G60.89

Hybrid Soft Computing – Architecture Optimization and Applications

Ajith Abraham

MS (NTU, Singapore), B Tech (Hons)
http://ajith.softcomputing.net

A thesis submitted in fulfillment of
the requirements for the Degree of
Doctor of Philosophy

Gippsland School of Computing and Information Technology
Monash University, Australia

January 2002

Acknowledgements

I would like to express my greatest appreciation to Associate Professor Baikunth Nath for his guidance and encouragement during the doctoral degree course. As my supervisor, his insights, observations and suggestions helped me to establish the overall direction of the research and success of the work. I also thank him for providing me with the opportunity to come to Gippsland and study for my PhD degree.

I would like to render my sincere thanks to Associate Professor Saratchandran (Nanyang Technological University, Singapore) for the valuable discussions on various topics. I am grateful to Dr G. Beliakov (Deakin University, Australia), Dr N.S. Philip (Cochin University of Science and Technology, India), Dr S. Petrovic-Lazarevic (Monash University, Peninsula campus), Dr K. Coghill (Monash University, Caulfield campus) and Dr D. Steinberg (Salford Systems Inc, USA), for various discussions and cooperative work in our joint publications, which enabled me with more insights and to explore different hybrid systems and applications. I am also obliged to Associate Professor G. Lu (Head of School) and other faculty members of the Gippsland School of Computing and Information Technology (GSCIT) for their various efforts in creating such a fresh and pleasant research environment.

I would like to thank the Monash Graduate School and GSCIT for the Scholarships and PG travel grants which I received during my PhD studies.

Finally, I would like to dedicate this thesis to my family without their understanding, patience and support this work would not have been possible to complete in time.

January 2002
Ajith Abraham

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published, unpublished work of others has been acknowledged in the text, and a list of references is given.

[redacted]

Ajith Abraham January 2002

Abstract

Soft computing is a well-established paradigm consisting of artificial neural networks, fuzzy inference systems, approximate reasoning and derivative free optimization techniques such as evolutionary computation etc. Several adaptive hybrid soft computing architectures have in recent years been developed for solving complicated real world problems. The hybridization aims at overcoming limitations of individual techniques through fusion of different techniques. Many of these approaches use the combination of different knowledge representation schemes, decision-making models, learning strategies and optimization techniques to solve a computational task. This thesis investigates the optimization of artificial neural networks and fuzzy inference systems using a combination of evolutionary algorithms and local search techniques.

We explored the performance of neural network supervised learning paradigms using first order and second order error information of the three popular chaotic time series. We implemented a Meta-Learning Evolutionary Artificial Neural Network (MLEANN) algorithm based on a hierarchical search process combining global search and local search procedures. Performance evaluation was made with conventionally designed neural networks using standard learning algorithms, cutting angle method of global optimization, Mamdani and Takagi-Sugeno neuro-fuzzy systems and multi variate adaptive regression splines.

We examined the different adaptation techniques for designing fuzzy inference systems. The different adaptation techniques using neural network learning algorithms and evolutionary computation were presented. We also compared the performance of some integrated neuro-fuzzy models using Mackey Glass time series. We further illustrated how neuro-fuzzy systems are implemented in practice. We used a concurrent neuro-fuzzy model for a stock market trend prediction and used Mamdani and Takagi-Sugeno integrated neuro-fuzzy models for modeling three real world problems. Performances of the neuro-fuzzy models were compared with different neural network learning techniques using 1st order and 2nd order error information.

Finally, we present Evolutionary Neuro-Fuzzy Systems (EvoNF) - a framework for optimization of fuzzy inference systems using neural network learning and evolutionary computation. Architecture of the evolutionary framework is presented and the representation of each layer of the hierarchical search process is discussed. We evaluated the performance of three different types of learning methods combining evolutionary algorithms and gradient descent technique. Empirical results were compared with MLEANN approach and Takagi-Sugeno neuro-fuzzy system.

Empirical results of the different optimized hybrid architectures clearly reveal the efficiency of the proposed algorithms at the expense of some trade off in computational cost.

Author's Publications

[1] Ajith Abraham and Baikunth Nath, A Neuro-Fuzzy Approach for Forecasting Electricity Demand in Victoria, Applied Soft Computing Journal, Elsevier Science, Volume 1(2), pp. 127-138, September 2001.

[2] Ajith Abraham and Baikunth Nath, Meta-Learning Evolutionary Artificial Neural Networks, Elsevier Neurocomputing Journal, October 2001 (Under review).

[3] Ajith Abraham, Baikunth Nath and P Saratchandran, Soft Computing and Computer Aided Reliability Engineering -A Synergism for Online Monitoring of Electronic Systems, Applied Soft Computing Journal, Elsevier Science, June 2001 (Under review).

[4] Ajith Abraham and Baikunth Nath, Neuro-Fuzzy Systems for Automation of Reactive Power Control, IEEE Transactions on Power Systems, August 2001 (Under review).

[5] Ajith Abraham and Ninan S Philip, Soft Computing Models for Weather Prediction, Transactions of The Society for Computer Simulation International July 2001(Under review).

[6] Ajith Abraham and Baikunth Nath, Is Evolutionary Design the Solution for Optimising Neural Networks? In Proceedings of Fifth International Conference on Cognitive and Neural Systems (ICCNS 2001), Published by Boston University Press, Boston, USA, 2001.

[7] Ajith Abraham, Ninan S Philip and Babu Joseph, Will We Have a Wet Summer? Long-term Rain Forecasting Using Soft Computing Models, Publication of the Society for Computer Simulation International, Kerckhoffs E J H & Snorek M (Editors), ISBN 1-56555-225-3, Prague, Czech Republic, pp. 1044-1048, 2001.

[8] Ajith Abraham and Baikunth Nath, ALEC - An Adaptive Learning Framework for Optimising Artificial Neural Networks, Computational Science, Springer-Verlag Germany, LNCS 2074, Vassil N Alexandrov et al (Editors), ISBN 3-540-42233-1, San Francisco, USA, pp. 171-180, 2001.

[9] Ajith Abraham and Dan Steinberg, MARS: Still an Alien Planet in Soft Computing? Computational Science, Springer-Verlag Germany, LNCS 2074, Vassil N Alexandrov et al (Editors), ISBN 3-540-42233-1, San Francisco, USA, pp. 235-244, 2001.

[10] Ajith Abraham, Baikunth Nath and Mahanti P K, Hybrid Intelligent Systems for Stock Market Analysis, Computational Science, Springer-Verlag Germany, LNCS 2074, Vassil N Alexandrov et al (Editors), ISBN 3-540-42233-1, San Francisco, USA, pp. 337-345, 2001.

[11] Ajith Abraham and Baikunth Nath, IT Impact On New Millennium Manufacturing, Computer Integrated Manufacturing, Singh J, Lew S C and Gay R (Editors), (ISBN 981-04-1923-6 & ISBN 981-04-1996-1), Singapore, pp. 321-332, 2000.

[12] Ajith Abraham, Neuro-Fuzzy Systems: State-of-the-Art Modeling Techniques Connectionist Models of Neurons, Learning Processes, and Artificial Intelligence, Springer-Verlag Germany, LNCS 2084, Jose Mira and Alberto Prieto (Eds.), ISBN 3-540-42235-8, Granada, Spain, pp. 269-276, 2001.

[13] Ajith Abraham and Baikunth Nath, Hybrid Heuristics for Optimal Design of Neural Nets, Recent Developments in Soft Computing, Robert John and Ralph Birkenhead (Editors), Springer Verlag- Germany (ISBN 3 7908 13613), England, pp. (8-15), 2000.

[14] Ajith Abraham and Baikunth Nath, Optimal Design of Neural Nets Using Hybrid Algorithms, In Proceedings of The Sixth Pacific Rim International Conference on Artificial Intelligence (PRICAI 2000), Lecture Notes in Artificial Intelligence, Mizoguchi R & Slaney J (Editors), Springer Verlag- Germany, (ISBN 3 540 67925 1), Melbourne, pp.510-520, 2000.

[15] Ajith Abraham and Baikunth Nath, Evolutionary Design of Neuro-Fuzzy Systems - A Generic Framework, In Proceedings of The 4-th Japan-Australia Joint Workshop on Intelligent and Evolutionary Systems (JA2000 - Japan), Published by the National Defence Academy (Japan) and University of New South Wales (Australia), Akira Namatame et al (Editors), (ISBN 07317 0504 1), Japan, pp. 106-113, 2000.

[16] Ajith Abraham and Baikunth Nath, Connectionist Models for Intelligent Reactive Power Control, In Proceedings of The Australasian MATLAB Users Conference 2000, Melbourne, Online Proceeding published by CEANET Pty Ltd Australia, 2000.

[17] Ajith Abraham and Baikunth Nath, An Evolving Fuzzy Neural Network Model Based Reactive Power Control, Industrial Convergence through Computers, In Proceedings of The Second International Conference on Computers In Industry, ICCI 2000, Industrial Convergence through Computers, Published by The Bahrain Society of Engineers, Majeed A Karim et al (Editors), Bahrain, pp. 247-253, 2000.

[18] Ajith Abraham and Baikunth Nath, A Soft Computing Approach for Fault Prediction of Electronic Systems, In Proceedings of The Second International Conference on Computers in Industry, ICCI 2000, Industrial Convergence through Computers, Published by The Bahrain Society of Engineers, Majeed A Karim et al (Editors), Bahrain, pp. 83-91, 2000.

[19] Ajith Abraham and Baikunth Nath, Designing Optimal Neuro-Fuzzy Systems for Intelligent Control, In Proceedings of The Sixth International Conference on Control, Automation, Robotics and Vision, ICARCV 2000, Wang J L (Editor), (ISBN 981 0434456) Singapore, 2000.

[20] Ajith Abraham and Baikunth Nath, Evolutionary Design of Fuzzy Control Systems - An Hybrid Approach, In Proceedings of The Sixth International Conference on Control, Automation, Robotics and Vision, ICARCV 2000, Wang J L (Editor), (ISBN 981 0434456) Singapore, (05-08) 2000.

[21] Ajith Abraham, Rajkumar Buyya and Baikunth Nath, Nature's Heuristics for Scheduling Jobs in Computational Grids, In Proceedings of 8th IEEE International Conference on Advanced Computing and Communications, ADCOM2000, Sinha P S and Gupta R (Editors), (ISBN 007 0435 480), Tata McGraw-Hill Publishing Co. Ltd, New Delhi, India, pp. 45-52, 2000.

[22] Ajith Abraham and Baikunth Nath, Failure Prediction of Critical Electronic Systems in Power Plants Using Artificial Neural Networks, In Proceedings of First International

Power & Energy Conference, INT-PEC'99, Isreb M (Editor), (ISBN 0732 620 945), Australia, 1999.

[23] Ajith Abraham and Baikunth Nath, Artificial Neural Networks for Intelligent Real Time Power Quality Monitoring Systems, In Proceedings of First International Power & Energy Conference, INT-PEC'99, Isreb M (Editor), (ISBN 0732 620 945), Australia, 1999.

[24] Gleb Beliakov and Ajith Abraham, Global Optimization of Neural Networks Using a Deterministic Hybrid Approach, Proceedings of the First International Workshop on Hybrid Intelligent Systems, Adelaide, Australia, Springer-Verlag Germany, pp. 79-92, 2002 (in press).

Table of Contents

List of Tables

List of Figures

Chapter 1:Introduction

1.0 Hybrid Intelligent Systems

When the computers first appeared in the early fifties, we admired it as an artificial brain, and we thought that we were successful in creating a low level decision making cognitive machine. Researchers coined the term artificial intelligence and waited for many potential applications to evolve. In the sixties computers failed to pass the Turing test due to the low processing speed of the computers.

However, in spite of the evolution in digital computers, after several years we realized that the so-called artificial intelligence (AI) was indeed very artificial in nature. It can be argued with some conviction that an AI algorithm that cannot solve new problems in new ways is emphasizing the, "artificial" and not the "intelligence". The vast majority of the AI algorithms have nothing to do with learning. Last few decades have seen a new era of AI on emulating humans, either in their behavior or in their neurophysiology. Rather than viewing humans as the premier example of intelligence, a broader and potentially more beneficial perspective views this species simply as a product of evolution, a process that generally produces organisms of increasing intellect. Recognizing the connection between evolution and intelligence makes it possible to overcome the limitations of conventional artificial intelligence techniques, and indeed to evolve such systems and create machine intelligence [128].

Hybridization of different intelligent systems represents the most exciting fruit of artificial intelligence to date. Such systems are starting to be employed in everyday life, and these applications rank amongst the most complex computer systems ever built; never before has the expertise of the human factors [13]. The integration of different learning and adaptation techniques, to overcome individual limitations and achieve synergetic effects through hybridization or fusion of these techniques, has in recent years contributed to a large number of new intelligent system designs [12].

Soft computing introduced by Professor Lotfi Zadeh (University of California, Berkeley) [255] is oriented towards the analysis and design of intelligent systems. It is a well-established paradigm, where new theories with a sound biological understanding have been evolving to construct computationally intelligent hybrid systems consisting of artificial neural network, fuzzy logic, approximate reasoning and derivative free optimization methods such as evolutionary computation etc. Nevertheless, hybrid intelligent system is an open instead of conservative concept. That is, it is evolving those relevant techniques together with the important advances in other new computing methods [76] [235].

Several adaptive hybrid intelligent systems have in recent years been developed for modelling expertise, decision support, financial modeling, process control, mechatronics, robotics and complicated automation tasks etc [127] [155] [231]. Many of these approaches use the combination of different knowledge representation schemes, decision making models and learning strategies to solve a computational task [129]. This integration aims at overcoming limitations of individual techniques through hybridization or fusion of various techniques. It is well known that the intelligent systems, which can provide human like expertise such as domain knowledge, uncertain reasoning, and adaptation to a noisy and time varying environment, are important in tackling practical computing problems. In contrast with conventional artificial intelligence techniques which only deal with precision, certainty and rigor the guiding principle of hybrid soft computing is to exploit the tolerance for imprecision, uncertainty, low solution cost, robustness, partial truth to achieve tractability, and better rapport with reality. Table 1.1 summarizes the comparison of neural networks, fuzzy inference system, evolutionary algorithms, symbolic artificial intelligence (AI) and control theory [11] [93].

To realize intelligent systems in practice, a synthesis of various techniques is required. Figure 1.1 shows the synthesis of neural networks, fuzzy logic and evolutionary algorithms and their mutual interaction leading to different architectures. Each technique plays a very important role in the development of hybrid intelligent systems. Experience has shown that it is crucial for the design of hybrid intelligent systems to primarily focus on the integration and interaction of different techniques rather than merge different methods to create ever-new techniques. Techniques already well understood, should be applied to solve specific domain problems within the system. Their weakness must be addressed by combining them with complementary methods.

Table 1.1. Comparison of different intelligent systems with classical approaches†.

	Fuzzy system	Neural network	Evolutionary algorithms	Symbolic AI	Control theory
Mathematical model	SG	B	B	SB	G
Learning ability	B	G	SG	B	B
Knowledge representation	G	B	SB	G	SB
Expert knowledge	G	B	B	G	SB
Nonlinearity	G	G	G	SB	B
Optimization ability	B	SG	G	B	SB
Fault tolerance	G	G	G	B	B
Uncertainty tolerance	G	G	G	B	B
Real time operation	G	SG	SB	B	G

†Fuzzy terms used for grading are good (G), slightly good (SG), slightly bad (SB) and bad (B).

Artificial neural networks offer a highly structured architecture with learning and generalization capabilities, which attempts to mimic the neurological mechanisms of the brain. A neural network stores knowledge in a distributive manner within its weights; which have been determined by learning with known samples. The generalization ability for new inputs is then based on the inherent algebraic structure of the network. However, it is very difficult to incorporate human a priori knowledge into a neural network. This is mainly because the connectionist paradigm gains most of its strength from a distributed knowledge representation.

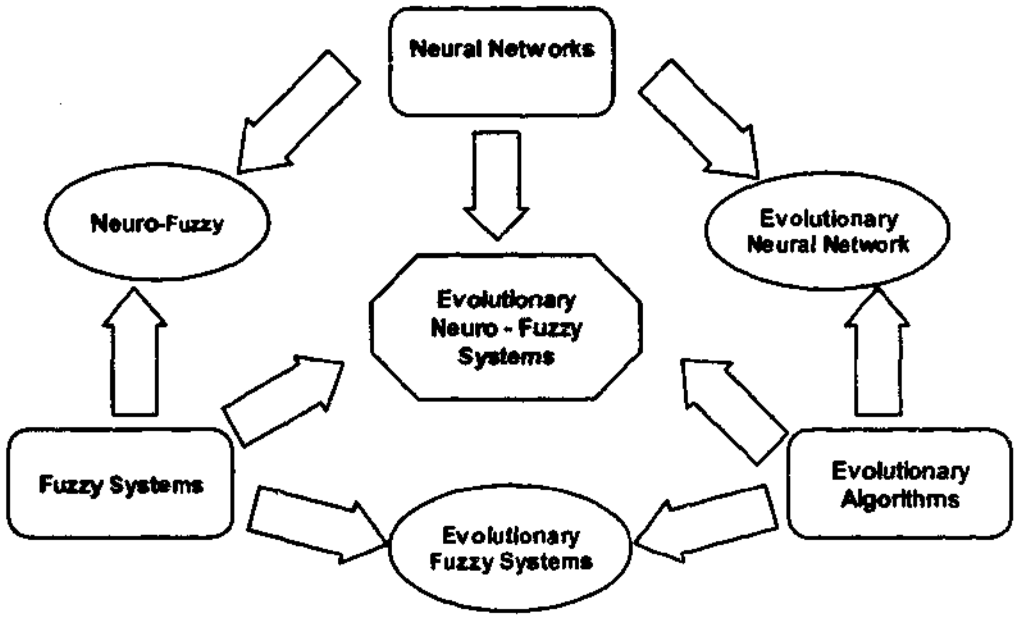

Neural Networks
Neuro-Fuzzy
Evolutionary Neural Network
Evolutionary Neuro - Fuzzy Systems
Fuzzy Systems
Evolutionary Algorithms
Evolutionary Fuzzy Systems

Figure 1.1 General framework for hybrid intelligent systems

In contrast, fuzzy systems exhibit complementary characteristics, offering a very powerful framework for approximate reasoning as it attempts to model the human reasoning process at a cognitive level. Fuzzy systems acquire knowledge from domain experts and this is encoded within the algorithm in terms of the set of if-then rules. Fuzzy systems employ this rule based approach and interpolative reasoning to respond to new inputs. The incorporation and interpretation of knowledge is straight forward, whereas learning and adaptation constitute major problems.

Usually grouped under the term evolutionary algorithms or evolutionary computation, we find the domains of genetic algorithms, evolution strategies, evolutionary programming, genetic programming and learning classifier systems. They all share a common conceptual base of simulating the evolution of individual structures via processes of selection, genetic operators, and reproduction. The processes depend on the perceived performance of the individual structures as defined by the environment (problem). These methods are fundamentally iterative generation and alteration processes operating on a set of candidate solutions that form a population. The entire population evolves towards better candidate solutions via the selection operation and genetic operators such as crossover and mutation. The selection operator decides which candidate solutions move on into the next generation, thus limits the search space. Referring to Figure 1.1, several hybrid combinations of neural networks, fuzzy systems and evolutionary computation could be explored.

1.1.Models Of Hybrid Systems

We broadly classify the various hybrid intelligent system architectures into 4 different categories based on the systems overall architecture: (a) Stand alone architectures (b) Transformational architectures (c) Hierarchical hybrid architectures and (d) Integrated hybrid architectures.

Stand Alone Architecture

Stand-alone models of hybrid systems consist of independent software components, which do not interact in anyway. Developing stand-alone systems can have several purposes. First they provide direct means of comparing the problem solving capabilities of different techniques with reference to a certain application. Running different techniques in a parallel environment permits a loose approximation of integration. Stand-alone models are often used to develop a quick initial prototype, while a more time-consuming application is developed. Figure 1.2 displays a stand-alone system where neural network and a fuzzy system are being used separately.

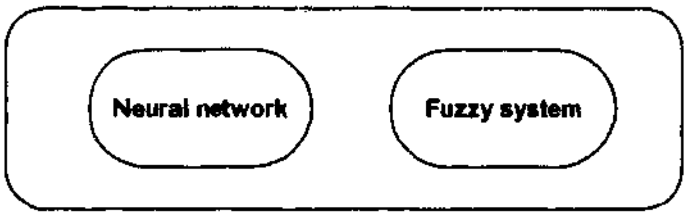


Figure 1.2. Stand –alone architecture

Some of the benefits are simplicity and ease of development using commercially available software packages. On the other hand, stand-alone techniques are not transferable; neither can support the weakness of the other technique.

Transformational Hybrid Architecture

In a transformational hybrid model, initially the system begins as one type of system and end up as the other. Determining which technique is used for development and which is used for delivery is based on the desirable features that the technique offers. Figure 1.3 shows the interaction between a neural network and an expert system in a transformational hybrid model [172]. Obviously, either the expert system is incapable of adequately solving the problem, or the speed, adaptability, and robustness of neural network is required. Knowledge from the expert system is used to set the initial conditions and training set for artificial neural network.

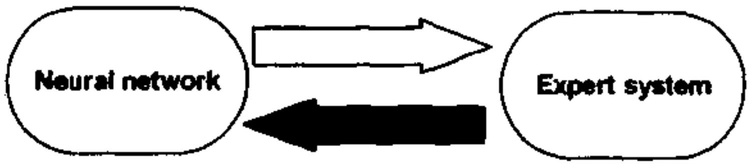


Figure 1.3. Transformational hybrid architecture

Transformational hybrid models are often quick to develop and ultimately require maintenance on only one system. Models can be developed suited to the environment and offer many operational benefits. Unfortunately, transformational models are significantly limited. Most of the developed models are just application oriented. For a different application, a totally new development effort might be required. A fully automated means of transforming an expert system to neural network and vice versa is required.

Hierarchical Hybrid Architectures

The architecture is built in a hierarchical fashion, associating a different functionality with each layer. The overall functioning of the model will depend on the correct functioning of all the layers. Figure 1.4 demonstrates a hierarchical hybrid architecture involving a neural network, evolutionary algorithm and a fuzzy system. Neural network uses an evolutionary algorithm to optimize its performance and the network output acts as a pre-processor to a fuzzy system, which then produces the final output. Poor performance in one of the layers will directly affect the final output.

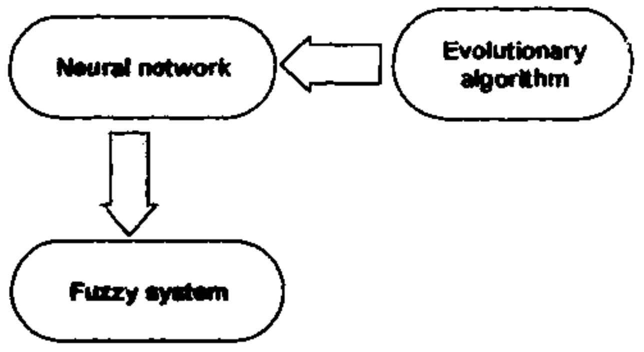


Figure 1.4. Hierarchical hybrid architectures

Integrated Hybrid Architectures

These models include systems, which combine different techniques into one single computational model. They share data structures and knowledge representations. In a truly integrated model the individual systems cannot be separated. This thesis deals with the various different integrated hybrid systems using neural networks, fuzzy systems and evolutionary algorithms. There are also several approaches to integrate hybrid systems. A simple approach is to put the various techniques on a side-by-side basis and focus on their interaction in the problem-solving task [124]. This method might allow integrating alternative techniques and exploiting their mutuality. Further more the conceptual view of the agent allows one to abstract from the individual techniques and focus on the global system behavior, as well as study the individual contribution of each component.

The benefits of fused architecture include robustness, improved performance and increased problem-solving capabilities. Finally, fully integrated models can provide a full range of capabilities such as adaptation, generalization, noise tolerance and justification. Fused systems have limitations caused by the increased complexity of the inter module interactions and specifying, designing, and building fully integrated models is complex.

1.3. Aim of the Thesis

The objective of this thesis is to investigate the representation of the various integrated hybrid systems using neural networks, fuzzy systems and evolutionary algorithms. The main contributions of this thesis are as follows:

1 Performance analysis of the different feedforward neural network supervised learning paradigms using first order and second order error information of the three popular chaotic time series [15]. To overcome the limitations we introduced the concept of meta-learning in artificial neural networks designed by evolutionary algorithms.

2 Development and implementation of Meta - Learning Evolutionary Artificial Neural Network (MLEANN) [15] [16] to optimize the neural network architecture, node transfer function, connection weights, different learning algorithms and its parameters. The performance of MLEANN is compared with another neural network global optimization approach [35], neuro-fuzzy systems [131] and Multivariate Adaptive Regression Splines (MARS) [17].

3 Performance analysis of different types of fuzzy inference systems to illustrate the role of the shape and quantity of membership functions per I/O variable, fuzzy operators, defuzzification method and the fuzzy inference method (eg. Mamdani, Takagi-Sugeno type etc.) [10].

4 Performance evaluation and technical analysis of different integrated neuro-fuzzy models [22].

4 To illustrate the cerebral quotient of neuro-fuzzy systems some practical applications of different types of neuro-fuzzy models are presented:

- A concurrent neuro-fuzzy model was used for stock market analysis [19]
- Integrated neuro-fuzzy model for the following practical applications

    ii Modeling electricity demand prediction in Victoria using a Mamdani fuzzy inference system [14].

    iii Automation of reactive power control using Takagi Sugeno and Mamdani fuzzy inference system [7].

    iv Weather forecast models using Mamdani fuzzy inference system [21].

5 Proposed and implemented a framework for Evolutionary design of Neuro-Fuzzy (EvoNF) systems. The proposed algorithm based on an adaptive evolutionary algorithm is

capable of adapting the membership functions, rule base, fuzzy operators, learning parameters and the fuzzy inference system [11]. Three different learning strategies for designing neuro-fuzzy systems are investigated.

- Combination of evolutionary algorithm and gradient descent (global search + local search)
- Pure evolutionary learning (equivalent to evolutionary fuzzy systems)
- Combination of evolutionary algorithm and gradient descent without fuzzy tuning

The developed three different neuro-fuzzy learning algorithms are applied to three chaotic time series and performances are evaluated. Performance evaluation and comparison with meta-learning evolutionary neural networks are also presented.

6. Scientific importance of the results obtained and some future research directions are also presented.

1.4. Organization of the Thesis

In Chapter 2, we present the fundamental theoretical aspects of artificial neural network learning paradigms namely backpropagation, conjugate gradient, quazi-Newton and Levenberg-Marquardt algorithms followed by some experimentations using three different chaotic time series to demonstrate the performance of the different learning algorithms when the architecture, node transfer functions etc. are changed.

Chapter 3 begins with the theoretical framework of the proposed meta-learning evolutionary artificial neural network algorithm based on a hierarchical evolutionary search process. We further illustrate the chromosome representation of the various layers and implementation details of the algorithm. Experiments were carried out on the chaotic time series and performance comparison was made with conventionally designed neural networks (pre-fixed architectures and node transfer functions) using standard learning algorithms, cutting angle method of global optimization of neural networks [35] and conventionally designed Mamdani and Takagi-Sugeno neuro-fuzzy systems and Multivariate Adaptive Regression Splines (MARS).

In Chapter 4, we present the concepts of fuzzy inference systems emphasizing on Mamdani and Takagi Sugeno fuzzy inference systems. A practical example was taken to demonstrate the importance of the shape of the membership functions, number of membership functions per input variable, fuzzy operators (T-norm and T-conorm) and the inference method itself. Adaptive framework for automatic optimal design of fuzzy inference method using evolutionary algorithms was also presented.

In Chapter 5, we review the different types of neuro-fuzzy systems. We presented the different types of cooperative neuro-fuzzy models followed by a concurrent neuro-fuzzy system with a demonstration using a practical example. We attempted to forecast the stock market trends using a concurrent neuro-fuzzy model implementing a Mamdani type neuro-fuzzy system. Different types of integrated neuro-fuzzy models were presented with some critical analysis and some empirical comparison of different neuro-fuzzy models using Mackey Glass chaotic time series are also presented towards the end of the chapter.

Chapter 6 focuses on 3 practical applications of integrated neuro-fuzzy systems. Performance of the neuro-fuzzy models are compared with different neural network learning techniques using 1st order and 2nd order error information.

In Chapter 7, we present the theoretical frameworks for evolutionary design of neuro fuzzy systems. Architecture of the evolutionary framework is presented and the functioning and representation of each layer is discussed. Three different learning algorithms were developed and experiments are carried out on the three chaotic time series. Empirical results were compared with evolutionary neural networks and conventionally designed neuro-fuzzy systems.

Finally in Chapter 8, conclusions and a number of topics for the future research in this direction are given.

Chapter 2: Neural Networks: Conventional Design Limitations

2.0 Introduction

The strong interest in neural networks in the scientific community is fueled by the many successful and promising applications especially to tasks of optimization [70], speech recognition [50], pattern recognition [43], signal processing [171], financial modeling [211], function approximation [242], control problems [4] [6] etc.

Even though artificial neural networks are capable of performing a wide variety of tasks, yet in practice sometimes they deliver only marginal performance. Inappropriate topology selection and learning algorithm are frequently blamed. There is little reason to expect that one can find a uniformly best algorithm for selecting the weights in a feedforward artificial neural network [237]. It is an NP-complete problem to find a set of weights for a given neural work and set of training examples to classify even two-thirds of them correctly [119] [134] [133]. In sum, one should be skeptical of claims in the literature on training algorithms that one being proposed is substantially better than most others. Such claims are often defended through some simulations based on applications in which the proposed algorithm performed better than some familiar alternative.

In this chapter, we review the state of art techniques of different neural network learning paradigms followed by some experimentation to demonstrate the difficulties in designing neural networks, which are smaller, faster and with a better generalization performance.

2.1 Artificial Neural Network Learning Algorithms

The artificial neural network (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, 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 our goal is not to reproduce it exactly but rather to construct a network function that generalizes well to new function values. We will try to address the problem of selecting the weights to learn the training set. The notion of closeness on the training set T is typically formalized through an error function of the form

$$\psi_T = \sum_{i=1}^{n} \left\| y_i - t_i \right\|^2 \qquad (2.1)$$

where y_i is the network output. Our 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 $\psi_T = \psi_T\ (w)$ is a function of w because $y = \eta$ depends upon the parameters w defining the selected network η. The objective function $\psi_T\ (w)$ 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 (tanh, sigmoid, logistic etc) are differentiable to arbitrary order through the chain rule of differentiation, which implies that the error is also differentiable to arbitrary order. Hence we are able to make a Taylor's series expansion in w for ψ_T. We shall first discuss the algorithms for minimizing ψ_T by assuming that we can truncate a Taylor's series expansion about a point w^o that is possibly a local minimum [84]. The gradient (first partial derivative) vector is represented by

$$g(w) = \nabla \psi_T \bigg| = \left[\frac{\partial \psi_T}{\partial w_i} \right] \qquad (2.2)$$

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

$$H(w) = H_{ij}(w) = \nabla^2 \psi_T(w) = \frac{\partial^2 \psi_T(w)}{\partial w_i \partial w_j} \qquad (2.3)$$

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

$$\psi_T(w) = \psi_T(w^0) + g(w^0)^T(w - w^0)^T + \frac{1}{2}(w - w^0)^T H(w^0)(w - w^0) + O(\| w - w^0 \|^2) \quad (2.4)$$

where $O(\upsilon)$ denotes a term that is of zero-order in small δ such that $\lim\limits_{\delta\to 0} \frac{O(\delta)}{\delta} = 0$.

If for example there is continuous derivative at w^0, then the remainder term is of order $\| w - w^0 \|^3$ and we can reduce (2.4) to the following quadratic model

$$m(w) = \psi_T(w^0) + g(w^0)^T(w - w^0) + \frac{1}{2}(w - w^0)^T H(w^0)(w - w^0) \quad (2.5)$$

Taking the gradient in the quadratic model of (2.5) yields

$$\nabla m = g(w^0) + H(w - w^0) \quad (2.6)$$

If we set the gradient $g = 0$ and solving for the minimizing w^* yields

$$w^* = w^0 - H^{-1}g \quad (2.7)$$

The model m can now be expressed in terms of minimum value of w^* as

$$\begin{aligned} m(w^*) &= m(w^0) + \frac{1}{2}g(w^0)^T H^{-1} g(w^0) \\ m(w) &= m(w^*) + \frac{1}{2}(w - w^*)^T H(w^*)(w - w^*) \end{aligned} \quad (2.8)$$

a result that follows from (2.5) by completing the square or recognizing that $g(w^*)=0$. Hence starting from any initial value of the weight vector, we can in the quadratic case move one step to the minimizing value when it exists. This is known as Newton's approach and can be used in the non-quadratic case where H is the Hessian and is positive definite [84].

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 confuse the analysis because we need to be assured that we are converging to the same local minimum as used in the expansion. How likely are we to encounter a sizable number of local minima? Empirical experience with training algorithm shows that different initialization yield different resulting networks. Hence the issue of many minima is a real one. According to Auer et al [30], a single node network with n training pairs and R^d inputs, could end up having $(\frac{n}{d})^d$ local minima. Hence not only multiple minima exist, but there may be huge numbers of them.

Different learning algorithms have their staunch proponents, who can always construct instances in which their algorithm performs better than most others. In practice, there are four types of optimization algorithms that are used to minimize $\Psi_T(w)$. The first three methods gradient descent, conjugate gradients and quasi-Newton are general optimization methods whose operation can be understood in the context of minimization of a quadratic error function. Although the error surface is surely not quadratic, for differentiable node functions it will be so in a sufficiently small neighborhood of a local minimum, and such an analysis provides information about the behavior of the training algorithm over the span of a few iterations and also as it approaches its goal. The fourth method of Levenberg and Marquardt [68] is specifically adapted to minimization of an error function that arises from a squared error criterion of the form we are assuming. Backpropagation calculation of gradient can be adapted easily to provide the information about the Jacobian matrix J needed for this method. A common feature of these training algorithms is the requirement of repeated efficient calculation of gradients.

2.1.1 Backpropagation Algorithm

Backpropagation provides an effective method for evaluating the gradient vector needed to implement the steepest descent, conjugate gradient, and quasi-Newton algorithms. BP 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. BP 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 $\{\Psi_T(w_k) = \Psi(k)\}$ converge to a small neighborhood of a good local minimum rather than the usually inaccessible global minimum.

$$\psi_T^* = min_{w \in W}\ \psi_T(w) \qquad (2.9)$$

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.

$$w_{k+1} = w_k - \alpha_k g_k \qquad (2.10)$$

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

$$\alpha_k^* = arg\ min_\alpha\ \psi(w_k + \alpha d_k) \qquad (2.11)$$

To carry out the minimization we use

$$\left.\frac{\partial \psi(w_{k+1})}{\partial \alpha}\right|_{\alpha=\alpha_k^*} = \left.\frac{\partial \psi(w_k + \alpha d_k)}{\partial \alpha}\right|_{\alpha=\alpha_k^*} = 0 \qquad (2.12)$$

To evaluate this equation, note that

$$\frac{\partial \psi(w_k + \alpha d_k)}{\partial \alpha} = g_{k+1}^T d_k \qquad (2.13)$$

and conclude that for optimal learning rate we must satisfy the orthogonality condition

$$g_{k+1}^T d_k = 0 \qquad (2.14)$$

When the error function is not specified analytically, then its minimization along d_k can be accomplished through a numerical line search for α_k or through numerical differentiation as noted herein. 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 as given in (2.5) and (2.6). In our current notation,

$$g_{k+1} - g_k = \alpha_k H d_k \tag{2.15}$$

Hence the optimality condition for the learning rate α_k derived from the orthogonality condition (2.14) becomes

$$\alpha_k^* = \frac{-d_k^T g_k}{d_k^T H d_k} \tag{2.16}$$

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

$$\alpha_k^* = \frac{-g_k^T M_k g_k}{g_k^T M_k H M_k g_k} \tag{2.17}$$

In the case of steepest descent for a quadratic error function, M_k is the identity and

$$\alpha_k^* = \frac{-g_k^T g_k}{g_k^T H g_k} \tag{2.18}$$

One can think of α_k^* as the reciprocal of an expected value of the eigen values $\{\lambda_i\}$ of the Hessian with probabilities determined by the squares of the coefficients of the gradient vector g_k expanded in terms of the eigen vectors $\{e_i\}$ of the Hessian.

$$\frac{1}{\alpha_k^*} = \sum_{i=1}^{p} q_i \lambda_i, \quad q_i = \frac{(g_k^T e_i)^2}{g_k^T g_k} \tag{2.19}$$

The algorithm, even in the context of a truly quadratic error surface and with line search, suffers from greed. The successive directions do not generally support each other in that after two steps; say, 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. To see this, note that

$$\psi(w) = \psi(w^*) + \frac{1}{2}(w - w^*)^T H (w - w^*) = \psi(w^*) + \tfrac{1}{2} g^T H^{-1} g \tag{2.20}$$

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

$$g_k = \left[\prod_{1}^{k}(I - \alpha_j H)\right] g_0 \qquad (2.21)$$

Hence for $k \geq p$, we can achieve $g_k = 0$ simply by choosing $\alpha_1, \ldots \alpha_p$ any permutation of $1/\lambda_1 \ldots\ldots 1/\lambda_p$, the reciprocals of the eigen values of the Hessian H; the resulting product of matrices is a matrix that annihilates each of the p eigen vectors and therefore any other vector that can be represented as their weighted sum. Of course, in practice, we do not know the eigen values and cannot implement this algorithm. However, this observation points out the distinction between optimality when one looks ahead only one step and optimality when one adopts a more distant horizon. Traditionally the step size is held at a constant value $\alpha_k = \alpha$. The simplicity of this approach is belied by the need to carefully select the learning rate. If the fixed step size is too large, then we leave ourselves open to overshooting the line search minimum, we may engage in oscillatory or divergent behavior, and we loose guarantees of monotone reduction of the error function ψ_T.If the step size is too small, then we may need a very large number of iterations T before we achieve a sufficiently small value of the error function. A variation on the constant learning rate is to adopt a deterministic learning rate schedule that varies the learning rate dependant on the iteration number.

