DOKUZ EYLÜL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

COMPARISON OF TRADITIONAL AND EVOLUTIONARY NEURAL NETWORKS FOR CLASSIFICATION

by

Asil ALKAYA

January, 2010

İZMİR

COMPARISON OF TRADITIONAL AND EVOLUTIONARY NEURAL NETWORKS FOR CLASSIFICATION

A Thesis Submitted to the

Graduate School of Natural and Applied Sciences of Dokuz Eylül University

In Partial Fulfillment of the Requirements for the Degree of Doctor of

Philosophy in Industrial Engineering, Industrial Engineering Program

by

Asil ALKAYA

January, 2010

İZMİR

Ph.D. THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"COMPARISON OF TRADITIONAL AND EVOLUTIONARY NEURAL NETWORKS FOR CLASSIFICATION"** completed by **ASİL ALKAYA** under supervision of **PROF.DR. G. MİRAÇ BAYHAN** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Doctor of Philosophy.

Prof.Dr. G. Miraç BAYHAN

Supervisor

Prof. Dr. Nihat BADEM	Asst. Prof. Dr. Yavuz ŞENOL
Thesis Committee Member	Thesis Committee Member
Prof.Dr. İ. Kuban ALTINEL	Asst. Prof.Dr. Özcan KILINÇCI
Examining Committee Member	Examining Committee Member

Prof.Dr. Cahit HELVACI
Director
Graduate School of Natural and Applied Sciences

ACKNOWLEDGMENTS

Many people have contributed to the completion of this dissertation in many ways: explicitly, unwittingly, and serendipitously. I hope to name them herein. As my memory is encased in a biological substrate, it is fallible. If your name does not appear here, forgive me: I am thankful to you as well.

First of all, I would like to thank:

- My advisor, Prof. Dr. G.Miraç BAYHAN, for her endless support and encouragement.
- My committee members. Prof. Dr. Nihat BADEM, Asst. Prof. Dr. Yavuz ŞENOL, Asst.Prof.Dr. Özcan KILINÇCI and Prof.Dr. Kuban ALTINEL.
- My parents, Saime and Gülman ALKAYA, for their own unique form of support, who kept asking, “haven’t you finished yet?”

Tg programmers who have most supported and encouraged me:

- Julien WINTZ is a software engineer working on scientific computing, parallel computing, and 3D visualization at Institut National De Recherche En Informatique Et En Automatique.
- Ph.D. Pierre KRAEMER, assistant professor in computer science at Strasbourg University.

Asil ALKAYA

COMPARISON OF TRADITIONAL AND EVOLUTIONARY NEURAL NETWORKS FOR CLASSIFICATION

ABSTRACT

Classification refers to the assignment of a finite set of alternatives into predefined groups. The limitation of the statistical models applied to the classification is that they work well only when the underlying assumptions are satisfied. Neural networks are universal functional approximators so that they can adjust themselves to the data without any explicit specification of functional or distributional form for the underlying model. Because of the difficulty of designing the artificial neural networks; evolutionary algorithms are embedded into artificial neural networks that are robust and probabilistic search strategies excel in large and complex problem spaces. In this thesis, two datasets are classified using evolutionary neural networks. In order to generate an optimal evolutionary neural network of each given dataset, the parameters are optimized including; number of neurons in the hidden layer, stepsize and momentum which makes the classification with high accuracy. Research involving the application of evolutionary algorithms to neural networks for benchmarking the classification performance of training and testing of the datasets with cross validation has been carried out. Performance is benchmarked by mean squared error, normalized mean squared error, mean absolute error, correlation coefficient and true classification rate that is referred to each attribute which is subject to be classified and evaluated with backpropagation and evolutionary neural networks whose parameters are selected using evolutionary algorithms. As argued in the literature; evolutionary neural networks having optimized parameters, get better performance values in classification than the artificial neural networks using the backpropagation algorithm with the same architecture.

Keywords : Evolutionary algorithms, artificial neural networks, classification

GELENEKSEL VE EVRİMSEL YAPAY SİNİR AĞLARININ SINIFLANDIRMA İÇİN KARŞILAŞTIRILMASI

ÖZ

Sınıflandırma, önceden tanımlanmış gruplara, alternatiflerden oluşan sonlu bir dizinin ataması olarak adlandırılır. Sınıflandırmaya uygulanan istatistiksel modeller, kısıt olarak, söz konusu varsayımların sadece geçerliliğini koruduğu sürece iyi şekilde işler. Yapay sinir ağları, evrensel fonksiyon tahminleyiciler olarak, ele alınan model için fonksiyonel ya da dağılımsal olarak belirgin nitelikte bir biçim olmaksızın veriye kendilerini uyarlayabilir. Yapay sinir ağlarının tasarımındaki güçlük nedeniyle, büyük ve karmaşık problem uzaylarında başarı gösteren, sağlam ve olasılığa dayalı arama stratejileri olan evrimsel algoritmalar, yapay sinir ağlarının içine yerleştirilmiştir. Bu tezde, evrimsel yapay sinir ağlarını kullanarak iki veri seti sınıflandırılmıştır. Her veri setinin optimal evrimsel yapay sinir ağını üretmek için, yüksek doğrulukta sınıflandırma yapmak amacıyla; gizli katmandaki nöron sayısı, adım büyüklüğü ve momentumu içeren parametreler optimize edilmiştir. Çapraz doğrulama ile veri kümelerinin öğrenme ve test alt kümelerinin sınıflandırma performanslarını kıyaslamak için, evrimsel algoritmaların yapay sinir ağlarına uygulanmasını içeren araştırma ortaya konmuştur. Performans, geriyayılım ve evrimsel algoritmalar kullanarak seçilen evrimsel yapay sinir ağları ile sınıflandırılmaya ve değerlendirilmeye konu olan her niteliğe karşılık gelen ortalama hata kare, normalleştirilmiş ortalama hata kare, ortalama mutlak hata, korelasyon katsayısı ve doğru sınıflandırma oranı ile kıyaslanmıştır. Literatürde de öne sürüldüğü gibi; optimize edilmiş parametrelere sahip evrimsel sinir ağlarının; sınıflandırma problemlerinde, geriyayılım algoritmasını kullanan aynı mimariye sahip yapay sinir ağlarına kıyasla daha iyi performans değerleri elde etmektedir.

Anahtar sözcükler : Evrimsel algoritmalar, yapay sinir ağları, sınıflandırma

CONTENTS

Page

PH.D. THESIS EXAMINATION RESULT FORM ii
ACKNOWLEDGEMENTS iii
ABSTRACT iv
ÖZ v

CHAPTER ONE – INTRODUCTION 1

1.1 Classification 1
1.2 Neural Networks 3
1.3 Evolutionary Algorithms 5
1.4 Evolutionary Artifical Neural Networks 6
1.5 Literature review for classification 6
1.5.1 Classification With Neural Networks 6
1.5.2 Classification With Evolutionary Neural Networks 7
1.6 Overview of Thesis 13

CHAPTER TWO – ARTIFICIAL NEURAL NETWORKS 15

2.1 The Neuron 15
2.2 Mechanics 15
2.3 Layer 16
2.4 Linear separability 18
2.5 Learning 19
2.5.1 Network Training 20
2.5.2 Momentum 21
2.5.3 Cross validation 23
2.5.4 Sensitivity Analysis 25
2.6 Artificial Neural Network Learning Algorithms 26

2.6.1 Multiple minima problem in neural networks 27
2.6.2 Backpropagation algorithm 29
2.6.2.1 Training Problems In Backpropagation 33
2.6.2.1.1 Initial Weights 34
2.6.2.1.2 Number of Hidden Units 35
2.6.2.1.3 Length of Training 36
2.6.2.1.4 Evaluation Strategies 37
2.6.2.2 Nonlinear Activation Functions 37
2.6.3 Conjugate gradient algorithm 38
2.6.3.1 Weight Update Equations 39
2.6.4 Levenberg–Marquardt algorithm 41

CHAPTER THREE – EVOLUTIONARY ALGORITHMS 45

3.1 Genetic Algorithms 45
3.2 Evolutionary Strategies 47
3.3 Genetic Programming 48
3.4. Evolutionary Programming 48
3.5. Genetic Operators 49
3.5.1 Crossover 49
3.5.1.1 Crossover Types 50
3.5.1.1.1 One Point Crossover 50
3.5.1.1.2 Two Point Crossover 50
3.5.1.1.3 Arithmetic Crossover 51
3.5.1.1.4 Heuristic Crossover 52
3.5.2 Mutation 52

CHAPTER FOUR – EVOLUTIONARY ARTIFICIAL NEURAL NETWORKS 54

4.1 Neural Networks For Genetic Based Classification 54
4.2 Motivation For Evolutionary Artificial Neural Networks 55

4.3. Types Of Evolutionary Artificial Neural Networks 56
4.3.1 Weight-Evolving Algorithms (WEAs) 56
4.3.2. Topology Evolving Algorithms (TEAs) 58
4.3.3. Hybrid Evolutionary Algorithms (HEAs) 59
4.4 Algortihms For Evolution 59
4.5 Performance Measures For EANN Based Classification 60
4.5.1 Correlation Coefficient 60
4.5.2 Confusion Matrix 61
4.5.3 Mean Square Error (MSE) 62
4.5.4 Mean Squared Error (NMSE) 63
4.5.5 Relative Percent Difference 63

CHAPTER FIVE – COMPARISON OF NEURAL NETWORKS WITH EVOLUTIONARY ALGORITHMS FOR CLASSIFICATION 64

5.1 General Information 64
5.2 Teaching Assistant Evaluation (tae) Dataset 67
5.2.1 The Comparison of Learning Algorithms Due To ANN Structure For tae Dataset 69
5.2.1.1 Momentum Learning 69
5.2.1.1.1 Testing of Training Subset Data For Momentum Learning Method 72
5.2.1.1.2 Testing of Cross Validation Subset Data For Momentum Learning Method 72
5.2.1.2 Conjugate-Gradient Learning Method 73
5.2.1.2.1 Testing Of Training Subset Data For Conjugate-Gradient Learning Method 75
5.2.1.2.2 Testing Of Cross Validation Subset Data For Conjugate-Gradient Learning Method 75
5.2.1.3 Levenberg-Marquardt Algorithm 76
5.2.1.3.1 Testing The Training Subset of Levenberg-Marquardt Algorithm 78

5.2.1.3.2 Testing of Cross Validation Subset Data For Levenberg-Marquardt Method..78
5.2.1.4 Accuracy Comparison of Testing The Training And Cross Validation Subsets of Learning Algorithms...................................79
5.2.2 The Optimum Momentum Rate For tae Dataset....................................80
5.2.2.1 Testing of Training Subset Data For Momentum Learning Method 82
5.2.2.2 Testing of Cross Validation Subset Data For Momentum Learning Method 83
5.2.3 The Optimum Hidden Layer Size 84
5.2.3.1 One Hidden Layered ANN 84
5.2.3.1.1 Testing The Training Subset of One Hidden Layer For Momentum Learning Algorithm 86
5.2.3.1.2 Testing The Cross Validation Subset of One Hidden Layer For Momentum Learning Algorithm 86
5.2.4 Optimum Number of Processing Elements of The Hidden Layer 87
5.3 Vehicle Silhouette (veh) Data Set 90
5.3.1 Statlog Vehicle Silhoutte (Veh) Database 90
5.3.2 Dataset Information 90
5.3.3 Dataset Description 90
5.3.4. Attribute Information 91
5.3.5 The Comparison of Learning Algorithms Due To ANN Structure 93
5.3.5.1 Momentum Learning 93
5.3.5.1.1 Testing The Training Subset of One Hidden Layer For Momentum Learning Algorithm 95
5.3.5.1.2 Testing The Cross Validation Subset of One Hidden Layer For Momentum Learning Algorithm 96
5.3.5.2 Conjugate-Gradient Learning Method 96
5.3.5.2.1 Testing of Training Subset Data For Conjugate-Gradient Learning Method 98
5.3.5.2.2 Testing of Cross Validation Subset Data For Conjugate-Gradient Learning Method 99

5.3.5.3. Levenberg-Marquardt Algorithm 99
5.3.5.3.1 Testing The Training Subset of Levenberg-Marquardt Algorithm 101
5.3.5.3.2 Testing of Cross Validation Subset Data For Levenberg-Marquardt Learning Algorithm 102
5.3.5.4 Comparison For Mean Accuracy Testing The Training And Cross Validation Subset Data of Learning Algorithms 102
5.3.6 The Optimum Momentum Rate 103
5.3.7 The Optimum Hidden Layer Size 106
5.3.7.1 Two hidden Layered ANN 106
5.3.8 Optimum Number of Processing Elements of The Hidden Layers 110
5.4 Parameter Optimization 112
5.4.1 Parameter Optimization For Teaching Assistant Evaluation (Tae) Dataset 116
5.4.2 Parameter Optimization For Vehicle Silhoutte (Veh) Dataset 123
5.5 Discussion 133

CHAPTER SIX – CONCLUSION AND FUTURE RESEARCH..................... 138

6.1 Contributions.. 139
6.2 Future Research.. 141

REFERENCES 143

APPENDIX 1 153
APPENDIX 2 156

CHAPTER ONE
INTRODUCTION

1.1 Classification

Decision making problems, according to their nature, the policy of the decision maker, and the overall objective of the decision, may require the choice of an alternative solution, the ranking of the alternatives from the best to the worst ones or the assignment of the considered alternatives into predefined homogeneous classes. This last type of decision problem is referred to as classification.

Classification problems are often encountered in a variety of fields including finance, marketing, environmental and energy management, human resources management, medicine, etc (Zopounidis & Doumpos, 2002).

The major practical interest of the classification problem has motivated researchers in developing an arsenal of methods for studying such problems, in order to develop mathematical models achieving the higher possible classification accuracy and predicting ability.

Classification refers to the assignment of a finite set of alternatives into predefined groups; as a general description. The task of classifying data is to decide class membership y' of an unknown data item x' based on a data set $D = (x_1, y_1)........(x_n, y_n)$ of data items x_i with known class memberships y_i. The x_i are usually m-dimensional vectors, the components of which are called input variables (by the machine learning community).

Traditional statistical classification procedures are built on the Bayesian decision theory. In these procedures, an underlying probability model must be assumed in order to calculate the posterior probability upon which the classification decision is made. One major limitation of the statistical models is that they work well only when the underlying assumptions are satisfied.

The effectiveness of these methods depends to a large extent on the various assumptions or conditions under which the models are developed. Users must have a good knowledge of both data properties and model capabilities before the models can be successfully applied.

In most problem domains, there is no functional relationship $y = f(x)$ between *y* and *x*. In this case, the relationship between x and y has to be described more generally by a probability distribution $P(x, y)$; one then assumes that the data set *D* contains independent samples from *P*. From statistical decision theory, it is well known that the optimal class membership decision is to choose the class label y that maximizes the posterior distribution $P(y|x)$.

For a general *M*-group classification problem in which each object has an associated attribute vector $\boldsymbol{x}$ of d dimensions. Let ω denote the membership variable that takes a value of w_j if an object is belong to group *j*. Define $P(w_j)$ as the prior probability of group j and $f(x|w_j)$ as the probability density function. According to the Bayes rule;

$$P(w_j \Big| x) = \frac{f(x|w_j)P(w_j)}{f(x)}$$

where $P(x|w_j)$ is the posterior probability of group *j* and $f(x)$ is the probability density function:

$$f(x) = \sum\nolimits_{j=1}^{M} f(x|w_j)P(w_j)$$

It is supposed that an object with a particular feature vector *x* is observed and a decision is to be made about its group membership. The probability of classification error is:

$$P(Error \mid x) = \sum_{i \neq j} P(w_i \mid x)$$

$$= 1 - P(w_j \mid x) \qquad \text{if } w_j \text{ decided.}$$

Hence if the purpose is to minimize the probability of total classification error (misclassification rate), the classification rule is:

Decide w_k for x if $P(w_k \mid x) = \max_{i=1,2,\ldots\ldots,M} P(w_i \mid x)$

There are two different approaches to data classification: the first considers only a binary distinction between the two classes, and assigns class labels 0 or 1 to an unknown data item. The second attempts to model $P(y \mid x)$; this yields not only a class label for a data item, but also a probability of class membership for multi-class problem domains (Dreiseitl & Ohno-Machado, 2002).

Table 1.1 Classification types

Classification Type	
Binary	**Multi-class**
Support vector machines	Logistic regression
	Decision trees
	k-nearest neighbors
	Artificial neural networks

1.2 Neural Networks

Neural networks have emerged as an important tool for classification. The recent vast research activities in neural classification have established that neural networks are a promising alternative to various conventional classification methods. The advantage of neural networks lies in the following theoretical aspects. First, neural networks are data driven self-adaptive methods in that they can adjust themselves to the data without any explicit specification of functional or distributional form for the underlying model. Second, they are universal functional approximators in that neural networks can approximate any function with arbitrary accuracy.

Neural networks are a non-symbolic approach to classification. Based on a loose paradigm of neurons in the brain, neural networks are able to pick out pertinent patterns in data, often when the data is corrupted, noisy, or uncertain. While their training processes can be slow, completed neural networks are generally quite fast in application. Their strengths include the ability to generalize large numbers of patterns into classes, and to learn from a presentation of example problems and solutions. One major obstacle to the design of neural networks is the selection of an ideal set of parameters for a particular problem.

Neural networks are hand-crafted by experts with years of experience. Two major drawbacks of this approach are a lack of experts, and a lack of a strict design methodology. The first problem is enough: there simply are not enough experts to attend to all the potential neural network projects the world has to offer. The second problem is somewhat more faint and difficult to analyze. No obedient algorithm exists to optimally determine the parameters for a particular neural network application. The science of designing neural systems at best is inaccurate as the result of this complexity. Firstly, the process is intuitionally driven. A system is needed to determine neural network designs more efficiently and effectively.

It is impossible to expect that any single neural network will be able to solve any problem regardless of complexity. To direct to a specific destination of this problem, research is being conducted into much complex systems. In these systems, several networks cooperate to solve a problem which would not be solvable by any single neural network architecture. While the power and flexibility of the resulting configuration has the potential to outperform simple neural networks, the combination of multiple networks increases the difficulty of managing the system.

Whereas before a designer had to manage only a single network, the problem becomes one of designing multiple networks while simultaneously enabling them to cooperate on the problem at hand. The work load and computation time rises exponentially with the size of the system.

Strictly speaking, a method is needed to free experts from the inaccurate run time of manually managed networks. One promising method of solving both problems is through the help of the use of evolutionary algorithms (EAs). This thesis presents a systematic approach to automating the design of neural networks for classification through the use of evolutionary algorithms.

1.3 Evolutionary Algorithms

Genetic algorithms were developed by John Holland at the University of Michigan. Holland set out to achieve two goals. First, to "abstract and explain the adaptive processes of natural systems", and second, to "design artificial systems mathematically that retains the important mechanisms of natural systems" (Goldberg, 1989). Holland showed how adaptive type of natural and biological systems can be applied to artificial systems.

Due to hardness in the process of creating and designing artificial neural networks, genetic algorithms have become a a point of concentration of study in the field. By the help of the genetic algorithms, it is possible to remove some of the trial and error partial design from the designer. On the other hand, the genetic algorithm is used to search a solution space for neural network parameters that are so much more effective.

Genetic algorithms have been used to select various features of neural networks. These include learning parameters, hidden units, topology, connections, and even to evolve the synaptic weights themselves achieved by the learning algorithm .

1.4 Evolutionary Artifical Neural Networks

Evolutionary artificial neural networks (EANNs) are the combination of artificial neural networks and evolutionary algorithms. This merge enabled these two methods to complement the disadvantages of the other methods. For example, a contribution by artificial neural networks was the flexibility of nonlinear function approximation, which cannot be easily implemented with prototype evolutionary algorithm. On the other hand, evolutionary algorithm has freed artificial neural networks from simple gradient descent approaches of optimization. But as a disadvantage, the inclusion of backpropagation training in the EANN have consequences of longer computation times, so alternatives to backpropagation should be tested in order to reduce time costs.

Indeed, traditional artificial neural networks based on backpropagation algorithms have some limitations. At first, the architecture of the artificial neural networks is fixed and a designer needs much knowledge to determine it. Also, error function of the learning algorithm must have a derivative.

Finally, it frequently gets stuck in local optima because it is based on gradient-based search without stochastic property.

1.5 Literature review

In this section, publications and approaches to classification with traditional and evolutionary neural networks in the literature are discussed.

1.5.1 Classification with Neural Networks

The theoretical relationship linking estimation of Bayesian posterior probabilities to minimizing squared error cost functions has long been known. The mapping function $F : x \rightarrow y$, which minimizes the expected squared error is shown as the conditional expectation $E[y|x]$ (Papoulis ,1991). Since in a classification problem the output y is a vector of binary values, it can be easily shown that

$E[y|x] = P(W|x)$. Since neural networks can approximate any function F with arbitrary accuracy (universal approximators), then neural network outputs are indeed good estimators of the posterior probabilities $P(W|x)$ (Hung, Hu, Patuwo & Shanker, 1996).

Bourlard & Wellekens (1989), Richard & Lippmann (1991), Shoemaker (1990), Wan (1990) and White (1989) have provided linkage between neural networks and posterior probabilities for squared error functions and for the cross-entropy error function.

Richard & Lippmann (1991), Foody (1995), Blamire (1996), Pal & Mather (2003) showed that neural networks minimizing squared-error and cross-entropy cost functions are capable of estimating posterior probabilities. The fact that neural networks can estimate posterior probabilities makes them powerful classification tools (Berardi et al., 2004).

Duin (1996) and Flexer (1996) compared neural networks and the other classifiers in the literature. Addition to comparison, the research topics taken into consideration is shown in Table 1.2.

1.5.2 Classification with Evolutionary Neural Networks

The use of EAs to design ANNs that are then trained using some parameter learning algorithm allows compact and effective structures to be built. However, imprecision in the evaluation of the candidate solutions must be taken into account due to possible sub-optimal convergence of the weight training procedure. Furthermore, the training of the ANN weights may be excessively slow for adequate exploration of the search space.

It is preferable to simultaneously optimise both the ANN architecture and the parameters. This can be done either by alternating steps of evolutionary structure optimisation with steps of standard (backpropagation) training of the parameters or by evolving at the same time both the connectivity and the weights.

Table 1.2 Publications related to the research topics of the neural network classification

Research Topic	Author(s)	Publication Year
Network Training	▪ Barnard	1992
	▪ Battiti R	1992
	▪ Hagan and Henhaj	1994
	▪ Nedeljkovic	1993
	▪ Roy, Kim, and Mukhopadhyay,	1993
Model design and selection	▪ Fujitao	1998
	▪ Hintz-Madsen, Hansen, Larsen and Pedersen	1998
	▪ Moody J. and Utans J.	1995
	▪ Murata N., Yoshizawa S. and Amari S.	1993
	▪ Murata N., Yoshizawa S., and Amari S.,	1994
	▪ Wang , Massimo ,Tham, Morris	1994
	▪ Yuan J.-L. and Fine T. L.	1998
Sample size issues	▪ Fukunaga K. and Hayes R.R.,	1989
	▪ Raudys S.,	1998
	▪ Raudys S. J. and Jain A. K.,	1991
Bayesian Analysis	▪ Lewicki M. S.	1994
	▪ D. C. MacKay	1992
	▪ P. Muller and D. R. Insua	1998

Stepniewski & Keane (1996) report applications of evolutionary algorithms to the design of ANN architectures coupled to customary weight training algorithms, a typical example being the evolution of multilayer perceptron (MLP) topologies with backpropagation training of the ANN parameters. Fitness evaluation is generally expressed as a multi-optimisation criterion that takes into account different requirements such as ANN performance, size and learning speed.

To address the design problem of the artificial neural networks (ANN), a population-based evolutionary approach called SEPA is developed (Structure Evolution and Parameter Adaptation) which replaces BPs (backpropagation) gradient descent heuristic by using a purely stochastic implementation (Palmes & Usui, 2005). It is carried out through the use of uniform crossover and Gaussian perturbation to effect mutations which are responsible for the changes in weights, and addition or deletion of nodes in a three-layered feed-forward ANN.

The simultaneous evolution of network structure, parameters, and weights by Gaussian mutation and uniform crossover coupled with rank selection, early stopping, elitism, and direct encoding are effective in searching for the appropriate network structure and weights with good generalization performance (Palmes & Usui, 2005).

Publications related to the evolutionary neural networks are listed in Table 1.3, Table 1.4, Table 1.5 and Table 1.6. Evolution is made on training or number of nodes in the hidden layer (topology) or both of them. Crossover and mutation are genetic operators that are basically used as the evolutionary algorithm.

Table 1.3 Publications related to the evolutionary neural networks

Evolution Type	**Evolutionary Algortihm**	**Learning Algorithm**	**Author(s)**	**Encoding type**	**Publication Year**
(Weight) Training	Genetic algorithm (crossover and mutation)	Back-propagation	Yao and Liu	Direct (binary)	1997
Weight and topology	Genetic algorithm (crossover and mutation)	Back-propagation	Moriarty and Miikkulainen	Direct (binary)	1997
(Weight) Training	Genetic algorithm (crossover and mutation)	Back-propagation	Garcia Pedrajas, Hervas-Martinez andMunoz-Perez	Direct (binary)	2003
Weight and topology	Genetic algorithm (crossover and mutation)	Back-propagation	Smalz and Conrad	Direct (binary)	1994

Table 1.4 Publications related to the evolutionary neural networks (continued from Table 1.3)

Evolution Type	Evolutionary Algortihm	Learning Algorithm	Author(s)	Encoding type	Publication Year
Weight (Training)	Genetic algorithm (crossover and mutation)	Back-propagation	Montana and Davis	Direct (binary)	1989
Weight (Training)	Genetic algorithm (crossover and mutation)	Back-propagation	Whitley and Hanson	Direct (binary)	1989
(Weight) Training	Genetic algorithm (crossover and mutation)	Back-propagation	Fogel et al.	Direct (binary)	1990
(Weight) Training	Genetic algorithm (crossover and mutation)	Back-propagation	Menczer and Parisi	Direct (binary)	1992
(Weight) Training	Genetic algorithm (crossover and mutation)	Back-propagation	Srinivas and Patnaik	Direct (binary)	1991
(Weight) Training	Genetic algorithm (crossover and mutation)	Back-propagation	Whitehead and Choate	Direct (binary)	1996
(Weight) Training	Genetic algorithm (crossover and mutation)	Back-propagation	Haussler et al.	Direct (binary)	1995
(Weight) Training	Genetic algorithm (crossover and mutation)	Back-propagation	Seiffert	Direct (binary)	2001
(Weight) Training	Genetic algorithm (crossover and mutation)	Back-propagation	Skinner and Broughton	Direct (binary)	1995
Weight and topology	Genetic algorithm (crossover and mutation)	Back-propagation	Angeline et al.,	Direct (binary)	1994
(Weight) Training	Genetic algorithm (crossover and mutation)	Back-propagation	Harp et al., 1990	Indirect	1990

Table 1.5 Publications related to the evolutionary neural networks (continued from Table 1.4)

Evolution Type	**Evolutionary Algortihm**	**Learning Algorithm**	**Author(s)**	**Encoding type**	**Publication Year**
(Weight) Training	Genetic algorithm (crossover and mutation)	Back-propagation	Kitano	Indirect	1990
Weight and topology	Genetic algorithm (crossover and mutation)	Back-propagation	Castillo et al	Indirect	2000
(Weight) Training	Genetic algorithm (crossover and mutation)	Back-propagation	Cangelosi and Elman	Direct (binary)	1995
Weight and topology	Genetic algorithm (crossover and mutation)	Back-propagation	Yao and Liu	Direct (binary)	1997 b
Weight and topology	Genetic algorithm (crossover and mutation)	Back-propagation	Odri, Petrovacki, and Krstonosic	Direct (binary)	1993
Weight and topology	Genetic algorithm (crossover and mutation)	Back-propagation	Hüsken and Igel	Direct (binary)	2002
(Weight) Training	Genetic algorithm (crossover and mutation)	Back-propagation	Caudell and Dolan	Direct (binary)	1989
(Weight) Training	Genetic algorithm (crossover and mutation)	Back-propagation	Branke	Direct (binary)	1995
(Weight) Training	Genetic algorithm (crossover and mutation)	Back-propagation	Cant-Paz and Kamath	Direct (binary)	2005
Weight and topology	Genetic algorithm (crossover and mutation)	Back-propagation	Miller, Todd and Hegde	Direct (binary)	1989
Weight and topology	Genetic algorithm (crossover and mutation)	Back-propagation	Seidlecki and Skalansky	Direct (binary)	1989

Table 1.6 Publications related to the evolutionary neural networks (continued from Table 1.5)

Evolution Type	**Evolutionary Algortihm**	**Learning Algorithm**	**Author(s)**	**Encoding type**	**Publicatio n Year**
Weight and topology	Genetic algorithm (crossover and mutation)	Back-propagation	Yang and Honavar	Direct (binary)	1998
Weight and topology	Genetic algorithm (crossover and mutation)	Back-propagation (no hidden layer)	Pao and Philips	Direct (binary)	1995
Weight and topology	Genetic algorithm (crossover and mutation)	Back-propagation (no hidden layer)	Pao and Takefuji	Direct (binary)	1992
Weight and topology	Genetic algorithm (crossover and mutation)	Back-propagation	Maniezzo	Direct (binary)	1994

By examining the literature from traditional neural networks to evolutionary neural networks, the interaction of classification with artificial neural networks started in 1989. In the meantime, within this year, genetic algorithms are embedded into artificial neural networks. Up to 1998, main issues of the artificial neural networks such as; training, sample size, design and posterior probabilities discussed in order to classify the datasets more accurate. In consequence of inflexibility of traditional neural networks to classifications, the researches and publications on this topic began to decline.

Genetic algorithms are introduced into artificial neural networks at the beginning of 1990s. Crossover and mutation operators are used for evolution. The selection of chromosome representation is important for the computation time and effort. Direct and indirect encoding used starting from the year 1989. Because the indirect encoding requires real representation, it’s not reasonable for large and complex data domains. Binary representation is used up to now as direct encoding that is more feasible for evolution. Both weight and topology evolution have been taken into consideration for better performance on true classification rate.

By 2005, alternatives as listed in Table 1.7, are applied to the algorithms. Backpropagation is omitted in order to give weight to evolutionary algorithm. Evolutionary programming is established and decision rule is embeded into the learning algorithm.

Table 1.7 Publications related to the classification with evolutionary neural networks

Evolution Type	**Evolutionary Algortihm**	**Learning Algorithm**	**Author(s)**	**Encoding type**	**Publication Year**
Weight and topology	Genetic algorithm (crossover and mutation)	SEPA (no back-propagation)	Palmes and Usui	Direct (binary)	2005
Weight and topology	Genetic algorithm (crossover and mutation)	Back-propagation	Rocha,Cortez and Neves	Direct (binary)	2007
Weight and topology	Evolutionary Programming (no crossover)	Back-propagation and decision rule	Martinez-Estudillo, Hervas-Martinez, Gutierrez and Martinez-Estudillo	Direct (binary)	2008
Weight and topology	Genetic algorithm (crossover and mutation)	Back-propagation	Ang Tan and Al-Mamun	Direct (binary)	2008
Weight and topology	Genetic algorithm (crossover and mutation)	Back-propagation	Castellani and Rowlands	Direct (binary)	2009

1.6 Overview of Thesis

The thesis consists of six parts. As the classification is explained in detail in Chapter I with a review, the other main subjects concerning evolutionary algorithms, artificial neural networks and linkage between the two subjects expressed briefly. In chapter II, the main components of an artificial neural network are introduced. The importance of linear separability and learning algorithms are discussed in a detailed manner. Neural network training needs some important arguments such as momentum and cross validation to get success. During learning process, there is a possibility to tackle up with a local minima. In order to cope with this problem, backpropagation algortihm is implemented as a supervised learning to the feedforward neural network.

In Chapter III, an overview of evolutionary algorithms including their paradigms, and a discussion of previous applications of evolutionary algorithms to neural networks has been presented.. The types of crossover and mutation operators are taken into consideration when designing an evolutionary algorithm based artificial neural network. Chapter IV defines the process of building an evolutionary artificial neural network. The evolution is implemented through different parts of the neural network mechanism, so each type of evolving neural networks are examined with a related litetature review. Chapter V presents the structure of the system developed for classification via evolutionary neural networks and also the two datasets that are structurally opposite due to attribute types performed and results obtained are discussed.

Finally, conclusions are drawn in Chapter VI, and directions for future work suggested.

CHAPTER TWO
ARTIFICIAL NEURAL NETWORKS

At the core of neural computation, the concepts of distributed, adaptive and nonlinear computing exist. Neural networks perform computation in a very different way than conventional computers, where a single central processing unit sequentially dictates every piece of the action. Neural networks are built from a large number of very simple processing elements that individually deal with pieces of a big problem.

2.1 The Neuron

A neuron is a computational unit which takes a vector of input values and produces an output value. Inputs can be received from other neurons or directly as input. A single output value is generated, which is either sent to each of the neurons in the next layer or becomes part of the final output of the network.

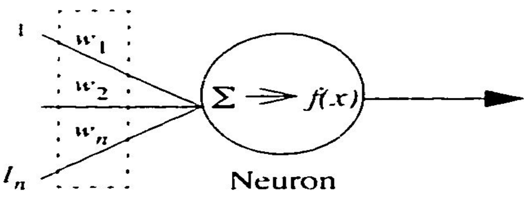

Figure 2.1 A simple neuron

A processing element (PE) simply multiplies an input by a set of weights, and nonlinearly transforms the result into an output value. The principles of computation at the PE level are deceptively simple. The power of neural computation comes from the massive interconnection among the PEs, which share the load of the overall processing task, and from the adaptive nature of the parameters (weights) that interconnect the PEs.

2.2 Mechanics

Neural networks are hand-crafted by experts with years of experience. Two major drawbacks of this approach are a lack of experts, and a lack of a strict design methodology. The first problem is enough: there simply are not enough experts to attend to all the potential neural network projects the world has to offer. The second problem is somewhat more faint and difficult to analyze. No obedient algorithm exists to optimally determine the parameters for a particular neural network application. The science of designing neural systems at best is inaccurate as the result of this complexity. The process is intuitionally driven. A system is needed to determine neural network designs more efficiently and effectively.

It is impossible to expect that any single neural network will be able to solve any problem regardless of complexity. To direct to a specific destination of this problem, research is being conducted into much complex systems. In these systems, several networks cooperate to solve a problem which would not be solvable by any single neural network architecture. While the power and flexibility of the resulting configuration has the potential to outperform simple neural networks, the combination of multiple networks increases the difficulty of managing the system.

Whereas before a designer had to manage only a single network, the problem becomes one of designing multiple networks while simultaneously enabling them to cooperate on the problem at hand. The work load and computation time rises exponentially with the size of the system.

Strictly speaking, a method is needed to free experts from the inaccurate run time of manually managed networks. One promising method of solving both problems is through the help of the use of evolutionary algorithms (EAs). This thesis presents a systematic approach to automating the design of neural networks for classification through the use of evolutionary algorithms

2.3 Layer

Normally, a neural network has several layers of PEs. What makes a layer an effective computational element is that each neuron has different synaptic weights which, when multiplied with the inputs, give each neuron a different value to which it applies its activation function. All the neurons in a layer have the same activation function. It is also possible, however, for different neurons in a layer to have different activation functions.

