbox. Even though the method introduced by Ransing (2002) decreased the number of epochs significantly as compared to the standard gradient-descent method, the proposed GDM/AG algorithm again outperformed with the lowest standard deviation value of 183.15.

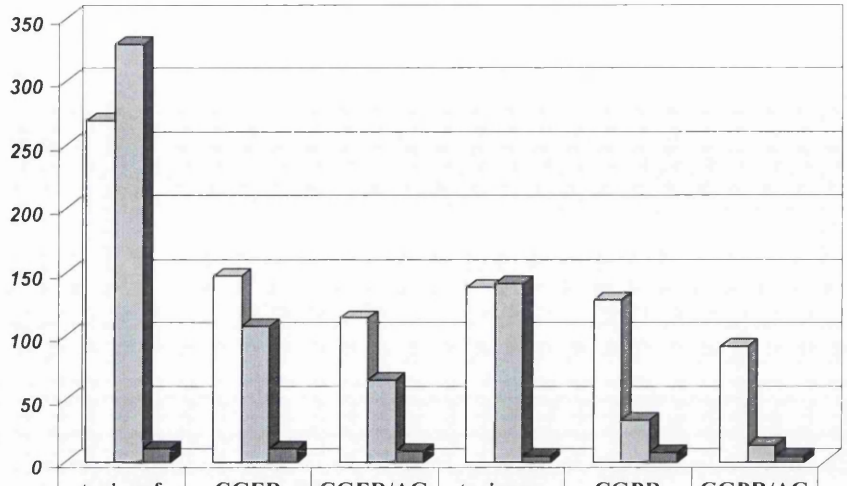


	traingdm	GDM	Ransing's method	GDM/AG
Mean no. of Epochs	14272	1347	717	537
Standard deviation	2719.62	509.08	306.58	183.15
CPU time (seconds)	140.02	51.15	24.04	21.42

Figure 4.13: Comparison of average CPU time and number of epochs required for convergence using the gradient-descent method for the Seven-bit parity problem.

In Figure 4.14, the proposed algorithm shows better results as compared with the conjugate gradient formulation because it converges in a smaller number of epochs as observed by a low value of the mean. The proposed algorithm (CGFR/AG) had two failure cases as compared to the CGPR/AG algorithm with one failure case. This makes the proposed algorithm (CGPR/AG) a better choice for this problem since it had only one failure case for the one hundred simulation runs. The proposed (CGPR/AG) algorithm not only decreases the number of iterations to converge but also it is more consistent as compared to other standard algorithms. Figure 4.14 clearly shows that within a hundred simulation runs of training the proposed method

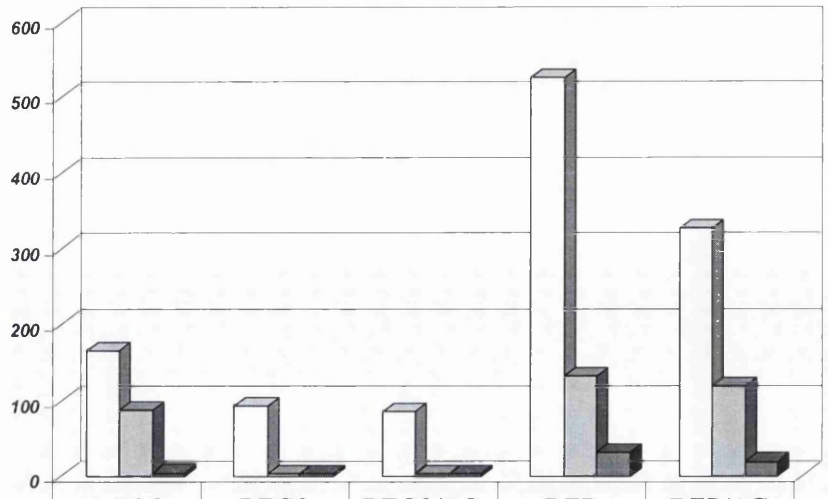
is stable and achieves a low standard deviation value of 13.66 as compared to other algorithms.



	traincgf	CGFR	CGFR/AG	traincgp	CGPR	CGPR/AG
Mean no. of Epochs	269	148	114	139	129	92
Standard deviation	329.09	107.48	65.31	141.30	33.21	13.66
CPU time (seconds)	10.42	10.74	8.78	4.79	7.92	4.86

Figure 4.14: Comparison of average CPU time and number of epochs required for convergence using the conjugate-gradient method for the Seven-bit parity problem.

Figure 4.15 illustrates the performance between the proposed method and neural-networks toolbox using Quasi-Newton formulation. It shows that gradient-based methods with the proposed implementation are substantially faster than the traditional gradient-based methods.



	trainbfg	BFGS	BFGS/AG	DFP	DFP/AG
Mean no. of Epochs	166	93	85	525	326
Standard deviation	86.89	3.83	3.01	131.68	119.13
CPU time (seconds)	4.50	4.02	3.76	31.35	19.48

Figure 4.15: Comparison of average CPU time and number of epochs required for convergence using the Quasi-Newton method for Seven-bit parity problem.

4.3.1.6 Glass-classification problem

(ftp://ftp.ics.uci.edu/pub/machine-learning-databases/glass)

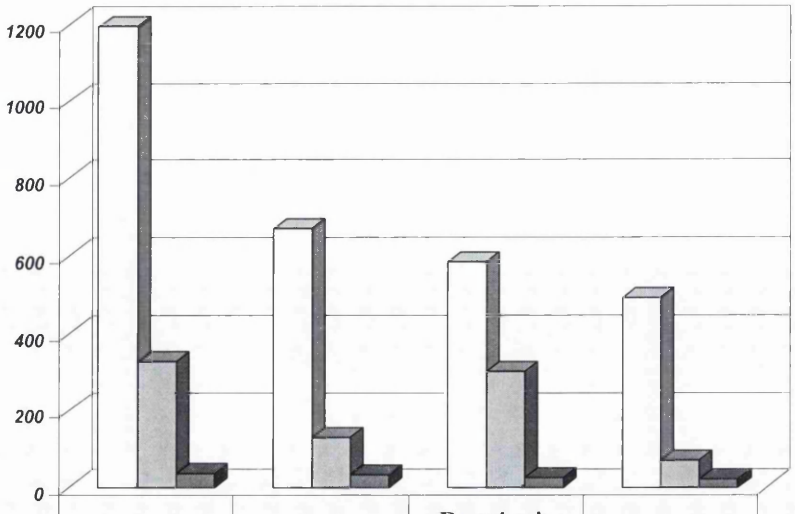
This data set aims to classify glass type and was created based on the 'glass' problem data set from the UCI repository or machine-learning databases. The results of chemical analysis of glass splinters (percentage content of eight different elements) plus the refractive index are used to classify the sample to be either float-processed or non-float-processed building windows, vehicle windows, containers, tableware or head lamps. This task was motivated by forensic needs in criminal investigations. The selected architecture of the Feed-forward Neural-network is 9-5-6. The target error is set to 0.01 and the maximum epochs to a thousand.

The results presented in Table 4.7 illustrate that all gradient-based methods coupled with the proposed method outperform other classical gradient-based methods. The different rates of convergence associated with different classes of gradient-based methods, and with the proposed method, are clearly noticeable.

	Glass classification problem (target error=0.01)					
	Mean no. of epochs	**CPU time(s) per Epoch**	**Total CPU time(s) to converge**	**SD**	**Accuracy (%)**	**Fails**
traingdm	1194	3.10×10^{-2}	36.98	3.27×10^{2}	93.97	9
GDM	670	4.66×10^{-2}	31.19	1.29×10^{2}	92.91	4
Ransing's method	583	3.98E-02	23.1922	301.05	92.64	3
GDM/AG	491	4.16×10^{-2}	20.40	6.94×10^{1}	93.03	3
traincgf	280	2.35×10^{-2}	6.58	2.06×10^{2}	93.39	6
CGFR	108	8.90×10^{-2}	9.60	1.31×10^{2}	92.27	3
CGFR/AG	73	7.74×10^{-2}	5.66	7.05×10^{1}	92.45	4
traincgp	75	3.79×10^{-2}	2.86	5.54×10^{1}	93.33	4
CGPR	36	6.35×10^{-2}	2.31	1.01×10^{1}	92.72	2
CGPR/AG	31	5.62×10^{-2}	1.74	1.34×10^{1}	93.55	3
trainbfg	**117**	**3.41×10^{-2}**	**3.99**	**4.07×10^{1}**	**93.22**	**6**
BFGS	**27**	**8.60×10^{-2}**	**2.34**	**1.19×10^{1}**	**92.61**	**4**
BFGS/AG	**15**	**6.07×10^{-2}**	**0.93**	**4.32×10^{0}**	**93.25**	**5**
DFP	715	7.09×10^{-2}	50.67	4.20×10^{2}	90.60	2
DFP/AG	409	6.83×10^{-2}	27.92	2.64×10^{2}	90.53	1

Table 4.7: Summary of algorithm performance for the Glass-classification problem

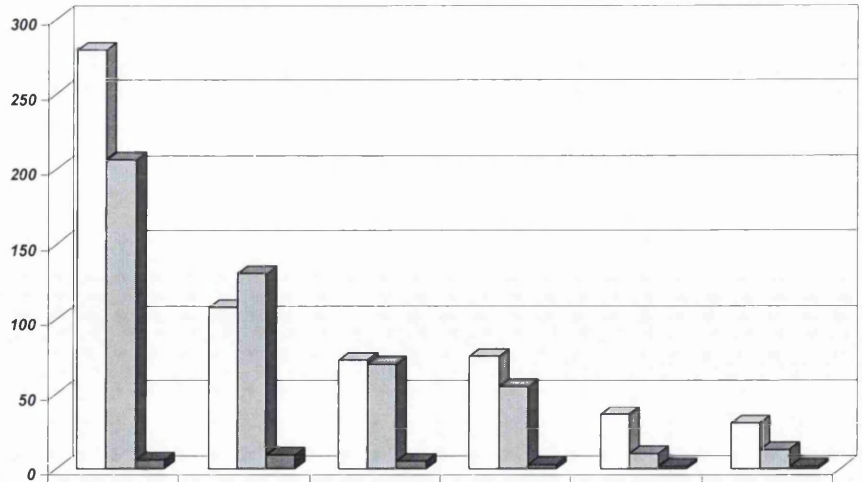
Figure 4.16 below compares the performance of the proposed method with the gradient-descent method. It shows that the proposed method is substantially faster than the traditional gradient-descent method and particularly the neural-network toolbox (*traingdm*) implementation. As can be noticed, the proposed method allows the learning algorithm to obtain a faster convergence speed. Although, the method introduced by Ransing also presents a good performance, that is much better than the standard GDM, but it is not consistent as the value of standard deviation is high as compared to the proposed method (GDM/AG).



	traingdm	GDM	Ransing's method	GDM/AG
Mean no. of Epochs	1194	670	583	491
Standard deviation	326.98	129.98	301.05	69.44
CPU time (seconds)	36.98	31.19	23.19	20.40

Figure 4.16: Comparison of average CPU time and number of epochs required for convergence using the gradient-descent method for the Glass-classification problem.

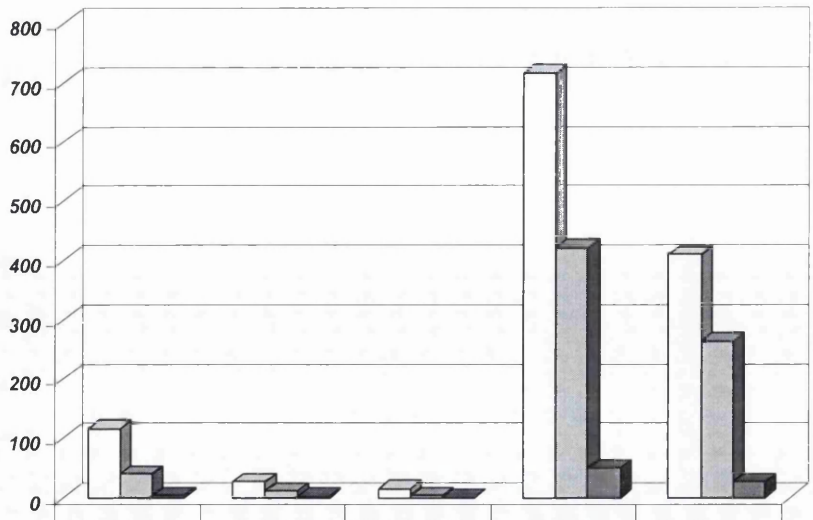
Figure 4.17 shows the comparison of average CPU time and number of epochs required for convergence using conjugate-gradient methods. The proposed formulation outperforms other standard algorithms. Among all methods the proposed method (CGPR/AG) performs the best with only thirty-one epochs and 1.74 seconds CPU time is required for convergence. The method presented by the neural-network toolbox '*traincgf*' yields very poor results as it requires the highest number of epochs with the highest standard deviation value which also indicates that the method is unstable in reaching the target value within a hundred simulation runs.



	traincgf	CGFR	CGFR/AG	traincgp	CGPR	CGPR/AG
Mean no. of Epochs	280	108	73	75	36	31
Standard deviation	206.27	131.06	70.49	55.38	10.14	13.36
CPU time (seconds)	6.58	9.60	5.66	2.86	2.31	1.74

Figure 4.17: Comparison of average CPU time and number of epochs required for convergence using the conjugate-gradient method for the Glass-classification problem.

The average CPU time and number of epochs for Quasi-Newton methods are plotted in Figure 4.18. From the figures the overall performance of the proposed method in achieving the target error is better as compared to other standard algorithms. As shown in the figure the proposed method (BFGS/AG) took only fifteen epochs to converge as compared to the neural-network toolbox (*trainbfg*) that required 117 epochs which is an improvement ratio of nearly 7.8 (refer to Appendix II.6 for a detailed explanation on BFGS/AG performance). Even though the number of epochs required by the proposed method DFP/AG is quite high as compared to BFGS/AG, it shows an improvement on its standard algorithm (DFP) with a more consistent performance (as shown by lower standard deviation value). This is attributed to the improvements of the search direction introduced by the proposed method.



	trainbfg	BFGS	BFGS/AG	DFP	DFP/AG
Mean no. of Epochs	117	27	15	715	409
Standard deviation	40.70	11.90	4.32	420.27	263.61
CPU time (seconds)	3.99	2.33	0.93	50.67	27.92

Figure 4.18: Comparison of average CPU time and number of epochs required for convergence using the Quasi-Newton method for the Glass-classification problem.

4.4 CONCLUSION

The novel method proposed in Chapter Three that improved the gradient-based search direction was validated in this chapter for its efficiency and accuracy on a variety of benchmark problems. The performance of the proposed method is validated in two ways: (a) the speed of convergence measured in the number of iterations and CPU time; and (b) the generalisation accuracy on testing data from the benchmark problems. The results were discussed and it showed that the proposed algorithm is generic, robust and easy to implement into all commonly used gradient-based optimisation processes. Furthermore, the results clearly showed that the proposed algorithm has significantly improved the neural-network training performance as compared to the existing methods including the one proposed by Ransing (2002).

CHAPTER 5

ENHANCEMENT TO THE METHOD PROPOSED FOR CONSTRUCTING OPTIMAL KNOWLEDGE HYPER-SURFACE

CHAPTER LAYOUT

This chapter is a rational attempt to carry forward the research task outlined in Section 1.3 of Chapter One. The first section of this chapter discusses the implementation of Lagrange Interpolation Polynomials into the current Knowledge Hyper-surface method and highlights important features of the current method in learning from examples. The limitations of using the current method are described next and the need for further improvement is identified. The next section describes the enhancements proposed to the current method by incorporating midpoints in the existing shape formulation. The results are compared with the neural-network method proposed in Chapter Four and the current Knowledge Hyper-surface method. The conclusions are then drawn in the last section to identify scope for further research.

5.1 INTRODUCTION

The research on the analysis of cause and effect relationships in castings has always been a centre of attention in the manufacturing industry. An intelligent diagnosis system should be able to diagnose effectively the causal representation and also justify its diagnosis. A previous method, known as the Knowledge Hyper-surface method, proposed by Ransing (Meghana R. Ransing, 2002), used Lagrange Interpolation polynomials to show that the belief value of the occurrence of cause with respect to the change in the belief value in the occurrence of effect can be modelled by linear, quadratic or cubic relationships. The current method retained the advantages of neural networks and overcomes their limitations in learning the input-output mapping function in the presence of noisy, limited and sparse data. The ability of the current method to constrain the belief values in the causes made it a better technique as compared to the multi-layer neural-network method. The comparison was done on real casting data (Meghana R. Ransing, 2002).

However, during the research work that is presented in this chapter, it was discovered that the methodology was unable to model exponential increase/decrease in belief values in cause and effect relationships. This chapter proposes an enhancement to the current Knowledge Hyper-surface method by introducing midpoints in the existing shape formulation which further constrains the shape of the Knowledge hyper-surfaces to model an exponential rise in belief values but without exposing the dataset to the limitations of 'overfitting'.

The first section starts with the overview of the current method which discusses the implementation of Lagrange Interpolation Polynomials into the current Knowledge Hyper-surface method and highlights important features of the current method in learning from examples. The limitations of using the current method are described next in Section 5.4. The enhancements and mathematical solution is presented in Section 5.5. The performance of the proposed method is further illustrated in Section 5.6 by comparing its solution with the previous version of the Knowledge Hyper-surface method and the neural-networks method on real casting data. Finally, the conclusions are drawn from the research presented in this chapter.

5.2 THE CURRENT KNOWLEDGE HYPER-SURFACE METHOD

Ransing (2002) proposed a method that retains advantages of regression analysis and neural-network techniques and at the same time overcomes the limitations of both techniques. The Knowledge Hyper-surface method described that the belief variation in the occurrence of a cause, with respect to a change in the belief value of the occurrence of an effect, follows a pattern. Such a variation is generally linear, quadratic or cubic and certainly not an arbitrary higher-ordered polynomial.

The method described that to model an n^{th} order relationship along a dimension, $(n+1)$ equidistant reference points between -1 and +1 are chosen. For each reference point 'i' ($i=1$ to n+1), a one-dimensional Lagrange Interpolation Polynomial is used based on the following formula:

$$l_i(\xi) = l_k^n(\xi)$$
$$= \frac{\xi-\xi_0}{\xi_k-\xi_0} * \frac{\xi-\xi_1}{\xi_k-\xi_1} * \frac{\xi-\xi_2}{\xi_k-\xi_2} * \cdots * \frac{\xi-\xi_{k-1}}{\xi_k-\xi_{k-1}} * \frac{\xi-\xi_{k+1}}{\xi_k-\xi_{k+1}} * \cdots \frac{\xi-\xi_n}{\xi_k-\xi_n} \quad (5.1)$$

where:

n : Order of the Lagrange Interpolation Polynomial (e.g. one for linear; two for quadratic; three for cubic; etc.)

k : A reference point at which the one-dimensional Lagrange Interpolation Polynomial $l_k^n(\xi)$ is constructed (k ranges from 0 to n).

i : Ranges from one to total number of reference points, i.e. $(n+1)$.

The variable ξ is used to store the belief value representing the strength of the corresponding effects, ranges from -1 to +1. For one-dimensional Lagrange Polynomial Interpolation the reference points are drawn along this dimension. Whereas for a given cause connected to ' p ' effects, the Lagrange

Interpolation Polynomial at a reference point 'i' is defined as 'p' dimensional and is given by the following equation:

$$l_i(\xi^1,\xi^2,\xi^3,...,\xi^j,...,\xi^p) = l_{k_1}^{n_1}(\xi^1) * l_{k_2}^{n_2}(\xi^2) * ... * l_{k_j}^{n_j}(\xi^j) * ... l_{k_p}^{n_p}(\xi^p) \quad (5.2)$$

where:

$$l_{k_j}^{n_j}(\xi^j) = \frac{\xi^j - \xi_0^j}{\xi_{k_j}^j - \xi_0^j} * \frac{\xi^j - \xi_1^j}{\xi_{k_j}^j - \xi_1^j} * ... * \frac{\xi^j - \xi_{k_j-1}^j}{\xi_{k_j}^j - \xi_{k_j-1}^j} * \frac{\xi^j - \xi_{k_j+1}^j}{\xi_{k_j}^j - \xi_{k_j+1}^j} * ... \frac{\xi^j - \xi_{n_j}^j}{\xi_{k_j}^j - \xi_{n_j}^j} \quad (5.3)$$

n_j : The order of one dimensional Lagrange Interpolation Polynomial $(l_{k_j}^{n_k}(\xi^j))$ corresponding to j^{th} dimension that represents the relationship between j^{th} effect and the cause under consideration.

k_j : Reference point along j^{th} dimension, at which the one-dimensional Lagrange Interpolation Polynomial $l_{k_j}^{n_k}(\xi^j)$ is evaluated. (k_j Independently ranges from 0 to n_j for each Lagrange Polynomial Interpolation).

$\xi_0^j, \xi_1^j, \xi_2^j, ..., \xi_{n_j}^j$ are $(n_j + 1)$ reference points along the j^{th} dimension.

i : for a 'p' dimensional case, 'i' ranges from one to the total number of reference points 'q' as given below:

$$q = (n_1 + 1) * (n_2 + 1) * (n_3 + 1) * ... * (n_j + 1) * ... * (n_p + 1) \quad (5.4)$$

The method also prescribed that a Lagrange Interpolation polynomial and a weight value can be associated with each of the said reference points as shown by the equation below:

$$\text{The belief value in the cause} = \sum_{i=1}^{q} w_i l_i(\xi^1,\xi^2,...,\xi^p) \quad (5.5)$$

where:

q : Total number of reference points.

$l_i(\xi^1,\xi^2,...,\xi^p)$ is given by Equation 2.24

w_i : Weight variable associated with the i^{th} reference point.

By considering a weight value at a reference point to be representative of the belief value in the cause, the total number of weights is therefore the same as the total number of reference points. However, this formulation had its own limitation. As the number of dimensions increased, the total number of weights in a network also increased exponentially. This rapidly increased the number of unknown variables within the network and it was not a practical implementation, as it would not only slow down the system, but also requires an excessively large training dataset.

In order to overcome that limitation, Ransing (2002) divided the reference points into two categories, referred to as primary and secondary reference points. Weight values associated with these primary reference points have been considered as independent variables (primary weight values) and other weight values associated with secondary reference points (secondary weight values), have been considered to be linearly dependent on one or more primary weight values (see Figure 5.1).