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

$$\Delta_{k+1} = w_{k+1} - w_k = \beta\Delta_k - \alpha_k g_k \text{ for } k \geq 0 \qquad (2.22)$$

what we gain is a high frequency smoothing effect through the momentum term. The change in parameter vector depends not only on the current gradient g_{k-1} but also, in an exponentially decaying fashion (provided that $0 \leq \beta < 1$), on all previous gradients. If the succession of recent gradients has tended to alternate directions, then the sum will be relatively small and we will make only small changes in the parameter vector. This could occur if we are in the vicinity of a local minimum, successive changes would just serve to bounce us back and forth past the minimum. If, however, recent gradients tends to align, then we will make an even larger change 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 α is well chosen, then successive gradients will tend to be orthogonal and a weighted sum will not cancel itself out [214].

2.1.2 Conjugate Gradient Algorithm

The motivation behind the conjugate gradient algorithm is that we wish to iteratively select search directions (d_k) that are non-interfering in the sense that successive minimizations along these directions do not undo the progress made by previous minimizations. The search direction is selected in such a way that at each iteratively selected parameter value w_k, the current gradient g_k is orthogonal to all previous search directions $d_1,....d_{k-1}$. Hence, at any given step in the iteration, the error surface has a direction of steepest descent that is orthogonal to the linear subspace of parameters spanned by the prior search directions. Steepest descent merely assured us that the current gradient is orthogonal to the last search direction. If the error function $\{\Psi_T(w_k)\}$ is quadratic with positive definite Hessian H, choosing the search directions (d_i) to be H-conjugate and the α_i to satisfy (2.16) is equivalent to the orthogonality between the current gradient and the past search directions given by

$$(\forall i < k < p) d_i^T g_k = 0 \tag{2.23}$$

it is easily verified that conjugate directions (d_i) also form a linearly independent set of directions in weight space. If weight space has, dimension p then of course there can be only p linearly independent directions of vectors. Hence, it is possible to represent any point as a linear combination of no more than p of the conjugate directions, and in particular if w^* is the sought location of the minimum of the error function, then there exist coefficients such that

$$w^* - w_0 = \sum_{i=0}^{p-1} \alpha_i^* d_i \tag{2.24}$$

Thus if the error surface is quadratic with a positive definite Hessian, then selecting H-conjugate search directions and learning rates according to (2.16) guarantees a minimum in no more than p iterations. To be able to apply the method of conjugate gradients we must be able to determine such a set of directions and then solve for the correct coefficients. Conventional conjugate gradient algorithms use a line search to find the minimizing step and are initialized as follows

$$d_0 = g_0 \tag{2.25}$$

introducing a scaling β_k to be determined, and then iterate with the simple recursion

$$d_{k+1} = -g_{k+1+\beta_k} d_k \tag{2.26}$$

According to the conjugacy condition in (2.23) and the recursion of (2.26) yield

$$d_k^T H d_{k+1} = 0 = d_k^T H(-g_{k+1+\beta_k} d_k) \tag{2.27}$$

Solving yields the necessary condition that

$$\beta_k = \frac{d_k^T H g_{k+1}}{d_k^T H d_k} \tag{2.28}$$

Induction can be established that this recursive definition of conjugate gradient search directions does indeed yield a fully conjugate set when the error function is quadratic, although the derivation of (2.28) only established that d_k and d_{k+1} are conjugate. A version of the conjugate gradient algorithm that does not require line searches was developed by Moller and uses the finite difference method for estimating Hd_k. To monitor the sign of the product $d_k^T H d_k$, define δ by

$$\delta = d_k^T H d_k \tag{2.29}$$

Moller introduces two new variables, λ and $\overline{\lambda}$, to define an altered value of δ, $\overline{\delta}$. These variables are charged with ensuring that $\overline{\delta} > 0$. Although this does not affect the error surface, and the Hessian with the quadratic approximation will still suggest there is a maximum along the search direction, the method produces a step size that shows good results in practice. $\overline{\delta}$ is defined as follows

$$\overline{\delta} = \delta + (\overline{\lambda} - \lambda) d_k^T d_k \tag{2.30}$$

The requirement for $\overline{\delta} > 0$ gives a condition for $\overline{\lambda}$

$$\bar{\lambda} > \lambda - \frac{\delta}{d_k^T d_k} \qquad (2.31)$$

Moller then sets $\bar{\lambda} = 2(\lambda - \frac{\delta}{d_k^T d_k})$ to satisfy (2.31) and so ensures $\bar{\delta} > 0$. Substituting this in (2.30)

$$\bar{\delta} = -\delta + \lambda d_k^T d_k \qquad (2.32)$$

In order to get a good quadratic approximation of the error surface, a mechanism to raise or lower λ is needed when the Hessian is positive definite. Detailed step-by-step description can be found in [186].

2.1.3 Quasi - Newton Algorithm

If the error surface is purely quadratic, as per (2.7) we can solve the minimizing weight vector in a single step through Newton's method. This solution requires knowledge of the Hessian and assumes it constant and positive definite. We need a solution method that can take into account the variation of *H(w)* with *w*, knowing the fact that the error function is at best only approximately quadratic and removal from a local minimum the approximating quadratic surface is likely to have a Hessian that is not positive definite and the evaluation of true Hessian is computationally too expensive.

The quasi- Newton method addresses themselves to these tasks by first generalizing the iterative algorithm to the form

$$w_{k+1} = w_k - \alpha_k M_k g_k \qquad (2.33)$$

The choice of step size α_k to use with a search direction $d_k = M_k g_k$ is determined by an approximate line search, and use of line search is essential to the success of this method. The quazi-Newton method iteratively tracks the inverse of the Hessian without ever computing it directly. Let $q_k = g_{k+1} - g_k$, and consider the expansion for the gradient (quadratic case)

$$q_k = H_k(w_{k+1} - w_k) = H_k p_k \tag{2.34}$$

If we can evaluate the difference of gradients for p linearly independent increments $p_0, \ldots p_{p-1}$ in the weight vectors, then we can solve for the Hessian (assumed constant). To do so, form the matrices P with ith column the vector p_{i-1} and Q with ith column the vector q_{i-1}. Then we have the matrix equation

$$Q = H P \tag{2.35}$$

which can be solved for the Hessian, when the columns of P are linearly independent, through

$$H = Q P^{-1} \tag{2.36}$$

Thus from the increments in the gradient induced by the increments in the weight vectors as training proceeds, we have some hope of being able to track the Hessian. An approximation to the inverse M of the Hessian is achieved by interchanging q_k and p_k in an approximation to the Hessian itself

$$M = P Q^{-1} \tag{2.37}$$

Hence the information is available in the sequence of gradients that determine the q_k, and the sequence of search directions and learning rates that determine the p_k, to infer to the inverse of the Hessian, particularly if it is only slowly varying.

The Broyden-Fletcher-Goldfarb-Shanno (BFGS) quasi – Newton algorithm [68] implements the update for the approximate inverse M of the Hessian by

$$M_{k+1} = M_k + \left(1 + \frac{q_k^T M_k q_k}{q_k^T p_k}\right) \frac{p_k p_k^T}{p_k p_k^T} - \frac{p_k q_k^T M_k + M_k q_k p_k^T}{q_k^T p_k} \tag{2.38}$$

This recursion is initialized by starting with a positive definite matrix such as the identity, $M_0 = I$. The Determination of the learning rates is critical, as was the case for the method of conjugate directions. Quasi-Newton methods enjoy asymptotically more rapid convergence than that of steepest descent or conjugate gradient methods.

2.1.4 Levenberg-Marquardt algorithm

The Levenberg-Marquardt (LM) algorithm [68] exploits the fact that the error function is a sum of squares as given in (2.1). Introduce the following notation for the error vector and its Jacobian with respect to the network parameters w

$$J = J_{ij} = \frac{\partial e_j}{\partial w_i}, i = 1 : p, j = 1 : n \tag{2.39}$$

The Jacobian matrix is a large $p \times n$ matrix, all [illegible] whose elements are calculated directly by backpropagation technique as presented in Section 2.1.1. 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

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

$$= \sum_{k=1}^{n} \left(e_k \frac{\partial^2 e_k}{\partial w_i \partial w_j} + J_{ik} J_{jk} \right) \tag{2.40}$$

Hence defining $D = \sum_{i=1}^{n} e_i \nabla^2 e_i$ yields the expression

$$H(w) = JJ^T + D \tag{2.41}$$

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

$$M_k = \left[JJ^T + \in I \right]^{-1}, w_{k+1} = w_k - \alpha_k M_k g(w_k) \tag{2.42}$$

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. Hence 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 [43].

2.2. Designing Artificial Neural Networks

The error surface of very small networks has been characterized previously. However, practical networks often contain hundreds of weights and in general, theoretical and empirical results on small networks do not scale up to large networks. To investigate the empirical performance with the different learning algorithms on different architectures and node transfer functions, we have choosen 3 famous chaotic time series benchmarks so that **a)** we know the best solution, **b)** can carefully control various parameters and **c)** know the effect of the different learning algorithms namely backpropagation (BP), scaled conjugate gradient (SCG), quasi-Newton algorithm (QNA) and Levenberg Marquardt algorithm (LM).

We also report some experimentation results related to convergence speed and generalization performance of the four different neural network-learning algorithms discussed in Section 2.1. Performances of the different learning algorithms were evaluated when the activation functions and architectures were changed.

We used a feedforward neural network with 1 hidden layer and the numbers of hidden neurons were varied (14,16,18,20,24) and the speed of convergence and generalization error for each of the four learning algorithms was observed. The effect of node activation functions, log-sigmoidal activation function (LSAF) and tanh-sigmoidal activation function (TSAF), keeping 24 hidden neurons for the four learning algorithms was also studied. Computational complexities of the different learning algorithms were also noted during each event. The experiments were replicated 3 times each with a different starting condition (random weights) and the worst 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 (2500 epochs).We used the following three chaotic time series:

a) Waste Water Flow Prediction

The problem is to predict the wastewater flow into a sewage plant [138]. The water flow was measured every hour. It is important to be able to predict the volume of flow *f(t+1)* as the collecting tank has a limited capacity and a sudden increase in flow will cause to overflow excess water. The water flow prediction is to assist an adaptive online controller. The data set is represented as *[f(t), f(t-1), a(t), b(t), f(t+1)*] where *f(t), f(t-1)* and *f(t+1)* are the water flows at time *t,t-1, and t+1* (hours) respectively. *a(t)* and *b(t)* are the moving averages for 12 hours and 24 hours. The time series consists of 475 data points. The first 240 data sets were used for training and remaining data for testing.

b) Mackey-Glass Chaotic Time Series

The Mackey-Glass differential equation [167] is a chaotic time series for some values of the parameters $x(0)$ and τ.

$$\frac{dx(t)}{dt} = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1\,x(t). \qquad (2.43)$$

We used the value *x(t-18), x(t-12), x(t-6), x(t)* to predict *x(t+6)*. Fourth order Runge-Kutta method was used to generate 1000 data series. The time step used in the method is 0.1 and initial condition were $x(0)=1.2$, $\tau=17$, $x(t)=0$ for $t<0$. First 500 data sets were used for training and remaining data for testing.

c) **Gas Furnace Time Series Data**

This time series was used to predict the CO_2 (carbon dioxide) concentration $y\ (t+1)$ [51]. In a gas furnace system, air and methane are combined to form a mixture of gases containing CO_2. Air fed into the gas furnace is kept constant, while the methane feed rate $u(t)$ can be varied in any desired manner. After that, the resulting CO_2 concentration y(t) is measured in the exhaust gases at the outlet of the furnace. Data is represented as $[u(t), y(t), y(t+1)]$. The time series consists of 292 pairs of observation and 50% of data was used for training and remaining for testing.

2.2.1 Simulation Results Using ANNs

Results for four different learning algorithms for different architectures, node transfer functions for the three different time series are presented in the following sections.

2.2.1.1 Network Architecture

This section investigates the training and generalization behavior of the networks when the architecture of the neural network was changed. The same architecture was used for the three different time series for the four learning algorithms using same node transfer function (tan sigmoidal). Tables 2.1 – 2.3 summarizes the empirical results of training and generalization. Figures 2.1 – 2.6 graphically depict the training and generalization performance for the different learning methods.

Table 2.1. Training and test performance for Mackey Glass Series for different architectures

Mackey Glass Time Series			
Learning algorithm	**Hidden Neurons**	**Root Mean Squared Error**	
		Training data	**Test data**
BP	14	0.0890	0.0880
	16	0.0824	0.0860
	18	0.0764	0.0750
	20	0.0452	0.0442
	24	0.0439	0.0437
SCG	14	0.0040	0.0051
	16	0.0053	0.0052
	18	0.0066	0.0067
	20	0.0058	0.0058
	24	0.0045	0.0045
QNA	14	0.0041	0.0040
	16	0.0031	0.0030
	18	0.0035	0.0036
	20	0.0038	0.0038
	24	0.0034	0.0036
LM	14	0.0016	0.0016
	16	0.0015	0.0015
	18	0.0015	0.0015
	20	0.0010	0.0011
	24	0.0009	0.0009

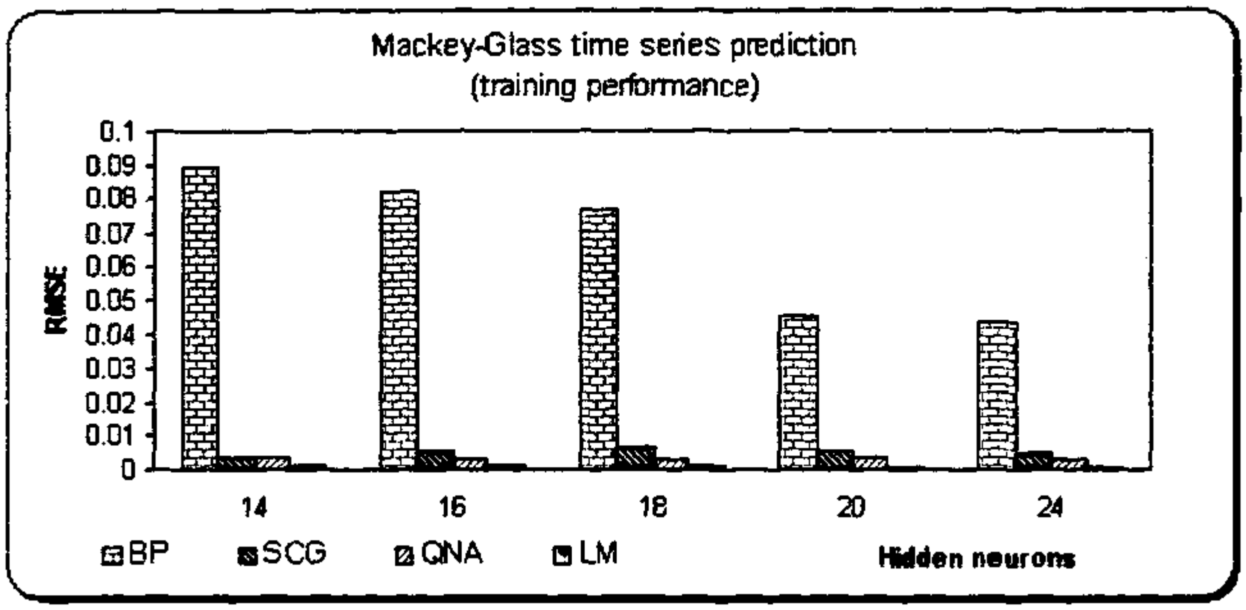


Figure 2.1. Architecture variation: Mackey-Glass time series training performance for different learning algorithms

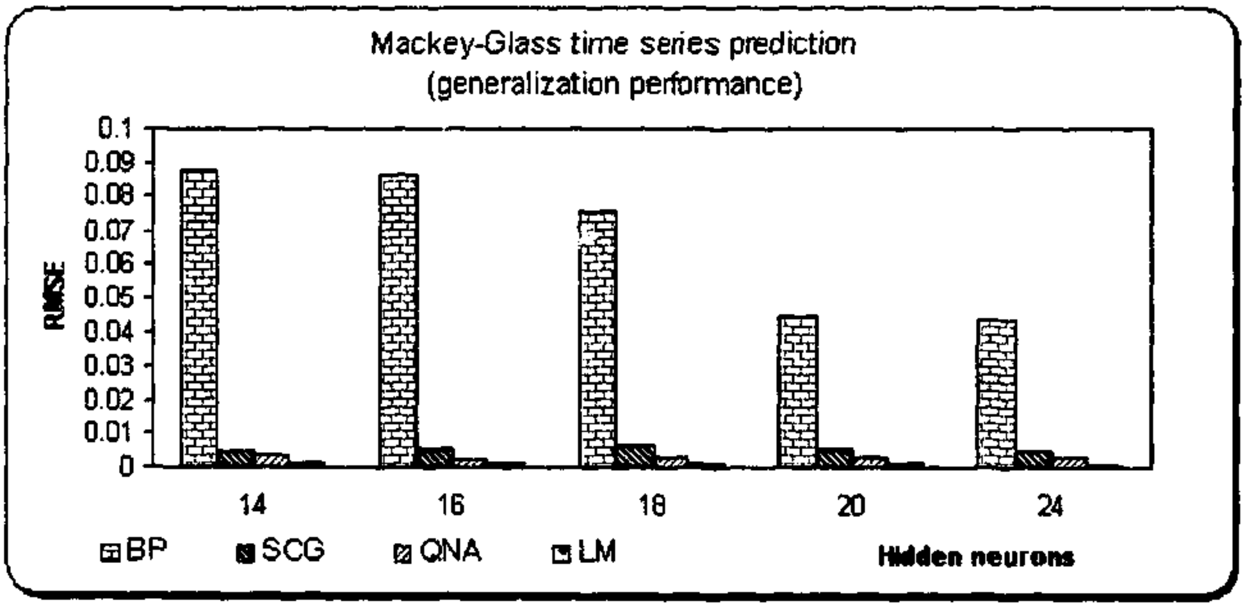


Figure 2.2. Architecture variation: Mackey-Glass time series generalization performance for different learning algorithms

Table 2.2. Training and test performance for gas furnace time series for different architectures

Gas Furnace Time Series			
Learning algorithm	**Hidden Neurons**	**Root Mean Squared Error**	
		Training data	**Test data**
BP	14	0.0760	0.1291
	16	0.0835	0.1056
	18	0.0716	0.0766
	20	0.0800	0.0950
	24	0.0663	0.0970
SCG	14	0.0160	0.0331
	16	0.0157	0.0330
	18	0.0165	0.0330
	20	0.0158	0.0361
	24	0.0153	0.0367
QNA	14	0.0137	0.0529
	16	0.0133	0.0465
	18	0.0133	0.0376
	20	0.0136	0.0410
	24	0.0128	0.0516
LM	14	0.0118	0.0450
	16	0.0140	0.0971
	18	0.0116	0.1080
	20	0.0100	0.1880
	24	0.0100	0.1856

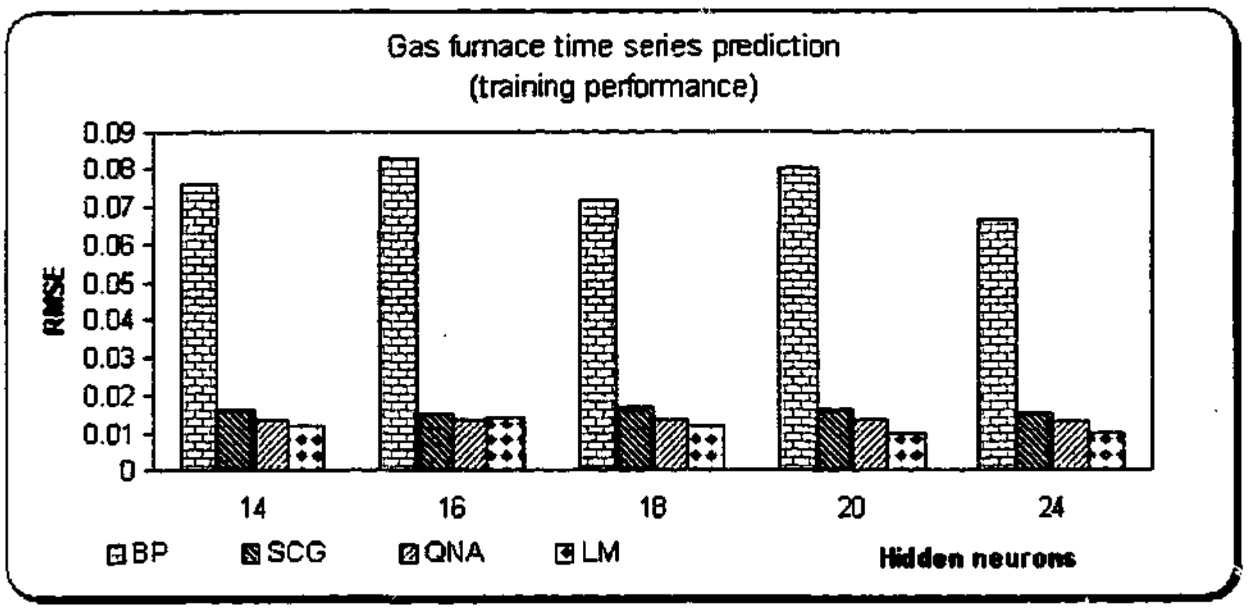


Figure 2.3. Architecture variation: Gas furnace time series training performance for different learning algorithms

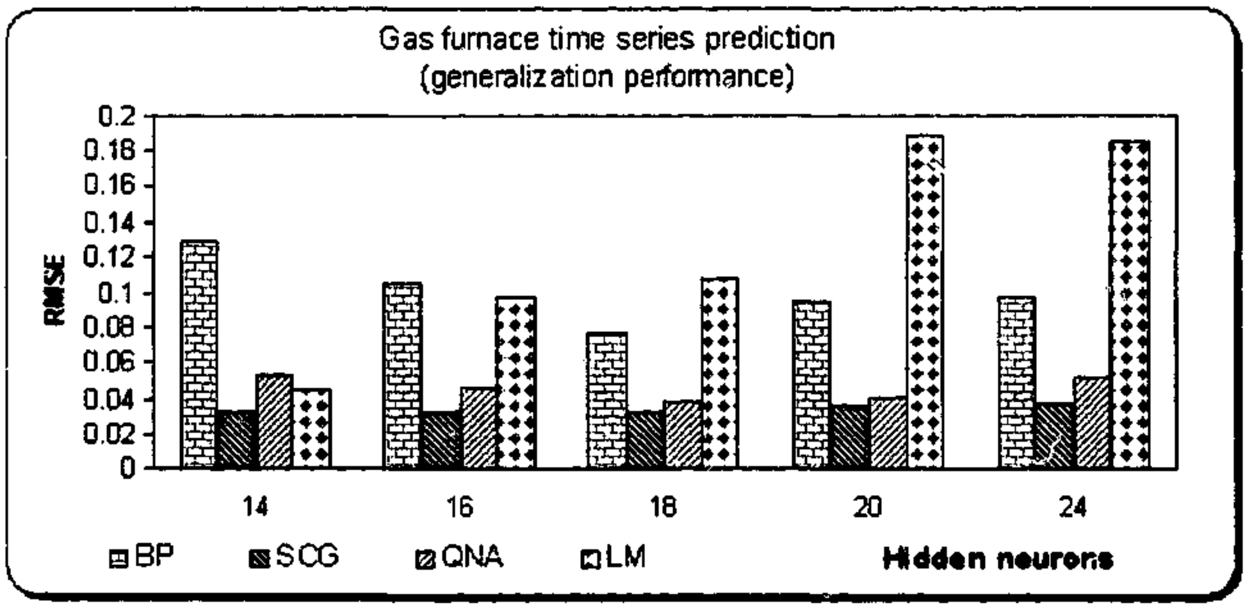


Figure 2.4. Architecture variation: Gas furnace time series generalization performance for different learning algorithms

Table 2.3. Training and test performance for wastewater flow series for different architectures

Wastewater Time Series			
Learning algorithm	**Hidden Neurons**	**Root Mean Squared Error**	
		Training data	**Test data**
BP	14	0.1269	0.1340
	16	0.1184	0.1360
	18	0.1182	0.1350
	20	0.1221	0.1370
	24	0.1169	0.1412
SCG	14	0.0459	0.0900
	16	0.0428	0.1130
	18	0.0425	0.1130
	20	0.0423	0.1626
	24	0.0400	0.0920
QNA	14	0.0423	0.1271
	16	0.0367	0.1369
	18	0.0363	0.1360
	20	0.0339	0.1450
	24	0.0316	0.2620
LM	14	0.0364	0.0950
	16	0.0303	0.1631
	18	0.0314	0.1800
	20	0.0259	0.1314
	24	0.0244	0.1560

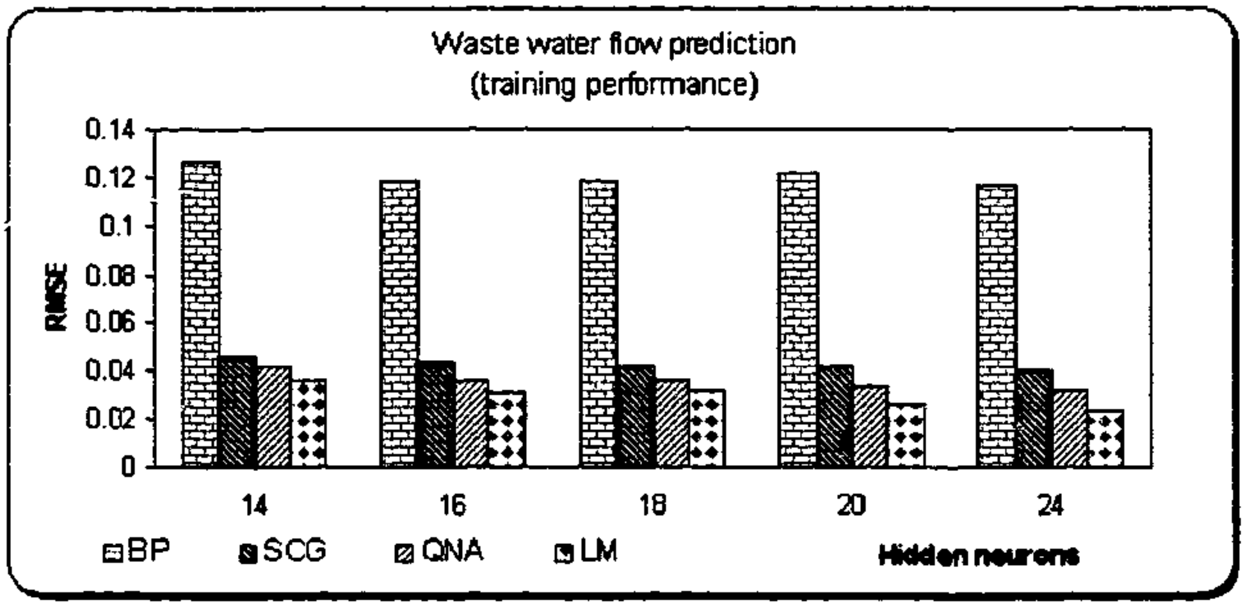


Figure 2.5. Architecture variation: Waste water time series training performance for different training algorithms

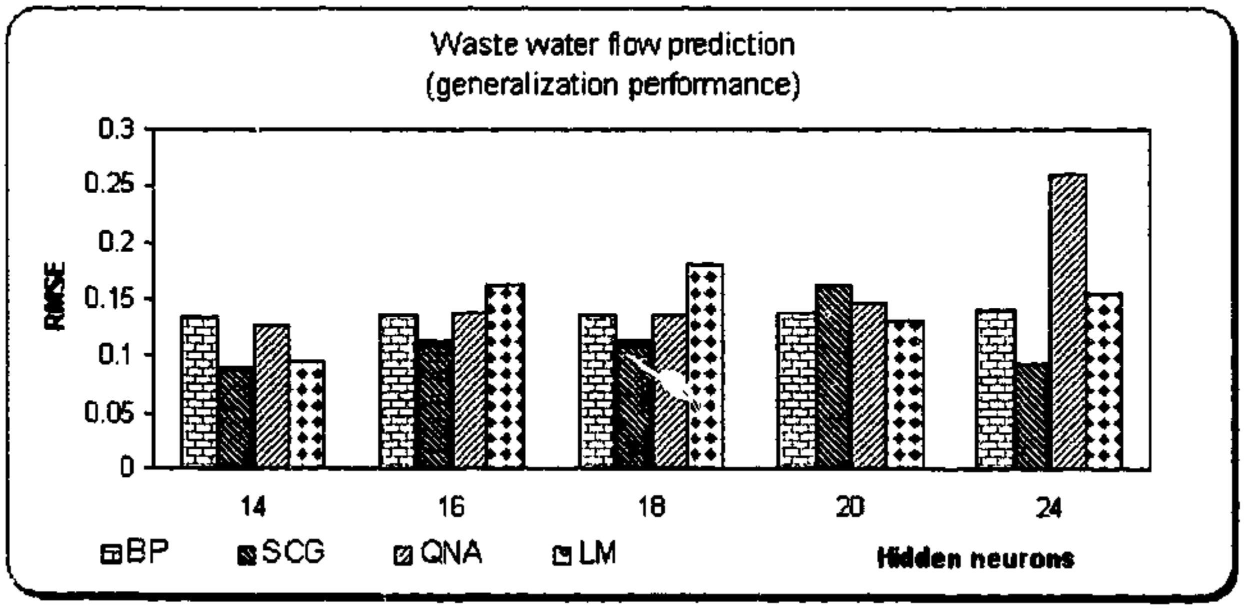


Figure 2.6. Architecture variation: Waste water time series generalization performance for different learning algorithms

2.2.1.2. Node transfer functions

This section investigates the effect of different node transfer functions on training and generalization performance for the four learning algorithms. To compare empirically we maintained the same architecture and only changing the node transfer functions and learning algorithms. All the networks were randomly initialized and trained for 2500 epochs. Tables 2.4 – 2.6 summarizes the empirical results of training and generalization for the two node transfer functions, tanh-sigmoidal activation function (TSAF) and log-sigmoidal activation function (LSAF), when the architecture was fixed with 24 hidden neurons. Figures 2.7 - 2.12 graphically depict the convergence characteristics of the four training algorithms for different node transfer functions during 2500 epochs training.