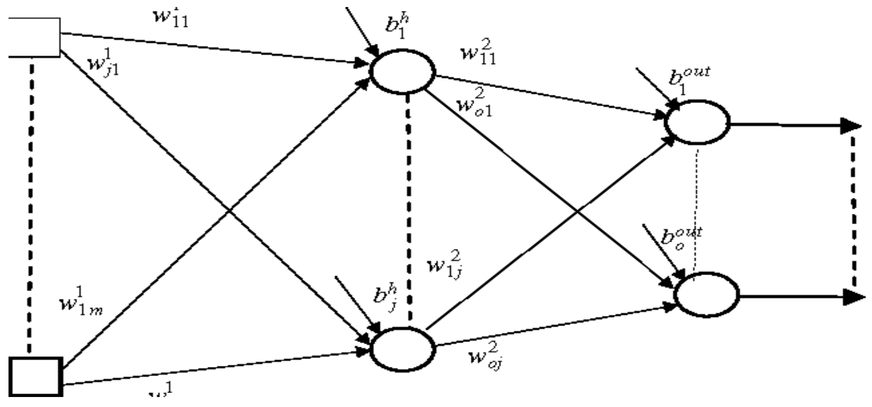

Figure 2.2 A Layer of Neurons

The diagram below illustrates a simple multilayer perceptron. The circles are the PEs arranged in layers. The left column is the input layer, the middle column is the hidden layer, and the right column is the output layer.

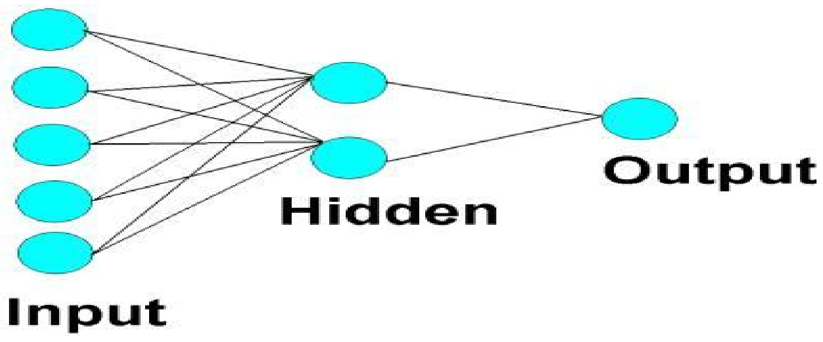

Figure 2.3 The simple multilayer perceptron

By adapting its weights, the neural network works towards an optimal solution based on a measurement of its performance. For supervised learning, the performance is explicitly measured column is the output layer. The lines represent weighted connections between processing elements in terms of a desired signal and an error criterion. For the unsupervised case, the performance is implicitly measured in terms of a learning law and topology constraints.

2.4 Linear Separability

By the comparison of the topology of two-layer and multilayer networks; two-layer networks are, with regard to fundamentals although not concerning details, have linear entities. By their nature, they can only classify data that is linearly separable.

A set of data is considered that is divisible into two classes. The data can be graphed in two dimensions and the two classes separated by a straight. For multidimensional data of *n* dimensions, the data will be separable with an *n*-dimensional separation. That is, data in three dimensions will be separable with a plane, and higher dimensions will be separable with an appropriate hyperplane (Wasserman, 1993).

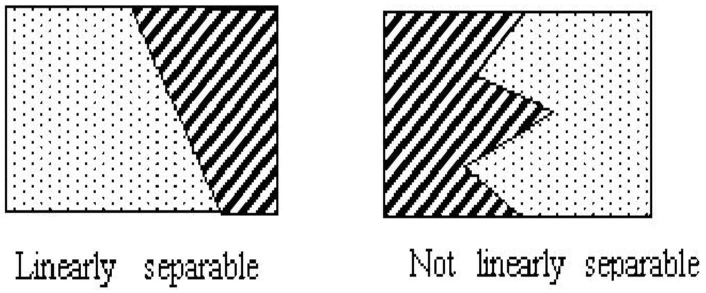

Figure 2.4 Separability

Some data, however, are not separable in this manner. The use of additional data in this manner is not always feasible, as analysis of the dataset to discover such data may be a non-trivial task.

In that case, a simple two-layer network could be used with an extra input factor Because of the lack of linear separability, a third dimension is needed that would create separable data can be understood of. A multilayer network can solve this problem which will not require the use of additional input factors.

The purpose, therefore, of multilayer networks is to solve problems in which the data is not linearly separable. If the data can be made separable by the addition of further input factors, this may be desirable as the resulting neural network would be simpler. However, as this is not always possible, multilayer networks are required.

Multilayer perceptrons (MLPs) overcome the linearity limitations associated with the perceptron. An MLP with one hidden layer is able to create a bump on the decision surface in the pattern space, a feature which is impossible with a single layer perceptron.

In general, adding enough nodes in hidden layers will allow the network to approximate any continuous function, but adding too many nodes increase the computational requirements of the network. It can also lead to overfitting to the training data, as the redundant hidden nodes tend to cause the network to memorize the training dataset rather than to reflect its general feature properties .

2.5 Learning

The network requires input data and a desired response to each input. The more data presented to the network, the better its performance will be. Neural networks take this input-output data, apply a learning rule and extract information from the data. Unlike other technologies that try to model the problem, artificial neural networks (ANNs) learn from the input data and the error. The network tries to adjust the weights to minimize the error. Therefore, the weights embody all of the information extracted during learning.

Essential to this learning process is the repeated presentation of the input-output patterns. If the weights change too fast, the conditions previously learned will be rapidly forgotten. If the weights change too slowly, it will take a long time to learn complicated input-output relations. The rate of learning is problem dependent and must be judiciously chosen. Each PE in the ANN will simply produce a nonlinear weighted sum of inputs. A good network output (a response with small error) is the right combinations of each individual PE response. Learning seeks to find this combination. In so doing, the network is discovering patterns in the input data that can solve the problem.

It is interesting that these basic principles are very similar to the ones used by biological intelligence. Information is gained and structured from experience, without explicit formulation. This is one of the exciting aspects of neural computation. These are probably the same principles utilized by evolution to construct intelligent beings. Like biological systems, ANNs can solve difficult problems that are not mathematically formulated. The systematic application of the learning rule guides the system to find the best possible solution.

2.5.1 Network Training

After taking care of the data collection and organization of the training sets, the network's topology must be selected. An understanding of the topology as a whole is needed before the number of hidden layers and the number of PEs in each layer can be estimated. This thesis will focus on multilayer perceptrons (MLPs) because they are the most common.

Hornick (1991) proved that a single hidden layer provides the network with the capability of approximating any measurable function from one finite dimensional space to another to any desired degree of accuracy. Indeed, ANNs having a single hidden layer have proven to be an important class of network for practical applications since they can approximate arbitrarily well any functional continuous mapping from one finite-dimensional space to another, provided the number of hidden units is sufficiently large (Bishop, 1995).

A multilayer perceptron with two hidden layers is a universal mapper (Hassoun 1995). Sontag (1992) showed that two hidden layers are required for approximating certain classes of discontinuous functions. A universal mapper means that if the number of PEs in each layer and the training time is not constrained, then it can be proved that the network has the power of solving any problem. This is a very important result but it is only an existence proof, so it does not say how such networks can be designed. The problem is to find out what is the right combination of PEs and layers to solve the problem with acceptable training time and performance.

In fact, unless the data is not linearly separable, it can be started without any hidden layers. The reason is that networks train progressively slower when layers are added. This error is propagated back through the network to train the weights. It is attenuated at each layer due to the nonlinearities.

So if a topology with many layers is chosen, the error to train the first layer's weights will be very small. Hence training times can become excruciatingly slow. As training times grow exponentially with the number of dimension of the network's inputs, all efforts should be made to make training easier.

This point has to be balanced with the processing purpose of the layers. Each layer increases the discriminant power of the network. For instance, a network without hidden layers is only able to solve classification problems where the classes can be separated by hyper-planes.

2.5.2 Momentum

The momentum term puts a weight on how much a synapse's previous weight adjustment should effect its current weight adjustment. The momentum term is multiplied by the previous result of the learning formula, that is, the previous weight adjustment.

An ad hoc departure from steepest descent is to add memory to the recursion through momentum term. The change in parameter vector depends not only on the current gradient but also on the most recent change in parameter vector:

$$\Delta_{k+1} = w_{k+1} - w_k = \beta\Delta_k - \alpha_k g_k \text{ for } k \geq 0$$

β is called the momentum constant. Wang & Principe (1999) recommend setting α to a value between 0.5 and 0.9. Using momentum with backpropagation both speeds up and stabilizes a neural network's convergence to a set of weight values. Momentum also helps a network to avoid local minima in the error function where gradient descent alone may cause the weight to become stuck. Momentum keeps the weights changing in the flat areas of the error curve and smooths out the weight's changes when the gradient bounces back and forth between the sides of a narrow dip in the error function curve. So a high frequency smoothing effect is gained through momentum term. The change in parameter vector depends not only on the current gradient g_{k-1} but also in an exponentially decaying manner $(0 \leq \beta < 1)$ on all gradients.

The benefit of a momentum term is two-fold, effectively dealing with both the major problems discussed above. First, the time it takes the network to train drops. This is due to the momentum term influencing the change in synaptic weights. Once the network is training in one direction toward the ideal point, the momentum term allows it to pick up speed.

Since momentum is applied to each iteration, the effect *snowballs*. The training actually picks up speed, making increasingly larger jumps toward the target value until it arrives at or passes over the target value. This leads to the second case, that of passing over the target value.

Momentum also solves the thrashing problem. When a network oversteps its target value, the next pass may recalculate the same amount of *correction* as the original error or some portion thereof to enable a cycle over several updates. With momentum, the adjustment in the new, opposite direction is added to a percentage of the direction in which the network was previously moving. In the case of an overstepped target, these two values will have opposite signs. While this may cause an overstep in the opposite direction, it must be less than the previous overstep due to the momentum term.

This process continues, with each overstep of the target value becoming smaller as the momentum term influences the current weight change with the previous one. Eventually, the synaptic weights will converge upon the target values.

If the succession of recent gradients has tended to alternate directions, then the sum will be relatively small and only small changes will be made in the parameter vector. This could occur in the local minimum area, successive changes would just serve to bounce back and forth past the minimum. If, however, recent gradients tend to align, larger changes needed in the parameter vector and thereby move more rapidly across a large region of descent and possibly across over a small region of ascent that screened off a deeper local minimum. Of course, if the learning rate is well chosen, then successive gradients will tend to be orthogonal and a weighted sum will not cancel itself out.

Thus, momentum allows a network to train faster, both by permitting a higher learning rate and snowballing synaptic weight adjustment. When using high learning rates, momentum also tempers a backpropagation network's tendency to thrash around the target values without ever actually achieving them.

2.5.3 Cross Validation

During training, the input and desired data will be repeatedly presented to the network. As the network learns, the error will drop towards zero. Lower error, however, does not always mean a better network. It is possible to overtrain a network. To avoid overtraining, a validation set should be used. The validation set is used as a pseudo-test set and is not used for training but for stopping criteria.

Training stops when minimum validation error is reached and the current network state is used on the testing set. However, as there are many local optima in the validation set, there are some issues when using it. During the initial phase of training, the error on validation set will be oscillatory .Also; Finnoff, Hergert & Zimmermann (1993), Lang, Waibel & Hinton (1990), Morgan & Bourlard (1990) and Prechelt (1994) suggested to proceed the training untill the error increases.

When using cross validation, a decision should be made to decide how to divide data into a training set and a validation set, also called the test set. The network is trained with the training set, and the performance checked with the test set. The neural network will find the input-output map by repeatedly analyzing the training set. This is called the network training phase. Most of the neural network design effort is spent in the training phase (Ang, Tan & Al-Mamun, 2008).

Training is normally slow because the network's weights are being updated based on the error information. At times, training will strain the patience of the designer. But a carefully controlled training phase is indispensable for good performance, so be patient.

There is a need to monitor how well the network is learning. One of the simplest methods is to observe how the cost, which is the square difference between the network's output and the desired response, changes over training iterations. This graph of the output error versus iteration is called the *learning curve*. The training phase also holds the key to an accurate solution, so the criterion to stop training must be very well delineated. The goal of the stop criterion is to maximize the network's generalization.

It is relatively easy to adapt the weights in the training phase to provide a good solution to the training data. However, the best test for a network's performance is to apply data that it has not yet seen.

To test the network, the weights must be freezed after the training phase and apply data that the network has not seen before. If the training is successful and the network's topology is correct, it will apply its past experience to this data and still produce a good solution. If this is the case, then the network will be able to generalize based on the training set.

A network with enough weights will always learn the training set better as the number of iterations is increased. However, this decrease in the training set error is not always coupled to better performance in the test set. When the network is trained too much, the network memorizes the training patterns and does not generalize well.

A practical way to find a point of better generalization is to set aside a percentage of the training set and use it for cross validation. the error in the training set and the validation set should be observed. When the error in the validation set increases, the training should be stopped because the point of best generalization has been reached. Cross validation is a powerful method to stop the training.

2.5.4 Sensitivity Analysis

As training a neural network, the effect that each of the network inputs is having on the network output should be observed. This provides feedback as to which input channels are the most significant. From there, the input space can be pruned by removing the insignificant channels. This will reduce the size of the network, which in turn reduces the complexity and the training times.

Sensitivity analysis is a method for extracting the cause and effect relationship between the inputs and outputs of the network. The network learning is disabled during this operation such that the network weights are not affected. The basic idea is that the inputs to the network are shifted slightly and the corresponding change in the output is reported either as a percentage or a raw difference.

2.6 Artificial Neural Network Learning Algorithms

The ANN methodology enables us to design useful nonlinear systems accepting large numbers of inputs, with the design based solely on instances of input–output relationships. For a training set *T* consisting of *n* argument value pairs and given a d-dimensional argument *x* and an associated target value *t* will be approximated by the neural network output. The function approximation could be represented as:

$$T = \{(x_i, t_i) : i = 1 : n\}$$

In most applications, the training set *T* is considered to be noisy and the goal is not to reproduce it exactly but rather to construct a network function that generalizes well to new function values by selecting the weights to learn the training set is a solution to the problem. The notion of closeness on the training set *T* is typically formalized through an error function of the form;

$$\varphi_T = \sum_{i=1}^{n} \|y_i - t_i\|^2$$

where y_i is the network output. The target is to find a neural network η such that the output $y_i = \eta(x_i, w)$ is close to the desired output t_i for the input x_i (w = strengths of synaptic connections). The error $\varphi_T = \varphi_T(w)$ is a function of w because $y = \eta$ depends upon the parameters w defining the selected network η.

The objective function $\varphi_T(w)$ for a neural network with many parameters defines a highly irregular surface with many local minima, large regions of little slope and symmetries. The common node functions such as hyperbolic tangent (*tanh*) are differentiable to arbitrary order through the chain rule of differentiation, which implies that the error is also differentiable to arbitrary order. For $\varphi_T(w)$ a Taylor's series expansion in w can be made so that a truncation can be met due to a local minimum.

The gradient (first partial derivative) vector is represented by

$$g(w) = \nabla \varphi_T \Big|_{w} = \left[\frac{\partial \varphi_T}{\partial w_i} \right] \Big|_{w}$$

The gradient vector points in the direction of steepest increase of φ_T and its negative points in the direction of steepest decrease. The second partial derivative also known as Hessian matrix is represented by H:

$$H(w) = H_{ij}(w) = \nabla^2 \varphi_T(w) = \frac{\partial^2 \varphi_T(w)}{\partial w_i \partial w_j}$$

The Taylor's series for φ_T, assumed twice continuously differentiable about w^0, can now be given as

$$\varphi_T(w) = \varphi_T(w^0) + g(w^0)^T (w - w^0)^T + \frac{1}{2}(w - w^0)^T H(w^0)(w - w^0) + O(\left\| w - w^0 \right\|^2)$$

Where $O(\delta)$ denotes a term that is of zero-order in small δ such that

$$\lim_{\delta \to 0} (O(\delta)/\delta) = 0$$

2.6.1 Multiple Minima Problem In Neural Networks

A long recognized bane of analysis of the error surface and the performance of training algorithms is the presence of multiple stationary points, including multiple minima. Analysis of the behavior of training algorithms generally use the Taylor's series expansions discussed earlier, typically with the expansion about a local minimum w^0.

However, the multiplicity of minima confuses the analysis because it can be possible to converge to the same local minimum. Hence the issue of many minima is a real one. According to Auer, Herbster & Warmuth (1996), to prevent this situation, differentiable learning algorithms can be used.

Different learning algorithms have their staunch proponents, who can always construct instances in which their algorithm perform better than most others. There are three types of optimization taken into consideration that are used to minimize the error function, $\varphi_T(w)$.

Gradient descent and conjugate gradient are general optimization methods whose operation can be understood in the context of minimization of a quadratic error function. Although the error surface is not quadratic, for differentiable node functions, it can be in the neighborhood of the local mininum, such an analysis provides information about the behaviour of the training algorithm over a number of iterations up to its goal.

The third method, Levenberg-Marquardt is specifically adapted to minimization of an error function that arises from a squared error criterion of the form.

When training a neural network, the output error should be minimized at each node. Gradient descent is an iterative optimization process which moves a weight towards the minimum of the error function. In essence, the process finds the slope of the error curve by taking its derivative; multiplies it by a stepsize factor, the learning rate discussed; and subtracts this result from the current weight value. the value is subtacted from the weight because the negative of the gradient represents the direction of steepest descent down the curve of the error function. As running through all the epochs in a training cycle, it will be possible to be closer to the minimum error and the weight at each node approaches an ideal value.

2.6.2 Backpropagation Algorithm

The problem with the neural network learning models described thus far is that they define weight changes for the output layer only; the weight changes are based on an error term only available at the output layer. This was the problem that Rosenblatt encountered: a lack of a teaching process (error term) for the hidden units. To solve linearly inseparable problems, multilayer networks are required. Thus, a method of training the hidden layer is called for.

Backpropagation refers to the backwards distribution of error used to train a multilayer network. In particular, backpropagation proposes a method of estimating the error of a hidden layer in a neural network and so permits the use of the learning law for hidden units. This allows for adjustment of the hidden layer's synapses even though the desired output of the hidden units is not known. The process could be recursively applied for more hidden layers.

Backpropagation is one of the most commonly used supervised training algorithms. However, because backpropagation is a supervised learning algorithm, it is required that a set of fact data be obtainable which associates input patterns with correct outputs. Also, backpropagation has few if any self-organizing aspects and as such a very good sense of the problem with regards to network topology (number of units per layer) is necessary (Blum, 1992).

Backpropagation provides an effective method for evaluating the gradient vector needed to implement the steepest descent, conjugate gradient, and Levenberg-Marquardt algorithms. Backpropagation differs from straightforward gradient calculations using the chain rule for differentiation in the way it organizes efficiently the gradient calculation for networks having more than one hidden layer.

Backpropagation iteratively selects a sequence of parameter vectors $\{w_k, k = 1:T\}$ for a moderate value of running time *T*, with the goal of having $\{\varphi_T(w_k) = \varphi(k)\}$

converge to a small neighbourhood of a good local minimum rather than the usually inaccessible global minimum (Fine, 1999).

$$\varphi_T^{\ *} = \min_{w \in W} \varphi_T(w)$$

The simplest steepest descent algorithm uses the following weight update in the direction of $d_k = -g_k$ with a learning rate or step size α_k.

$$w_{k+1} = w_k - \alpha_k g_k$$

A good choice $\alpha_k^{\ *}$ for the learning rate α_k for a given choice of descent direction d_k is the one that minimizes $\varphi_{(k+1)}$.

$$\alpha_{k+1} = \arg\min_{\alpha} \varphi(w_k + \alpha d_k)$$

To carry out the minimization,

$$\left.\frac{\partial \varphi(w_k + 1)}{\partial \alpha}\right|_{\alpha = \alpha_k^{\ *}} = \left.\frac{\partial \varphi(w_k + \alpha d_k)}{\partial \alpha}\right|_{\alpha = \alpha_k^{\ *}} = 0$$

To evaluate this equation, it must be noted that

$$\frac{\partial \varphi(w_k + \alpha d_k)}{\partial \alpha} = g_{k+1}^{\ \ T} d_k$$

and for optimal learning rate, the orthogonality condition should be satisfied

$$g_{k+1}^{\ \ T} d_k = 0$$

When the error function is not specified analytically, then its minimization along d_k can be accomplished through a numerical line search for d_k or through numerical differentiation. The line search avoids the problem of setting a fixed step size. Analysis of such algorithms often examine their behavior when the error function is truly a quadratic. In the current notation,

$$g_{k+1} - g_k = \alpha_k H d_k$$

Hence the optimality condition derived from the orthogonality condition for the learning rate α_k becomes

$$\alpha_k^* = \frac{-d_k^T g_k}{d_k^T H d_k}$$

When search directions are chosen via $d_k = -M_k g_k$, with M_k symmetric, then the optimal learning rate is

$$\alpha_k^* = \frac{-g_k^T M_k g_k}{g_k^T M_k H M_k g_k}$$

$$\alpha_k^* = \frac{-g_k^T g_k}{g_k^T H g_k}$$

α_k^* is the reciprocal of an expected value of the eigenvalues $\{\lambda_i\}$ of the Hessian with probabilities determined by the squares of the coefficients of the gradient vector g_k expanded in terms of the eigenvectors $\{e_i\}$ of the Hessian:

$$\frac{1}{\alpha_k^*} = \sum_{i=1}^{P} q_i \lambda_i \rightarrow q_i = \frac{(g_k^T e_i)^2}{g_k^T g_k}$$

That algorithm suffers from greed. The successive directions do not generally support each other in that after two steps; the gradient is usually no longer orthogonal to the direction taken in the first step.

In the quadratic case, there exists a choice of learning rates that will drive the error to its absolute minimum in no more than $p + 1$ steps where p is the number of parameters (Fine, 1999). To see this,

$$\varphi(w) = \varphi(w^*) + \frac{1}{2}(w - w^*)^T H (w - w^*) = \varphi(w^*) + \frac{1}{2} g^T H^{-1}$$

It is easily verified that if $g_k = g(w_k)$ then

$$g_k = \left[\prod_{1}^{k} (I - \alpha_j H) \right] g_0$$

For $k \geq p$, it can be achieved that $g_k = 0$ by choosing $x_1, \ldots, x_p$ any permutation of $1/\lambda_1, \ldots, 1/\lambda_p$, the reciprocals of the eigenvalues of the Hessian *H*; the resulting matrix annihilates each of the *p* eigenvectors and any other vector can be shown as their weighted sum. The step size is held at a constant value $\alpha_k = \alpha$,

The simplicity of this approach is belived by the need to carefully select the learning rate. If the fixed step size is too large, then there may be oscillation or divergent behaviour so that monotone reduction of the error function φ_T can be lost. A high learning rate may encounter a thrashing problem. Since the learning rate is constant, the network will thrash; that is, it will alternate between values on both sides of the desired value without ever actually converging to the desired value.

As the network converges upon the ideal value for the data, the high learning rate causes it to overshoot that ideal value. Then, on the next training cycle, the network will generate a synaptic update in reverse (the error will have the opposite sign) in an attempt to remedy the over-learning on the previous pass. If the step size is too small, much more iterations will be needed to get better results.

It is not practical to determine the optimal setting for the learning rate before training, and, in fact, the optimal learning rate changes during the training process, as the algorithm moves across the performance surface.

An adaptive learning rate attempts to keep the learning step size as large as possible while keeping learning stable. The learning rate is made responsive to the complexity of the local error surface. Basically, the initial network output and error are calculated. At each epoch new weights and biases are calculated using the current learning rate. New outputs and errors are then calculated.

Backpropagation changes each weight of the network based on its localized portion of the input signal and its localized portion of the error. The change has to be proportional (a scaled version) of the product of these two quantities. The mathematics may be complicated, but the idea is very simple. When this algorithm is used for weight change, the state of the system is doing gradient descent; moving in the direction opposite to the largest local slope on the performance surface. In other words, the weights are being updated in the direction of down.

2.6.2.1 Training Problems In Backpropagation

The search for the optimal weight values can get caught in local minima, so that the algorithm thinks it has arrived at the best possible set of weights even though there are other solutions that are better. Backpropagation is also slow to converge. In making the process simple, the search direction is noisy and sometimes the weights do not move in the direction of the minimum. Finally, the learning rates must be set heuristically.

The problems of backpropagation can be reduced. The slowness of convergence can be improved by speeding up the original gradient descent learning. Momentum learning is often recommended due to its simplicity and efficiency with respect to the standard gradient.

2.6.2.1.1 Initial Weights. The most obvious problem encountered in training backpropagation networks is that of their initial weights.

Gradient descent refers to the practice of minimizing the error of a function over several iterations. In a backpropagation network, a generalized least mean squared algorithm is used to modify network weights. The goal is to minimize the mean squared error between the desired and actual outputs of the network. The error for a pattern p is given by :

$$E_p = \sum_k (l_{p,k})^2$$

with k the node from the output layer and l the squared error between the output and desired value, backpropagation must discover a vector that minimizes E_p. Since the output of the network is a function of its weights, so must E be a function of the network weights (Mehrotra, 1997).

Thus, the starting weights of a neural network affect not only the initial outputs but also the error and, thus, the gradient descent. In other words, every set of starting weights for a backpropagation network has a different gradient descending to the state of minimum error.

The most commonly used method to combat the problem of initial starting weights is to run multiple trials of multiple networks. The idea is to eliminate the problem by running a particular network architecture with a number of different starting weights.

The performance of a particular set of network parameters is determined by considering all the sample runs of that network and comparing it to those of networks with other parameters. Since there are so many network parameters, such as hidden units, layers, activations, and so forth, adding even more trials makes the number of potential runs far too large for an exhaustive search. Rather, trials tend to be guided either by previous knowledge about the data, or the intuition of the expert crafting the network.

Clearly a system which removes the burden of this trial and error process from the neural network professional is desirable. The running time of such a system need not be an improvement over the previous method; it is the selection strategy that must first be optimized.

2.6.2.1.2 Number of Hidden Units. One of the most difficult choices a neural network designer must make in designing a backpropagation network is how many hidden units to employ. To begin, a small number of hidden units is usually better at generalizing to unseen data. A large number of hidden units tends to be a superior memorizer; however, the data to be learned can make a significant difference.

The number of nodes in the hidden layers defines the complexity and the power of the neural network model to describe underlying relationships and structures inherent in a training data (Kavzoglu & Mather, 2003).

A three-layer network (input, hidden, and output layers) is sufficient to approximate continuous functions arbitrarily well over a bounded compact set. Kimes, Gastellu-Etchegorry & Este (2002) highlights that a network with one hidden layer can form an arbitrarily close approximation to any continuous nonlinear mapping, assuming only that the transfer function computed by a neuron is nonconstant, bounded, continuous and monotone increasing.

Berberoglu, Curran, Lloyd & Atkinson (2007), Aitkenhead & Aalders (2008) and Kavzoglu (2009) have reported the advantages of the use of networks two hidden layers in classification.

For simple data for which the dimensionality, or number of classes, is known, it is often optimal to choose one hidden unit per data class. Unfortunately, data sets from the real world are not always well structured. Classes may overlap, be discontiguous, or have other properties which mean that a greater or fewer number of hidden nodes may actually be optimal. Combining this with the fact that each network has to be run many times with different starting values, and a guaranteed optimal solution becomes intractable.

A simple example of an analytic guideline for choosing the number of hidden units is the geometric pyramid rule (Koehler,1991). It asserts that the appropriate number of neurons follows a pyramid shape, with number decreasing from the input toward the output. In particular, the number of neurons assigned to each layer follows a geometric progression. Thus, if a three-layer network with *n* input neurons and *m* output neurons is being designed, the hidden layer would have the square root of *m*n* neurons. A similar rule applies to four-layer networks, as follows:

$$r = (n / m)^{1/3}$$

$$NumberOfHiddenUnits_1 = mr^2$$

$$NumberOfHiddenUnits_2 = mr$$

2.6.2.1.3 Length Of Training. Once a few networks are chosen with numbers of hidden units that are likely to work well. The designer must decide how long to train the network for. If, as is usually the case, the network will have to generalize previously unseen data, then training the network for a long period of time may be counterproductive; the network will memorize the data rather than extract the patterns contained therein. This is less of a problem if one of the networks has an appropriate number of hidden nodes to generalize sufficiently.

However, the previous section discussed the problems in determining such a number to any degree of certainty. Similarly, training the network for too short a time results in a network that performs sub-optimally on the known data. Ideally, if a method could be devised for selecting the number of hidden units with near-optimality, this problem would largely disappear. Alternately, a method of stopping training at the optimal point in training would also solve this problem. Again, trial runs of different lengths only compound the number of required trials, as they must be combined with the previous problems.

2.6.2.1.4 Evaluation Strategies. The final stumbling point in this maze of pitfalls is the evaluation strategy. If data that the network has been trained on is used, this biases the networks in favor of memorized patterns. Using test data (data withheld during the training phase) to evaluate the network and determine how to use that network's parameters in future iterations can be considered to contradict the idea of test data.

That is, the held out data is, in fact, influencing the network architecture. In some cases, it is therefore deemed necessary to hold out a third set of data as the final test set. Thus, in addition to selecting the discussed training parameters of the networks, selecting the evaluation strategy is itself no trivial decision.

2.6.2.2 Nonlinear Activation Functions

Backpropagation uses an optimization algorithm called gradient descent to determine each node's contribution to the final network output in order to adjust the weights when training the network. Gradient descent requires taking the derivative of the activation function, but the threshold step function traditionally used in the multi perceptron neuron and the perceptron is not differentiable. To solve this problem, a differentiable, nonlinear function can be used with a sigmoid shape (an S shape) instead of the standard threshold. Two common, nonlinear activation functions used in artificial neural networks, the logistic and hyperbolic tangent functions are main with the threshold activation function .

The most commonly employed sigmoid function is the logistic function:

$$f(x) = \frac{1}{1+e^{-x}}$$

which is a strictly increasing function that exhibits smoothness and asymptotic properties.

The hyperbolic tangent (*tanh*) function is as:

$$f(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$$

The practical difference between these two sigmoid functions is that the logistic function outputs values between (0,1), whereas the hyperbolic tangent outputs from (-1,1). Due to its structure, hyperbolic tangent (*tanh*) function is an ideal activation function. Therefore *tanh* function shall be used for activation function (Kalman & Kwasny, 1992).

2.6.3 Conjugate Gradient Algorithm

Conjugate gradient algorithm is a member of a class of learning algorithms called *second order methods*.

Standard gradient descent algorithms (like *step* and *momentum*) use only the local approximation of the slope of the performance surface (error versus weights) to determine the best direction to move the weights in order to lower the error. Second order methods use or approximate second derivatives (the curvature instead of just the slope) of the performance surface to determine the weight update.

If the performance surface is quadratic (which is only true in general for linear systems), then using a second order method can find the exact minimum in one step.

In nonlinear systems like neural networks, multiple steps will be needed. Each step, however, will typically lower the error much more than a standard gradient descent step. The problem with second order methods is that they require many more computations for each weight update. An algorithm that makes many poor decisions may perform better on average than a much slower algorithm that makes very good decisions.

2.6.3.1 Weight Update Equations

The conjugate gradient method is an excellent tradeoff between speed of computation and performance. The conjugate gradient method can move to the minimum of a *N*-dimensional quadratic function in *N* steps. By always updating the weights in a direction that is conjugate to all past movements in the gradient, all of the zig-zagging of first order gradient descent methods can be avoided. At each step, a new conjugate direction is determined and movement to the minimum error along this direction is maintained.

If the performance surface is quadratic, information from the Hessian can determine the exact position of the minimum along each direction, but for non-quadratic surfaces, a line search is typically used. In theory, there are only *N* conjugate directions in a space of *N* dimensions, so the algorithm is reset each *N* iterations. The advantage of conjugate gradient method is that there is no need to store, compute, or invert the Hessian matrix. The equations are:

$$\Delta w = \alpha(n)p(n)$$

$$p(0) = -g(0)$$

$$p(n+1) = -g(n+1) + \beta(n)p(n)$$

$$\beta(n)\frac{g^T(n+1)g(n)}{g^T(n)g(n)}$$

$$\alpha(n) = \arg\min(E(w(n) + \eta p(n))$$

where *w* are the weights, *p* is the current direction of weight movement, *g* is the gradient (backprop information), *p* is a parameter that determines how much of the past direction is mixed with the gradient to form the new conjugate direction. The equation for a is a line search to find the minimum mean squared error (MSE) along the direction p. The line search in the conjugate gradient method is critical to finding the right direction to move next. If the line search is inaccurate, then the algorithm may become brittle.

The *Scaled Conjugate Gradient* method (SCG) is the method that avoids the line search procedure. One key advantage of the SCG algorithm is that it has no real parameters. The algorithm is based on computing *H*d* where *d* is a vector. The Hessian times a vector can be efficiently computed in *O(W)* operations and contains only *W* elements. To ensure that the Hessian is positive definite, an offset is added to the Hessian, $H + \lambda l$. The formula for the step size a as in the conjugate gradient is:

$$\alpha = -\frac{p^T g}{p^T (H + \lambda l) p + \lambda |p|^2}$$

where p is the direction vector and g is the gradient vector as in the conjugate gradient method. The parameter λ varies from iteration to iteration when λ is high, the learning rate is small (the Hessian cannot be trusted), and when it is low the learning rate is large.

Doing a first order approximation, this approximation can be made:

$$s = (H + \lambda l) p \approx \frac{E'(w + \sigma p) - E'(w)}{\sigma} + \lambda p$$

which means that the Hessian calculations can be replaced with one additional evaluation of the gradients (backpropagation pass). The parameter λ must be set to ensure that the $H + \lambda l$ is positive definite so that the denominator will always be positive. If the value of the denominator is negative, λ is increased by a value $\bar{\lambda}$ so that it will be positive. Additionally, λ is adjusted based upon how closely the current point in the performance surface approximates a quadratic if the performance surface is far from quadratic, λ can be increased that would cause resulting in a smaller step size. The value Δ is used to determine *closeness to quadratic* and is estimated via:

$$\Delta = \frac{2(E(w) - E(w+\alpha p)}{\alpha p^T g}$$

This algorithm requires a number of global scalar computations. All matrix calculations can be done locally (parallel). It also requires one backpropagation pass to compute $E'(w+\sigma p)$ and one forward pass to compute $E(w+\alpha p)$. Conjugate gradient learning requires batch learning in the network. In general, each conjugate gradient batch weight update will take twice as long as a standard batch weight update using step or momentum gradient search.

2.6.4 Levenberg–Marquardt Algorithm

The Levenberg-Marquardt (LM) algorithm is one of the most appropriate higher-order adaptive algorithms known for minimizing the mean squared error of a neural network. It is a member of a class of learning algorithms called *pseudo second order methods*. Standard gradient descent algorithms use only the local approximation of the slope of the performance surface (error versus weights) to determine the best direction to move the weights in order to lower the error. Second order methods use the Hessian or the matrix of second derivatives (the curvature instead of just the slope) of the performance surface to determine the weight update, while pseudo-second order methods approximate the Hessian.

In particular, the Levenberg-Marquardt utilizes so called Gauss-Newton approximation that keeps the Jacobian matrix and discards second order derivatives of the error. If the performance surface is quadratic (which is only true in general for linear systems) then using a second order method can find the exact minimum in one step.

In nonlinear systems like neural networks, the big issue is that the performance surface may be non convex, and so quadratic approximations may require several steps for convergence, or more importantly they may diverge. A key advantage of the LM approach is that it defaults to the gradient search when the local curvature of the performance surface deviates from a parabola, which may happen often in neural computing.