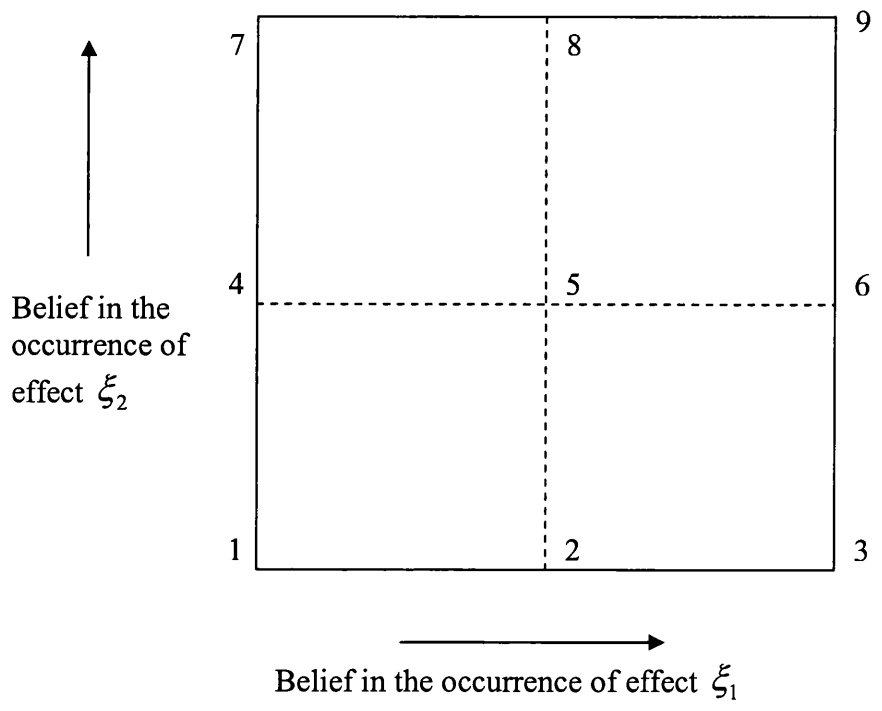


Figure 5.1: Dependent (1,2,3,4,7) and Independent Weight Values (5,6,8,9) associated with Reference Points 1 to 9.

For a ' p ' dimensional problem, the total number of primary weights is calculated as:

$$\text{Primary weights} = \left[\left(\sum_{j=1}^{p} n_j + 1\right) - (p-1)\right] \tag{5.6}$$

As we can see from Figure 5.1, weights associated with primary reference points 1, 2, 3, 4 and 7 are primary weights. The secondary weight values at locations 5, 6, 8 and 9 are expressed as a linear combination of the primary weights and in particular:

$$w_5 = \frac{c(w_2 + w_4)}{2} \tag{5.7}$$

$$w_6 = \frac{c(w_3 + w_4)}{2} \tag{5.8}$$

$$w_8 = \frac{c(w_2 + w_7)}{2} \tag{5.9}$$

$$w_9 = \frac{c(w_3 + w_7)}{2} \tag{5.10}$$

5.2.1 Analogy of the belief variation in a cause and effect relationship for the medical field

An analogy has been sought in the medical field to further illustrate the belief-variation concept (Meghana R. Ransing, 2002). Generally medical experts characterise the strength of a symptom by adjectives e.g. *low, medium, high* and *very high* fever. Similarly the belief in the proposition that 'typhoid is a cause for a symptom fever' is also characterised as *low, medium, high* and *very high.* Such belief variations in a one- dimensional cause and effect relationship can be graphically plotted as shown in Figure 5.2. These relationships are easy to visualise in one dimension, however, as the number of symptoms increase the shape of the resulting hyper-surface becomes much more complex. For a one-dimensional example, as shown in Figures 5.5 and 5.6, if the chest pain is *very high* then the belief that the patient is suffering from a 'heart attack' is *high* and the belief that the patient has a 'hair fracture or small muscular twist in the chest' is *low.*

The output value in the following Figures (5.3-5.7) represents a belief value in the occurrence of the cause corresponding to the strength of the effect. These figures show graphical representation of some general one-dimensional cause and effect relationship as shown in Figure 5.2.

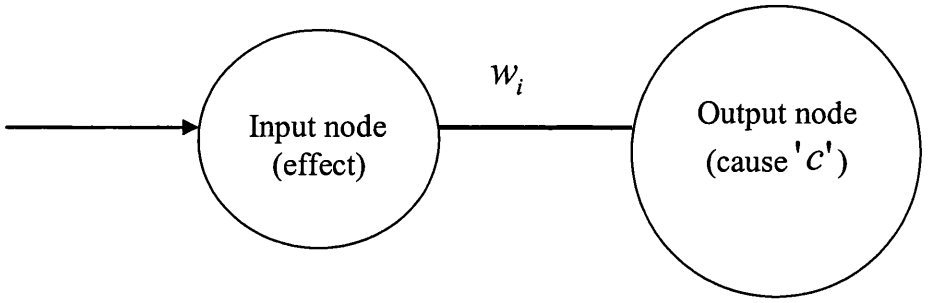


Figure 5.2: Schematic representation of a 'single effect – cause relationship'.

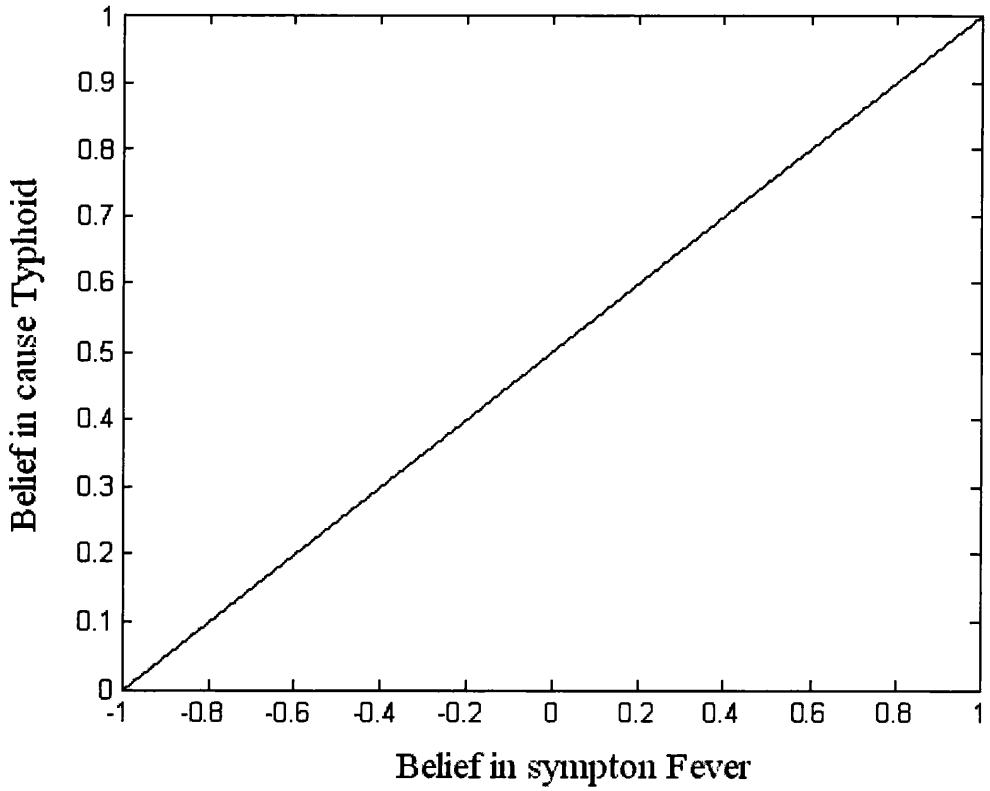


Figure 5.3: Belief that 'Typhoid' is a cause for a symptom 'Fever'.

The belief value quantifying the occurrence, or non-occurrence, of an effect associated with a particular cause is normalised between +1 to -1, respectively, and the belief value, which quantifies the extent of occurrence of the cause under consideration, is also normalised from zero to unity.

Figure 5.3 showed that the linear variation in belief values, which when the belief value representing the strength of the symptom 'Fever' is at its minimum, the belief that 'Typhoid' is a cause for a symptom 'Fever' is also at its minimum. The belief value in the occurrence of the cause is linearly increased as the strength of the symptom or effect start to increase.

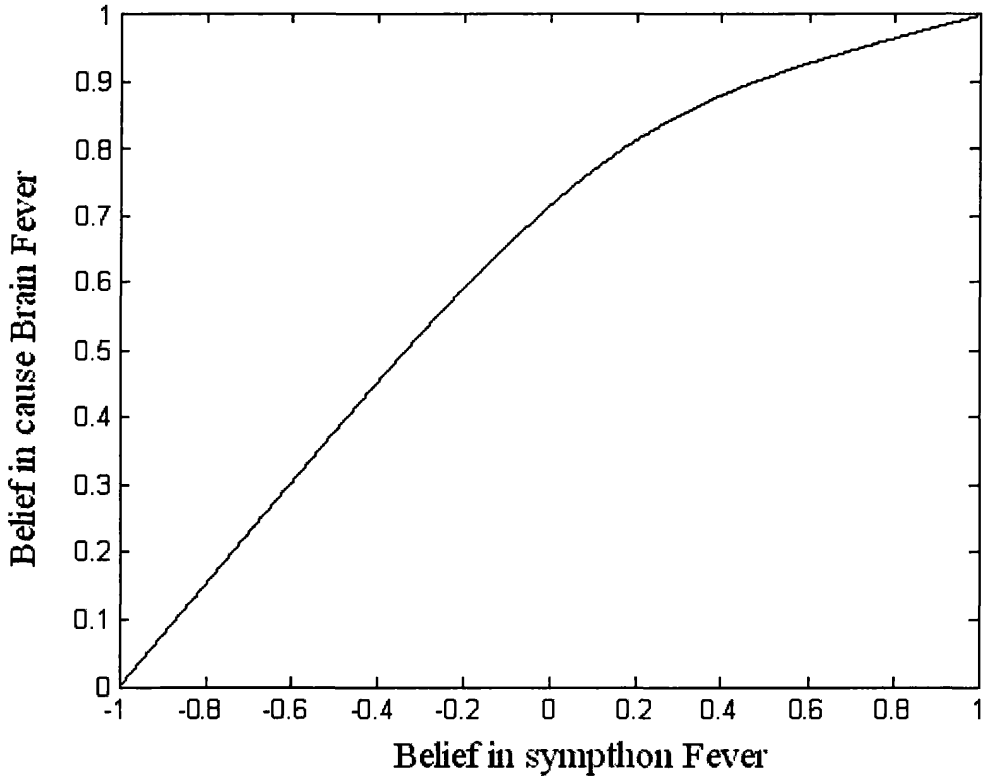


Figure 5.4: Belief that 'Brain Fever' is a cause for a symptom 'Fever'.

Figure 5.4 demonstrates a quadratic variation of cause and effect. It shows that the belief that 'Brain Fever (Encephalitis)' is a cause for symptom 'Fever' is at its minimum when the belief value representing the strength of the effect or symptom 'Fever' is at its minimum. The belief value on the occurrence of the corresponding cause starts to increase as the strength of the effect starts to increase. It is noticed that

when the strength of the affect is around half of its maximum value, the rate of increase in the belief value slows down and reaches its maximum value when the strength of the effect also reaches its maximum value.

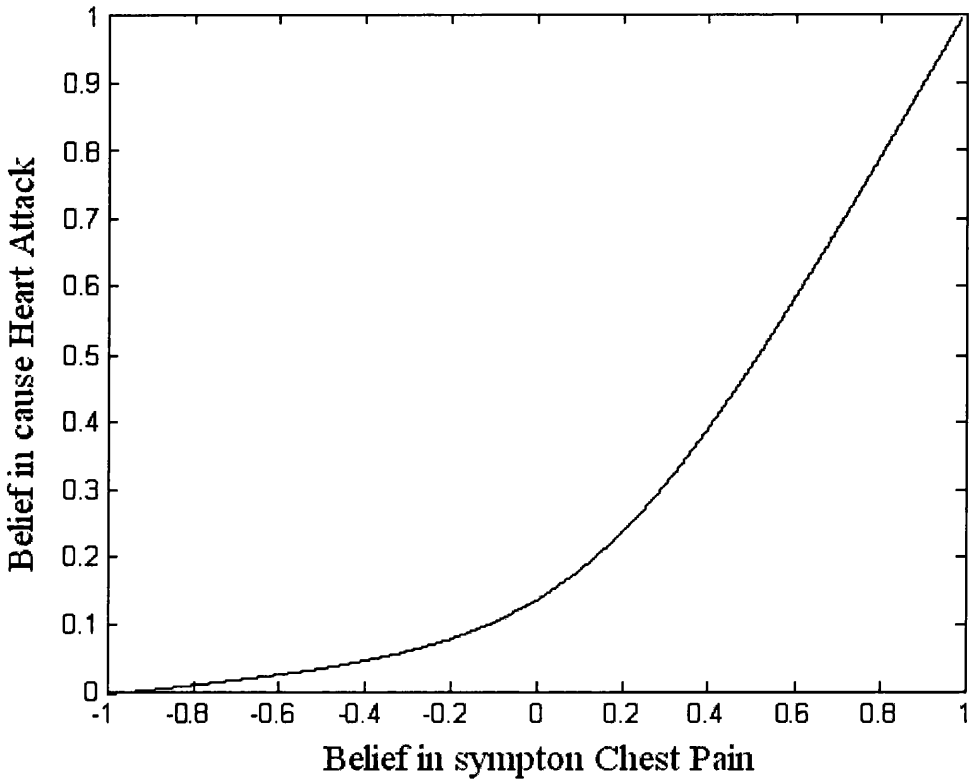


Figure 5.5: Belief that 'Heart Attack' is a cause for a symptom 'Chest Pain'.

Figure 5.5 above also shows a quadratic variation of cause and effect, in which, when the belief value representing the strength of the effect or symptom 'Chest Pain' is at its minimum, then the belief that 'Heart Attack' is a cause for a symptom 'Chest Pain' is also at its minimum. The belief value in the occurrence of the corresponding cause starts to increase slowly as the strength of the effect starts to increase. The belief value in the occurrence of the cause suddenly increases and reaches its maximum value as the strength of the effect increases to about half of its maximum value.

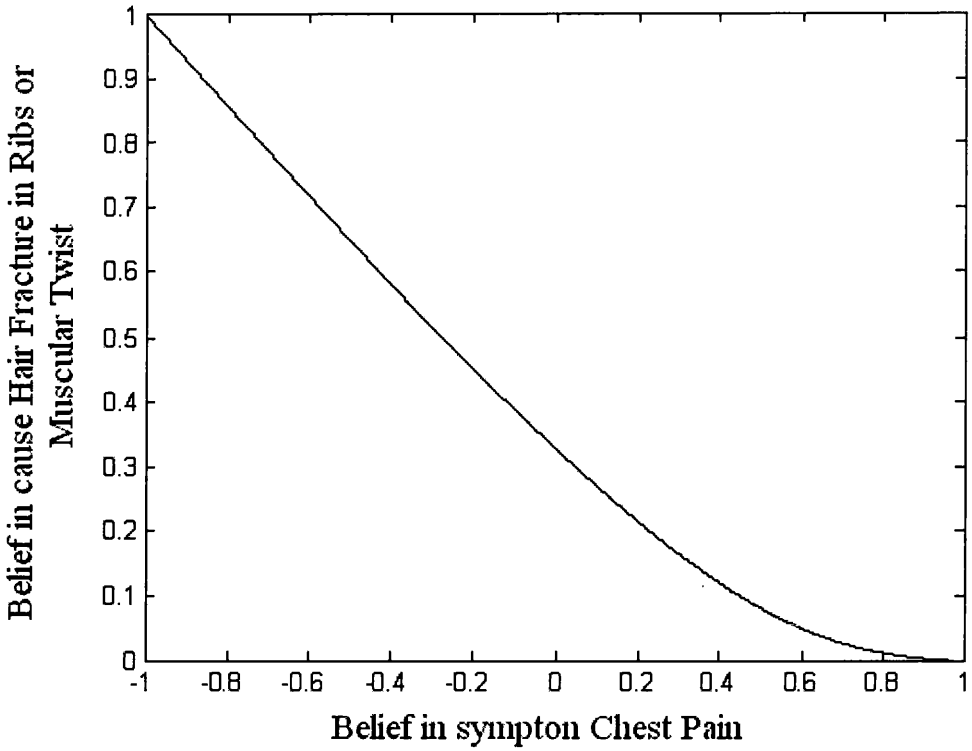


Figure 5.6: Belief that 'Hair Fracture in Ribs or Muscular Twist' is a cause for a symptom 'Chest Pain'.

In Figure 5.6, a quadratic variation of cause and effect shows that the belief that 'Hair Fracture in Ribs or Muscular Twist' is a cause for a symptom 'Chest Pain' is at its maximum when the belief value representing the strength of the effect or symptom 'Chest Pain' is at its minimum. Its clearly shows that there is a quick reduction in the belief value in the occurrence of the cause as the strength of the effect starts to increase. The belief value decreases slowly until its reaches minimum value as the strength of the effect increases to about half of its maximum value.

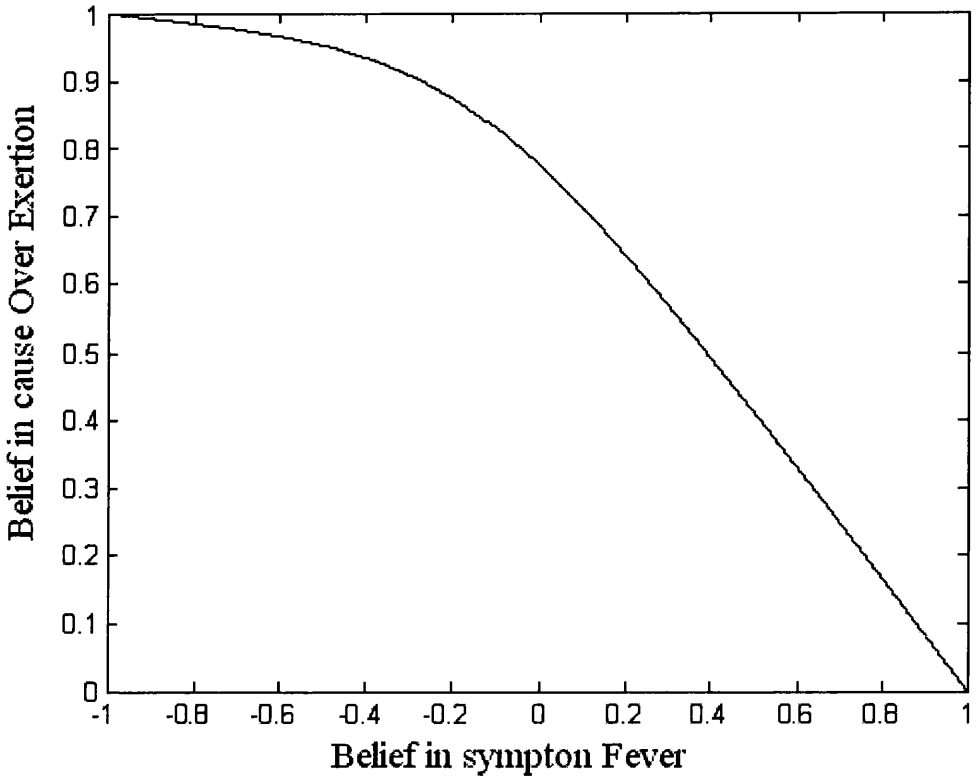


Figure 5.7: Belief that 'Over Exertion' is a cause for a symptom 'Fever'.

Figure 5.7 demonstrates that for a quadratic belief variation in a cause and effect relationship, when the belief value representing the strength of the effect or symptom 'Fever' is at its minimum, then the belief that 'Over-Exertion' is a cause for a symptom 'Fever' is at its maximum. The belief value in the occurrence of the cause slowly starts to decrease as the strength of the effect increases. The belief value starts to decrease quickly when the strength of the effect reaches about half of its maximum and then reaches its minimum when the strength of the effect is at its maximum.

If more than one effect is associated with a cause, the cause shown in Figures 5.3 to 5.7 would generate a multidimensional hyper surface. Following the previous works, it is assumed that these hyper surfaces have similar curvatures to their one-dimensional counterparts. And as a result, in order to construct such smooth multi-dimensional hyper-surfaces, Ransing (2002) proposed to use linear, quadratic or cubic one-dimensional Lagrange Interpolation polynomials.

The next section identifies some advantages and limitations of the current Knowledge Hyper-surface method and the remedies for the limitations are presented in the following sections.

5.3 ADVANTAGES OF THE CURRENT KNOWLEDGE HYPER-SURFACE METHOD

- The current method was capable *a priori* of storing any known information about the cause-effect relationship within the network and at the same time was able to learn from examples. For some selected datasets the proposed algorithm has shown superior extrapolation abilities as compared to the multi-layer neural network. The extrapolation ability was enhanced by the network's ability to constrain the shape of the resulting multi-dimensional hyper-surface to the known variation in the belief values in causes and effects.
- The dependence of the secondary weight values on the primary weight values had reduced the number of unknowns to an acceptable number.

In the following section some limitations of the Knowledge Hyper-surface method are identified and then an improvement has been proposed to overcome these limitations.

5.4 LIMITATIONS OF THE CURRENT KNOWLEDGE HYPER-SURFACE METHOD

Despite the superior extrapolation abilities of the current knowledge Hyper-surface method, two major limitations have been identified.

- Use of higher ordered polynomials can lead to the 'over-fitting' effect as observed in other interpolation techniques including neural networks.
- An exponential rise in the belief value (as shown in Figure 5.9) cannot be modelled by lower-ordered polynomials such as quadratic and cubic Lagrange interpolation polynomials.

To demonstrate the over-fitting effect, the following dataset is created by choosing a few data points and then a maximum of twenty percent noise with normal distribution with mean zero and unit standard deviation value is added randomly. The variations are plotted using linear-, quadratic- and quartic-shape functions to observe the performance of the current method as shown in Figure 5.8.

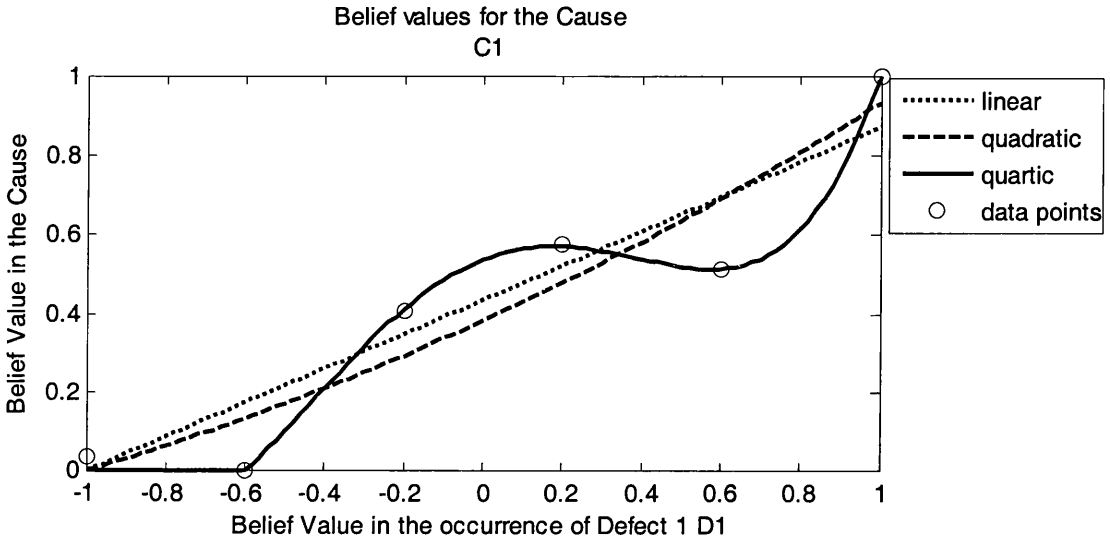


Figure 5.8: Data points plotted with linear-, quadratic- and quartic-shape functions to demonstrate the over-fitting effect caused by the current Knowledge Hyper-surface method.