Table 2.4. Mackey Glass time series: Training and generalization performance for different activation functions

Time series	Learning algorithm	Activation function	Root Mean Squared Error	
			Training	**Test**
Mackey Glass	**BP**	TSAF	0.0439	0.0437
		LSAF	0.0970	0.0950
	SCG	TSAF	0.0045	0.0045
		LSAF	0.0076	0.0074
	QNA	TSAF	0.0033	0.0034
		LSAF	0.0029	0.0029
	LM	TSAF	0.0009	0.0009
		LSAF	0.0009	0.0010

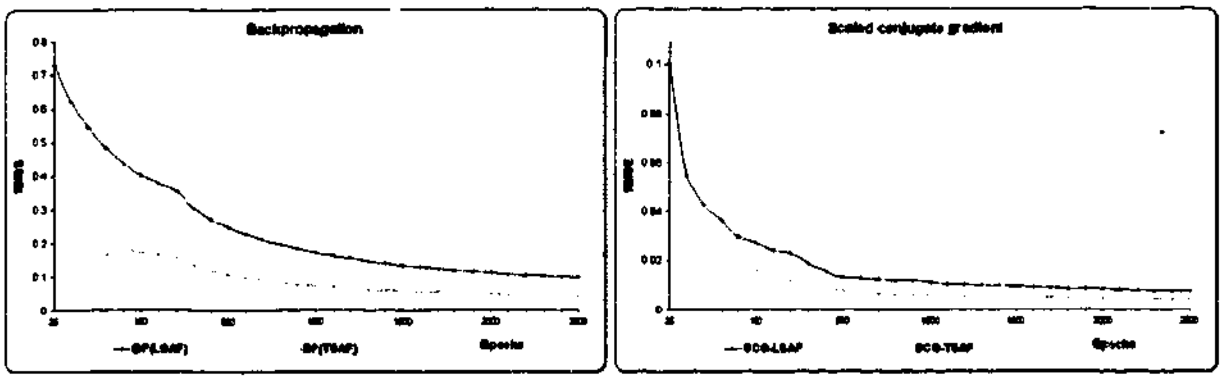


Figure 2.7. Mackey glass time series: Convergence of training when node transfer function is changed (a) backpropagation training (b) scaled conjugate gradient algorithm

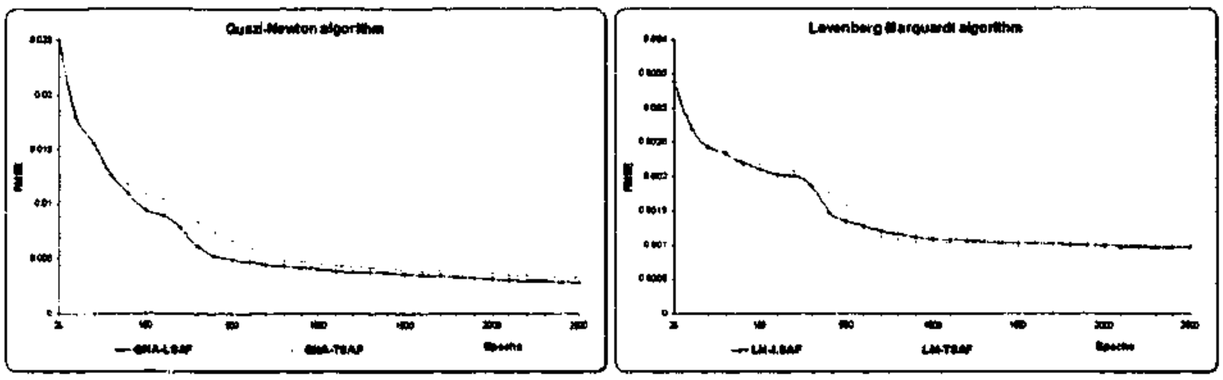


Figure 2.8. Mackey glass time series: Convergence of training when node transfer function is changed (a) Quasi-Newton algorithm (b) Levenberg Marquardt algorithm

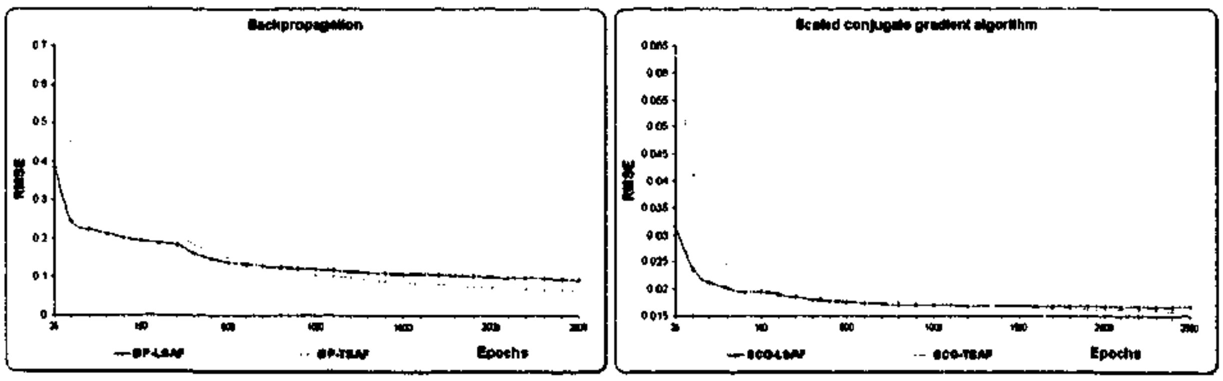


Figure 2.9. Gas Furnace time series: Convergence of training when node transfer function is changed (a) backpropagation training (b) scaled conjugate gradient algorithm

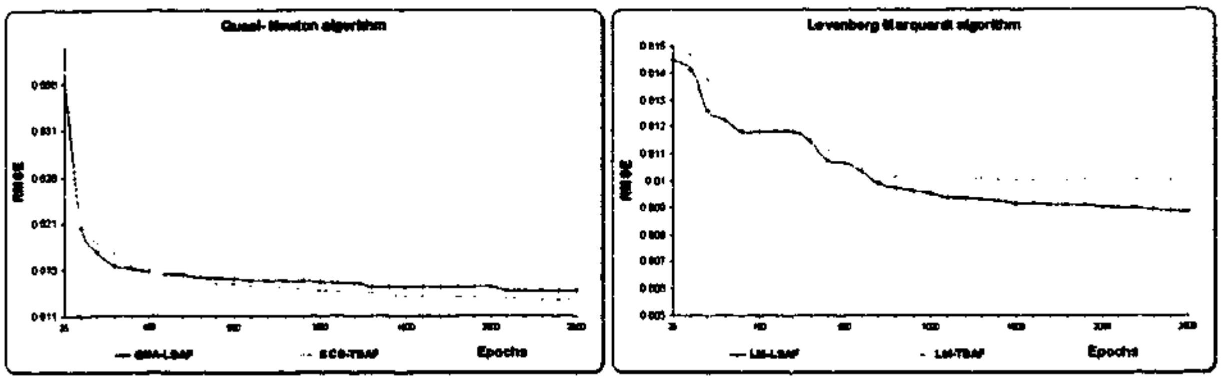


Figure 2.10. Gas furnace series: Convergence of training when node transfer function is changed (a) Quasi-Newton algorithm (b) Levenberg Marquardt algorithm

Table 2.5. Gas furnace series: Training and generalization performance for different activation functions

Time series	Learning algorithm	Activation function	Root Mean Squared Error	
			Training	Test
Gas furnace	BP	TSAF	0.0663	0.0970
		LSAF	0.0940	0.1025
	SCG	TSAF	0.0153	0.0367
		LSAF	0.0162	0.0367
	QNA	TSAF	0.0128	0.0516
		LSAF	0.0137	0.0420
	LM	TSAF	0.0100	0.1856
		LSAF	0.0089	0.1009

Table 2.6. Waste water time series: Training and generalization performance for different activation functions

Time series	**Learning algorithm**	**Activation function**	**Root Mean Squared Error**	
			Training	**Test**
Wastewater	**BP**	TSAF	0.1169	0.1412
		LSAF	0.0156	0.1600
	SCG	TSAF	0.0400	0.0920
		LSAF	0.0420	0.0820
	QNA	TSAF	0.0316	0.4600
		LSAF	0.0256	0.2110
	LM	TSAF	0.0244	0.1560
		LSAF	0.2160	0.1770

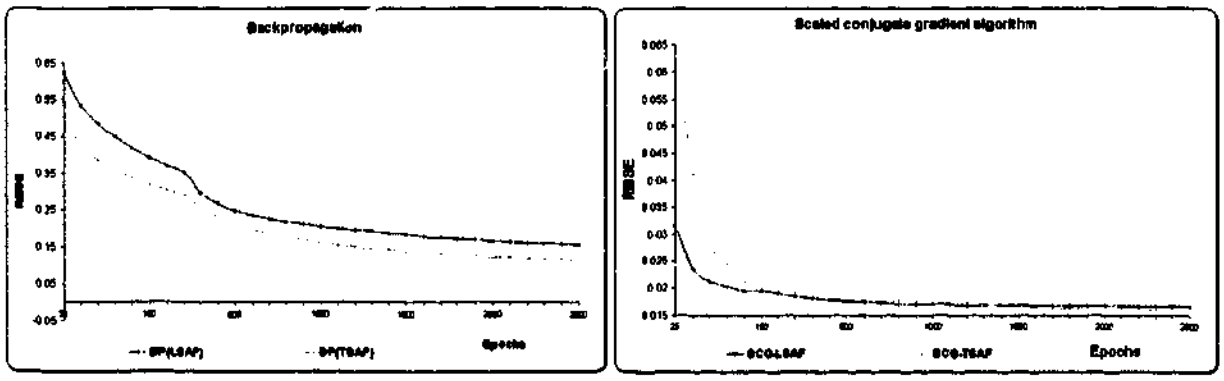


Figure 2.11. Wastewater time series: Convergence of training when node transfer function is changed (a) backpropagation training (b) scaled conjugate gradient algorithm

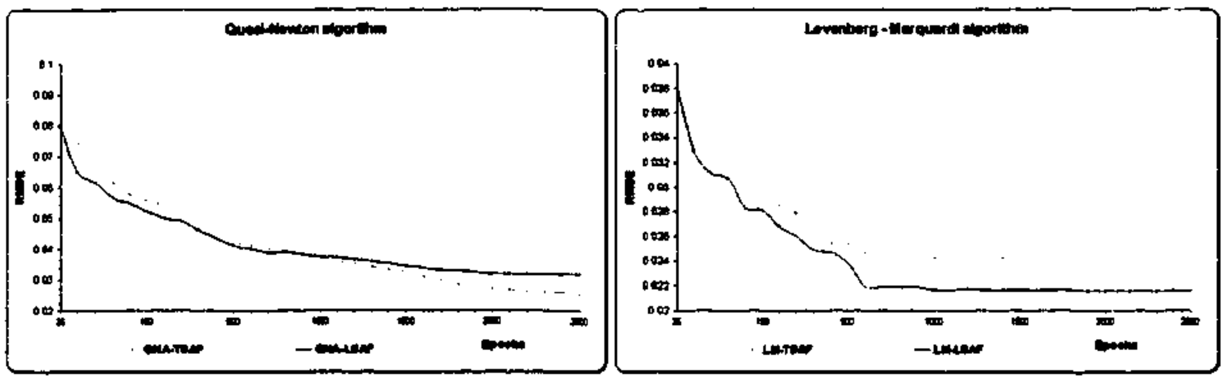


Figure 2.12. Wastewater time series: Convergence of training when node transfer function is changed (a) Quasi-Newton algorithm (b) Levenberg Marquardt algorithm

2.2.1.3 Computational Complexity of Learning algorithms

This section investigates the computational complexity of the different learning algorithms when the architecture of the hidden layer is varied using tanh-sigmoidal activation function. The networks were randomly initialized and trained for 2500 epochs using the different learning algorithms. Table 2.7 summarizes the empirical values of the computational load for the different learning methods for the three different time series.

2.3.Discussion of Results Obtained and Further Work

In this Section we would like to evaluate and summarize the results of the various experimentations mentioned in Section 2.2.1.

For Mackey Glass series (Table 2.1) all the 4 learning algorithms tend to generalize well as the hidden neurons were increased. However, the generalization was better when the hidden neurons were using TSAF. LM showed the fastest convergence regardless of architecture and node activation function. However, the computational complexity of LM algorithm is very amazing as depicted in Table 2.7. For Mackey glass series (with 14 hidden neurons), when BP was using 0.625 billion flops, LM technique required 29.4 billion flops. When the hidden neurons were increased to 24, BP used 1.064 billion flops and LM's share jumped to 203.10 billion flops. LM gave the lowest generalization RMSE of 0.0009 with 24 hidden neurons.

Table 2.7. Approximate computational load for the different time series using the different training algorithms

Learning algorithm	Hidden Neurons	Computational Load (billion flops)		
		Mackey Glass	Gas Furnace	Waste water
BP	14	0.625	0.142	0.301
	16	0.713	0.305	0.645
	18	0.800	0.488	0.880
	20	0.888	0.690	1.460
	24	1.064	0.932	1.970
SCG	14	1.256	0.286	0.604
	16	1.429	0.326	0.689
	18	1.605	0.366	0.774
	20	1.781	0.406	0.859
	24	2.133	0.486	1.029
QNA	14	2.570	0.679	1.910
	16	3.319	0.8899	2.582
	18	4.221	0.9000	3.388
	20	5.313	1.131	4.384
	24	7.989	2.193	6.925
LM	14	29.40	3.930	12.46
	16	57.51	8.355	27.72
	18	93.29	14.03	93.79
	20	137.83	21.10	118.53
	24	203.10	31.83	175.22

As shown in Table 2.2, for gas furnace series the generalization performance were entirely different for the different learning algorithms. BP gave the best generalization RMSE of 0.0766 with 18 hidden neurons. RMSE for SCG, QNA and LM were 0.0330 (16 neurons), 0.0376 (18 neurons) and 0.045 (14 neurons) respectively. As depicted in Figures 2.9 and 2.10 the node transfer function also has an effect on the training speed and generalization performance. LM algorithm converged much faster and gave a better generalization performance when the node transfer function was changed to LSAF (Refer to Figure 2.10(b)).

Waste water prediction series also showed a different generalization performance when the architecture was changed for the different learning algorithms (Refer to Table 2.3). BP's best generalization RMSE was 0.135 with 18 hidden neurons using TSAF and that of SCG, QNA and LM were 0.0900, 0.1271 and 0.095 with 14 neurons each respectively. LM algorithm converged much faster and gave a better generalization performance when the node transfer function was changed to LSAF (Refer to Figure 2.12(b)).

In spite of computational complexity, LM performed well for Mackey Glass series. For gas furnace and waste water prediction SCG algorithm performed better. However, the speed of convergence of LM in all the three cases is worth noting. This leads us to the following questions:

- What is the optimal architecture (number of neurons and hidden layers) for a given problem?
- What node transfer functions should one choose?
- What is the optimal learning algorithm and its parameters?

From the above discussion it is clear that the selection of the topology of a network and the best learning algorithm and its parameters is a tedious task for designing an optimal artificial neural network, which is smaller, faster and with a better generalization performance. Evolutionary algorithm is an adaptive search technique based on the principles and mechanisms of natural selection and survival of the fittest from natural evolution [85]. The interest in evolutionary search procedures for designing neural network topology has been growing in recent years as they can evolve towards the optimal architecture without outside interference, thus eliminating the tedious trial and error work of manually finding an optimal network [15] [250]. In Chapter 3, we will introduce the evolutionary design of neural networks and the concept of meta-learning in evolutionary artificial neural networks [15].

Chapter 3: Meta-Learning in Evolutionary Artificial Neural Networks

3.0 Introduction

At present, neural network design relies heavily on human experts who have sufficient knowledge about the different aspects of the network and the problem domain. As the complexity of the problem domain increases, manual design becomes more difficult and unmanageable. Evolutionary artificial neural networks (EANNs) refer to a special class of artificial neural networks (ANNs) in which evolution is another fundamental form of adaptation in addition to learning [5] [15]. Evolutionary algorithms (EA) are used to adapt the connection weights, network architecture and learning rules according to the problem environment. A distinct feature of EANNs is their adaptability to a dynamic environment. In other words EANNs can adapt to an environment as well as changes in the environment. The two forms of adaptation: evolution and learning in EANNs make their adaptation to a dynamic environment much more effective and efficient. In Section 3.1, we present the fundamental concepts of EA's followed by state of the art design of EANNs in Section 3.2. In Section 3.3, we then present our work on evolutionary neural networks based on meta-learning (MLEANN) followed by experimentation results and discussions.

3.1 Evolutionary Algorithms

EAs are population based adaptive methods, which may be used to solve optimization problems, based on the genetic processes of biological organisms [85] [86]. Over many generations, natural populations evolve according to the principles of natural selection and "Survival of the Fittest", first clearly stated by Charles Darwin in "*On the Origin of Species*". By mimicking this process, EAs are able to "evolve" solutions to real world problems, if they have been suitably encoded. The procedure may be written as the difference equation:

$$x[t+1] = s(v(x[t])) \qquad (3.1)$$

```
1. Generate the initial population $P(0)$ at random and set $i=0$;
2. Repeat until the number of iterations or time has reached the
   preset limit or the population has converged.
     • Evaluate the fitness of each individual in $P(i)$
     • Select parents from $P(i)$ based on their fitness in $P(i)$
     • Apply reproduction operators to the parents and produce
       offspring, the next generation, $P(i+1)$ is obtained from the
       offspring and possibly parents.
```

Figure 3.1. Pseudo code of an evolutionary algorithm

3.2 Evolutionary Artificial Neural Networks

Many of the conventional ANNs now being designed are statistically quite accurate but they still leave a bad taste with users who expect computers to solve their problems accurately. The important drawback is that the designer has to specify the number of neurons, their distribution over several layers and interconnection between them. Several methods have been proposed to automatically construct ANNs for reduction in network complexity that is to determine the appropriate number of hidden units, layers, etc. Topological optimization algorithms such as Extentron [31], Upstart [90], Pruning [199] [223] and Cascade Correlation [82] etc. got its own limitations.

The interest in evolutionary search procedures for designing ANN architecture has been growing in recent years as they can evolve towards the optimal architecture without outside interference, thus eliminating the tedious trial and error work of manually finding an optimal network [5] [15] [28] [52] [53] [54] [55] [87] [96] [178] [234] [247] [248] [249] [251]. The advantage of the automatic design over the manual design becomes clearer as the complexity of ANN increases. EANNs provide a general framework for investigating various aspects of simulated evolution and learning [32] [45] [46] [152] [165].

3.2.1 General Framework for EANNs

In EANN's evolution can be introduced at various levels. At the lowest level, evolution can be introduced into weight training, where ANN weights are evolved. At the next higher level, evolution can be introduced into neural network architecture adaptation, where the architecture (number of hidden layers, no of hidden neurons and node transfer functions) is evolved. At the highest level, evolution can be introduced into the learning mechanism. A general framework of EANNs which includes the above three levels of evolution is given in Figure 3.2 [5] [15].

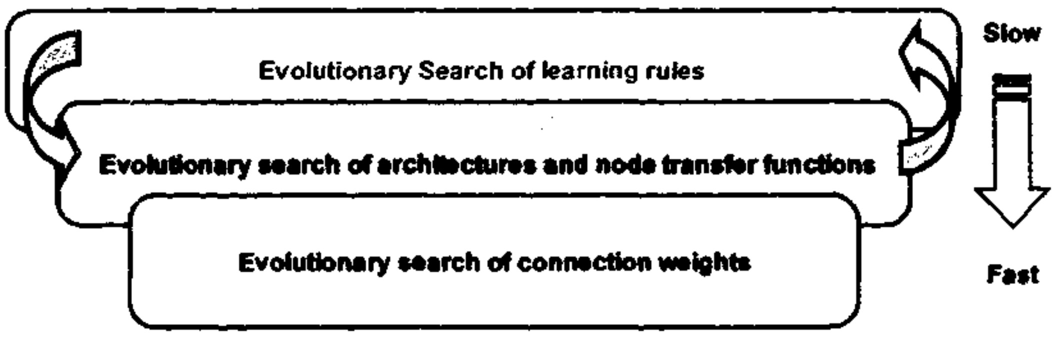

Evolutionary Search of learning rules
Evolutionary search of architectures and node transfer functions
Evolutionary search of connection weights
Slow
Fast

Figure 3.2. A General Framework for EANNs

From the design point of view, the decision on the level of evolution depends on what kind of prior knowledge is available. If there is more prior knowledge about EANN's architectures than that about their learning rules or a particular class of architectures is pursued, it is better to implement the evolution of architectures at the highest level because such knowledge can be used to reduce the search space and the lower level evolution of learning rules can be more biased towards this kind of architectures. On the other hand, the evolution of learning rules should be at the highest level if there is more prior knowledge about them available or there is a special interest in certain type of learning rules.

3.2.1.1 Evolutionary Search of Connection weights

The shortcomings of the BP algorithm mentioned in Section 2.1 could be overcome if the training process is formulated as a global search of connection weights towards an optimal set defined by the evolutionary algorithm. Optimal connection weights can be formulated as a global search problem wherein the architecture of the neural network is pre-defined and fixed during the evolution.

Connection weights may be represented as binary strings represented by a certain length. The whole network is encoded by concatenation of all the connection weights of the network in the chromosome. A heuristic concerning the order of the concatenation is to put connection weights to the same node together. Fig 3.3 illustrates the binary representation of connection weights wherein each weight is represented by 4 bits.

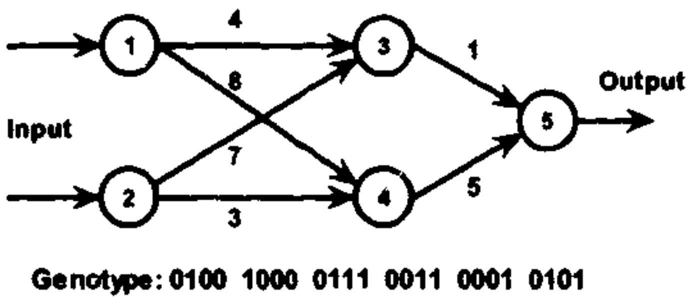


Figure 3.2. Connection weight chromosome encoding using binary representation

Real numbers have been proposed to represent connection weights directly [209]. A representation of the ANN could be (2.0, 6.0, 5.0, 1.0, 4.0, 10.0). However proper genetic operators are to be chosen depending upon the representation used.

Evolutionary Search of connection weights can be formulated as follows:

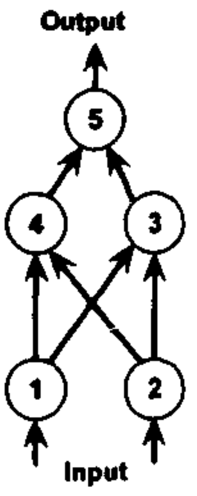
1) *Generate an initial population of N weight chromosomes. Evaluate the fitness of each EANN depending on the problem.*
2) *Depending on the fitness and using suitable selection methods reproduce a number of children for each individual in the current generation.*
3) *Apply genetic operators to each child individual generated above and obtain the next generation.*
4) *Check whether the network has achieved the required error rate or the specified number of generations has been reached. Go to Step 2.*
5) *End*

While gradient based techniques are very much dependant on the initial setting of weights, the proposed algorithm can be considered generally much less sensitive to initial conditions. They always search for a global optimal solution, while any gradient descent or second order optimization technique can only find local optimum in a neighborhood of the initial solution. Performance by using the above approach will directly depend on the problem.

3.2.1.2 Evolutionary Search of Architectures

Evolutionary architecture adaptation can be achieved by constructive and destructive algorithms. Constructive algorithms, which add complexity to the network starting from a very simple architecture until the entire network is able to learn the task [90] [177] [170]. Destructive algorithms start with large architectures and remove nodes and interconnections until the ANN is no longer able to perform its task [199] [223]. Then the last removal is undone. Figure 3.3 demonstrates how typical neural network architecture could be directly encoded and how the genotype is represented. For an optimal network, the required node transfer function (gaussian, sigmoidal, etc.) can be formulated as a global search problem, which is evolved simultaneously with the search for architectures [164].

To minimize the size of the genotype string and improve scalability, when priori knowledge of the architecture is known it will be efficient to use some indirect coding (high level) schemes. For example, if two neighboring layers are fully connected then the architecture can be coded by simply using the number of layers and nodes. The blueprint representation is a popular indirect coding scheme where it assumes architecture consists of various segments or areas. Each segment or area will define a set of neurons, their spatial arrangement and their efferent connectivity. Several high level coding schemes like graph generation system [151], Symbiotic Adaptive Neuro-Evolution (SANE) [208] [187], marker based genetic coding [95], L-systems [44], cellular encoding [102], fractal representation [174] etc are some of the rugged techniques.



From / To	1	2	3	4	5	Bias	Gene
1	0	0	0	0	0	0	000000
2	0	0	0	0	0	0	000000
3	1	1	0	0	0	1	110001
4	1	1	0	0	0	1	110001
5	0	0	1	1	0	1	001101

Genotype: 000000 000000 110001 110001 **001101**

Figure 3.3. Architecture chromosome using binary coding

Global search of transfer function and the connectivity of the ANN using evolutionary algorithms can be formulated as follows

1) *The evolution of architectures has to be implemented such that the evolution of weight chromosomes are evolved at a faster rate i.e. for every architecture chromosome, there will be several weight chromosomes evolving at a faster time scale*
2) *Generate an initial population of N architecture chromosomes. Evaluate the fitness of each EANN depending on the problem.*
3) *Depending on the fitness and using suitable selection methods reproduce a number of children for each individual in the current generation.*
4) *Apply genetic operators to each child individual generated above and obtain the next generation.*
5) *Check whether the network has achieved the required error rate or the specified number of generations has been reached. Go to Step 3.*
6) *End*

3.2.1.3 Evolutionary Search of Learning Rules

For the neural network to be fully optimal the learning rules are to be adapted dynamically according to its architecture and the given problem. Deciding the learning rate and momentum can be considered as the first attempt of learning rules [150]. The basic learning rule can be generalized by the function

$$\Delta w(t) = \sum_{k=1}^{n} \sum_{i_1,i_2,\ldots,i_k=1}^{n} (\theta_{i_1,i_2,\ldots,i_k} \prod_{j=1}^{k} x_{ij}(t-1)) \qquad (3.2)$$

Where t is the time, Δw is the weight change, $x_1, x_2, \ldots\ldots x_n$ are local variables and the *θ's* are the real values coefficients which will be determined by the global search algorithm. In the above equation different values of *θ's* determine different learning rules. The above equation is arrived based on the assumption that the same rule is applicable at every node of the network and the weight updating is only dependent on the input/output activations and the connection weights on a particular node. Genotypes (*θ's*) can be encoded as real-valued coefficients and the global search for learning rules using the hybrid algorithm can be formulated as follows:

1. *The evolution of learning rules has to be implemented such that the evolution of architecture chromosomes are evolved at a faster rate i.e. for every learning rule chromosome, there will be several architecture chromosomes evolving at a faster time scale*
2. *Generate an initial population of N learning rules. Evaluate the fitness of each EANN depending on the problem.*
3. *Depending on the fitness and using suitable selection methods reproduce a number of children for each individual in the current generation.*
4. *Apply genetic operators to each child individual generated above and obtain the next generation.*
5. *Check whether the network has achieved the required error rate or the specified number of generations has been reached. Go to Step 3.*
6. *End*

Several researches have been going on about how to formulate different optimal learning rules [15] [34] [88] [247]. The adaptive adjustment of BP algorithm's parameters, such as the learning rate and momentum, through evolution could be considered as the first attempt of the evolution of learning rules [111]. Chalmers [66] defined the form of learning rules as a linear function of four local variables and their six pair wise products [88] [34].

Global optimization of neural network has been widely addressed using several other techniques [64] [78] [89] [206] [217] [216] [218] [219] [257]. Sexton et al [217] used simulated annealing algorithm for optimization of learning. For optimization of the neural network learning, in many cases, a pre-defined architecture was used and in a few cases architectures were evolved together. No work has been reported to the best of our knowledge, where the network is fully automated (interaction of the different evolutionary search mechanisms) using the generic framework mentioned in Section 3.2. Many a times, the search space is narrowed down by pre-defined architecture, node transfer functions and learning rules.

3.3 Meta Learning Evolutionary Artificial Neural Networks (MLEANN)

One major problem of evolutionary algorithm is their inefficiency in fine tuning local search although they are good at global search. The efficiency of evolutionary training can be improved significantly by incorporating a local search procedure into the evolution. Evolutionary algorithms are used to first locate a good region in the space and then a local search procedure is used to find a near optimal solution in this region. It is interesting to consider finding good initial weights as locating a good region in the space. Defining that the basin of attraction of a local minimum is composed of all the points, sets of weights in this case, which can converge to the local minimum through a local search algorithm, then a global minimum can easily be found by the local search algorithm if the evolutionary algorithm can locate any point, i.e, a set of initial weights, in the basin of attraction of the global minimum. Referring to Figure 3.4, G_1 and G_2 could be considered as the initial weights as located by the evolutionary search and W_A and W_B the corresponding final weights fine-tuned by the meta-learning technique.

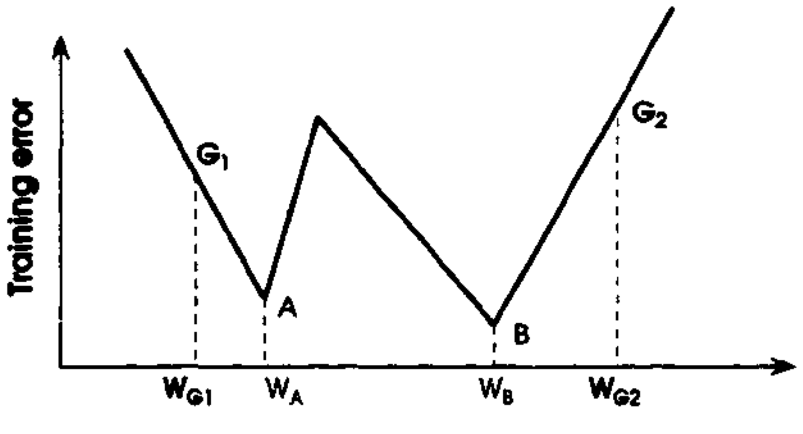


Figure 3.4. Fine tuning of weights using meta-learning

Figure 3.5 illustrates the general interaction mechanism with the learning mechanism of the EANN evolving at the highest level on the slowest time scale. All the randomly generated architecture of the initial population are trained by four different learning algorithms (backpropagation-BP, scaled conjugate gradient-SCG, quasi-Newton algorithm-QNA and Levenberg-Marquardt-LM) and evolved in a parallel environment. Parameters controlling the performance of the learning algorithm will be adapted (example, learning rate and momentum for BP) according to the problem. Figure 3.6 depicts the basic algorithm of proposed meta-learning EANN. Architecture of the chromosome is depicted in Figure 3.7.

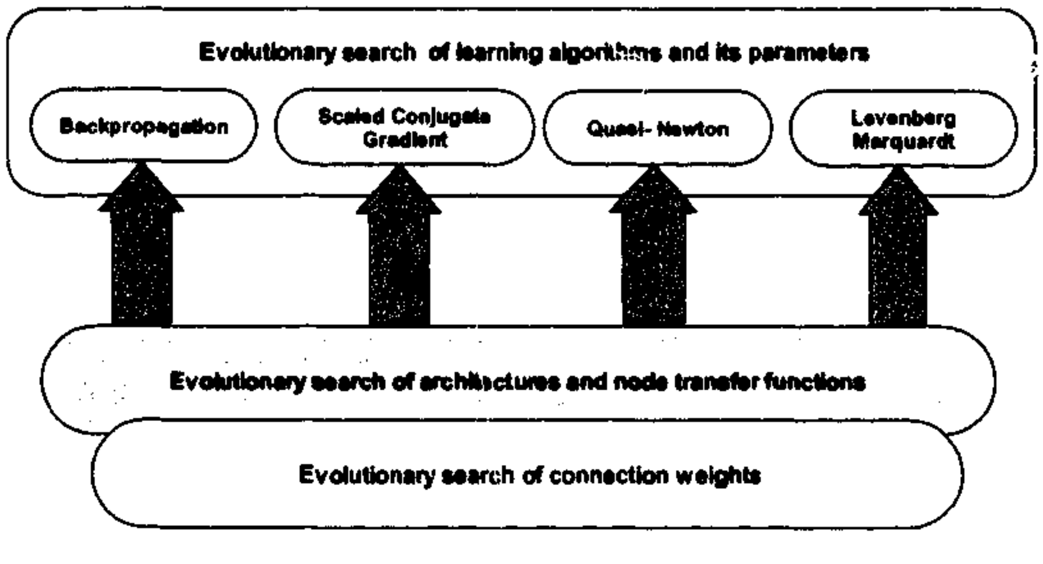
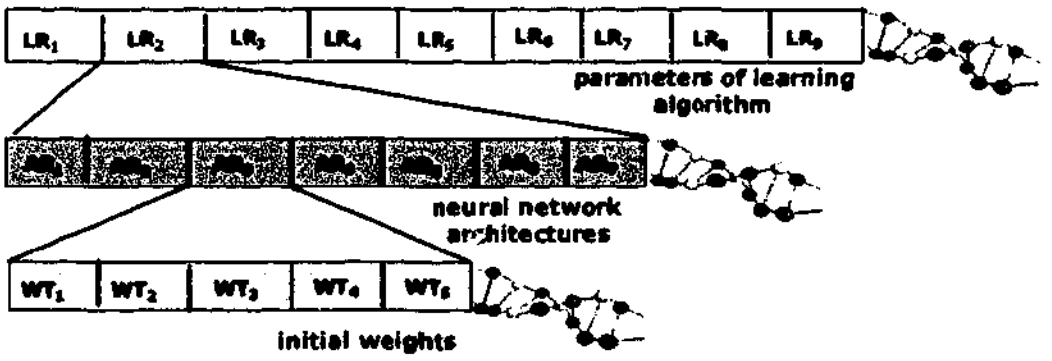

Evolutionary search of learning algorithms and its parameters
Backpropagation
Scaled Conjugate Gradient
Quasi- Newton
Levenberg Marquardt
Evolutionary search of architectures and node transfer functions
Evolutionary search of connection weights

Figure 3.5. Interaction of various evolutionary search mechanisms

1. *Set t=0 and randomly generate an initial population of neural networks with architectures, node transfer functions and connection weights assigned at random.*
2. *In a parallel mode, evaluate fitness of each ANN using BP/SCG/QNA and LM*
3. *Based on fitness value, select parents for reproduction*
4. *Apply mutation to the parents and produce offspring (s) for next generation. Refill the population back to the defined size.*
5. *Repeat step 2*
6. *STOP when the required solution is found or number of iterations has reached the required limit.*

Figure 3.6. Meta-learning algorithm for EANNs



LR1
LR2
LR3
LR4
LR5
LR6
LR7
LR8
LR9
parameters of learning algorithm
neural network architectures
WT1
WT2
WT3
WT4
WT5
initial weights

Figure 3.7. Chromosome representation of the proposed EANN

3.3.1 MLEANN: Experimentation Setup

We have applied the proposed technique to the three-time series prediction problems discussed in Chapter 2. For performance comparison, we used the same set of training and test data that were used for experimentations with neural networks. For performance evaluation, the parameters used in our experiments were set to be the same for all the 3 problems. Fitness value is calculated based on the RMSE achieved on the test set. In this experiment, we have considered the best-evolved neural network as the best individual of the last generation. As the learning process is evolved separately, user has the option to pick the best neural network (e.g. less RMSE or less computational expensive etc.) among the four learning algorithms. All the genotypes were represented using binary coding and the initial populations were randomly generated based on the following parameters shown in Table 3.1. The parameter settings, which were evolved for the different learning algorithms, are illustrated in Table 3.2. We also investigated the performance of the proposed method with a restriction of architecture (no of hidden neurons). We set a maximum number of 4 hidden neurons and evaluated the learning performance. The experiments were repeated three times and the worst RMSE values are reported.

3.3.2 MLEANN: Experimentation Results

Table 3.3 displays empirical values of RMSE on test data for the three time series problems without architecture restriction. For comparison purposes, test set RMSE values using conventional design techniques are also presented in Table 3.3 (adapted from Chapter2). Table 3.4 illustrates the RMSE values on training/test set data using the meta-learning technique when the architecture restriction was imposed. Run times for the two different experimentations are also presented. Figures 3.8, 3.9 and 3.10 illustrates the test results of the three data sets using the meta-learning approach (using BP algorithm). Figures 3.11, 3.12 and 3.13 displays the convergence of the meta-learning algorithm during the 40 generations for the three data sets.