In that learning algorithm, the error function is a sum of squares. Notation to the error vector and its Jacobian with respect to the network parameters w:

$$J = J_{ij} = \frac{\partial e_j}{\partial w_i}, i = 1, \ldots, p, j = 1, \ldots, n$$

The Jacobian matrix is a large $p \times n$ matrix, all of whose elements are calculated directly by backpropagation technique. The p dimensional gradient g for the quadratic error function can be expressed as:

$$g(w) = \sum_{i=1}^{n} e_i \nabla e_i(w) = Je$$

and the Hessian matrix by,

$$H = H_{ij} = \frac{\partial^2 \varphi_T}{\partial w_i \partial w_j} = \frac{1}{2} \sum_{k=1}^{n} \frac{\partial^2 e_k^2}{\partial w_i \partial w_j} = \sum_{k=1}^{n} \left(e_k \frac{\partial^2 e_k}{\partial w_i \partial w_j} + \frac{\partial e_k \partial e_k}{\partial w_i \partial w_j} \right)$$

where,

$D = \sum_{i=1}^{n} e_i \nabla^2 e_i$ and the expression yields as:

$$H(w) = JJ^T + D$$

The key to the LM algorithm is to approximate this expression for the Hessian by replacing the matrix D involving second derivatives by the much simpler positively scaled unit matrix $\in I$. The LM is a descent algorithm using this approximation in the form,

$$M_k = [JJ^T + \in I]^{-1}$$
$$w_{k+1} = w_k - \alpha_k M_k g(w_k)$$

Successful use of LM requires approximate line search to determine the rate α_k. The matrix JJ^T is automatically symmetric and non-negative definite. The typically large size of J may necessitate careful memory management in evaluating the product JJ^T. Any positive $\in$ will ensure that M_k is positive definite, as required by the descent condition. The performance of the algorithm thus depends on the choice of $\in$.

When the scalar $\in$ is zero, this is just Newton's method, using the approximate Hessian matrix. When $\in$ is large, this becomes gradient descent with a small step size. As Newton's method is more accurate, $\in$ is decreased after each successful step (reduction in performance function) and is increased only when a tentative step would increase the performance function. By doing this, the performance function will always be reduced at each iteration of the algorithm.

There is only one parameter that can be set the *Initial Lambda*. The Lambda parameter governs the step size, and it is dynamically adjusted based on the direction of the error. If the error decreases between weight updates, then the Lambda parameter is decreased by a factor of 10. Conversely, if the error increases then Lambda is increased by a factor of 10. The Initial Lambda is only specifies the Lambda for the first epoch. This normally does not need to be changed from the default.

In this thesis, basically, multilayer perceptron neural network with one hidden layer and optimum number of hidden layers were varied from 1 to 5 and the speed of convergence and generalization error for each of the three learning algorithms was observed. The effect of node activation functions and *tanh-sigmoidal activation function* (TSAF). Computational complexities of the different learning algorithms were also noted during each event. The experiments were replicated 5 times each with a different starting condition (random weights) and the best (minimum) errors were reported. No stopping criterion, and no method of controlling generalization is used other than the maximum number of updates (epochs). All networks were trained for an identical number of stochastic updates .

CHAPTER THREE
EVOLUTIONARY ALGORITHMS

Evolutionary algorithms are identified by four main paradigms, which are summarized as follows (Freitas, 2000):

- Genetic algorithms
- Genetic programing
- Evolutionary strategies
- Evolutionary programming

3.1 Genetic Algorithms

Genetic algorithms are a randomized search method and just one form of evolutionary algorithms. They use randomness combined with laws of probability to direct search in a direction where improvement is likely. More correctly, genetic algorithms direct a search in many directions that are likely. Goldberg (1989) identifies four ways that genetic algorithms differ from the traditional methods discussed above:

- Genetic algorithms work with a coding of the parameter set, not with the parameters themselves. That is the algorithm which generates many possibilities simultaneously and then evaluates them.
- Genetic algorithms use a population of potential solutions. They are inherently parallel, and not restricted to considering a single point at a time as other methods are. This also is the reason for their robustness and ability to overcome local peaks in a search space.
- Genetic algorithms use a fitness, or objective function to determine viability of potential solutions. They do not use derivatives, or "other auxiliary knowledge." Thus they can be tailored to any domain where some judgement of the "goodness" of a result can be made, regardless of whether or not the domain has a search space that conforms nicely to the laws of calculus.

- Genetic algorithms guide the transitions in the search space using probabilistic, not deterministic rules. Unlike a hill climbing search, for example, which might have a rule to the effect of "if a higher point exists adjacent to the current one, choose it", genetic algorithms assign likelihood of a good search direction based on the results of the pay off function relative to other directions being explored.

Thus, genetic algorithms produce a robust search algorithm which works across a wide variety of domains, many of which are not suited to traditional search algorithms. Their inherent parallelism allow them to search a space more efficiently and quickly than many traditional algorithms. Though they are randomized, they are not random techniques; they use randomness as a tool to direct search in promising directions.

Genetic algorithms can be useful for finding a good solution to a problem when it is not known how to formulate an algorithm to find the ideal solution in a direct, step-by-step manner. Starting with a random population of individuals, these individuals each have a chromosome composed of genes which encode a solution to the problem trying to be solved. For initial population, the gene values are assigned to a random value.

Thus, the population of individuals starts off as random solutions to the problem. Because they are randomly generated, these solutions tend to be poor ones. Although genetic algorithm (GA) does not require us to know how to create a good solution, it does require to be able to determine how good a given solution is. This is most often an easy task. In classification, benchmark can be made on how well a classifier solution performs with a test dataset. GA uses what is called a fitness function to determine how good a solution is. This fitness function is used to assign the individuals in the population a fitness value.

Genetic algorithms simulate natural evolution, and just like real-world entities. It is the fitter individual that tends to survive. The fitter individual has a more likely chance of mating, having offspring, and thereby further spreading its genetic properties within the population. These individuals are taken through many generations, and although they start out as random and generally poor solutions to the problem at hand, the process of artificial evolution hones their chromosomes until the fittest individual is picked out of the population and create a very good solution from the blueprint encoded in its genes.

The steps for a basic genetic algorithm are as follows :

1. Create an initial population with random gene values.
2. Apply the fitness function to individuals to determine their fitness.
3. Loop until a stopping criterion is met.
4. Select individuals based on fitness.
5. Create offspring by applying genetic operators to the selected individuals.
6. Apply the fitness function to the offspring.
7. Update the current population.

3.2 Evolutionary Strategies

Evolutionary Strategies (ES) uses a real-valued vector to represent an individual. The representation often includes parameters to control mutation operations on the individual in addition to the values for the variables in the problem being solved. Mutation is the primary genetic operator; however, later implementations began to use crossover as well.The mutation operator is usually applied according to a normal distribution by which small mutations are more probable than large ones.

```
Set generation t = 0.
Create the initial population, P(t).
Evaluate the fitness of each individual in P(t).
While ending condition end(P(t),t) is not satisfied, do
Calculate P'(t)= recombination(P(t)), and P"(t) = mutation(P'(t))
        Evaluate P"(t).
        Q = set of individuals chosen from the original population, P(t).
        Reproduce next population, P(t + 1) = selection(P"(t) U Q),
        Set t = t + 1.
```

Figure 3.1 The pseudocode for an evolutionary algorithm

3.3 Genetic Programming

Genetic Programming (GP) is frequently used to evolve programs to perform a specific task. The individual is often represented as a binary tree. This representation is useful for programming languages such as LISP where the operator or function can form the root node of a subtree and the operands form the leaf nodes.

3.4 Evolutionary Programming

Evolutionary Programming (EP) is similar to ES in that it uses a real-valued vector to represent an individual. EP systems use a similar, normally distributed, mutation operator as their exploratory operator, and they generally do not use crossover. EP was originally researched as a technique to evolve finite-state machines .

These four paradigms of evolutionary algorithms are similar in many ways. It can be difficult to define clear boundaries between the types as they all use a similar methodology and, within a given paradigm, many different algorithms are possible.

Table 3.1 The differences of evolutionary algorithm paradigms

	GA	GP	ES	EP
Chromosome	binary string	tree-structured program	real vector + strategy parameters	finite state machine
Mutation	reverse 1-bit	replace random subtree	perturb strategy param. then mutate target vector	node, link operators
Recombination	crossover (primary)	subtree crossover (primary)	separate recombination on target vector and parameters	not used
Selection	probabilistic	varies	deterministic	deterministic

Evolutionary strategies and evolutionary programming differ from genetic algorithms and genetic programming in that they usually apply genetic operators before selecting individuals based on their fitness. Genetic algorithms and genetic programming generally apply natural selection first and then apply crossover and mutation operators. As research continues, however, characteristics of one paradigm are adopted by other paradigms, an experiencing an overall unification effect in the field of evolutionary algorithms.

3.5 Genetic Operators

For every generation in a genetic algorithm, genetic operators can be applied on selected individuals in the population. The two most common genetic operators in genetic algorithm are crossover and mutation.

3.5.1 Crossover

The primary genetic operator in GA systems is generally crossover, also called recombination. It simulates the process of mating and having offspring in nature. In biological sexual reproduction, the offspring receive a part of each of the two parents' genetic material. The same is true of crossover in genetic algorithms. Two parent individuals are selected from the population to create children, whose genetic material is a combination of that of their parents.

To create the chromosomes in the children, a randomly determined locus or crossover point is set in the parents' chromosomes. Most genetic algorithm implementations create two children as the result of crossover. The first child receives a copy of the first parent's genes up to the crossover point and the second parent's genes after that point. The opposite is true for the second child; it receives the second parent's genes to the left of the locus, and the first parent's genes to the right.

3.5.1.1 Crossover Types

Crossover types used for application are; one point, two point, arithmetic and heuristic crossovers.

3.5.1.1.1 One Point. Randomly selects a crossover point within a chromosome then interchanges the two parent chromosomes at this point to produce two new offspring. Consider the following two parents that have been selected for crossover. The "|" symbol indicates the randomly chosen crossover point.

```
Parent 1:    11001|010
```

```
Parent 2:    00100|111
```

After interchanging the parent chromosomes at the crossover point, the following offspring are produced:

```
Offspring1: 11001|111
```

```
Offspring2:  00100|010
```

3.5.1.1.2 Two Point. Randomly selects two crossover points within a chromosome then interchanges the two parent chromosomes between these points to produce two new offspring. Consider the following two parents that have been selected for crossover. The "|" symbols indicate the randomly chosen crossover points.

```
Parent 1:    110|010|10
```

```
Parent 2:    001|001|11
```

After interchanging the parent chromosomes at the crossover point, the following offspring are produced:

```
Offspring1: 110|001|10
```

```
Offspring2: 001|010|11
```

3.5.1.1.3 Arithmetic crossover. Linearly combines two parent chromosome vectors to produce two new offspring according to the following equations:

$$\boxed{\mathit{Offspring1} = a * \mathit{Parent1} + (1- a) * \mathit{Parent2}}$$

$$\boxed{\mathit{Offspring2} = (1 - a) * \mathit{Parent1} + a * \mathit{Parent2}}$$

where *a* is a random weighting factor (chosen before each crossover operation). If the chromosomes contain any integer genes, these genes are rounded after the linear combination operation. If the chromosome contains any binary genes, uniform crossover is performed on these genes since arithmetic crossover does not apply.

*3.5.1.1.4 Heuristic crossove*r. Uses the fitness values of the two parent chromosomes to determine the direction of the search. The offspring are created according to the following equations:

$$\boxed{Offspring1 = BestParent + \mathrm{r} * (BestParent - WorstParent)}$$

$$\boxed{Offspring2 = BestParent}$$

where *r* is a random number between 0 and 1. It is possible that *Offspring1* will not be feasible. This can happen if *r* is chosen such that one or more of its genes fall outside of the allowable upper or lower bounds. For this reason, heuristic crossover has a parameter (*n*) for the number of times to try and find an *r* that results in a feasible chromosome. If a feasible chromosome is not produced after *n* tries, the *worst parent* is returned as *Offspring1*. If the chromosomes contain any integer genes, these genes are rounded after the heuristic crossover operation. If the chromosome contains any binary genes, uniform crossover is performed on these genes since heuristic crossover does not apply.

Likewise, schema found in individuals with lower fitness values will tend to decrease in future generations.

3.5.2 Mutation

Mutation is generally considered to be a secondary genetic operator in genetic algorithm systems, after crossover. Mutation, however, can play an important role in the exploration of the search space because it actually changes the gene alleles to new values rather than simply recombining what already exists in the population. Mutation can, therefore, introduce a gene value which does not currently exist in any individual in the population.

In its simplest form, when an individual's chromosome consists of a binary string, the mutation operator inverts the bit value for a randomly selected gene. (It flips the bit from "1" to "0" or from "0" to "1".) A more conservative method is not to automatically invert the bit, but rather to calculate a new bit value (with a 50% chance for a "0" or a "1"). The new allele may or may not match the original value.

More complex chromosomes, those which encode integer or real number values, can use a mutation operator which adds or subtracts a random amount within specified bounds from the gene. This is sometimes called creep mutation. Mutation traditionally occurs at the end of a crossover operation. The mutation rate is a GA parameter that defines a set chance that a child individual will receive a gene mutation when that child is created. This research uses a method which creates mutated clones of individuals independently of the crossover genetic operator.

CHAPTER FOUR
EVOLUTIONARY ARTIFICIAL NEURAL NETWORKS

4.1 Neural Networks For Genetic Based Classification

Having discussed the basics of backpropagation networks and genetic algorithms, the question of combining the two is raised. While neural networks can be powerful tools for pattern recognition, optimization, classification, and mapping problems in general, they are by no means easily constructed.

Traditionally, neural networks are designed and implemented by specialists-professionals with in-depth knowledge of the strengths and weaknesses of various network architectures. While this results in well designed networks, it can also give rise to certain problems. While the underlying algorithms may be relatively simple, network parameters such as: learning rate, momentum, initial weights, number of layers, and number of units per hidden layer play a large part in the ability of a particular network to solve a given problem.

Even when selected and implemented by an expert with knowledge both of neural networks and the problem domain, the process is often little better than trial and error. A better way to determine optimal parameter settings for a neural network is required. The goal of applying an evolutionary algorithm is to automate ad hoc process of neural network design (Castillo-Valdivieso, Merelo & Prieto, 2002).

The main steps for evolutionary artificial neural network (EANN) based classification is as follows:

1. Generate the initial population with multilayer perceptrons with random weight values in a specified range and specified initial number of hidden layer sizes.
2. Repeat for n generations:

(a) Evaluate the new MLP's (individuals): train them using the training set and obtain their fitness according to the number of correct classifications on the validation set and the network size (number of weights).

(b) Select the *s* best individuals in the population, according to their fitness, to mate using the genetic operators to obtain the new individuals.

(c) Replace the *s* worst individuals in the population by the new ones.

3. Use the best individual found to obtain the testing error using the test set.

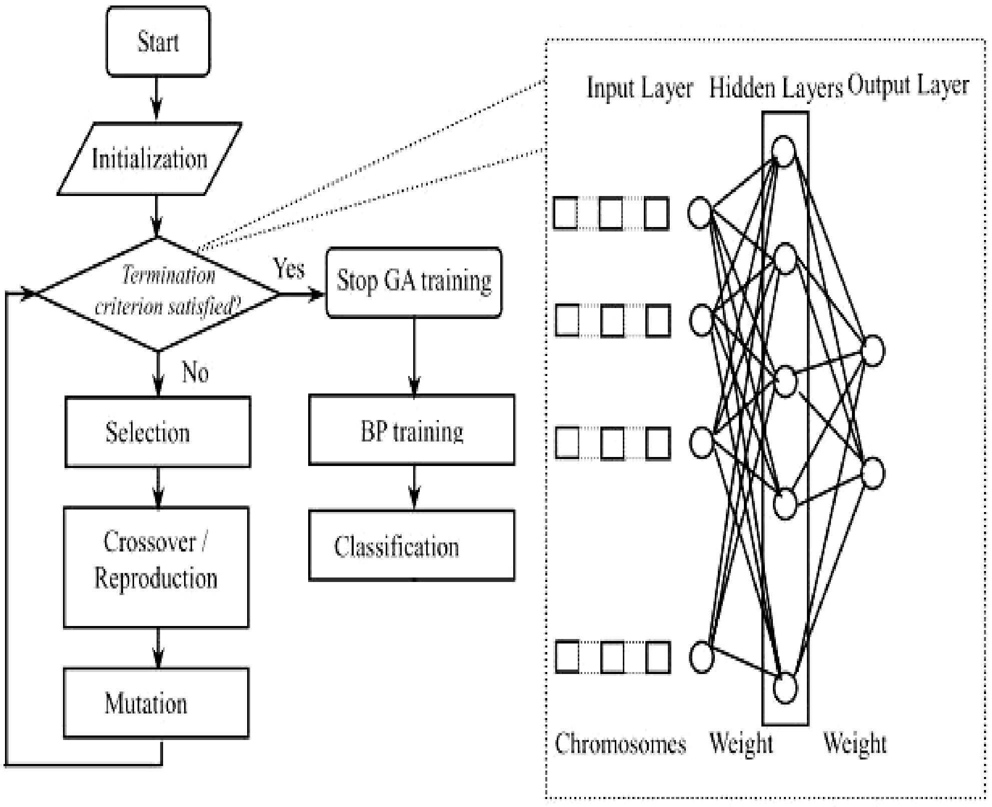

Figure 4.1. General framework for EANN based classification

4.2 Motivation For Evolutionary Artificial Neural Networks

EAs offer a much more flexible approach. Neural networks are, in essence, a hill climbing search. As such, they are subject to the pitfalls of getting stuck on local features of the solution space. Neural networks use an error calculation to compute a gradient to direct the search; the backpropagation network. These methods require smooth, continuous activation functions in order to derive gradient information .

In contrast, evolutionary algorithms do not perform direct calculation of gradients. Instead, they focus on blanketing the search space with potential solutions. This results in a far more global search which is much less likely to succumb to local features of the solution space. These advantages give evolutionary algorithms a much wider range of options; an evolutionary algorithm might use linear thresholds, splines, or product units where traditional neural networks might require a smooth sigmoid function. Further, computation of gradients in the more complex neural networks, such as recurrent networks, can be quite costly. EAs do not require these expensive calculations.

Thus, evolutionary algorithms complement the traditional neural network gradient-descent techniques quite well. Their simultaneous global search allows large, irregular search spaces to be covered in an automated manner, removing human drudgery, and human error, from the equation.

4.3 Types Of Evolutionary Artificial Neural Networks

EANNs differ from standard ANNs in that they have an extra stage of adaptation and learning based on an evolutionary or genetic system. There are three types of evolutionary algorithm based neural networks (Yao, 1999):

I. weight-evolving algorithms (WEAs),
II. topology-evolving algorithms (TEAs),
III. hybrid evolving algorithms (HEAs).

4.3.1 Weight-Evolving Algorithms (WEAs)

The process of backpropagation can be long and computationally intensive, and in some cases it does not result in an effective solution. In such a case a weight-evolving algorithm (WEA) can be applied, which may speed up the search for a solution.. The weights contained in the nodes are in the form of matrices that contain information from prior input data. Through backpropagation, these weights are updated and the overall network is trained to recognize certain patterns.

In a WEA system, the set of weights in the network nodes is evolutionarily adapted. Standard backpropagation would perform the same feat, but also could become trapped in a non-optimal solution. Using a GA, this is less likely to occur. In order to use a WEA, first a representation of the data must be chosen. There are two popular formats: binary and real number. The second phase of developing the WEA involves choosing the operators for mutation and crossover and deciding whether or not either or both will be used.

Binary representation is commonly used to represent data in genetic algorithms. It makes the operations of mutation and crossover easy to perform but consistency checking must be applied so that offspring are functional rather than illegal or inoperable. It is simple to use binary representation of the data. First, an algorithm is defined to extract the weights from the ANN in a specific order. Then the weights are converted into a fixed length binary string. Once the data is converted, the GA is performed on the dataset and the information is converted back to its standard form with a reversal algorithm. Finally, the information is placed in an offspring for the next iteration of the genetic algorithm (Janson & Frenzel, 1993; Tsukimoto & Hatano, 2003; Yao, 1999).

Real number representations can also be used to encode the weights of an ANN. The same method is used as in binary representation to extract and then re-encode information back into the ANN. However in real number representations, instead of changing the extracted weights to binary, they are represented by a single real number (Alsultanny & Aqel, 2003; Yao, 1999). While this scheme is easy to encode and decode, its primary operator is mutation and crossover is considerably harder to implement here than in a binary representation. This can hinder the efficiency of the algorithm but will not completely halt its progress; it has been shown that GAs can operate effectively using only one of their two major operations (Siebel, Krause, & Sommer, 2007).

4.3.2 Topology Evolving Algorithms (TEAs)

The next type of EANN is the topology evolving algorithms, which evolves ANN architectures or topologies. An ANN can be accurately represented by a graph. An ANN is a graph-like structure and has an architecture or topology that can be modified. Changing an ANN's topology can drastically improve or deteriorate its performance. In the past, engineering the topology of an ANN has been a job for a human being; this was a trial-and-error procedure. Since there is an infinite set of possible network structures available to solve each problem, a human being may not be able to find an efficient architecture. However, a TEA can be employed to find an efficient ANN topology that solves the problem.

This system can be more complex than the WEA method. This is because the entire structure of the network may be changed by the TEA and then must be completely retrained. However, it can also be more robust. The changed structures of the network may be capable of retaining very different patterns of information. The algorithm may find a structure that performs excellently that the human designers may never have conceived.

Because of the efficient encoding possibility, TEA is more applicable to WEA. (Boozarjomehry & Svrcek, 2001; Castillo, Merelo, Prieto, Rivas, & Romero, 2000; Janson & Frenzel, 1993). TEAs have also been modified to perform optimization as well as topographical evolution (Sexton, Dorsey & Sikander, 2004). One of the problems with TEAs is that the ANNs developed with them can grow to be extremely large and convoluted. Fortunately the algorithm can be adapted to perform self-pruning as it is evolving more efficient ANNs. Unnecessary weights and hidden nodes can thus be identified and removed from the ANN, which keeps the network smaller and more efficient (Blanco, Delgado, & Pegalajar, 2000; Castillo et al., 2000).

4.3.3 Hybrid Evolutionary Algorithms (HEAs)

The third type of EANN systems, HEAs, is a unification of the two systems described above. These systems adapt both the weight and topology of an ANN. This can be a complex process, but it can also be extremely effective. Both the adaptation of ANN weights and the adaptation of their topologies are effective means for searching a problem space. Combining these two techniques can result in a faster method for finding a solution (Stanley, 2004).

4.4 Algortihms For Evolution

The training mechanism is usually an iterative gradient descent algorithm, designed to minimize, step by step, the difference between the actual output vector of the network and the desired output vector, such as backpropagation in its different versions does encounter certain difficulties in practice:

(1) the convergence tends to be extremely slow;
(2) convergence to the global optimum is not guaranteed;
(3) learning constants and other parameters must be arrived at heuristically.

Incremental algorithms, are based on adding hidden neurons to a network of minimum size until the required precision is reached. These methods start with few hidden neurons and increase their number until the error is sufficiently small. One problem of these methods is that once the hidden neurons have been added they cannot be suppressed to reduce the size (the redundant information stored in the weights is never eliminated) and huge ANNs are usually obtained.

Furthermore, since the weights of existing neurons are frozen, the added ones are usually inefficient feature detectors, so the algorithm has to add even more units to improve the results obtained. In general, adding new units leads to overfitting (Castillo et al., 2000).

In this thesis, evolutionary algorithms are used to search for the optimal learning parameters, including weights, having pre-established the number of neurons and the connectivity between them.

4.5 Performance Measures For EANN Based Classification

4.5.1 Correlation Coefficient

The size of the mean square error (MSE) can be used to determine how well the network output fits the desired output, but it doesn't necessarily reflect whether the two sets of data move in the same direction. By simply scaling the network output, MSE can be changed without changing the directionality of the data. The correlation coefficient (*r*) solves this problem. By definition, the correlation coefficient between a network output *x* and and a desired output *d* is:

$$r = \frac{\dfrac{\sum_i (x_i - \bar{x})(d_i - \bar{d})}{N}}{\sqrt{\dfrac{\sum_i (d_i - \bar{d})^2}{N}}\sqrt{\dfrac{\sum_i (x_i - \bar{x})^2}{N}}}$$

The correlation coefficient is confined to the range $[-1,1]$. When $r = 1$ there is a perfect positive linear correlation between *x* and *d*, that is, they covary, which means that they vary by the same amount. When $r = -1$, there is a perfectly linear negative correlation between *x* and *d*, that is, they vary in opposite ways (when *x* increases, *d* decreases by the same amount). When $r = 0$ there is no correlation between *x* and *d*, so that the variables are called uncorrelated.

4.5.2 Confusion Matrix

A confusion matrix is a simple methodology for displaying the classification results of a network. The confusion matrix is defined by labeling the desired classification on the rows and the predicted classifications on the columns. For each exemplar, a 1 is added to the cell entry defined by (desired classification, predicted classification). Since the aim is matching the predicted classification to be the same as the desired classification, the ideal situation is to have all the exemplars end up on the diagonal cells of the matrix (the diagonal that connects the upper-left corner to the lower right).

There can be four different outcomes with regards to binary classification problems when a classifier makes a prediction about the class membership of a particular instance. The classifier may predict that an instance belongs to the positive class when in fact, it belongs to the positive class or it may predict that an instance belongs to the negative class when it in fact belongs to the negative class.

These two outcomes are called true positive (TP) and true negative (TN) respectively and are correct predictions. The other two outcomes are incorrect predictions or misclassifications. One type of misclassification is called false positive (FP), where a classifier predicts that an instance is a member of the positive class when it in fact is a member of the negative class. Respectively, the other type of misclassification is called false negative (FN). In this case, the classifier predicts that an instance is a member of the negative class when it in fact is a member of the positive class.

Table 4.1 depicts the four types of outcomes that can be produced by a binary classifier. This kind of tabular depiction of these four outcomes is called a *confusion matrix* or a contingency table. The classifier may use a classification strategy where it uses a parameter, called the decision threshold t $(0 < t < 1)$, in order to decide the class membership of a given instance. The default decision threshold equals 0.5 and specifies that the probability of an instance belonging to the positive class is equal to the probability of belonging to the negative class. The default decision threshold does usually not cause the classifier to perform optimal when the given dataset is imbalanced with regards to class distribution or when the costs for both types of misclassification are not equal. The decision threshold can be changed to account for the imbalance in the dataset and for unequal costs of misclassification.

Table 4.1 Confusion matrix

	Predicted as positive	**Predicted as negative**
Positive instance	True positive (TP)	False negative (FN)
Negative instance	False positive (FP)	True negative (TN)

The confusion matrix provides the sum for each type of outcome with regards to the total number of instances in the underlying dataset. It shows the total number of true positives (#TP), the total number of true negatives (#TN), the total number of false positives (#FP) and the total number of false negatives (#FN). These four values form the basis for performance metrics of confusion matrix. As the number of TP and TN values get higher and oppositely lower values of FP and FN, the better the rule will be determined and high accuracy values will be gained (Freitas, 2002).

4.5.3 Mean Square Error (MSE)

The formula for the mean squared error is:

$$MSE = \frac{\sum_{j=0}^{P}\sum_{i=0}^{N}(d_{ij} - y_{ij})^2}{NP}$$

where:

P=number of ouput processing elements

N=numbe rof exemplars in the data set

y_{ij}=network output for exemplar i at processing elemnt j

d_{ij}=desired output for exemplar i at processing element j

4.5.4 Normalized Mean Squared Error (NMSE)

The normalized mean squared error is defined by the following formula:

$$NMSE = \frac{P \times N \times MSE}{\sum_{j=0}^{P} \frac{N\sum_{i=0}^{N} d_{ij}^{2} - (\sum_{i=0}^{N} d_{ij})^{2}}{N}}$$

where

P=number of output processing elements

N=number of exemplars in the data set

MSE =mean squared error

d_{ij}=desired output for exemplar i at processing element j

4.5.5 Relative Percent Difference

The relative percent difference (RPD) is defined by the following formula:

$$\%Error = \frac{100}{N \times P} \sum_{j=0}^{P} \sum_{i=0}^{N} \frac{|dy_{ij} - dd_{ij}|}{dd_{ij}}$$

where

P=number of ouput processing elements

N=number of exemplars in the data set

dy_{ij}=denormalized network output for exemplar i at processing element j

dd_{ij}=denormalized desired output for exemplar i at processing element j

CHAPTER FIVE
COMPARISON OF NEURAL NETWORKS WITH EVOLUTIONARY ALGORITHMS FOR CLASSIFICATION

5.1 General Information

The aim of this thesis is to tune learning parameters (stepsize and momentum) and to set the initial weights and hidden layer size (number of hidden units) of a multilayer perceptron, based on an evolutionary algorithm and backpropagation. The neural network will produce from each set of inputs to a set of outputs. Given a random set of initial weights, the outputs of the network will be very different from the desired classifications. As the network is trained, the weights of the system are continually adjusted to incrementally reduce the difference between the output of the system and the desired response. This difference is referred to as the error and can be measured in different ways. The most common measurement is the mean squared error (MSE). The MSE is the average of the squares of the difference between each output processing element and the desired output.

The capacities of both of the two algorithms are intended to be used: the ability of evolutionary algorithm to end a solution close to the global optimum, and the ability of the backpropagation to tune a solution and reach the nearest local minimum by means of local search from the solution found by the evolutionary algorithm. The topology is selected from the incremental algorithm applied to the neural network.

Genetic operators; mutation, one point crossover, two point crossover, arithmetic crossover, heuristic crossover, incremental algorithm, the number of hidden units, and backpropagation applied as operator to the individuals of the population, are used.

Thus, the evoutionary algorithm searches and optimizes the architecture (number of hidden units), the initial weight setting for that architecture and the learning rate and momentum for that neural net. Unlike other approaches the maximum size of the hidden layer is not bounded in advance.

The classification accuracy or number of hits is obtained by dividing the number of hits among the total number of examples in the testing set according to the training and cross validation subsets.

The mutation operator modifies the weights of certain neurons, at random, depending on the specified application rate. It is based on the algorithm which modifies the weights of the network after each epoch of network training and adding and subtracting a small random number that follows uniform distribution with the interval [- 0.1; 0.1].

The learning rate is modified by the adaptive algorithm in order to avoid the negative effects of steps' handicaps. This operator is used with an application probability of 10%, that is 10% of weights are changed, which was found empirically to obtain better results than did lower probabilities. The crossover operator carries out the one point, two point, artihmetic and heuristic type crossovers.

By incremental algorithm, the difficulty is in guessing the number of the hidden layer neurons. By adding hidden neurons, it is not necessary to set the size of the EA search space. This operator is intended to perform incremental learning; it starts with a small structure and increments it, if neccesary, by adding new hidden units (Castillo et al., 2000).

To determine a satisfactory solution that meets the GA stopping criteria, every 5 generations the neural network is trained on for 1000 epochs by default.

To avoid overfitting, the usual procedure is followed of splitting the input data into training, cross validation and test sets. The training set with 50% of the whole exemplars will be used to evaluate the fitness of the individuals. The cross validation set consists of the 25% of the whole exemplars and will be used to estimate the generalization ability of the best result found during each generation.

The stopping criterion is reached if one of the following conditions is fulfilled: a number of generations are reached or the variance of the fitness of the best 10% of the population is less than 10^{-4}.

This illustrates the basic ingredients required in neural computation. The network requires input data and a desired response to each input. The more data presented to the network, the better its performance will be. Neural networks take this input-output data, apply a learning rule and extract information from the data. The network tries to adjust the weights to minimize the error. Therefore, the weights embody all of the information extracted during learning.

Two types of dataset is considered for performance evaluation of evolutionary neural networks. Each dataset are maintained from UC Irvine Machine Learning Repository.

Teaching assistant evaluation (tae) dataset has categorical-driven attribute types and on contrary, there is only one numerical attribute. Loh & Shih (1997) first reported this dataset. Classification tree is maintained as a search algorithm for this dataset to be classified. Because of lack of information on the results, no performance value gained to be benchmarked.

Lim, Loh & Shih (1999) used *tae* dataset for classification by neural networks. Learning vector quantiaztion and radial basis functions are types of neural networks used for classification performance. Mean error rate is calculated as a performance measure.

The other dataset is the *vehicle silhoutte (veh) dataset* that all the attributes are numerical. Lim, Loh & Shih (1999) used *veh* dataset for classification by neural networks. Learning vector quantization and radial basis functions are types of neural networks used for classification performance. Mean error rate is calculated as a performance measure.

Roscher & Föstner (2009) used bounded logistic regression (BLR) and support vector machines (SVMs) for classification. Mean error rate is calculated as a performance measure.

Because the two datasets have opposite attribute types in common, the effects of numerical and categorical attributes can be examined.

5.2 Teaching Assistant Evaluation (*tae*) Dataset

The data consist of evaluations of teaching performance over three regular semesters and two semesters of 151 teaching assistant (TA) assignments at the Statistics Department of the University of Wisconsin-Madison. The scores are grouped into three roughly equal-sized categories (low, medium and high) to form the class attribute. The predictor attributes are:

- ENG : Whether or not the TA is a native English speaker (binary)
- INST: Course instructor (25 categories)
- CRS: Course (26 categories)
- SMSTR: Summer or regular semester (binary)
- CLSSIZE: Class size (numerical)

Its main characteristic of this dataset is that there are two categorical attributes with large numbers of categories such as CRS has 26 categories and INST has 25 categories. Contrary to these categorical attributes, there is only one numerical attribute, CLSSIZE.

Table 5.1 Attribute information for *tae* dataset

Number of original attributes				Total
Numerical	Categorical			
	2	25	26	
1	2	1	1	5

As seen from the table, because of the large numbers of categories, large additional data is added to the problem identification. To identify the attiribute type from the exemplar set, (S) prefix is used as symbolic (categorical) to name the type of data. The attribute type that does not have the (S) prefix means that the data is numerical and can get different values.

The raw data shown in Appendix 1 is translated due to the categorical attributes. All the categorical data types inverted to a column and added to the data matrix. The translated data matrix is split into training, cross validation and test subsets.

Population size and number of generations affect processing time because the fitness value must be calculated for every chromosome in every generation. In the preliminary study; for determining the appropriate population size and generation number by trial and error, when the first implementation started with the population size of 50 and the generation number of 2500, the best fitness value was improved.

After a large number of generations, very much processing time required that is approximately 8.5 hours on average. When the population size was set to 30 and the generation number to 1000, the run values were nearly the same. The processing time decreased 4.6 hours on average. Therefore, the population size and generation number were set to 30 and 1000, respectively.

5.2.1 The Comparison Of Learning Algorithms Due To ANN Structure For Tae Dataset

According to the general classification with artificial neural networks, one hidden layered multilayer perceptron is enough. Basically, the selection of the learning algorithm is important. Three learning algorithms are taken into consideration:

- Momentum learning
- Conjugate-gradient
- Levenberg-Marquardt

One hidden layer is usually enough for training and testing purposes. Also, the incremental algorithm is taken into consideration. Starting from the simple neural network architecture, the system will be made complex to get more true classified results.

5.2.1.1 Momentum Learning

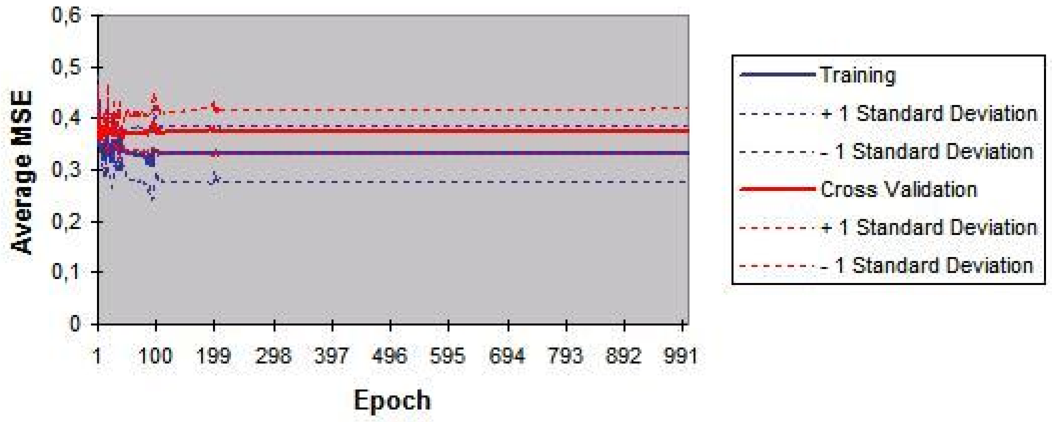

Figure 5.1 Average MSE with standard deviation boundaries for 5 runs

As seen from the Figure 5.1, average MSE values for training and cross validation behave similar. As the training moves on through the 100 epochs, training MSE values converge to minima; the cross validation MSE values converge a bit quicker due to lack of information.