Figure 5.8 clearly shows that the use of quartic-shape functions in the current Knowledge Hyper-surface method had fitted all the data points perfectly as compared to the others, but the resulting shape of the decision hyper-surface is unrealistic and is a clear case of 'over-fitting' to the data points.

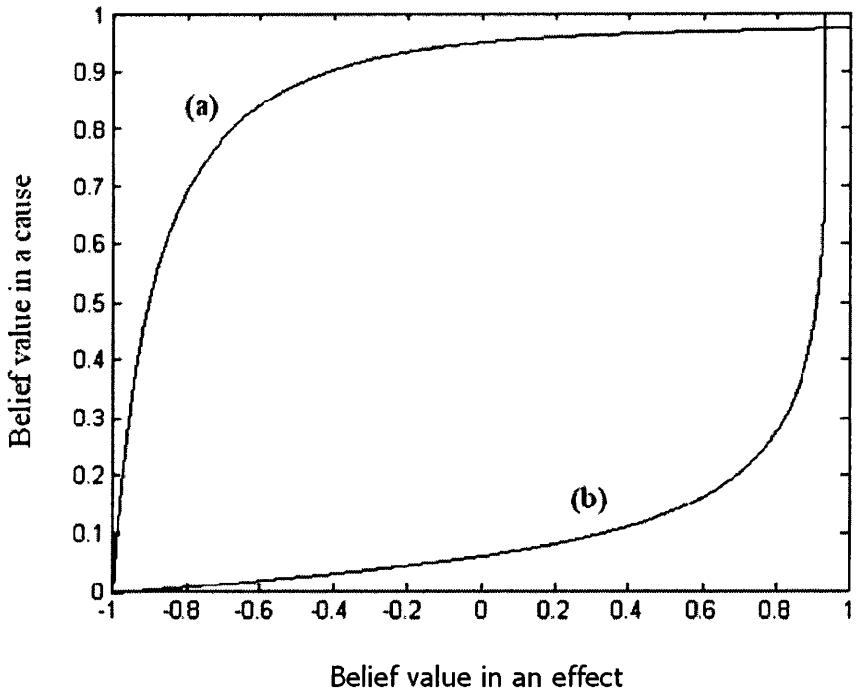


Figure 5.9: Exponential increase in the belief value of a cause.

The performance of the current Knowledge Hyper-surface method is assessed on data points generated from curves (a) and (b) in Figure 5.9, and is shown in Figures 5.10 and 5.11, respectively. It is clear that lower-ordered polynomials cannot model the exponential rise in belief values where in the higher-ordered polynomials tend to overfit the data points.

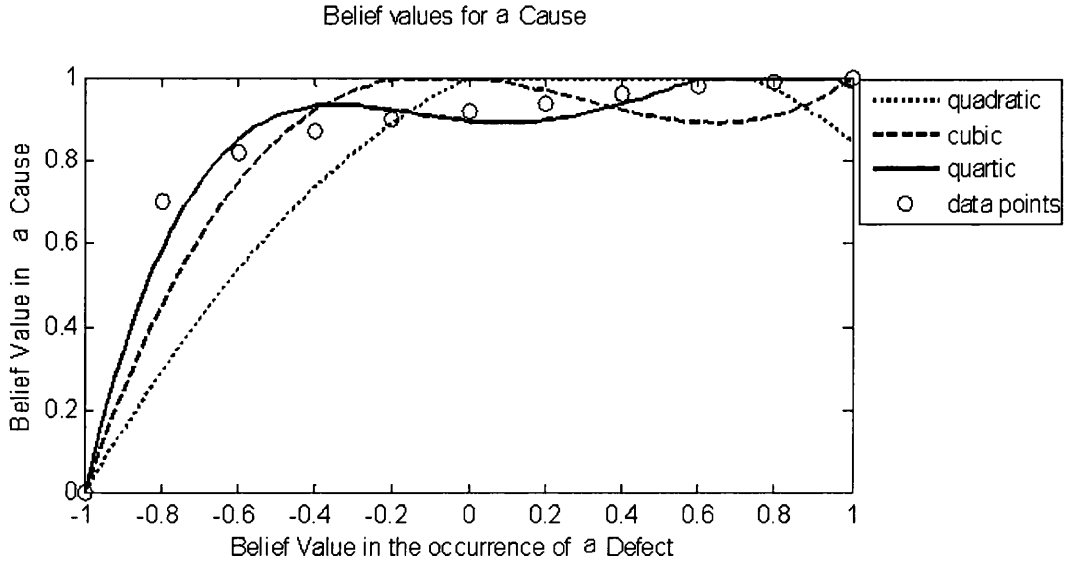


Figure 5.10: Belief value variation modelled by quadratic, cubic and quartic polynomials on a set of data points as shown.

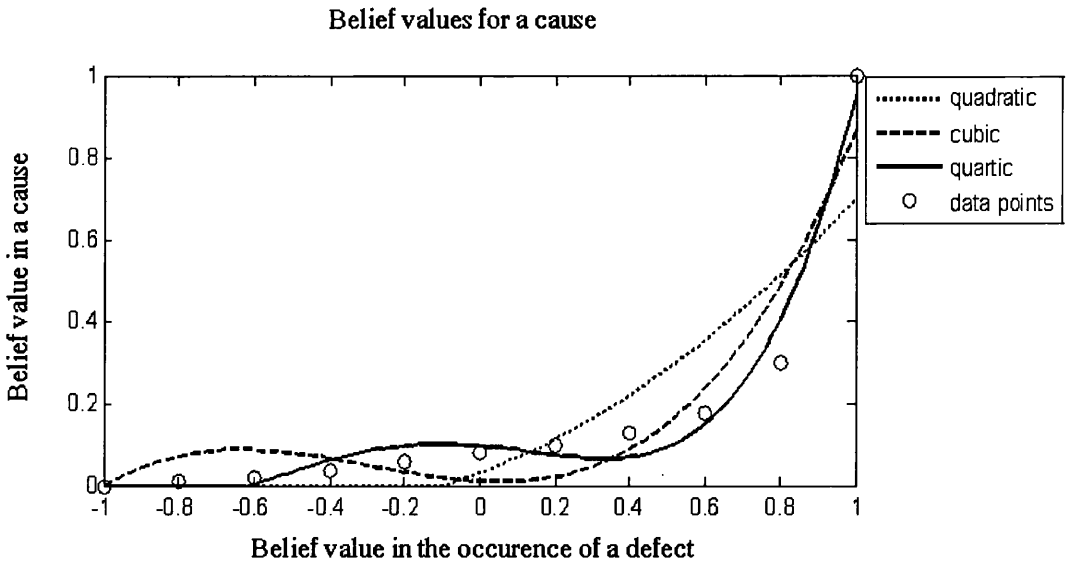


Figure 5.11: Belief value variation modelled by quadratic, cubic and quartic polynomials on a set of data points as shown.

The following section will discuss the modification that has been proposed to overcome the above-mentioned limitations within the existing technique.

5.5 ENHANCEMENT TO THE CURRENT KNOWLEDGE HYPER-SURFACE METHOD

In the current Knowledge Hyper-surface method, the multi-dimensional hyper surface is constructed from one-dimensional belief curves. Once the shape of each one-dimensional curve is determined, the shape of the hyper surface gets automatically determined. Hence the challenge for the proposed enhancement is to be able to model the exponential rise by higher-ordered polynomials without introducing the side-effects of over fitting the possible noise in data points.

This is achieved by a two-stage optimisation process. As can be seen in Figure 5.12, first the belief values at the end of points and the midpoint are determined using a quadratic Lagrange interpolation polynomial and employing the current Knowledge

Hyper-surface method. This method determines the primary weight values at the end- and mid-reference points. The exponential rise is either in the first half of the belief curve or in the second half.

This effect is modelled by introducing a further reference point between the end- and mid-reference point. The primary weights determined previously at end- and mid-reference points are kept constant and optimal values for the two new reference points (x_1 and x_2) are determined by a second-stage optimisation process using the current knowledge hyper method using fourth-ordered (quartic) Lagrange interpolation polynomials.

The primary weight values at the two new reference values are constrained such as that they lie between the corresponding primary-weight values at the neighbouring end- and mid-reference points as shown in Figure 5.12.

In the proposed new method, midpoints are constructed between each primary weight along each dimension such that:

1. 0 (i.e. origin at point 1) <= x_1 <= primary-weight at point 2, and
2. Primary-weight at point 2 <= x_2 <= primary-weight at point 3.

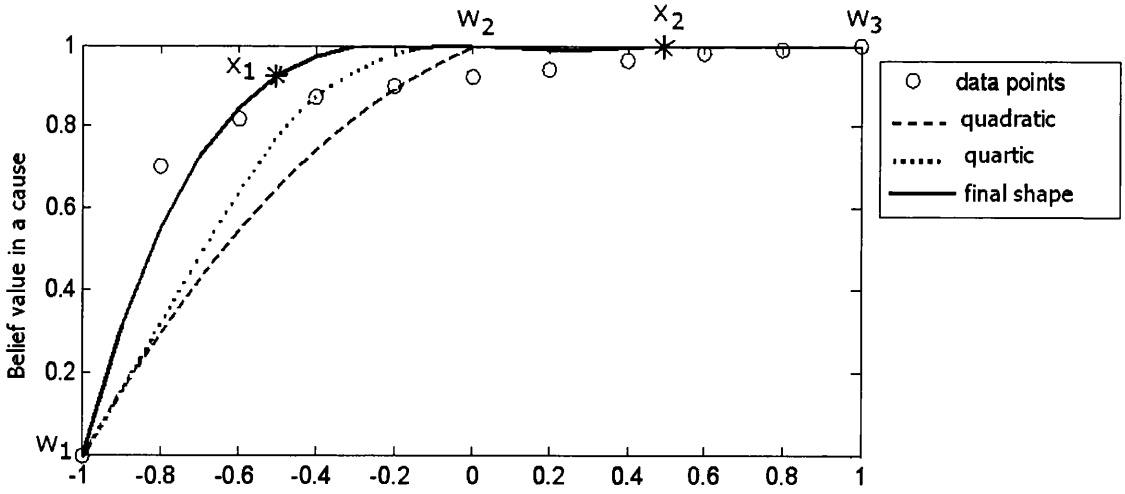


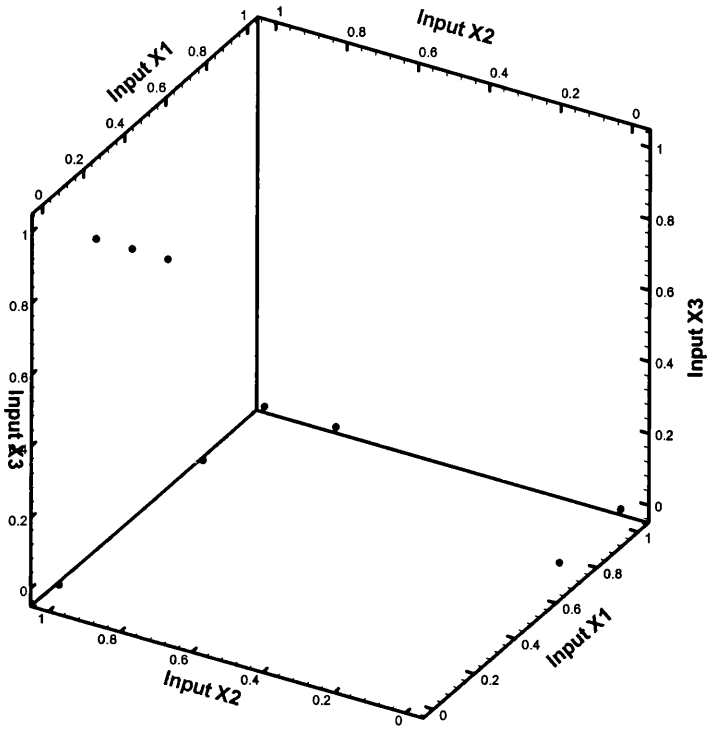
Figure 5.12: Shape of the resulting curve after a two-stage optimisation process. It ensures that x_1 is between w_1 and w_2 and x_2 between w_2 and w_3.

5.6 THE PERFORMANCE COMPARISON OF THE PROPOSED METHOD WITH THE CURRENT METHOD AND NEURAL NETWORK ON A REAL DATASET

The abilities of the proposed method to capture the exponential change in the belief variation of the cause when the belief in the effect is at its minimum is compared with the outputs from both the current Knowledge Hyper-surface method and a multi-layer neural network on a real dataset. This dataset was also used by Ransing (2002). The data was collected from 'Kaye Preistigne'– a pressure die-casting foundry. A total of fourteen defects were identified and associated with forty-three process, material or design parameters. The data was collected for similar components over a period of one year. A total of sixty representative examples were finalised. For this case study as shown in Table 5.4, sixteen process parameters, three defects and eleven examples were chosen. The same information was also used by Ransing (2002).

A belief value in the occurrence of defects was calculated as corresponding to the belief values representing the occurrence and non-occurrence of associated process, design and material parameters as given by the experts in the foundry. Three defects known as 'Porosity', 'Mismakes' and 'Dimensional' are identified and all defects chosen are represented as defects A, B and C, as shown in Table 5.4.

For the purpose of comparison, the graphical variation of belief surfaces learnt by neural network, the current method and the proposed method are shown only on two defects which are 'Porosity' and 'Mismakes'. Sixteen associated process, material and design parameters were identified to create a neural network with two input nodes corresponding to defects 'A' and 'B', and sixteen output nodes corresponding to the sixteen process, material and design parameters. The belief values which were used in a training dataset are shown in Table 5.4.



X1: Belief value in the occurrence of defect 'Porosity'.

X2: Belief value in the occurrence of defect 'Mismakes'.

X3: Belief value in the occurrence of defect 'Dimensional'.

Figure 5.13: Data points used in the training dataset as tabulated in Table 5.4.

Defects:	Defect A	Defect B	Defect C
Strength of Defects:			
	1	0	0
	1	0	0
	1	1	0
	0.7	1	0
	1	0.8	0
	0.7	1	0
	0	1	0
	0	1	0
	0	1	0
	0	0.8	1
	0	0.9	1

Output node Numbers:	1	2	3	4	5	6	7	8
Target Output Values:								
	0.8	0	0	0	1	1	1	1
	0.8	0	0	0	1	1	1	1
	0.8	0.7	0	0	1	1	1	1
	0.8	0.7	0	0	1	1	1	1
	0.8	0.7	0	0	1	1	1	1
	0.8	0.7	0	0	1	1	1	1
	0	0	0	0	0	0	0	0.8
	0	0	0	0	0	0	0	0.8
	0	0	0	0	0	0	0	0.9
	0	0.7	1	0.7	0	0	0	0.7
	0	0.8	1	0.7	0	0	0	0.8

Output Node Numbers:	9	10	11	12	13	14	15	16
Target Output Values:								
	1	0.9	1	0.8	0	0	0	0.9
	1	0.9	1	0.8	0	0	0	0.9
	1	1	1	0.8	0.8	0.7	0	0.9
	1	1	1	0.8	0.8	0.7	0	0.8
	1	1	1	0.8	0.8	0.7	0	0.9
	1	1	1	0.8	0.8	0.7	0	0.8
	0	0.9	0.7	0	0.7	0	0	0
	0	0.9	0.7	0	0.7	0	0	0
	0	0.9	0.7	0	0.8	0	0	0
	0	0.9	0	0.7	0.7	0	0	0
	0	1	0.7	0.7	0.7	0	0	0

Table 5.4: The training dataset with target output values for the input defects plotted in Figure 5.13.

The proposed conjugate gradient neural-network method (CGPR/AG) with five hidden nodes is constructed and trained on the training dataset with a learning rate equal to 0.4 and with a target error of 0.001. Since a neural network uses sigmoid activation function, the input data for the neural network was scaled between [0, 1]. A quadratic variation between input and output relationships was assumed in both the current method and the proposed method. Both networks were trained on the training dataset as shown in Figure 5.13. Codes for all methods have been written in MATLAB.

All networks achieved the target error of 0.001 and seemed to have learnt the training dataset. The speed of all networks in learning the training dataset is not the main concern in this test, as the resulting shape of the hyper surface is of importance. The belief surface has been plotted for cause 'The position of gate' (cause number 8) which influences the occurrence of 'Porosity' (defect A) and 'Mismakes' (defect B) as this data requires to model the exponential rise in the belief values variation.

The variation in the belief value in the occurrence of 'The position of gate' for defect A, i.e. 'Porosity' using the current method and the proposed method is plotted when only defect A is connected to the cause (one-dimensional case) and when both defects (i.e. defects A and B) are connected to the cause (two-dimensional case). The results are shown in Figures 5.14 and 5.15. Since the proposed method is able to model an exponential increase in belief values, it was shown to be a better fit to data points using the quadratic polynomials as compared to the current method. This is because of the introduction of midpoints which gives an additional degree of freedom to control the resulting curve. Furthermore, Figure 5.16 also demonstrates that neural networks showed a reasonable fit to these data points. However, as demonstrated by Ransing (2002) neural networks do not guarantee a better shape for hyper surfaces. Neural networks tend to interpolate better point and exhibit all the limitations as identified by Ransing (2002).

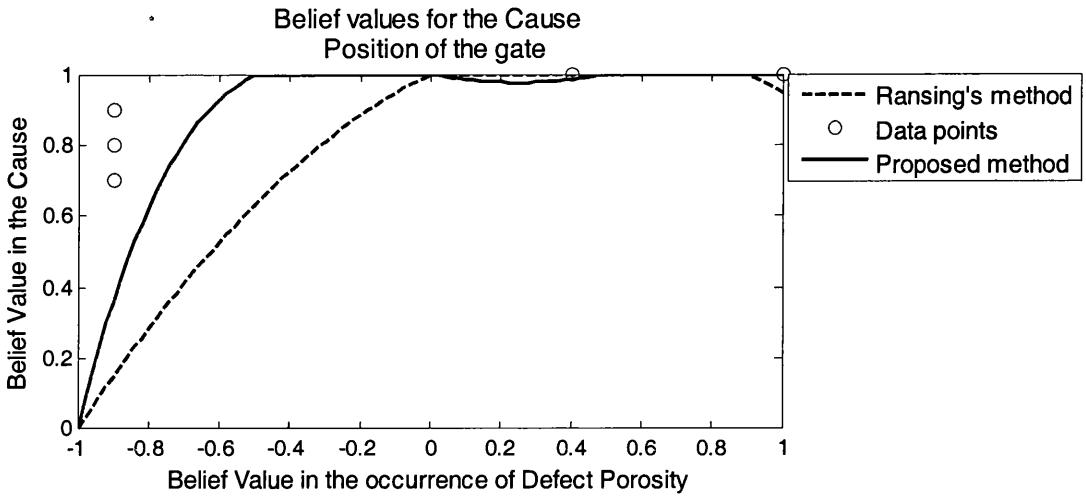


Figure 5.14: The performance of Ransing's method and the proposed method for one-dimensional belief-value variation modelled by quadratic polynomials for defect Porosity.

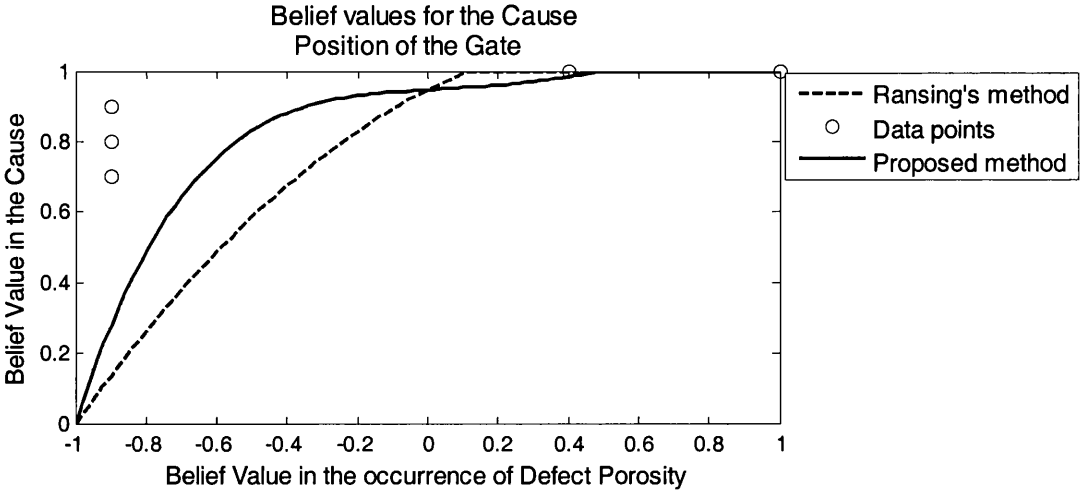


Figure 5.15: The performance of Ransing's method and the proposed method for 2D belief-value variation modelled by quadratic polynomials for defect Porosity.

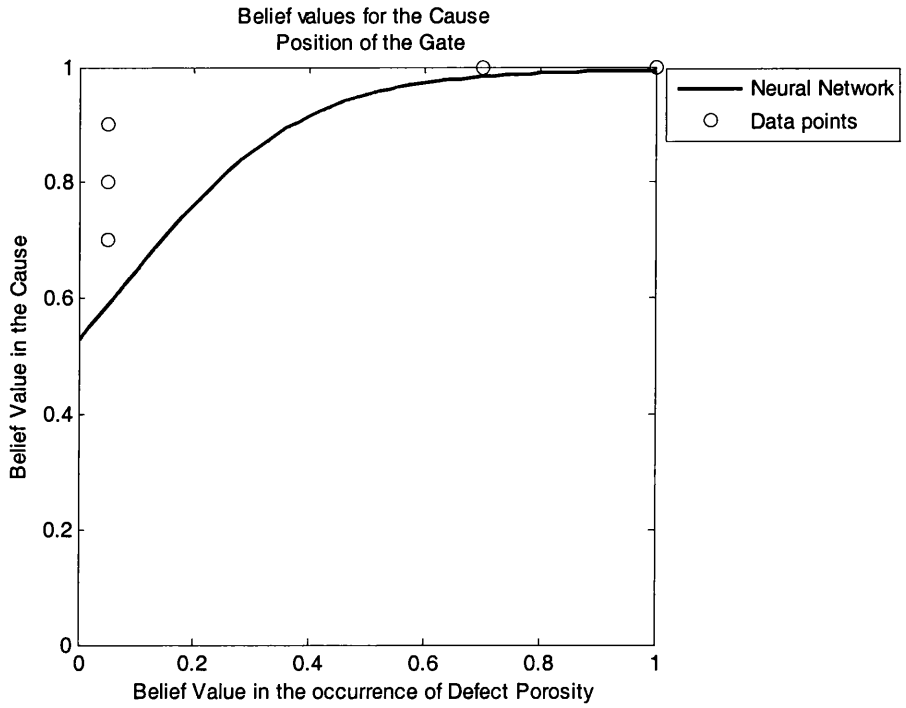


Figure 5.16: The performance of neural-network method for a 2D belief-value variation for defect Porosity.