Table 3.1. Parameters used for evolutionary design of artificial neural networks

Population size	40
Maximum no of generations	40
Number of hidden nodes	• Experiment 1: 5-16 hidden nodes • Experiment 2: maximum 4 neurons
Activation functions	tanh *(T)*, logistic *(L)*, sigmoidal *(S)*, tanh-sigmoidal *(T*)*, log-sigmoidal *(L*)*
Output neuron	linear
Training epochs	500
Initialization of weights	+/- 0.3
Ranked based selection	0.50
Elitism	5 %
Mutation rate	0.40

Table 3.2. Parameters settings of the learning algorithms

Learning algorithm	**Parameter**	**Setting**
Backpropagation	Learning rate	0.25-0.05
	Momentum	0.25-0.05
Scaled conjugate gradient algorithm	Change in weight for second derivative approximation	0 - 0.0001
	Regulating the indefiniteness of the Hessian	0 – 1.0 E-06
Quasi-Newton algorithm	Step lengths	1.0E-06 – 100
	Limits on step sizes	0.1 – 0.6
	Scale factor to determine performance	0.001 – 0.003
	Scale factor to determine step size.	0.1 - 0.4
Levenberg Marquardt	Learning rate	0.001 – 0.02

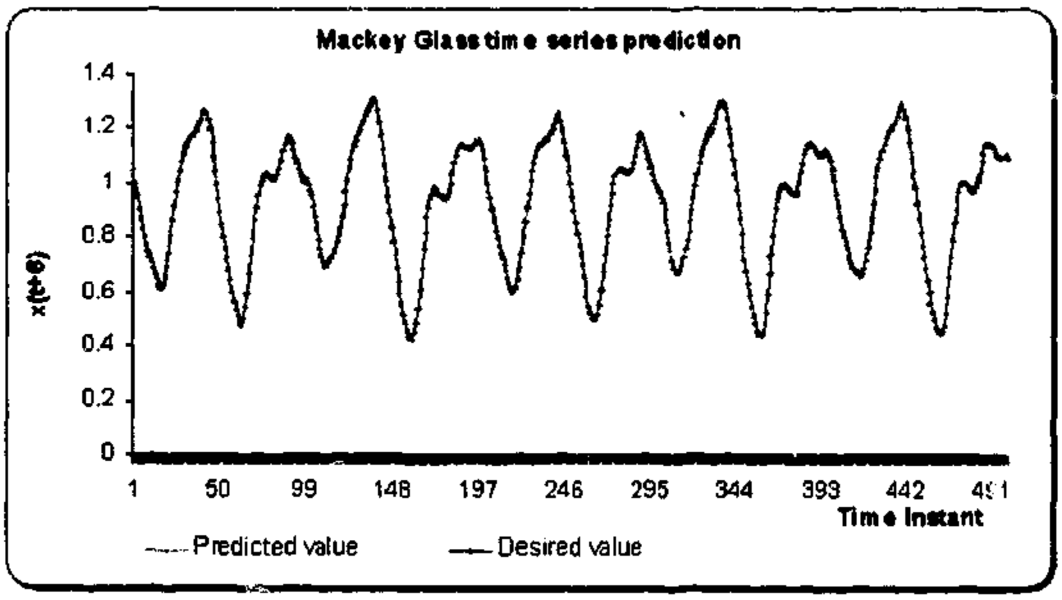


Figure 3. 8. Test results using 500 epochs BP meta-learning for Mackey Glass series

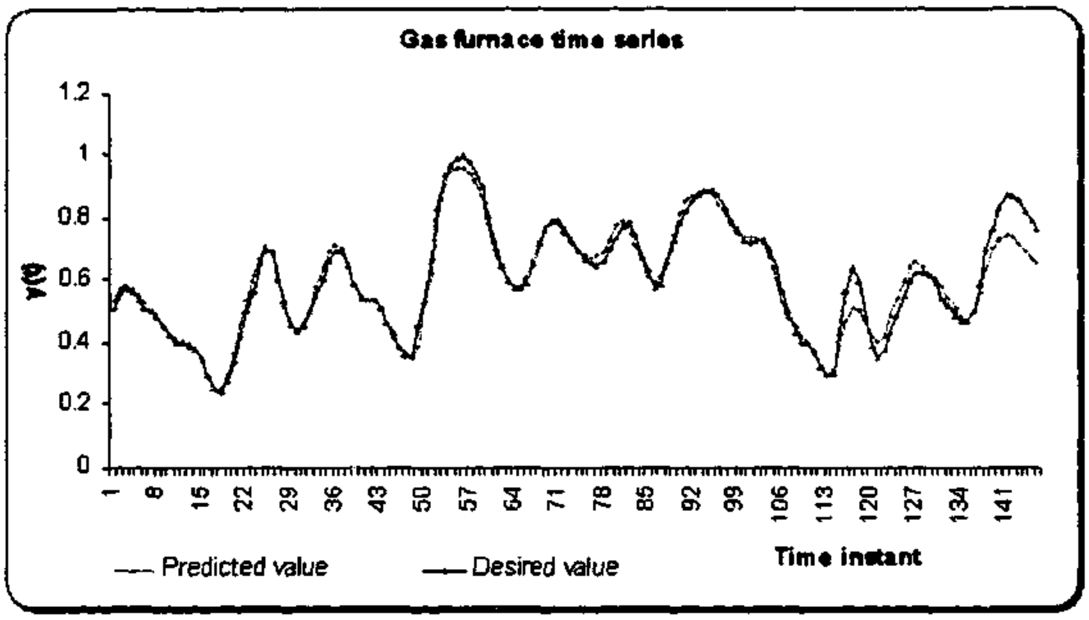


Figure 3.9. Test results using 500 epochs BP meta-learning for Gas furnace series

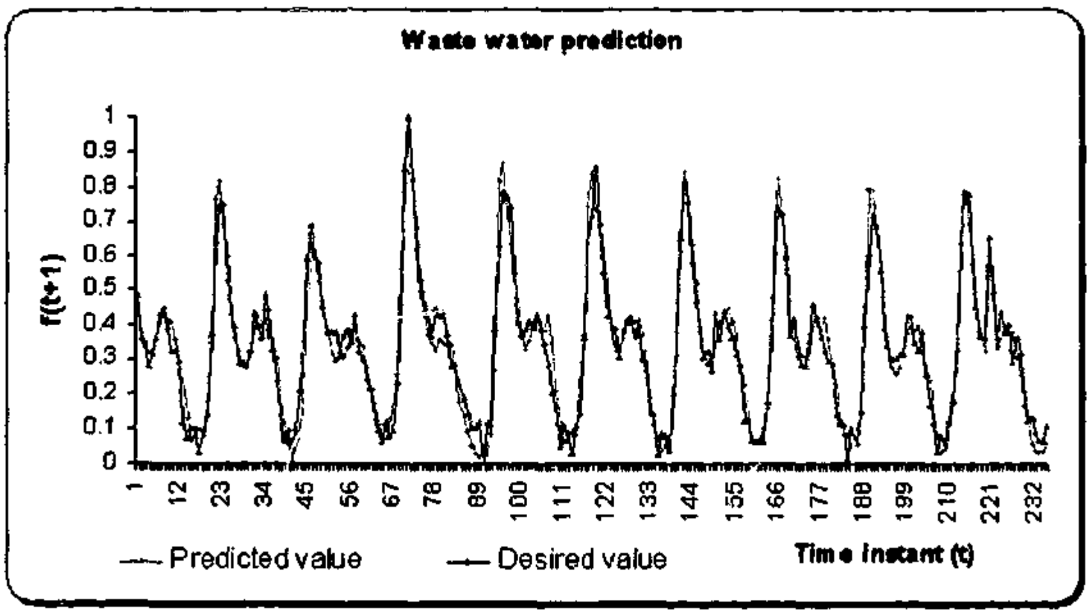


Figure 3.10. Test results using 500 epochs BP meta-learning for wastewater flow series

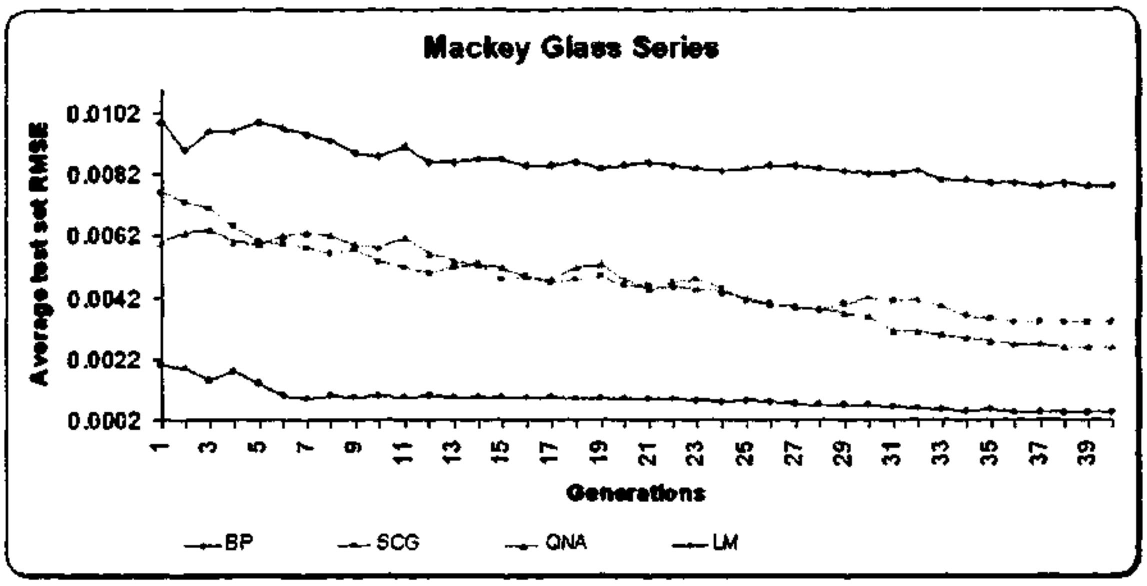


Figure 3.11. Mackey Glass time series: Average test set RSME values during the 40 generations and meta-learning

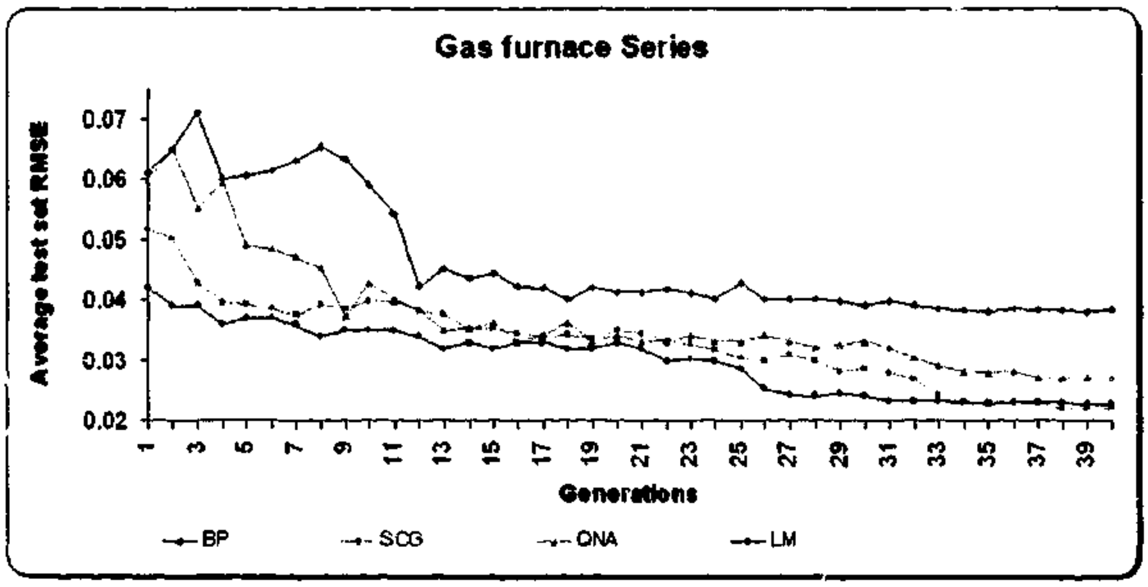


Figure 3.12. Gas furnace time series: Average test set RSME values during the 40 generations and meta-learning

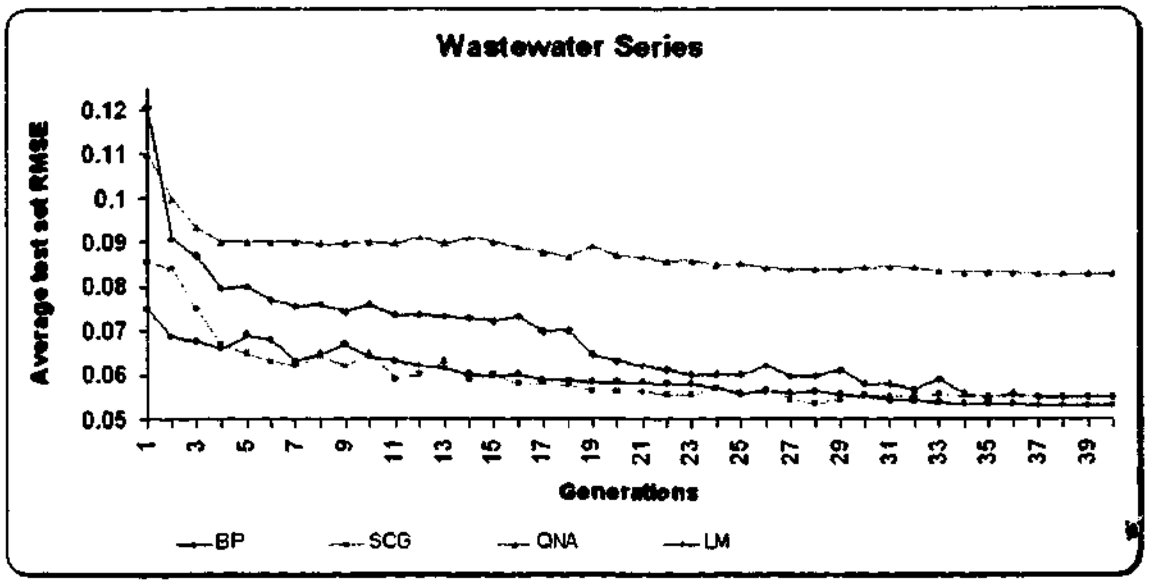


Figure 3.13. Wastewater time series: Average test set RSME values during the 40 generations and meta-learning

Table 3.3. Performance comparison between MLEANN (without architecture restriction) and ANN

Time series	Learn Algo.	MLEANN			ANN	
		RMSE		Architecture	RMSE	Architecture
		Training	Test			
Mackey Glass	BP	0.0072	0.0077	7 T, 3 L	0.0437	24 T*
	SCG	0.0030	0.0031	11 T	0.0045	24 T*
	QNA	0.0024	0.0027	6 T, 4 T*	0.0034	24 T*
	LM	0.0004	†0.0004	8 T, 2 T* 1 L*	0.0009	24 T*
Gas Furnace	BP	0.0159	0.0358	8 T	0.0766	18 T*
	SCG	0.0110	†0.0210	8 T, 2 T*	0.0330	16 T*
	QNA	0.0115	0.0256	7 T, 2 L*	0.0376	18 T*
	LM	0.0120	0.0223	6 T, 1 L, 1 T*	0.0451	14 T*
Waste Water	BP	0.0441	0.0547	6 T, 5 T*,1 L	0.1360	16 T*
	SCG	0.0457	0.0579	6 T, 4 L*	0.0820	14 T*
	QNA	0.0673	0.0823	5 T, 5 TS	0.1276	14 T*
	LM	0.0425	†0.0521	8 T, 1 LS	0.0951	14 T*

Table 3.4. Performance results and run time comparison of MLEANN (architecture restriction- maximum 4 hidden nodes) and ANN.

Time series	**Learn Algo.**	**MLEANN**			***Run time in minutes**	
		RMSE		**Architecture**		
		Training	**Test**		**$^{++}$A**	**$^{+}$B**
Mackey Glass	BP	0.0166	0.0168	4T	1181	288
	SCG	0.0062	0.0067	3 T, 1 T*	2066	504
	QNA	0.0059	0.0058	3 T*, 1 L	2169	528
	LM	0.0056	†0.0061	2 L*, 2 T*	2463	602
Gas Furnace	BP	0.0189	0.0371	3 L	305	62
	SCG	0.0179	0.0295	1 T*, 2 L	629	121
	QNA	0.0156	0.0295	2 T*, 1 L*, 1 L	661	128
	LM	0.0181	†0.0290	1 T, 1 L, 1 T*	696	132
Waste Water	BP	0.0647	0.0639	2T, 2T*	702	146
	SCG	0.0580	0.0600	2 T*, 1 T, 1 L	1254	267
	QNA	0.0590	0.0596	3 T*, 1L*	1291	279
	LM	0.0567	†0.0591	2 L, 1 T, 1 T*	1176	294

$^{++}$ without architecture restriction, $^{+}$ with architecture restriction

† Lowest RMSE error, * On a P II, 450 MHz, 256 MB RAM machine

3.3.3 Comparison with Neuro-Fuzzy Systems and Other Intelligent Techniques

In this Section, we compare the performance of MLEANN (RMSE values on training and test sets) with two popular neuro-fuzzy models, global optimization technique using cutting angle method [35] and multivariate adaptive regression splines (MARS) [17]. The neuro-fuzzy models considered were Dynamic Evolving Fuzzy Neural Networks (dmEFuNN) [139] implementing a Mamdani fuzzy inference system [169] and an Adaptive Neuro-Fuzzy

Inference System (ANFIS) [130] implementing a Takagi-Sugeno fuzzy inference system [224]. The same training and test sets of the three time series were used to compare the performance of the different intelligent systems. Training and test results for neuro-fuzzy systems are depicted in Table 3.5.

Table 3.6 shows performance comparison of MLEANN and the recently developed Cutting Angle Method (CAM) of deterministic global optimization [35]. Table 3.7 compares MLEANN with multivariate adaptive regression splines (MARS), a popular regression based approach, which was the winner of the famous KDD 2000 data mining competition.

Table 3.5. Performance comparison between MLEANN and Neuro-Fuzzy Systems

Time series	**RMSE**					
	EANN		**Mamdani - NF**		**Takagi Sugeno - NF**	
	Training	**Test**	**Training**	**Test**	**Training**	**Test**
Mackey Glass	0.0004	0.0004	0.0023	0.0042	0.0019	0.0018
Gas Furnace	0.0110	0.0210	0.0140	0.0490	0.0137	0.0570
Waste Water	0.0425	0.0521	0.0019	0.0750	0.0530	0.0810

Table 3.6. Performance comparison between MLEANN and CAM

Data set	**MLEANN**			**CAM**		
	RMSE (train)	**RMSE (test)**	**Architecture**	**RMSE (train)**	**RMSE (test)**	**Architecture**
Mackey-Glass	0.0056	0.0061	2 T, 2 T*	0.0085	0.0091	4 S
Gas Furnace	0.0181	0.0290	1 T, 1 L, 1 T*	0.0173	0.0384	3 S
Waste water	0.0567	0.0591	2 L, 1 T, 1 T*	0.057	0.066	4 S

Table 3.7. Performance comparison between MLEANN and MARS

Data set	MLEANN			MARS		
	RMSE (train)	**RMSE** (test)	**Architecture**	**RMSE** (train)	**RMSE** (test)	**Architecture** (basis functions)
Gas Furnace	0.0110	0.0210	8 T, 2 T*	0.0185	0.0413	5

3.4 Discussions and Conclusions

Table 3.3 shows comparative performance between MLEANN and a conventional ANN without any architecture restriction. For Mackey glass series, using 500 epochs of BP learning, RMSE on test set was reduced by 82% (BP), 31% (SCG), 29% (QNA) and 56% (LM). At the same time, number of hidden neurons got reduced by approximately 58% (BP), 54% (SCG), 58% (QNA) and 55% for LM. LM algorithm gave the best RMSE error on test set (0.0004) even though it is highly computational expensive as demonstrated in Table 2.7.

For the gas furnace time series, RMSE on test set was reduced by 53%% (BP), 36% (SCG), 69% (QNA) and 73% (LM). Savings in hidden neurons amounted to 55% (BP), 37% (SCG), 50% (QNA) and 55% (LM). SCG training gave the best RMSE value (0.0210) for gas furnace series.

For the wastewater time series, RMSE on test set was reduced by 60% (BP), 29% (SCG), 35% (QNA) and 45% (LM). Savings in hidden neurons amounted to 25% (BP), 29% (SCG), 29% (QNA) and 36% (LM). LM learning gave the best RMSE value (0.0521) for wastewater series.

To have an empirical comparison, we deliberately terminated the training after 500 epochs (regardless of early stopping in some cases). In some cases the generalization performance could have been further improved. As depicted in Table 3.4, our experimentations with limited architecture also reveal the efficiency of MLEANN technique. The gas furnace time series and wastewater series could be learned just with 4 hidden neurons using LM algorithm. However, for Mackey glass series the results were not that encouraging when compared with the conventional design using 24 hidden neurons. Perhaps Mackey Glass series requires more hidden neurons to learn the problem within the required accuracy.

Table 3.5 - 3.7 depicts empirical comparison between MLEANN and some popular intelligent systems. As evident, MLEANN has outperformed all the intelligent systems in terms of the lowest RMSE values on test set for the time series considered.

Selection of the architecture (number of layers, hidden neurons, activation functions and connection weights) of a network and correct learning algorithm is a tedious task for designing an optimal artificial neural network. Moreover, for critical applications and hardware implementations optimal design often becomes a necessity. In this paper, we have formulated and explored; MLEANN: an adaptive computational framework based on evolutionary computation for automatic design of optimal artificial neural networks. Empirical results are promising and show the importance and efficacy of the technique.

In MLEANN, our work was mostly concentrated on the evolutionary search of optimal learning algorithms. For the evolutionary search of architectures, it will be interesting to model as co-evolving [75] sub-networks [246] instead of evolving the whole network [232]. Further, it will be worthwhile to explore the whole population information of the final generation for deciding the best solution. We used a fixed chromosome structure (direct encoding technique) to represent the connection weights, architecture, learning algorithms and its parameters. As size of the network increases, the chromosome size grows. Moreover, implementation of crossover is often difficult due to production of non-functional offspring's. Parameterized encoding overcomes the problems with direct encoding but the search of architectures is restricted to layers. In the grammatical encoding rewriting grammar is encoded. So the success will depend on the coding of grammar (rules). Cellular configuration might be helpful to explore the architecture of neural networks more efficiently. Gutierrez et al [107] has shown that their cellular automata technique performed better than direct coding.

In the Chapter 4, we will investigate how fuzzy inference systems could be used for modeling uncertainty and to make decisions from imprecise data.

Chapter 4: The Need for Adaptation of Fuzzy Inference Systems

4.0 Introduction

The human brain interprets imprecise and incomplete sensory information provided by the various perceptive organs[138]. Fuzzy set theory [254] provides a systematic calculus to deal with such vague information linguistically, and it performs numerical computation by using linguistic labels stipulated by membership functions [79] [146] [154] [183] [204]. Some works have also demonstrated the equivalence of fuzzy logic system and feedforward neural networks [67] [159]. Even though a fuzzy inference system is highly interpretable (*if-then* rules), it lacks the adaptability to deal with changing external environments [238].

This chapter begins with some fundamental theoretical aspects of fuzzy modeling [126] and how fuzzy inference systems [67] could be designed for solving practical problems. To demonstrate the difficulties in modeling fuzzy inference systems, we consider the reactive power prediction problem for automating the power flow control to a plant [7]. The effect of different membership functions (shape and quantity), fuzzy inference method, reasoning mechanism, defuzzification method etc are studied and demonstrated by modeling this problem. We further illustrate how this could be overcome by adaptation of fuzzy inference systems using evolutionary search procedures.

4.1 Fuzzy Sets and *If-Then* Rules

The world of information is surrounded by uncertainty and imprecision. The human reasoning process can handle inexact, uncertain and vague concepts in an appropriate manner. Usually, the human thinking, reasoning and perception process cannot be expressed precisely. These types of experiences can rarely be expressed or measured using statistical or probability theory. Fuzzy logic provides a framework to model uncertainty, human way of thinking, reasoning and the perception process [176] [204]. Fuzzy systems was first introduced by Lotfi

A Zadeh, a professor at the University of California at Berkeley, in his seminal paper published in 1965 [253].

Let X be a space of objects and x be a generic element of X. A classical set A, $A \subseteq X$, is defined as a collection of elements or objects $x \in X$, such that x can either belong or not belong to the set A. A fuzzy set A in X is defined as a set of ordered pairs

$$A = \{(x, \mu_A(x)) \mid x \in X\} \tag{4.1}$$

Where $\mu_A(x)$ is called the membership function (MF) for the fuzzy set A. The MF maps each element of X to a membership grade (or membership value) between 0 and 1. Obviously, (4.1) is a simple extension of the definition of a classical set in which the characteristic function is permitted to have any values between 0 and 1. Corresponding to the ordinary set operations of union and intersection, fuzzy sets have similar operations as illustrated in Figures 4.1 (a - c).

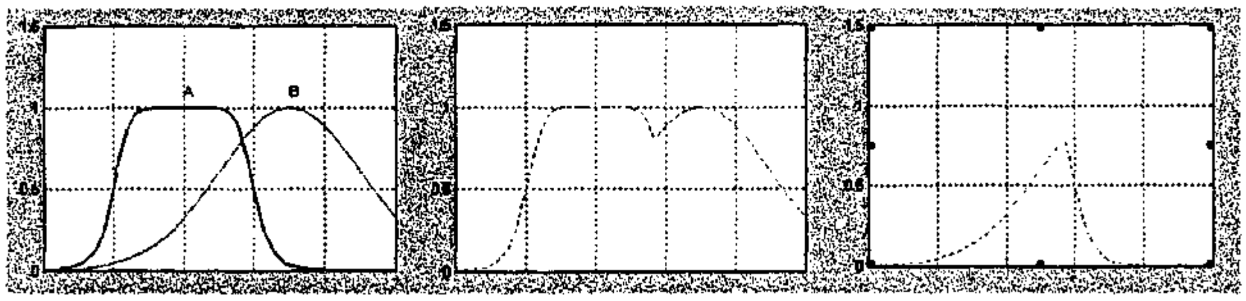


Figure 4.1. (a) Fuzzy sets A and B **(b)** $A \cup B$ **(c)** $A \cap B$

The intersection of two fuzzy sets A and B is a fuzzy set C, denoted by $C = A \cap B$, or $C = A$ **AND** B, whose MF is related to those of A and B by

$$\mu_C(x) = \min(\mu_A(x), \mu_B(x)) = \mu_A(x) \wedge \mu_B(x) \tag{4.2}$$

The intersection of two fuzzy sets A and B is specified in general by a function T: $[0,1] \times [0,1] \rightarrow [0,1]$, which aggregates two membership grades as follows:

$$\mu_{A \cap B}(x) = T(\mu_A(x), \mu_B(x)) = \mu_A(x) \bar{*} \mu_B(x) \tag{4.3}$$

where $\bar{*}$ is a binary operator for the function *T*. This class of fuzzy intersection operators are usually referred to as T-norm (triangular norm) operators [106] [131]. Four of the most frequently used T-norm operators are

$$\textit{Minimum}: T_{min}(a,b) = min(a,b) = a \wedge b \tag{4.4}$$

$$\textit{Algebraic product}: T_{ap}(a,b) = ab \tag{4.5}$$

$$\textit{Bounded product}: T_{bp}(a,b) = 0 \vee (a+b-1) \tag{4.6}$$

$$\textit{Drastic product}: T_{dp}(a,b) = \begin{cases} a, \textit{if } b = 1 \\ b, \textit{if } a = 1 \\ 0, \textit{if } a,b < 1 \end{cases} \tag{4.7}$$

The union of two fuzzy sets *A* and *B* is a fuzzy set *C*, denoted by $C = A \cup B$, or $C = A$ **OR** *B*, whose MF is related to those of *A* and *B* by

$$\mu_C(x) = max(\mu_A(x), \mu_B(x)) = \mu_A(x) \vee \mu_B(x) \tag{4.8}$$

Like intersection the fuzzy union operator is specified in general by a function *S*: [0,1] × [0,1] → [0,1], which aggregates two membership grades as follows:

$$\mu_{A \cup B}(x) = S(\mu_A(x), \mu_B(x)) = \mu_A(x) \bar{+} \mu_B(x) \tag{4.9}$$

where $\bar{+}$ is the binary operator for the function *S*. This class of fuzzy union operators are often referred to as T-conorm (or *S-norm*) operators. Four of the most frequently used T-conorm operators are

$$\textit{Maximum}: S_{max}(a,b) = max(a,b) = a \vee b \tag{4.10}$$

$$\textit{Algebraic sum}: S_{as}(a,b) = a + b - ab \tag{4.11}$$

$$\textit{Bounded sum}: S_{bs}(a,b) = 1 \wedge (a+b) \tag{4.12}$$

$$\text{Drastic sum: } S_{ds}(a,b) = \begin{cases} a, \text{if } b = 0 \\ b, \text{if } a = 0 \\ 1, \text{if } a,b > 0 \end{cases} \qquad (4.13)$$

Both the intersection and union operators retain some properties of the classical set operation. In particular, they are associative and commutative.

A more general concept, which plays an important role is the fuzzy conditional statements or the fuzzy *if-then* rules. They are expressions of the form "**If *x* is *A* then *y* is *B***", where A and B are linguistic values defined by fuzzy sets on universe of discourse X and Y respectively. Often ***x is A*** is called the antecedent or premise, while ***y is B*** is called the consequence or conclusion. Due to their concise form, they are often employed to capture the imprecise mode of reasoning which plays an essential role in the human ability to make decision in an environment of uncertainty and imprecision. The compositional rule of inference plays a key role in fuzzy reasoning. The basic rule of inference in traditional two-valued logic is modus ponens, according to which we can infer the truth of a proposition B from the truth of A and the implication $A \rightarrow B$.

Let A, A′, and B be fuzzy sets of X, X', and Y respectively. Assume that the fuzzy implication $A \rightarrow B$ is expressed as a fuzzy relation R on $X \times Y$. Then the fuzzy set induced by "*x* is *A*" and the fuzzy rule "**If *x* is *A* then *y* is *B***" is defined by

$$\mu_B(y) = max_x min[\mu_{A'}(x), \mu_R(x,y)] = \vee_x [\mu_{A'}(x) \wedge \mu_R(x,y)] \qquad (4.14)$$

Fuzzy *if-then* rules and fuzzy reasoning are the backbone of fuzzy inference systems, which are the most important modeling tools based on fuzzy set theory. Figure 4.1 shows a basic configuration of a fuzzy inference system. They are a fuzzification interface, a fuzzy rule base (knowledge base), an inference engine (decision-making logic), and a defuzzification interface.

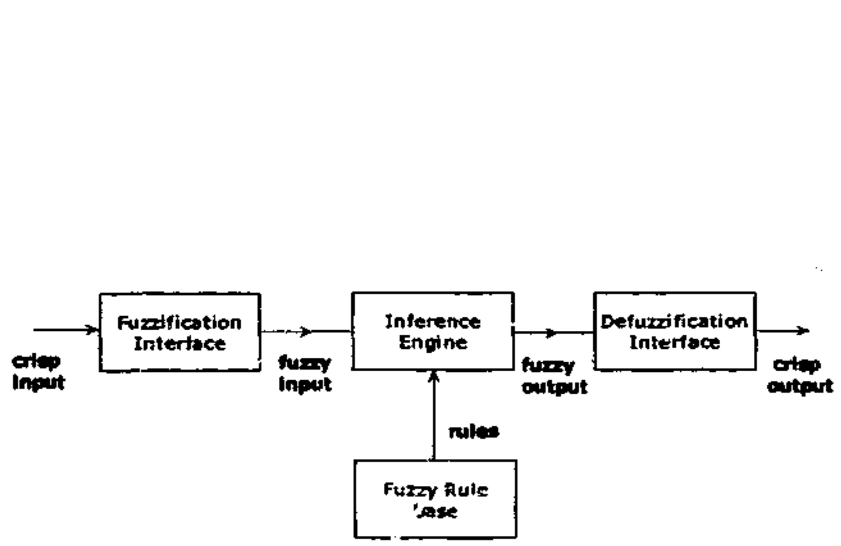


Figure 4.2. Basic architecture of a fuzzy inference system

The fuzzy rule base is characterized in the form of *if-then* rules in which preconditions and consequents involve linguistic variables. The collection of these fuzzy rules forms the rule base for the fuzzy logic system. The basic fuzzy inference system can take either fuzzy inputs or crisp inputs, but the outputs it produces are always fuzzy sets. The defuzzification task extracts the crisp output that best represents the fuzzy set. With crisp inputs and outputs, a fuzzy inference system implements a nonlinear mapping from its input space to output space through a number of fuzzy *if-then* rules. In what follow, we shall introduce the two most popular fuzzy inference systems that have been widely deployed in various applications. The differences between these two fuzzy inference systems lie in the consequents of their fuzzy rules, and thus their aggregation and defuzzification procedures differ accordingly.

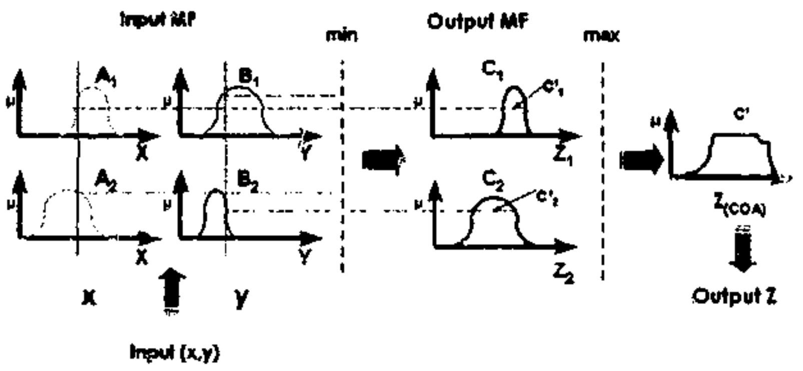


Figure 4.3. Mamdani fuzzy inference system using min and max for T-norm and T-conorm operators

Most fuzzy systems employ the inference method proposed by Mamdani [169] in which the rule consequence is defined by fuzzy sets (Figure 4.3) and has the following structure:

$$if\ x\ is\ A_1\ and\ y\ is\ B_1\ then\ z_1 = C_1' \qquad (4.15)$$

There are several defuzzification techniques. However, the most widely used defuzzification technique uses the centroid of area method as follows

$$\text{Centroid of area } Z_{COA} = \frac{\int_Z \mu_A(z)\, z\, dz}{\int_Z \mu_A(z)\, dz} \tag{4.16}$$

where $\mu_A(z)$ is the aggregated output MF.

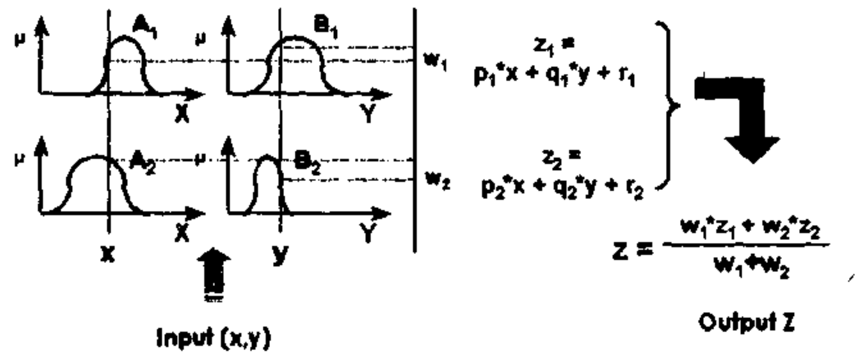


Figure 4.4. Takagi-Sugeno fuzzy inference system using a min or product as T-norm operator.

Takagi, Sugeno and Kang (TSK) proposed an inference scheme in which the conclusion of a fuzzy rule is constituted by a weighted linear combination of the crisp inputs rather than a fuzzy set [224]. A basic Takagi-Sugeno fuzzy inference system is illustrated in Figure 4.4 and the rules has the following structure

$$\textit{if } x \textit{ is } A_1 \textit{ and } y \textit{ is } B_1, \textit{then } z_1 = p_1 x + q_1 y + r_1 \tag{4.17}$$

where p_1, q_1 and r_1 are linear parameters. TSK fuzzy controller usually needs a smaller number of rules, because their output is already a linear function of the inputs rather than a constant fuzzy set.

4.2 Fuzzy Modeling

Fuzzy modeling can be pursued using the following steps.

- Select relevant input and output variables. Determine the number of linguistic terms associated with each I/O variables. Also choose the appropriate family of parameterized MFs, fuzzy operators, reasoning mechanism etc.

- Choose a specific type of fuzzy inference system
- Design a collection of fuzzy *if-then* rules (knowledge base)

It is typically advantageous if the fuzzy rule base is adaptive to a certain application. In the following section we will try to demonstrate the difficulties to design fuzzy inference system s for solving practical problems.

4.2.1 Designing Fuzzy Inference Systems in Practice

We attempted to develop a power factor forecast model using Mamdani and TSK fuzzy inference systems for automating the control of reactive power flow [7]. Utility companies base their tariffs upon the total amount of power provided, including kilowatts (KW) and kilo volt-amperes reactive (KVAR), though in most cases KW is the only utilized energy. By providing appropriate reactive power compensation methods, using power capacitors, to increase power factor ratings, the utility customer is able to substantially reduce these charges, improve the utilization of electrical power and decrease the risk of downtime within the plant. Usually power capacitors are turned on manually by human operators or by timer controlled switching relays [179]. The load consumption of manufacturing plants follow a similar pattern every day as long as the production capacity is unaltered. By knowing the load at a particular time instant, if we are able to forecast the reactive power requirement at the next time instant we will be able to switch on the required quantity of power capacitors and thereby avoid inefficient (using operators or timer controlled) switching of power capacitors. The developed model would forecast the reactive power at time $t+1$ just by knowing the load current at time t. The prediction models were trained with a 24-hour load demand pattern of a heavy automobile manufacturing plant and performance of the proposed method is evaluated by comparing the test results with the known values of reactive power [8].

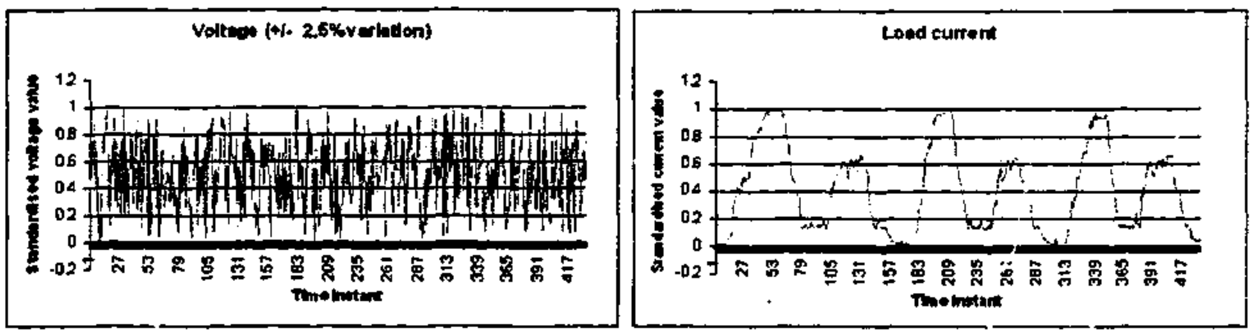


Figure. 4.5. Testing data **(a)** input voltage **(b)** load current

The experimental system consists of two stages: Developing the fuzzy inference system and performance evaluation using the test data. A heavy automobile manufacturing plant was considered for the prediction of reactive power. The dataset comprises of 24-hour load flow patterns (1440 data sets) representing the 24-hour period. The input parameters considered are the phase voltage (*V*) and current (*I*). The normal value of input parameter voltage (*V*) was fluctuated with +/- 2.5% of the normal value. All the data sets were scaled to (0-1). The input voltage was fluctuated to test the modeling capability and robustness of the fuzzy inference system. As shown in Figure 4.5, fluctuated voltage appears to be a heavy noise to the FIS. This also ensures that the developed FIS could predict the reactive power accurately even during worst conditions in the grid voltage regardless of the plant load. Training and testing data sets were extracted randomly from the complete dataset. 60% of data was used for training and remaining 40% for testing.

4.2.1.1 Design and Experimentations: Fuzzy Inference Systems

In this section we will analyze the effects (a) shape and quantity of membership functions (b) T-norm and T-conorm operators (c) defuzzification methods and (d) inference method for designing the FIS. Experimentations were carried out using 4 different settings using the same rule base and are reported as follows:

Experiment 1 (To evaluate the effect on the number of MFs)

We used the following setting for designing the FIS:

1. 2 triangular MF's for each input variable and 4 triangular MF's for the output variable (power). The rule base consisted of 4 *if-then* rules.
2. 3 triangular MF's for each input variable and 9 triangular MF's for the output variable (power). The rule base consisted of 9 *if-then* rules.

We used "*min*" and "*max*" as T-norm and T-conorm operators and the centroid method of defuzzification for Mamdani FIS and weighted average defuzzification method for Takagi-Sugeno FIS. The developed fuzzy inference systems using Mamdani and Takagi-Sugeno models are depicted in Figures 4.6 – 4.9. Table 4.1 summarizes the training and testing RMSE values.