Table 5.2 MSE statistics for training and cross validation subsets.

All Runs	Training Minimum	Training Standard Deviation	Cross Validation Minimum	Cross Validation Standard Deviation
Average of Minimum MSEs	0.266685786	0.055611345	0.35123632	0.005835692
Average of Final MSEs	0.330142037	0.055171731	0.375022402	0.0442916

Table 5.3 Best MSE for subsets

Best Networks	Training	Cross Validation
Run #	1	5
Epoch #	96	86
Minimum MSE	0,205366115	0,341732857
Final MSE	0,231447844	0,355215165

Seventy five exemplars are used for training purposes (50% of the total exemplars), and also 25% of total exemplars for both cross validation and testing purposes. Although most of the research papers suggest to use 75% of total exemplars for better training ability for the artificial neural network; because the classification problem is so categorical for *tae* dataset and there are few examplars such as 151 as numbers. This situation forces to choose the high percentage of cross validation and testing data. Of the total exemplars, 25% are used for both the cross validation and testing as subsets.

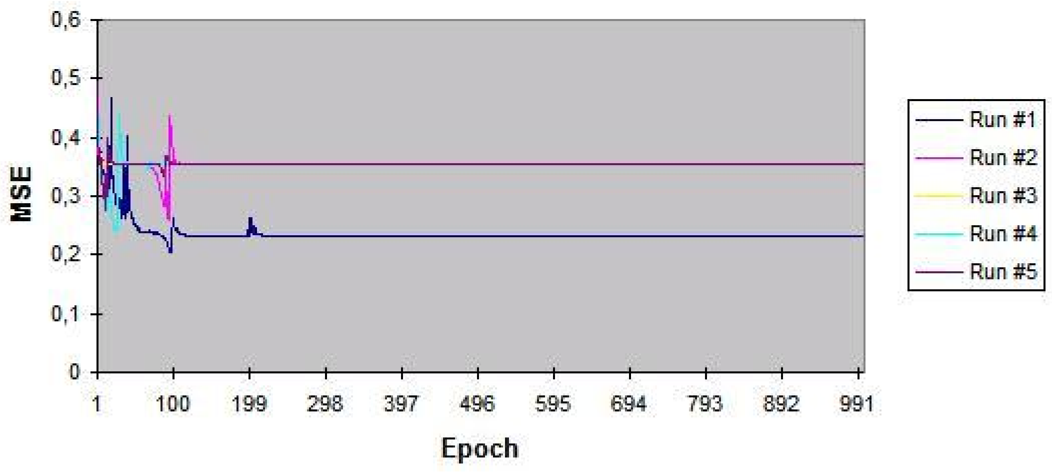

Figure 5.2 Training MSE

Training for each run shows similar values except run 2. The momentum value is 0.7 and the learning process is done in 90 epochs.

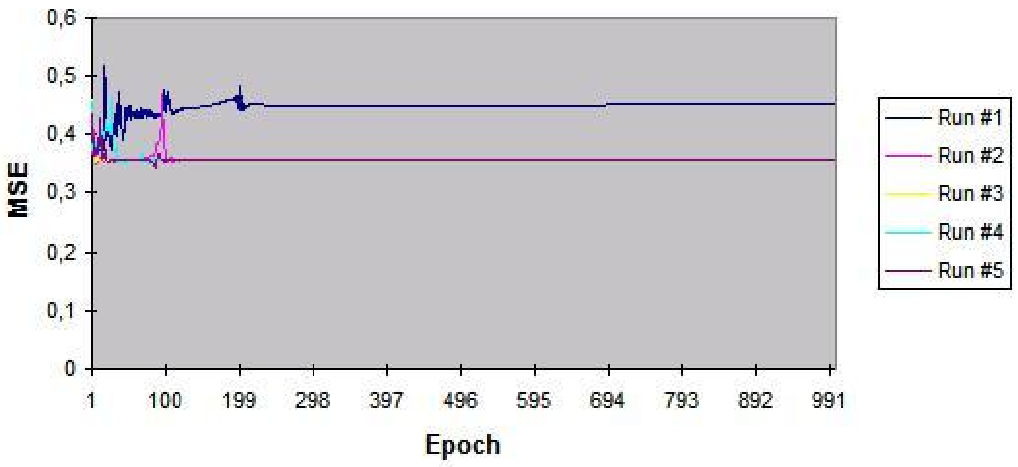

Figure 5.3 Cross validation MSE

Cross validation values memorize the values so that MSE values for the weight space get bigger. At that point, training stops and artificial neural network stops to learn. The training MSE gets stuck on the final MSE value 0.231447844. As seen from Figure 5.3, the learning process ends up nearly at epoch number 100 and overtraining starts.

Table 5.4 Minimum MSE values for momentum learning method

Subset	Minimum for Run #1	Minimum for Run #2	Minimum for Run #3	Minimum for Run #4	Minimum for Run #5
Training	0.20536612	0.25774687	0.35481534	0.24037249	0.27512811
Cross validation	0.35752953	0.3513768	0.35365703	0.35188538	0.34173286

As stated in the *tae* dataset information, the class attributes expressed as:

Table 5.5 Class attribute expression for tae dataset

CLSATTR(1)	Low
CLSATTR(2)	Medium
CLSATTR(3)	High

5.2.1.1.1 Testing Of Training Subset Data For Momentum Learning Method. By testing the training subset with momentum learning; “low” and “high” attributes are classified with acceptable accuracy. Also, correlation coefficients of “low” and “high” attributes are positive so that they act with MSE in general. On contrary, “medium”class attribute has the opposite sign.

Table 5.6 Confusion matrix for training subset data with momentum learning

Output / Desired	*CLSATTR(1)*	*CLSATTR(2)*	*CLSATTR(3)*
CLSATTR(1)	12	0	0
CLSATTR(2)	2	10	8
CLSATTR(3)	5	18	20

Table 5.7 Performance values for testing of training subset data with momentum learning

Performance	***CLSATTR(1)***	***CLSATTR(2)***	***CLSATTR(3)***
MSE	0.143512948	0.243078422	0.234345411
NMSE	0.758703318	1.038994016	1.001666366
MAE	0.3381967	0.479692429	0.472169182
Min Abs Error	0.170225037	0.335880312	0.358345517
Max Abs Error	0.825095007	0.662703527	0.6313603
r	0.628520881	-0.17457581	0.369111709
Percent Correct	63.15789474	35.71428571	71.42857143

5.2.1.1.2 Testing Of Cross Validation Subset Data For Momentum Learning Method. For the cross validation subset, the prediction for classification is not sufficient. The class attributes, “low” and “medium” are not classified with high accuracy.

Table 5.8 Confusion matrix for cross validation subset data with momentum learning

Output / Desired	*CLSATTR(1)*	*CLSATTR(2)*	*CLSATTR(3)*
CLSATTR(1)	3	1	0
CLSATTR(2)	1	2	3
CLSATTR(3)	6	9	14

Table 5.9 Performance values for testing of cross validation subset data with momentum learning

Performance	CLSATTR(1)	CLSATTR(2)	CLSATTR(3)
MSE	0.169440081	0.221146157	0,252267418
NMSE	0.888684009	1.038158349	1.025932468
MAE	0.347208433	0.456216919	0.493791711
Min Abs Error	0.169145245	0.335081526	0.358345517
Max Abs Error	0.831650983	0.664310315	0.627873589
r	0.361163896	0.01682359	0.325787126
Percent Correct	30	16.66666667	82.35294118

5.2.1.2 Conjugate-gradient learning method

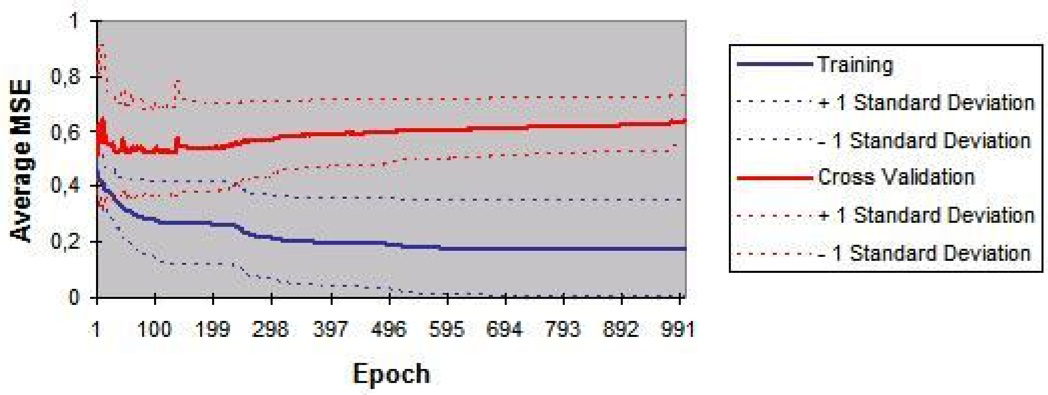

Figure 5.4 Average MSE for 5 runs

The average MSE values for the conjugate-gradient learning converge to minimum smoothly. As examining the statistical results in Table 5.10, standart deviation of the training values is wider than expected. For the sensitivity, the expected minimum and maximum average MSE values are twice the average MSE.

Table 5.10 MSE statistics for training and cross validation subsets.

All Runs	Training Minimum	Training Standard Deviation	Cross Validation Minimum	Cross Validation Standard Deviation
Average of Minimum MSEs	0.177516198	0.173992156	0.39301851	0.062992181
Average of Final MSEs	0.177516198	0.173992156	0.636851602	0.0897314

Table 5.11 Best MSE for subsets

Best Networks	Training	Cross Validation
Run #	5	2
Epoch #	854	150
Minimum MSE	0.07375151	0.355448257
Final MSE	0.07375151	0.531735682

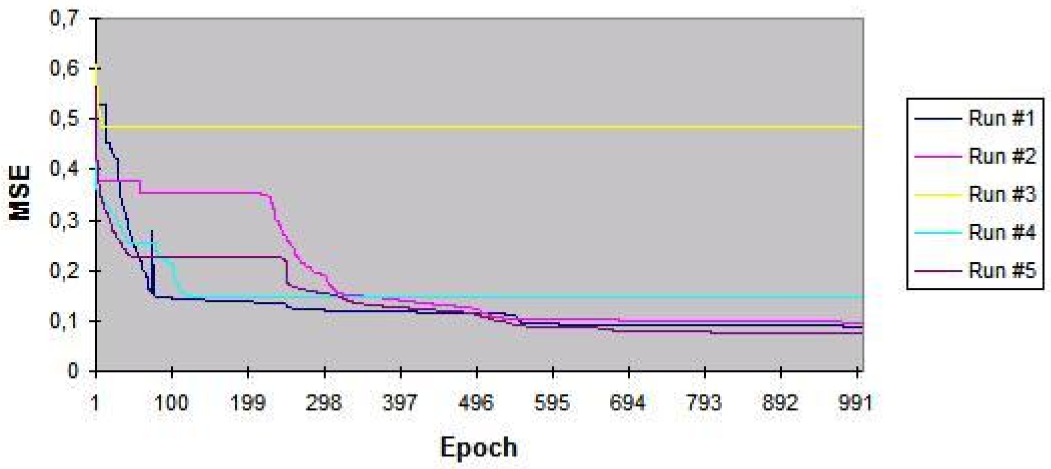

Figure 5.5 Training MSE

Conjugate-gradient learning algorithm learns faster than momentum but it's possible to be not training and as a result nor to be learning due to its structure. Run no 2 and run no 5 are far from minimization of the energy function, so the acuuracy of the networks decreases.

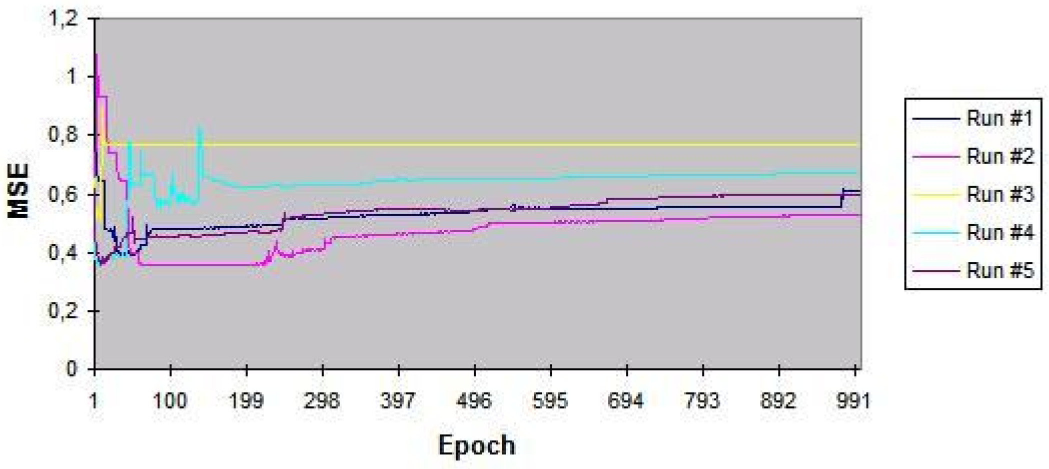

Figure 5.6 Cross validation MSE

Table 5.12 Minimum MSE values for conjugate-gradient learning method

Subset	**Minimum for Run #1**	**Minimum for Run #2**	**Minimum for Run #3**	**Minimum for Run #4**	**Minimum for Run #5**
Training	0.08605644	0.09718816	0.48491541	0.14566947	0.07375151
Cross validation	0.38787827	0.35544826	0.5032102	0.36202097	0.35653485

5.2.1.2.1 Testing Of Training Subset Data For Conjugate-Gradient Learning Method. As seen from the Table 5.13, the testing result shows that "medium" and "high" class attributes cannot be classified in a true manner. Because there is no data gathered for these attributes, *r* values cannot be seen further on.

Table 5.13 Confusion matrix for conjugate-gradient learning method (training subset)

Output / Desired	*CLSATTR(1)*	*CLSATTR(2)*	*CLSATTR(3)*
CLSATTR(1)	19	28	28
CLSATTR(2)	0	0	0
CLSATTR(3)	0	0	0

Table 5.14 Performance values for conjugate-gradient learning method (training subset)

Performance	***CLSATTR(1)***	***CLSATTR(2)***	***CLSATTR(3)***
MSE	0.261068455	0.380833276	0.367656031
NMSE	1.380178625	1.627801807	1.571478096
MAE	0.263488871	0.384166837	0.370985391
Min Abs Error	0	0	0
Max Abs Error	1	1	1
r	-	-	-
Percent Correct	100	0	0

5.2.1.2.2 Testing Of Cross Validation Subset Data For Conjugate-Gradient Learning Method. As seen in Table 5.13, only the "low" attribute has the true information to be classified.

Table 5.15 Confusion matrix for conjugate-gradient learning method (cross validation subset)

Output / Desired	*CLSATTR(1)*	*CLSATTR(2)*	*CLSATTR(3)*
CLSATTR(1)	10	12	17
CLSATTR(2)	0	0	0
CLSATTR(3)	0	0	0

Table 5.16 Performance values for conjugate-gradient learning method (cross validation subset)

Performance	CLSATTR(1)	CLSATTR(2)	CLSATTR(3)
MSE	0,232225728	0,321375717	0,452186406
NMSE	1,217983907	1,508680451	1,838971986
MAE	0,236880376	0,3278437	0,45866939
Min Abs Error	0	0	0
Max Abs Error	1	1	1
r	-	-	-
Percent Correct	100	0	0

5.2.1.3 Levenberg-Marquardt Algorithm

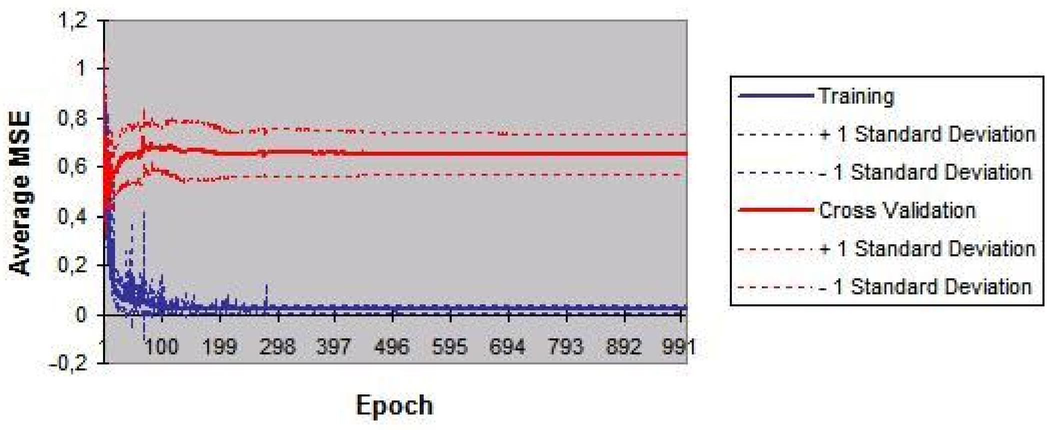

Figure 5.7 Average MSE for 5 runs

By using the Hessian matrix, Levenberg-Marquardt algorithm converges to the minimum in the first 100 epochs. Although the converging performance is awesome, the testing results show that this algortihm is not suitable for learning.

Table 5.17 MSE statistics for training and cross validation subsets.

All Runs	Training Minimum	Training Standard Deviation	Cross Validation Minimum	Cross Validation Standard Deviation
Average of Minimum MSEs	0.022446472	0.014444599	0.394822974	0.054634375
Average of Final MSEs	0.022446472	0.014444599	0.651251064	0.084005489

Levenberg-Marquardt algorithm produces very low values of average of minimum MSEs but cross validation subset values are rather high, showing that memorizing is a handicap for this algorithm.

Table 5.18 Best MSE for subsets

Best Networks	***Training***	***Cross Validation***
Run #	3	2
Epoch #	1000	5
Minimum MSE	0.009600004	0.350335159
Final MSE	0.009600004	0.757165837

As MSE values of training subset data converge to zero, MSE values of cross validation subset data increase as the epoch number gets bigger.

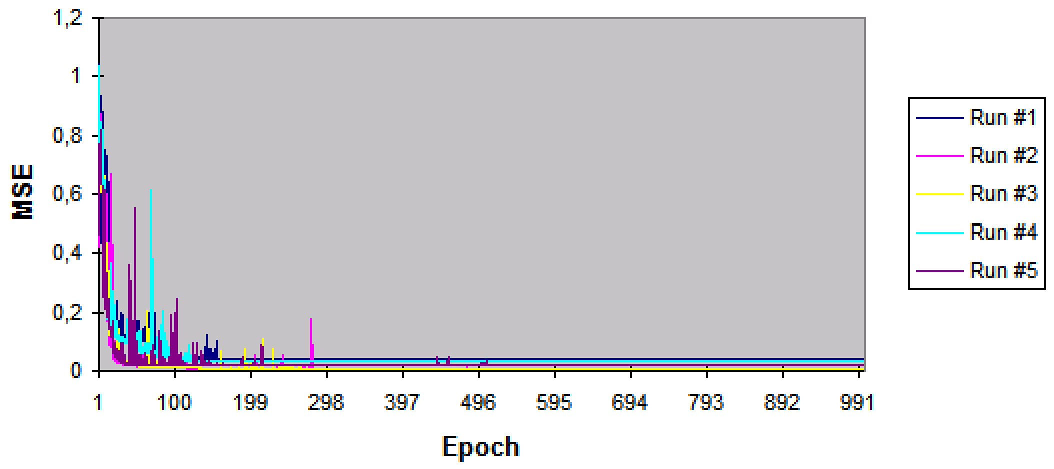

Figure 5.8 Training MSE for Levenberg-Marquardt algorithm (5 runs)

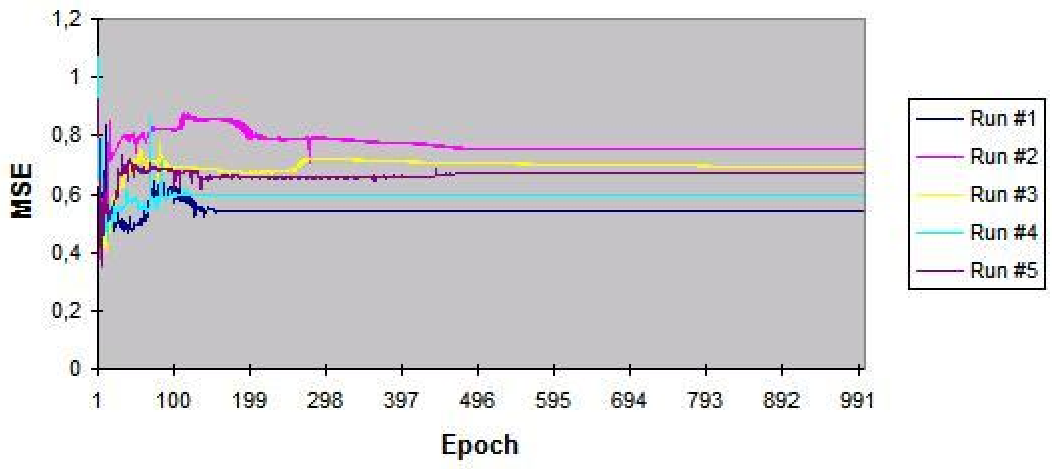

Figure 5.9 Cross validation MSE for Levenberg-Marquardt algorithm (5 runs)

Cross validation stops memorizing the predicted values between 150-400 epochs in Figure 5.9.

Table 5.19 Minimum MSE values for Levenberg-Marquardt algorithm

Subset	**Minimum for Run #1**	**Minimum for Run #2**	**Minimum for Run #3**	**Minimum for Run #4**	**Minimum for Run #5**
Training	0.04322379	0.01001963	0.0096	0.03074938	0.01863956
Cross validation	0.46639767	0.35033516	0.35944545	0.44110866	0.35682793

5.2.1.3.1 Testing The Training Subset Of Levenberg-Marquardt Algorithm

Table 5.20 Confusion matrix for Levenberg-Marquardt learning method

Output / Desired	*CLSATTR(1)*	*CLSATTR(2)*	*CLSATTR(3)*
CLSATTR(1)	19	28	27
CLSATTR(2)	0	0	0
CLSATTR(3)	0	0	1

Table 5.21 Performance values for Levenberg-Marquardt learning method

Performance	***CLSATTR(1)***	***CLSATTR(2)***	***CLSATTR(3)***
MSE	0.255209987	0.379171033	0.366399284
NMSE	1.349206932	1.620696855	1.566106362
MAE	0.258335537	0.382475643	0.369332358
Min Abs Error	0	0	0
Max Abs Error	1	1	1
r	-	-	-
Percent Correct	100	0	3.571428571

5.2.1.3.2 Testing Of Cross Validation Subset Data For Levenberg-Marquardt Method

Table 5.22 Confusion matrix for conjugate-gradient learning method

Output / Desired	*CLSATTR(1)*	*CLSATTR(2)*	*CLSATTR(3)*
CLSATTR(1)	9	12	17
CLSATTR(2)	0	0	0
CLSATTR(3)	1	0	0

Table 5.23 Performance values of cross validation subset data for levenberg-marquardt learning method

Performance	CLSATTR(1)	CLSATTR(2)	CLSATTR(3)
MSE	0.240779959	0.318548653	0.456010808
NMSE	1.262849371	1.495408952	1.854525236
MAE	0.246790633	0.324977649	0.461731447
Min Abs Error	0	0	0
Max Abs Error	1	1	1
r	-	-	-
Percent Correct	90	0	0

5.2.1.4 Accuracy Comparison Of Testing The Training And Cross Validation Subsets Of Learning Algorithms

Table 5.24 Mean accuracy performance of learning algorithms

Algorithm/ Subset	**Momentum**	**Conjugate-gradient**	**LM**
Training	56,77%	33,33%	34,52%
Cross validation	43.00%	33,33%	30%

As seen from the Table 5.24, the comparison is made on a simple multilayer perceptron that is having one hidden layer, an input and output layer. The network architecture is 56-4-3, that is the base design for the tae dataset. For the conjugate-gradient and Levenberg-Marquardt algorithms, there is not much difference between each other on the accuracy performance for the subsets.

Because the weight matrix of the hidden layer, for these two higher order gradient techniques, lacks of information, there is no enough prediction to classify the data. Backpropagation algorithm with momentum memorizes the values and makes updates that are suitable to converge to minimize the error function. As seen from Table 5.24, accuracy for both training and cross validation is a bit low.

As expected, the cross validation loses accuracy performance at about 25% compared to training. Conjugate-gradient and Levenberg-Marquardt algorithms make true classification approximately of 60% of momentum learning algortihm compared as the training subset data and approximately of 70% compared as the cross validation subset data.

5.2.2 The Optimum Momentum Rate For Tae Dataset

Table 5.25 Network parameters

Dataset used	**tae**
ANN Architecture	56-4-3
Number of epochs	500
Number of runs	5
Momentum Rate	0.7

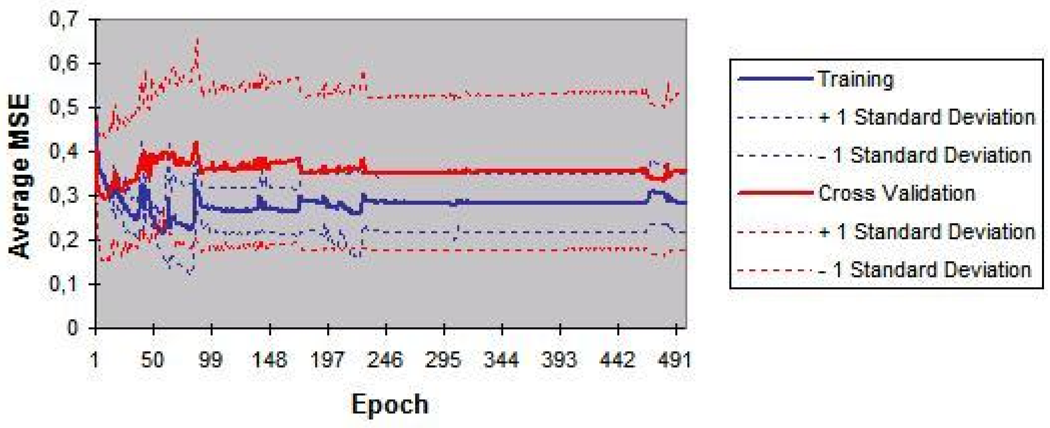

Figure 5.10 Average MSE

Starting from the first epoch, the learning process is active through the epoch number 83, and gets minimum at the value 0.1141442. As the weight matrix changes with the new network parameters, the average MSE jumps vertically.

Table 5.26 MSE statistics for training and cross validation subsets

All Runs	*Training Minimum*	*Training Standard Deviation*	*Cross Validation Minimum*	*Cross Validation Standard Deviation*
Average of Minimum MSEs	0.168046732	0.073360051	0.285392259	0.140346698
Average of Final MSEs	0.283385375	0.065373146	0.355764764	0.176579799

Table 5.27 Best MSE for subsets

Best Networks	*Training*	*Cross Validation*
Run #	3	1
Epoch #	83	13
Minimum MSE	0.1141442	0.034766387
Final MSE	0.231664072	0.050184

The optimum momentum rate found is between the possible momentum rates; 0.1, 0.3, 0.5, 0.7 and 0.9. As seen from Figure 5.11, the neural network makes much effort on training. The graph sometimes draws scatters moving down and then sudden upward movements shows the new weight introduced to the network. This is due to the momentum value.

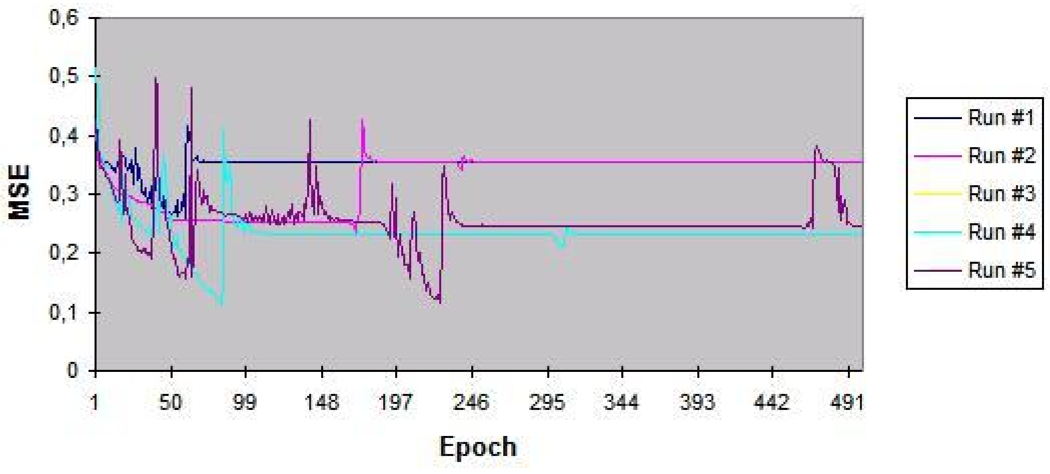

Figure 5.11 Training MSE

As seen from Figure 5.12, the training gets the MSE values decreased as the cross validation shows increase. The learning process moves around the value 0.25.

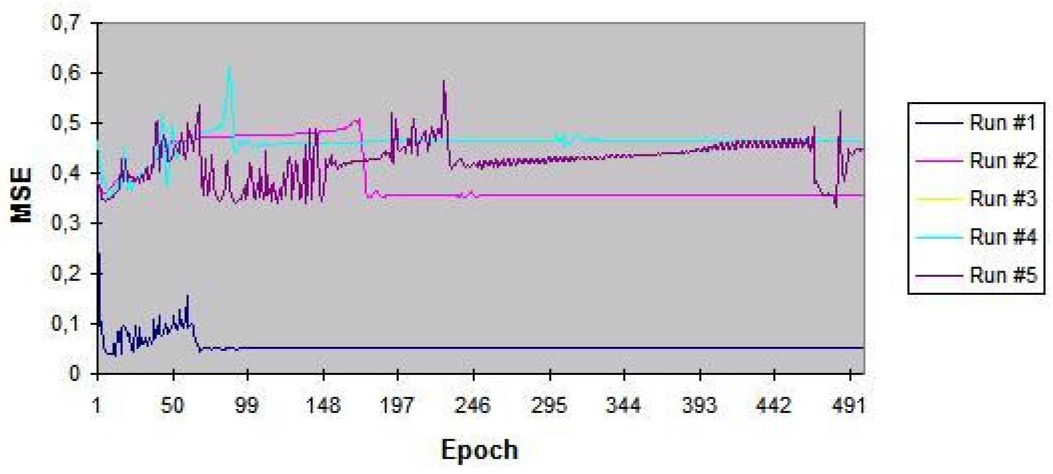

Figure 5.12 Cross validation MSE

Table 5.28 Minimum MSE values of one hidden layer, for momentum rate=0,7

Subset	Minimum for Run #1	Minimum for Run #2	Minimum for Run #3	Minimum for Run #4	Minimum for Run #5
Training	0.25976118	0.23598706	0.1141442	0.1141442	0.11619701
Cross validation	0.03476639	0.35046852	0.35387934	0.35387934	0.33396772

5.2.2.1 Testing Of Training Subset Data For Momentum Learning Method

Table 5.29 Testing the training subset of one hidden layer, momentum rate=0,7

Output / Desired	*CLSATTR(1)*	*CLSATTR(2)*	*CLSATTR(3)*
CLSATTR(1)	16	12	10
CLSATTR(2)	3	16	10
CLSATTR(3)	0	0	8

Table 5.30 Performance values for the training subset of one hidden layer, momentum rate=0,7

Performance	***CLSATTR(1)***	***CLSATTR(2)***	***CLSATTR(3)***
MSE	0.157552141	0.235002837	0.234861174
NMSE	0.83292368	1.004476411	1.003870897
MAE	0.374051306	0.470804544	0.469789571
Min Abs Error	0.178215631	0.359607698	0.350010665
Max Abs Error	0.690756763	0.634292157	0.647414842
r	0.551614941	0.217434096	0.296913688
Percent Correct	84.21052632	57.14285714	28.57142857

5.2.2.2 Testing Of Cross Validation Subset Data For Momentum Learning Method

Table 5.31 Testing the cross validation subset of one hidden layer, momentum rate=0,7

Output / Desired	*CLSATTR(1)*	*CLSATTR(2)*	*CLSATTR(3)*
CLSATTR(1)	6	9	9
CLSATTR(2)	4	0	5
CLSATTR(3)	0	3	3

Table 5.32 Performance values of testing the cross validation subset of one hidden layer, momentum rate=0,7

Performance	***CLSATTR(1)***	***CLSATTR(2)***	***CLSATTR(3)***
MSE	0.220191967	0.222488954	0.256012147
NMSE	1.154868902	1.044462033	1.041161698
MAE	0.441688976	0.460858301	0.493555038
Min Abs Error	0.178215631	0.360154196	0.352603501
Max Abs Error	0.783466027	0.638657773	0.649129907
r	0.050822935	-0.020310868	0.100077017
Percent Correct	60	0	17.64705882

Table 5.33 Accuracy performance of momentum rates

Momentum value/ Subset	β=**0.1**	β=**0.3**	β=**0.5**	β=**0.7**	β=**0.9**
Training	58.15%	38.1%	51.63%	62,65%	61.40%
Cross validation	35.2%	40.2%	39.35%	34.05%	41.1%

According to the Table 5.33, there is no slight difference of accuracy performance between the momentum rates. For that reason, the statistical results of average minimum MSE values becomes the selective criterion for performance. As the minimum MSE statistics and also the highest accuracy for the training subset data, are for β=0.7, presented above.

5.2.3 The Optimum Hidden Layer Size

In order to construct the ANN needed, the optimum hidden layer size should be defined. Starting from the one hidden layer, the hidden layer size is increased up to five hiden layers and each hadden layer sizes' performances are compared. For each hidden layer size performance evaluation, 5 runs made and 1000 epochs generated per run.The momentum rate is 0.7 as found optimum before.

5.2.3.1 One Hidden Layered ANN

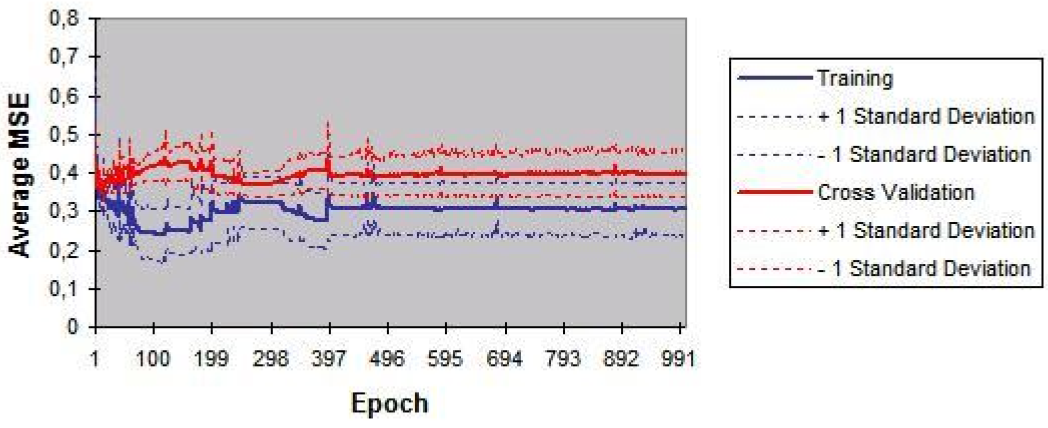

Figure 5.13 Average MSE for one hidden layer (tae) dataset

Training values wander around the value, 0.3. Training stops about the 400 epochs. Because the number of exemplars of *tae* dataset is not much, the subsets including both the training and cross validation do not give enough ability to overfitting.