Figures 5.17, 5.18 and 5.19 show the variation in the belief value in the occurrence of the 'The position of gate' for defect B, i.e. 'Mismakes' using the proposed method, the method proposed by Ransing (2002) and neural-network method plotted for both one-dimensional and two-dimensional cases. The results demonstrate that the proposed method has modelled the exponential rise in the data points better than both Ransing's and the neural-network methods.

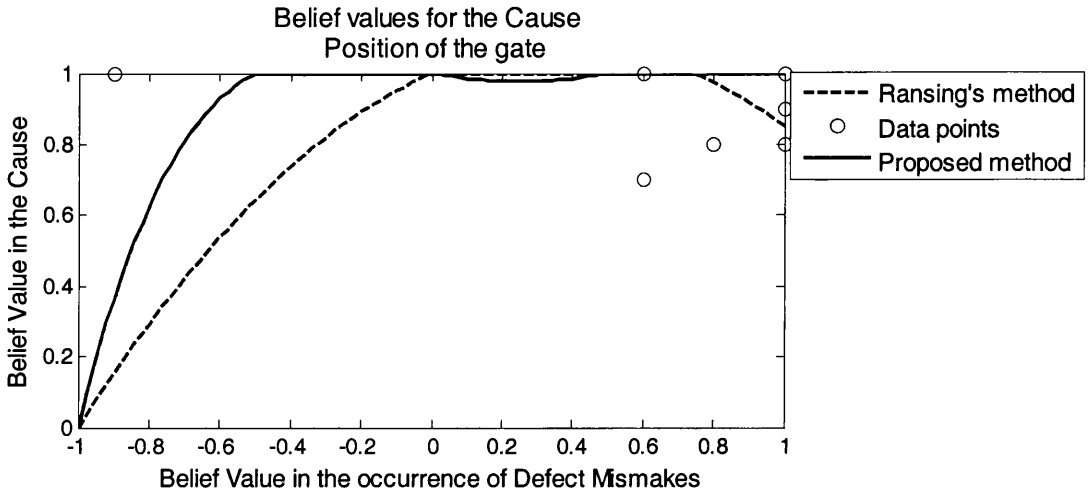


Figure 5.17: The performance of Ransing's method and the proposed method for 1D belief variation modelled by quadratic polynomials for defect Mismakes.

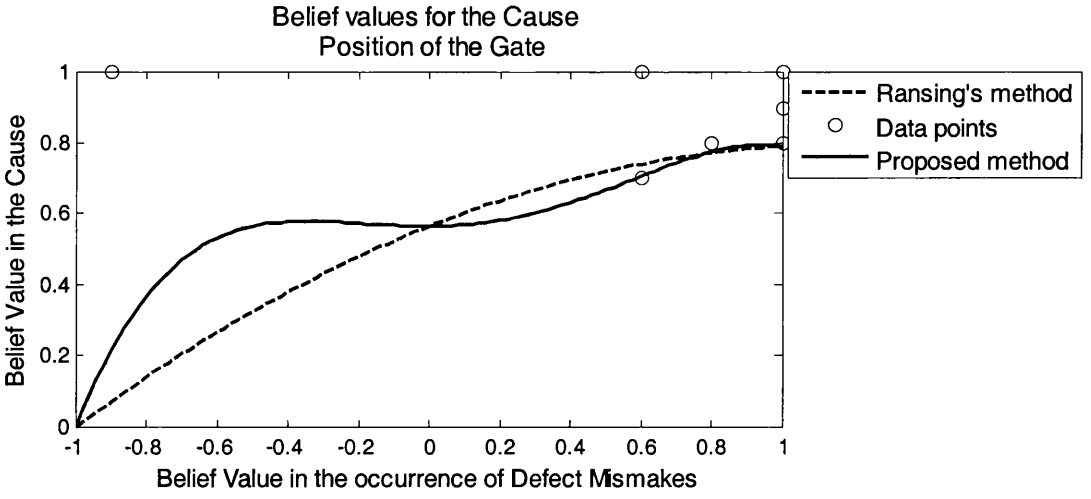


Figure 5.18: The performance of Ransing's method and the proposed method for 2D belief variation modelled by quadratic polynomials for defect Mismakes.

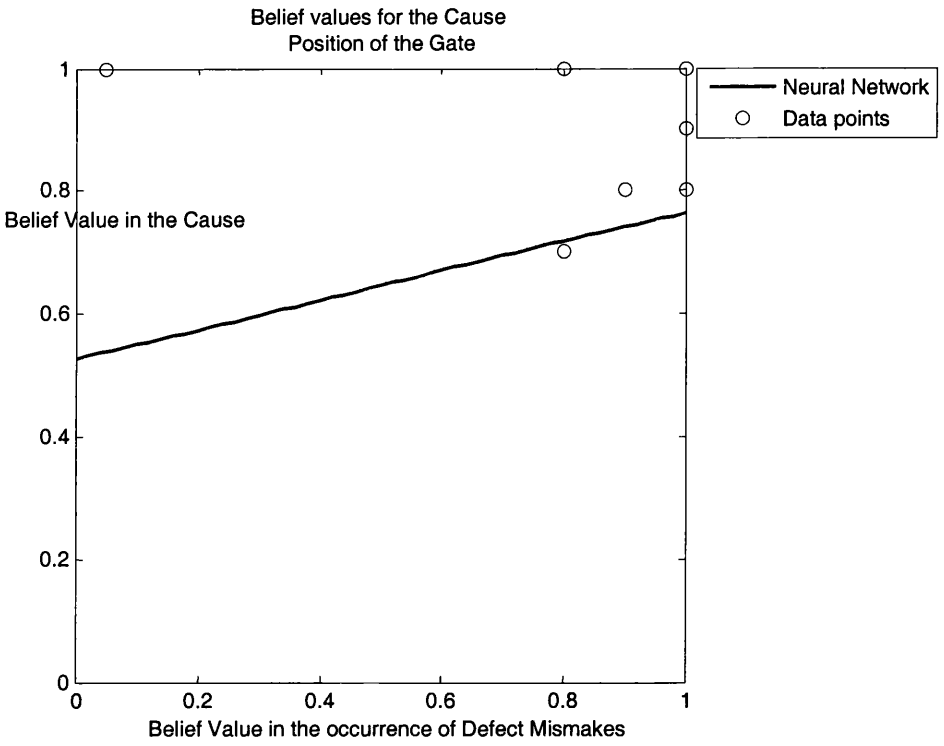


Figure 5.19: The performance of neural network method for 2D belief-value variation for defect Mismakes.

Figures 5.20, 5.21 and 5.22 show the variation in the belief values in the occurrence of 'The position of gate' for belief values for defects 'Porosity' and 'Mismakes' using the proposed method, Ransing's method and the neural-network method. It can easily be observed that the proposed method has an ability to accurately model the exponential rise in the belief values rather than the other two techniques.

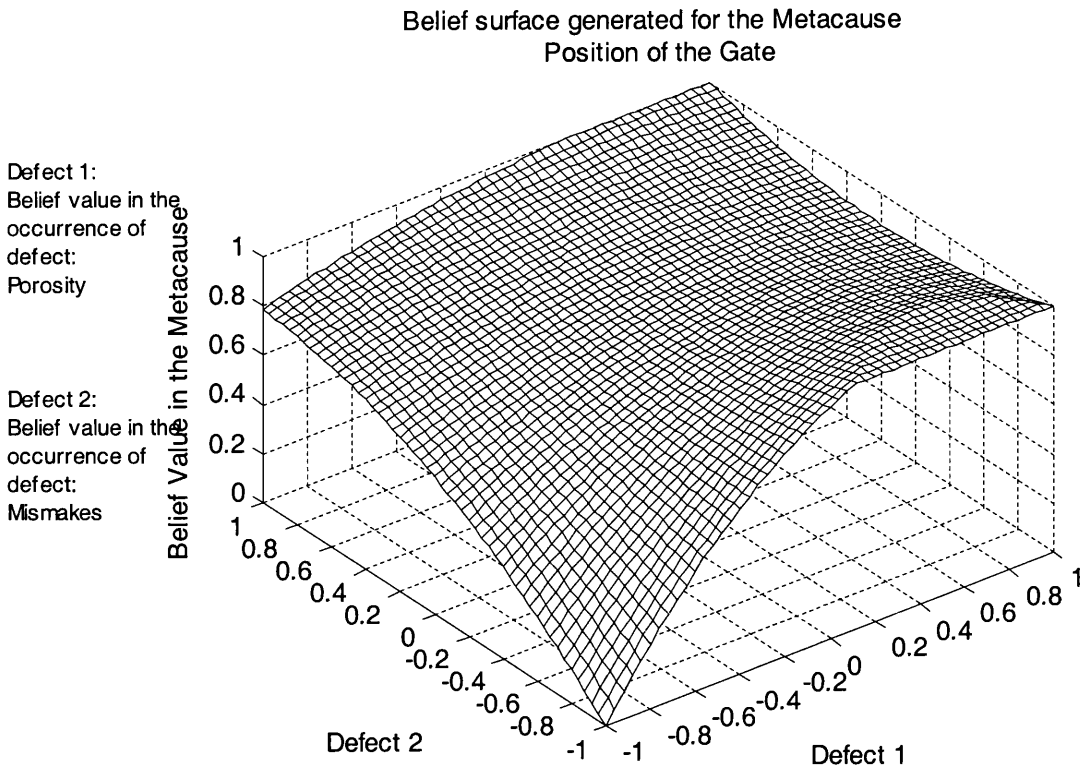


Figure 5.20: 2D quadratic output surface for defects Porosity and Mismakes generated by Ransing's method.

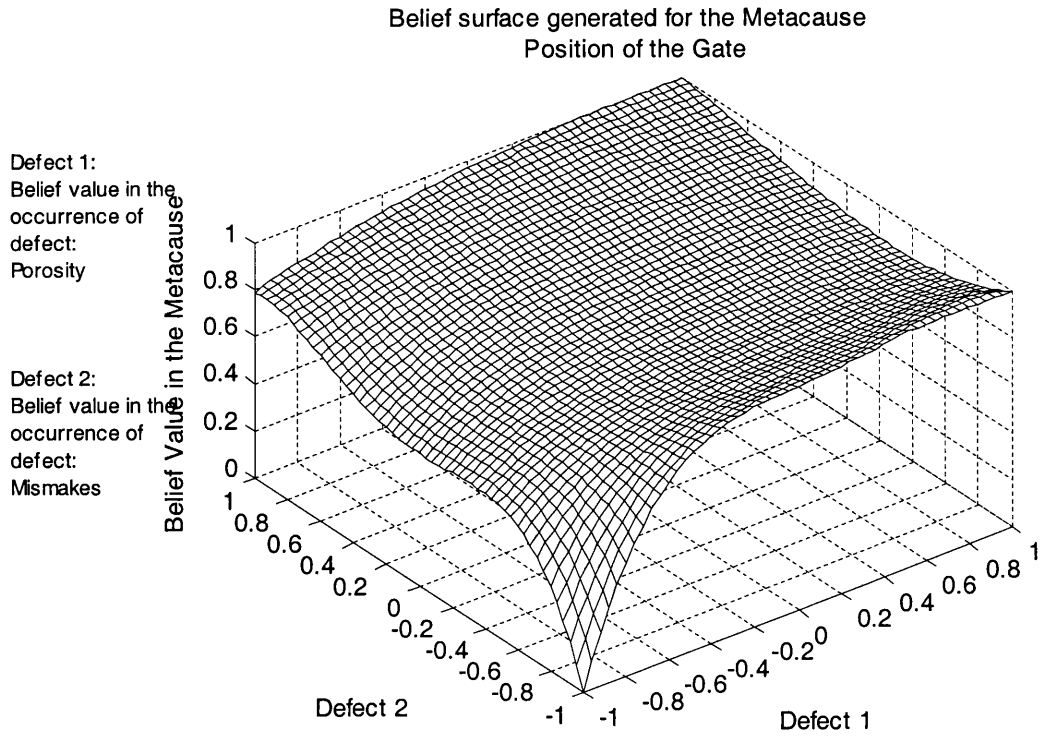


Figure 5.21: 2D quadratic output surface for defects Porosity and Mismakes generated by the proposed method.

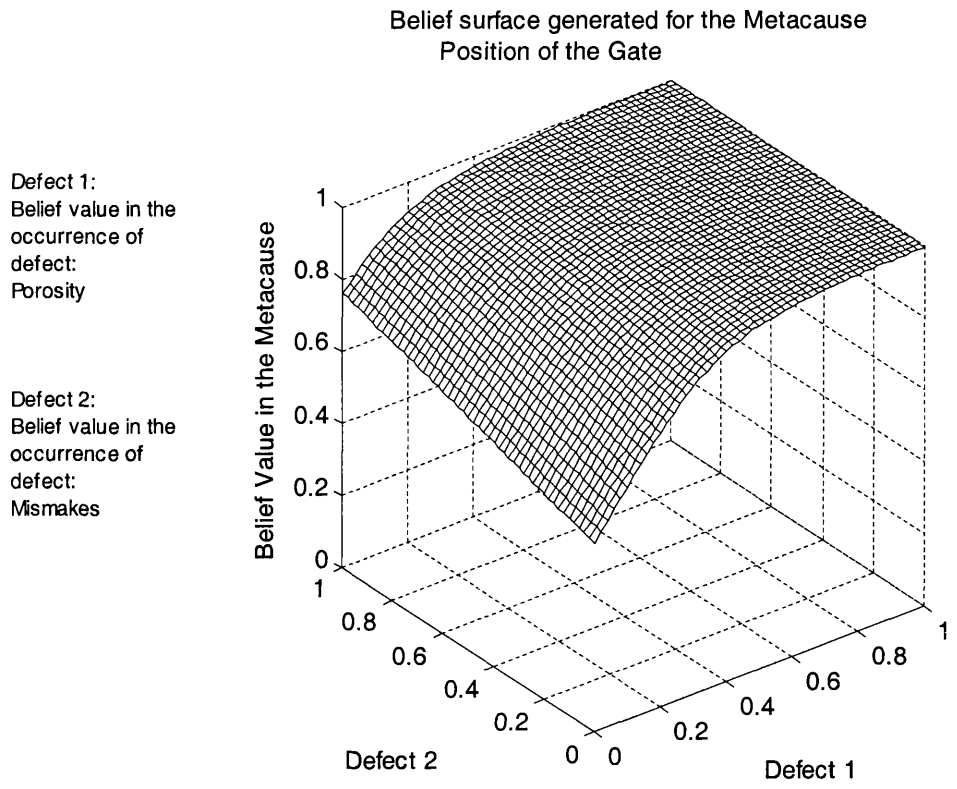


Figure 5.22: 2D output surface for defects Porosity and Mismakes generated by the neural-network method.

5.6.1 Importance of the need for accurately monitoring the exponential rise in belief values

The major objective of a robust parameter design methodology is to make the system insensitive or 'robust' to a process variation. In a robust parameter-design method, the output variation can be lowered by reducing either the sensitivities to the variation in the design factor or sensitivities to noise factors. Figure 5.23 shows how a factor setting may influence the variation of the output depending on the occurrence of the belief variation. When design factor setting one is chosen, more variation is transmitted from a small change design factor value to its output due to the exponential rise in the slope of the belief curve. This makes the corresponding output more sensitive to the variation of design factor setting one. Whereas for factor setting two even a larger change in values will not influence the output value. Design factor

setting two thus offers a robust design setting as the process is insensitive to its variation. The proposed method has an ability to accurately model the exponential rise in the data values. This has significantly improved the applicability of the Knowledge Hyper-surface method in addressing robust design problems.

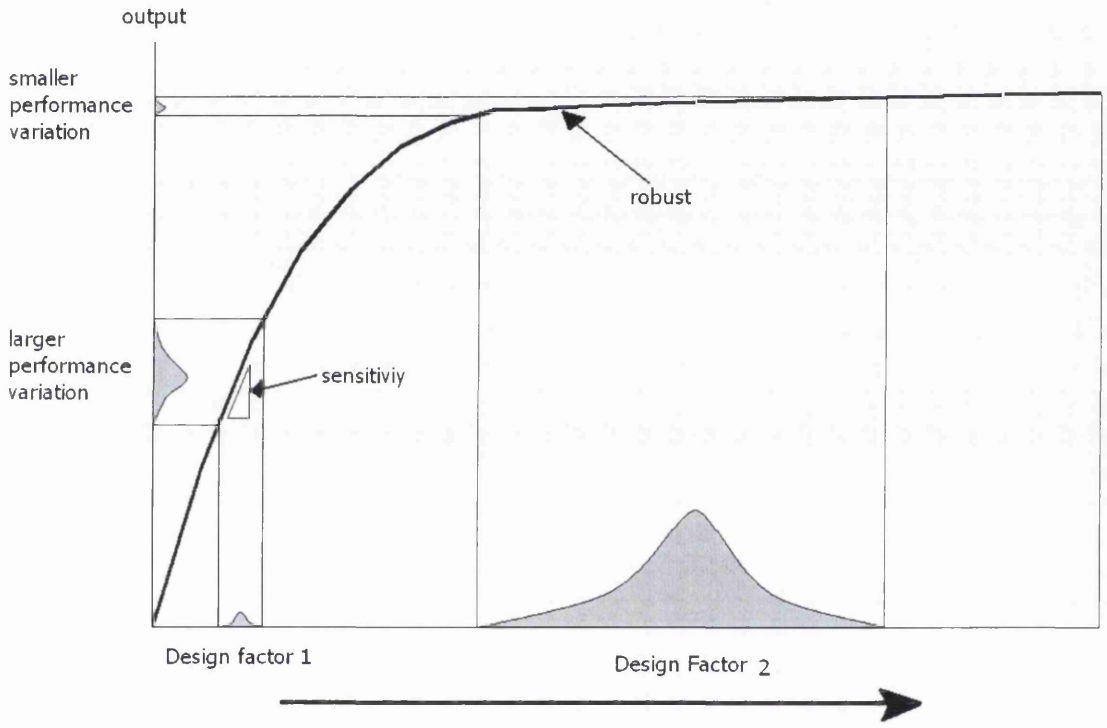


Figure 5.23: Robust design principles: assessing the sensitivity of output variation to the change in design-factor settings.

5.7 CONCLUSION

An enhancement to the current Knowledge Hyper-surface method has been proposed in this chapter. The method introduces midpoints in the existing shape-function formulation so that an exponential rise in the belief-value variation can be modelled without introducing the effects of 'overfitting'. The performance of the proposed method was compared with the method proposed by Ransing (2002) and the neural-network method on the same casting data used by Ransing. The proposed method does not have limitations of neural-network techniques as identified by Ransing (2002).

If the function $y = f(x)$ increases exponentially, a small change in the 'x' value produces a large change in results. The process parameters that show such variation need close monitoring in a manufacturing process and in robust design applications, in particular. Hence, the ability of the network to model an exponential increase in belief values is a significant step forward in the research direction.

CHAPTER 6

CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE WORK

6.1 RESEARCH CONCLUSION

The work presented in this thesis focuses on improving the computational efficiency of neural-network training algorithms and investigates the applicability of its 'learning from examples' feature in improving the performance of a current intelligent diagnostic system. The back propagation (BP) algorithm which is one of the best known and widely used learning algorithms for neural networks is reviewed in detail and the limitations of the conventional BP training algorithm is highlighted. Two major areas of improving the BP algorithm are identified in the literature as: (a) the use of heuristic-based techniques that modify network parameters such as learning rate value, momentum term, activation function, and topology optimisation; and (b) the integration of (a) with second-order optimisation techniques for minimising the error. The ability of the current method proposed by Ransing (2002) in modelling the exponential increase/decrease of belief values in cause and effect relationships also has been discussed in detail and remedies have been recommended. The aspects researched and refined were:

- Investigate further on improvements to the BP algorithm proposed by early researchers, particularly the work presented by Ransing (2002), on using the adaptive-gain variation in improving the training efficiency.
- Discover the misunderstanding of previous researchers in their claims that the training efficiency of the gradient-descent formulation was improved because the gain variation influenced the learning rate.
- Propose a novel approach that improves the search direction and hence improves the training efficiency of BP neural-network algorithms.

- Implement the novel approach into other well-known optimisation methods with an objective of improving the computational efficiency of neural-networks training process.
- Revisit the current Knowledge Hyper-surface method proposed by Ransing (2002) and identify some limitations posed by the existing version in modelling the exponential increase/decrease in belief values in cause and effect relationships.
- Propose a strategy that is computationally efficient and able to model the exponential increase/decrease in belief values in cause and effects relationships without introducing the side-effects of 'over-fitting'.

The following conclusions may be drawn from the work presented in this thesis:

1) This study suggests that adaptive-gain variation as introduced by previous researchers including Ransing (2002) has a significant effect in improving the search direction not the learning rate. A novel method to improve the training efficiency of BP algorithms with respect to adaptive-gain variation of activation function has been successfully developed. The proposed method not only coupled the gain update expressions for output, as well as the hidden nodes as derived by Ransing (2002), but also coupled with the adaptive-learning rate. Furthermore, the generic nature of the proposed method has been demonstrated by successfully implementing its formulation into other well-known optimisation methods to yield significant improvements in the computational speed. To the best of the author's knowledge, this was the first instance when the adaptive gain of activation function has been implemented with almost all commonly used gradient-based optimisation algorithms. The theoretical formulation of the proposed method has been expressed in terms of three major optimisation methods: gradient-descent method; conjugate-gradient method; and Quasi-Newton method.

2) For implementation into a computer code, MATLAB programming language was chosen as the language that combines comprehensive math and graphics functions with a powerful high-level language beyond those provided by languages such as FORTRAN and C. The correctness of implementing the code was tested on data generated using a simple sine curve. The computed numerical results were

compared with the previous version introduced by Ransing (2002). The comparison showed significant improvements using the proposed method.

3) The efficiency of the proposed method (with respect to computer run time and generalisation accuracy) implemented with other optimisation methods was investigated by using benchmark problems. The benchmark problems used to verify the proposed algorithm are taken from the open literature (Lutz Prechelt, 1994). The computed numerical results as well as graphical results of the proposed method were compared with other standard algorithms as well as Ransing's method in terms of training efficiency. The results clearly showed that the proposed method substantially improved the computational efficiency of the training process. In addition, the proposed algorithm is generic, robust and easy to implement into all commonly used gradient-based optimisation methods.