Table 4.1. Empirical comparison of fuzzy inference systems and quantity of MFs

No. of MF's	**Mamdani FIS**		**Takagi - Sugeno FIS**	
	RMSE		**RMSE**	
	Training	**Test**	**Training**	**Test**
2	0.401	0.397	0.024	0.023
3	0.348	0.334	0.017	0.016

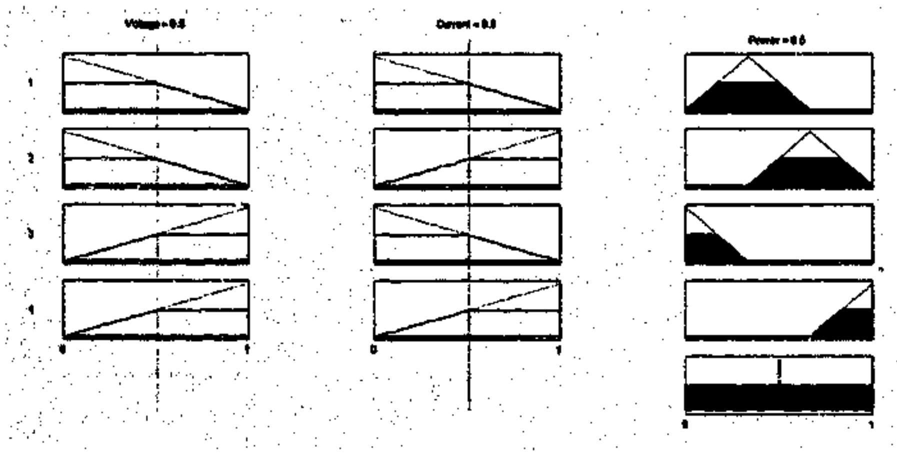


Figure 4.6. Mamdani fuzzy inference system using two triangular MF's for input variables

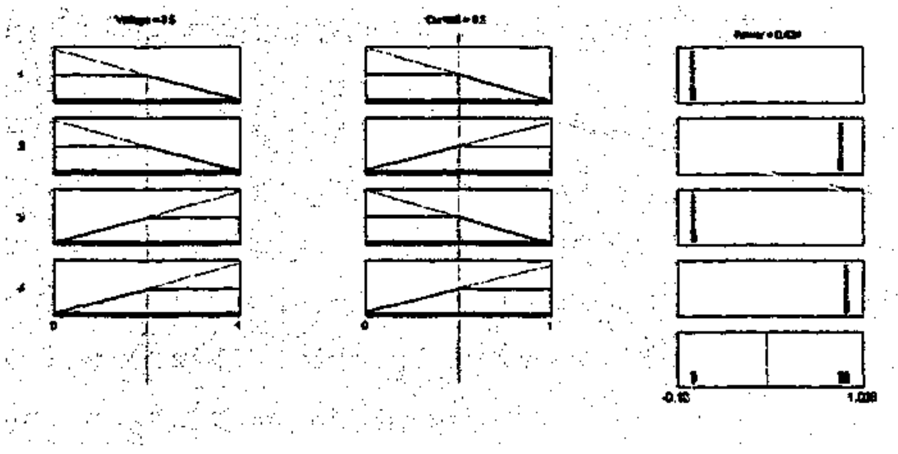


Figure 4.7. Takagi-Sugeno fuzzy inference system using 2 triangular MF's for input variables

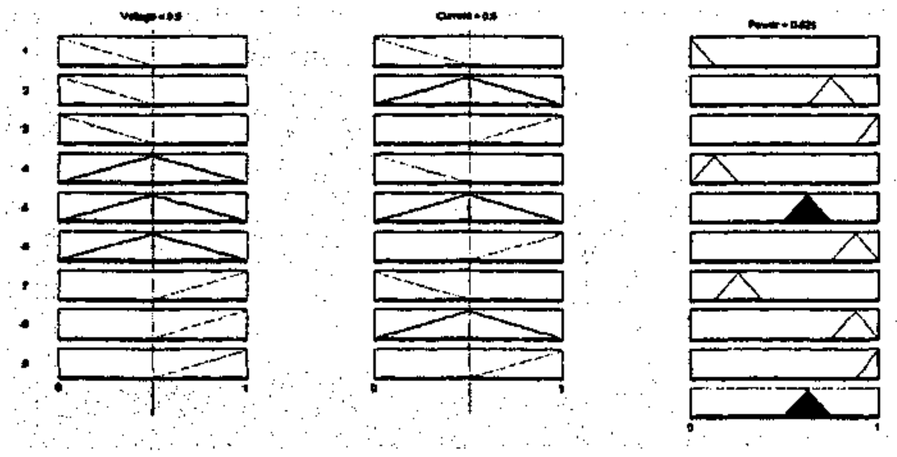


Figure 4.8. Mamdani fuzzy inference system using 3 triangular MF's for input variables

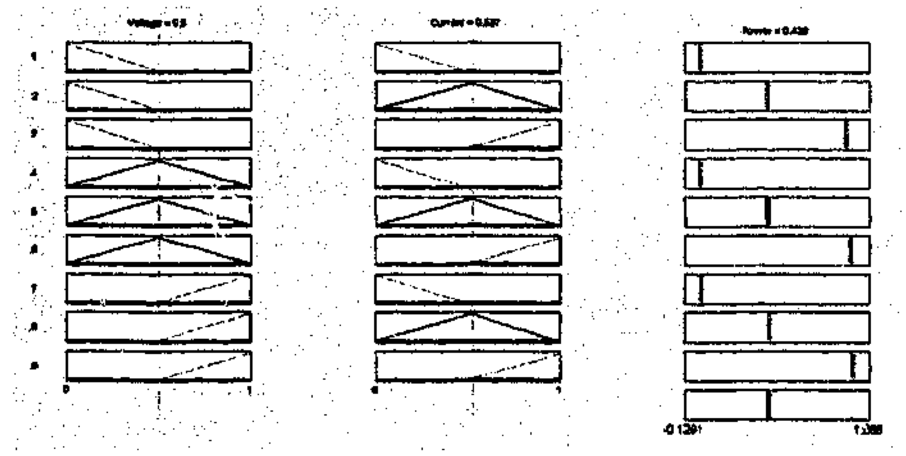


Figure 4.9. Takagi-Sugeno fuzzy inference system using 3 triangular MF's for input variables

Experiment 2 (To evaluate the effect of shape of MFs)

We used 3 Gaussian MF's for each input variable and 9 Gaussian MF's for the output variable. The rule base consisted of 9 *if-then* rules. We used "*min*" and "*max*" as T-norm and T-conorm operators and the centroid method of defuzzification for Mamdani FIS and weighted average defuzzification method for Takagi-Sugeno FIS. The developed fuzzy inference systems using Mamdani and Takagi-Sugeno models are depicted in Figures 4.10 – 4.11. Table 4.2 summarizes the training and testing RMSE values.

Table 4.2. Empirical comparison of fuzzy inference systems using Gaussian MF's

No. of MF's	Mamdani FIS		Takagi - Sugeno FIS	
	RMSE		RMSE	
	Training	Test	Training	Test
3	0.243	0.240	0.021	0.019

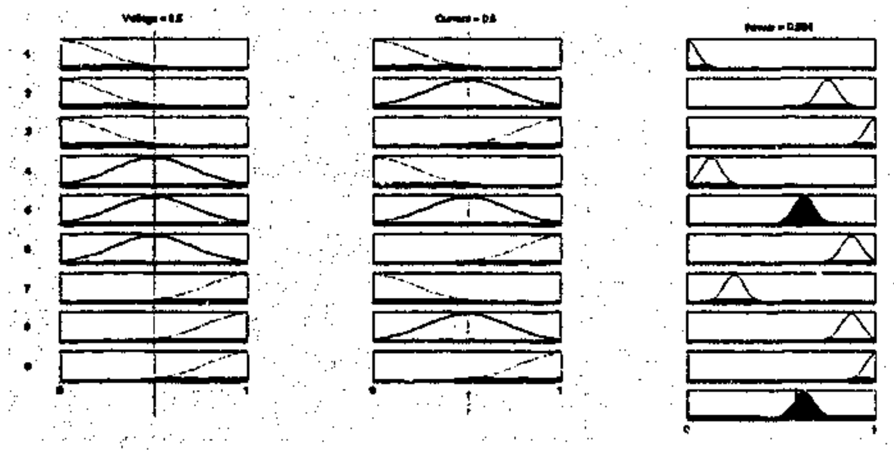


Figure 4.10. Mamdani fuzzy inference system using 3 Gaussian MF's for input variables

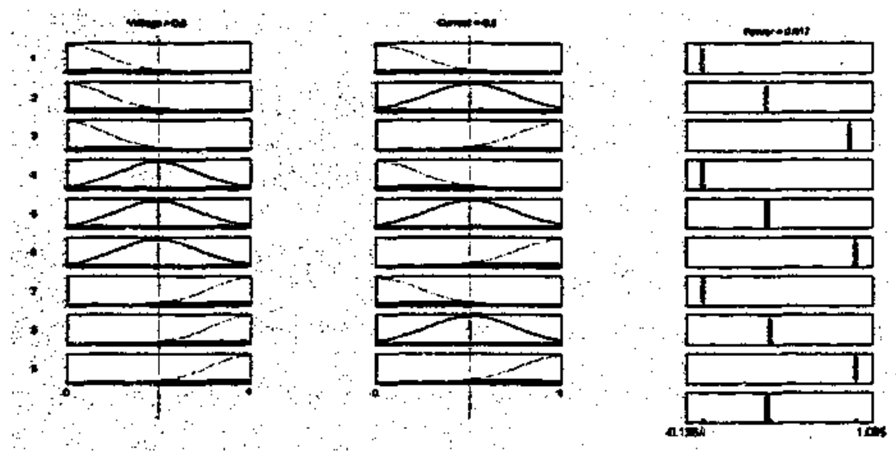


Figure 4.11. Takagi-Sugeno fuzzy inference system using 3 Gaussian MF's for input variables

Experiment 3 (To evaluate the effect of fuzzy operators)

We used 3 Gaussian MF's for each input variable and 9 Gaussian MF's for the output variable. The rule base consisted of 9 *if-then* rules. We used "product" and "sum" as T-norm and T-conorm operators and the centroid method of defuzzification for Mamdani FIS and weighted average defuzzification method for Takagi-Sugeno FIS. Table 4.3 summarizes the training and testing RMSE values.

Table 4.3. Empirical comparison of fuzzy inference systems for different fuzzy operators

No. of MF's	Mamdani FIS		Takagi - Sugeno FIS	
	RMSE		RMSE	
	Training	Test	Training	Test
3	0.221	0.219	0.019	0.018

Experiment 4 (To evaluate the effect of defuzzification operators)

We used 3 Gaussian MF's for each input variable and 9 Gaussian MF's for the output variable. The rule base consisted of 9 *if-then* rules. We used "product" and "sum" as T-norm and T-conorm operators and the following defuzzification operators for Mamdani FIS.

- Centroid
- Bisector of Area (BOA)
- Mean of Maximum (MOM)
- Smallest of Maximum (SOM)

For Takagi-Sugeno FIS, the weighted sum and weighted average defuzzification method were used. Table 4.4 summarizes the training and testing RMSE values.

Table 4.4. Empirical comparison of fuzzy inference systems for different defuzzification operators.

Mamdani FIS			**Takagi - Sugeno FIS**		
Defuzzification	**RMSE**		**Defuzzification**	**RMSE**	
	Training	**Test**		**Training**	**Test**
Centroid	0.221	0.0219	Weighted sum	0.019	0.018
MOM	0.230	0.232	Weighted average	0.085	0.084
BOA	0.218	0.216			
SOM	0.229	0.232			

4.2.1.2 Discussions of Results and Problem Solution

In this section we would like to evaluate and summarize the various experimentation results reported in Section 4.2.1.1. As depicted in Table 4.1, when the number of input MFs were increased from 2 to 3, the RMSE values reduced regardless of the inference system used. However, when the shape of the MF was changed to Gaussian, Mamdani FIS improved the RMSE but the RMSE values increased for Takagi-Sugeno FIS (Table 4.2). Using Gaussian MFs, when the T-norm and T-Conorm operators were changed to "product" and "sum" (instead of "min" and "max") both the inference methods performed better. This is reported in

Experimentation 3 (Table 4.3). Finally the selection of an ideal defuzzification operator also has a direct influence in the performance of FIS as shown in Table 4.4. BOA defuzzification method gave the lowest RMSE value while the weighted sum seems to work well for Takagi Sugeno FIS.

The design of the rule base (number of rules and how the inputs and outputs are related) is also very important for the good performance of FIS. For this problem, the rule base was created based on author's previous knowledge. When expert knowledge is not available, initial rule base could be created with the assistance of some input space-partitioning or clustering algorithm. The role of weighting factors emphasizing the importance of certain rules also bear a prominent role for the overall performance.

We have considered only two most popular fuzzy inference methods. The problem would become more complicated when we have to consider other FIS models [233].

We have demonstrated the difficulties in designing a fuzzy inference system for a function approximation problem involving just 2 inputs and 1 output. When the input / output dimensions becomes larger, manual design becomes tedious and sometimes could even lead to poor design and implementation. This leads us to the following questions:

- What is the best shape and number of membership functions for each I/O variable?
- How to design the knowledge base (size and optimal combination of *if-then* rules)?
- What are best combinations of fuzzy operators (implication/aggregators) and defuzzification operators?
- Which inference system will give the best results?

In section 4.3, we will focus on how the various component/parameter design process could be adapted according to the problem environment in order to automate the FIS design faster and efficient.

4.3 Adaptation in Fuzzy Inference Systems

A conventional fuzzy controller makes use of a model of the expert who is in a position to specify the most important properties of the process. Expert knowledge is often the main source to design the fuzzy inference systems. Figure 4.12 shows the architecture of the fuzzy inference system controlling a process. According to the performance measure of the problem environment, the MFs, rule bases and the inference mechanism are to be adapted.

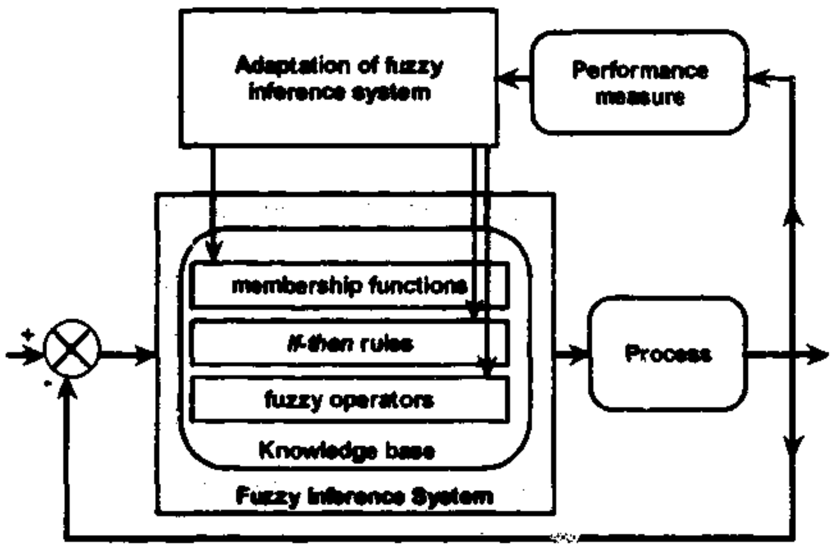


Figure 4.12. Architecture of adaptive fuzzy inference systems

Several research works are going on exploring the adaptation of fuzzy inference systems [1] [2] [24] [47] [65] [104] [162] [166] [196] [212] [229] [239] . These include the adaptation of membership functions, rule bases, aggregation operator etc. These techniques include but are not limited to:

- Self-organizing process controller by Procyk et al [210], which considered the issue of rule generation and adaptation.
- Gradient descent and its variants have been applied to fine-tune the parameters of the input and output membership functions [197] [240].
- Pruning the quantity and adapting the shape of input/output membership functions [241].
- Tools to identify the structure of fuzzy models [225].
- Fuzzy discretization and clustering techniques [252].
- In most cases the inference of the fuzzy rules is carried out using the '*min*' and '*max*' operators for fuzzy intersection and union. If the T-norm and T-conorm operators are

parameterized then gradient descent technique could be used in a supervised learning environment to fine-tune the fuzzy operators.

To formulate the initial rule base, the input space is divided into multi-dimensional partitions and then assign actions to each of the partitions. In most applications, the partitioning is achieved using one dimensional membership functions using fuzzy *if-then* rules as illustrated in Figure 4.13. The consequent parts of the rule represent the actions associated with each partition. It is evident that the MFs and the number of rules are tightly related to the partitioning.

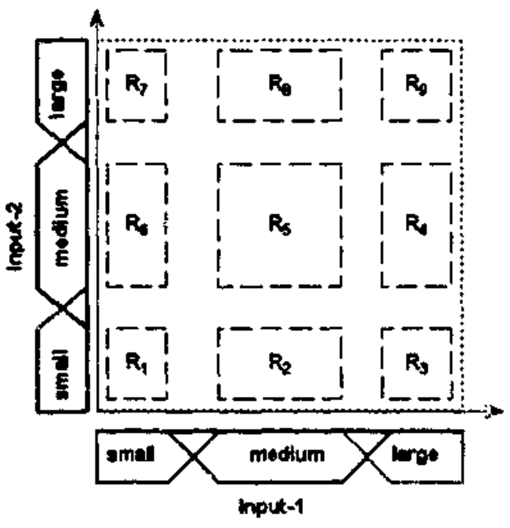


Figure 4.13. Example showing how the 2 dimensional spaces are partitioned using 3 trapezoidal membership functions per input dimension. A simple if-then rule will appear as ***If input-1 is medium and input 2 is large then rule R_8 is fired.***

Adaptation of fuzzy inference systems using evolutionary computation techniques has been widely explored [72] [175] [202] [213]. We proposed an adaptive framework based on evolutionary computation wherein the membership functions, rule base and fuzzy operators are adapted according to the problem [10]. The evolutionary search of MFs, rule base, fuzzy operators etc would progress on different time scales to adapt the fuzzy inference system according to the problem environment. Figure 4.14 illustrates the general interaction mechanism of the proposed framework with the evolutionary search of fuzzy inference system (Mamdani, Takagi -Sugeno etc) evolving at the highest level on the slowest time scale. For each evolutionary search of fuzzy operators (best combination of T-norm and T-conorm, defuzzification strategy etc), the search for the fuzzy rule base progresses at a faster time scale in an environment decided by the problem. In a similar manner, evolutionary search of membership functions proceeds at a faster time scale (for every rule base) in the environment decided by the problem. The problem representation (genetic coding) is illustrated in Figure 4.15.

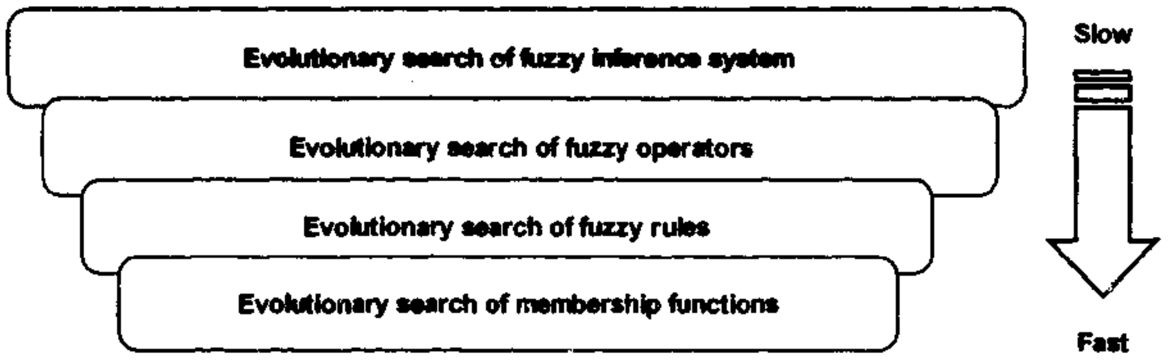

Evolutionary search of fuzzy inference system
Evolutionary search of fuzzy operators
Evolutionary search of fuzzy rules
Evolutionary search of membership functions
Slow
Fast

Figure 4.14. Interaction of evolutionary search mechanisms in the adaptation of fuzzy inference system

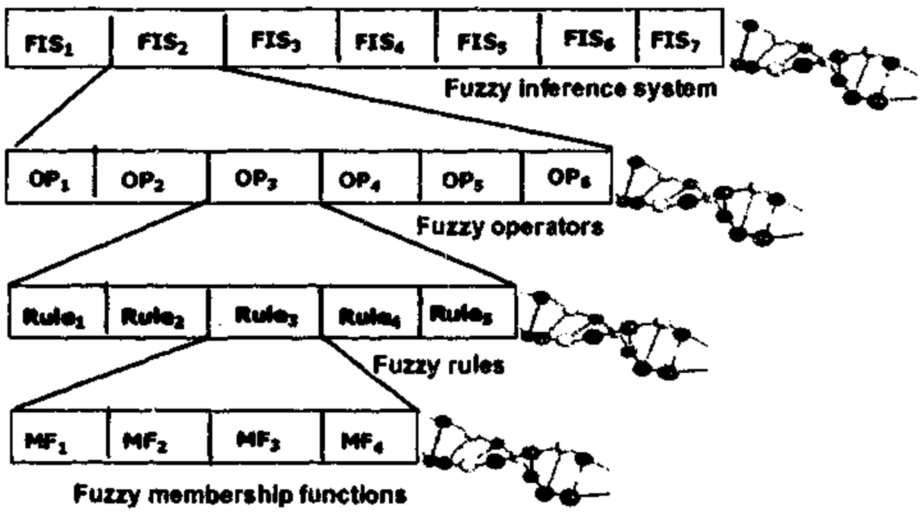

FIS1 FIS2 FIS3 FIS4 FIS5 FIS6 FIS7
Fuzzy inference system
OP1 OP2 OP3 OP4 OP5 OP6
Fuzzy operators
Rule1 Rule2 Rule3 Rule4 Rule5
Fuzzy rules
MF1 MF2 MF3 MF4
Fuzzy membership functions

Figure 4.15. Chromosome representation of the adaptive fuzzy inference system using evolutionary computation

Automatic adaptation of membership functions is popularly known as self tuning and the genome encodes parameters of trapezoidal, triangle, logistic, hyperbolic-tangent, Gaussian membership functions etc.[33] [48] [72] [115] [136] [197] [200].

Evolutionary search of fuzzy rules [23] [25] [97] [98] [116] [117] [118] [157] [185] can be carried out using three approaches. In the first method (Michigan approach) the fuzzy knowledge base is adapted as a result of antagonistic roles of competition and cooperation of fuzzy rules. Each genotype represents a single fuzzy rule and the entire population represents a solution. A classifier rule triggers whenever its condition part matches the current input, in which case the proposed action is sent to the process to be controlled. The global search algorithm will generate new classifier rules based on the rule strengths acquired during the entire process. The fuzzy behavior is created by an activation sequence of mutually collaborating fuzzy rules. The entire knowledge base is build up by a cooperation of competing multiple fuzzy rules [49].

The second method (Pittsburgh approach) evolves a population of knowledge bases rather than individual fuzzy rules [103]. Genetic operators serve to provide a new combination of rules and new rules. In some cases, variable length rule bases are used; employing modified genetic operators for dealing with these variable length and position independent genomes. The disadvantage is the increased complexity of search space and additional computational burden especially for online learning.

The third method (iterative rule learning approach) is very much similar to the first method with each chromosome representing a single rule, but contrary to the Michigan approach, only the best individual is considered to form part of the solution, discarding the remaining chromosomes in the population. The evolutionary learning process builds up the complete rule base through a iterative learning process [100].

4.4 Conclusions

In this Chapter, we have presented the fundamental concepts of fuzzy modeling and demonstrated the difficulties to design optimal fuzzy inference systems for solving practical problems. Empirical results clearly reveal the importance of the shape and quantity of membership functions for each input variable, fuzzy operators, inference method, knowledge base etc. for designing optimal fuzzy systems. In Section 4.3, we have presented the different adaptation techniques for designing fuzzy systems focusing evolutionary approach. We also presented a framework for optimal design of fuzzy inference systems based on a hierarchical evolutionary approach, which is also similar to MLEANN architecture discussed in Chapter 3.

In Chapter 5, we will present hybrid combinations of neural networks and fuzzy systems with some practical applications. We will focus on the different types of integrated neuro-fuzzy systems that has evolved during the last decade with some empirical comparison towards the end.

In Chapter 7, we will demonstrate how fuzzy inference systems could be adapted and optimized using a hybrid approach involving neural network learning algorithms and evolutionary computation techniques.

Chapter 5: Integration of Neural Networks and Fuzzy Inference Systems

5.0 Introduction

Hayashi et al [113] showed that a feedforward neural network could approximate any fuzzy rule based system and any feedforward neural network may be approximated by a rule based fuzzy inference system [36]. Fusion of artificial neural networks and fuzzy inference systems have attracted the growing interest of researchers in various scientific and engineering areas due to the growing need of adaptive intelligent systems to solve the real world problems [27] [29] [40] [57] [59] [60] [63] [74] [80] [81] [91] [94] [99] [101] [104] [105] [109] [110] [112] [114] [222]. A neural network learns from scratch by adjusting the interconnections between layers. Fuzzy inference system [67] is a popular computing framework based on the concept of fuzzy set theory, fuzzy *if-then* rules, and fuzzy reasoning. The advantages of a combination of neural networks and fuzzy inference systems are obvious [58] [61] [131] [162] [255]. An analysis reveals that the drawbacks pertaining to these approaches seem complementary and therefore it is natural to consider building an integrated system combining the concepts [143] [180]. While the learning capability is an advantage from the viewpoint of fuzzy inference system, the automatic formation of linguistic rule base will be advantage from the viewpoint of neural network. There are several works related to the integration of neural networks and fuzzy inference systems [77] [120] [122] [123] [125] [138] [148] [149] [158] [161] [181] [188] [201] [205] [220] [221] [227] [228] [236] [243] [245].

In this Chapter, we discuss the integration of neural networks and fuzzy inference systems into three main categories: Cooperative, concurrent and integrated neuro-fuzzy models [196]. We present 3 different types of cooperative neuro-fuzzy models namely fuzzy associative memories, fuzzy rule extraction using self-organizing maps and systems capable of learning fuzzy set parameters. We also illustrate a concurrent neuro-fuzzy system and a practical application on stock market analysis. Different Mamdani [169] and Takagi-Sugeno [224] type integrated neuro-fuzzy systems are further introduced with a focus on some of the salient features and advantages of the different types of neuro-fuzzy models that have been evolved during the last decade [61]. Some discussions and conclusions are also provided towards the end of the chapter.

5.1 Cooperative Neuro-Fuzzy Systems

In the simplest way, a cooperative model can be considered as a preprocessor wherein artificial neural network (ANN) learning mechanism determines the fuzzy inference system (FIS) membership functions or fuzzy rules from the training data. Once the FIS parameters are determined, ANN goes to the background.

Fuzzy Associative Memories (FAM) by Kosko [153], fuzzy rule extraction using self organizing maps by Pedrycz et al [203] and the systems capable of learning of fuzzy set parameters by Nomura et al [197] are some good examples of cooperative neuro-fuzzy systems.

5.1.1 Fuzzy Associative memories

Kosko interprets a fuzzy rule as an association between antecedent and consequent parts [153]. If a fuzzy set is seen as a point in the unit hypercube and rules are associations, then it is possible to use neural associative memories to store fuzzy rules. A neural associative memory can be represented by its connection matrix. Associative recall is equivalent to multiplying a key factor with this matrix. The weights store the correlations between the features of the key k and the information part i. Due to the restricted capacity of associative memories and because of the combination of multiple connection matrices into a single matrix is not recommended due to severe loss of information, it is necessary to store each fuzzy rule in a single FAM. Rules with n conjunctively combined variables in their antecedents can be represented by n FAMs, where each stores a single rule. The FAMs are completed by aggregating all the individual outputs (maximum operator in the case of Mamdani fuzzy system) and a defuzzification component.

Learning could be incorporated in FAM, as learning the weights associated with FAMs output or to create FAMs completely by learning. A neural network-learning algorithm determines the rule weights for the fuzzy rules. Such factors are often interpreted as the influence of a rule and are multiplied with the rule outputs. Rule weights can be replaced equivalently by modifying the membership functions. However, this could result in misinterpretation of fuzzy sets and identical linguistic values might be represented differently in different rules. Kosko suggests a form of adaptive vector quantization technique to learn the FAMs. This approach is termed as differential competitive learning and is very similar to the learning in self-organizing maps.

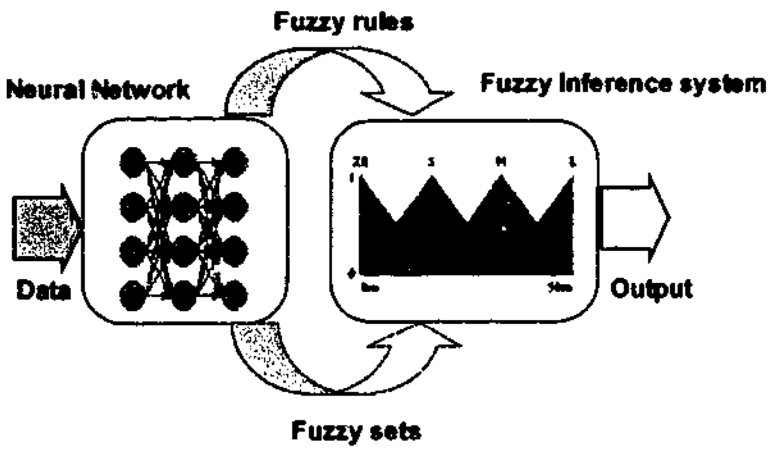


Figure 5.1. Cooperative neuro-fuzzy model

Figure 5.1 depicts a cooperative neuro-fuzzy model where the neural network learning mechanism is used to determine the fuzzy rules, parameters of fuzzy sets, rule weights etc. Kosko's adaptive FAM is a cooperative neuro-fuzzy model because it uses a learning technique to determine the rules and its weights. The main disadvantage of FAM is the weighting of rules. Just because certain rules, doesn't have much influence doesn't mean that they are totally unimportant. Hence the reliability of FAMs for certain applications is questionable. Due to implementation simplicity FAMs are used in many applications.

5.1.2 Fuzzy Rule Extraction Using Self Organizing Maps

Pedryz et al [203] used self-organizing maps with a planar competition layer to cluster training data, and they provide means to interpret the learning results. The learning results could show whether two input vectors are similar to each other or belong to the same class. However, in the case of high-dimensional input vectors, the structure of the learning problem can rarely be detected in the two dimensional map. Pedryz et al provides a procedure for interpreting the learning results using linguistic variables.

After the learning process, the weight matrix W represents the weight of each feature of the input patterns to the output. Such a matrix defines a map for a single feature only. For each feature of the input patterns, fuzzy sets are specified by a linguistic description B (one fuzzy set for each variable). They are applied to the weight matrix W to obtain a number of transformed matrices. Each combination of linguistic terms is a possible description of a pattern subset or cluster. To check a linguistic description B for validity, the transformed maps are intersected and a matrix D is obtained. Matrix D determines the compatibility of the learning result with the linguistic description B. $D^{(B)}$ is a fuzzy relation, and $d^{(B)}$ is interpreted as the degree of support of B. By describing $D^{(B)}$ by its α-cuts D_{α}^{B} [204] one obtains subsets of output nodes, whose degree of membership is at least α such that the confidence of all patterns X_a belong to the class described by B vanishes with decreasing α. Each B is a valid description of a cluster if $D^{(B)}$ has a non-empty α-cut D_{α}^{B}. If the features are separated into input and output features according to the application considered, then each B represents a linguistic rule, and by examining each combination of linguistic values a complete fuzzy rule base can be created. This method also shows which patterns belong to a fuzzy rule, because they are not contained in any subset X_a. An important advantage when compared to FAMs is that the rules are not weighted. The problem is with the determination of the number of output neurons and the α values for each learning problem. Compared to FAM, since the form of the membership function determines a crucial role in the performance the data could be better exploited. Since Kosko's learning procedure doesn't take into account of the neighborhood relation between the output neurons, perfect topological mapping from the input patterns to the output patterns might not be obtained sometimes. Thus the FAM learning procedure is more dependent on the sequence of the training data than Pedryz et al procedure.

Pedryz et al initially determines the structure of the feature space and then the linguistic descriptions best matching the learning results by using the available fuzzy partitions are obtained. If a large number of patterns fit none of the descriptions, this may be due to an insufficient choice of membership functions and they can be determined anew. Hence for learning the fuzzy rules this approach is preferable compared to FAM [42]. Performance of this method still depends on the learning rate and the neighborhood size for weight modification, which is problem dependant and could be determined heuristically. Fuzzy c-means algorithm also has been explored to determine the learning rate and neighborhood size by Bezdek et al [42] and Höppner et al [121].

5.1.3. Systems Capable of Learning Fuzzy Set Parameters

Nomura et al [197] proposed a supervised learning technique to fine-tune the fuzzy sets of an existing Sugeno type fuzzy system. Parameterized triangular membership functions were used for the antecedent part of the fuzzy rules. The learning algorithm is a gradient descent procedure that uses an error measure *E* (difference between the actual and target outputs) to fine-tune the parameters of the MF. Because the underlying fuzzy system uses neither a defuzzification procedure nor a non-differentiable t-norm to determine the fulfillment of rules, the calculation of the modifications of the MF parameters. The procedure is very similar to the delta rule for multilayer perceptrons. The learning takes place in an offline mode. For the input vector, the resulting error *E* is calculated and based on that the consequent parts (a real value) are updated. Then the same patterns are propagated again and only the parameters of the MFs are updated. This is done to take the changes in the consequents into account when the antecedents are modified. A severe drawback of this approach is that the representation of the linguistic values of the input variables depends on the rules they appear in. Initially identical linguistic terms are represented by identical membership functions. During the learning process, they may be developed differently, so that identical linguistic terms are represented by different fuzzy sets. The proposed approach is applicable only to Sugeno type fuzzy inference system. Using a similar approach, Miyoshi et al [182] adapted fuzzy T-norm and T-conorm operators [73] while Yager et al [244] adapted the defuzzification operator using a supervised learning algorithm.

5.2. Concurrent Neuro-Fuzzy System

In a concurrent model, neural network assists the fuzzy system continuously (or vice versa) to determine the required parameters especially if the input variables of the controller cannot be measured directly. Such combinations do not optimize the fuzzy system but only aids to improve the performance of the overall system. Learning takes place only in the neural network and the fuzzy system remains unchanged during this phase. In some cases the fuzzy outputs might not be directly applicable to the process. In that case neural network can act as a postprocessor of fuzzy outputs. Figure 5.2 depicts a concurrent neuro-fuzzy model where in the input data is fed to a neural network and the output of the neural network is further processed by the fuzzy system.

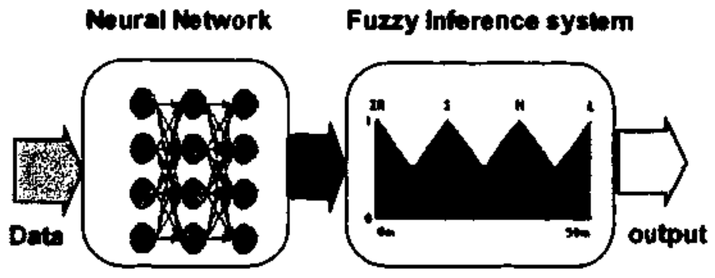


Figure 5.2. Concurrent neuro-fuzzy model

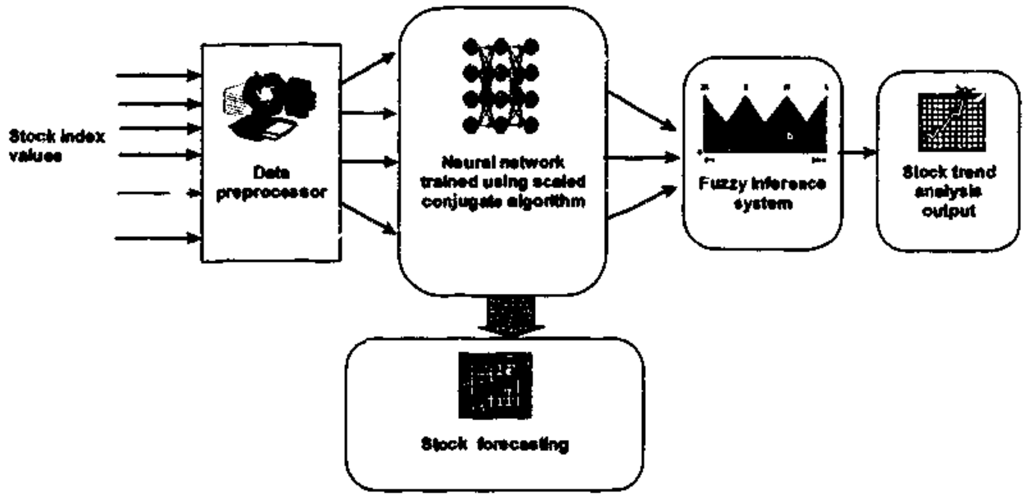

Stock index values
Data preprocessor
Neural network trained using scaled conjugate algorithm
Fuzzy inference system
Stock trend analysis output
Stock forecasting

Figure 5.3. Block diagram showing a concurrent neuro-fuzzy model for stock market analysis

5.2.1. Nasdaq Stock Market Analysis Using Concurrent Neuro-Fuzzy System

During the last decade, stocks and futures traders have come to rely upon various types of intelligent systems to make trading decisions. Several hybrid intelligent systems have in recent years been developed for modeling expertise, decision support, complicated automation tasks etc. We present a concurrent neuro-fuzzy system for predicting the stock value and the collective trend [19].