Table 5.34 MSE statistics for training and cross validation subsets of tae dataset (One hidden layer)

All Runs	***Training Minimum***	***Training Standard Deviation***	***Cross Validation Minimum***	***Cross Validation Standard Deviation***
Average of Minimum MSEs	0.184258716	0.020733666	0.348079253	0.004354931
Average of Final MSEs	0.305357202	0.067724117	0.39908647	0.060181746

Table 5.35 Best MSE for subsets

Best Networks	*Training*	*Cross Validation*
Run #	4	4
Epoch #	117	14
Minimum MSE	0.170868494	0.340908419
Final MSE	0.354815721	0.3552145

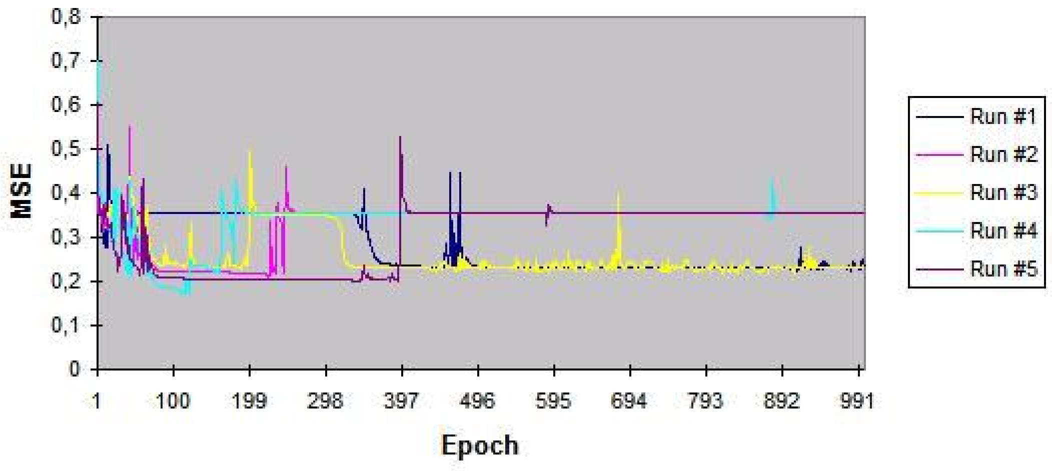

Figure 5.14 Training MSE for one hidden layer (tae dataset)

Run number 2 and run number 5 show the same characteristics during the training process.

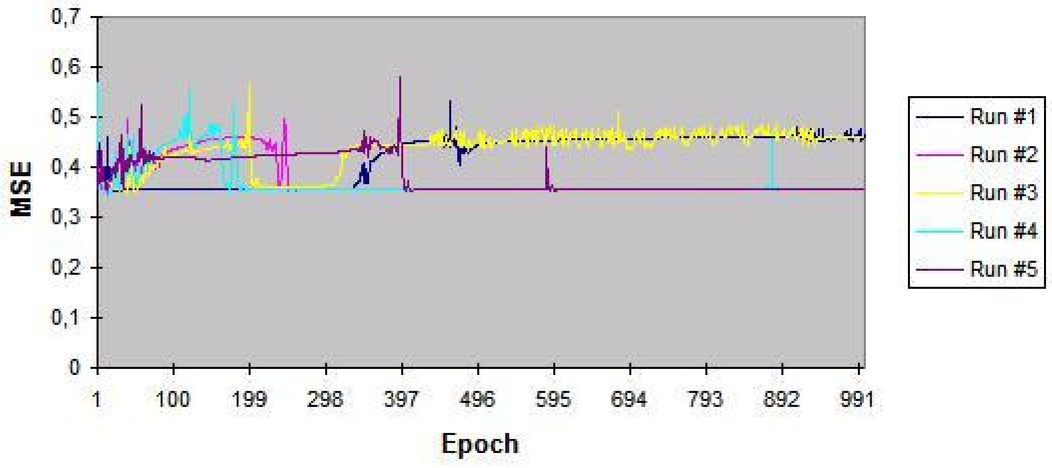

Figure 5.15 Cross validation MSE for one hidden layer (tae dataset)

Table 5.36 Minimum MSE values of subsets for each run of one hidden layer size ANN topology

Subset	**Minimum for Run #1**	**Minimum for Run #2**	**Minimum for Run #3**	**Minimum for Run #4**	**Minimum for Run #5**
Training	0.2230688	0.20598448	0.21977485	0.17086849	0.20159696
Cross validation	0.35125135	0.34830631	0.34806584	0.34090842	0.35186434

5.2.3.1.1 Testing The Training Subset Of One Hidden Layer For Momentum Learning Algorithm

Table 5.37 Confusion matrix for the training subset of one hidden layer, momentum rate=0,7

Output / Desired	*CLSATTR(1)*	*CLSATTR(2)*	*CLSATTR(3)*
CLSATTR(1)	14	8	2
CLSATTR(2)	3	8	2
CLSATTR(3)	2	12	24

Table 5.38 Performance values for the training subset of one hidden layer, momentum rate=0,7

Performance	***CLSATTR(1)***	***CLSATTR(2)***	***CLSATTR(3)***
MSE	0.162748993	0.237032447	0.173933401
NMSE	0.860397639	1.013151607	0.743446338
MAE	0.36149553	0.466413639	0.384538476
Min Abs Error	0.172563086	0.344181761	0.185617497
Max Abs Error	0.789697678	0.655026087	0.814401274
r	0.510759319	-0.100723691	0.57358051
Percent Correct	73.68421053	28.57142857	85.71428571

5.2.3.1.2 Testing The Cross Validation Subset Of One Hidden Layer For Momentum Learning Algorithm

Table 5.39 Confusion matrix for testing the cross validation subset of one hidden layer, momentum rate=0,7

Output / Desired	*CLSATTR(1)*	*CLSATTR(2)*	*CLSATTR(3)*
CLSATTR(1)	7	6	3
CLSATTR(2)	7	1	0
CLSATTR(3)	6	3	4

Table 5.40 Performance values for the cross validation subset of one hidden layer, momentum rate=0,7

Performance	CLSATTR(1)	CLSATTR(2)	CLSATTR(3)
MSE	0.300989046	0.211372444	0.170408388
NMSE	1.21192354	1.071736577	1.110900396
MAE	0.509767486	0.443608536	0.374476393
Min Abs Error	0.173550906	0.344966607	0.185995391
Max Abs Error	0.820143485	0.655463878	0.808294716
r	0.085404164	-0.061327255	0.223502109
Percent Correct	35	10	57.14285714

Table 5.41 Mean accuracy performance of hidden layers

Hidden layer size/ Subset	Hidden Layer=1	Hidden Layer=2	Hidden Layer=3	Hidden Layer=4	Hidden Layer=5
Training	62.65%	59.02%	61.90%	59.52%	57.14%
Cross validation	34.05%	43.33%	34.64%	34.53%	36.59%

Mean accuracy values are shown in Table 5.41 which include the performance values. The true classified exemplars are gathered on the orthogonal of the confusion matrix which are named as the true classification.

As seen from Table 5.41, the neural network structure with one hidden layer and with three hidden layers behave nearly the same. They have similar mean accuracy performance values. But due to the calculations needed and the time required, a simpler neural network model is preferred here. One hidden layered neural network for *tae* dataset fits the best to classify.

5.2.4 Optimum Number Of Processing Elements Of The Hidden Layer

A network with too few hidden units is often not able to learn well enough, a network with too many hidden units is not able to generalise well enough. When teaching a network to classify data, the key is to choose an appropriate number of hidden units. Not too many degrees of freedom should be given through the network.

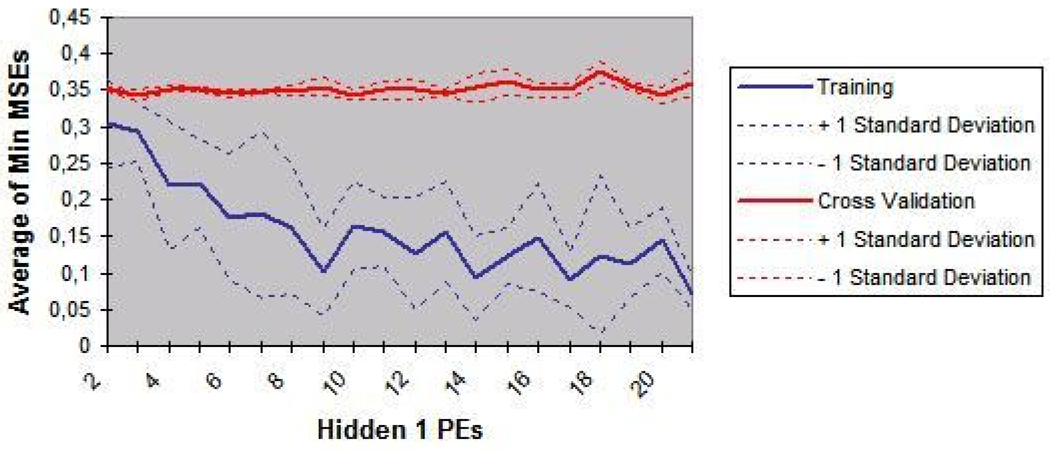

Figure 5.16 Average values of minimum MSEs for each scenario of number of units in the hidden layer. (tae dataset)

MSE values of cross validation are important for the performance of the network training. For *tae* dataset; because the classification is made between three class attributes (low, medium and high), the calculations for the number of hidden units in hidden 1 layer started from three to unlimited size. Incremental algorithm is applied to the hidden 1 layer. A loop is implemented to the weight matrix in the hidden 1 layer, so that; when the error in the cross validation minimizes, the minimal point is where the number of hidden units in the hidden 1 layer should be chosen.

As seen from Figure 5.16, the average of minimum MSE values for different number of hidden units (processing elements) do not change up to the fourteen number of hidden units in hidden layer 1. From Figure 5.16, one can argue that the minimum average MSE values comes to minimal values through 18 to 20 number of hidden units but cross validation values start to increase. For that reason, fourteen number of hidden units for hidden 1 layer is chosen for optimal. So, the network architecture for the minimum error of average MSE of cross validation is 56-14-3 architecture.

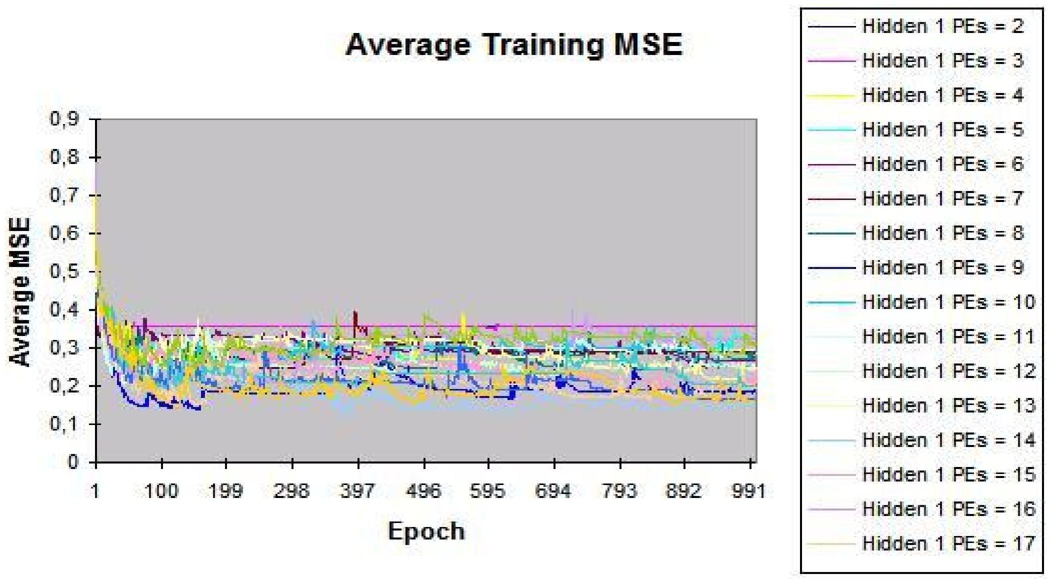

Figure 5.17 Average training MSE values for each processing element scenario

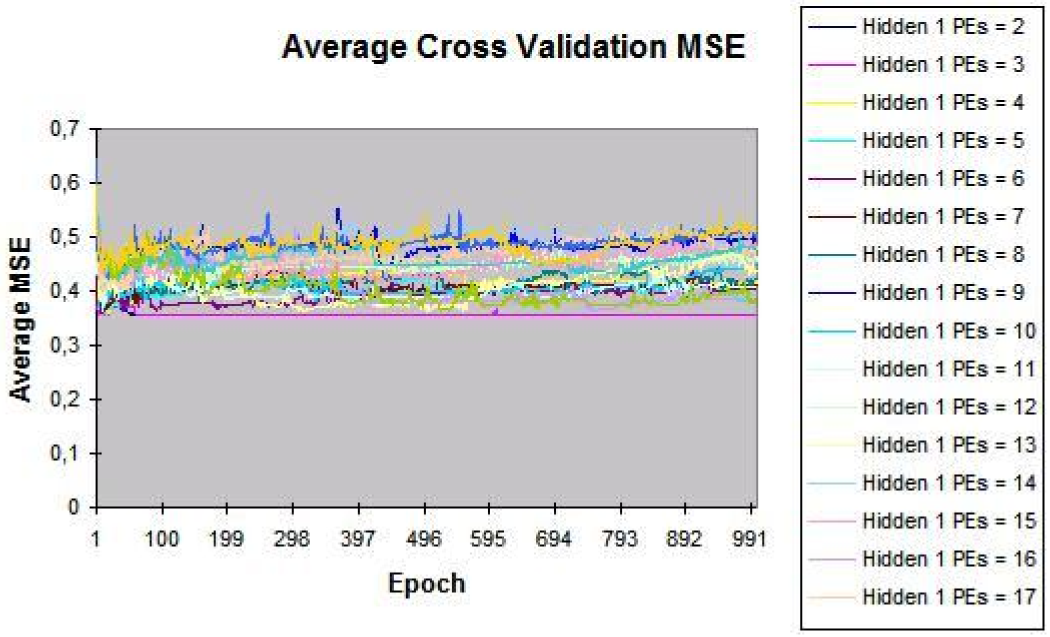

Figure 5.18 Average cross validation MSE values for each processing element scenario

Table 5.42 Minimum MSE values for subsets of 14 hidden processing elements

Subset	**Minimum for Run #1**	**Minimum for Run #2**	**Minimum for Run #3**	**Minimum for Run #4**	**Minimum for Run #5**
Training	0.11225502	0.14273578	0.08877673	0.09465382	0.17952491
Cross validation	0.35260078	0.39015894	0.36047447	0.35982629	0.34476902

5.3 Vehicle Silhouette Data Set

5.3.1 Statlog Vehicle Silhoutte (veh) Database

The purpose is to classify a given silhouette as one of four types of vehicle, using a set of features extracted from the silhouette. The vehicle may be viewed from one of many different angles. The representative dataset matrix is shown in Appendix 2.

5.3.2 Data Set Information

This data was originally gathered at the TI in 1986-87 by JP Siebert. It was partially financed by Barr and Stroud Ltd. The original purpose was to find a method of distinguishing 3D objects within a 2D image by application of an ensemble of shape feature extractors to the 2D silhouettes of the objects. Measures of shape features extracted from example silhouettes of objects to be discriminated were used to generate a classification rule tree by means of computer induction.

This object recognition strategy was successfully used to discriminate between silhouettes of model cars, vans and buses viewed from constrained elevation but all angles of rotation.

5.3.3 Dataset Description

The features were extracted from the silhouettes by the HIPS (Hierarchical Image Processing System) extension BINATTS, which extracts a combination of scale independent features utilising both classical moment based measures such as scaled variance, skewness and kurtosis about the major/minor axes and heuristic measures such as hollows, circularity, rectangularity and compactness.

Four "Corgie" model vehicles were used for the experiment: a double decker bus, Cheverolet van, Saab 9000 and an Opel Manta 400. This particular combination of vehicles was chosen with the expectation that the bus, van and either one of the cars would be readily distinguishable, but it would be more difficult to distinguish between the cars.

The images were acquired by a camera looking downwards at the model vehicle from a fixed angle of elevation (34.2 degrees to the horizontal). The vehicles were placed on a diffuse backlit surface (lightbox). The vehicles were painted matte black to minimise highlights. The images were captured using a CRS4000 framestore connected to a vax 750. All images were captured with a spatial resolution of 128x128 pixels quantised to 64 greylevels. These images were thresholded to produce binary vehicle silhouettes, negated (to comply with the processing requirements of BINATTS) and thereafter subjected to shrink-expand-expand-shrink HIPS modules to remove "salt and pepper" image noise.

The vehicles were rotated and their angle of orientation was measured using a radial graticule beneath the vehicle. 0 and 180 degrees corresponded to "head on" and "rear" views respectively while 90 and 270 corresponded to profiles in opposite directions. Two sets of 60 images, each set covering a full 360 degree rotation, were captured for each vehicle. The vehicle was rotated by a fixed angle between images. These datasets are known as e2 and e3 respectively.

A further two sets of images, e4 and e5, were captured with the camera at elevations of 37.5 degs and 30.8 degs respectively. These sets also contain 60 images per vehicle apart from e4.van which contains only 46 owing to the difficulty of containing the van in the image at some orientations.

5.3.4 Attribute Information

This dataset has eighteen attributes and all these attributes are numerical that are opposite to the *tae* dataset. The atrributes are:

1-*Compactness* =(average perim)^{**2} / area
2-*Cırcularıty* =(average radius)^{**2} / area
3-*Dıstance Cırcularıty* = area / (av.distance from border)^{**2}
4-*Radıus Ratıo* = (maximum radius-minimum radius) / average radius
5-*Pr.Axıs Aspect Ratıo* = (minor axis) /(major axis)
6-*Max.Length Aspect Ratıo* = (length perp. max length)/(max length)
7- *Scatter Ratıo* = (inertia about minor axis) / (inertia about major axis)

8- *Elongatedness* = area / (shrink width)^{**2}

9- *Pr.Axıs Rectangularıty* = area / (pr.axis length*pr.axis width)

10- *Max.Length Rectangularıty* = area / (max.length*length perp. to this)

11- *Scaled Varıance Along Major Axıs* = (2nd order moment about minor axis) /area

12- *Scaled Varıance Along Mınor Axıs* = (2nd order moment about major axis) / area

13- *Scaled Radıus Of Gyration* = (mavar+mivar) / area

14- *Skewness About Major Axis* = (3rd order moment about major axis)/sigma_min^{**3}

15- *Skewness About Mınor Axıs* = (3rd order moment about minor axis) / sigma_maj^{**3}

16- *Kurtosıs About Mınor Axıs* = (4th order moment about major axis) / sigma_min^{**4}

17- *Kurtosıs About Major Axıs* = (4th order moment about minor axis) / sigma_maj^{**4}

18- *Hollows Ratıo* = (area of hollows) / (area of bounding polygon)

where sigma_maj^{**2} is the variance along the major axis and sigma_min^{**2} is the variance along the minor axis, and

area of hollows= area of bounding poly - area of object

The area of the bounding polygon is found as a side result of the computation to find the maximum length. Each individual length computation yields a pair of calipers to the object orientated at every 5 degrees. The object is propagated into an image containing the union of these calipers to obtain an image of the bounding polygon.

In this dataset, a total of 846 exemplars exist. 423 exemplars are obtained for training subset, 212 exemplars for cross validation and 211 exemplars for testing subset.

5.3.5 The Comparison Of Learning Algorithms Due To ANN Structure

According to the general classification ANN, one hidden layered MLP is enough. Primarily, the selection of the learning algorithm is important. Three learning algorithms are taken into consideration:

- Momentum learning
- Conjugate-gradient
- Levenberg-Marquant

To compare the algorithms, one hidden layered ANN is constructed basically. The aim is to search for the answer which algorithm causes the minimum average MSE value.

5.3.5.1 Momentum Learning

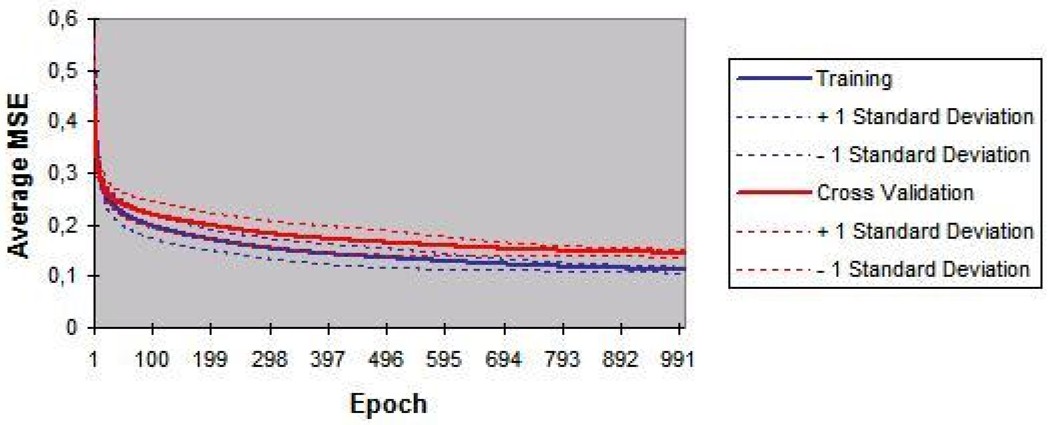

Figure 5.19 Average MSE with Standard Deviation Boundaries for 5 Runs

Because of the number of numerical attributes is high such as 18, presented in the *veh* database, the learning process cannot be easily seen. There is a slow convergence to minima.

Examining the figures 5.20 and 5.21, the training and cross validation sets behave in the same way. That means, independently, cross validation is trained in a good manner. So, high values of mean accuracy is expected for classification.

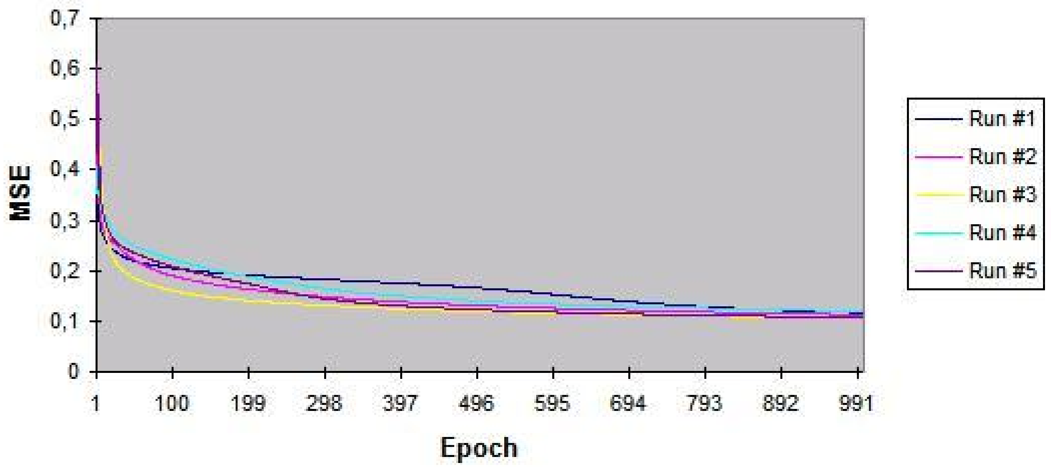

Figure 5.20 Training MSE for momentum learning

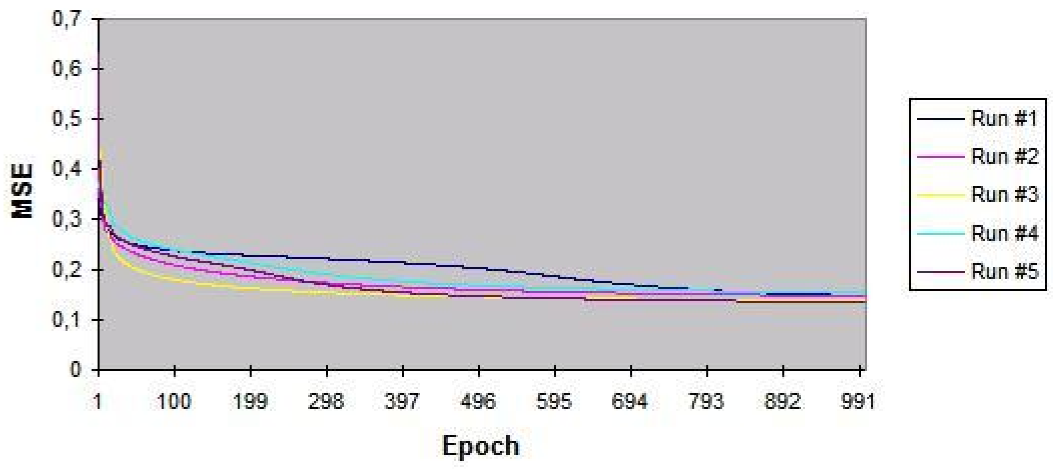

Figure 5.21 Cross validation MSE for momentum learning

Table 5.43 MSE statistics for training and cross validation subsets of veh dataset (One hidden layer)

All Runs	*Training Minimum*	*Training Standard Deviation*	*Cross Validation Minimum*	*Cross Validation Standard Deviation*
Average of Minimum MSEs	0.112427005	0.005874629	0.145232203	0.007579711
Average of Final MSEs	0.112427005	0.005874629	0.145232203	0.007579711

Momentum learning produces statistically consistent values such as low level standard deviation and mean compared the two other learning algorithms. MSE values of both training and cross validation converge to average minimum by means of low standard deviation.

Table 5.44 Best MSE values for subsets

Best Networks	***Training***	***Cross Validation***
Run #	3	5
Epoch #	1000	1000
Minimum MSE	0.10643069	0.136003509
Final MSE	0.10643069	0.136003509

Table 5.45 Minimum MSE values for momentum learning

Subset	**Minimum for Run #1**	**Minimum for Run #2**	**Minimum for Run #3**	**Minimum for Run #4**	**Minimum for Run #5**
Training	0.1166005	0.11236181	0.10643069	0.11980858	0.10693344
Cross validation	0.14835984	0.14816958	0.13902865	0.15459943	0.13600351

5.3.5.1.1 Testing The Training Subset Of One Hidden Layer For Momentum Learning Algorithm. As the number of true classified exemplars increases, *r* values and correct percent values start to increase, as well.

Table 5.46 Confusion matrix for testing of training subset data for momentum learning method

Output / Desired	*VEHCL(opel)*	*VEHCL(bus)*	*VEHCL(saab)*	*VEHCL(van)*
VEHCL(opel)	75	1	32	0
VEHCL(bus)	2	118	3	1
VEHCL(saab)	19	1	63	0
VEHCL(van)	2	1	1	104

Table 5.47 Performance values for testing of training subset data for momentum learning method

Performance	***VEHCL(opel)***	***VEHCL(bus)***	***VEHCL(saab)***	***VEHCL(van)***
MSE	0.093429388	0.021114498	0.092435561	0.008568728
NMSE	0.524873691	0.103387775	0.515631701	0.04591776
MAE	0.202379966	0.073715939	0.198357207	0.057388675
Min Abs Error	2.2378E-05	9.35122E-05	1.21128E-05	0.000717191
Max Abs Error	1.007517011	0.962872018	1.041621548	0.780103013
r	0.697593828	0.94862126	0.697551594	0.980074652
Percent Correct	76.53061224	97.52066116	63.63636364	99.04761905

5.3.5.1.2 Testing The Cross Validation Subset Of One Hidden Layer For Momentum Learning Algorithm. *VEHCL(opel)* and *VEHCL(saab)* attributes have more misclassified exemplars. In that case, the MSE values for these attributes are expected to be high, and opposite of that situaiton, the *r* and percent correct values to below.

Table 5.48 Confusion Matrix for testing of cross validation subset data for momentum learning

Output / Desired	*VEHCL(opel)*	*VEHCL(bus)*	*VEHCL(saab)*	*VEHCL(van)*
VEHCL(opel)	41	0	20	1
VEHCL(bus)	4	42	1	1
VEHCL(saab)	20	2	33	2
VEHCL(van)	2	0	2	41

Table 5.49 Performance values

Performance	***VEHCL(opel)***	***VEHCL(bus)***	***VEHCL(saab)***	***VEHCL(van)***
MSE	0.147398289	0.030277335	0.122852711	0.036700385
NMSE	0.681901049	0.184088819	0.632038946	0.219489303
MAE	0.28725699	0.093958983	0.277117391	0.086943913
Min Abs Error	0.000329809	0.000155947	0.000311987	0.00094469
Max Abs Error	0.953404705	0.908973466	0.812204558	0.939407299
r	0.565119798	0.906119231	0.616379211	0.890464251
Percent Correct	61.19402985	95.45454545	58.92857143	91.11111111

5.3.5.2 Conjugate-Gradient Learning Method

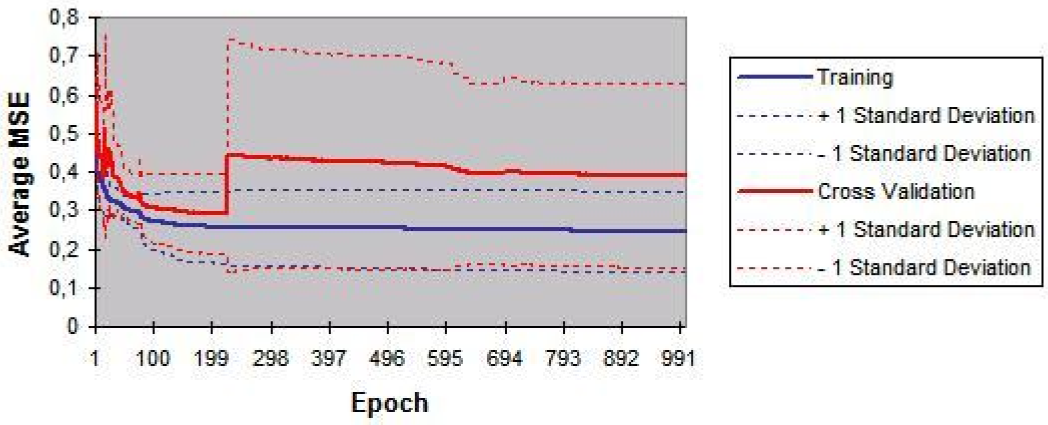

Figure 5.22 Average MSE for 5 runs

As seen from Figure 5.24, cross validation subset data of run number 2 moves vertically and gets higher MSE values and that causes an increment seen in average MSEs.

Table 5.50 MSE statistics for training and cross validation subsets for veh dataset

All Runs	***Training Minimum***	***Training Standard Deviation***	***Cross Validation Minimum***	***Cross Validation Standard Deviation***
Average of Minimum MSEs	0.242653495	0.103842625	0.267493606	0.096332087
Average of Final MSEs	0.242653495	0.103842625	0.389636393	0.238195555

Table 5.51 Best MSE values

Best Networks	***Training***	***Cross Validation***
Run #	5	5
Epoch #	1000	882
Minimum MSE	0.097817965	0.139471865
Final MSE	0.097817965	0.140007674

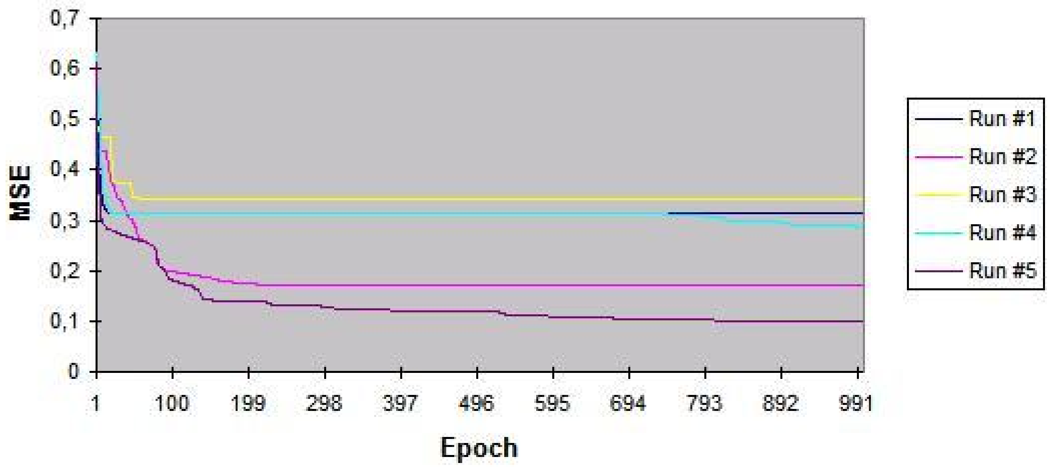

Figure 5.23 Training MSE

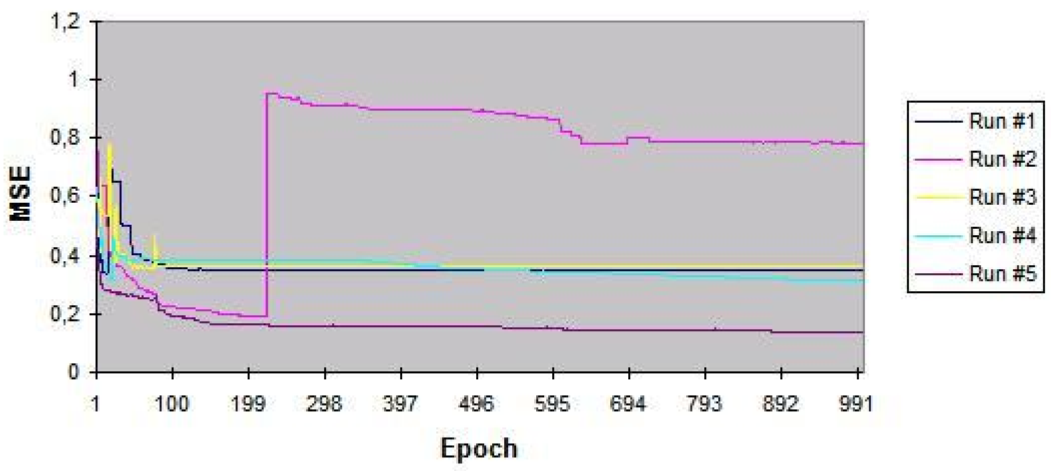

Figure 5.24 Cross validation MSE

Table 5.52 Minimum MSE values for conjugate-gradient learning method

Subset	Minimum for Run #1	Minimum for Run #2	Minimum for Run #3	Minimum for Run #4	Minimum for Run #5
Training	0.31355234	0.17124406	0.34241099	0.28824211	0.09781796
Cross validation	0.33636168	0.1908967	0.35665164	0.31408615	0.13947187

5.3.5.2.1 Testing Of Training Subset Data For Conjugate-Gradient_Learning Method

Table 5.53 Confusion Matrix for testing of training subset data for conjugate-gradient_learning method

Output / Desired	*VEHCL(opel)*	*VEHCL(bus)*	*VEHCL(saab)*	*VEHCL(van)*
VEHCL(opel)	67	1	27	0
VEHCL(bus)	2	117	4	0
VEHCL(saab)	29	0	68	0
VEHCL(van)	0	3	0	105

Table 5.54 Performance values

Performance	***VEHCL(opel)***	***VEHCL(bus)***	***VEHCL(saab)***	***VEHCL(van)***
MSE	0.108138256	0.024754818	0.099318379	0.010996027
NMSE	0.607506121	0.121212708	0.554026009	0.058925073
MAE	0.223696678	0.082022537	0.208532684	0.054647872
Min Abs Error	0.000306308	0.000139652	0.000151154	4.35091E-05
Max Abs Error	0.975661846	1.053480303	1.003342612	1.055555089
r	0.627401776	0.941033412	0.669451564	0.973740456
Percent Correct	68.36734694	96.69421488	68.68686869	100

5.3.5.2.2 Testing Of Cross Validation Subset Data For Conjugate-Gradient Learning Method

Table 5.55 Confusion matrix for testing of cross validation subset data for momentum learning

Output / Desired	*VEHCL(opel)*	*VEHCL(bus)*	*VEHCL(saab)*	*VEHCL(van)*
VEHCL(opel)	41	0	19	1
VEHCL(bus)	3	44	1	1
VEHCL(saab)	21	0	36	3
VEHCL(van)	2	0	0	40

Table 5.56 Performance values

Performance	***VEHCL(opel)***	***VEHCL(bus)***	***VEHCL(saab)***	***VEHCL(van)***
MSE	0.158041189	0.02302683	0.12416261	0.040584113
NMSE	0.731137747	0.140005122	0.638777969	0.24271622
MAE	0.29150031	0.08269755	0.251839916	0.09982497
Min Abs Error	0.000675544	0.000701821	0.00097184	0.00157529
Max Abs Error	0.951789706	0.88237859	0.943830165	0.975203154
r	0.528481975	0.931759244	0.61042324	0.882024132
Percent Correct	61.19402985	100	64.28571429	88.88888889

5.3.5.3 Levenberg-Marquardt Algorithm

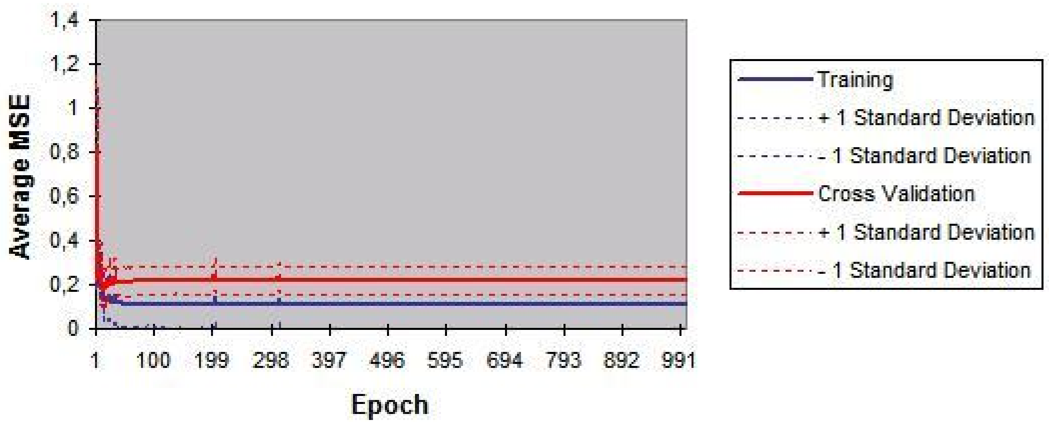

Figure 5.25 Average MSE for 5 runs

Table 5.57 MSE statistics for training and cross validation subsets for veh dataset

All Runs	*Training Minimum*	*Training Standard Deviation*	*Cross Validation Minimum*	*Cross Validation Standard Deviation*
Average of Minimum MSEs	0.1092397	0.108394453	0.153391457	0.047318505
Average of Final MSEs	0.1092397	0.108394453	0.218606887	0.062058229

Levenberg-Marquardt algorithm converges so rapidly. In about 40 epochs, the average MSE comes to 0.15 value levels. Levenberg-Marquardt algorithm is more successful if compared to conjugate-gradient algorithm. The standard deviation and the average of minimum MSEs of Levenberg-Marquardt are nearly half the value of standard deviation of conjugate-gradient algorithm.