4) This study also explored limitations of the existing Knowledge Hyper-surface method proposed by Ransing (2002) in learning cause and effect relationships. A new approach to enhance the performance of the current Knowledge Hyper-surface method has successfully been proposed. The theoretical formulation of the approach has been expressed by constructing midpoints between each primary weight along each dimension by using a quadratic Lagrange interpolation polynomial. The new secondary-weight values, generated due to addition of midpoints, were also represented as a linear combination of the corresponding primary/axial weight values. An algorithm to constrain the shape of the surface in two-dimensional and multi-dimensional cases has been successfully developed in order to produce more realistic and acceptable results as compared to the previous version.

5) The ability of the proposed approach to model the exponential increase/decrease in the belief values by using high-ordered polynomials without introducing 'over-fitting' effects was investigated. The performance of the proposed method in modelling the exponential increase/decrease in belief values was carried out on real cases taken from real casting data used by Ransing (2002). The computed graphical result of the proposed method was compared with the current Knowledge Hyper-surface and neural-network methods. As a result of this research achievement, it will now be possible to correctly predict the sensitivity of process-parameter variations with the occurrence of defects. This is an important area of research in a robust design methodology.

In general, the proposed approach has shown significant improvement on computational efficiency and at the same has provided industry with an efficient self-learning decision-making tool, which has the knowledge of current/past rejection levels within the manufacturing set up. The tool automatically learned a cause and effect relationship by using the diagnosis information provided by experts. This learning ability has the potential to help managers not only to quantify the influence of causes on defects for existing products but also to be very computationally efficient for use in manufacturing new quality products.

6.2 PROPOSAL FOR FUTURE WORK

The method developed to improve the current Knowledge Hyper-surface method in providing industry with an efficient self-learning decision-making tool has been shown to be capable of giving good results in all case studies in Chapter Five. In addition, a novel method was developed to improve the computational efficiency of neural-network algorithms as described in Chapters Three and Four. The recommendations regarding the further development are given below:

(1) The proposed method significantly improved the computer training efficiency as demonstrated in Chapters Three and Four, the improvement is actually the result of automatically varying gain parameters and learning rates during training. It was also noticed that the success of neural-network models largely depended on their architecture, which is usually determined by a trial and error process (E. Cantu-Paz, 2003). It would be useful to develop an automatic and effective neural-networks training model, as shown in Figure 6.1, that combines the proposed method together with techniques such as network pruning (K. Suzuki *et al.*, 2001) which can automatically optimise their network architecture. Furthermore the implementation of this combination into all gradient-based optimisation methods can simplify network training without human intervention while at the same time further improve the computational efficiency.

(2) A good method has been successfully developed by Smart and Zhang (2004) which integrated the gradient-descent method with genetic programming (GP) in investigating for object-classification problems. The results showed that the new

method outperformed the basic GP method on all cases in both classification accuracy and training generations. However, that method was only tested on the gradient-descent method. It is also possible to explore the effect of other gradient-based methods as proposed in Chapters Three and Four with GP on even more difficult image-classification problems such as face-recognition problems and satellite image-detection problems.

(3) The enhancement proposed to the Knowledge Hyper-surface method can be extended to methods used in solving problems that are generally addressed by Taguchi's methods (S.M. Phadke, 1989). The ability of the method to model an exponential change will also help in gaining an insight into the 'tolerance design' process for various machine parameters. These are a few immediate milestones that need to be achieved in the process of realising the dream of designing an 'Intelligent and Autonomous Foundry' of the future.

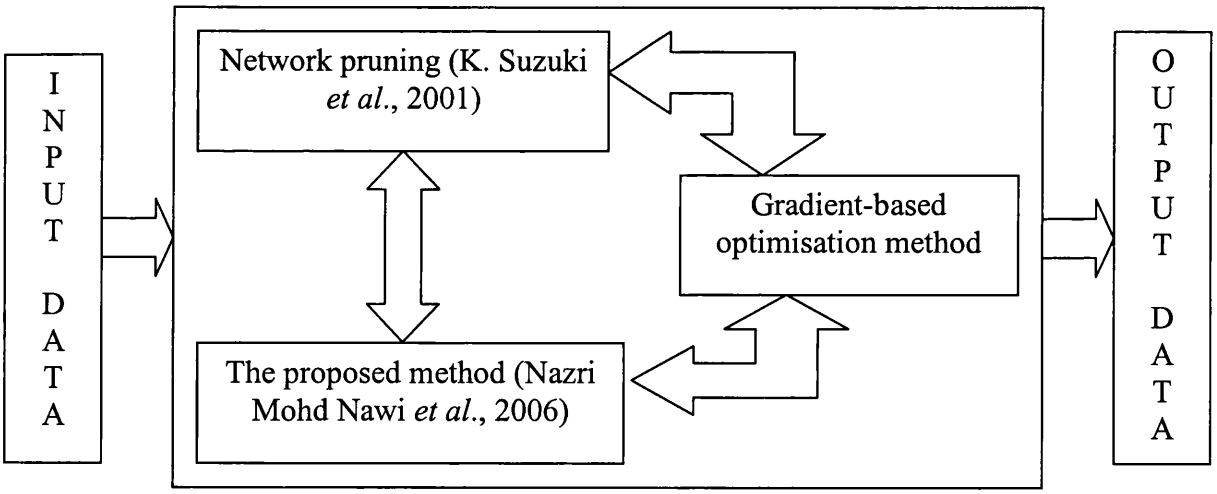


Figure 6.1: Automatic neural-networks training model.

REFERENCES

A. van Ooyen, and B. Nienhuis, 1992, Improving the convergence of the back-propagation algorithm: Neural Networks, v. 5, p. 465-471.

Adrian J. Sheperd, 1997, Second Order Methods for Neural Networks-Fast and Reliable Training Methods for Multi-layer Perceptrons, Springer, 143 p.

B. Lally, L. T. Biegler, and H. Henein, 1991, Optimisation and Continous casting. 1. Problem Formulation and Solution Strategy: Metallurgical Transactions B - Process Metallugy, v. 22, p. 641-648.

Barschdorff D., Monostori L., Wijstenkiihler G.W., Egresits Cs., and Kadar B., 1997, Approaches to coupling connectionist and expert systems in intelligent manufacturing: Computers in Industry, v. 33, p. 95-102.

Bendall A., 1988, Introduction to Taguchi methodology: Proceedings of the 1988 European Conference, p. 1-14.

Bishop C. M., 1995, Neural Networks for Pattern Recognition, Oxford University Press.

Curtis F. Gerald, and Patrick O. Wheatley, 2004, Applied Numerical Analysis. Seventh Edition, Addison-Wesley.

D. W. Marquardt, 1963, An algorithm for least-squares estimation of non-linear parameters: Journal of the Society of Industrial and Applied Mathematics, v. 11, p. 431-441.

D.E. Rumelhart, G.E. Hinton, and R.J. Williams, 1986, Learning internal representations by error propagation: in D.E. Rumelhart and J.L. McClelland (eds), Parallel Distributed Processing, v. 1, p. 318-362.

Danilo P. Mandic, J. A. C., 1999, Relating the Slope of the Activation Function and the Learning Rate within a Recurrent Neural Network: Neural Computation, v. 11, p. 1069-1077.

Dave Watkins, 1997, Clementine's Neural Networks Technical Overview: Technical Report.

David L. Rodriguez, 2003, Response Surface based Optimization with a Cartesian CFD method: American Institute of Aeronautics and Astronautics, p. 1-9.

E. Cantu-Paz, 2003, Pruning Neural Networks with Distribution Estimation Algorithms: Genetic and Evolutionary Computation Conference on The advanced information system and technology, p. 1-13.

E. M. L. Beale., 1972, A derivation of conjugate gradients.: In F. A. Lootsma, editor, Numerical methods for nonlinear optimization, p. 39-43.

E.K. Blum, 1989, Approximation of Boolean functions by sigmoidal networks: Part I: XOR and other two-variable functions: Neural Computation, v. 1, p. 532-540.

Eom K., and Jung K., 2003, Performance Improvement of Back propagation algorithm by automatic activation function gain tuning using fuzzy logic: Neurocomputing, v. 50, p. 439-460.

Erik Hjelmas, and P.W. Munro, 1999, A comment on parity problem: Technical Report, p. 1-7.

F. Fnaiech, D. Bastard, V. Buzenac, R. Settineri, and M. Najim, 1994, A Fast Kalman Filter based new algorithm for training feedforward neural networks: Proceedings of EUSPCO (European signal processing conference).

Fahlman S. E., 1988, Faster Learning Variations on Back-propagation: An Empirical Study: Proceedings of the 1988 Connectionist Models Summer School, p. 38-51.

Fisher R.A., 1936, The use of multiple measurements in taxonomic problems: Annals of Eugenics, v. 7, p. 179 -188.

Fletcher R., and Reeves R. M., 1964, Function minimization by conjugate gradients: Comput. J., v. 7, p. 149-160.

Funahashi K., 1989, On the Approximate Realization of Continuous Mappings by Neural Networks: Neural Networks, v. 2, p. 183-192.

G. P. Syrcos, 2003, Die casting process optimization using Taguchi methods: Journal of Materials Processing Technology, v. 135, p. 68-74.

Groot C. de, and Würtz D., 1994, Plain Backpropagation and Advanced Optimization Algorithms: A Comparative Study: Neurocomputing, NEUCOM 291, v. 6, p. 153-161.

Hahn-Ming Lee, Tzong-Ching Huang, and Chih-Ming Chen, 1999, Learning Efficiency Improvement of Back Propagation Algorithm by Error Saturation Prevention Method: IJCNN '99, v. 3, p. 1737-1742.

Hanselman, Duane & Littlefield, and Bruce, 1997, The student edition of MATLAB: version 5, user's guide.: New Jersey: Prentice-Hall, Inc., Prentice-Hall, Inc.

Hush D. R., Horne B., and Salas J. M., 1992, Error surfaces for multilayer Perceptrons.: IEEE Transactions on Systems, Man, and Cybernetics, v. 22, p. 1152-1161.

Hush D. R., and Salas J. M., 1988, Improving the learning rate of backpropagation with the gradient reuse algorithm.: In Proceedings of the IEEE Conference on Neural Networks, v. 1, p. 441-447.

J. E. Dennis, and R. B. Schnabel, 1983, NumericalMethods for Unconstrained Optimization and Nonlinear Equations: Englewood Cliffs, NJ, Prentice-Hall.

J. Grum, and J.M. Slabe, 2004, The use of factorial design and response surface methodology for fast determination of optimal heat treatment conditions of different Ni-Co-Mo surfaced layers: Journal of Materials Processing Technology, v. 155-156, p. 2026-2032.

Jacobs R. A., 1988, Increased Rates of Convergence Through Leaming Rate Adaptaion: Neural Networks, v. 1, p. 561-573.

John K. Kruschke, and Javier R. Movellan, 1991, Benefits of Gain: Speeded Learning and Minimal Hidden Layers in Back-Propagation Networks: IEEE Transaction on System, Man, and Cybernetics, v. 21, p. 273-280.

K. Levenberg, 1944, A method for the solution of certain non-linear problems in least squares.: Quaterly Journal of Applied Mathematics, v. 2, p. 164-168.

Kackar R.N., 1985, Off-line quality control, parameter design and the Taguchi method: Journal of Quality Technology, v. 17, p. 176-88.

Kamat Y. V., and Rao M. V., 1994, A Taguchi optimization of the manufacturing process for die cast components: Proc. 6th AIMTDR Conference, p. 174-179.

Kolen J. F., and Pollack J. B., 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, v. 3, p. 860-867.

Krzyzak A., Dai W., and Suen C. Y., 1990, Classification of large set of handwritten characters using modified back propagation model.: Proceedings of the International Joint Conference on Neural Networks, v. 3, p. 225-232.

Lee B. W., and Sheu B. J., 1993, Paralleled hardware annealing for optimal solutions on electronic neural networks: IEEE Transactions on Neural Networks, v. 4, p. 588-599.

Lee Y., Oh S.-H., and Kim M. W., 1993, An analysis of premature saturation in backpropagation learning.: Neural Networks, v. 6, p. 719-728.

Lin H., Yih Y., and Salvendy, 1995, Neural Network based fault diagnosis of hydraulic forging presses in China: International Journal of Production Research, v. 33, p. 1939-1951.

Lippman R. P., 1987, An Introduction to Computing with Neural Nets: IEEE ASSP magazine.

Lutz Prechelt, 1994, PROBEN1 - a set of neural network benchmark problems and benchmarking rules. (WEB:ftp://ftp.ira.uka.de/pub/neuron/proben1.tar.gz): Technical report 21/94, p. 1-4.

M. F. Moller, 1993, A scaled conjugate gradient algorithm for fast supervised learning: Neural Networks, v. 6, p. 525-533.

M. J. D. Powell, 1977, Restart procedures for the conjugate gradient method.: Mathematical Programming, v. 12, p. 241-254.

M. Perzyk, and A. Kochanski, 2003, Detection of cause of casting defetcs assisted by artificial neural netwroks: Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, v. 217, p. 1279-1284.

M.T. Hagan, and M. Menhaj, 1994, Training feedforward networks with the Marquardt algorithm: IEEE Trans. Neural Networks, v. 5, p. 187-199.

Magoulas G. D., Vrahatis M. N., and Androulakis G. S., 1996, A new method in neural network supervised training with imprecision.: In Proceedings of the IEEE 3rd International Conference on Electronics, Circuits and Systems, p. 287-290.

Magy. M. Kandil, Fayza. A. Mohamed, Fathy Saleh, and Magda Fayek, 2005, A New Approach For Optimizing Back Propagation Training With Variable Gain Using PSO: GVIP 05 Conference, p. 1-7.

Mangasarian O. L., and Wolberg W. H., 1990, Cancer diagnosis via linear programming: SIAM News, v. 23, p. 1-18.

Marco Gori, and Alberto Tesi, 1992, On the problem of local minima in back-propagation: IEEE Transactions on Pattern Analysis and Machine Intelligence, v. 14, p. 76-86.

Martinez E. E., Smith A. E., and Idanda B., 1994, Reducing waste in casting with a predictive neural model: Journal of Intelligent Manufacturing, v. 5, p. 277-286.

Meghana R. Ransing, 2002, Issues in Learning Cause and Effect Relationships from Examples: With particularly emphasis on casting process, University of Wales Swansea, Swansea.

Murphy Hot, and Hiroaki Kurokawa, 1998, Characteristics of Gradient Descent Learning with Neuronal Gain Control.: IEEE, p. 74-77.

Murray Smith, 1993, Neural Networks for Statistical Modeling: New York, NY, USA, John Wiley & Sons, Inc., 235 p.

Nazri Mohd Nawi, Meghana R. Ransing, and Rajesh S. Ransing, 2006, Improving the gradient based Search Direction to Enhance training Efficiency of Back Propagation based Neural Network algorithms: Proceedings of the 26th

International Conference of Innovative Techniques and Applications of Artificial Intelligent (SGAI□06), p. 45-48.

Nguyen D., and Widrow B., 1990, Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights: Proceedings of the International Joint Conference on Neural Networks, v. 3, p. 21-26.

Parker D., 1985, Learning-Logic: Technical Report TR-47.

Phadke S. M., 1989, Quality Engineering Using Robust Design: Englewood Cliffs, N.J., Prentice Hall.

Polak E., 1971, Computational Methods in Optimization: (New York: Academic Press).

Pravin Chandra, and Yogesh Singh, 2004, An activation function adapting training algorithm for sigmoidal feedforward networks: Neurocomputing, v. 61, p. 429-437.

R. Battiti., 1992, First and second order methods for learning: Between steepest descent and Newton□s method: Neural Computation, v. 4, p. 141-166.

R. Fletcher, and C. M. Reeves, 1964, Function minimization by conjugate gradients: Computer Journal, v. 7, p. 149-154.

R. Fletcher, and M. J. D. Powell, 1963, A rapidly convergent descent method for minimization: British Computer J., p. 163-168.

Rezgui A., and Tepedelenlioglu N., 1990, The effect of the slope of the activation function on the back propagation algorithm: In Proceedings of the International Joint Conference on Neural Networks, v. 1, p. 707-710.

Rumelhart D.E., Hinton G.E., and Williams R.J., 1986, Learning internal representations by error propagation: Parallel Dist. Process, v. 1.

S. C. Ng, C. C. Cheung, S. H. Leung, and A. Luk, 2003, Fast convergence for back propagation network with magnified gradient function: Proceedings of the International Joint Conference on Neural Networks 2003, v. 3, p. 1903-1908.

Sang-Hoon Oh, and Youngjik Lee, 1995, A Modified Error Function to Improve the Error Back-Propagation Algorithm for Multi-Layer Perceptrons: ETRI Journal, v. 17, p. 11-22.

Sang Hoon Oh, 1997, Improving the Error Backpropagation Algorithm with a Modified Error Function: IEEE TRANSACTIONS ON NEURAL NETWORKS, v. 8, p. 799-803.

Singh H., and Kumar P., 2005, Optimizing cutting force for turned parts using Taguchi□s parameter design approach: Indian J. Eng. Mater. Sci., v. 12, p. 97-103.

Stuart Geman, Elie Bienenstock, and Rene Doursat, 1992, Neural networks and the bias/variance dilemma.: Neural Computation, v. 4, p. 1-58.

Suzuki K., Horiba I., and Sugie N., 2001, A Simple Neural Network Pruning Algorithm with Application to Filter Synthesis: Neural Processing Letters, v. 13, p. 43-53(11).

T.P. Vogl, J.K. Mangis, A.K. Zigler, W.T. Zink, and D.L. Alkon, 1988, Accelerating the convergence of the backpropagation method: Biol. Cybernet., v. 59, p. 256-264.

Tai-Hoon Cho, Richard W. Conners, and A. Philip A, 1991, Fast Back-Propagation Learning Using Steep Activation Functions and Automatic Weight Reinitialization: Conference Proceedings1991 IEEE International Conference on Systems, Man, and Cybernetics, v. 3, p. 1587-1592.

Theodore T. Allen, and Liyang Yu, 2002, Low-Cost Response Surface Methods from Simulation Optimization: Quality and Reliability Engineering International, v. 18, p. 5-17.

Thimm G., Moerland F., and Emile Fiesler, 1996, The Interchangeability of Learning Rate an Gain in Back propagation Neural Networks: Neural Computation, v. 8, p. 451-460.

V. D. Tsoukalas, St. A. Mavrommatis, N. G. Orfanoudakis, and A. K. Baldoukas, 2004, A study of porosity formation in pressure die casting using the Taguchi approach: Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, v. 218, p. 77-86.

W. L. Buntine, and A. S. Weigend, 1993, Computing second derivatives in feed-forward networks: A review: IEEE Transactions on Neural Networks, v. 5, p. 480-488.

Weir M. K., 1991, A method for self-determination of adaptive learning rates in back propagation.: Neural Networks, p. 371-379.

Werbos P. J., 1974, Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences, Harvard University, USA.

Widrow B., and Lehr M. A., 1990, 30 years of adaptive neural networks:perceptron, madaline,and backpropagation: Proceeding of the IEEE, v. 78, p. 1415-42.

Will Smart, and Mengjie Zhang, 2004, Applying Online Gradient Descent Search to Genetic Programming for Object Recognition: The Australasian Workshop on Data Mining and Web Intelligence(DMWI2004),Conferences in Research and Practice in Information Technology, v. 32.

Wilson R. L., and Sharda R., 1994, Bankruptcy prediction using neural networks: Decision Support Systems, v. 11, p. 545-557.

X.G. Wang, Z. Tang, H. Tamura, M. Ishii, and W.D. Sun, 2004, An improved back propagation algorithm to avoid the local minima problem, Neurocomputing: Elsevier, v. 56, p. 455-460.

Y. LeCun, I. Kanter, and S.A. Solla, 1991, Second order properties of error surfaces: Learning time and generalization: In R.P. Lippmann, J.E. Moody, and D.S. Touretzky, editors, Neural Information Processing Systems, v. 3, p. 918-924.

Yamada K., Kami H., Tsukumo J., and Temma T., 1989, Handwritten numeral recognition by multilayer neural network with improved learning algorithm.: In Proceedings of the International Joint Conference on Neural Networks, v. 2, p. 259-266.

Yarlagadda Prasad K., 2000, Prediction of die casting process parameters by using an artificial neural network model for zinc alloys: International Journal of Production Research, v. 38, p. 119-139.

Zang H. C., and Huang S. H., 1995, Applications of neural networks in manufacturing: A state-of-the-art survey: International Journal of Production Research, v. 33, p. 705-728.

APPENDIX (CONTENTS)

AI : The performance comparison of the proposed method with other optimisation methods

AI: The performance comparison of the proposed method with other optimisation methods

This Appendix is to illustrate the performance of the proposed training method introduced in Chapter Three implemented with other optimisation methods. The performance of the proposed method is tested on Sine curve.

I.1 Sine curve without noise

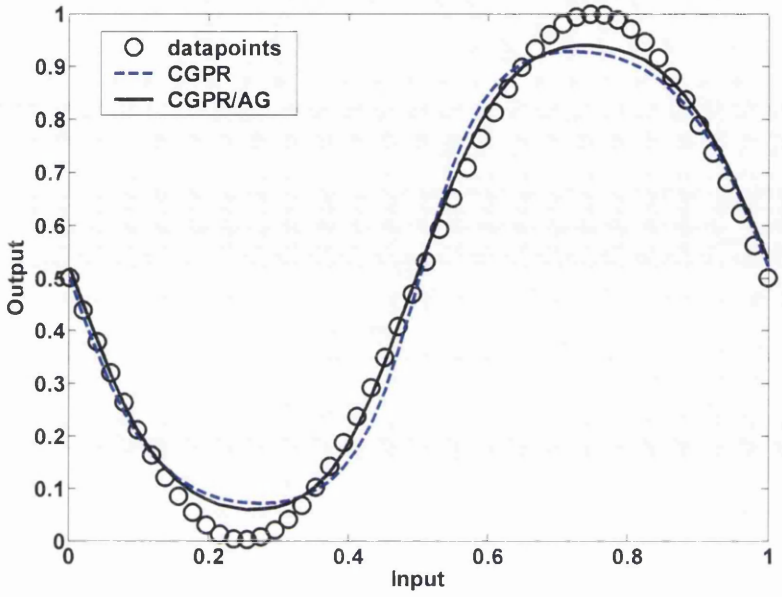


Figure AI.1: Output of neural network trained to learn a sine curve using the proposed conjugate gradient method with Polak-Ribiere formulation.