Nasdaq-100 index reflects Nasdaq's largest companies across major industry groups, including computer hardware and software, telecommunications, retail/wholesale trade and biotechnology [189]. The Nasdaq-100 index is a modified capitalization-weighted index, which is designed to limit domination of the Index by a few large stocks while generally retaining the capitalization ranking of companies. Through an investment in Nasdaq-100 index tracking stock, investors can participate in the collective performance of many of the Nasdaq stocks that are often in the news or have become household names. In this experiment, we attempt to forecast the values of six individual stocks and group index (using neural networks) as well as the trend analysis of the different stocks (using fuzzy inference system). Individual stock forecasts and group trend analysis might give some insights of the actual performance of the whole index in detail. To demonstrate the efficiency of the proposed hybrid system we considered the two years stock chart information (ending 20 March 2001) of six major industry groups listed on the national market tier of the Nasdaq Stock Market^{SM} (Nasdaq-100 index).

For the stock forecasting purpose, we made use of a neural network trained using scaled conjugate gradient algorithm. However, the forecasted stock values might deviate from the actual values. We modeled the deviation of the predicted value from the required value as a fuzzy variable and used a fuzzy inference system to account for the uncertainty and decision-making. Figure 5.3 depicts the concurrent neuro-fuzzy model for stock market analysis. We start with data preprocessing, which consists of all the actions taken before the actual data analysis process starts. The preprocessed data is fed into the neural network trained using a scaled conjugate gradient algorithm for forecasting the stock outputs.

The forecasted outputs by the neural network are further analyzed using the fuzzy inference system. This time our aim is to analyze the upward and downward trends of the different forecasted stocks. Since the forecasted values will deviate from the desired value (depending upon the prediction efficiency of the neural network), we propose to make use of the uncertainty modeling capability of fuzzy inference system. The developed fuzzy inference system is trained using the trend patterns of the different stock values. The difference between the day's stock value and the previous day was calculated and used for training the fuzzy inference system. If all the stock values were increasing we classified it as positive trend "1" and "0" otherwise. The proposed fuzzy inference system is capable of providing detailed trend analysis of individual stocks and also interdependencies of various stocks and how they affect the overall index.

5.2.1.1. Experimentation setup and test results

We considered 24 months stock data for training and analyzing the efficiency of the proposed concurrent neuro-fuzzy system. We used Nasdaq-100 main index values and six other companies listed in the Nasdaq-100 index. Apart from the Nasdaq-100 index (IXNDX); the other companies considered were Microsoft Corporation (MSFT), Yahoo! Inc. (YHOO), Cisco Systems Inc. (CSCO), Sun Microsystems Inc. (SUNW), Oracle Corporation (ORCL) and Intel Corporation (INTC). Figures 5.4 and 5.5 depict the variation of stock values for a 24 months period from 22 March 1999 to 20 March 2001.

For each *t*, the stock values *x (t)* were first scaled by the data preprocessor. 80% of the data was used for training and remaining was used for testing and validation. The same set of data was used for training and testing the fuzzy inference system. Test data was presented to the network and the output from the network was compared with the actual stock values in the time series.

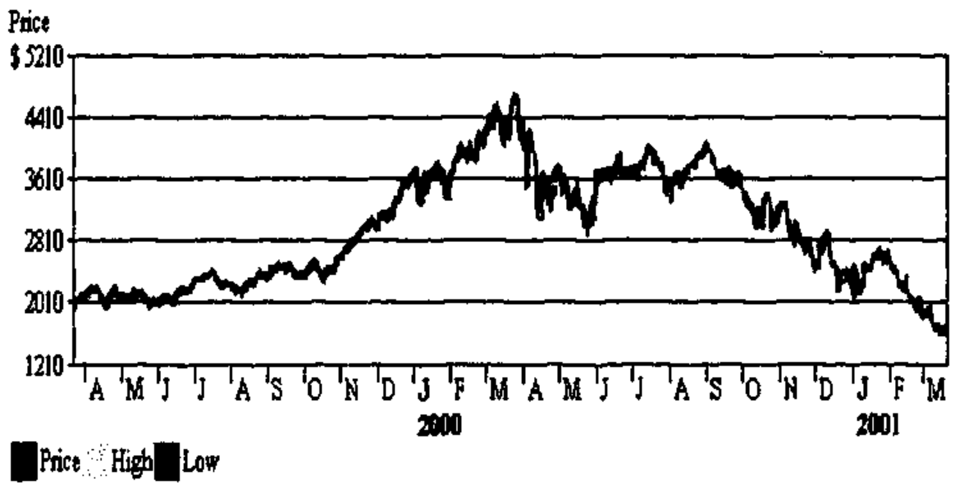


Figure 5.4. 24 months data of Nasdaq-100 index adapted from [189]

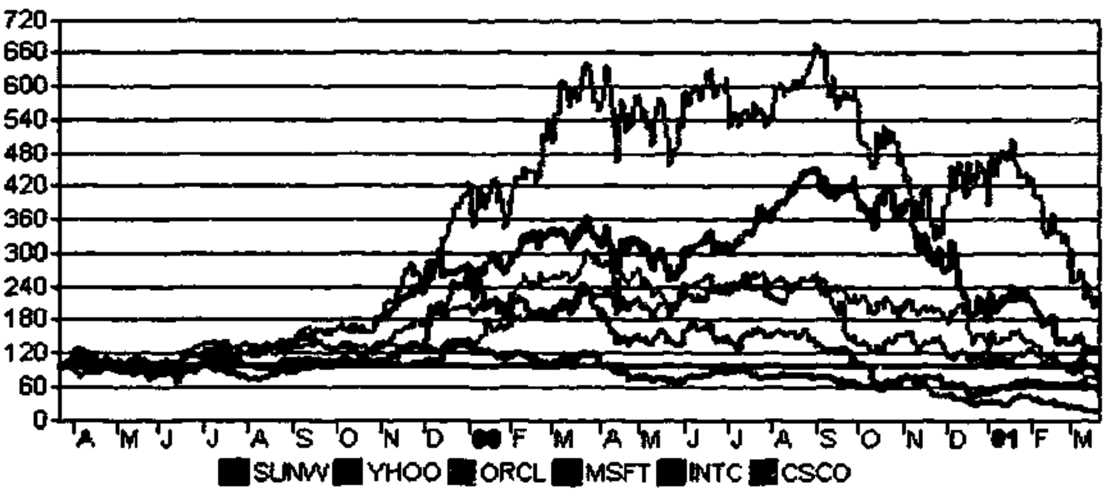


Figure 5.5. 24 months data of 6 companies adapted from [189]

- **Training the neural network**

We used a feedforward neural network with 8 input nodes and two hidden layers consisting of 20 neurons each. We used tanh-sigmoidal activation function for the hidden neurons. The training was terminated after 2000 epochs. The test data was passed through the network after the training was completed.

- **Building and training the fuzzy inference system**

Each of the input variables consists of the difference in the stock value (for example, today's value – yesterday's value). For building the fuzzy inference system, we had 8 input variables (the scaled stock value differences and the time factor). We used 4 membership functions for each of the 8 input variable and we used a neural network learning method to build up the knowledge base automatically [139]. We report only the collective trend of all the seven stock values. If all the trends were increasing we classified it as "*1*" and "*0*" otherwise.

- **Performance and Results Achieved**

Table 5.1 summarizes the training and test results achieved for the different stock values. Figure 5.6 and 5.7 depicts the test results for the prediction of Nasdaq-100 index and other company stock values. Table 5.2 summarizes the trend prediction results and Figure 5.8 illustrates the trend classification test results using the fuzzy inference system.

Table 5.1. Training and testing results using neural network

	Nasdaq	Microsoft	Sun	Cisco	Yahoo	Oracle	Intel
Testing error (RMSE)	0.028	0.034	0.023	0.030	0.021	0.026	0.034
Learning epochs	2000						
Training error (RMSE)	0.0256						

Table 5.2. Test results of trend classification using fuzzy inference system

	Actual quantity	Fuzzy system classification	% Success
Positive trends	22	22	100
Non-positive trends	78	78	100

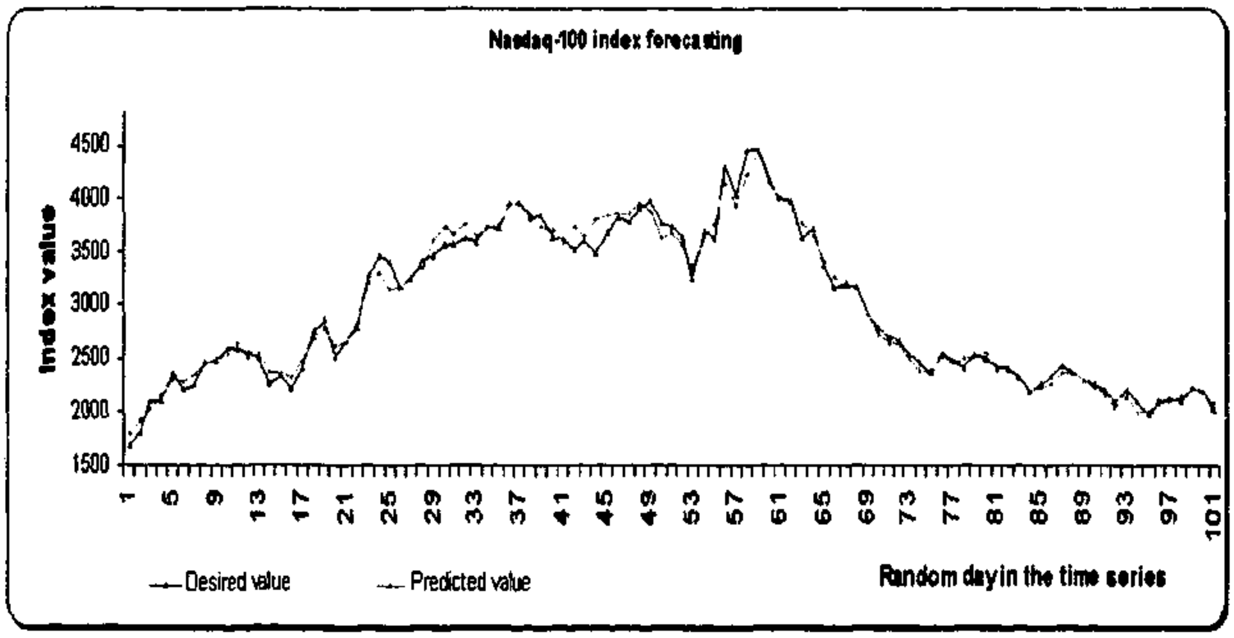


Figure 5.6. Forecast test results for Nasdaq-100 index

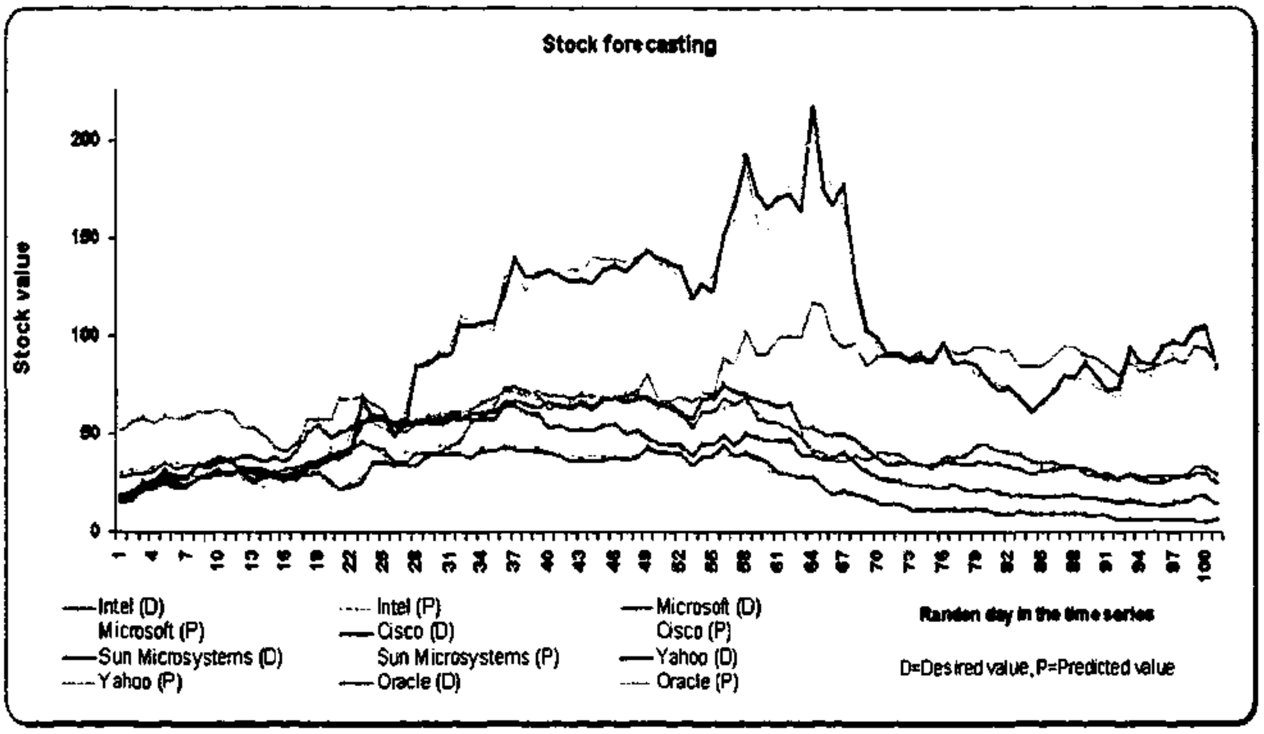


Figure 5.7. Forecast test results for the six companies listed under Nasdaq-100 index

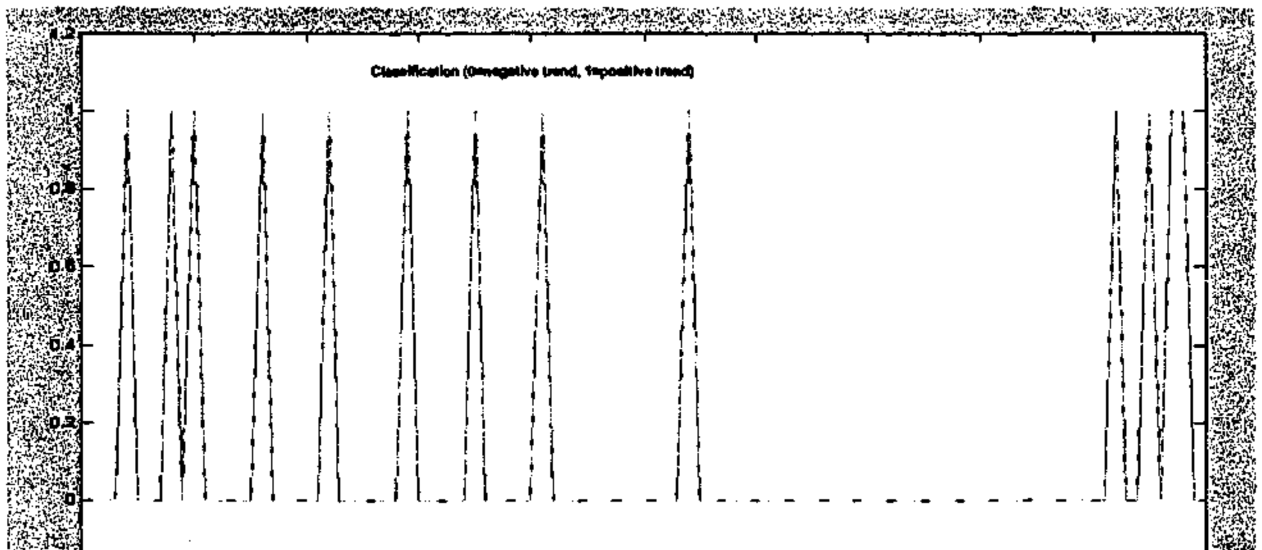


Figure 5.8. Test results for the collective trend prediction of Nasdaq-100 index and the six 6 company stock values using fuzzy inference system.

For forecasting stocks, the RMSE on test data are comparatively small showing the reliability of the developed prediction model. The fuzzy inference system also gave 100% trend prediction showing the efficiency of the technique. From the viewpoint of the stock exchange owner, participating companies, traders and investors the technique might help for better understanding of the day-to-day stock market performance. The stock forecast error could have been improved by providing more input variable information (stock volume etc) and if individual neural networks were used rather than a single network. Various trend analyses could have been done using the proposed fuzzy inference system. Some of the possible analyses are individual stock trend predictions, interdependency of different stocks with respect to the main index as well as individual companies. More details about the model and results could be obtained from [19].

5.3.Integrated Neuro-Fuzzy Systems

In an integrated model, neural network learning algorithms are used to determine the parameters of fuzzy inference systems. Integrated neuro-fuzzy systems share data structures and knowledge representations. A fuzzy inference system can utilize human expertise by storing its essential components in rule base and database, and perform fuzzy reasoning to infer the overall output value. The derivation of *if-then* rules and corresponding membership functions depends heavily on the *a priori* knowledge about the system under consideration. Howevur there is no systematic way to transform experiences of knowledge of human experts to the knowledge base of a fuzzy inference system. There is also a need for adaptability or some learning algorithms to produce outputs within the required error rate. On the other hand, neural network learning mechanism does not rely on human expertise. Due to the homogenous structure of neural network, it is hard to extract structured knowledge from either the weights or the configuration of the network. The weights of the neural network represent the coefficients of the hyper-plane that partition the input space into two regions with different output values. If we can visualize this hyper-plane structure from the training data then the subsequent learning procedures in a neural network can be reduced. However, in reality, the a priori knowledge is usually obtained from human experts and it is most appropriate to express the knowledge as a set of fuzzy if-then rules and it is very difficult to encode into an neural network. Table 5.3 summarizes the comparison between neural networks and fuzzy inference system [26].

Table 5.3. Comparison between neural networks and fuzzy inference systems

Artificial Neural Networks	**Fuzzy Inference System**
Prior rule-based knowledge cannot be used	Prior rule-base can be incorporated
Learning from scratch	Cannot learn (use linguistic knowledge)
Black box	Interpretable (if-then rules)
Complicated learning algorithms	Simple interpretation and implementation
Difficult to extract knowledge	Knowledge must be available

To a large extent, the drawbacks pertaining to these two approaches seem complementary. Therefore, it seems natural to consider building an integrated system combining the concepts of FIS and ANN modeling. A common way to apply a learning algorithm to a fuzzy system is to represent it in a special neural network like architecture. However the conventional neural network learning algorithms (gradient descent) cannot be applied directly to such a system as the functions used in the inference process are usually non differentiable. This problem can be tackled by using differentiable functions in the inference system or by not using the standard neural learning algorithm. In Section 5.3.1 –5.3.2, we will discuss how to model integrated neuro-fuzzy systems implementing Mamdani and Takagi - Sugeno FIS.

5.3.1. Integrated Neuro-Fuzzy System (Mamdani FIS)

A Mamdani neuro-fuzzy system uses a supervised learning technique (backpropagation learning) to learn the parameters of the membership functions. The detailed function of each layer (as depicted in Figure 5.9) is as follows:

- Layer -1*(input layer):* No computation is done in this layer. Each node in this layer, which corresponds to one input variable, only transmits input values to the next layer directly. The link weight in layer 1 is unity.

- Layer-2 *(fuzzification layer):* Each node in this layer corresponds to one linguistic label (excellent, good, etc.) to one of the input variables in layer 1. In other words, the output link represent the membership value, which specifies the degree to which an input value belongs to a fuzzy set, is calculated in layer 2. A clustering algorithm will decide the initial number and type of membership functions to be allocated to each of the input variable. The final shapes of the MFs will be fine tuned during network learning.

- Layer-3 *(rule antecedent layer):* A node in this layer represents the antecedent part of a rule. Usually a T-norm operator is used in this node. The output of a layer 3 node represents the firing strength of the corresponding fuzzy rule.

- Layer-4 *(rule consequent layer):* This node basically has two tasks. To combine the incoming rule antecedents and determine the degree to which they belong to the output linguistic label (high, medium, low, etc.). The number of nodes in this layer will be equal to the number of rules.

- Layer-5 *(Combination and defuzzification layer):* This node does the combination of all the rules consequents using a T-conorm operator and finally computes the crisp output after defuzzification.

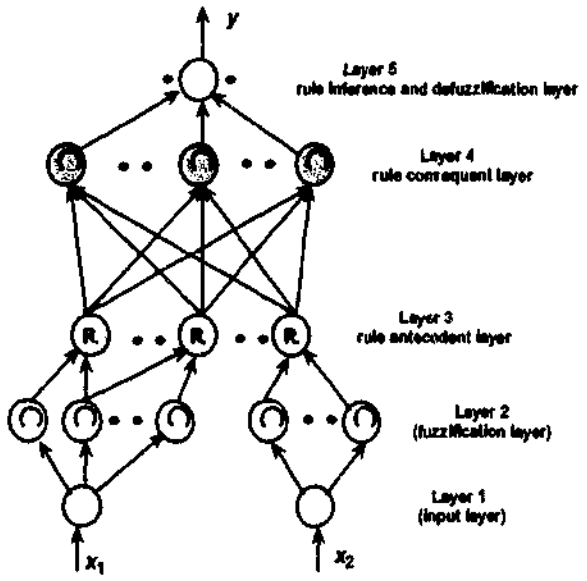


Figure 5.9..Mamdani neuro-fuzzy system

5.3.2 Integrated Neuro-fuzzy system (Takagi-Sugeno FIS)

Takagi Sugeno neuro-fuzzy systems make use of a mixture of back propagation to learn the membership functions and least mean square estimation to determine the coefficients of the linear combinations in the rule's conclusions. A step in the learning procedure got two parts: In the first part the input patterns are propagated, and the optimal conclusion parameters are estimated by an iterative least mean square procedure, while the antecedent parameters (membership functions) are assumed to be fixed for the current cycle through the training set. In the second part the patterns are propagated again, and in this epoch, back propagation is used to modify the antecedent parameters, while the conclusion parameters remain fixed. This procedure is then iterated. The detailed functioning of each layer (as depicted in Figure 5.10) is as follows:

- Layers 1,2 and 3 functions the same way as Mamdani FIS.
- Layer 4 *(rule strength normalization):* Every node in this layer calculates the ratio of the *i*-th rule's firing strength to the sum of all rules firing strength

  $$\overline{w_i} = \frac{w_i}{w_1 + w_2}, \; i = 1,2.....$$

- Layer-5 *(rule consequent layer):* Every node *i* in this layer is with a node function

  $$\overline{w_i} f_i = \overline{w_i}\,(p_i x_1 + q_i x_2 + r_i),$$

  where $\overline{w_i}$ is the output of layer 4, and $\{p_i, q_i, r_i\}$is the parameter set. A well-established way is to determine the consequent parameters using the least means squares algorithm.

- Layer-6 *(rule inference layer)* The single node in this layer computes the overall output as the summation of all incoming signals: $Overall\ \ output = \sum_i \overline{w_i} f_i = \frac{\sum_i w_i f_i}{\sum_i w_i}$.

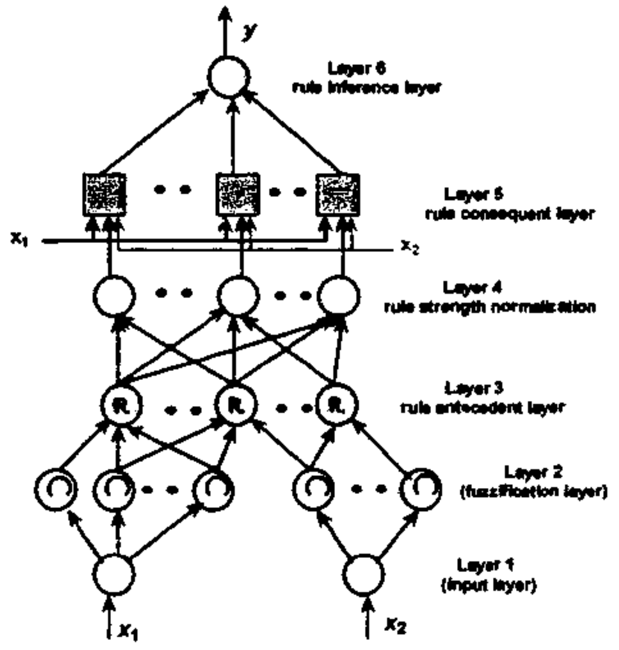


Figure 5.10. Takagi Sugeno neuro-fuzzy system

In the following sections we briefly discuss the different integrated neuro-fuzzy models that make use of the complementarities of neural networks and fuzzy inference systems implementing a Mamdani or Takagi Sugeno fuzzy inference system. Some of the major woks in this area are GARIC [39], FALCON [160], ANFIS [130], NEFCON [191], NEFCLASS [193], NEFPROX [196], FUN [226], SONFIN [83], FINEST [230], EFuNN [142], dmEFuNN[142], evolutionary design of neuro fuzzy systems [147], and many others [108] [137] [184] [256] [258].

5.3.3. Adaptive Network Based Fuzzy Inference System (ANFIS)

ANFIS [130] is perhaps the first integrated hybrid neuro-fuzzy model. ANFIS structure as shown in Figure 5.11 is capable of implementing the Takagi and Sugeno FIS. A modified version of ANFIS as shown in Figure 5.13 is capable of implementing the Tsukamoto fuzzy model (Figure 5.12). In the Tsukamoto FIS, the overall output is the weighted average of each rule's crisp output induced by the rule's firing strength (the product or minimum of the degrees of match with the premise part) and output membership functions. The output membership functions used in this scheme must be monotonically non-decreasing. ANFIS functions exactly as discussed in Section 5.3.2. The first hidden layer is for fuzzification of the input variables and T-norm operators are deployed in the second hidden layer to compute the rule antecedent part. The third hidden layer normalizes the rule strengths followed by the fourth hidden layer where the consequent parameters of the rule are determined. Output layer computes the overall input as the summation of all incoming signals.

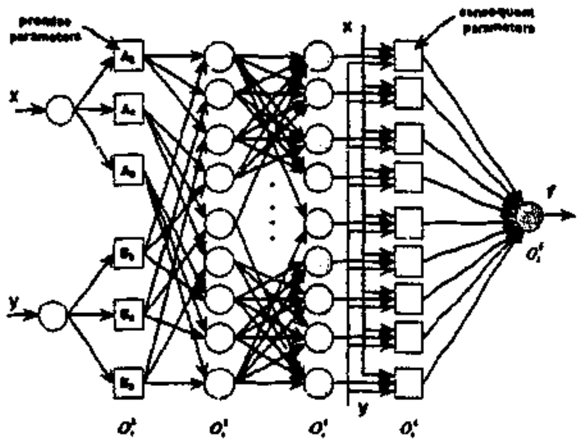
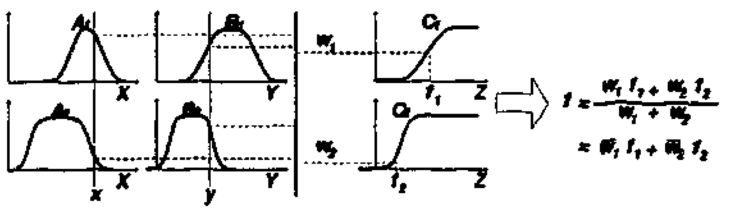


Figure.5.11. Architecture of ANFIS implementing a Takagi Sugeno fuzzy inference system



w_1
w_2
$f = \frac{w_1 f_1 + w_2 f_2}{w_1 + w_2} = \bar{w}_1 f_1 + \bar{w}_2 f_2$

Figure 5.12. Tsukamoto fuzzy reasoning

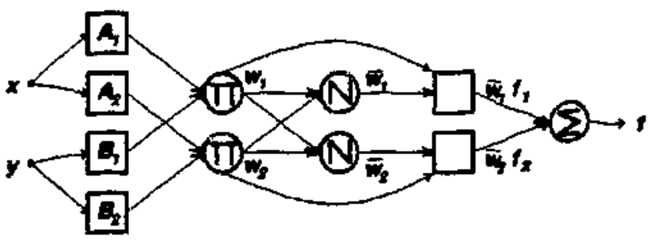


Figure 5.13. Architecture of ANFIS implementing Tsukamoto fuzzy inference system

In ANFIS the adaptation (learning) process is only concerned with parameter level adaptation within fixed structures. For large-scale problems, it will be too complicated to determine the optimal premise-consequent structures, rule numbers etc. The structure of ANFIS ensures that each linguistic term is represented by only one fuzzy set. However, the learning procedure of ANFIS does not provide the means to apply constraints that restrict the kind of modifications applied to the membership functions. When using Gaussian membership functions, operationally ANFIS can be compared with a radial basis function network.

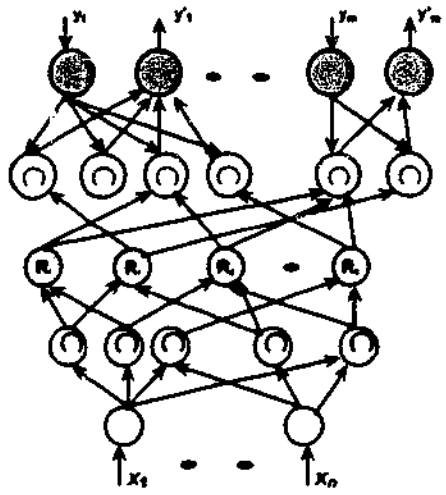


Figure 5.14. Architecture of FALCON

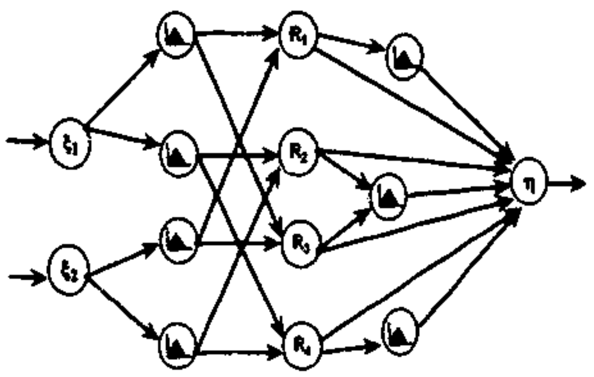


Figure 5.15. ASN of GARIC

5.3.4 Fuzzy Adaptive learning Control Network (FALCON)

FALCON [160] has a five-layered architecture as shown in Figure 5.14 and implements a Mamdani type FIS. There are two linguistic nodes for each output variable. One is for training data (desired output) and the other is for the actual output of FALCON. The first hidden layer is responsible for the fuzzification of each input variable. Each node can be a single node representing a simple membership function (MF) or composed of multilayer nodes that compute a complex MF. The Second hidden layer defines the preconditions of the rule followed by rule consequents in the third hidden layer. FALCON uses a hybrid-learning algorithm comprising of unsupervised learning and a gradient descent learning to optimally adjust the parameters to produce the desired outputs. The hybrid learning occurs in two different phases. In the initial phase, the centres and width of the membership functions are determined by self-organized learning techniques analogous to statistical clustering techniques. Once the initial parameters are determined, it is easy to formulate the rule antecedents. A competitive learning algorithm is used to determine the correct rule consequent links of each rule node. After the fuzzy rule base is established, the whole network structure is established. The network then enters the second learning phase to adjust the parameters of the (input and output) membership functions optimally. The backpropagation algorithm is used for the supervised learning. Hence FALCON algorithm provides a framework for structure and parameter adaptation for designing neuro-fuzzy systems [162].

5.3.5. Generalized Approximate Reasoning based Intelligent Control (GARIC)

GARIC [39] is an extended version of Berenji's Approximate Reasoning based Intelligent Control (ARIC) that implements a fuzzy controller by using several specialized feedforward neural networks[38]. Like ARIC, it consists of an Action state Evaluation Network (AEN) and an Action Selection Network (ASN). The AEN is an adaptive critic that evaluates the actions of the ASN. The ASN does not use any weighted connections, but the learning process modifies parameters stored within the units of the network. Architecture of the GARIC – ASN is depicted in Figure 5.15. ASN of GARIC is feedforward network with five layers. The first hidden layer stores the linguistic values of all the input variables. Each input unit is only connected to those units of the first hidden layer, which represent its associated linguistic values. The second hidden layer represents the fuzzy rules nodes, which determine the degree of fulfillment of a rule using a *softmin* operation. The third hidden layer represents the linguistic values of the control output variable η. Conclusions of the rule are computed depending on the strength of the rule antecedents computed by the rule node layer. GARIC makes use of local mean-of-maximum method for computing the rule outputs. This method needs a crisp output value from each rule. Therefore, the conclusions must be defuzzified before they are accumulated to the final output value of the controller. The learning algorithm of the AEN of GARIC is equivalent to that of its predecessor ARIC[37]. However, the ASN learning procedure is different from the procedure used in ARIC. GARIC uses a mixture of gradient descent and reinforcement learning to fine-tune the node parameters. The hybrid learning stops if the output of the AEN ceases to change. The interpretation of GARIC is improved compared to GARIC. The relatively complex learning procedure and the architecture of GARIC can be seen as a main disadvantage of GARIC.

5.3.6. Neuro-Fuzzy Controller (NEFCON)

The learning algorithm defined for NEFCON is able to learn fuzzy sets as well as fuzzy rules implementing a Mamdani type FIS [191]. This method can be considered as an extension to

GARIC that also use reinforcement learning but need a previously defined rule base. Figure 5.16(a) illustrates the basic NEFCON architecture with 2 inputs and five fuzzy rules [192]. The inner nodes $R_1, \ldots, R_5$ represent the rules, the nodes ξ_1, ξ_2, and η the input and output values, and μ_r, V_r the fuzzy sets describing the antecedents and consequents. In contrast to neural networks, the connections in NEFCON are weighted with fuzzy sets instead of real numbers. Rules with the same antecedent use so-called shared weights, which are represented by ellipses drawn around the connections as shown in the figure. They ensure the integrity of the rule base. The knowledge base of the fuzzy system is implicitly given by the network structure. The input units assume the task of fuzzification interface, the inference logic is represented by the propagation functions, and the output unit is the defuzzification interface. The learning process of the NEFCON model can be divided into two main phases. The first phase is designed to learn the rule base and the second phase optimizes the rules by shifting or modifying the fuzzy sets of the rules. Two methods are available for learning the rule base. *Incremental rule learning* is used when the correct out put is not known and rules are created based on estimated output values. As the learning progresses more rules are added according to the requirement. For *decremental rule learning,* initially rules are created due to fuzzy partitions of process variables and unnecessary rules are eliminated in the course of learning. Decremental rule learning is less efficient compared to incremental approach. However it can be applied to unknown processes without difficulty, and there is no need to know or to guess an optimal output value. Both phases use a fuzzy error E, which describes the quality of the current system state, to learn or to optimize the rule base. To obtain a good rule base it must be ensured that the state space of the process is sufficiently covered during the learning process. Due to the complexity of the calculations required, the decremental learning rule can only be used, if there are only a few input variables with not too many fuzzy sets. For larger systems, the incremental learning rule will be optimal. Prior knowledge whenever available could be incorporated to reduce the complexity of the learning [198]. Membership functions of the rule base are modified according to the Fuzzy Error Backpropagation (FEBP) algorithm. The FEBP algorithm can adapt the membership functions, and can be applied only if there is already a rule base of fuzzy rules. The idea of the learning algorithm is identical: increase the influence of a rule if its action goes in the right direction (rewarding), and decrease its influence if a rule behaves counter productively (punishing). If there is absolutely no knowledge about initial membership function, a uniform fuzzy partition of the variables should be used.