Table 5.58 Best MSE values

Best Networks	*Training*	*Cross Validation*
Run #	5	5
Epoch #	304	14
Minimum MSE	0.057162131	0.125025029
Final MSE	0.057162131	0.189835384

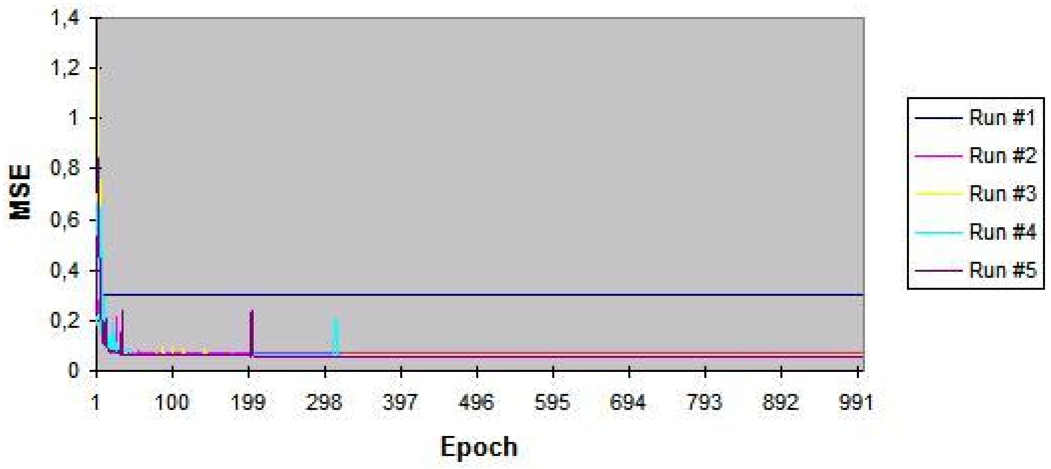

Figure 5.26 Training MSE for Levenberg-Marquardt algorithm (5 runs)

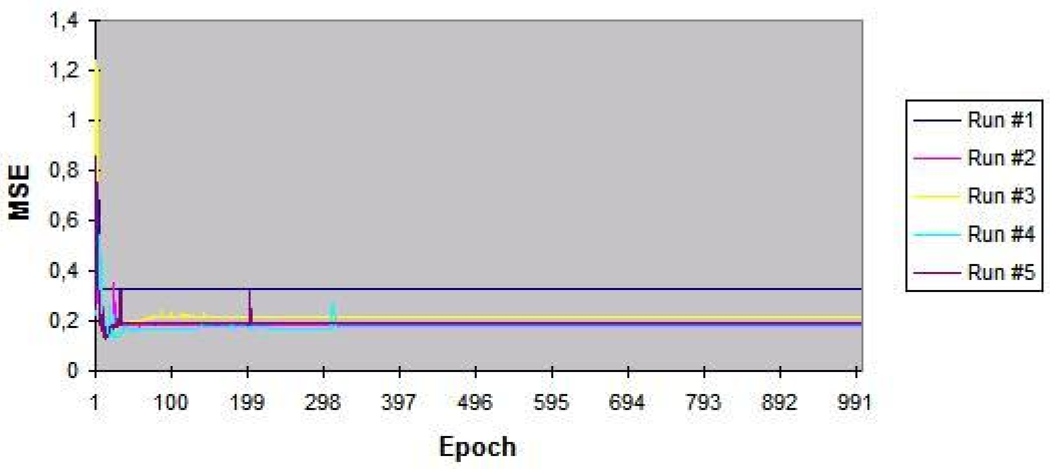

Figure 5.27 Cross validation MSE for Levenberg-Marquardt algorithm (5 runs)

Table 5.59 Minimum MSE values for Levenberg-Marquardt algorithm

Subset	Minimum for Run #1	Minimum for Run #2	Minimum for Run #3	Minimum for Run #4	Minimum for Run #5
Training	0.30298325	0.06811015	0.06026892	0.05767405	0.05716213
Cross validation	0.23715843	0.14284108	0.12800621	0.13392654	0.12502503

5.3.5.3.1 Testing The Training Subset Data Of Levenberg-Marquardt Algorithm

Table 5.60 Confusion matrix for testing of training subset data for Levenberg-Marquardt algorithm

Output / Desired	*VEHCL(opel)*	*VEHCL(bus)*	*VEHCL(saab)*	*VEHCL(van)*
VEHCL(opel)	80	1	45	1
VEHCL(bus)	1	119	7	0
VEHCL(saab)	16	0	47	0
VEHCL(van)	1	1	0	104

Table 5.61 Performance values

Performance	***VEHCL(opel)***	***VEHCL(bus)***	***VEHCL(saab)***	***VEHCL(van)***
MSE	0.096826771	0.02321515	0.101048921	0.006477804
NMSE	0.543959729	0.113673681	0.563679461	0.034712999
MAE	0.213487683	0.072859601	0.208991081	0.044667627
Min Abs Error	0.000282074	4.71014E-05	8.34322E-05	3.69534E-05
Max Abs Error	0.940714589	0.986980909	1.033269704	0.667709778
r	0.67896186	0.943890477	0.684888374	0.985941769
Percent Correct	81.63265306	98.34710744	47.47474747	99.04761905

5.3.5.3.2 Testing Of Cross Validation Subset Data For Levenberg-Marquardt Learning Method

Table 5.62 MSE statistics for training and cross validation subsets for veh dataset

Output / Desired	*VEHCL(opel)*	*VEHCL(bus)*	*VEHCL(saab)*	*VEHCL(van)*
VEHCL(opel)	52	1	24	2
VEHCL(bus)	2	43	1	2
VEHCL(saab)	11	0	31	1
VEHCL(van)	2	0	0	40

Table 5.63 Performance values

Performance	***VEHCL(opel)***	***VEHCL(bus)***	***VEHCL(saab)***	***VEHCL(van)***
MSE	0.134552119	0.024971264	0.117782757	0.032705684
NMSE	0.622471479	0.151827446	0.605955613	0.195598705
MAE	0.275709599	0.07824892	0.246527429	0.076094073
Min Abs Error	0.000154448	0.000503627	0.001180166	1.63692E-06
Max Abs Error	0.987732991	0.940266699	0.935175262	1.019431174
r	0.616837365	0.926676438	0.637887897	0.909338418
Percent Correct	77.6119403	97.72727273	55.35714286	88.88888889

5.3.5.4 Comparison For Mean Accuracy Testing The Training And Cross Validation Subset Data Of Learning Algorithms

Table 5.64 Mean accuracy performance of learning algorithms

Algorithm/ Subset	**Momentum**	**Conjugate-gradient**	**LM**
Training	84.18%	83.44%	81.63%
Cross validation	76.67%	78.59%	79.9%

“Percent correct” row of each performance value table is handled as: Table 5.47 for training of momentum algortihm, Table 5.49 for cross validation of momentum algorithm, Table 5.54 for training of conjugate-gradient algorithm, Table 5.56 for cross validation of conjugate-gradient algorithm, Table 5.61 for training of Levenberg-Marquardt algorithm and Table 5.63 for cross validation of Levenberg-Marquardt algorithm.

Mean accuracy performance values for the numerical driven attributes are nearly the same. But, looking at the MSE statistics of the learning algortihms, the momentum learning and Levenberg-Marquardt algorithm have much less minimum average MSE values. Contrary to this situation, conjugate-gradient algorithm has disadvantages to converge to the average minimum. At Figure 5.24, cross validation MSE values started to increase vertically because of the wrong gradient direction.

5.3.6 The Optimum Momentum Rate

Table 5.65 Network parameters for the optimum momentum rate

Dataset used	**veh**
ANN Architecture	18-4-4
Number of epochs	1000
Number of runs	5
Momentum Rate	0.9

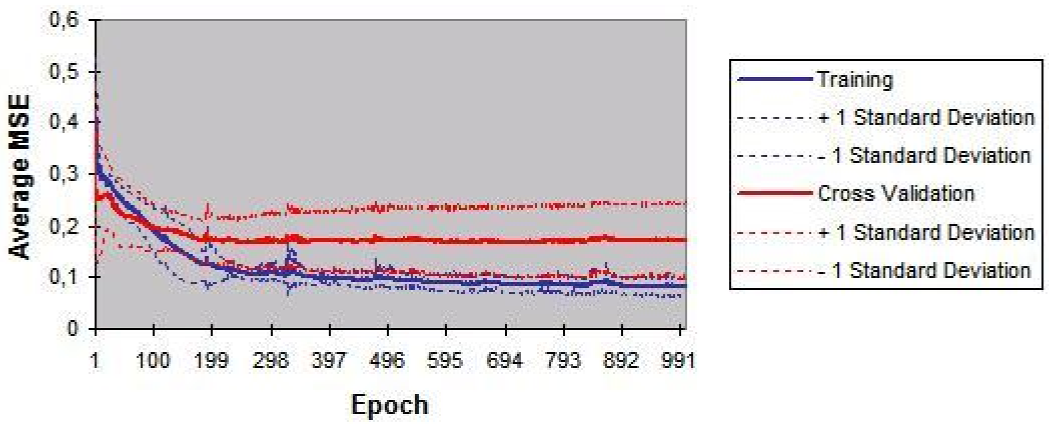

Figure 5.28 Average MSE

The optimum momentum rate found is among the possible momentum rates; 0.1, 0.3, 0.5, 0.7 and 0.9. Examining the tables mentioned; the minimum of average of minimum MSEs is 0.0794 for β=0.9. So, β=0.9 is chosen as the optimum momentum rate.

Table 5.66 MSE statisitcs for the optimum momentum rate

All Runs	***Training Minimum***	***Training Standard Deviation***	***Cross Validation Minimum***	***Cross Validation Standard Deviation***
Average of Minimum MSEs	0.079449192	0.017419894	0.109239352	0.04645294
Average of Final MSEs	0.079903202	0.018010406	0.171239116	0.071065264

Table 5.67 Best MSE values

Best Networks	***Training***	***Cross Validation***
Run #	3	1
Epoch #	1000	3
Minimum MSE	0.057525099	0.028554256
Final MSE	0.057525099	0.296527252

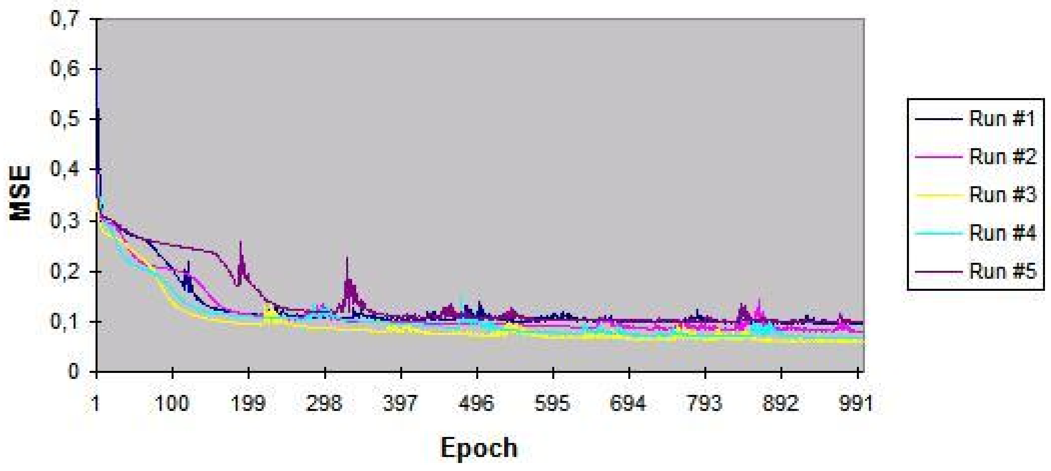

Figure 5.29 Training MSE

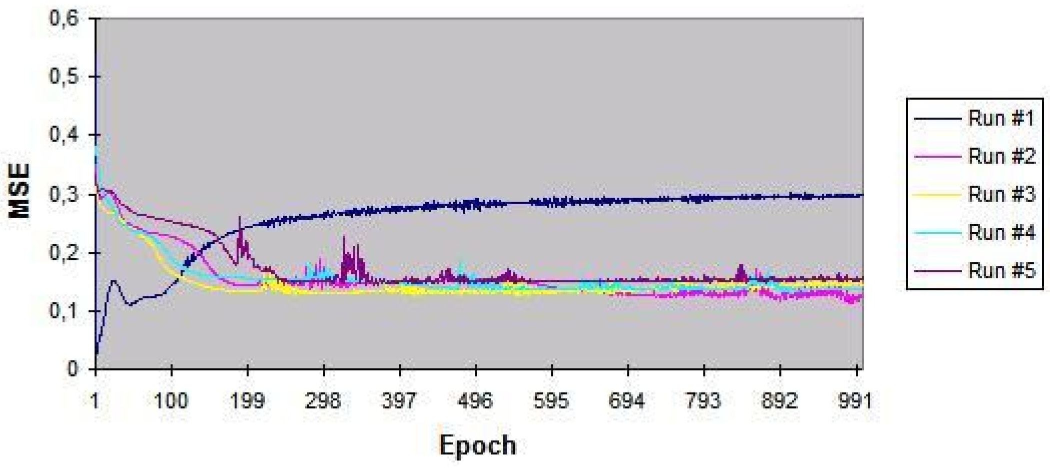

Figure 5.30 Cross validation MSE

Table 5.68 Minimum MSE values for momentum rate=0,9

Subset	**Minimum for Run #1**	**Minimum for Run #2**	**Minimum for Run #3**	**Minimum for Run #4**	**Minimum for Run #5**
Training	0.09526639	0.0998127	0.09977897	0.09652657	0.11622639
Cross validation	0.13188073	0.13811468	0.14802913	0.14721098	0.15441076

Table 5.69 Confusion matrix for testing the training subset of one hidden layer, momentum rate=0,9

Output / Desired	*VEHCL(opel)*	*VEHCL(bus)*	*VEHCL(saab)*	*VEHCL(van)*
VEHCL(opel)	69	0	27	0
VEHCL(bus)	0	119	4	0
VEHCL(saab)	27	1	67	0
VEHCL(van)	2	1	1	105

Table 5.70 Performance values (training)

Performance	***VEHCL(opel)***	***VEHCL(bus)***	***VEHCL(saab)***	***VEHCL(van)***
MSE	0.105641636	0.021068844	0.100429624	0.008599026
NMSE	0.593480448	0.103164229	0.560224848	0.04608012
MAE	0.211926839	0.082762048	0.21101388	0.06113804
Min Abs Error	0.00039978	0.000328017	0.000504421	2.09743E-05
Max Abs Error	1.013030418	0.778413834	1.020201788	0.76959377
r	0.638941354	0.947963627	0.663645649	0.979739911
Percent Correct	70.40816327	98.34710744	67.67676768	100

Table 5.71 Confuison matrix for testing the cross validation subset of one hidden layer, momentum rate=0,9

Output / Desired	*VEHCL(opel)*	*VEHCL(bus)*	*VEHCL(saab)*	*VEHCL(van)*
VEHCL(opel)	41	0	21	1
VEHCL(bus)	3	44	0	2
VEHCL(saab)	21	0	35	0
VEHCL(van)	2	0	0	42

Tablo 5.72 Performance values (cross validation)

Performance	***VEHCL(opel)***	***VEHCL(bus)***	***VEHCL(saab)***	***VEHCL(van)***
MSE	0.161504525	0.024304293	0.112727474	0.028503339
NMSE	0.747159996	0.147772204	0.579947754	0.170466275
MAE	0.275615104	0.085690461	0.244870874	0.093145763
Min Abs Error	3.45259E-05	0.00121531	0.000375239	0.00022727
Max Abs Error	0.984461199	0.909582052	0.917315016	0.947606892
r	0.528742463	0.926859502	0.651468553	0.921119997
Percent Correct	61.19402985	100	62,5	93,33333333

Table 5.73 Mean accuracy performance of momentum rates (veh dataset)

Momentum value/ Subset	β **=0.1**	β **=0.3**	β **=0.5**	β **=0.7**	β =0.9
Training	45.41 %	31.64 %	80.00 %	84.08%	84.11%
Cross validation	46.54 %	31.8 %	76.67 %	76.67%	79.25%

For β=0.9, the accuracy of both training and cross validation subsets have the highest values. So, among the momentum rates given, choosing the momentum rate for 0.9 is suitable.

5.3.7 The Optimum Hidden Layer Size

In order to construct the ANN needed, the optimum hidden layer size should be defined. Starting from the one hidden layer, the hidden layer size is increased up to five hiden layers and each hadden layer sizes' performances are compared. For each hidden layer size performance evaluation, 5 runs made and 1000 epochs generated per run.The momentum rate is 0.9 as found optimum before.

5.3.7.1 Two hidden Layered ANN

The optimum hidden layer size is found as two hidden layered ANN.

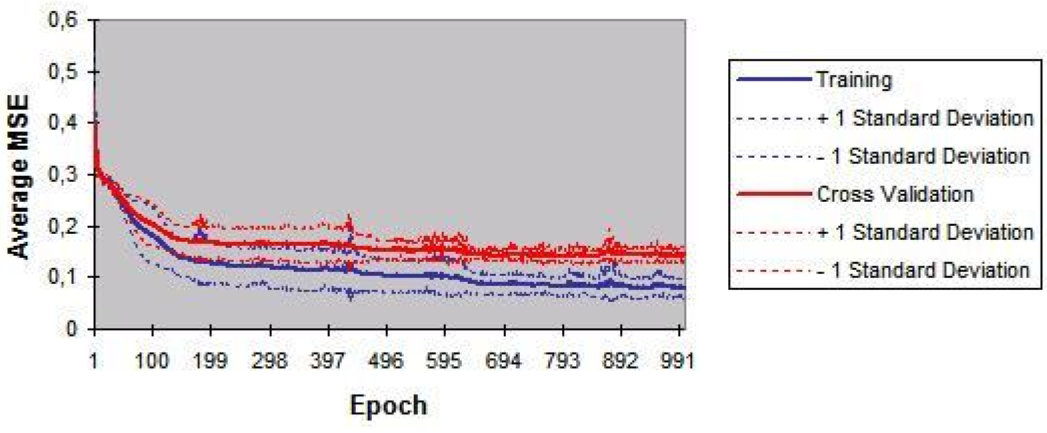

Figure 5.31 Average MSE

Table 5.74 MSE statistics

All Runs	*Training Minimum*	*Training Standard Deviation*	*Cross Validation Minimum*	*Cross Validation Standard Deviation*
Average of Minimum MSEs	0.076109765	0.019130725	0.131692191	0.006750379
Average of Final MSEs	0.0824498	0.016021286	0.143002858	0.010210213

Table 5.75 Best MSE values

Best Networks	*Training*	*Cross Validation*
Run #	3	5
Epoch #	937	987
Minimum MSE	0.053807166	0.126436126
Final MSE	0.069359214	0.134385163

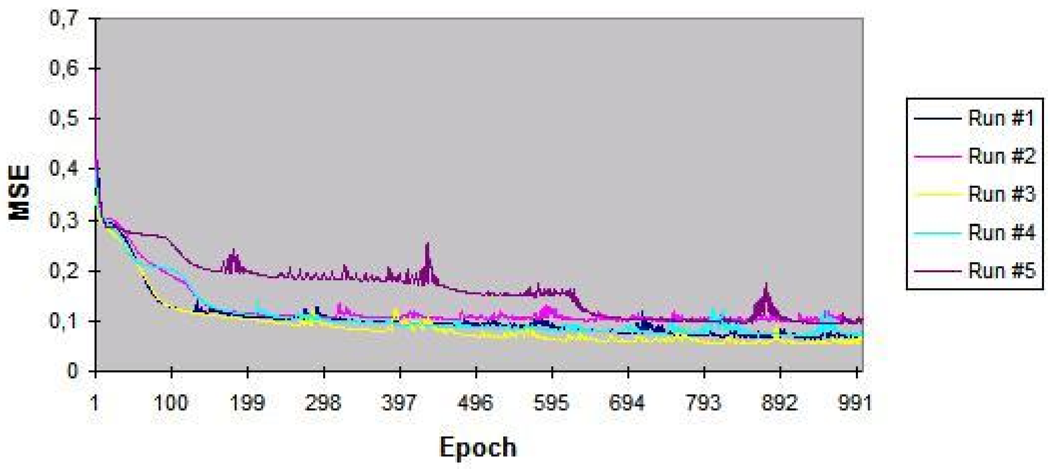

Figure 5.32 Training MSE

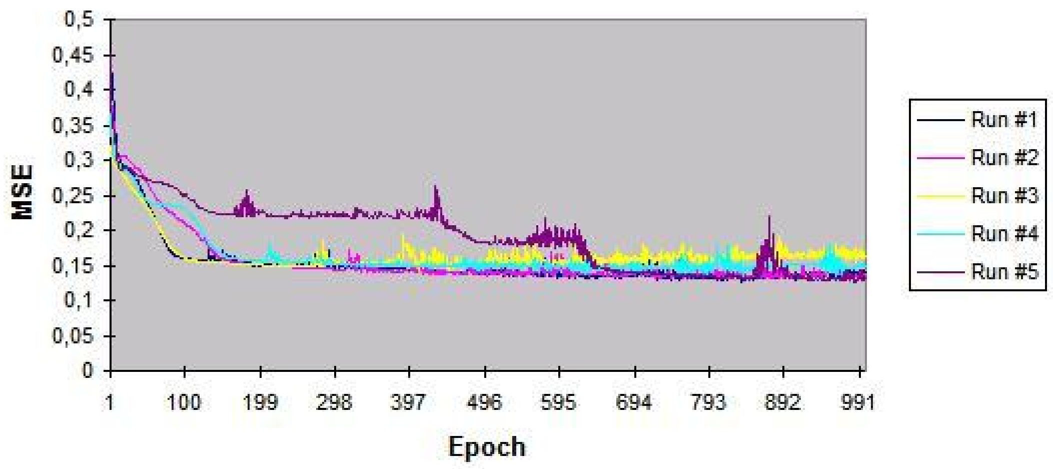

Figure 5.33 Cross validation MSE

Table 5.76 Minimum MSE values for each run of two hidden layer size ANN topology

Subset	Minimum for Run #1	Minimum for Run #2	Minimum for Run #3	Minimum for Run #4	Minimum for Run #5
Training	0.06343473	0.09810928	0.05380717	0.07169742	0.09350024
Cross validation	0.12705209	0.13027827	0.14314187	0.13155261	0.12643613

Table 5.77 Confusion matrix for testing the training subset of two hidden layers, momentum rate=0,9

Output / Desired	VEHCL(opel)	VEHCL(bus)	VEHCL(saab)	VEHCL(van)
VEHCL(opel)	46	0	18	0
VEHCL(bus)	0	118	1	0
VEHCL(saab)	51	1	79	0
VEHCL(van)	1	2	1	105

Table 5.78 Performance values for the training subset data (veh dataset)

Performance	VEHCL(opel)	VEHCL(bus)	VEHCL(saab)	VEHCL(van)
MSE	0.108927159	0.012367696	0.107167001	0.008262857
NMSE	0.611938071	0.060558796	0.59780784	0.044278668
MAE	0.232725706	0.038301686	0.227173695	0.047847358
Min Abs Error	5.43513E-05	2.63465E-05	0.00066498	0.002123372
Max Abs Error	1.017762345	0.961997335	0.946268247	0.896269902
r	0.624232906	0.970181406	0.634660012	0.979687715
Percent Correct	46.93877551	97.52066116	79.7979798	100

Table 5.79 Confusion matrix for testing the cross validation subset of two hidden layers, momentum rate=0,9

Output / Desired	*VEHCL(opel)*	*VEHCL(bus)*	*VEHCL(saab)*	*VEHCL(van)*
VEHCL(opel)	32	2	9	0
VEHCL(bus)	3	42	0	2
VEHCL(saab)	30	0	47	1
VEHCL(van)	2	0	0	42

Table 5.80 Performance values for the cross validation subset data (veh dataset)

Performance	***VEHCL(opel)***	***VEHCL(bus)***	***VEHCL(saab)***	***VEHCL(van)***
MSE	0.143302882	0.024724454	0.122465123	0.023065753
NMSE	0.66295468	0.150326819	0.630044929	0.137946401
MAE	0.289808693	0.052091883	0.266209465	0.070505278
Min Abs Error	9.9691E-05	4.87393E-06	0.000790238	0.004297736
Max Abs Error	0.994809899	0.981629281	0.769235028	1.006620327
r	0.583613636	0.924460062	0.61229832	0.935782648
Percent Correct	47.76119403	95.45454545	83.92857143	93.33333333

Table 5.81 Mean accuracy performance of hidden layers

Hidden layer size/ Subset	**Hidden Layer=1**	**Hidden Layer=2**	**Hidden Layer=3**	**Hidden Layer=4**	**Hidden Layer=5**
Training	84.1%	81.07%	33.28%	25.00%	25.00%
Cross validation	79,26%	80.12%	34.37%	25.00%	25.00%

Mean accuracy values are presented seen in Table 5.81 that are including the performance values. The true classified exemplars are gathered on the orthogonal of the confusion matrix which are named as the true classification.

5.3.8 Optimum Number Of Processing Elements Of The Hidden Layers

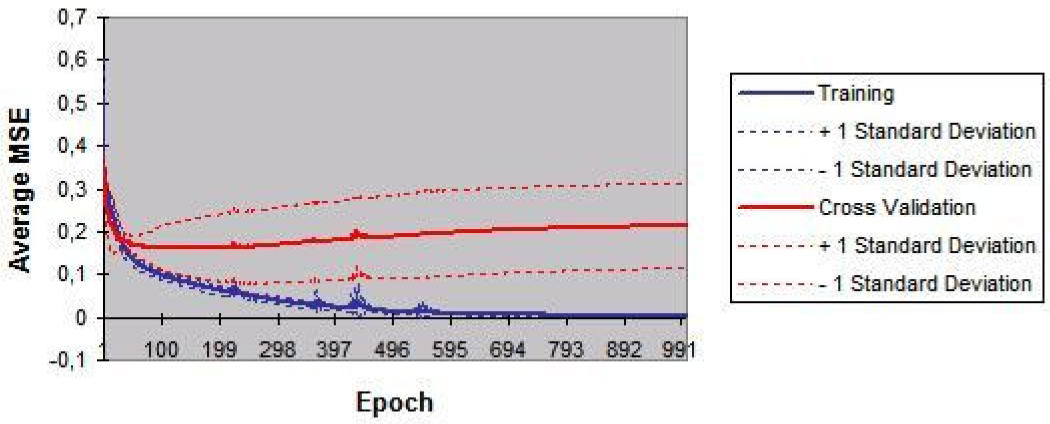

Figure 5.34 Average values of minimum MSEs

Table 5.82 MSE statistics

All Runs	*Training Minimum*	*Training Standard Deviation*	*Cross Validation Minimum*	*Cross Validation Standard Deviation*
Average of Minimum MSEs	0.003037419	0.000950724	0.115104295	0.012618901
Average of Final MSEs	0.003037419	0.000950724	0.213971905	0.098673842

Table 5.83 Best MSE values

Best Networks	*Training*	*Cross Validation*
Run #	2	1
Epoch #	1000	15
Minimum MSE	0.001682337	0.094483546
Final MSE	0.001682337	0.388886229

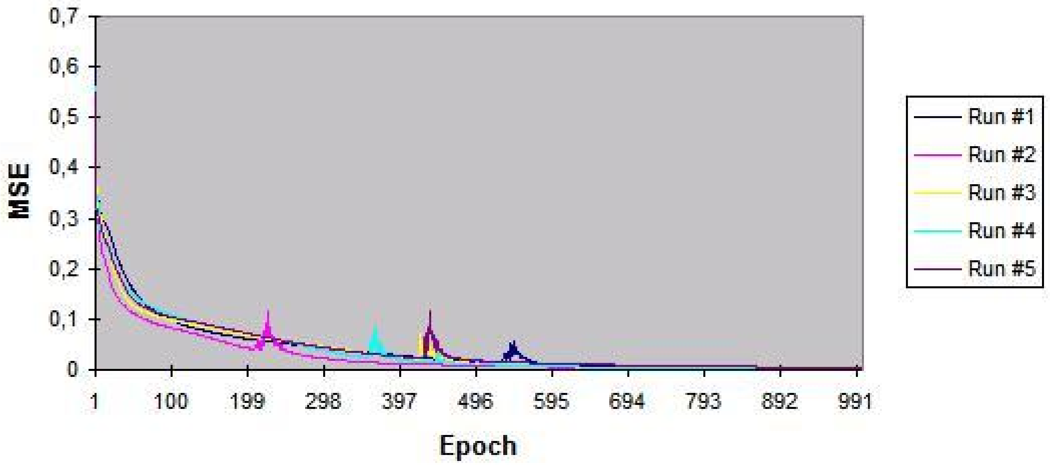

Figure 5.35 Training MSE

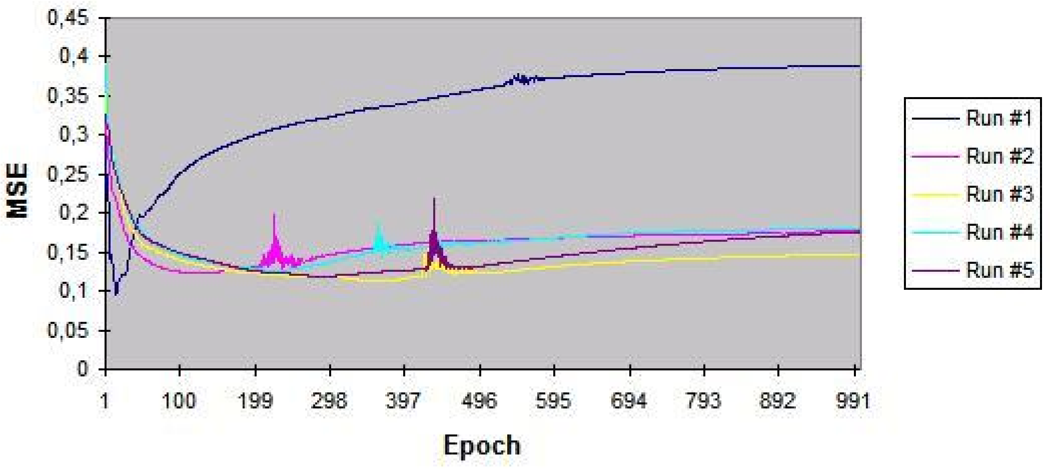

Figure 5.36 Cross validation MSE

Because there are four different classes, as an output feature, the minimum processing element in Hidden1 layer and Hidden2 layer is four. Starting from the initial point, in each hidden layer, the number of processing elements (neurons) in the hidden layer are increased by incremental algorithm. After running the simulation, that is converging to minima, the optimal number of prcessing elements in hidden layer 1 is 29. For hidden layer 2, the optimal number of prcessing elements is 10. So, the proposed neural network structure is as 18-29-10-4.

5.4 Parameter Optimization

The performance of the backpropagation network (BPN) is affected by the network architecture and parameter settings. The user must therefore determine the required numbers of optimal layers and neurons in the hidden layers (Liu, Liu,Wang & Niu, 2004).

However, there has been no clearly defined theory for calculating the ideal parameter settings, and even slight parameter changes can cause major variations in the behaviour of almost all networks. In neural network models, these factors have been determined heuristically and by trial and error, which are both time consuming and tedious. By the help of genetic parameters, the evolving mechanism here works with not one but multiple populations, all of which evolve separately most of the time, except for once every several generations that are applied as a crossover operation from different populations. Since sometimes it could happen for a single population scheme that though the neural network could theoretically solve a certain classification problem, the system may not return a correct solution. This is because of the random nature of the algorithm and its reliance on natural selection; mutation and crossover. Hancock (1992), discussed the permutation problem. Thus, it could happen that a certain flow of events that would lead to a correct solution will not occur and thus a solution will not be found. For that reason, genetic parameters that are crossover and mutation operator values must be selected carefully. However, by using several unrelated populations, the probability has been decreased of this occurrence, since if some population has poor individuals the solution could still be found at another.

The purpose of this study is to apply genetic algorithms (GAs) to determine the number of neurons in the hidden layers, the momentum, and the learning rates for minimizing the time and effort required to find the optimal architecture and parameters of the backpropagation algorithm.

It also focused on improving the accuracy of classification and verifying the performance and validitation of optimizing both the neural network size and its parameters using GAs.

But as soon as problems to be solved are getting more complex, backpropagation more and more fails due to its inherent gradient descent. Backpropagation is needed in these cases, several starts with varying initial weights to meet the desired error or is not able to solve the required task at all while genetic algorithms are still performing very well.