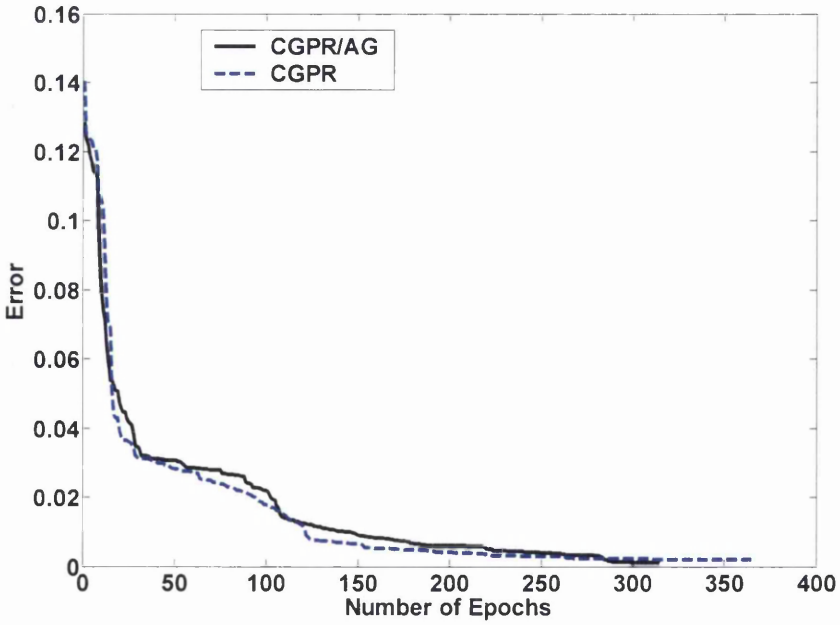


Figure AI.2: Error versus number of epochs required to achieve the target error of 0.001 for conjugate gradient method with Polak-Ribiere formulation.

Comments on AI.1 and AI.2: As can be seen from Figure AII.1 both methods performed almost the same results in learning the data sets. However the proposed method showed a faster result by taking 315 epochs as compared to standard algorithm which need 365 epochs to reach the target error.

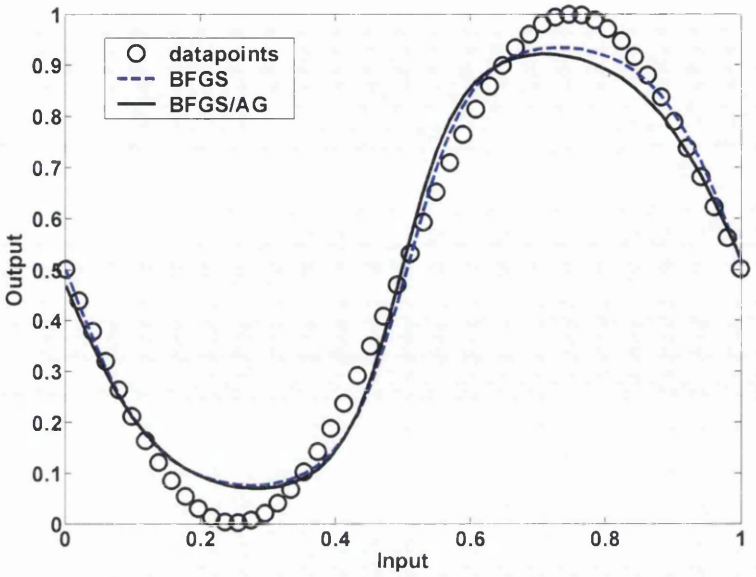


Figure AI.3: Output of neural network trained to learn a sine curve using the proposed conjugate gradient method with Broyden-Fletcher-Goldfarb-Shanno formulation.

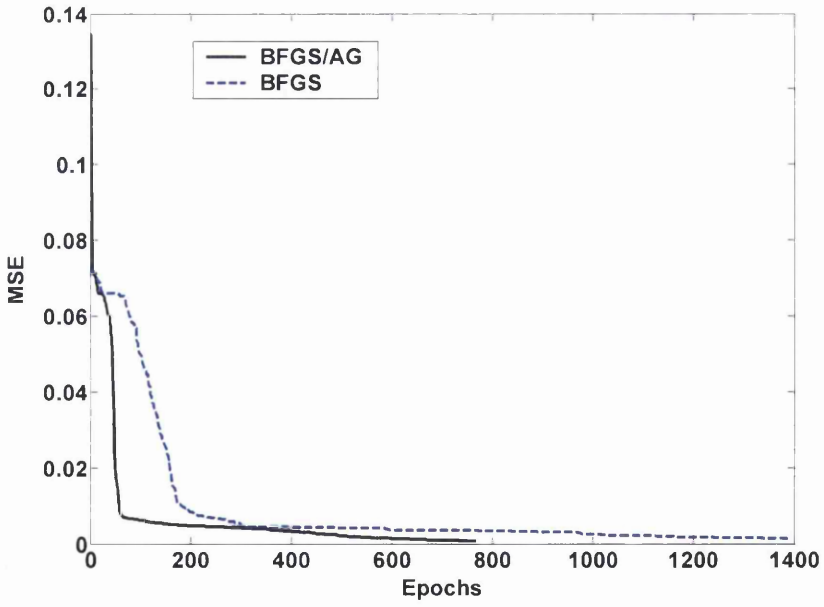


Figure AI.4: Error versus number of epochs required to achieve the target error of 0.001 for conjugate gradient method with Broyden-Fletcher-Goldfarb-Shanno formulation.

Comments on AI.3 and AI.4: Both methods performed same generalisation results but the proposed (BFGS/AG) method significantly reduce number of epochs almost twice faster as compared to standard algorithm (BFGS) with unity gain value.

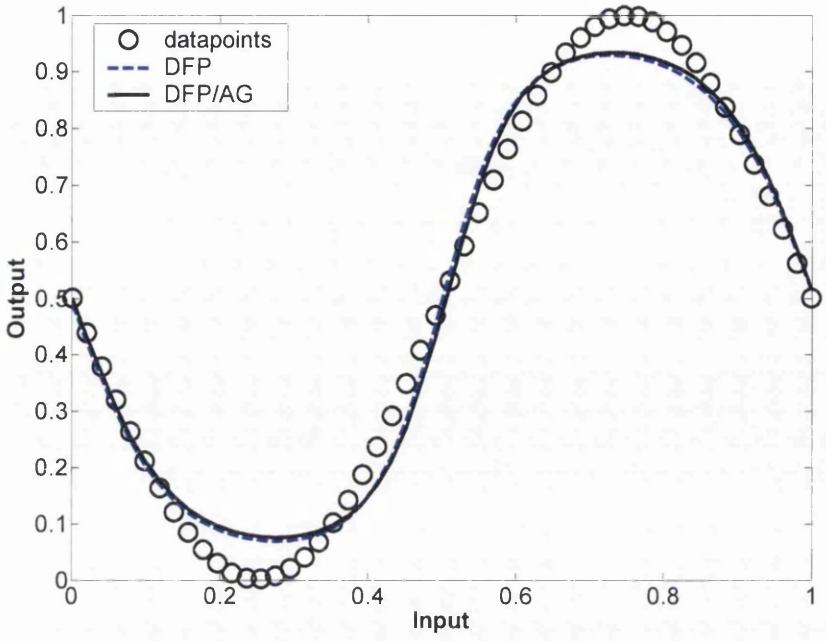


Figure AI.5: Output of neural network trained to learn a sine curve using the proposed conjugate gradient method with Davidon-Fletcher-Power formulation.

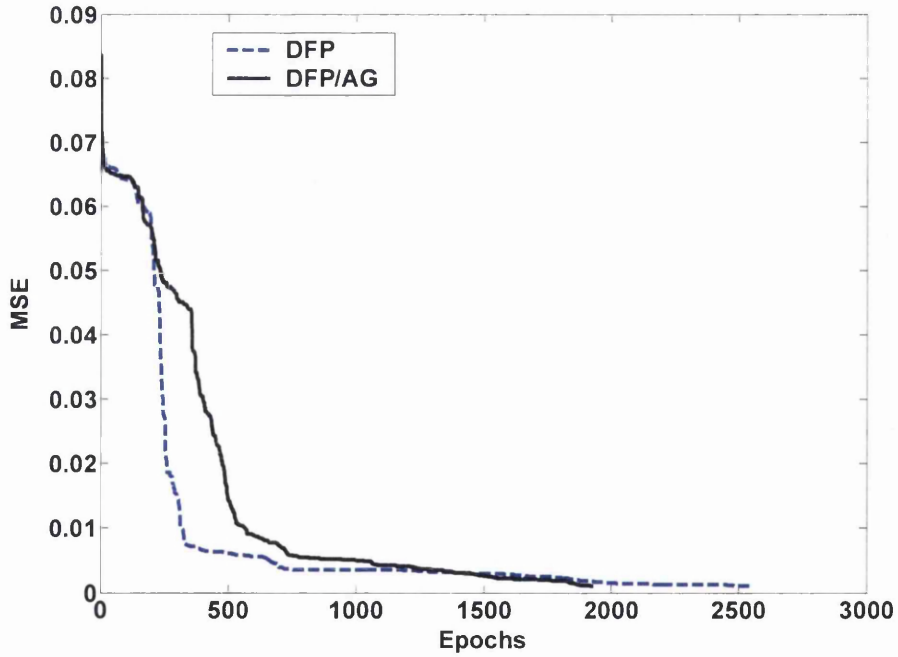


Figure AI.6: Error versus number of epochs required to achieve the target error of 0.001 for conjugate gradient method with Davidon-Fletcher-Power formulation.

Comments on AI.5 and AI.6: The proposed method (DFP/AG) outperformed the standard algorithm (DFP) by taking 1932 epochs to reach the target error as compared to 2546 needed by the standard algorithm (DFP).

II.2 Sine curve with noise

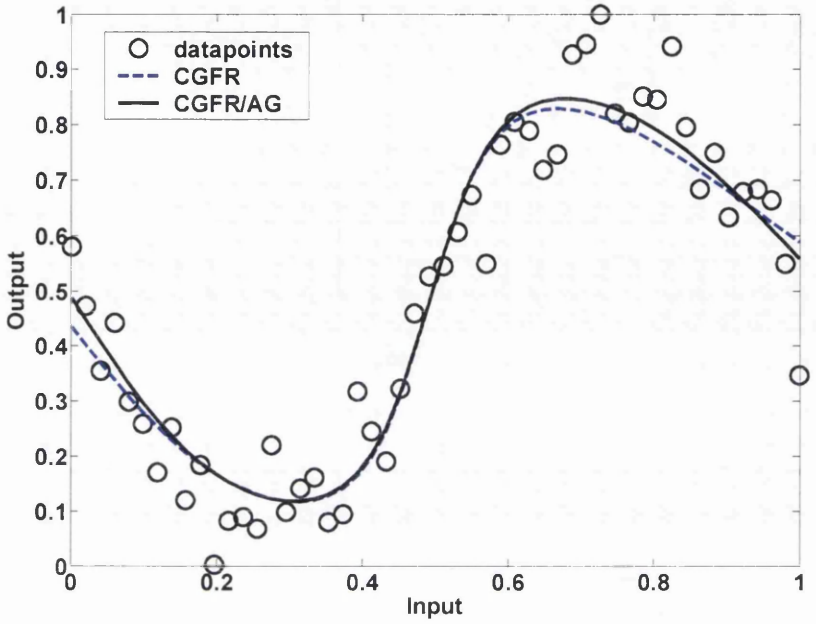


Figure AI.7: Output of neural network trained to learn a sine curve with 20% random Gaussian noise using the proposed method with Fletcher-Reeves formulation.

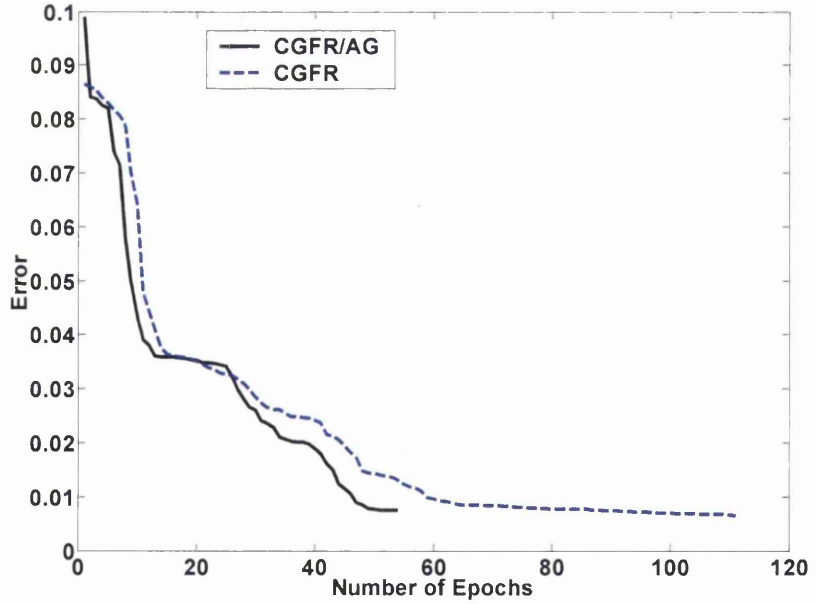


Figure AI.8: Error versus number of epochs required to achieve the target error of 0.01 using conjugate gradient method with Fletcher-Reeves formulation.

Comments on AI.7 and AI.8: Both methods performed almost the same results on generalisation. However, the proposed method (CGPR/AG) only took 111 epochs to achieve the target error as compared to the standard algorithm (CGPR) which took 167 epochs.

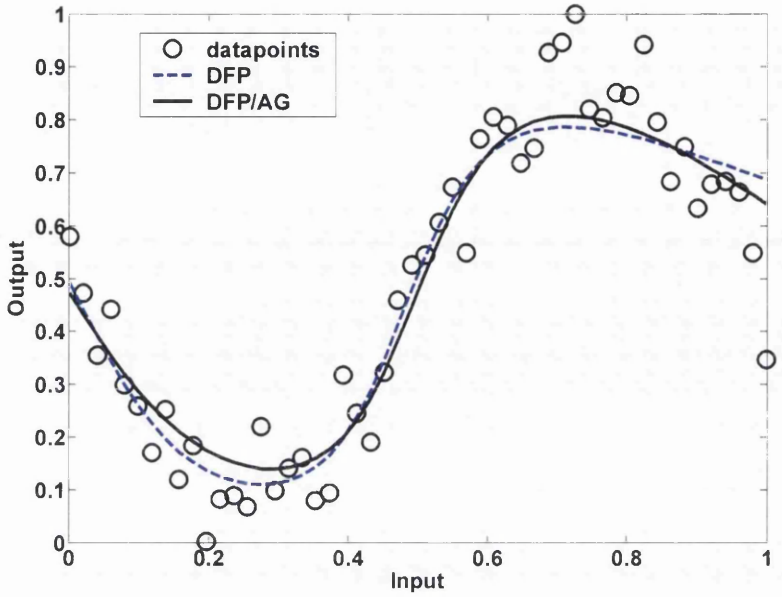


Figure AI.9: Output of neural network trained to learn a sine curve with 20% random Gaussian noise using the proposed method with Davidon-Fletcher-Power formulation.

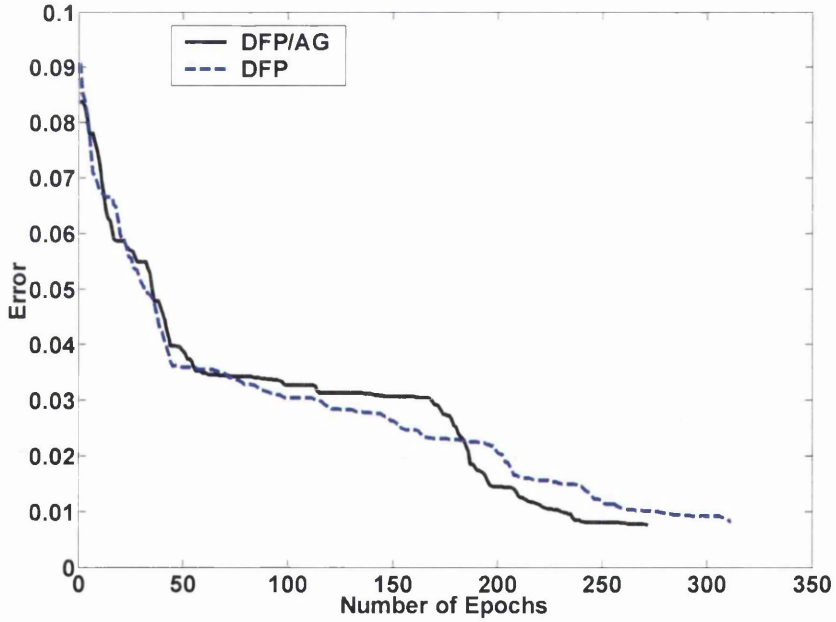


Figure AI.10: Error versus number of epochs required to achieve the target error of 0.01 using conjugate gradient method with Davidon-Fletcher-Power formulation.

Comments on AI.9 and AI.10: The proposed method (DFP/AG) significantly reduced the number of epochs and outperformed the standard algorithm (DFP) for almost 1.5 times faster without losing the generalisation performance.

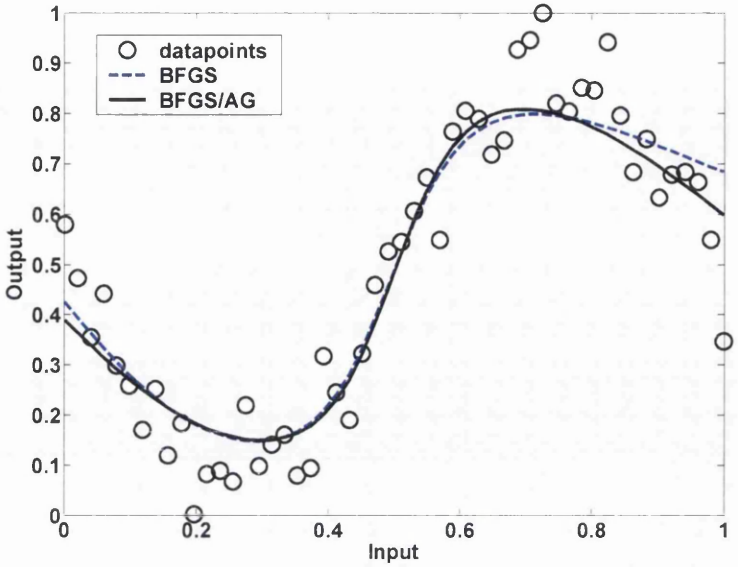


Figure AI.11: Output of neural network trained to learn a sine curve with 20% random Gaussian noise using the proposed method with Broyden-Fletcher-Goldfarb-Shanno formulation.

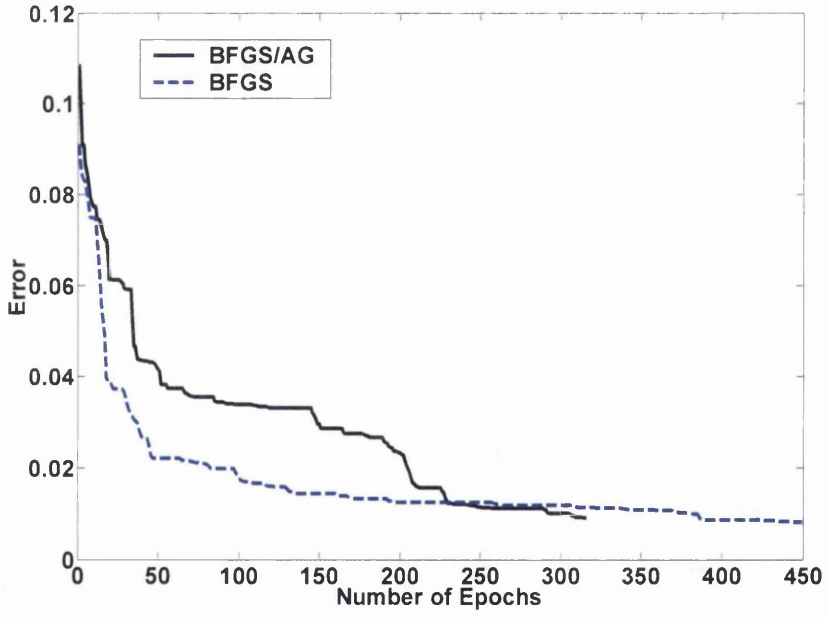


Figure AI.12: Error versus number of epochs required to achieve the target error of 0.01 using conjugate gradient method with Broyden-Fletcher-Goldfarb-Shanno formulation.

Comments on AI.10 and AI.11: The proposed method (BFGS/AG) outperformed the standard algorithm (BFGS) in term of number of epochs with ration of 1.4.

AII: Comparison of the proposed method on benchmark problems

This Appendix is to illustrate the detail calculation procedure for evaluating the performance of the proposed method on benchmark problems as mentioned in Chapter Four.