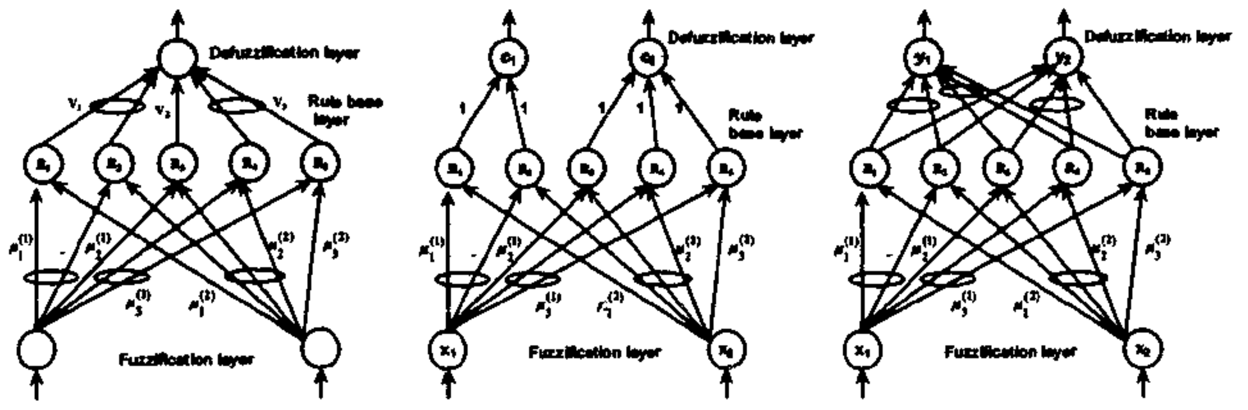


Figure 5.16. (a) NEFCON **(b)** NEFCLASS **(C)** NEFPROX.

5.3.7 Neuro-Fuzzy Classification (NEFCLASS)

NEFCLASS is used to derive fuzzy rules from a set of data that can be separated in different crisp classes [193]. The rule base of a NEFCLASS system approximates an (unknown) function φ that represents the classification problem and maps an input pattern x to its class C_i:

$$\varphi : R^n \rightarrow \{0,1\}^m,\ \varphi(x) = (c_1,\ldots,c_m),\ with\ c_i = \begin{cases} 1 & if\ x \in C_i \\ 0 & otherwise. \end{cases}$$

Because of the propagation procedures used in NEFCLASS the rule base actually does not approximate φ but a function $\varphi^{\circ} : R^n \rightarrow \{0,1\}^m$. We obtain $\varphi\ (x)$ from the equality $\varphi\ (x) = \varphi\ (\varphi\ (x))$, where φ reflects the interpretation of the classification result obtained from a NEFCLASS system [194]. Figure 5.16 (b) illustrates the NEFCLASS system that maps patterns with two features into two distinct classes by using five linguistic rules. The NEFCLASS very much resemble the NEFCON system except the slight variation in the learning algorithm and the interpretation of the rules. As in NEFCON system in NEFCLASS identical linguistic values of an input variable are represented by the same fuzzy set. As classification is the primary task of NEFCLASS, there should be two rules with identical antecedents and each rule unit must be connected to only one output unit. The weights between rule layer and the output layer only connect the units. A NEFCLASS system can be built from partial knowledge about the patterns, and can then be refined by learning, or it can be created from scratch by learning. A user must define a number of initial fuzzy sets that partition the domains of the input features, and specify a value for *k*, i.e. the maximum number of rule nodes that may be created in the hidden layer. NEFCLASS makes use of triangular membership functions and the learning algorithm of the membership functions uses an error measure that tells whether the degree of fulfillment of a rule has to be higher or lower. This information is used to change the input fuzzy sets. Being a classification system, we are not much interested in the exact output values. In addition, we take a winner-takes-all interpretation for the output, and we are mainly interested in the correct classification result. The incremental rule learning in NEFCLASS is much less expensive than decremental rule learning in NEFCON. It is possible to build up a rule base in a single sweep through the training set. Even for higher dimensional problems, the rule base is completed after at most three cycles. Compared to neural networks, NEFCLASS uses a much simpler learning strategy. There is no vector quantization involved in finding the rules (clusters, and there is no gradient information needed to train the membership functions. Some other advantages are interpretability, possibility of initialization (incorporating prior knowledge) and its simplicity.

5.3.8. Neuro-Fuzzy Function Approximation (NEFPROX)

NEFPROX system is based on plain supervised learning (fixed learning problem) and it is used for function approximation [195]. It is a modified version of the NEFCON model without the reinforcement learning. The advantage of neuro-fuzzy models is that we can incorporate prior knowledge; where as conventional neural networks have to learn from scratch. NEFPROX is very much similar to NEFCON and NEFCLASS except the fact that NEFCON have only a single output node, and NEFCLASS systems do not use membership functions on the conclusion side [196]. We can initialize the NEFPROX system if we already know suitable rules or else the system is capable to incrementally learn all rules. NEFPROX architecture is as shown in Figure 5.16(c). While ANFIS is capable to implement only Sugeno models with differentiable functions, NEFPROX can learn common Mamdani type of fuzzy system from data. Further NEFPROX is much faster compared to ANFIS to yield results. However ANFIS yields better approximation results.

5.3.9. Fuzzy Inference Environment Software with Tuning (FINEST)

FINEST is designed to tune the fuzzy inference itself. FINEST is capable of two kinds of tuning process, the tuning of fuzzy predicates, combination functions and the tuning of an implication function [230]. The three important features of the system are:

- The generalized modus ponens is improved in the following four ways (1) Aggregation operators that have synergy and cancellation nature (2) A parameterized implication function (3) A combination function, which can reduce fuzziness (4) Backward chaining based on generalized modus ponens.

- Aggregation operators with synergy and cancellation nature are defined using some parameters, indicating the strength of the synergic affect, the area influenced by the effect, etc., and the tuning mechanism is designed to tune also these parameters. In the same way the tuning mechanism can also tune the implication function and combination function.

- The software environment and the algorithms are designed for carrying out forward and backward chaining based on the improved generalized modus ponens and for tuning various parameters of a system.

FINEST make use of a backpropagation algorithm for the fine-tuning of the parameters. Figure 16 shows the layered architecture of FINEST and the calculation process of the fuzzy inference. The input values (x_i) are the facts and the output value (y) is the conclusion of the fuzzy inference. Layer 1 is a fuzzification layer and layer 2 aggregates the truth-values of the conditions of Rule *i*. Layer 3 deduces the conclusion from Rule *I* and the combination of all the rules is done in Layer 4. Referring to Figure 5.17, the function and_i, I_i and *comb* respectively represent the function characterizing the aggregation operator of rule *i*, the implication function of rule *i*, and the global combination function. The functions and_i, I_i, *comb* and membership functions of each fuzzy predicate are defined with some parameters.

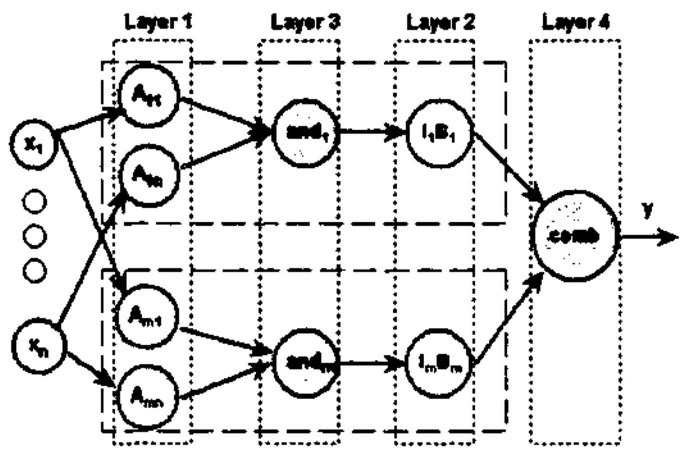


Figure 5.17. Architecture of FINEST

Back-propagation method is used to tune the network parameters. It is possible to tune any parameter, which appears in the nodes of the network representing the calculation process of the fuzzy data if the derivative function with respect to the parameters is given.

Thus FINEST framework provides a mechanism based on the improved generalized modus ponens for fine tuning of fuzzy predicates and combination functions and tuning of the implication function. Parameterization of the inference procedure is very much essential for proper application of the tuning algorithm.

5.3.10. Self Constructing Neural Fuzzy Inference Network (SONFIN)

SONFIN implements a Takagi-Sugeno type fuzzy inference system. Fuzzy rules are created and adapted as online learning proceeds via a simultaneous structure and parameter identification [83]. In the structure identification of the precondition part, the input space is partitioned in a flexible way according to an aligned clustering based algorithm. As to the structure identification of the consequent part, only a singleton value selected by a clustering method is assigned to each rule initially. Afterwards, some additional significant terms (input variables) selected via a projection-based correlation measure for each rule will be added to the consequent part (forming a linear equation of input variables) incrementally as learning proceeds. For parameter identification, the consequent parameters are tuned optimally by either Least Mean Squares [LMS] or Recursive Least Squares [RLS] algorithms and the precondition parameters are tuned by back propagation algorithm. To enhance knowledge representation ability of SONFIN, a linear transformation for each input variable can be incorporated into the network so that much fewer rules are needed or higher accuracy can be achieved. Proper linear transformations are also learned dynamically in the parameter identification phase of SONFIN. Figure 5.18 illustrates the 6-layer structure of SONFIN.

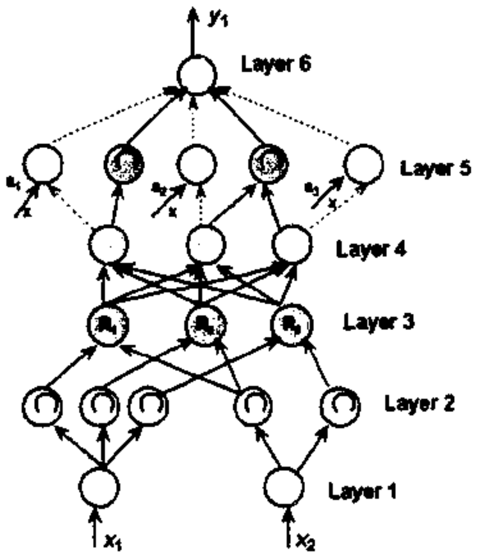


Figure 5.18. Six layered architecture of SONFIN

Learning progresses concurrently in two stages for the construction of SONFIN. The *structure* learning includes both the precondition and consequent structure identification of a fuzzy *if-then* rule. The parameter learning is based on supervised learning algorithms, the parameters of the linear equations in the consequent parts are adjusted by either LMS or RLS algorithms and the parameters in the precondition part are adjusted by the backpropagation algorithm. SONFIN can be used for normal operation at anytime during the learning process without repeated training on the input-output pattern when online operation is required. In SONFIN rule base is dynamically created as the learning progresses by performing the following learning processes:

- **Input-output space partitioning**

The way the input space is partitioned determines the number of rules extracted from the training data as well as the number of fuzzy sets on the universal of discourse of each input variable. For each incoming pattern x the strength a rule is fired can be interpreted as the degree the incoming pattern belongs to the corresponding cluster. The center and width of the corresponding membership functions (of the newly formed fuzzy rules) are assigned according to the first-neighbor heuristic. For each rule generated, the next step is to decompose the multidimensional membership function to corresponding *1-D* membership function for each input variable. For the output space partitioning, almost a similar measure is adopted. Performance of SONFIN can be enhanced by incorporating a transformation matrix *R* into the structure, which accommodates all the *a priori* knowledge of the data set.

- **Construction of fuzzy rule base**

Generation of new input cluster corresponds to the generation of a new fuzzy rule, with its precondition part constructed by the learning algorithm in process. At the same time we have to decide the consequent part of the generated rule. This is done using a algorithm based on the fact that different preconditions of rules may be mapped to the same consequent fuzzy set. Since only the center of each output membership function is used for defuzzification, the consequent part of each rule may simply be regarded as a singleton. Compared to the general fuzzy rule based models with singleton output where each rule has its own singleton value, fewer parameters are needed in the consequent part of the SONFIN, especially for complicated systems with a large number of rules.

- **Optimal consequent structure identification**

TSK model can model a sophisticated system with a few rules. In SONFIN, instead of using the linear combination of all input variables as the consequent part, only the most significant input variables are used as the consequent terms of the SONFIN. The significant terms will be chosen and added to the network incrementally any time when the parameter learning cannot improve the network output accuracy anymore during the online learning process. The consequent structure identification scheme in SONFIN is a kind of node growing method in ANNs. When the effect of the parameter learning diminished (output error is not decreasing), additional terms are added to the consequent part.

- **Parameter identification.**

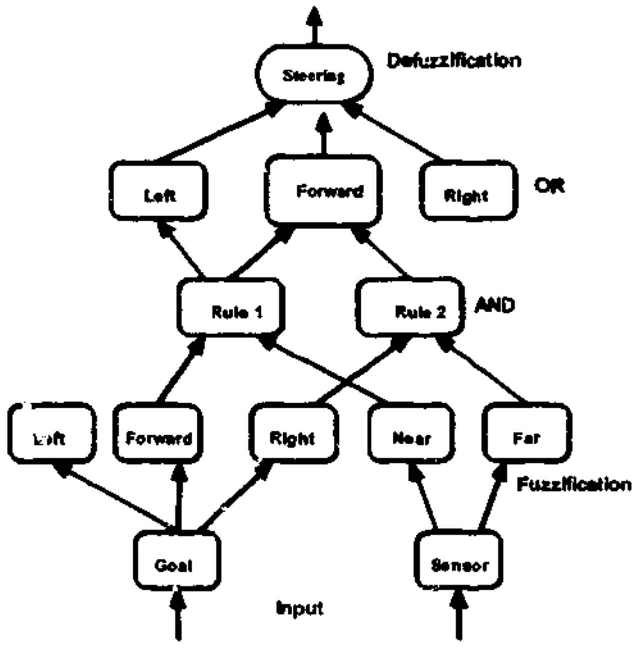
After the network structure is adjusted according to the current training pattern, the network then enters the parameter identification phase to adjust the parameters of the network optimally based on the same training pattern. Parameter learning is performed on the whole network after structure learning, no matter whether the nodes (links) are newly added or are existent originally. Backpropagation algorithm is used for this supervised learning. For each training data set, starting at the input nodes, a forward pass is used to compute the activity levels of all the nodes in the network to obtain the current output. Then starting at the output nodes, a backward pass is used to compute $\frac{\partial E}{\partial w}$ for all the hidden nodes of all the layers. If w is the adjustable parameter in a node, the general rule used is:

$W(t + 1) = w(t) + \eta\left(-\frac{\partial E}{\partial w}\right)$, where η is the learning rate.

SONFIN is perhaps one of the most computational expensive among all neuro-fuzzy models. The network is adaptable to the users specification of required accuracy.

5.3.11. FUzzy Net [FUN]

In FUN inorder to enable an unequivocal translation of fuzzy rules and membership functions into the network, special neurons have been defined, which, through their activation functions, can evaluate logic expressions [226]. The network consists of an input, an output and three hidden layers. The neurons of each layer have different activation functions representing the different stages in the calculation of fuzzy inference. The activation function can be individually chosen for problems. The network is initialized with a fuzzy rule base and the corresponding membership functions. Figure 5.19 illustrates the FUN network. The input variables are stored in the input neurons. The neurons in the first hidden layer contain the membership functions and this performs a fuzzification of the input values. In the second hidden layer, the conjunctions (fuzzy-AND) are calculated. Membership functions of the output variables are stored in the third hidden layer. Their activation function is a fuzzy-OR. Finally the output neurons contain the output variables and have a defuzzification activation function.



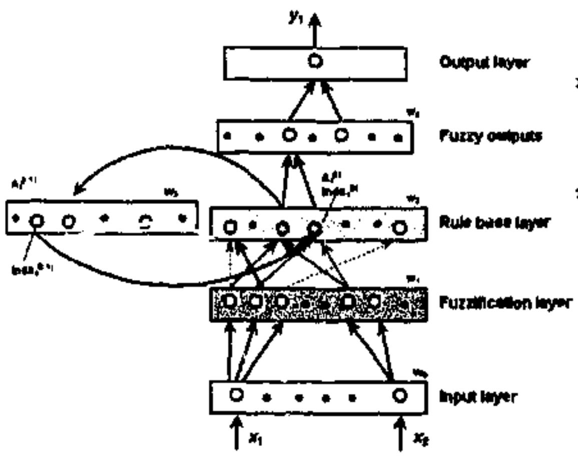
***Rule:** IF (Goal IS forward AND Sensor IS near) OR (goal IS right AND sensor IS far) THEN steering = forward*

Figure 5.19. Architecture of the FUN showing the implementation of a sample rule

The rules and the membership functions are used to construct an initial FUN network. The rule base can then be optimized by changing the structure of the net or the data in the neurons. To learn the rules, the connections between the rules and the fuzzy values are changed. To learn the membership functions, the data of the nodes in the first and three hidden layers are changed. FUN can be trained with the standard neural network training strategies such as reinforcement or supervised learning.

- **Learning of the rules and membership functions**

The rules are represented in the net through the connections between the layers. The learning of the rules is implemented as a stochastic search in the rule space: a randomly chosen connection is changed and the new network performance is verified with a cost function. If the performance is worse, the change is undone, otherwise it is kept and some other changes are tested, until the desired output is achieved. As the learning algorithm should preserve the semantic of the rules, it has to be controlled in such a way that no two values of the same variable appear in the same rule. This is achieved by swapping connections between the values of the same variable. FUN uses a mixture of gradient descent and stochastic search for updating the membership functions. A maximum change in a random direction is initially assigned to all Membership function Descriptors (MFDs). In a random fashion one MFD of one linguistic variable is selected, and the network performance is tested with this MFD altered according to the allowable change for this MFD. If the network performs better according to the given cost function, the new value is accepted and next time another change is tried in the same direction. Contrary if the network performs worse, the change is reversed. To guarantee convergence, the changes are reduced after each training step and shrink asymptotically towards zero according to the learning rate. As evident, FUN system is initialized by specifying a fixed number of rules and a fixed number of initial fuzzy sets for each variable and the network learns through a stochastic procedure that randomly changes parameters of membership functions and connections within the network structure Since no formal neural network learning technique is used it is questionable to call FUN a neuro-fuzzy system.



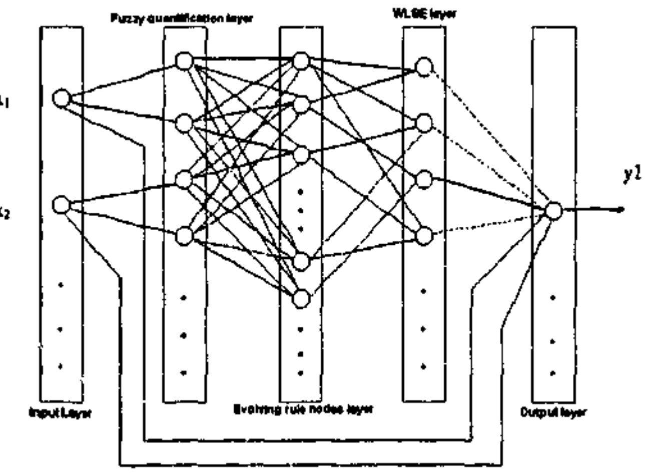


Figure 5. 20(a) Architecture of EFuNN **(b)** Architecture of dmEFuNN

5.3.12. Evolving Fuzzy Neural Networks (EFuNNs and mEFuNNs)

EFuNNs [139] and dmEFuNNs [142] are based on the ECOS (Evolving COnnectionist Systems) framework [141] for adaptive intelligent systems formed because of evolution and incremental, hybrid (supervised / unsupervised), online learning. They can accommodate new input data, including new features, new classes, and etc. through local element tuning [140].

In EFuNNs all nodes are created during learning. EFuNN has a five-layer architecture as shown in Figure 5.20(a). The input layer is a buffer layer representing the input variables. The second layer of nodes represents fuzzy quantification of each input variable space. Each input variable is represented here by a group of spatially arranged neurons to represent a fuzzy quantization of this variable. The nodes representing membership functions (triangular, Gaussian, etc) can be modified during learning. The third layer contains rule nodes that evolve through hybrid supervised/unsupervised learning. The rule nodes represent prototypes of input-output data associations, graphically represented as an association of hyper-spheres from the fuzzy input and fuzzy output spaces. Each rule node r is defined by two vectors of connection weights: $W_1(r)$ and $W_2(r)$, the latter being adjusted through supervised learning based on the output error, and the former being adjusted through unsupervised learning based on similarity measure within a local area of the input problem space [144]. The fourth layer of neurons represents fuzzy quantification for the output variables. The fifth layer represents the real values for the output variables. In the case of "one-of-n" EFuNNs, the maximum activation of the rule node is propagated to the next level. In the case of "*many-of-n*" mode, all the activation values of rule nodes that are above an activation threshold are propagated further in the connectionist structure.

5.3.12.1 Dynamic Evolving Fuzzy Neural Networks (dmEFuNNs)

Dynamic Evolving Fuzzy Neural Networks (dmEFuNN) model is developed with the idea that not just the winning rule node's activation is propagated but a group of rule nodes is dynamically selected for every new input vector and their activation values are used to calculate the dynamical parameters of the output function. While EFuNN make use of the weighted fuzzy rules of Mamdani type, dmEFuNN uses the Takagi-Sugeno fuzzy rules.

The first, second and third layers of dmEFuNN have exactly the same structures and functions as the EFuNN. The fourth layer, the fuzzy inference layer, selects m rule nodes from the third layer which have the closest fuzzy normalised local distance to the fuzzy input vector, and then, a Takagi-Sugeno fuzzy rule will be formed using the weighted least square estimator.

The last layer calculates the output of dmEFuNN. Please refer to Figure 20(b) for details about the dmEFuNN architecture.

The number m of activated nodes used to calculate the output values for a dmEFuNN is not less than the number of the input nodes plus one. Like the EFuNNs, the dmEFuNNs can be used for both off-line learning and online learning thus optimising global generalization error, or a local generalization error. In dmEFuNNs, for a new input vector (for which the output vector is not known), a subspace consisted of *m* rule nodes are found and a first order Takagi-Sugeno fuzzy rule is formed using the least square estimator method. This rule is used to calculate the dmEFuNN output value. In this way a dmEFuNN acts as a universal function approximator using *m* linear functions in a small *m*-dimensional node subspace. The accuracy of approximation depends on the size of the node subspaces, the smaller the subspace is, the higher the accuracy. It means that if there are sufficient training data vectors and sufficient rule nodes are created, a satisfying accuracy can be obtained.

5.4.Discussions Related to Neuro-Fuzzy Models

As evident, both cooperative and concurrent models are not fully interpretable due to the presence of neural network (black box concept). Whereas an integrated neuro-fuzzy model is interpretable and capable of learning in a supervised mode (or even reinforcement learning like NEFCON). In FALCON, GARIC, ANFIS, NEFCON, SONFIN, FINEST and FUN the learning process is only concerned with parameter level adaptation within fixed structures. For large-scale problems, it will be too complicated to determine the optimal premise-consequent structures, rule numbers etc. User has to provide the architecture details (type and quantity of MF's for input and output variables), type of fuzzy operators etc. FINEST provides a mechanism based on the improved generalized modus ponens for fine tuning of fuzzy predicates and combination functions and tuning of an implication function. An important feature of EFuNN and dmEFuNN is the one pass (epoch) training, which is highly capable of online learning. Table 5.4 provides a comparative performance of some neuro fuzzy systems for predicting the Mackey-Glass chaotic time series [167]. Training was done using 500 data sets and NF models were tested with another 500 data sets [9].

Table 5.4. Performance of neuro-fuzzy systems

System	Epochs	Test RMSE
ANFIS	75	0.0017
NEFPROX	216	0.0332
EFuNN	1	0.0140
dmEFuNN	1	0.0042
SONFIN	1	0.0180

Among NF models ANFIS has the lowest Root Mean Square Error (RMSE) and NEPROX the highest. This is probably due to Takagi-Sugeno rules implementation in ANFIS compared to the Mamdani-type fuzzy system in NEFPROX. However NEFPROX outperformed ANFIS in terms of computational time. Due to fewer numbers of rules SONFIN, EFuNN and dmEFuNN are also able to perform faster than ANFIS. Hence, there is a tradeoff between interpretability and accuracy. Takagi Sugeno type inference systems are more accurate but require more computational effort. While Mamdani type inference, systems are more interpretable and required less computational load but often the accuracy is not that high.

As the problem become, more complicated manual definition of NF architecture/parameters becomes complicated. The following questions remain unanswered:

- What is the optimal quantity of membership functions and their shape?
- What is the optimal structure (rule base) and fuzzy operators?
- What are the optimal learning parameters?
- Which fuzzy inference system will work the best for a given problem?

We will try to address the above questions in Chapter 7. In that Chapter, we will show how the integrated neuro-fuzzy systems are implemented in practice.

Chapter 6: Integrated Neuro-Fuzzy Systems in Practice

6.0 Introduction

Neuro-fuzzy computing is a popular framework for solving complex problems. When knowledge is expressed in linguistic rules, we can build a fuzzy inference system, and if we have data, or can learn from a simulation (training) then we can use neural networks. For building a fuzzy inference system, we have to specify the fuzzy sets, fuzzy operators and the rule base. Similarly, for constructing a neural network for an application, the user needs to specify the architecture, learning algorithm and several parameters. An integrated neuro-fuzzy system is a combination of neural network and fuzzy inference system in that neural network learning algorithms are used to determine the parameters of the fuzzy inference system. An even more important aspect is that the system should always be interpretable in terms of fuzzy *if-then* rules, because it is based on the fuzzy system reflecting vague knowledge.

In this chapter we present 3 real life applications of integrated neuro-fuzzy systems. We begin with an introduction of learning in neuro-fuzzy systems emphasizing Takagi Sugeno and Mamdani fuzzy inference systems. The three applications are (1) modeling electricity demand in Victoria (2) automation of reactive power control and (3) developing rainfall prediction models.

6.1. Learning in Adaptive Neuro-Fuzzy Inference Systems

The basic architecture and functioning of the different layers of Adaptive Neuro-Fuzzy Inference System (ANFIS) was presented in Chapter 5, Section 5.3.3. In this section we will present the learning mechanism in ANFIS to learn the fuzzy inference system automatically. ANFIS uses a hybrid learning rule with a combination of gradient descent and least squares estimate [130]. Assuming a single output ANFIS represented by

$$output = F(\vec{I}, S) \qquad (6.1)$$

where I is the set of input variables and S is the set of parameters, if there exist a function H such that the composite function $H \circ F$ is linear in some of the elements of S, then these elements can be identified by the least squares method [132]. More formally, the parameter set S can be decomposed into two sets:

$$S = S_1 \oplus S_2 \text{ (where } \oplus \text{ represents direct sum),} \qquad (6.2)$$

such that $H \circ F$ is linear in the elements of S_2. Then upon applying H to equation (6.1), we have:

$$H(output) = H \circ F(\vec{I}, S) \qquad (6.3)$$

which is linear in the elements of S_2. Now the given values of elements of S_1, we can plug P training data sets into (6.3), and obtain a matrix equation:

$$AX = B \text{ (} X = \text{unknown vector whose elements are parameters in } S_2 \text{)} \qquad (6.4)$$

If $|S_2|=M$, (M= number of linear parameters) then the dimensions of A, X and B are $P \times M$, $M \times 1$ and $P \times 1$ respectively. Since P is always greater than M, there is no exact solution to equation *(6.4)*. Instead a Least Square Estimate (LSE) of X, X^*, is sought to minimize the squared error $\|AX - B\|^2$. X^* is computed using the pseudo-inverse of X:

$$X^* = (A^T A)^{-1} A^T B \qquad (6.5)$$

where A^T is the transpose of A and $(A^T A)^{-1} A^T$ is the pseudo-inverse of A where $A^T A$ is non-singular. Due to computational complexity, in ANFIS a sequential method is deployed as follows:

Let the *i-th* row vector of matrix A defined in equation 6.4 be a_i^T and *i-th* element of matrix B defined be b_i^T, then X can be calculated iteratively using the following sequential formulae:

$$\begin{aligned} X_{i+1} &= X_i + S_{i+1} a_{i+1} (b_{i+1}^T - a_{i+1}^T X_i) \\ S_{i+1} &= S_i - \frac{S_i a_{i+1} a_{i+1}^T S_i}{1 + a_{i+1}^T S_i a_{i+1}}, \quad i = 0,1,\ldots\ldots, P-1 \end{aligned} \qquad (6.6)$$

where S_i is often called the covariance matrix and the least squares estimate X^* is equal to X_P. The initial condition to bootstrap (6.6) are $X_O=0$ and $S_O=\gamma I$, where γ is a positive large number and I is the identity matrix of dimension $M \times M$. For a multi output ANFIS, (6.6) is still applicable except the $output = F(\vec{I}, S)$ will become a column vector. Each epoch of this hybrid learning procedure is composed of a forward pass and a backward pass. In the forward pass, we have to supply the input data and functional signals go forward to calculate each node output until the matrices A and B in (6.4) are obtained, and the parameters in S_2 are identified by the sequential least squares formulae given in (6.6). After identifying parameters in S_2, the functional signals keep going forward till the error measure is calculated. In the backward pass, the error rates propagate from the output layer to the input layers, and the parameters in S_1 are updated by the gradient method given by

$$\Delta\alpha = -\eta \frac{\partial E}{\partial \alpha} \tag{6.7}$$

where α is the generic parameter, η is a learning rate and E the error measure. For given fixed values of parameters in S_1, the parameters in S_2 thus found are guaranteed to be the global optimum point in the S_2 parameter space due to the choice of the squared error measure [131] [132].

The procedure mentioned above is mainly for offline learning version. However, the procedure can be modified for an online version by formulating the squared error measure as a weighted version that gives higher weighting factors to more recent data pairs. This amounts to the addition of a forgetting factor λ to (6.6).

$$\begin{aligned} X_{i+1} &= X_i + S_{i+1}a_{i+1}(b_{i+1}^T - a_{i+1}^T X_i) \\ S_{i+1} &= \frac{1}{\lambda}\left[S_i - \frac{S_i a_{i+1} a_{i+1}^T S_i}{\lambda + a_{i+1}^T S_i a_{i+1}}\right] \quad i = 0,1,\ldots\ldots, P-1 \end{aligned} \tag{6.8}$$

The value of λ is between 0 and 1. The smaller the λ is, faster the effects of old data decay. However, a smaller λ sometimes causes numerical instability and should be avoided.

6.2. Learning in Evolving Fuzzy Neural Network

We have presented the architecture of the Evolving Fuzzy Neural Network (EFuNN) in Chapter 5, Section 5.3.12. In this section, we will discuss the online learning process in EFuNN. A short description of the different layers in EFuNN is shown in Figure 5.20(a). The third layer contains rule nodes that evolve through hybrid supervised/unsupervised learning. The rule nodes represent prototypes of input-output data associations, graphically represented as an association of hyper-spheres from the fuzzy input and fuzzy output spaces. Each rule node, e.g. r_j, represents an association between a hyper-sphere from the fuzzy input space and a hyper-sphere from the fuzzy output space; $W_1(r_j)$ connection weights representing the co-ordinates of the center of the sphere in the fuzzy input space, and $W_2(r_j)$ – the co-ordinates in the fuzzy output space. The radius of an input hyper-sphere of a rule node is defined as (1-*Sthr*), where *Sthr* is the sensitivity threshold parameter defining the minimum activation of a rule node (e.g., r_1, previously evolved to represent a data point (X_{d1},Y_{d1})) to an input vector (e.g., (X_{d2},Y_{d2})) in order for the new input vector to be associated with this rule node. Two pairs of fuzzy input-output data vectors $d_1=(X_{d1},Y_{d1})$ and $d_2=(X_{d2},Y_{d2})$ will be allocated to the first rule node r_1 if they fall into the r_1 input sphere and in the r_1 output sphere, i.e. the local normalised fuzzy difference between X_{d1} and X_{d2} is smaller than the radius r and the local normalised fuzzy difference between Y_{d1} and Y_{d2} is smaller than an error threshold *Errthr*. The local normalised fuzzy difference between two fuzzy membership vectors d_{1f} and d_{2f} that represent the membership degrees to which two real values d_1 and d_2 data belong to the pre-defined MF, are calculated as $D(d_{1f},d_{2f}) = sum(abs(d_{1f} - d_{2f}))/sum(d_{1f} + d_{2f})$.

If data example $d_1 = (X_{d1},Y_{d1})$, where X_{d1} and X_{d2} are correspondingly the input and the output fuzzy membership degree vectors, and the data example is associated with a rule node r_1 with a centre r_1^1, then a new data point $d_2=(X_{d2},Y_{d2})$, will also be associated with this rule node through the process of associating (learning) new data points to a rule node. The centres of this node hyper-spheres adjust in the fuzzy input space depending on a learning rate lr_1, and in the fuzzy output space depending on a learning rate lr_2, on the two data point's d_1 and d_2. The adjustment of the centre r_1^1 to its new position r_1^2 can be represented mathematically by the change in the connection weights of the rule node r_1 from $W_1(r_1^1)$ and $W_2(r_1^1)$ to $W_1(r_1^2)$ and $W_2(r_1^2)$ according to the following vector operations:

$$W_2(r_1^2) = W_2(r_1^1) + lr_2 \,.\, Err(Y_{d1}, Y_{d2}) \,.\, A_1(r_1^1) \qquad (6.9)$$

$$W_1(r_1^2) = W1\,(r_1^1) + lr_1 \,.\, Ds\,(X_{d1}, X_{d2}) \qquad (6.10)$$

where $Err(Y_{d1},Y_{d2}) = Ds(Y_{d1},Y_{d2}) = Y_{d1}\text{-}Y_{d2}$ is the signed value rather than the absolute value of the fuzzy difference vector; $A_1(r_1^1)$ is the activation of the rule node r_1^1 for the input vector X_{d2}.

While the connection weights from W_1 and W_2 capture spatial characteristics of the learned data (centres of hyper-spheres), the temporal layer of connection weights W_3 captures temporal dependencies between consecutive data examples. If the winning rule node at the moment (*t-1*) (to which the input data vector at the moment (*t-1*) was associated) was $r_1 = inda_1(t\text{-}1)$, and the winning node at the moment t is $r_2 = inda_1(t)$, then a link between the two nodes is established as follows:

$$W_3(r_1,r_2)^{(t)} = W_3(r_1,r_2)^{(t-1)} + lr_3 .\, A_1(r_1)^{(t-1)} A_1(r_2))^{(t)}, \qquad (6.11)$$

where: $A_1(r)^{(t)}$ denotes the activation of a rule node r at a time moment (t); lr_3 defines the degree to which the EFuNN associates links between rules (clusters, prototypes) that include consecutive data examples (if lr_3=0, no temporal associations are learned in an EFuNN structure).

The learned temporal associations can be used to support the activation of rule nodes based on temporal, pattern similarity. Here, temporal dependencies are learned through establishing structural links. The ratio spatial-similarity/temporal-correlation can be balanced for different applications through two parameters S_s and T_c such that the activation of a rule node r for a new data example d_{new} is defined as the following vector operations:

$$A_1(r) = f\,(\,S_s \,.\, D(r, d_{new}) + T_c \,.\, W_3(r^{(t-1)}, r)) \qquad (6.12)$$

where f is the activation function of the rule node r, $D(r, d_{new})$ is the normalised fuzzy distance value and $r^{(t-1)}$ is the winning neuron at the previous time moment. The fourth layer of neurons represents fuzzy quantification for the output variables. The fifth layer represents the real values for the output variables.

EFuNN evolving algorithm is given as a procedure of consecutive ste ›s [139].

1. Initialize an EFuNN structure with a maximum number of neurons and zero value connections. If initially there are no rule nodes connected to the fuzzy input and fuzzy output neurons, then create the first node r_j=1 to represent the first data example EX= (X_{d1}, Y_{d1}) and set its input $W_1 (r_j)$ and output $W_2 (r_j)$ connection weights as follows: *<Create a new rule node r_j>* to represent a data sample EX: $W_1 (r_j)=EX$: $W_2 (r_j)= TE$, where TE is the fuzzy output vector for the (fuzzy) example EX.