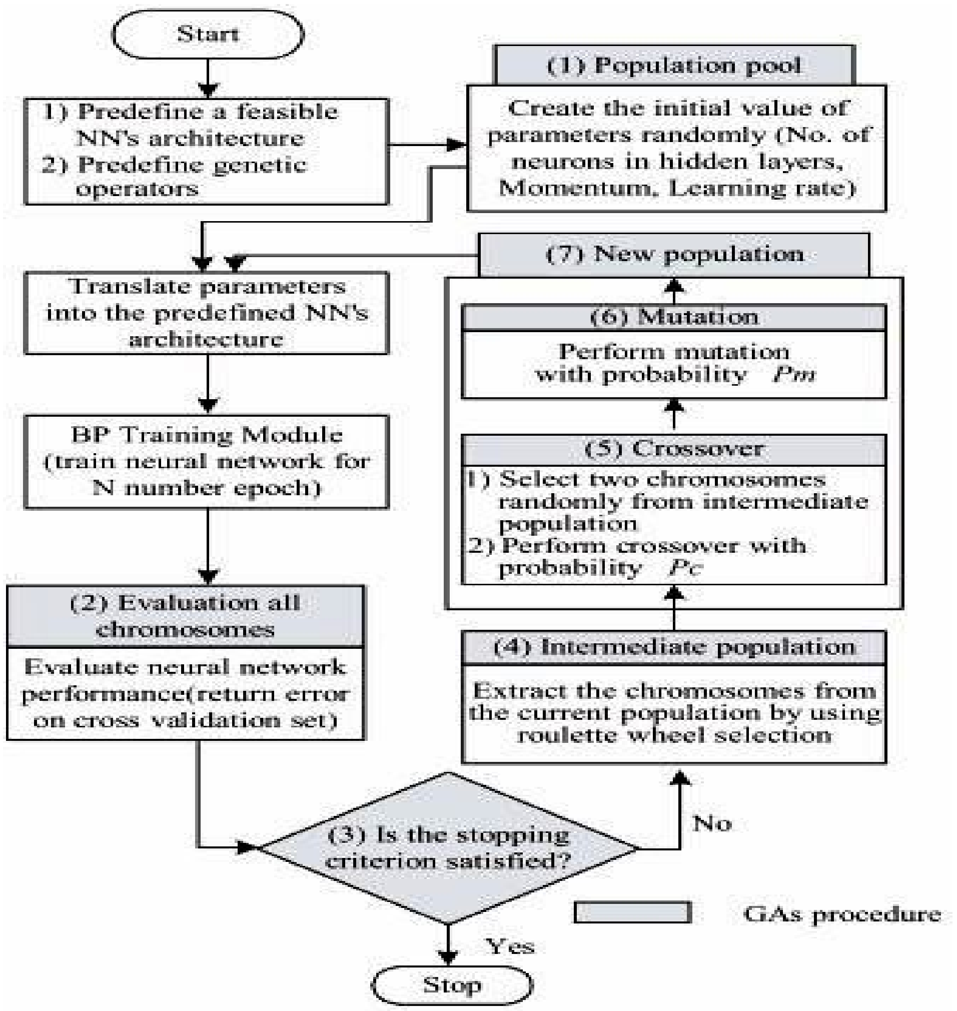
Start
1) Predefine a feasible NN's architecture
2) Predefine genetic operators
(1) Population pool
Create the initial value of parameters randomly (No. of neurons in hidden layers, Momentum, Learning rate)
(7) New population
Translate parameters into the predefined NN's architecture
(6) Mutation
Perform mutation with probability Pm
BP Training Module (train neural network for N number epoch)
(5) Crossover
1) Select two chromosomes randomly from intermediate population
2) Perform crossover with probability Pc
(2) Evaluation all chromosomes
Evaluate neural network performance(return error on cross validation set)
(4) Intermediate population
Extract the chromosomes from the current population by using roulette wheel selection
(3) Is the stopping criterion satisfied?
No
Yes
GAs procedure
Stop

Figure 5.37 Framework for EANN parameter optimization

Table 5.84 Optimized parameters of hidden layer 1 for tae dataset (crossover values (0.7-0.9), mutation values (0.05, 0.1, 0.3, 0.5))

Genetic Parameters	Crossover Type	hidden layer 1		
		optimum PE	stepsize	momentum
crossover 0.9 mutation 0.05	one point	3	0.73150936	0.240180652
	two point	3	0.43076188	0.387495726
	arithmetic	13	0.65257714	0.581940507
	heuristic	3	0.02556954	0.693615146
crossover 0.9 mutation 0.1	one point	6	0.55674446	0.281066738
	two point	3	0.64099817	0.126715657
	arithmetic	18	0.68266252	0.399812089
	heuristic	3	0.53945832	0.418522512
crossover 0.9 mutation 0.3	one point	3	0.11468645	0.016847524
	two point	4	0.60162268	0.474645512
	arithmetic	17	0.92741906	0.614680999
	heuristic	3	0.87018994	0.002328123
crossover 0.9 mutation 0.5	one point	4	0.88114978	0.234085507
	two point	3	0.75098245	0.116068873
	arithmetic	12	0.41713642	0.595252206
	heuristic	3	0.37974655	0.499281889
crossover 0.7 mutation 0.05	one point	4	0.32681967	0.079966614
	two point	3	0.60043827	0.587494441
	arithmetic	8	0.48178512	0.533627331
	heuristic	3	0.97777139	0.815553682
crossover 0.7 mutation 0.1	one point	7	0.07847586	0.178002265
	two point	3	0.29226171	0.597501592
	arithmetic	8	0.00348756	0.350691609
	heuristic	3	0.7269501	0.187918958
crossover 0.7 mutation 0.3	one point	6	0.88924089	0.240369414
	two point	4	0.17073162	0.217492215
	arithmetic	7	0.36689023	0.316115925
	heuristic	3	0.64213635	0.612095758
crossover 0.7 mutation 0.5	one point	5	0.02118434	0.012264858
	two point	13	0.21862656	0.035031489
	arithmetic	10	0.44708816	0.921060563
	heuristic	3	0.77685623	0.044067876

The longer the chromosomes the more generations are required. In general, genetic algorithms are inherently slower than backpropagation. This could be expected due to their global search technique compared to the highly directed gradient descent learning of backpropagation.

Optimized parameters mentioned in the tables have been maintained by the run of evolutionary neural networks stated with the parameters. At the end of each run; MSE statistics have been determined, minimum MSE and average MSE values. As seen , there is a conjuction with the parameters of the two types of tables; optimized parameters and MSE statistics. In every possible genetic parameter selection; all these EANN parameters are gathered to analyze.

The evolutionary neural network parameters were 20 generations run, 1000 epochs per generation, the population size to set 50 (chromosome number). Because of the big population size; the time needed for calculation increases so much as 6 to 7 hours. Table 5.84 and Table 5.85 give the optimized stepsize and momentum values and also the number of processing elements in the hidden layer 1. Table 5.84 shows that whatever the crossover and the mutation parameter are, the optimum processing elements needed in the hidden layer are 2-4 times much more than the other crossover types. The optimized parameters are the selected parameters for each run of the evolutionary neural network related with the genetic parameters given.

From experiences of the study, mutation values are restricted up to 0.5. If higher mutation values are taken into consideration, the mutational search changes to a random search which means the possibility to tackle up a local minima is getting higher.

In order to evaluate the performance of the EANN, the basic ANN statistics are compared with. The relative percent difference (RPD) is calculated as:

$$difference\% = \frac{\min(MSE)_{EANN} - \min(MSE)_{ANN}}{\min(MSE)_{ANN}} x100$$

5.4.1 Parameter Optimization For Teaching Assistant Evaluation (tae) Dataset

The neural network topology for the teaching assistant evaluation dataset is set. The optimum number of hidden layers is one and the optiumum number of hidden units in the hidden layer is 14. At that point, these values for the parameters are found at fixed weight matrix and no genetic operators are implemented to the neural network design.

By embedding the evolutionary algorithm to the network weight space, new neural network arhitectures are gained in a population pool. New parameters extracted from the genetic operators are put in the proposed neural network and statistical performance values (minimum MSE and average MSE) are taken out. For each genetical scenario, the statistical results are gathered at a performance table and these results are compared with the traditional neural network structure defined before.

Table 5.85 Optimized parameters of hidden layer 1 for tae dataset (crossover values (0.1), mutation values (0.05, 0.1, 0.3, 0.5))

Genetic Parameters	**Crossover Type**	**hidden layer 1**		
		optimum PE	**stepsize**	**momentum**
crossover	**one point**	5	0.228667	0.033171
0.1	**two point**	3	0.380425	0.317079
mutation	**arithmetic**	14	0.166905	0.202487
0.05	**heuristic**	4	0.788608	0.336321
crossover	**one point**	5	0.592122	0.191385
0.1	**two point**	3	0.567506	0.190046
mutation	**arithmetic**	12	0.051978	0.464272
0.1	**heuristic**	4	0.848474	0.483011
crossover	**one point**	6	0.768924	0.342008
0.1	**two point**	3	0.721577	0.078652
mutation	**arithmetic**	17	0.376445	0.556608
0.3	**heuristic**	4	0.949445	0.833346
crossover	**one point**	4	0.304445	0.312772
0.1	**two point**	4	0.017837	0.17135
mutation	**arithmetic**	10	0.06739	0.773389
0.5	**heuristic**	4	0.864954	0.335542

Table 5.86 Optimized parameters of hidden layer 1 for tae dataset (crossover values (0.3-0.5), mutation values (0.05, 0.1, 0.3, 0.5))

Genetic Parameters	Crossover Type	hidden layer 1		
		optimum PE	stepsize	momentum
crossover 0.5 mutation 0.05	one point	4	0.829473656	0.154846304
	two point	3	0.237959042	0.087478378
	arithmetic	9	0.860443377	0.408215093
	heuristic	3	0.704181555	0.075193669
crossover 0.5 mutation 0.1	one point	5	0.587704987	0.213099964
	two point	3	0.342049777	0.507790161
	arithmetic	7	0.89050051	0.978083575
	heuristic	3	0.875925333	0.882730772
crossover 0.5 mutation 0.3	one point	6	0.534752125	0.300233845
	two point	5	0.402786642	0.109532591
	arithmetic	8	0.934373671	0.832060151
	heuristic	4	0.619565791	0.962117942
crossover 0.5 mutation 0.5	one point	3	0.246249197	0.343565768
	two point	3	0.526623434	0.080084856
	arithmetic	10	0.65241926	0.993482987
	heuristic	3	0.941239698	0.06133486
crossover 0.3 mutation 0.05	one point	4	0.886471064	0.207323016
	two point	3	0.641827019	0.439928246
	arithmetic	15	0.562916541	0.00776508
	heuristic	3	0.428609898	0.329548231
crossover 0.3 mutation 0.1	one point	4	0.78071679	0.008922956
	two point	4	0.414314222	0.126228985
	arithmetic	15	0.920649611	0.757566181
	heuristic	4	0.079784386	0.217889502
crossover 0.3 mutation 0.3	one point	18	0.65412166	0.256123363
	two point	3	0.731228032	0.349469001
	arithmetic	15	0.581840502	0.763365527
	heuristic	9	0.797058013	0.159718011
crossover 0.3 mutation 0.5	one point	4	0.956701127	0.008711356
	two point	3	0.305796922	0.533341844
	arithmetic	13	0.043253544	0.098009882
	heuristic	3	0.626679383	0.11928196

Table 5.87 Minimum and average MSE values for EANN design (tae dataset)

Crossover Value	Performance Type	mutation=0.05			
		one point	two point	arithmetic	heuristic
0.9	Min MSE	0.184515	0.193226	0.176466	0.179978
	Avg MSE	0.300455	0.357999	0.33765	0.304546
0.7	Min MSE	0.181724	0.177355	0.187003	0.184149
	Avg MSE	0.334855	0.317202	0.365957	0.309382
0.5	Min MSE	0.185062	0.194295	0.206236	0.167622
	Avg MSE	0.32942	0.340194	0.379594	0.317473
0.3	Min MSE	0.179579	0.202073	0.18986	0.180831
	Avg MSE	0.344877	0.361766	0.349243	0.33478
0.1	Min MSE	0.177645	0.192304	0.204626	0.186952
	Avg MSE	0.359675	0.342767	0.413944	0.339928

Table 5.88 The relative percent difference between ANN and EANN performance values

Crossover Value	Performance Type	mutation=0.05			
		one point	two point	arithmetic	heuristic
0.9	Min MSE	0.14%	4.87%	**-4.23%**	**-2.32%**
	Avg MSE	**-1.61%**	17.24%	10.58%	**-0.27%**
0.7	Min MSE	**-1.38%**	**-3.75%**	1.49%	**-0.06%**
	Avg MSE	9.66%	3.88%	19.85%	1.32%
0.5	Min MSE	0.44%	5.45%	11.93%	**-9.03%**
	Avg MSE	7.88%	11.41%	24.31%	3.97%
0.3	Min MSE	**-2.54%**	9.67%	3.04%	**-1.86%**
	Avg MSE	12.94%	18.47%	14.37%	9.64%
0.1	Min MSE	**-3.59%**	4.37%	11.05%	1.46%
	Avg MSE	17.79%	12.25%	**35.56%**	11.32%

As stated in Table 5.88, as the mutation value as at low levels, the higher crossover values help EANN to perform better minimum and average MSE solutions. One point crossover has better minimum MSE solutions for highest and lowest crossover values. In general, two point crossover does not give much good minimum MSE solutions than the generational ANN. Arithmetic crossover is opposite to the mutation value. As the mutation rate gets smaller in value, the possible good minimum MSE solutions for arithmetic crossover can be seen in Table 5.88 and Table 5.90, in high crossover values. The opposite situation can be seen in Table 5.94. Because the mutation rate is so small, the heuristic crossover operator behaves as a random operator. So, almost at all levels of crossover, an improvement in the minimum MSE minimization can be seen.

Although one point crossover gives better solution values for minimum MSE, the average MSE values can be worse. At Table 5.88, only for crossover rate 0.9, the average MSE gave better solution value than the traditional ANN. At other levels, the average MSE values for one point crossover 10% worse approximately. Arithmetic operators shows the worst performance for the average MSE values.

Increasing the mutation rate from 0.05 to 0.1 does not change the situation between the crossover types. At Table 5.90, arithmetic operator produces much worse average MSE values. One point crossover does not produce further improvement.

From Table 5.92, as the mutation increased to 0.3, which is an important operator to generate new individuals, randomness gets higher. Randomness has the same effect with the heuristic crossover.

Table 5.89 Minimum and average MSE values for EANN design (tae dataset)

Crossover Value	Performance Type	mutation=0.1			
		one point	two point	arithmetic	heuristic
0.9	Min MSE	0.183849	0.189584	0.184098	0.17782
	Avg MSE	0.321902	0.322333	0.35367	0.300545
0.7	Min MSE	0.187784	0.203728	0.201665	0.179272
	Avg MSE	0.347136	0.380452	0.401657	0.316814
0.5	Min MSE	0.198693	0.185993	0.186103	0.173962
	Avg MSE	0.358461	0.31858	0.343865	0.314338
0.3	Min MSE	0.190051	0.16559	0.201521	0.189786
	Avg MSE	0.333013	0.339525	0.401266	0.37751
0.1	Min MSE	0.207438	0.184873	0.192138	0.191442
	Avg MSE	0.372974	0.333297	0.364317	0.36352

Table 5.90 The relative percent difference between ANN and EANN performance values

Crossover Value	Performance Type	mutation=0.1			
		one point	two point	arithmetic	heuristic
0.9	Min MSE	**-0.22%**	2.89%	**-0.09%**	**-3.49%**
	Avg MSE	5.42%	5.56%	15.82%	**-1.58%**
0.7	Min MSE	1.91%	10.57%	9.45%	**-2.71%**
	Avg MSE	13.68%	24.59%	**31.54%**	3.75%
0.5	Min MSE	7.83%	0.94%	1.00%	**-5.59%**
	Avg MSE	17.39%	4.33%	12.61%	2.94%
0.3	Min MSE	3.14%	**-10.13%**	9.37%	3.00%
	Avg MSE	9.06%	11.19%	**31.41%**	23.63%
0.1	Min MSE	12.58%	0.33%	4.28%	3.90%
	Avg MSE	22.14%	9.15%	19.31%	19.05%

Table 5.91 Minimum and average MSE values for EANN design (tae dataset)

Crossover Value	Performance Type	mutation=0.3			
		one point	two point	arithmetic	heuristic
0.9	Min MSE	0.194421	0.197464	0.18491	0.179279
	Avg MSE	0.338284	0.367537	0.391568	0.34347
0.7	Min MSE	0.189479	0.186085	0.195875	0.183886
	Avg MSE	0.353423	0.33069	0.379913	0.339743
0.5	Min MSE	0.191511	0.193757	0.201105	0.182305
	Avg MSE	0.378879	0.363758	0.391021	0.316363
0.3	Min MSE	0.189359	0.192335	0.198526	0.187614
	Avg MSE	0.389787	0.341277	0.385752	0.352999
0.1	Min MSE	0.189208	0.182881	0.195454	0.17896
	Avg MSE	0.365237	0.323015	0.393626	0.372949

Table 5.92 The relative percent difference between ANN and EANN performance values

Crossover Value	Performance Type	mutation=0.3			
		one point	two point	arithmetic	heuristic
0.9	Min MSE	5.52%	7.17%	0.35%	**-2.70%**
	Avg MSE	10.78%	20.36%	28.23%	12.48%
0.7	Min MSE	2.83%	0.99%	6.30%	**-0.20%**
	Avg MSE	15.74%	8.30%	24.42%	11.26%
0.5	Min MSE	3.94%	5.15%	9.14%	**-1.06%**
	Avg MSE	24.08%	19.13%	28.05%	3.60%
0.3	Min MSE	2.77%	4.38%	7.74%	1.82%
	Avg MSE	27.65%	11.76%	26.33%	15.60%
0.1	Min MSE	2.69%	**-0.75%**	6.08%	**-2.88%**
	Avg MSE	19.61%	5.78%	28.91%	22.14%

Table 5.93 Minimum and average MSE values for EANN design (tae dataset)

Crossover Value	Performance Type	mutation=0.5			
		one point	two point	arithmetic	heuristic
0.9	Min MSE	0.191169	0.182371	0.189458	0.17833
	Avg MSE	0.373317	0.366227	0.380388	0.3674
0.7	Min MSE	0.204179	0.201667	0.198073	0.181379
	Avg MSE	0.382814	0.399454	0.364372	0.343713
0.5	Min MSE	0.194121	0.191101	0.190737	0.178933
	Avg MSE	0.351897	0.377449	0.375544	0.335193
0.3	Min MSE	0.18581	0.187488	0.193053	0.172098
	Avg MSE	0.333571	0.35295	0.378377	0.35586
0.1	Min MSE	0.196609	0.179915	0.178663	0.174804
	Avg MSE	0.389925	0.342659	0.37174	0.354238

Moving the mutation value to 0.5, one point crossover does not improve any statistical performance value. Heuristic crossover uses its advantage, high mutation rate does not affect its mechanics. At all levels of crossover in Table 5.94, it has better results than the traditional 56-4-3 network.

Table 5.94 The relative percent difference between ANN and EANN performance values

Crossover Value	Performance Type	mutation=0.5			
		one point	two point	arithmetic	heuristic
0.9	Min MSE	3.75%	**-1.02%**	2.82%	**-3.22%**
	Avg MSE	22.26%	19.93%	24.57%	20.32%
0.7	Min MSE	10.81%	9.45%	7.50%	**-1.56%**
	Avg MSE	25.37%	**30.82%**	19.33%	12.56%
0.5	Min MSE	5.35%	3.71%	3.52%	**-2.89%**
	Avg MSE	15.24%	23.61%	22.99%	9.77%
0.3	Min MSE	0.84%	1.75%	4.77%	**-6.60%**
	Avg MSE	9.24%	15.59%	23.91%	16.54%
0.1	Min MSE	6.70%	**-2.36%**	**-3.04%**	**-5.13%**
	Avg MSE	27.69%	12.22%	21.74%	16.01%

Two point and arithmetic crossover types show better performance than the basic ANN for high and low crossover points when the mutation rate is high.

5.4.2 Parameter Optimization For Vehicle Silhoutte (veh) Dataset

The neural network topology for the vehicle silhoutte (*veh*) dataset is set. The optimum number of hidden layers is two and the optimum momentum rate is 0.9. As there are two hidden layers exist for the topology, the basic ANN topology is implemented as 18-4-4-4.

Table 5.95 Optimized parameters of hidden layer 1 for tae dataset (crossover values (0.1), mutation values (0.05, 0.1, 0.3, 0.5))

Genetic Values	**Crossover type**	Hidden layer 1		
		opt PE	stepsize	momentum
crossover	one point	16	0.644326813	0.352986604
0.1	two point	20	0.060817322	0.231555361
mutation	arithmetic	22	0.7153409	0.096027724
0.05	heuristic	20	0.080998495	0.356019577
crossover	one point	19	0.754102965	0.113731649
0.1	two point	30	0.770179519	0.058653143
mutation	arithmetic	11	0.610429353	0.396056352
0.1	heuristic	11	0.21231876	0.312689071
crossover	one point	9	0.347335331	0.152195178
0.1	two point	14	0.254836305	0.090009665
mutation	arithmetic	16	0.475974884	0.294499587
0.3	heuristic	19	0.493676218	0.089035444
crossover	one point	14	0.25827033	0.197631032
0.1	two point	15	0.179849973	0.234581227
mutation	arithmetic	14	0.160482264	0.167729741
0.5	heuristic	14	0.045357284	0.489698402

Table 5.96 Optimized parameters of hidden layer 1 for veh dataset (crossover values (0.7-0.9), mutation values (0.05, 0.1, 0.3, 0.5))

Genetic Values	**Crossover type**	Hidden layer 1		
		opt PE	stepsize	momentum
crossover	one point	29	0.731131732	0.180706456
0.9	two point	13	0.598734231	0.135238261
mutation	arithmetic	23	0.383692276	0.746376718
0.05	heuristic	25	0.061717978	0.742125685
crossover	one point	16	0.13288304	0.790880532
0.9	two point	15	0.567804211	0.821727453
mutation	arithmetic	9	0.276056446	0.715809371
0.1	heuristic	6	0.17453568	0.301922153
crossover	one point	11	0.16585049	0.996871098
0.9	two point	21	0.418494086	0.472754749
mutation	arithmetic	25	0.728537204	0.65661347
0.3	heuristic	7	0.026606511	0.611669395
crossover	one point	22	0.127194705	0.109461119
0.9	two point	13	0.220622711	0.434537178
mutation	arithmetic	11	0.104285872	0.939742975
0.5	heuristic	16	0.617710333	0.760309386
crossover	one point	11	0.236349002	0.259651447
0.7	two point	29	0.022882492	0.745005318
mutation	arithmetic	22	0.680386298	0.479643406
0.05	heuristic	5	0.074268236	0.750166307
crossover	one point	28	0.488491655	0.821246086
0.7	two point	7	0.583741485	0.485529282
mutation	arithmetic	7	0.158327421	0.070181069
0.1	heuristic	29	0.300531577	0.010028216
crossover	one point	13	0.63513043	0.365626125
0.7	two point	24	0.032180012	0.083931138
mutation	arithmetic	16	0.426116722	0.236978262
0.3	heuristic	8	0.359481865	0.265168658
crossover	one point	30	0.240521449	0.36664013
0.7	two point	14	0.048300096	0.219123706
mutation	arithmetic	15	0.740171187	0.266201726
0.5	heuristic	21	0.522609395	0.266852163

Table 5.97 Optimized parameters of hidden layer 1 for tae dataset (crossover values (0.3, 0.5), mutation values (0.05, 0.1, 0.3, 0.5))

Genetic Values	**Crossover type**	Hidden layer 1		
		opt PE	stepsize	momentum
crossover 0.5 mutation 0.05	one point	24	0.463598757	0.926746939
	two point	28	0.156988464	0.302167522
	arithmetic	14	0.490223059	0.487381976
	heuristic	29	0.296740884	0.183369686
crossover 0.5 mutation 0.1	one point	19	0.299920733	0.378182613
	two point	11	0.603199339	0.226041547
	arithmetic	22	0.605653353	0.355670106
	heuristic	18	0.425742687	0.170712246
crossover 0.5 mutation 0.3	one point	5	0.184241977	0.083488916
	two point	7	0.139402898	0.472462966
	arithmetic	7	0.431058627	0.178983243
	heuristic	13	0.115400651	0.294612211
crossover 0.5 mutation 0.5	one point	18	0.363043718	0.1966738
	two point	20	0.656682242	0.152477884
	arithmetic	7	0.558312521	0.026228541
	heuristic	16	0.451297951	0.040072031
crossover 0.3 mutation 0.05	one point	30	0.349966758	0.746379131
	two point	16	0.069807036	0.493402154
	arithmetic	21	0.109333933	0.224802457
	heuristic	18	0.233272445	0.465346672
crossover 0.3 mutation 0.1	one point	10	0.607745975	0.249403197
	two point	7	0.173709094	0.190872284
	arithmetic	10	0.077024596	0.035406426
	heuristic	14	0.492649302	0.099028802
crossover 0.3 mutation 0.3	one point	8	0.55870235	0.285588293
	two point	29	0.259491492	0.259397283
	arithmetic	13	0.013004258	0.480006323
	heuristic	26	0.106394458	0.300096836
crossover 0.3 mutation 0.5	one point	28	0.026080326	0.200235962
	two point	17	0.468385149	0.366402004
	arithmetic	29	0.033144072	0.248680745
	heuristic	14	0.587780485	0.286396946

Table 5.98 Optimized parameters of hidden layer 2 for veh dataset (crossover values (0.5, 0.7), mutation values (0.05, 0.1, 0.3, 0.5))

Genetic Values	**Crossover type**	Hidden layer 2		
		opt PE	stepsize	momentum
crossover	one point	9	0.274840303	0.700503666
0.7	two point	11	0.885403682	0.766710157
mutation	arithmetic	14	0.295167251	0.183321405
0.05	heuristic	4	0.773724967	0.356331823
crossover	one point	4	0.777690122	0.980055104
0.7	two point	4	0.654506382	0.369583922
mutation	arithmetic	8	0.205629721	0.148070099
0.1	heuristic	5	0.605126941	0.97650962
crossover	one point	6	0.358363818	0.631788601
0.7	two point	15	0.680976722	0.49113705
mutation	arithmetic	9	0.899049914	0.793311424
0.3	heuristic	4	0.982339803	0.148992704
crossover	one point	5	0.531609951	0.709586812
0.7	two point	4	0.894198142	0.414844732
mutation	arithmetic	5	0.860526522	0.706131514
0.5	heuristic	16	0.548266189	0.969215099
crossover	one point	9	0.431113302	0.861491783
0.5	two point	8	0.281698273	0.77473919
mutation	arithmetic	16	0.484641668	0.847296276
0.05	heuristic	6	0.02463816	0.928584323
crossover	one point	8	0.201159777	0.31775389
0.5	two point	5	0.746631505	0.050907944
mutation	arithmetic	7	0.778632815	0.625943794
0.1	heuristic	8	0.093567208	0.445260088
crossover	one point	9	0.063804124	0.867786583
0.5	two point	10	0.360432331	0.664174738
mutation	arithmetic	10	0.210385414	0.097927674
0.3	heuristic	9	0.576409262	0.886767806
crossover	one point	11	0.847640025	0.705812743
0.5	two point	7	0.955516844	0.170821791
mutation	arithmetic	15	0.643077156	0.671235253
0.5	heuristic	6	0.111990082	0.537553622

Table 5.99 Optimized parameters of hidden layer 2 for veh dataset (crossover values (0.1, 0.3), mutation values (0.05, 0.1, 0.3, 0.5))

Genetic Values	**Crossover type**	Hidden layer 2		
		opt PE	stepsize	momentum
crossover 0.3 mutation 0.05	one point	3	0.912563257	0.436229633
	two point	7	0.369082959	0.982928089
	arithmetic	7	0.152389558	0.656250436
	heuristic	14	0.787869701	0.15655433
crossover 0.3 mutation 0.1	one point	5	0.356594261	0.325111764
	two point	7	0.011539621	0.119508741
	arithmetic	6	0.020010836	0.515361509
	heuristic	8	0.967749561	0.912200251
crossover 0.3 mutation 0.3	one point	5	0.484849858	0.953358726
	two point	6	0.133065809	0.170892297
	arithmetic	5	0.288564583	0.620631444
	heuristic	15	0.05641197	0.442243824
crossover 0.3 mutation 0.5	one point	9	0.310497395	0.556184416
	two point	7	0.34237292	0.620827775
	arithmetic	13	0.941960781	0.291629874
	heuristic	7	0.396300889	0.197463045
crossover 0.1 mutation 0.05	one point	5	0.072694801	0.742238893
	two point	5	0.915352476	0.837752896
	arithmetic	7	0.462651526	0.419824021
	heuristic	9	0.103972089	0.351954819
crossover 0.1 mutation 0.1	one point	14	0.781850819	0.773386778
	two point	16	0.342553704	0.631916215
	arithmetic	7	0.917533132	0.70720565
	heuristic	10	0.637788308	0.172991888
crossover 0.1 mutation 0.3	one point	7	0.05431525	0.15737485
	two point	6	0.352954438	0.805715483
	arithmetic	4	0.165300028	0.594149941
	heuristic	4	0.224060597	0.880928094
crossover 0.1 mutation 0.5	one point	14	0.736643082	0.651728804
	two point	10	0.69950394	0.355593528
	arithmetic	4	0.303498401	0.524408641
	heuristic	9	0.66898389	0.13252092

Table 5.100 Optimized parameters of hidden layer 2 for veh dataset (crossover values (0.9), mutation values (0.05, 0.1, 0.3, 0.5))

Genetic Values	**Crossover type**	Hidden layer 2		
		opt PE	stepsize	momentum
crossover 0.9 mutation 0.05	one point	27	0.433650672	0.343120277
	two point	14	0.832994449	0.816267941
	arithmetic	12	0.456014526	0.982881807
	heuristic	10	0.394280454	0.950045431
crossover 0.9 mutation 0.1	one point	4	0.310434189	0.930591856
	two point	8	0.549460847	0.026387242
	arithmetic	14	0.966494451	0.703005274
	heuristic	11	0.779814259	0.873394518
crossover 0.9 mutation 0.3	one point	14	0.756411434	0.896345201
	two point	16	0.730573733	0.811199545
	arithmetic	8	0.945908603	0.272854883
	heuristic	13	0.464050275	0.691238966
crossover 0.9 mutation 0.5	one point	5	0.137676704	0.960989461
	two point	11	0.113842204	0.684879896
	arithmetic	7	0.768571134	0.240737352
	heuristic	9	0.383434429	0.385284699

Table 5.101 Minimum and average MSE values for EANN design (veh dataset)

Crossover Value	**Performance Type**	**mutation=0.05**			
		one point	**two point**	**arithmetic**	**heuristic**
0.9	**Min MSE**	0.075718	0.086105	0.076751	0.072204
	Avg MSE	0.083667	0.096844	0.082112	0.075364
0.7	**Min MSE**	0.039974	0.076885	0.075217	0.06996
	Avg MSE	0.082463	0.089926	0.084912	0.082143
0.5	**Min MSE**	0.072453	0.094007	0.086036	0.078561
	Avg MSE	0.09386	0.107972	0.088238	0.08126
0.3	**Min MSE**	0.085513	0.081762	0.093731	0.095225
	Avg MSE	0.086938	0.086622	0.103299	0.100153
0.1	**Min MSE**	0.099972	0.093167	0.09279	0.091526
	Avg MSE	0.100512	0.107654	0.104595	0.092351

Table 5.102 The relative percent difference between ANN and EANN performance values

Crossover Value	Performance Type	mutation=0.05			
		one point	two point	arithmetic	heuristic
0.9	Min MSE	**-0.52%**	13.13%	0.84%	**-5.13%**
	Avg MSE	1.48%	17.46%	**-0.41%**	**-8.59%**
0.7	Min MSE	**-47.48%**	1.02%	**-1.17%**	**-8.08%**
	Avg MSE	0.02%	9.07%	2.99%	**-0.37%**
0.5	Min MSE	**-4.81%**	23.51%	13.04%	3.22%
	Avg MSE	13.84%	30.96%	7.02%	**-1.44%**
0.3	Min MSE	12.35%	7.43%	23.15%	25.12%
	Avg MSE	5.44%	5.06%	25.29%	21.47%
0.1	Min MSE	31.35%	22.41%	21.92%	20.25%
	Avg MSE	21.91%	30.57%	26.86%	12.01%

Table 5.103 Minimum and average MSE values for EANN design (veh dataset)

Crossover Value	Performance Type	mutation=0.1			
		one point	two point	arithmetic	heuristic
0.9	Min MSE	0.080027	0.073198	0.078625	0.072771
	Avg MSE	0.096409	0.084352	0.089128	0.094581
0.7	Min MSE	0.075468	0.099111	0.085782	0.088113
	Avg MSE	0.082922	0.113006	0.095283	0.096643
0.5	Min MSE	0.084575	0.093523	0.090744	0.081835
	Avg MSE	0.089086	0.100173	0.100747	0.090066
0.3	Min MSE	0.085362	0.084353	0.082114	0.081014
	Avg MSE	0.09185	0.085183	0.100492	0.119342
0.1	Min MSE	0.098404	0.091791	0.087469	0.093407
	Avg MSE	0.124449	0.098183	0.093544	0.125622

Table 5.104 The relative percent difference between ANN and EANN performance values

Crossover Value	Performance Type	mutation=0.1			
		one point	two point	arithmetic	heuristic
0.9	Min MSE	5.15%	-3.83%	3.30%	-4.39%
	Avg MSE	16.93%	2.31%	8.10%	14.71%
0.7	Min MSE	-0.84%	30.22%	12.71%	15.77%
	Avg MSE	0.57%	37.06%	15.56%	17.21%
0.5	Min MSE	11.12%	22.88%	19.23%	7.52%
	Avg MSE	8.05%	21.50%	22.19%	9.24%
0.3	Min MSE	12.16%	10.83%	7.89%	6.44%
	Avg MSE	11.40%	3.32%	21.88%	44.75%
0.1	Min MSE	29.29%	20.60%	14.92%	22.73%
	Avg MSE	50.94%	19.08%	13.46%	52.36%

Table 5.105 Minimum and average MSE values for EANN design (veh dataset)

Crossover Value	Performance Type	mutation=0.3			
		one point	two point	arithmetic	heuristic
0.9	Min MSE	0.077631	0.08759	0.07827	0.077513
	Avg MSE	0.093486	0.098436	0.091665	0.103597
0.7	Min MSE	0.086563	0.089774	0.086182	0.095886
	Avg MSE	0.115684	0.100704	0.095224	0.107835
0.5	Min MSE	0.080343	0.095503	0.098585	0.090153
	Avg MSE	0.088577	0.111584	0.099483	0.123994
0.3	Min MSE	0.084202	0.083201	0.088284	0.0805
	Avg MSE	0.096255	0.105228	0.091589	0.103662
0.1	Min MSE	0.080533	0.087841	0.100112	0.089507
	Avg MSE	0.084367	0.103941	0.129797	0.101394

Table 5.106 The relative percent difference ANN and EANN performance values

Crossover Value	Performance Type	mutation=0.3			
		one point	two point	arithmetic	heuristic
0.9	Min MSE	2.00%	15.08%	2.84%	1.84%
	Avg MSE	13.39%	19.39%	11.18%	25.65%
0.7	Min MSE	13.73%	17.95%	13.23%	25.98%
	Avg MSE	40.31%	22.14%	15.49%	30.79%
0.5	Min MSE	5.56%	25.48%	29.53%	18.45%
	Avg MSE	7.43%	35.34%	20.66%	50.39%
0.3	Min MSE	10.63%	9.32%	16.00%	5.77%
	Avg MSE	16.74%	27.63%	11.08%	25.73%
0.1	Min MSE	5.81%	15.41%	31.54%	17.60%
	Avg MSE	2.33%	26.07%	57.43%	22.98%

At Table 5.102, when the mutation rate is low, usually crossover operators at high levels perform better. One point crossover gives better minimum MSE solutions for the crossover rates; 0.5, 0.7 and 0.9. Also, arithmetic and heuristic crossover operators behave in the same manner. But two point crossover operator has the worst values for minimum MSE. At low mutation values, as the crossover rate starts to decrease, the average MSE values start to get worse, too. Two point and arithmetic operators perform in a bad way for average MSE.