Table II.1: Detail version of the Conjugate Gradient with Polak-Ribiere formulation performance for Thyroid problem

	traincgp			CGPR			CGPR-AG		
Trials	Epoch	CPU time	accuracy	epoch	CPU time	accuracy	epoch	CPU time	accuracy
1	7	5.44	93.48	12	4.89	90.37	10	3.36	89.10
2	15	6.13	90.74	10	3.42	91.17	9	2.08	89.96
3	248	44.20	92.46	10	3.42	90.14	9	3.94	91.01
4	11	4.31	92.39	11	3.80	90.34	6	2.47	88.65
5	6	4.17	89.89	16	6.56	90.30	11	2.67	90.22
6	12	4.90	89.79	24	8.20	90.68	11	4.11	89.12
7	24	5.72	90.90	10	3.38	90.16	7	3.20	93.95
8	10	3.89	92.17	11	4.44	90.26	10	3.61	92.92
9	165	30.52	91.77	12	4.23	91.02	7	2.50	91.82
10	-	-	-	12	4.23	91.01	9	3.22	89.88
11	286	48.33	92.44	10	3.02	91.03	11	3.55	91.44
12	48	10.88	92.32	12	5.22	90.50	11	3.61	89.41
13	101	18.84	89.68	11	3.91	90.04	8	2.99	89.94
14	7	2.98	91.55	11	4.89	90.57	10	4.17	88.28
15	8	3.03	89.61	10	3.53	90.36	5	1.64	88.34
16	232	45.30	92.14	14	6.23	91.88	5	1.63	92.84
17	6	3.28	93.48	14	6.36	90.58	7	2.34	88.89
18	18	5.09	94.03	11	4.06	90.08	4	1.30	89.85
19	-	-	-	15	6.48	90.88	15	4.89	88.42
20	-	-	-	15	6.70	90.57	16	5.94	90.54
21	139	24.44	90.77	18	7.81	90.26	7	1.59	90.45
22	5	2.52	90.45	11	3.89	91.28	8	3.41	92.70
23	343	61.19	92.81	21	6.11	91.11	9	3.70	90.00
24	6	3.16	90.03	17	6.77	90.57	7	2.80	90.89
25	30	7.48	89.73	13	4.50	90.31	9	1.94	89.08
26	4	4.94	92.32	13	5.56	90.48	5	1.17	90.92
27	8	3.41	91.89	15	6.77	91.65	5	1.06	90.50
28	6	3.23	93.33	10	3.58	90.02	7	1.53	90.80
29	-	-	-	14	5.00	90.04	10	2.36	92.74
30	-	-	-	10	5.47	90.20	8	1.73	90.29
31	6	2.48	91.50	12	2.70	91.02	11	2.36	92.77
32	20	5.75	90.30	12	2.75	91.01	9	2.03	91.23
33	22	5.66	89.57	17	5.98	89.96	8	1.86	89.12
34	-	-	-	15	5.30	90.86	15	3.55	85.81
35	76	13.92	90.22	16	5.95	90.38	10	2.28	90.47
36	12	4.28	94.39	19	6.88	92.02	12	2.81	91.47
37	7	2.94	91.18	16	5.84	90.22	15	3.59	90.14

38	162	30.72	90.12	13	5.95	90.68	4	0.89	90.86
39	11	3.94	92.35	13	2.80	90.41	7	1.67	90.35
40	-	-	-	11	2.31	90.08	11	2.69	91.39
41	27	6.69	91.96	19	8.58	90.88	4	0.91	91.07
42	17	4.78	92.37	11	4.59	90.57	6	1.55	86.96
43	7	3.49	91.04	12	2.66	91.54	6	1.49	81.75
44	5	2.31	91.96	13	3.03	91.07	7	1.52	94.69
45	151	27.23	92.18	15	6.20	91.34	10	2.25	89.58
46	34	7.02	91.72	11	2.59	90.58	8	3.00	84.31
47	5	2.64	90.48	15	6.77	91.37	17	4.02	93.47
48	16	4.22	91.14	-	-	-	12	3.11	90.67
49	5	2.16	93.18	16	7.03	90.26	7	1.69	90.43
50	6	2.33	93.06	11	2.64	91.28	10	2.56	91.58
51	8	2.77	90.60	11	3.33	90.66	14	3.11	90.21
52	19	5.09	90.69	17	6.00	91.11	15	3.36	90.03
53	7	2.94	91.74	-	-	-	8	1.33	89.79
54	6	2.50	90.51	15	4.55	90.31	14	3.08	90.27
55	-	-	-	17	6.95	90.48	7	1.45	90.78
56	6	2.59	91.53	15	5.16	1.65	15	4.13	90.27
57	8	3.06	92.28	10	2.53	90.02	16	3.53	90.84
58	7	2.69	92.21	-	-	-	12	3.11	91.23
59	11	3.52	92.94	15	3.45	90.20	14	3.11	89.44
60	227	34.02	92.26	15	6.69	91.05	14	3.05	89.63
61	8	2.91	91.91	13	3.25	90.84	7	1.50	91.53
62	35	6.81	92.47	12	2.84	90.93	12	2.69	91.51
63	20	5.84	90.94	-	-	-	9	3.42	90.76
64	6	2.19	90.97	12	2.86	90.28	10	2.30	89.17
65	5	2.30	90.43	12	2.88	90.29	16	3.77	89.86
66	-	-	-	11	2.81	90.86	6	1.28	89.66
67	7	2.91	90.97	13	3.33	90.08	10	2.44	90.21
68	7	2.72	91.74	10	2.30	91.83	9	2.16	91.16
69	6	2.50	91.05	10	2.34	91.32	11	2.77	91.39
70	18	4.50	91.18	17	5.01	90.84	8	1.91	92.37
71	7	2.78	91.33	14	4.61	90.16	9	2.11	90.77
72	7	2.52	91.13	10	3.33	91.90	15	3.28	90.30
73	5	2.36	90.69	12	4.02	91.27	12	2.70	93.10
74	7	2.66	91.41	11	4.06	91.04	13	2.81	90.27
75	5	2.33	92.19	15	5.33	90.13	10	2.19	89.99
76	11	2.89	94.12	15	5.92	90.16	9	1.95	91.56
77	13	3.28	89.70	15	5.17	90.17	10	2.24	91.31
78	10	2.91	90.21	12	3.92	91.37	13	2.89	90.44
79	17	4.30	90.19	12	4.94	90.85	-	-	-
80	10	3.09	90.91	13	5.24	91.53	13	2.88	90.34
81	6	2.30	92.64	18	5.94	90.04	10	2.22	90.33
82	17	4.09	95.24	15	5.34	90.39	10	2.17	89.68
83	6	2.50	91.46	11	3.94	90.68	10	2.38	92.43
84	7	2.78	94.17	-	-	-	6	1.41	88.32
85	7	3.11	95.64	10	3.19	90.59	9	1.97	92.61
86	8	2.69	92.14	13	4.94	91.72	8	1.88	90.91
87	25	4.72	91.60	12	4.58	90.68	14	3.02	90.29
88	10	3.25	90.54	20	5.01	90.51	-	-	-
89	5	2.25	90.13	10	3.25	90.78	14	3.33	91.70

90	21	4.59	90.58	-	-	-	14	3.08	90.94
91	18	4.25	89.74	13	3.99	91.34	8	1.66	91.69
92	6	2.48	92.48	13	4.55	91.62	17	3.61	91.20
93	5	2.20	90.65	11	3.48	91.18	15	3.13	87.68
94	5	2.08	90.32	13	4.00	91.47	-	-	-
95	10	3.13	90.73	11	3.39	91.63	10	2.47	88.77
96	9	3.17	92.61	23	8.39	91.35	8	1.81	90.93
97	18	4.99	91.38	12	3.72	91.73	12	2.64	89.92
98	43	7.61	92.68	11	3.59	92.52	12	3.06	91.67
99	10	2.94	94.33	12	3.67	91.06	8	3.19	89.72
100	10	2.80	92.67	10	3.05	91.50	7	1.72	90.33
Mean	**34**	**7.46**	**91.64**	**13**	**4.62**	**89.85**	**10**	**2.58**	**90.37**
SD	**66**	**10.88**		**3.28**	**1.85**		**3.53**	**1.01**	
succ.	**91**			**94**			**97**		
fail	**9**			**6**			**3**		

Table II.2: Detail version of the Conjugate Gradient with Polak-Ribiere formulation performance for Cancer problem

	traincgp			CGPR			CGPR-AG		
Trials	epoch	CPU time	accuracy	epoch	CPU time	accuracy	epoch	CPU time	accuracy
1	9	4.20	89.31	25	1.11	89.73	14	0.63	89.50
2	11	4.34	89.56	27	1.19	89.66	21	0.94	98.06
3	45	4.77	88.45	42	1.91	89.87	9	0.38	89.75
4	20	3.45	87.57	25	1.24	89.69	23	1.03	89.76
5	16	2.78	89.37	41	2.09	89.91	19	0.84	98.10
6	-	-	-	24	1.08	89.87	16	0.70	89.75
7	12	2.02	89.87	32	1.58	89.71	28	1.23	89.89
8	43	3.21	89.33	24	1.06	89.66	20	0.91	89.95
9	13	1.94	89.55	25	1.11	89.73	9	0.38	89.58
10	32	3.34	88.11	43	2.14	89.68	14	0.61	89.74
11	16	2.33	89.78	29	1.30	89.77	35	1.61	89.92
12	18	2.00	89.51	27	1.22	89.77	29	1.33	89.74
13	71	3.63	89.51	25	1.11	98.10	27	1.22	89.73
14	17	2.88	89.66	28	1.41	89.79	31	1.41	89.88
15	18	2.75	89.93	68	3.34	89.78	26	1.19	89.73
16	12	2.59	89.78	49	2.23	89.95	19	0.84	89.74
17	15	3.53	89.75	27	1.19	89.77	28	1.27	95.01
18	17	5.72	89.12	23	1.11	89.78	9	0.38	89.81
19	21	5.23	88.00	33	1.63	89.77	23	1.05	89.94
20	11	2.77	98.03	42	1.94	89.78	23	1.05	89.75
21	-	-	-	24	1.08	89.75	17	0.75	89.61
22	15	2.75	89.55	25	1.23	89.70	21	0.95	89.88
23	14	2.45	89.71	27	1.34	89.73	25	1.13	89.83
24	13	2.64	98.01	26	1.16	89.63	60	2.89	89.82
25	21	2.19	89.58	46	2.11	89.75	19	0.86	89.66
26	19	2.33	89.84	23	1.03	89.68	32	1.47	89.78
27	17	2.88	89.60	23	1.02	89.90	-	-	-
28	34	3.22	87.45	26	1.16	89.69	24	1.09	89.69
29	52	3.47	89.49	24	1.16	89.85	29	1.33	89.94
30	31	2.80	89.85	25	1.22	89.75	19	0.86	89.92
31	13	2.20	98.07	106	5.39	89.83	60	2.89	95.49
32	16	2.39	89.57	-	-	-	23	1.11	89.72
33	18	2.39	89.89	84	4.22	89.83	19	0.86	89.94
34	27	2.48	89.61	26	1.28	89.69	29	1.33	89.85
35	45	4.22	89.57	42	1.92	89.81	92	4.73	89.81
36	145	5.86	89.73	26	1.31	89.80	20	0.91	89.84
37	12	2.11	89.50	24	1.08	89.69	33	1.50	95.34
38	22	2.39	89.83	24	1.08	89.80	24	1.08	89.68
39	25	2.34	89.45	31	1.41	89.79	18	0.81	89.80
40	28	2.77	88.55	38	1.91	89.77	27	1.22	89.83
41	12	2.03	89.53	32	1.59	89.75	21	0.92	89.68
42	16	2.08	89.57	50	2.52	89.82	17	0.77	89.96
43	34	2.61	89.30	24	1.16	89.66	43	2.00	89.89
44	14	2.38	89.43	25	1.11	89.75	15	0.69	89.43
45	16	2.17	89.80	39	1.78	89.72	14	0.61	89.73

46	36	2.72	89.65	31	1.53	89.86	12	0.52	89.88
47	22	2.33	89.81	44	2.20	89.82	16	0.69	89.81
48	52	3.00	89.87	22	1.08	89.80	25	1.14	89.73
49	22	2.45	89.55	37	1.67	89.78	33	1.53	89.61
50	19	2.09	89.45	34	1.55	89.74	35	1.61	98.01
51	12	1.80	89.69	22	1.08	89.62	11	0.47	89.90
52	18	2.52	89.38	35	1.78	89.98	34	1.56	98.15
53	17	2.19	89.42	43	2.19	89.71	13	0.58	89.75
54	12	2.50	89.21	23	1.14	89.73	18	0.78	89.74
55	12	2.00	89.90	22	1.02	89.66	18	0.81	89.62
56	-	-	-	23	1.17	89.81	24	1.09	89.78
57	12	2.95	89.76	30	1.33	89.95	20	0.91	89.78
58	22	2.41	89.60	24	1.06	89.77	19	0.84	89.63
59	-	-	-	21	1.05	98.13	22	1.00	89.89
60	18	2.02	89.77	28	1.24	89.74	31	1.42	89.81
61	18	2.11	89.42	29	1.28	89.71	14	0.63	89.78
62	260	9.02	98.13	46	2.05	89.85	11	0.47	89.89
63	33	2.83	89.69	24	1.03	89.73	20	0.88	89.72
64	15	2.02	89.75	28	1.38	89.69	13	0.58	89.76
65	16	1.89	89.88	34	1.52	89.64	41	1.89	89.79
66	59	3.19	89.68	103	5.20	89.70	37	1.72	89.79
67	34	4.01	89.12	27	1.20	89.78	23	1.05	89.80
68	19	2.03	89.69	-	-	-	23	1.02	89.65
69	10	2.44	89.49	33	1.47	89.72	22	1.00	89.81
70	12	1.86	89.60	24	1.17	89.73	20	0.88	89.78
71	39	3.09	89.74	39	1.95	89.76	23	1.05	89.78
72	29	2.45	89.57	37	1.86	89.71	28	1.28	89.69
73	19	2.13	89.66	28	1.39	89.58	13	0.56	89.85
74	273	8.47	98.00	77	3.86	89.68	33	1.53	89.79
75	12	2.81	89.84	22	1.08	89.88	37	1.73	89.83
76	18	2.09	89.71	35	1.78	89.65	-	-	-
77	12	1.80	98.07	43	2.19	89.73	32	1.63	89.95
78	20	2.25	89.85	26	1.17	89.73	25	1.11	89.86
79	21	2.39	89.87	21	1.05	89.76	15	0.66	89.72
80	8	1.92	98.41	29	1.30	89.81	29	1.47	89.78
81	22	2.45	89.91	26	1.16	89.67	12	0.53	89.91
82	128	5.64	89.62	28	1.38	89.75	17	0.77	89.89
83	19	2.16	89.45	27	1.20	89.76	17	0.75	89.78
84	19	3.72	89.90	28	1.39	89.84	27	1.22	89.75
85	89	4.39	89.74	32	1.44	89.74	29	1.31	98.12
86	12	2.19	89.82	22	1.11	89.81	15	0.66	89.74
87	16	2.08	89.73	23	1.02	89.61	25	1.25	89.87
88	12	1.92	98.47	31	1.55	89.90	14	0.61	89.82
89	14	2.06	89.83	24	1.17	89.69	21	0.97	90.00
90	15	2.13	98.11	29	1.28	89.76	25	1.14	89.88
91	11	2.03	98.14	37	1.86	89.80	15	0.67	89.70
92	53	3.25	89.75	22	1.11	89.76	31	1.42	89.96
93	10	2.00	89.97	24	1.14	89.77	20	0.89	89.78
94	12	1.92	89.77	51	2.63	89.70	15	0.64	89.77
95	16	2.14	89.56	25	1.23	89.81	24	1.09	89.89
96	12	6.28	98.00	32	1.45	89.68	14	0.61	89.73
97	16	2.19	89.67	22	1.08	89.73	12	0.53	89.68

98	27	2.58	89.70	28	1.27	89.63	25	1.13	89.87
99	21	2.31	89.38	33	1.50	89.70	13	0.56	89.55
100	11	2.56	89.74	24	1.05	89.79	33	1.50	89.89
Mean	**29**	**2.82**	**90.49**	**33**	**1.56**	**89.93**	**24**	**1.08**	**90.38**
SD	**41**	**1.38**		**15.11**	**0.80**		**11.58**	**0.60**	
succ.	**98**			**98**			**98**		
fail	**3**			**2**			**2**		

Table II.3: Detail version of the Conjugate Gradient with Fletcher-Reeves formulation performance for Diabetes problem

	traincgf			CGFR			CGFR-AG		
Trials	epoch	CPU time	accuracy	epoch	CPU time	accuracy	epoch	CPU time	accuracy
1	64	6.28	91.58	81	3.94	91.36	29	1.30	92.32
2	66	5.30	91.81	47	2.16	92.04	38	1.70	91.78
3	-	-	-	48	2.19	90.89	25	1.16	88.70
4	104	3.39	91.74	46	2.09	91.31	29	1.28	92.68
5	77	3.59	92.25	50	2.28	91.50	58	2.70	90.92
6	76	4.81	92.04	59	2.78	90.25	27	1.33	90.96
7	106	3.48	92.08	114	6.92	58.17	-	-	-
8	229	5.88	91.90	58	2.72	90.38	17	0.75	92.56
9	32	2.13	92.49	35	1.58	91.37	48	2.17	90.89
10	25	2.20	92.19	67	3.19	91.52	48	2.20	89.39
11	359	8.72	91.72	65	3.13	90.84	46	2.09	89.82
12	60	2.78	92.31	57	2.67	91.36	37	1.66	88.63
13	63	2.81	92.33	24	1.06	91.62	22	0.97	91.83
14	72	2.92	91.77	39	1.75	91.74	28	1.25	92.64
15	72	2.58	91.53	26	1.16	90.48	43	1.92	90.19
16	75	2.69	92.34	54	2.52	90.30	61	2.89	90.04
17	61	2.36	92.57	66	3.17	90.80	71	3.39	88.73
18	82	2.95	91.95	46	2.13	90.97	22	0.97	92.67
19	60	2.39	92.91	92	5.36	92.59	46	2.13	89.54
20	27	2.05	91.22	54	2.47	91.35	46	2.08	90.54
21	31	2.31	91.35	47	2.39	90.19	32	1.61	91.53
22	251	8.41	91.59	46	2.33	90.45	44	2.17	91.85
23	64	3.05	91.10	34	1.69	91.67	33	1.63	92.16
24	67	2.84	91.51	47	2.39	84.15	28	1.38	90.04
25	105	3.73	91.22	27	1.34	92.10	55	2.89	85.45
26	47	2.99	91.28	39	1.91	91.36	26	1.31	92.16
27	27	2.16	91.16	77	4.02	91.17	44	2.31	90.43
28	75	3.14	91.03	40	2.02	89.93	37	1.88	93.01
29	56	2.80	91.49	58	3.00	90.83	46	2.41	93.22
30	73	3.13	91.31	38	1.94	91.20	28	1.39	90.99
31	524	16.91	91.35	54	2.88	90.56	54	2.80	89.29
32	182	7.81	91.23	43	2.19	91.48	38	1.94	91.41
33	63	5.11	91.38	46	2.33	90.96	25	1.23	92.47
34	70	3.84	91.38	45	2.41	90.82	37	1.77	92.45
35	73	3.33	91.42	36	1.80	91.08	21	1.06	91.93
36	63	2.97	91.51	32	1.63	91.14	42	2.08	90.02
37	115	4.72	91.39	55	2.94	90.26	53	2.77	90.29
38	66	3.33	91.25	49	2.55	90.73	43	2.22	93.22
39	83	3.39	91.23	66	3.58	90.25	28	1.49	92.68
40	39	2.88	91.51	37	1.91	91.11	28	1.39	91.65
41	60	3.49	91.51	-	-	-	53	2.77	92.19
42	28	3.41	91.25	51	2.67	91.51	41	2.09	90.99
43	82	3.34	91.15	45	2.25	88.87	54	2.86	91.60
44	43	2.70	91.21	64	3.50	91.40	-	-	-
45	144	4.98	91.30	79	4.63	82.36	34	2.55	90.32
46	-	-	-	66	3.39	90.84	20	0.99	91.93

47	92	3.59	91.43	45	2.23	91.22	19	0.92	91.91
48	120	4.30	91.34	54	2.72	91.18	20	0.95	91.93
49	121	4.56	91.46	56	2.78	90.11	64	3.50	90.74
50	74	3.38	91.18	76	4.25	91.12	24	1.20	92.50
51	61	2.73	91.19	51	2.66	90.38	73	4.13	90.95
52	67	3.09	91.18	54	2.84	91.93	33	1.66	91.70
53	208	6.34	91.48	43	2.19	91.82	20	0.99	92.39
54	60	2.83	91.31	61	3.30	92.04	46	2.09	90.73
55	57	3.00	91.17	85	4.81	89.17	35	1.77	91.70
56	236	7.88	91.58	37	1.86	91.19	19	1.91	93.60
57	340	10.58	91.22	40	2.02	89.67	39	1.91	91.77
58	132	4.73	91.80	71	3.98	89.27	51	2.56	90.39
59	215	6.20	91.23	82	4.67	91.00	48	2.45	90.18
60	60	2.70	91.45	23	1.13	92.64	21	0.91	91.55
61	345	10.86	91.17	51	2.66	91.36	34	1.74	91.86
62	25	3.61	91.34	49	2.53	90.19	-	-	-
63	97	3.73	91.26	51	2.67	90.93	33	1.69	92.87
64	61	2.61	91.38	-	-	-	43	2.17	92.34
65	61	2.77	91.69	43	2.23	91.38	56	2.83	88.19
66	-	-	-	64	3.41	90.21	42	2.14	90.01
67	55	2.86	91.25	57	3.09	89.98	48	2.53	90.82
68	70	3.06	91.44	57	3.06	93.38	55	2.84	80.46
69	28	2.19	91.16	20	0.95	91.71	52	2.61	90.88
70	64	3.02	91.33	64	3.31	92.01	23	1.14	92.22
71	62	2.70	91.34	51	2.55	90.91	22	1.08	92.10
72	69	3.16	91.41	42	2.11	90.60	39	1.99	91.50
73	86	3.50	91.21	55	2.75	89.69	26	1.27	92.87
74	59	2.77	91.59	40	1.97	91.08	47	2.41	91.12
75	25	2.28	91.49	43	2.13	91.00	47	2.42	90.30
76	32	2.34	91.43	29	1.39	91.81	19	0.92	88.86
77	90	3.69	91.24	102	5.99	71.14	44	2.19	91.00
78	121	4.56	91.32	37	1.84	92.93	37	1.86	80.36
79	59	2.77	91.29	47	2.34	90.54	42	2.13	89.47
80	117	4.81	91.64	42	2.08	90.19	60	3.27	91.12
81	25	2.05	91.20	-	-	-	28	1.42	92.05
82	59	2.97	91.71	25	1.23	92.08	36	1.89	91.69
83	62	5.38	91.77	66	3.61	90.97	-	-	-
84	-	-	-	52	2.69	90.38	65	3.55	90.81
85	232	7.56	91.53	26	1.30	91.44	27	1.31	91.75
86	49	2.97	91.42	27	1.33	91.64	18	0.89	93.11
87	60	2.55	91.23	51	2.67	72.74	23	1.14	93.11
88	102	3.86	91.29	36	1.81	90.92	29	1.42	92.28
89	115	3.89	91.85	55	2.94	89.87	63	3.39	89.59
90	76	3.39	91.28	45	2.27	90.60	22	1.09	91.55
91	60	2.98	91.62	21	1.03	90.74	15	0.73	92.18
92	66	3.02	91.09	41	2.08	92.79	46	2.27	90.22
93	65	2.91	91.39	61	3.38	90.83	70	3.91	87.02
94	223	6.86	91.42	30	1.50	91.97	55	2.92	89.93
95	70	3.14	91.10	39	1.99	91.28	50	2.59	90.05
96	46	2.69	91.32	50	2.59	90.22	52	2.59	90.47
97	32	2.30	91.23	54	2.88	90.05	52	2.72	83.15
98	-	-	-	70	3.88	75.92	114	6.70	87.54

99	390	11.89	91.58	50	2.59	90.44	40	2.02	81.91
100	67	3.00	91.52	39	1.98	90.79	31	1.56	90.75
Mean	**98**	**4.03**	**91.50**	**51**	**2.61**	**89.97**	**40**	**2.01**	**90.70**
SD	**87**	**2.48**		**17.03**	**1.11**		**15.94**	**0.97**	
succ.	**95**			**97**			**96**		
Fail	**5**			**3**			**4**		