At Table 5.106, as the mutation moves to 0.3, any of the crossover operator makes an improvement at any level of crossover. It is not logical to go further at this point, but; for the experimentations' sake, examining the Table 5.108, at all levels of crossover, for the mutation rate 0.5 which is very high, all the crossover operators produce bad results for both the minimum and average MSE values. Because the mutation is very high, improvement is not possible.

Table 5.107 Minimum and average MSE values for EANN design (veh dataset)

Crossover Value	Performance Type	mutation=0.5			
		one point	two point	arithmetic	heuristic
0.9	Min MSE	0.076964	0.084205	0.081483	0.080551
	Avg MSE	0.112792	0.10314	0.096668	0.115677
0.7	Min MSE	0.090686	0.105076	0.095825	0.086411
	Avg MSE	0.120297	0.138325	0.129223	0.116275
0.5	Min MSE	0.085793	0.092728	0.095327	0.091432
	Avg MSE	0.114022	0.09971	0.108583	0.094445
0.3	Min MSE	0.093826	0.098623	0.087461	0.087
	Avg MSE	0.1084	0.10392	0.109852	0.10928
0.1	Min MSE	0.086971	0.095297	0.083904	0.096262
	Avg MSE	0.089706	0.098037	0.117816	0.106418

Table 5.108 The relative percent difference between ANN and EANN performance values

Crossover Value	Performance Type	mutation=0.5			
		One point	two point	arithmetic	heuristic
0.9	Min MSE	1.12%	10.64%	7.06%	5.84%
	Avg MSE	36.80%	25.09%	17.24%	40.30%
0.7	Min MSE	19.15%	38.06%	25.90%	13.54%
	Avg MSE	45.90%	67.77%	56.73%	41.02%
0.5	Min MSE	12.72%	21.84%	25.25%	20.13%
	Avg MSE	38.29%	20.93%	**31.70%**	14.55%
0.3	Min MSE	23.28%	29.58%	14.91%	14.31%
	Avg MSE	**31.47%**	26.04%	33.23%	32.54%
0.1	Min MSE	14.27%	25.21%	10.24%	26.48%
	Avg MSE	8.80%	18.91%	42.89%	29.07%

5.5 Discussion

Two types of dataset are considered for evaluation of classification performance of evolutionary neural networks. *Teaching assistant evaluation (tae) dataset* has categorical-driven attribute types and on contrary, there is only one numerical attribute and others have categorical attributes.

The other dataset is the *vehicle silhoutte (veh) dataset* that all the attributes are numerical. Because the two datasets have opposite type of attribtures in common, the effects of numerical and categorical attributes can be examined.

The number of chromosomes in the evolutonary algortihm plays an important role on processing time. This parameter is chosen by trial and error. The minimum and average MSE values give opinion about the parameter performance. The first impression selecting the population size depends on the researches made before. Most of the publications chose a maximum of value of 50 chromosomes for classification. But this much longer chromosome type causes longer processing tiimes such as 8.5 hours. So, shorter possible chromosome length should be selected. When the population size was set to 30 and the generation number to 1000, the run values were nearly the same. The processing time decreased 4.6 hours on average.

The selection of the percentage of the whole raw data to split into training, cross validation and test subset is important. For *tae* dataset, there are just 151 exemplars in total. Because of the lack of information, the splitting percentage of cross validation and test subset should be higher than the sugested value between 10%-20%. In this thesis, the splitting percentages are; 50% for training subset, 25% for cross validation subset and 25% for testing subset. However, the *veh* dataset, in the opposite of *tae* dataset, has more information as 846 exemplars. For this dataset, the same splitting percentages applied for performance comparison.

Momentum, conjugate-gradient and Levenberg-Marquardt algorithms are compared to choose the algorithms that minimize the error performance. Minimum average MSE values for both conjugate-gradient and Levenberg-Marquardt algorithms produce better values compared to momentum but with high standard deviation , these algorithms do not classify with high true classificiton rate.

As mean accuracy performances of learning algorithms are compared, for the basic ANN design for *tae* dataset is 56-4-3, not much difference seen betweeen each other. Conjugate-gradient and Levenberg-Marquardt algorithms make true classification approximately of 60% of momentum learning algortihm compared as the training subset data and approximately of 70% compared as the cross validation subset data.

For *tae* dataset, by comparing all the momentum levels examining the training and cross validation mean accuracy values, optimum momentum rate is 0.7. As comparing the training and cross validation subset's mean accuracy values, there seems not much difference between the selected performance values. One hidden layered ANN and three hidden layered ANN show nearly the same mean accuracy values. The aim is to simplify the network, so one hidden layered topology is chosen. By implementing the incremental algortihm, for one hidden layered ANN as the determined topology, 14 neurons minimize the average values of minimum MSEs for *tae* dataset.

veh dataset has opposing characteristics of attribute information compared to *tae* dataset. There are 18 attributes which are all numerical. All these attributes provide information about a vehicle's feature selection. This dataset has a total of 846 exemplars exist. 423 exemplars are obtained for training subset, 212 exemplars for cross validation and 211 exemplars for testing subset. Because of having a big dataset, it takes time to train the dataset and to converge to a minima.

Comparing the learning algorithms, momentum learning produces statistically consistent values such as low level standard deviation and mean compared to conjugate-gradient and Levenberg-Marquardt algorithms.

As the number of true classified exemplars increases, *r* values and correct percent values start to increase, as well.

For one hidden layered ANN topology for *veh* dataset, at least two class attributes have high misclassification rates. That means the weight matrix of the hidden layer does not give enough response for the classification. At that point, the number of hidden layers should be argued. As the misclassification occurs for any attribute, the MSE values for these attributes are expected to be high, and opposite of that situaiton, the *r* and percent correct values to be low.

Conjugate-gradient algorithm gets higher MSE values and that causes an increment seen in average MSEs which is not supposed to be. The standard deviation and the average of minimum MSEs of Levenberg-Marquardt are nearly half the value of standard deviation of conjugate-gradient algorithm. Examining the MSE statistics of momentum learning and Levenberg-Marquardt algorithm learning; the standard deviation values of training and cross validation for momentum learning are much less. The other reason for selecting the momentum learning is the very fast convergence of the Levenberg-Marquardt algorithm. It may be not useful for the datasets that have more categorical attributes that may cause overfitting of the data.

Mean accuracy values of learning algorithms show very slight differences between each other.

The optimum momentum rate for *veh* dataset is 0.9 but the mean accuracy values for β=0.5 and β=0.7 are very close to each other. Higher momentum rate helps to converge to minima in less processing time.

The number of hidden layer size is important up to the required level. As stated before that, for *veh* database, one hidden layered ANN is not suitable for classification as at least two attributes have high misclassification. But increasing the number of hidden layer causes a big decrease in the training and cross validation subsets' mean accuracy values.

Datasets with more numerical attributes have more sense to number of hidden layers.

Optimum number of processing elements of the hidden layers is obtained from the incremental algorithm. Doing the experimantation, 29 neurons for hidden 1 layer and 10 neurons for hidden 2 layer is calculated for the minimum average MSE value. In this experiment, there is no upper limit for the neurons but the lower limit should be equal to at least the number of attributes that are classified. In that situation, the network topology is 18-29-10-4.

During the parameter optimization; to identify the neural network parameters, the evolving mechanism is used such as mutation and crossover. Genetic algoritms are used to determine the number of neurons in the hidden layers, the momentum, and the learning rates for minimizing the time and effort required to find the optimal architecture and parameters of the back-propagation algorithm.

Crossover values (0.1, 0.3, 0.5, 0.7, 0.9) and mutation values (0.05, 0.1, 0.3, 0.5) are put into the genetic algorithm and for each level, the optimized EANN parameters and also the statistical values have been determined. The performance values; minimum MSE and average MSE are compared with the basic ANN statistical values which has been defined before manually.

For tae dataset, at low mutation levels; for higher crossover values, one point crossover, arithmetic and heuristic crossover operators perform better. Especially, one point crossover operator shows better minimum MSE at all levels of minimum MSE. Two point crossover shows little worse valued increments as the crossover value decreases but the worst crossover operator is the arithmetic crossover operator that is not suitable for both low levels of crossover and mutation.

As the mutation operator value starts to increase, one point operator is not able to reach better MSE statistics.As a result, only the heuristic crossover operator, because of the ability of randomness, gets better results than the basic ANN topology.

At the mutation rate is 0.5, two point and heuristic crossover operators show better results at high and low crossover values. One point crossover has an opposite manner to the mutation rate.

For veh database, at low levels of mutation; one point, arithmetic and heuristic crossover operators perform well through high crosover values. But decreasing the crossover value causes the MSE performance values to get worse immediately. As mutation increases, no further improvement on MSE performance values can be seen at any crossover operator. As the number of numerical attributes increases, the ability to perform better solutions at high mutation rates decreases.

CHAPTER SIX
CONCLUSION AND FUTURE RESEARCH

In this thesis, a systematic approach to automating the design of neural networks for classification through the use of evolutionary algorithms is presented. This study shows that evolutionary algorithms are used to evolve the optimal number of hidden neurons and weights required by neural networks for good classification accuracy. The selected parameters can be used effectively to classify given dataset with optimized parameters of the neural network compared to traditional (backpropagation) neural networks.

Training neural networks using genetic algorithm based evolutionary techniques has been proposed as a new algorithm to evolve the near optimal number of hidden neurons and weights required by neural networks for good classification accuracy. In addition, the self-evolving version of genetic algorithm based neural network is able to automate the process of finding a suitable weight and hidden node matrixes through the generations. The performance of the proposed algorithms is greatly enhanced by growing the neural networks at different rates based on a Gaussian distribution thus avoiding being trapped in local optima.

The proposed algorithms are tested with real-world problems and results from experiments show that evolutionary neural networks are able to evolve networks with high classification accuracy and low architecture complexity for all problems. The performances of learning algorithms are comparable to each other; however, the self-adaptive version has the advantage of not requiring the value of evolution parameters to be determined beforehand.

An interesting finding from the experimental results showed that though chromosomes are grown at every generation, it is the growth probability rather than the generation number that has a greater influence on the mean number of hidden neurons. This prevents the number of hidden neurons to grow too large when a large number of generations are used.

6.1 Contributions

- Neural networks are a non-symbolic approach to classification. They have the ability to generalize large numbers of exemplars into classes, and to learn from a presentation of datasets given.
- A multilayer perceptron neural network trained with a genetic algorithm, is suitable for discrimination and modelling of strongly nonlinear classification problems.
- Evolutionary algorithms do not make any hypothesis either on the data (non-parametric method) or on the transfer functions that can be used in the neurons, or on the error function.
- In relation to a multilayer perceptron neural network trained with back propagation, evolutionary neural network overcomes the problems that arise from the lack of continuity in the error functions for classification problems and allows one to consider a priori probabilities and loss functions in the discrimination and / or modelling problems. A good consequence of this is that the user need not define (either directly or indirectly) any threshold to construct the decision rule or the model box. Therefore, the method is highly respectful with the training data and with the information they contain.
- Evolutionary neural network based on genetic algorithm improves its efficiency with respect to dimension of the problem.
- Evolutionary neural network is applied to various well known and claassical examples and also to some complex classification problems. The comparison made with the results obtained by other classification techniques (parametric or not) shows similar efficiency in the data tested. The main result is its ability to simultaneously optimize sensibility and specificity in class-modelling problems, as can be appreciated in the cases considered.
- The result of this study indicates that the EAs to train backpropagation neural network yields very high accuracy and much improved execution time. The results of this research are expected to be applied to a wide

variety of applications to improve the accuracy and execution time of classification problems with multidimensional input patterns.

- The genetic operators; crossover and mutation presented to the neural network. By the help of these genetic operators, evolutionary neural network is structured. As seen from the results of each crossover and mutation rate; crossover rate between 0.7 and 0.9 produce considerably good results for EANN classification. On the contrary, low mutation rates are much more efficient for classification.
- The selection of crossover type is important. Two point and artihmetic crossover types show much more sense to mutation rate. As the mutation rate starts to increase in EANN, the ability to classify the dataset gets worse than the backpropagation neural network results.
- The learning algorithms: momentum, conjugate-gradient and Levenberg-Marquardt for the neural network are selected. Momentum and Levenberg-Marquardt algorithms helps the neural network learn more quick. Minimum and average MSE values show approximately the same results. As Levenberg-Marquardt algorithm is a second-order approxmiator, it learns 4-5 times faster than the other algorithms. But this quickness helps the neural network to overfit the data as well.
- The number of hidden layers and the number of hidden nodes in a layer play an important role in complexification of nonlinear mapping. As the neural network complexity grows, the computation time gets much longer, but also overfitting occurs occasionally which means that learning stops for the evolutionary neural network.
- The attributes of the dataset are important for classification accuracy. The more categorical attribute of a dataset means bigger vector calculations needed to classify the dataset. At that point, the true classification rate is not expected to be high. On the contrary, the more numerical attribute of a data means much more ability and information needed to classify the dataset. The true classification rate is expected have a success of getting high values.

- The design of the neural network is important because the work load and computation time rises exponentially with the size of the system.The optimum number of hidden layer size is obtained form the MSE statistical values and the mean accuracy performance value. For each possible matching of these genetic operators, the network parameters related to genetic results, the number of processing elements, stepsize and momentum rates are optimized. The MSE statistical values are tabled and the performance differences are put into percentage to be compared.

6.2 Future Research

Evolutionary algorithms (EAs) help to solve the manual design process problem of neural networks causing inaccurate run time. To get rid of this problem; HyperNEAT is a good solution which is a neuroevolution method that evolves artificial neural networks through an evolutionary algorithm can be used further on for designing all the EANN structure. It is extended from a prior neuroevolution algorithm called NeuroEvolution of Augmenting Topologies (NEAT). HyperNEAT is based on a theory of representation that hypothesizes that a good representation for an artificial neural network should be able to describe its pattern of connectivity compactly. This kind of description is called an *encoding*.

The encoding in HyperNEAT, called compositional pattern producing networks (CPPNs), is designed to represent patterns with regularities such as symmetry, repetition, and repetition with variation. Thus HyperNEAT is able to evolve neural networks with these properties. The main implication of this capability is that HyperNEAT can efficiently evolve very large neural networks that look more like neural connectivity patterns in the brain (which are repetitious with many regularities, in addition to some irregularities) that are generally much larger than what prior approaches to neural learning could produce.

It evolves the connectivity pattern for a neural network with a particular substrate geometry. It actually sees the geometry of its inputs (and outputs) and can exploit that geometry to significantly enhance learning. To put it more technically, HyperNEAT computes the connectivity of its neural networks as a function of their geometry. By automating the design process of evolutionary artificial neural networks, more accurate true classification rate values can be gained. (Stanley, 2009)

REFERENCES

Aitkenhead, M.J., & Aalders, I.H. (2008). Classification of Landsat Thematic Mapper imagery for land cover using neural networks. *International Journal of Remote Sensing,* 29 (7), 2075-2084.

Alsultanny, Y.A., & Aqel, M.M. (2003). Pattern recognition using multilayer neural-genetic algorithm. *Neurocomputing*, 51, 237–247.

Alkaya, A., & Bayhan, G.M. (2009). The Classification of a Simulation Data of a Servo System via Evolutionary Artificial Neural Networks. *International Conference on Intelligent Computing Proceedings*, 48-54.

Ang, J.H., Tan, K.C., & Al-Mamun, A. (2008). Training neural networks for classification using growth probability-based evolution. *Neurocomputing*, 71, 3493–3508.

Auer, P., Herbster, M., & Warmuth, M. (1996). Exponentially many local minima for single neurons. *Advances in Neural Information Processing Systems,* 8, MIT Press, Cambridge, MA, 316–322.

Barnard, E. (1992). Optimization for training neural nets. *IEEE Transactions on Neural Networks*, 3, 232–240.

Battiti, R. (1992). First and second-order methods for learning; between steepest descent and Newton's method. *Neural Computation.*, 4, 141–166.

Berardi, V.L., Patuwo, B.E., & Hu, M.Y. (2004). A principled approach for building and evaluating neural network classification models. *Decision Support Systems*, 38, 233– 246.

Berberoglu, S., Curran, P.J., Lloyd, C.D., & Atkinson, P.M. (2007). Texture classification of Mediterranean land cover. *International Journal of Applied Earth Observation and Geoinformation*, 9 (3), 322-334.

Blamire, P.A. (1996). The influence of relative sample size in training artificial neural networks. *International Journal of Remote Sensing*, 17 (1), 223–230.

Blanco, A., Delgado, M., & Pegalajar, M.C. (2001). A real-coded genetic algorithm for training recurrent neural networks. *Neural Networks*, 14 (1), 93-105.

Blum, A., (1992). *Neural networks in C++: an object-oriented framework for building connectionist systems*. John Wiley & Sons, Inc., 86-103.

Bourlard, H., & Wellekens, C.J. (1989). Links between markov models and multilayer perceptrons. *Advances in Neural Information Processing Systems*, Morgan Kaufmann, San Mateo, CA, 502–510.

Castillo, P. A., Carpio, J., Merelo, J.J., Prieto, A., Rivas, V., & Romero, G. (2000). Evolving multilayer perceptrons. *Neural Processing Letters*, 12, 115-127.

Castillo-Valdivieso, P.A., Merelo, J.J., & Prieto, A. (2002). Statistical analysis of the parameters of a neuro-genetic algorithm. *IEEE Transactıons On Neural Networks*, 13 (6), 1374-1394.

Dreiseitl, S., & Ohno-Machado, L. (2002). Logistic regression and artificial neural network classification models: a methodology review. *Journal of Biomedical Informatics*, 35 (5-6), 352-359.

Duin, R.P.W. (1996). A note on comparing classifiers. *Pattern Recognition. Letters*, 17, 529–536.

Fine, T.L. (1999). *Feedforward Neural Network Methodology*, Springer, New York, 129-194.

Finnoff, W., Hergert, F., & Zimmermann, H.G. (1993). Improving model selection by non-convergent methods, *Neural Networks* ,6, 771–783.

Flexer, A. (1996). Statistical evaluation of neural network experiments: Minimum requirements and current practice. *Proc. 13th Eur. Meeting Cybernetics Systems Research*, R. Trappl, Ed., 1005–1008.

Foody, G.M. (1995). Using prior knowledge in artificial neural network classification with a minimal training set. *International Journal of Remote Sensing*, 16 (2), 301–312.

Freitas, A. (2002). *A survey of evolutionary algorithms for data mining and knowledge*, NY :Springer Verlag.

Fujita, O. (1998). Statistical estimation of the number of hidden units for feedforward neural networks. *Neural Networks*, 11, 851–859.

Fukunaga, K., & Hayes, R.R. (1989). Effects of sample size in classifier design. *IEEE Trans. Pattern Anal. Machine Intell.*, 11, 873–885.

Giles, C.L., Miller, C.B., Chen, D., Chen, H.H., Sun, G.Z., & Lee, Y.C. (1992). Learning and extracting finite State automata with second-order recurrent neural networks. *Neural Computation*, 4 (3), 393-405.

Goldberg, D.E. (1989). *Genetic algorithms in search, optimization and machine learning*. Boston :Addison-Wesley Longman Publishing.

Hagan, M.T., & Henhaj, M. (1994). Training feedforward networks with the Marquardt algorithm. *IEEE Transactions on Neural Networks*, 5, 989–993.

Hancock, P.J.B. (1992). Genetic algorithms and permutation problems: A comparison of recombination operators for neural net structure specication. *International Workshop On Combinations of Genetic Algorithms and Neural Networks,* COGANN-92, 108-122.

Haykin, S. (1999). *Neural Networks: A comprehensive foundation* (2nd ed.). NJ: Prentice-Hall, Englewood Cliffs.

Hintz-Madsen, M., Hansen L.K., Larsen J., Pedersen, M.W., & Larsen, M. (1998). Neural classifier construction using regularization, pruning and test error estimation. *Neural Networks*, 11, 1659–1670.

Hung, M.S., Hu, M.Y., Patuwo, B.E., & Shanker, M. (1996). Estimating posterior probabilities in classification problems with neural networks. *International Journal of Computational Intelligence and Organizations* ,1 , 49– 60.

Jenkins, W.M. (2006). Neural network weight training by mutation. *Computers and Structures*, 84, 2107–2112.

Kalman, B.L., & Kwasny, S.C. (1992). Why tanh: choosing a sigmoidal function. *Neural Networks*, International Joint Conference, 4, 578-581.

Kavzoglu, T., & Mather, P.M. (2003). The use of backpropagating artificial neural networks in land cover classification. *International Journal of Remote Sensing*, 24 (23), 4907-4938.

Kimes, D., Gastellu-Etchegorry, J., & Esteve, P. (2002). Recovery of forest canopy characteristics through inversion of a complex 3D model. *Remote Sensing of Environment* ,79, 320– 328.

Lang, K.J., Waibel, A.H., & Hinton, G.E. (1990). A time-delay neural network architecture for isolated word recognition. *Neural Networks*, 3 (1), 33–43.

Lewicki, M.S. (1994). Bayesian modeling and classification of neural signals, *Neural Comput.*, 6, 1005–1030.

Liu, Z., Liu, A., Wang, C., & Niu, Z. (2004). Evolving neural network using real coded genetic algorithm (GA) for multispectral image classification. *Future Generation Computer Systems*, 20 (7), 1119-1129.

MacKay, D.C. (1992). Bayesian interpolation, *Neural Comput.*, 4, 415–447.

MacKay, D.C. (1992). A practical Bayesian framework for backpropagation networks. *Neural Comput.*, 448–472.

Mangal, M., & Singh, M.P. (2007). Analysis of pattern classification for the multidimensional parity-bit-checking problem with hybrid evolutionary feed-forward neural network. *Neurocomputing*, 70, 1511-1524.

Mehrotra, K., Mohan, C.K., & Ranka, S. (1997). *Elements of Artificial Neural* Networks, MIT Press, Cambridge, MA.

Moody, J. & Utans, J. (1995). Architecture selection strategies for neural networks: Application to corporate bond rating prediction. *Neural Networks in the Capital Markets*, 277–300.

Morgan, N., & Bourlard, H. (1990). Generalization and parameter estimation in feedforward nets: some experiments. *Advances In Neural Information Processing Systems*, 2, 630–637.

Muller, P., & Insua, D.R. (1998). Issues in Bayesian analysis of neural network models. *Neural Comput*ing, 10, 749–770.

Murata, N., Yoshizawa, S., & Amari, S. (1992). Learning curves, model selection and complexity of neural networks. *Advances in Neural Information Processing Systems,* 5, 607-614.

Murata, N., Yoshizawa, S., & Amari, S. (1994). Network information criterion determining the number of hidden units for artificial neural network models. *IEEE Transactions on Neural Networks,* 5, 865–872.

Nedeljkovic, V. (1993). A novel multilayer neural networks training algorithm that minimizes the probability of classification error. *IEEE Transactions on Neural Networks*, 4, 650–659.

Pal, M., & Mather, P.M. (2003). An assessment of effectiveness of decision tree methods for land cover classification. *Remote Sensing of Environment*, 86 (4), 554–565.

Palmes, P.P., & Usui, S. (2005). Robustness, evolvability, and optimality of evolutionary neural networks. *BioSystems*, 82, 168–188.

Papoulis, A. (1991). *Probability, random variables, and stochastic processes,*. (3rd ed). New York: McGraw-Hill.

Pendharkar, P.C. (2001). An empirical study of design and testing of hybrid evolutionary–neural approach for classification. *Omega*, 29, 361-374.

Prechelt, L. (1996). A quantitative study of experimental evaluation of neural network algorithms: Current research practice, *Neural Networks*, 9 (3), 457–462.

Prechelt, L. (1994). PROBEN1: a set of neural network benchmark problems and benchmarking rules. *Technical Report,* 21/94, Department of Informatics, University of Karlsruhe, Germany.

Raudys, S. (1998). Evolution and generalization of a single neurone: Complexity of statistical classifiers and sample size considerations. *Neural Networks*, 11, 297–313.

Raudys, S.J., & Jain, A.K. (1991). Small sample size effects in statistical pattern recognition: Recommendations for practitioners. *IEEE Trans. Pattern Anal. Machine Intell.*, 13, 252–264.

Richard, M.D., & Lippmann, R.P. (1991). Neural network classifiers estimate Bayesian a-posteriori probabilities. *Neural Computation* , 3 (4), 461–483.

Rocha M., Cortez P., & Neves, J. (2007). Evolution of neural networks for classification and regression. *Neurocomputing*, 70, 2809-2816.

Roy, A., Kim, S., & Mukhopadhyay, S. (1993). A polynomial time algorithm for the construction and training of a class of multilayer perceptrons. *Neural Networks*, 6, 535–545.

Ruck, D.W., Rogers, S.K., Kabisky, M., Oxley, M.E., & Suter, B.W. (1990). The multilayer perceptron as an approximation to a Bayes optimal discriminant function. *IEEE Transactions on Neural Networks*, 2 (1), 296–298.

Salzberg, S.L. (1997). On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining Knowl. Disc.*, 1, 317–328.

Schissmann, W., Joost, M., & Werner, R. (1993). Comparison of optimized backpropagation algorithms. *Proceedings of the European Symposium on Artificial Neural Networks*, Brussels, Belgium, 97–104.

Schmittlein, D.C., Kim, J., & Morrison, D.G. (1990). Combining forecasts: Operational adjustments to theoretically optimal rules. *Management Science*, 36 (9), 1044-1056.

Shoemaker, P.A. (1990). A note on least-squares learning procedures and classification by neural networks. *IEEE Transactions on Neural Networks*, 2 (1), 158–160.

Siebel, N.T., Krause, J., & Sommer, G. (2007). Efficient learning of neural networks with evolutionary algorithms. *Lecture Notes in Computer Science*, 4713/2007, 466-475.

Stanley, K. (2009). The Hybercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT) Users Page, retrieved September 12, 2009 from http://eplex.cs.ucf.edu/hyperNEATpage/HyperNEAT.html

Stepniewski, S.W., & Keane, A.J. (2006). Topology design of feedforward neural networks by genetic algorithms. *Lecture Notes in Computer Science*, 1141/1996, 771-780.

Teaching Assistant Evaluation Dataset. (1997). Retrieved August 6, 2009, from http://archive.ics.uci.edu/ml/machine-learning-databases/tae/

Vehicle Silhouettes (Statlog) Dataset. (1987). Retrieved August 24, 2009, from http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/vehicle/

Wan, E.A. (1990). Neural network classification: a Bayesian interpretation. *IEEE Transactions on Neural Networks*, 1 (4), 303–375.

Wang, Z., Massimo, C.D., Tham, M.T., & Morris, A.J. (1994). A procedure for determining the topology of multilayer feedforward neural networks. *Neural Networks*, 7, 291–300.

Wang, C., & Principe, J.C. (1999). Training neural networks with additive noise in the desired signal. *IEEE Transactions on Neural Networks*, 10 (6), 1511-1517.

White, H. (1989). Learning in artificial neural networks: A statistical perspective. *Neural Computation*, 1 , 425–464.

Yao, X. (1999). Evolving artificial neural networks. *Proceedings of IEEE International Conference on Evolutionary Computation*, 87 (9), 670-675.

Yuan, J.L., & Fine, T.L. (1998). Neural-network design for small training sets of high dimension. *IEEE Transactions on Neural Networks*, 9, 266–280.

Zhang, G.P. (2000). Neural networks for classification: A survey. *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and reviews*, 30 (4), 451-462.

Zopounidis, C., & Doumpos, M. (2002). Multicriteria classification and sorting methods: A literature review. *European Journal of Operational Research*, 138 (2), 229-246.

APPENDICES

APPENDIX 1

Raw Data for *tae* dataset

(S)ENG	(S)INST	(S)CRS	(S)SMSTR	CLSSIZE	(S)CLSATTR
1	23	3	1	19	***3***
2	15	3	1	17	***3***
1	23	3	2	49	***3***
1	5	2	2	33	***3***
2	7	11	2	55	***3***
2	23	3	1	20	***3***
2	9	5	2	19	***3***
2	10	3	2	27	***3***
1	22	3	1	58	***3***
2	15	3	1	20	***3***
2	10	22	2	9	***3***
2	13	1	2	30	***3***
2	18	21	2	29	***3***
2	6	17	2	39	***3***
2	6	17	2	42	***2***
2	6	17	2	43	***2***
2	7	11	2	10	***2***
2	22	3	2	46	***2***
2	13	3	1	10	***2***
2	7	25	2	42	***2***
2	25	7	2	27	***2***
2	25	7	2	23	***2***
2	2	9	2	31	***2***
2	1	15	1	22	***2***
2	15	13	2	37	***2***
2	7	11	2	13	***2***
2	8	3	2	24	***2***
2	14	15	2	38	***2***
2	21	2	2	42	***1***
2	22	3	2	28	***1***
2	11	1	2	51	***1***
2	18	5	2	19	***1***
2	13	1	2	31	***1***
1	13	3	1	13	***1***
2	5	2	2	37	***1***
2	16	8	2	36	***1***
2	4	16	2	21	***1***
2	5	2	2	48	***1***
2	14	15	2	38	***1***
1	23	3	1	19	***3***
2	15	3	1	17	***3***
1	23	3	2	49	***3***
1	5	2	2	33	***3***
2	7	11	2	55	***3***
2	23	3	1	20	***3***
2	9	5	2	19	***3***
2	10	3	2	27	***3***
1	22	3	2	58	***3***
2	15	3	1	20	***3***

(S)ENG	(S)INST	(S)CRS	(S)SMSTR	CLSSIZE	(S)CLSATTR
2	10	22	2	9	***3***
2	13	1	2	30	***3***
2	18	21	2	29	***3***
2	6	17	2	39	***3***
2	6	17	2	42	***2***
2	6	17	2	43	***2***
2	7	11	2	10	***2***
2	22	3	2	46	***2***
2	13	3	1	10	***2***
2	7	25	2	42	***2***
2	25	7	2	27	***2***
2	25	7	2	23	***2***
2	2	9	2	31	***2***
2	1	15	1	22	***2***
2	15	13	2	37	***2***
2	7	11	2	13	***2***
2	8	3	2	24	***2***
2	14	15	2	38	***2***
2	21	2	2	42	***1***
2	22	3	2	28	***1***
2	11	1	2	51	***1***
2	18	5	2	19	***1***
2	13	1	2	31	***1***
1	13	3	1	13	***1***
2	5	2	2	37	***1***
2	16	8	2	36	***1***
2	4	16	2	21	***1***
2	5	2	2	48	***1***
2	14	15	2	38	***1***
1	23	3	1	25	***3***
1	13	3	1	17	***3***
2	16	19	2	11	***3***
2	9	2	2	39	***3***
2	13	3	1	11	***3***
2	18	21	2	19	***3***
1	22	3	2	45	***3***
2	7	11	1	20	***3***
2	23	3	1	20	***3***
1	23	3	1	20	***3***
1	23	3	2	38	***3***
2	14	22	2	17	***3***
1	17	17	2	19	***3***
2	9	5	2	24	***3***
2	18	25	2	25	***3***
1	17	17	2	31	***3***
2	1	15	2	31	***3***
2	1	8	2	18	***2***
1	11	16	2	22	***2***
1	22	13	2	27	***2***
2	9	2	2	14	***2***
2	13	1	2	20	***2***

(S)ENG	(S)INST	(S)CRS	(S)SMSTR	CLSSIZE	(S)CLSATTR
1	6	17	2	35	*2*
2	23	3	1	20	*2*
1	23	3	1	20	*2*
2	6	17	2	37	*2*
1	22	3	2	15	*2*
2	20	2	2	25	*2*
2	23	3	2	10	*2*
2	20	2	2	14	*1*
1	23	3	2	38	*1*
2	13	1	2	29	*1*
2	10	3	2	19	*1*
2	7	11	2	30	*1*
1	14	15	2	32	*1*
2	8	3	2	27	*1*
2	12	7	2	34	*1*
2	8	7	2	23	*1*
2	15	1	2	66	*1*
2	23	3	2	12	*1*
2	2	9	2	29	*1*
2	15	1	2	19	*1*
2	20	2	2	3	*1*
2	13	14	2	17	*3*
2	9	6	2	7	*3*
1	10	3	2	21	*3*
2	14	15	2	36	*3*
1	13	1	2	54	*3*
1	8	3	2	29	*3*
2	20	2	2	45	*3*
2	22	1	2	11	*2*
2	18	12	2	16	*2*
2	20	15	2	18	*2*
1	17	18	2	44	*2*
2	14	23	2	17	*2*
2	24	26	2	21	*2*
2	9	24	2	20	*2*
2	12	8	2	24	*2*
2	9	6	2	5	*2*
2	22	1	2	42	*2*
2	7	11	2	30	*1*
2	10	3	2	19	*1*
2	23	3	2	11	*1*
2	17	18	2	29	*1*
2	16	20	2	15	*1*
2	3	2	2	37	*1*
2	19	4	2	10	*1*
2	23	3	2	24	*1*
2	3	2	2	26	*1*
2	10	3	2	12	*1*
1	18	7	2	48	*1*
2	22	1	2	51	*1*
2	2	10	2	27	*1*

APPENDIX 2
Representative Data For *veh* Dataset

18 Numerical Attributes and 1 categorical attribute: (S)VEHCL

Exemplar No	CMPCT	CRCLR	DIST_CRCLR	RAD_RAT	AX_ASP_RAT	MAX_LEN_ASP_RAT	SCAT_RAT	ELON	AX_RECT	MAX_LEN_RECT	SCA_VAR	SCA_VAR_MIN	GYRA	SKEW_MAJ	SKEW_MIN	KURT_MIN	KURT_MAJ	HOLLOW	(S)VEHCL
1	95	48	83	178	72	10	162	42	20	159	176	379	184	70	6	16	187	197	***van***
2	91	41	84	141	57	9	149	45	19	143	170	330	158	72	9	14	189	199	***van***
3	104	50	106	209	66	10	207	32	23	158	223	635	220	73	14	9	188	196	***saab***
4	93	41	82	159	63	9	144	46	19	143	160	309	127	63	6	10	199	207	***van***
5	85	44	70	205	103	52	149	45	19	144	241	325	188	127	9	11	180	183	***bus***
6	107	57	106	172	50	6	255	26	28	169	280	957	264	85	5	9	181	183	***bus***

.
.
.

The dataset matrix is 846x19

Exemplar No	CMPCT	CRCLR	DIST_CRCLR	RAD_RAT	AX_ASP_RAT	MAX_LEN_ASP_RAT	SCAT_RAT	ELON	AX_RECT	MAX_LEN_RECT	SCA_VAR	SCA_VAR_MIN	GYRA	SKEW_MAJ	SKEW_MIN	KURT_MIN	KURT_MAJ	HOLLOW	(S)VEHCL
841	93	34	66	140	56	7	130	51	18	120	151	251	114	62	5	29	201	207	***opel***
842	93	39	87	183	64	8	169	40	20	134	200	422	149	72	7	25	188	195	***saab***
843	89	46	84	163	66	11	159	43	20	159	173	368	176	72	1	20	186	197	***van***
844	106	54	101	222	67	12	222	30	25	173	228	721	200	70	3	4	187	201	***saab***
845	86	36	78	146	58	7	135	50	18	124	155	270	148	66	0	25	190	195	***saab***
846	85	36	66	123	55	5	120	56	17	128	140	212	131	73	1	18	186	190	***van***