Table II.4: Detail version of the Conjugate Gradient with Polak-Ribiere formulation performance for IRIS problem

	traincgp			CGPR			CGPR-AG		
Trials	Epoch	CPU time	accuracy	epoch	CPU time	accuracy	epoch	CPU time	accuracy
1	13	3.58	95.54	25	1.11	97.73	14	0.63	97.50
2	18	2.59	96.18	27	1.19	97.66	21	0.94	98.06
3	14	3.08	95.85	42	1.91	97.87	9	0.38	97.75
4	13	2.16	95.80	22	0.97	97.69	23	1.03	97.76
5	17	2.88	95.05	13	0.55	97.91	19	0.84	98.10
6	29	2.47	98.96	24	1.08	97.87	16	0.70	97.75
7	14	2.25	95.92	18	0.80	97.71	28	1.23	97.89
8	42	2.45	99.66	24	1.06	97.66	20	0.91	97.95
9	37	2.55	97.48	25	1.11	97.73	9	0.38	97.58
10	57	2.88	96.29	21	0.94	97.68	14	0.61	97.74
11	45	2.86	96.57	29	1.30	97.77	35	1.61	97.92
12	-	-	-	27	1.22	97.77	29	1.33	97.74
13	22	2.14	95.52	25	1.11	98.10	27	1.22	97.73
14	17	2.47	96.11	18	0.78	97.79	31	1.41	97.88
15	11	2.30	97.71	68	3.34	97.78	26	1.19	97.73
16	24	2.30	95.89	49	2.23	97.95	19	0.84	97.74
17	91	3.74	99.81	27	1.19	97.77	28	1.27	95.01
18	11	2.06	95.30	22	0.97	97.78	9	0.38	97.81
19	21	2.31	95.13	20	0.89	97.77	23	1.05	97.94
20	-	-	-	42	1.94	97.78	23	1.05	97.75
21	16	2.25	96.15	24	1.08	97.75	17	0.75	97.61
22	44	3.17	97.78	15	0.66	97.70	21	0.95	97.88
23	55	2.89	97.46	20	0.89	97.73	25	1.13	97.83
24	16	2.08	96.70	26	1.16	97.63	60	2.89	97.82
25	37	2.80	97.34	46	2.11	97.75	19	0.86	97.66
26	14	2.34	97.43	23	1.03	97.68	32	1.47	97.78
27	26	2.50	98.30	23	1.02	97.90	26	1.19	97.79
28	384	10.55	95.57	26	1.16	97.69	24	1.09	97.69
29	17	2.17	95.23	19	0.84	97.85	29	1.33	97.94
30	33	2.55	99.46	18	0.80	97.75	19	0.86	97.92
31	55	2.94	98.82	106	5.39	97.83	60	2.89	95.49
32	16	2.39	96.87	23	1.03	97.73	23	1.11	97.72
33	470	12.38	95.16	84	4.22	97.83	19	0.86	97.94
34	17	2.11	95.42	14	0.61	97.69	29	1.33	97.85
35	16	2.20	95.23	42	1.92	97.81	92	4.73	97.81
36	47	2.91	97.69	17	0.75	97.80	20	0.91	97.84
37	39	2.73	95.23	24	1.08	97.69	33	1.50	95.34
38	15	2.05	95.01	24	1.08	97.80	24	1.08	97.68
39	16	2.06	95.19	31	1.41	97.79	18	0.81	97.80
40	129	4.27	95.29	19	0.84	97.77	27	1.22	97.83
41	-	-	-	15	0.64	97.75	21	0.92	97.68
42	17	2.23	95.17	17	0.75	97.82	17	0.77	97.96
43	21	2.23	95.66	18	0.78	97.66	43	2.00	97.89
44	27	3.23	96.71	25	1.11	97.75	15	0.69	97.43
45	18	1.98	95.60	39	1.78	97.72	14	0.61	97.73
46	49	4.30	98.17	20	0.88	97.86	12	0.52	97.88

47	22	3.17	95.22	21	0.94	97.82	16	0.69	97.81
48	21	2.42	95.92	19	0.84	97.80	25	1.14	97.73
49	13	2.80	95.06	37	1.67	97.78	33	1.53	97.61
50	42	2.80	95.30	34	1.55	97.74	35	1.61	98.01
51	18	2.16	95.73	18	0.80	97.62	11	0.47	97.90
52	101	3.69	95.76	17	0.74	97.98	34	1.56	98.15
53	-	-	-	22	0.97	97.71	13	0.58	97.75
54	10	2.19	95.05	17	0.75	97.73	18	0.78	97.74
55	37	2.95	97.51	-	-	-	18	0.81	97.62
56	71	3.86	95.22	19	0.88	97.81	24	1.09	97.78
57	26	2.41	95.38	30	1.33	97.95	20	0.91	97.78
58	41	2.67	96.37	24	1.06	97.77	19	0.84	97.63
59	-	-	-	15	0.66	98.13	22	1.00	97.89
60	11	2.17	95.67	28	1.24	97.74	31	1.42	97.81
61	11	5.58	95.25	29	1.28	97.71	14	0.63	97.78
62	-	-	-	46	2.05	97.85	11	0.47	97.89
63	11	2.16	95.11	24	1.03	97.73	20	0.88	97.72
64	33	2.42	96.25	20	0.88	97.69	13	0.58	97.76
65	78	3.30	98.92	34	1.52	97.64	41	1.89	97.79
66	12	3.06	95.86	103	5.20	97.70	37	1.72	97.79
67	47	2.81	95.51	27	1.20	97.78	23	1.05	97.80
68	9	3.27	96.31	25	1.11	97.80	23	1.02	97.65
69	27	4.73	95.29	33	1.47	97.72	-	-	-
70	9	3.44	95.38	19	0.83	97.73	20	0.88	97.78
71	-	-	-	18	0.78	97.76	23	1.05	97.78
72	12	3.05	95.22	20	0.89	97.71	28	1.28	97.69
73	30	2.42	96.95	18	0.78	97.58	13	0.56	97.85
74	18	2.03	95.02	77	3.86	97.68	-	-	-
75	23	2.28	95.20	17	0.75	97.88	37	1.73	97.83
76	14	2.39	95.02	18	0.77	97.65	42	1.95	97.75
77	47	3.42	95.91	-	-	-	9	0.38	97.95
78	20	1.92	96.40	26	1.17	97.73	25	1.11	97.86
79	48	2.77	97.50	19	0.84	97.76	15	0.66	97.72
80	18	2.64	97.82	29	1.30	97.81	7	0.28	97.78
81	39	2.42	96.35	26	1.16	97.67	12	0.53	97.91
82	-	-	-	21	0.94	97.75	17	0.77	97.89
83	21	2.77	95.10	27	1.20	97.76	17	0.75	97.78
84	33	2.94	97.22	21	0.94	97.84	27	1.22	97.75
85	68	3.50	96.73	32	1.44	97.74	29	1.31	98.12
86	78	3.22	96.38	20	0.88	97.81	15	0.66	97.74
87	40	2.80	97.62	23	1.02	97.61	9	0.41	97.87
88	30	4.30	95.56	16	0.70	97.90	14	0.61	97.82
89	-	-	-	-	-	-	21	0.97	98.00
90	10	3.05	95.02	29	1.28	97.76	25	1.14	97.88
91	16	2.38	95.88	20	0.89	97.80	15	0.67	97.70
92	10	3.16	95.41	12	0.52	97.76	31	1.42	97.96
93	20	2.20	96.49	21	0.94	97.77	20	0.89	97.78
94	-	-	-	18	0.77	97.70	15	0.64	97.77
95	32	4.61	98.98	22	0.95	97.81	24	1.09	97.89
96	21	2.19	95.93	32	1.45	97.68	-	-	-
97	60	2.73	95.52	22	0.97	97.73	12	0.53	97.68
98	16	5.30	96.57	28	1.27	97.63	25	1.13	97.87

99	12	2.16	95.01	-	-	-	13	0.56	97.55
100	35	2.27	96.62	24	1.05	97.79	33	1.50	97.89
Mean	**39**	**2.99**	**96.28**	**28**	**1.25**	**97.77**	**23**	**1.06**	**97.73**
SD	**63**	**1.67**		**16.33**	**0.86**		**11.97**	**0.62**	
Succ.	**90**			**96**			**97**		
fail	**10**			**4**			**3**		

Table II.5: Detail version of the Broyden-Fletcher-Goldfarb-Shanno performance for 7 bit Parity problem

	trainbfg			BFGS			BFGS-AG		
Trials	epoch	CPU time	accuracy	epoch	CPU time	accuracy	epoch	CPU time	accuracy
1	138	4.22	91.32	86	3.67	91.02	89	4.00	90.29
2	167	7.25	90.78	84	3.31	91.02	84	3.66	90.11
3	213	5.70	91.47	85	3.41	91.17	82	3.44	90.02
4	67	2.58	87.05	87	3.81	89.06	84	3.48	89.02
5	286	6.94	89.82	85	3.75	91.07	85	3.58	90.06
6	70	2.80	84.95	86	3.78	89.00	84	3.44	91.05
7	228	6.08	90.56	90	3.95	90.05	85	3.64	91.02
8	84	3.14	89.18	80	3.48	90.01	81	3.66	89.11
9	179	4.91	92.50	-	-	-	87	3.92	91.02
10	186	4.97	93.33	84	3.67	90.11	84	3.83	91.24
11	239	5.91	91.90	94	4.20	90.04	85	3.81	90.02
12	127	3.59	90.26	95	4.24	90.50	86	3.84	91.02
13	409	9.78	84.27	98	4.39	91.04	84	3.80	90.06
14	258	6.50	89.14	96	4.28	91.07	82	3.67	91.05
15	76	2.70	90.81	93	4.11	89.03	84	3.77	89.02
16	218	5.58	82.47	90	3.69	91.22	98	4.50	90.11
17	303	7.41	89.00	91	3.89	91.09	83	3.69	91.02
18	109	3.59	88.48	96	4.33	91.00	86	4.39	91.24
19	170	4.61	83.57	92	4.13	91.02	83	3.70	91.38
20	196	5.83	85.33	90	4.00	89.01	85	3.81	90.00
21	100	3.13	88.14	90	3.95	91.02	84	3.77	91.27
22	73	2.67	89.09	91	4.03	90.17	84	3.73	91.16
23	256	5.92	85.45	99	4.44	90.06	86	3.86	90.48
24	81	2.77	91.78	90	3.94	90.07	86	3.92	91.12
25	97	3.09	88.12	95	4.20	90.00	89	5.47	91.04
26	112	3.24	89.30	92	4.03	90.05	82	3.50	91.03
27	288	6.41	93.33	99	4.42	91.10	82	3.97	89.45
28	245	5.64	90.13	96	4.33	91.12	83	3.70	91.39
29	227	5.24	83.89	95	4.53	91.04	87	4.16	91.74
30	147	3.67	91.47	99	4.55	88.99	83	3.45	88.00
31	352	7.78	83.85	88	5.34	91.14	85	3.58	91.01
32	174	4.28	87.09	93	4.81	91.04	85	3.58	92.32
33	70	2.58	85.75	101	4.17	89.44	83	3.44	91.12
34	344	7.27	81.97	94	3.88	91.42	94	4.95	90.05
35	68	2.58	87.76	96	3.91	91.06	82	3.39	91.07
36	65	2.38	89.49	97	4.48	91.07	87	3.67	89.01
37	169	3.99	89.91	96	4.31	91.10	84	3.86	89.45
38	107	3.30	87.70	93	3.81	91.09	86	3.72	91.06
39	212	5.02	87.58	96	4.20	87.98	90	3.80	91.19
40	260	5.84	90.00	90	3.80	91.01	82	3.44	91.11
41	305	6.77	88.50	93	4.03	91.00	82	3.77	91.34
42	83	2.92	87.08	92	3.91	90.12	79	3.49	90.02
43	91	2.88	92.53	93	4.02	89.03	84	3.55	90.09
44	422	8.97	88.33	92	3.70	91.36	91	3.86	89.00
45	270	6.09	84.24	92	3.83	89.06	85	3.58	91.15
46	221	5.05	93.00	103	4.44	91.05	-	-	-

47	272	6.09	91.17	94	3.97	90.11	82	3.42	91.12
48	336	7.28	92.79	93	3.84	89.68	85	3.56	91.12
49	287	6.30	83.60	95	3.94	91.11	84	3.53	89.11
50	144	3.78	91.00	97	3.98	91.03	84	3.52	91.47
51	-	-	-	95	4.14	87.57	82	3.42	91.01
52	125	3.22	89.40	93	4.14	91.13	88	3.69	91.18
53	63	2.34	90.14	95	4.31	91.04	82	3.42	91.13
54	141	3.63	90.63	93	4.17	91.04	80	3.33	91.73
55	180	4.33	90.00	93	4.16	90.06	86	3.61	91.14
56	111	3.14	91.52	94	4.22	87.22	85	3.63	91.03
57	51	3.72	87.32	96	4.31	91.05	89	4.08	91.03
58	70	2.53	89.63	94	4.19	91.09	88	4.05	91.45
59	133	3.80	80.61	96	4.34	91.10	82	3.77	91.02
60	248	5.83	87.11	93	4.14	91.23	83	3.80	91.14
61	64	7.44	86.94	92	4.13	91.00	89	4.06	91.04
62	88	3.31	90.13	97	4.38	91.04	91	4.17	91.02
63	104	3.30	88.49	95	4.30	91.05	84	3.80	91.07
64	250	5.72	84.97	95	4.00	90.28	84	3.80	90.09
65	219	5.25	81.26	96	4.06	91.03	84	3.81	90.50
66	79	2.75	90.54	92	3.70	90.02	88	3.99	90.02
67	172	4.28	85.73	93	3.75	91.11	82	3.72	90.27
68	217	5.20	90.00	98	4.45	87.57	85	3.83	90.20
69	220	5.30	87.49	96	4.36	91.18	85	3.88	91.04
70	133	3.84	91.47	98	4.23	91.05	84	3.83	89.01
71	71	4.72	86.27	95	4.02	91.01	83	3.77	88.33
72	192	4.64	91.85	90	3.61	91.01	88	4.03	91.11
73	107	3.23	92.19	89	3.56	91.06	86	3.91	91.24
74	217	5.24	88.66	92	3.73	91.00	86	3.91	91.25
75	80	2.83	89.55	92	3.97	91.01	85	3.86	92.32
76	263	6.14	84.13	93	3.92	91.05	81	3.63	90.31
77	65	2.52	91.49	93	3.89	90.09	82	3.70	91.31
78	-	-	-	94	3.88	90.24	85	3.86	91.07
79	57	2.48	89.72	92	3.72	85.34	82	3.69	91.65
80	65	2.50	87.04	92	3.72	87.57	88	4.00	91.05
81	77	2.81	92.55	-	-	-	86	3.89	89.33
82	202	5.58	82.93	94	3.95	90.15	85	3.56	91.29
83	121	3.48	90.47	99	4.17	90.45	84	3.50	87.57
84	99	3.16	89.05	92	3.75	90.12	88	3.83	91.07
85	236	5.41	91.32	100	4.17	90.15	82	3.50	91.22
86	225	5.16	89.90	92	3.72	89.66	78	3.22	91.04
87	169	4.50	90.99	89	3.59	91.03	84	3.58	91.07
88	70	2.59	88.70	96	4.00	91.05	84	3.53	91.12
89	131	3.70	88.79	91	3.74	91.05	87	3.63	91.04
90	86	2.97	88.88	95	3.95	89.66	84	3.50	91.04
91	201	5.55	87.94	95	3.92	91.22	86	3.69	89.68
92	144	3.97	93.99	92	3.81	90.13	87	3.66	91.21
93	276	6.14	89.80	95	3.94	91.16	84	3.53	91.10
94	-	-	-	90	3.69	91.13	86	3.61	91.05
95	105	3.22	90.57	93	3.94	91.11	85	3.58	91.23
96	164	4.42	91.18	92	3.78	91.04	88	3.75	90.05
97	182	4.63	89.00	93	3.84	91.01	90	4.84	90.02
98	108	3.24	88.99	94	3.92	91.02	88	3.70	91.20

99	95	3.17	89.47	96	3.94	91.44	81	3.36	91.03
100	72	2.70	88.77	99	4.11	91.01	88	3.72	91.19
Mean	**166**	**4.50**	**88.74**	**93**	**4.02**	**90.43**	**85**	**3.76**	**90.66**
SD	**87**	**1.79**	**15.48**	**3.83**	**0.64**	**12.77**	**3.01**	**0.50**	**9.11**
Succ.	**97**			**98**			**99**		
Fail	**3**			**2**			**1**		

Table II.6: Detail version of the Broyden-Fletcher-Goldfarb-Shanno performance for Glass classification problem

	trainbfg			BFGS			BFGS-AG		
Trials	epoch	CPU time	accuracy	epoch	CPU time	accuracy	epoch	CPU time	accuracy
1	66	6.52	91.76	16	1.13	90.52	14	0.95	90.89
2	109	5.86	92.14	21	1.12	96.48	12	0.88	93.38
3	89	4.80	92.22	16	1.06	90.82	12	0.86	93.63
4	208	8.94	93.18	34	2.22	93.86	23	1.92	92.56
5	167	6.97	94.26	45	3.77	93.58	15	1.14	91.38
6	-	-	-	33	2.22	92.19	12	0.92	90.09
7	116	5.95	93.34	22	2.22	92.06	19	1.36	92.60
8	167	7.38	93.53	-	-	-	21	1.59	92.34
9	147	6.77	93.58	34	3.11	90.71	12	0.91	93.70
10	97	4.98	93.40	20	1.41	90.81	16	1.14	94.08
11	95	3.77	93.52	16	1.08	92.25	11	0.56	93.76
12	96	3.58	93.41	34	3.22	93.46	12	0.59	93.85
13	74	3.00	94.00	34	3.77	92.54	19	0.98	93.41
14	88	3.09	92.05	32	2.22	92.99	14	0.72	93.06
15	111	3.61	93.15	16	1.08	90.41	19	1.13	93.96
16	73	2.83	93.43	20	1.34	95.99	22	1.20	93.90
17	131	4.05	92.98	16	1.08	90.46	-	-	-
18	113	3.66	91.58	17	1.17	93.53	15	0.80	93.49
19	74	2.84	92.83	23	1.66	92.21	13	0.75	92.60
20	68	2.75	93.69	18	1.22	94.85	12	0.64	90.83
21	131	3.91	94.31	32	3.22	93.33	12	0.64	93.57
22	94	5.11	92.75	16	1.06	93.87	27	1.41	93.62
23	137	4.31	94.07	17	1.11	92.70	41	2.36	93.74
24	103	3.45	92.30	-	-	-	15	0.80	92.70
25	222	5.77	93.70	16	1.06	93.82	12	0.64	96.95
26	89	3.13	93.32	36	4.22	92.56	12	0.70	93.46
27	192	5.17	93.02	16	1.08	92.23	12	0.78	92.83
28	142	4.17	93.88	34	3.32	96.85	15	0.88	90.89
29	180	4.95	92.83	17	1.13	90.44	20	1.17	93.39
30	102	3.30	92.95	15	1.03	90.14	18	0.91	92.08
31	102	3.33	92.42	16	1.09	93.75	12	0.59	93.18
32	157	4.42	92.80	35	4.22	93.69	15	0.75	93.65
33	157	4.39	93.95	36	4.22	90.26	19	0.98	92.62
34	162	4.41	94.73	16	1.08	94.07	15	0.81	90.74
35	75	2.84	93.31	16	1.06	93.30	15	0.94	94.70
36	-	-	-	16	1.11	93.91	14	0.88	94.55
37	130	3.95	93.33	17	1.14	92.50	18	1.16	94.00
38	65	2.70	92.55	17	1.13	91.82	21	1.55	92.82
39	107	3.53	92.54	16	1.08	93.75	20	1.39	92.91
40	78	2.97	94.26	16	1.08	93.80	13	0.83	92.18
41	88	3.36	91.95	34	3.23	93.22	14	0.81	92.89
42	53	2.45	92.95	45	3.22	92.89	12	0.75	90.79
43	227	6.34	93.54	54	5.23	93.51	16	1.16	93.79
44	191	5.38	93.16	33	2.22	93.14	11	0.67	90.99
45	132	4.22	92.24	34	3.23	94.09	12	0.69	93.54
46	166	4.69	94.30	17	1.27	90.64	14	0.81	90.89

47	79	2.94	93.62	44	3.22	90.31	19	1.06	93.14
48	114	3.67	93.56	34	3.22	90.69	22	1.33	94.79
49	99	3.38	92.83	16	1.14	90.13	13	0.75	93.41
50	85	3.05	93.20	23	2.22	90.60	12	0.78	93.90
51	94	3.13	92.91	43	3.22	93.45	19	1.28	90.23
52	112	3.70	92.99	23	2.22	93.09	11	0.77	93.64
53	212	6.45	93.24	34	3.22	93.06	-	-	-
54	101	3.42	91.91	16	1.14	93.33	17	1.22	91.76
55	107	4.25	93.22	21	2.14	90.11	18	1.14	90.26
56	-	-	-	16	1.45	94.25	11	0.59	94.50
57	-	-	-	54	4.22	93.21	11	0.55	96.80
58	102	3.56	93.81	55	4.77	96.55	13	0.69	94.77
59	166	4.52	92.64	34	4.22	90.53	18	1.13	94.05
60	253	7.06	93.81	54	4.77	90.69	12	0.66	94.89
61	142	4.20	93.97	33	2.11	96.00	16	0.92	94.82
62	98	3.38	93.38	45	3.22	90.79	19	0.99	93.18
63	109	3.47	93.66	16	1.09	90.06	18	0.97	95.58
64	59	3.20	92.61	45	5.33	90.48	22	1.39	94.71
65	100	3.36	92.87	-	-	-	17	0.98	94.92
66	94	3.31	93.23	18	1.27	93.91	11	0.58	94.43
67	134	4.17	93.77	33	3.12	93.67	17	0.84	93.21
68	89	3.06	93.82	15	1.14	93.95	13	0.64	90.54
69	134	4.11	94.32	45	3.22	94.70	15	0.75	93.32
70	92	3.34	93.34	34	2.12	93.67	18	0.91	92.89
71	99	3.24	94.06	18	1.22	94.23	14	0.70	90.10
72	122	3.77	93.66	45	3.88	94.84	17	0.84	93.78
73	69	2.70	93.24	17	1.27	93.44	12	0.59	93.63
74	105	3.39	92.76	16	1.48	96.36	14	0.70	94.52
75	102	3.38	92.94	56	6.33	90.69	-	-	-
76	62	2.56	92.26	16	1.59	90.86	14	1.09	93.23
77	154	4.55	93.12	18	2.45	90.29	12	1.06	90.59
78	115	3.50	94.01	34	3.22	90.03	13	0.92	90.57
79	88	3.16	94.45	23	2.22	93.39	12	0.74	93.01
80	101	3.31	93.71	18	1.52	92.13	16	0.89	94.78
81	112	3.61	92.67	54	5.33	91.30	12	0.64	93.30
82	-	-	-	18	1.16	92.71	13	0.70	94.94
83	65	2.53	92.11	45	5.33	92.54	12	0.69	93.21
84	126	3.77	93.59	17	1.05	93.50	14	0.81	93.46
85	131	3.69	93.51	18	1.06	92.41	16	0.91	93.11
86	102	3.42	94.10	24	2.11	92.34	15	0.86	96.94
87	121	3.72	93.95	-	-	-	13	0.89	94.57
88	89	3.00	93.02	35	3.23	93.24	19	1.42	92.89
89	95	3.11	93.26	45	5.33	93.64	-	-	-
90	151	4.28	94.11	16	1.05	93.06	11	0.73	94.41
91	122	3.73	92.48	20	1.42	92.68	11	0.69	94.84
92	65	2.52	92.92	24	2.01	92.94	16	1.03	90.31
93	97	3.17	92.79	25	2.25	93.01	12	0.75	96.61
94	70	2.61	93.31	18	1.90	96.52	18	1.17	93.64
95	139	3.86	92.73	28	2.97	90.32	20	1.30	92.76
96	123	3.63	93.51	28	2.90	90.75	-	-	-
97	103	3.41	92.55	31	3.01	90.43	17	1.09	92.62
98	182	5.05	93.35	22	1.17	90.55	13	0.83	94.44