2. While <there are data examples> Do
   Enter the current, example (X_{di}, Y_{di}), EX being the fuzzy input vector (the vector of the degrees to which the input values belong to the input membership functions). If there are new variables that appear in this example and have not been used in previous examples, create new input and/or output nodes with their corresponding membership functions.
3. Find the normalized fuzzy similarity between the new example EX (fuzzy input vector) and the already stored patterns in the case nodes $r_j= r_1, r_2,\ldots,r_n$
   $D(EX,r_j) = sum\ (abs\ (EX - W_1(r_j)))\ /\ sum\ (W_1(r_j) + EX)$
4. Find the activation $A_1 (r_j)$ of the rule nodes $r_j= r_1, r_2,\ldots,r_n$. Here radial basis activation (*radbas*) function, or a saturated linear (*satlin*) one, can be used, i.e.
   $A_1 (r_j) = radbas\ (S_s\, D(EX, r_j - T_c\, W_3)$, or $A_1 (r_j) = satlin\ (1- S_s\, D(EX, r_j + T_c\, W_3))$.
5. Update the pruning parameter values for the rule nodes, e.g. age, average activation as pre-defined.
6. Find m case nodes r_j with an activation value $A_1 (r_j)$ above a predefined sensitivity threshold $Sthr$.
7. From the m case nodes, find one rule node $inda_1$ that has the maximum activation value $maxa_1$.
8. If $maxa_1 < Sthr$, then, <*create a new rule node*> using the procedure from step 1.

Else

9. Propagate the activation of the chosen set of *m* rule nodes $(r_{j1},\dots,r_{jm})$ to the fuzzy output neurons: $A_2 = satlin\ (A_1(r_{j1},\dots,r_{jm})\ .\ W_2)$
10. Calculate the fuzzy output error vector

    $Err = A_2 - TE$
11. If $(D(A_2,TE) > Errthr)$ <*create a new rule node*> using the procedure from step 1.
12. Update (a) the input, and (b) the output of the m-1 rule nodes $k = 2 : j_m$ in case of a new node was created, or *m* rule nodes $k = j_1 : j_m$, in case of no new rule was created:

    $Ds(EX\text{-}W_1(r_k)) = EX - W_1(r_k);\ W_1(r_k) = W_1(r_k) + lr_1\ Ds(EX\text{-}W_1(r_k))$, where lr_1 is the learning rate for the first layer;

    $A_2(r_k) = satlin\ (W_2(r_k).A_1(r_k));\ Err(rk) = TE\text{-}A_2(r_k);$

    $W_2(r_k) = W_2(r_k) + lr_2\ .\ Err\ (r_k)\ .A_1(r_k)$, where lr_2 is the learning rate for the second layer.
13. Prune rule nodes r_j and their connections that satisfy the following fuzzy pruning rule to a pre-defined level representing the current need of pruning:

    IF (a rule node r_j is OLD) and (average activation $A_1av(r_j)$ is LOW) and (the density of the neighboring area of neurons is HIGH or MODERATE) (i.e. there are other prototypical nodes that overlap with j in the input-output space; this condition apply only for some strategies of inserting rule nodes as explained below) THEN the probability of pruning node (r_j) is HIGH. The above pruning rule is fuzzy and it requires that the fuzzy concepts as OLD, HIGH, etc. are predefined.
14. Aggregate rule nodes, if necessary, into a smaller number of nodes. A C-means clustering algorithm can be used for this purpose.
15. End of the *while* loop and the algorithm

The rules that represent the rule nodes need to be aggregated in clusters of rules. The degree of aggregation can vary depending on the level of granularity needed. At any time (phase) of the evolving (learning) process, fuzzy, or exact rules can be inserted and extracted [145]. Insertion of fuzzy rules is achieved through setting a new rule node for each new rule, such as the connection weights W_1 and W_2 of the rule node represent the fuzzy or the exact rule. The process of rule extraction can be performed as aggregation of several rule nodes into larger hyper-spheres. For the aggregation of two-rule nodes r_1 and r_2, the following aggregation rule is used

$$If\ (D(W_1(r_1), W_1(r_2)) <= Thr_1)\ and\ (D(W_2(r_1), W_2(r_2)) <= Thr_2) \quad (6.13)$$

then aggregate r_1 and r_2 into r_{agg} and calculate the centres of the new rule node as

$$W_1(r_{agg}) = average\ (W_1(r_1), W_1(r_2)),\ W_2(r_{agg}) = average\ (W_2(r_1), W_2(r_2)) \quad (6.14)$$

Here the geometrical center between two points in a fuzzy problem space is calculated with the use of an average vector operation over the two fuzzy vectors. This is based on a presumed piece-wise linear function between two points from the defined through the parameters *Sthr* and *Errthr* input and output fuzzy hyper-spheres.

6.3.Neuro-Fuzzy Applications

6.3.1 Modeling Electricity Demand Prediction in Victoria (Australia)

The prediction of electricity demand has been of much interest to the electricity supply industry for some years, both to aid long term planning strategies, involving the forecasting of seasonal peak demands, and for use in the short term (up to 24 hours) operation of generating plant. The nature of electricity market is changing very rapidly with a widespread international movement towards competitiveness. Traditionally, the energy sector, and particularly the electricity sector, has been dominated by monopoly or near monopoly enterprises, typically either owned or regulated by government. The recent privatization of the electricity supply industry has brought a renewed interest in this subject.

Some countries, such as Norway, Chile, Japan, UK and the United States have commonly been supplied electricity by a large number of different regional Generators and have developed a variety of mechanisms to allow some form of trade between them. In 1994 Victoria started the process of privatization and restructuring electricity industry to generate competition. The objective was to promote a more flexible, cost-effective and efficient electricity industry with the aim of delivering cheaper electricity to business and the general community. Following success of this operation, Australia started the process of implementing a unified National Electricity Market in December 1998 [14].

To meet the electricity market demands a highly reliable supply and delivery system is required. Additionally, in order to gain a competitive advantage in this market through the competitive spot-market pricing an accurate forecast of electricity demand at regular time intervals is essential. Until 1996, Victorian Power Exchange (VPX) the body responsible for the secure operations of the power system, generated electricity demand forecasts based on weather forecasts and historical demand patterns. Our research is focused on developing more accurate and reliable forecasting models that improve the current forecasting methods. Our approach is to develop reliable and accurate prediction models predicting 96 half-hourly (two days ahead) demands for electricity, and compares their performance with forecasts used by VPX. We considered an integrated neuro-fuzzy system and a feedforward artificial neural network trained using the scaled gradient conjugate algorithm and backpropagation algorithm. For developing the forecasting models, we used the energy demand data for ten months period from 27th January to 30th November 1995 in the State of Victoria. We also made use of the associated data stating the minimum and maximum temperature of the day, time of day, season and the day of week. The forecasting models were trained using 3 randomly selected samples containing 20% of the data during the period 27th January 1995 to 28 November 1995. To ascertain the forecasting accuracy the developed models were tested to predict the demand for the period (29-30) November 1995.

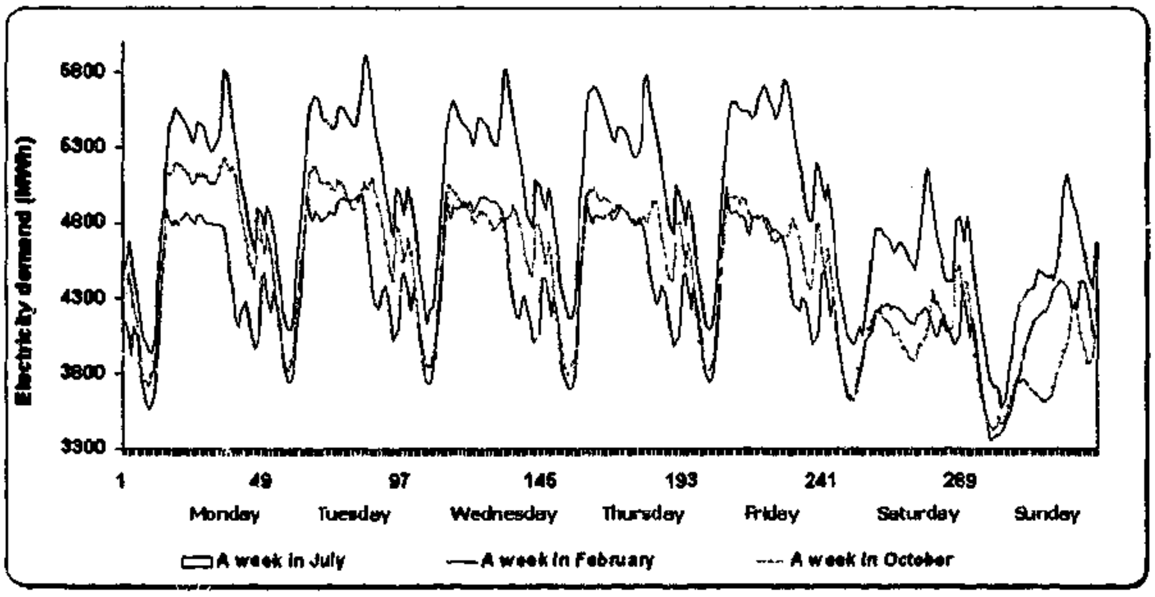


Figure 6.1. Typical weekly demand variations

The data for our study were the recorded half-hourly actual electricity demand for the ten months period from January to November 1995 in the State of Victoria. Figure 6.1 shows a typical weekly cycle of electricity demand during three different months of the year. Fluctuations in daily demand are prevalent with peaks occurring around midday. Extreme weather conditions in winter and summer months accentuate peaks in electricity demand due to the widespread use of electricity for heating and cooling. Other times, electricity demand is dominated primarily by ambient temperature, time of day, working or non-working day and the day of week.

The experimental system consists of two stages: modeling the prediction systems (training in the case of soft computing models) and performance evaluation. For network training, the six selected input descriptor variables were: the *minimum* and *maximum recorded temperatures*, *previous day's demand*, a value expressing the *half-hour period of the day*, *season*, and the *day of week*. To evaluate the learning capability of the soft computing models, the network was trained only on 20% of the randomly selected data. We created 3 different samples of training data to study the effect of random sampling and periodicity. Each training sample consisted of 2937 data sets representing 20% random data.

Our objective is to develop an efficient forecasting model capable of producing a short-term forecast of demand for electricity. The required time-resolution of the forecast is half-hourly, and the required time-span of the forecast is 2 days. This means that the system should be able to produce a forecast of electricity demand for the next 96 time periods. The training was replicated three times using three different samples of training data and different combinations of network parameters.

- **Neuro-Fuzzy Training**

We used 4 Gaussian membership functions for each input variable and the following evolving parameters: sensitivity threshold *Sthr*=0.99, error threshold *Errthr*=0.001 and learning rates for first and second layer = 0.05. EFuNN uses a one pass training approach. The network parameters were determined using a trial and error approach. The training was repeated three times after reinitializing the network and the worst errors were reported. Online learning in

EFuNN resulted in creating 2122 rule nodes. Training results and test results are summarized in Table 6.1.

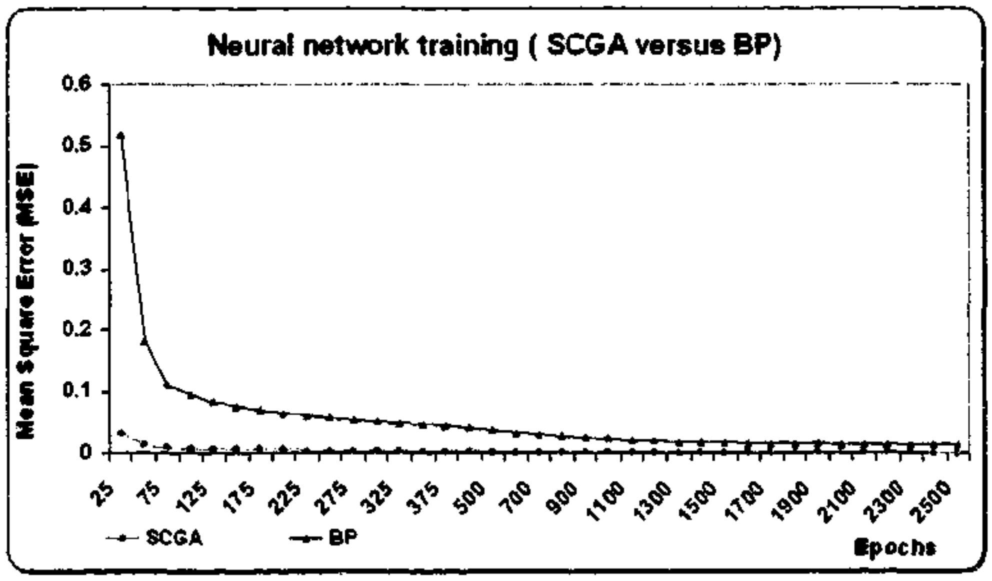


Figure 6.2. Convergence of neural network training

- **Neural Network training**

Our preliminary experiments helped us to formulate a feedforward neural network with 1 input layer, 2 hidden layers and an output layer [6-40-40-1]. Input layer consists of 6 neurons corresponding to the input variables. The first and second hidden layers consist of 40 neurons respectively using tanh-sigmoidal activation functions. To illustrate the convergence feature of Scaled Conjugate Gradient Algorithm (SCGA) we also trained a neural network (with same architecture) using backpropagation (BP) algorithm. To evaluate the neural network performance, training was terminated after 2500 epochs. Training and testing errors are summarized in Table 6.1. Figure 6.2 shows the convergence of SCGA with respect to BP algorithm. Figure 6.3 depicts the test results for the different prediction models considered. To have a performance evaluation the actual energy demand and the forecasts used by VHP and Box - Jenkins ARIMA model [51] are also plotted in Figure 6.3.

Compared to neural networks, an important advantage of neuro-fuzzy systems is its reasoning ability (*if-then* rules) of any particular state. A fully trained EFuNN could be replaced by a set of *if-then* rules [145]. A simple example of a learned EFuNN learned rule is illustrated below.

"If the *maximum temperature* of the day is HIGH **and** *minimum temperature* of the day is LOW **and** *previous days demand* is MEDIUM **and** it is *summer* (HIGH) **and** *9.00 AM* (HIGH) **and** a *Monday* (HIGH) **then** the *electricity demand* is MEDIUM."

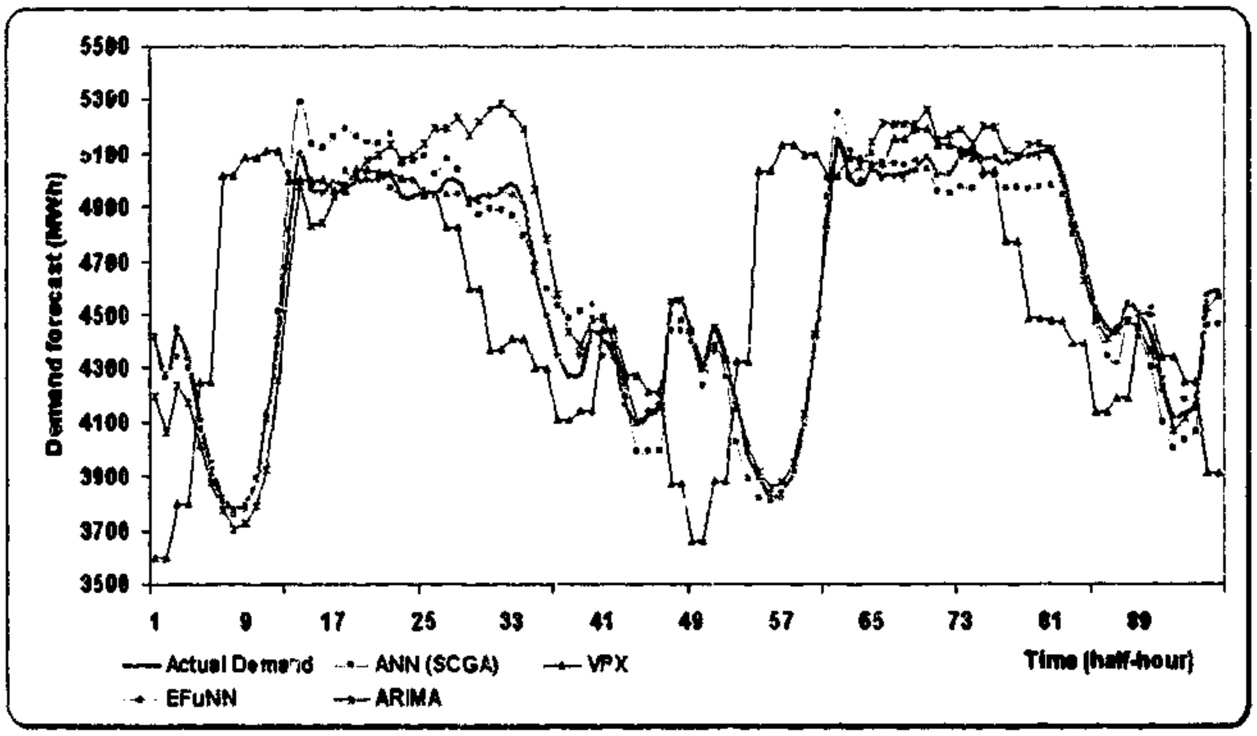


Figure 6.3. Test results and performance comparison of demand forecasts (2 days)

As EFuNN adopts a single pass training (1 epoch) it is more adaptable and easy for further online training which might be highly useful for online forecasting and bidding. Another important feature of EFuNN is that the user has the flexibility to construct the network (by selecting the parameters). Hence, for applications where speed is more important than the accuracy a faster network can be selected. However, an important disadvantage of EFuNN is the determination of the network parameters like number and type of membership functions for each input variable, sensitivity threshold, error threshold and the learning rates. Even though a trial and error approach is practical, when the problem becomes complicated (large number of input variables) determining the optimal parameters will be a tedious task.

Table 6.1. Test results and performance comparison of demand forecasting

	EFuNN	ANN (BP)	ANN (SCGA)	ARIMA
Learning epochs	1	2500	2500	-
Training error (RMSE)	0.0013	0.116	0.0304	-
Testing error (RMSE)	0.0092	0.118	0.0323	*0.0423
Computational load (in billion flops)	0.536	87.2	175.0	-

* results adapted from [190]

Our experiments on three separate data samples reveal that the results are not dependent on the data sample. We used only 20% of the total data to evaluate the learning capability of the soft computing models. Network performance could have been further improved by providing more training data. Another interesting fact about the considered soft computing models is their robustness and capability to handle noisy and approximate data that are typical in power systems, and therefore should be more reliable in worst situations.

6.3.2 Automation of Reactive Power Control

In this experiment, we present a comparative performance of two neuro-fuzzy models and an artificial neural network for automating the control of reactive power flow, which we had discussed in Chapter 4, Section 4.2.1. It is a well-established fact that improvement of the power factor and the addition of reactive power devices to the system can reduce the costs and release electrical capacity of the power distribution system. Most of the utility companies use a complex set of formulas, rewards/penalties etc. to receive an adequate return for their considerable investment in the larger capacity generators, transformers, cables and switchgear required to provide necessary KVA service to their customers. These formulas are generally referred to as power factor adjustments or KVAR reactive demand charges. In recent years, increased attention has been given to plant automation to reduce operational costs. Many manufacturing industries use human operators or timer controlled switching relays to turn on the power capacitors to compensate the reactive power requirement. Operational costs could be reduced and utilization efficiency improved if the power capacitor switching on/off process is automated using some intelligent techniques. We proposed a neuro-fuzzy approach to predict the reactive power trend (at time $t+1$) just by knowing the load current (at time t). Efficient usage of the VA loading will not only improve the overall grid condition but also reduce the consumer's industrial tariffs. Depending on the predicted reactive power demand, power factor corrective measures could be turned on or off to control the VA inflow into the plant. The developed prediction system will be extremely useful for automated control of power inflow, especially in the countries where there are limitations on the usage of consumers' peak VA maximum demand [3].

- **Importance of Reactive Power Control**

The ratio of active power (P) measured in watts to the apparent power (S) in volt-amperes is termed the power factor:

$$\text{Power factor} = \cos(\varphi) = \frac{P}{S} = \frac{resistance\,R}{impedance\,Z} \qquad (6.15)$$

It has become a normal practice to say that the power factor is lagging when the current lags the supply voltage and leading when the current leads the supply voltage. This means that the supply voltage is regarded as the reference quantity. A majority of loads served by a power utility draw current at a lagging power factor. When the power factor of the load is unity, active power equals apparent power ($P = S$). But, when the power factor of the load is less than unity, say 0.6, the power utilized is only 60%. This means that 40% of the apparent power is being utilized to supply the reactive power, VAR, demand of the system. It is therefore clear that the higher the power factor of the load, the greater the utilization of the apparent power. For the generating and transmission stations, lower the power factor the larger must be the size of the source to generate that power, and greater must be the cross-sectional area of the conductor to transmit it. In other words, the greater is the cost of generation and transmission of the power. Moreover, lower power factor will also increase the I^2R (I denotes current) losses in lines/equipment as well as result in poor voltage regulation.

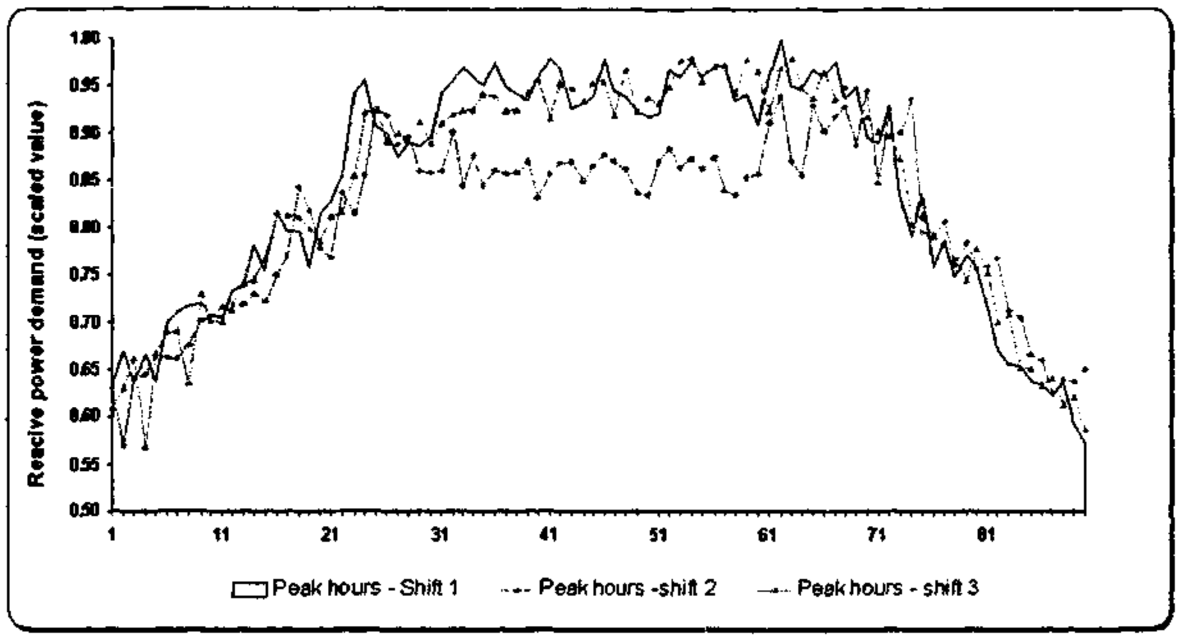


Figure 6.4. Reactive power demand variations during peak hours

We considered a heavy automobile manufacturing industry that works on 3 shifts of 8 hours duration for studying the load demand patterns. Observed data for a 24 hour period shows that the maximum and minimum VAR requirements are 2.96 MVAR and 0.014 MVAR, respectively. If suitable power factor compensation was made when the reactive power demand was increasing, the plant might not have drawn much apparent power from the grid. The task is to predict the upward and downward trend of the reactive power demand and provide required reactive power compensation. Load flow analysis of the captioned plant reveals that the demand patterns are very similar every day (as long as the production of automobiles remain fixed). Neuro-fuzzy systems and neural networks are perhaps the best techniques for learning relationships amongst variables (function approximation). For this problem we used two neuro-fuzzy models implementing a Takagi-Sugeno fuzzy inference system and a Mamdani fuzzy inference system. We used the adaptive network based fuzzy inference system to implement the Takagi-Sugeno fuzzy inference system and the evolving fuzzy neural network to implement the Mamdani fuzzy inference system. For comparison purposes, we also trained a feedforward artificial neural network using the backpropagation algorithm. The proposed neuro-fuzzy models and neural network were trained on the data taken at every minute for a 24-hour period to predict the reactive power demand, and tested to evaluate the prediction accuracy. To evaluate the efficiency of the prediction models, three different training and testing data sets were extracted and the experiments were performed three times.

Experimentation setup and test results

The experimental system consists of two stages: network (connectionist model) training and performance evaluation. A heavy automobile manufacturing plant was considered for the prediction of reactive power. 24-hour load flow patterns were used to train the neuro-fuzzy models and neural network. The training data comprises of 1440 data sets representing the 24-hour period. The input parameters considered are the phase voltage (*V*) and current (*I*). The normal value of input parameter voltage (*V*) was fluctuated with +/- 2.5% of the normal value. All the data sets were scaled to (0-1). The input voltage was fluctuated to test the learning capability and robustness of the considered connectionist models. As shown in Figure 4.5(a), fluctuated voltage appears to be a heavy noise to the network. This also ensures that the proposed models could predict the reactive power accurately even during worst conditions in the grid voltage regardless of the plant load. Training and testing data sets were extracted randomly from the complete dataset. 60% of data was used for training and remaining 40% for testing. To ensure that the data sample does not have any bias, we created 3 sets of data for training and testing (random extraction). Experiments with all 3 data sets were repeated 3 times for all the connectionist models.

- **Neural network training**

We used a feedforward neural network with 2 hidden layers and trained using the backpropagation algorithm. The 2 input neurons correspond to the input variables and 1 output neuron for predicting reactive power. Initial weights, learning rate and momentum used were 0.3, 0.1 and 0.1, respectively. The training was terminated after 700 epochs.

- **ANFIS training**

In the ANFIS network, we used 3 Gaussian membership functions for each input parameter variable for predicting the reactive power demand. Nine rules were learned based on the training data. The training was terminated after 50 epochs.

- **EFuNN training**

We used 3 Gaussian membership functions and the following evolving parameters: sensitivity threshold *Sthr*=0.95, error threshold *Errthr*=0.05 and 544 rule nodes were created during training.

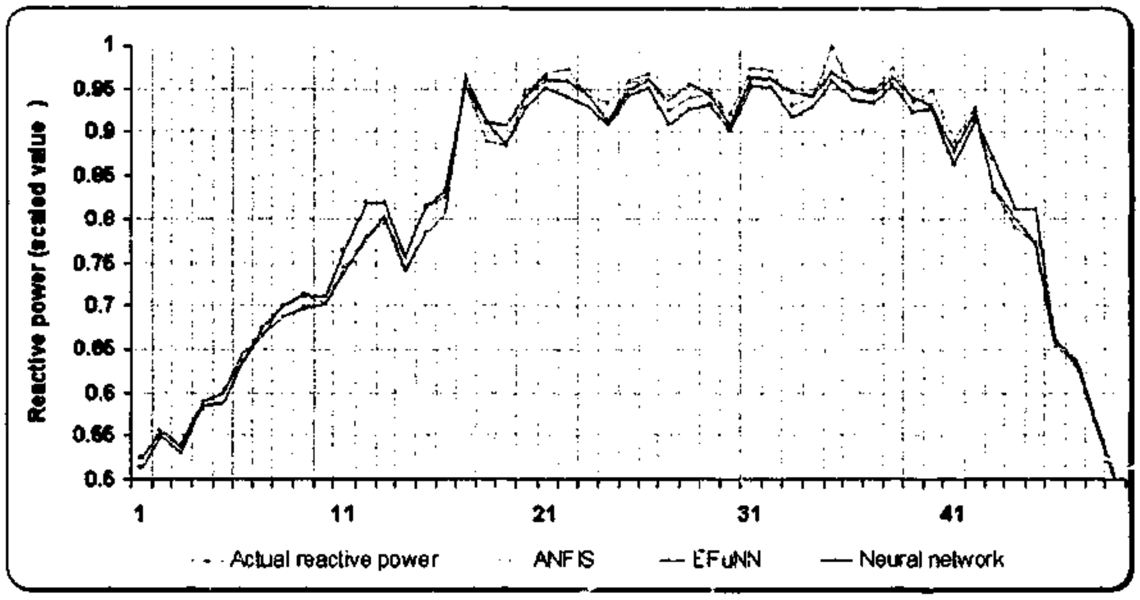


Figure 6.5. Test results showing the predicted reactive power using different models during the peak hours of shift 1.

Table 6.2. Reactive power prediction –comparative performance

	ANFIS	EFuNN	ANN
Learning epochs	50	1	700
Training time (seconds)	36	25	188
Training error (RMSE)	0.0103	0.0116	0.0142
Testing error (RMSE)	0.0102	0.0120	0.0130

- **Performance and results achieved**

Test data (input parameters) is passed through the trained connectionist models and the predicted output value is compared with the observed reactive power value to calculate the

RMSE. Figures 6.5 illustrates the test results for predicted outputs using ANFIS, EFuNN and ANN. Table 6.2 shows an empirical comparative performance of the different connectionist models for the reactive power prediction problem. The empirical values shown in Table 6.2 are the worst values of the three trials with the three data sets for each model.

Among all the connectionist models neuro-fuzzy systems performed better than artificial neural network in terms of performance error achieved and training time. ANFIS performed marginally better than EFuNN in terms of low error. However ANFIS took more training time than EFuNN. Hence, there is a compromise between performance error and training time. An important advantage of the EFuNN network is its online learning capability. Hence future training would be much easier. The predicted RMSE values are within acceptable rates and hence the developed models are reliable. The prediction accuracy could have been improved if we had not used the noisy input parameter (voltage) or if the actual voltage values were used. The results also show that the considered connectionist models are very robust, capable of handling the noisy and approximate data that are typical in power systems, and therefore should be more reliable during worst conditions. By implementing the proposed technique, reactive power flow could be managed more efficiently by avoiding the use of operators and timer controlled switching relays (which could be inefficient sometimes).

6.3.3 Weather Forecast Models Using Neuro-Fuzzy Systems

Rain is one of the nature's greatest gifts and in third world countries like India; the entire agriculture depends upon rain. It is thus a major concern to identify any trends for rainfall to deviate from its periodicity, which would disrupt the economy of the country. This fear has been aggravated due to threat by the global warming and green house effect. The geographical configuration of India with the three oceans, namely Indian Ocean, Bay of Bengal and the Arabian Sea bordering the peninsula gives her a climate system with two monsoon seasons and two cyclones interspersed with hot and cold weather seasons. The parameters that are required to predict the rainfall are enormously complex and subtle so that the uncertainty in a prediction using all these parameters even for a short period. The period over which a prediction may be made is generally termed the event horizon and in best results, this is not more than a week's time. Thus it is generally said that the fluttering wings of a butterfly at one corner of the globe may cause it to produce a tornado at another place geographically far away. Edward Lorenz (meteorologist at MIT) discovered this phenomenon in 1961 and is popularly known as the butterfly effect. In our research, we aim to find out how well the proposed soft computing models are able to understand the periodicity in these patterns so that long-term predictions can be made. This would help one to anticipate with some degree of confidence the general pattern of rainfall to be expected in the coming years. In pace with the global interest in climatology, there has been a rapid updating of resources in India also to access and process climatological database. There are various data acquisition centres in the country that record daily rainfall along with other measures such as sea surface pressure, temperature etc. that are of interest to climatological processing. These centres are also associated with the World Meteorological Organization (WMO) [21].

Long-term rainfall prediction is very important to countries thriving on agro-based economy. The parameters that are required to predict the rainfall are enormously complex and subtle so that uncertainty in a prediction using all these parameters is enormous even for a short period. In this research, we analysed 87 years of rainfall data in Kerala state, the southern part of Indian Peninsula situated at latitude-longitude pairs (8°29′ N - 76°57′ E). We attempted to train 5 soft computing based prediction models with 40 years of rainfall data. For performance evaluation, network predicted outputs were compared with the actual rainfall data.

We used an artificial neural network using backpropagation (variable learning rate), adaptive basis function neural network [207], neural network using scaled conjugate gradient algorithm and an Evolving Fuzzy Neural Network for predicting the rainfall time series. The soft computing models described above were trained on the rainfall data corresponding to a certain period in the past and cross validate the prediction made by the network over some other period.

- **Neural Networks with Variable Learning Rates**

With standard steepest descent, the learning rate is held constant throughout the training. If the learning rate is too high, the algorithm may oscillate and become unstable. If the learning rate is too small, the algorithm will take too long to converge. 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. The performance of the steepest descent algorithm can be improved by using an adaptive learning rate, which will keep the learning step size as large as possible while keeping learning stable. The learning rate is made adaptive to the complexity of the local error surface. If the new error exceeds the old error by more than a predefined ratio (typically 1.04), the new weights are discarded. In addition, the learning rate is decreased (typically by 70%). Otherwise the new weights are kept. If the new error is less than the old error, the learning rate is increased (typically by 5%). Thus a near optimal learning rate is obtained for the local terrain. When a larger learning rate could result in stable learning, the learning rate is also increased. When the learning rate is too high to guarantee a decrease in error, it gets decreased until stable learning resumes.

Adaptive Basis Function Neural Network (ABFNN) performs better than the standard BP networks in complex problems [207]. The ABFNN works on the principle that the neural network always attempt to map the target space in terms of its basis functions or node functions. In standard BP networks, this function is a fixed sigmoid function that can map between zero and plus one (or between minus one and plus one) the input applied to it from minus infinity to plus infinity. It has many attractive properties that made the BP an efficient tool in a wide verity of applications. However some studies conducted on the BP algorithm have shown that in spite of its wide spread acceptance, they systematically outperform other classification procedures only when the targeted space has a sigmoidal shape. This implies that one should choose a basis function such that the network may represent the target space as a nested sum of products of the input parameters in terms of the basis function. The ABFNN thus starts with the standard sigmoid basis function and alters its non-linearity by an algorithm similar to the weight update algorithm used in BP. Instead of the standard sigmoid function, ABFNN uses a variable sigmoid function defined as:

$$O_f = \frac{a + \tanh(x)}{1 + a} \qquad (6.16)$$

where *a* is the control parameter that is initially set to unity and is modified along with the connection weights along the negative gradient of the error function. Such a modification could improve the speed of convergence and accuracy with which the network could approximate the target space.

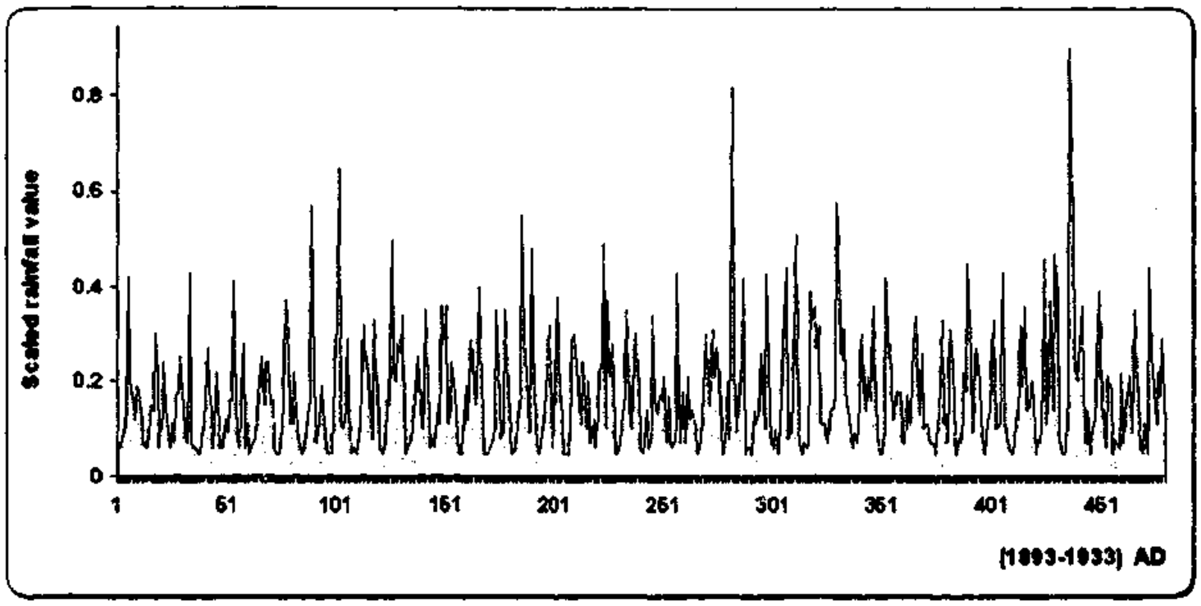


Figure 6.6. Complexity of the rainfall database: Average monthly rainfall from (1893-1933) AD (training data)

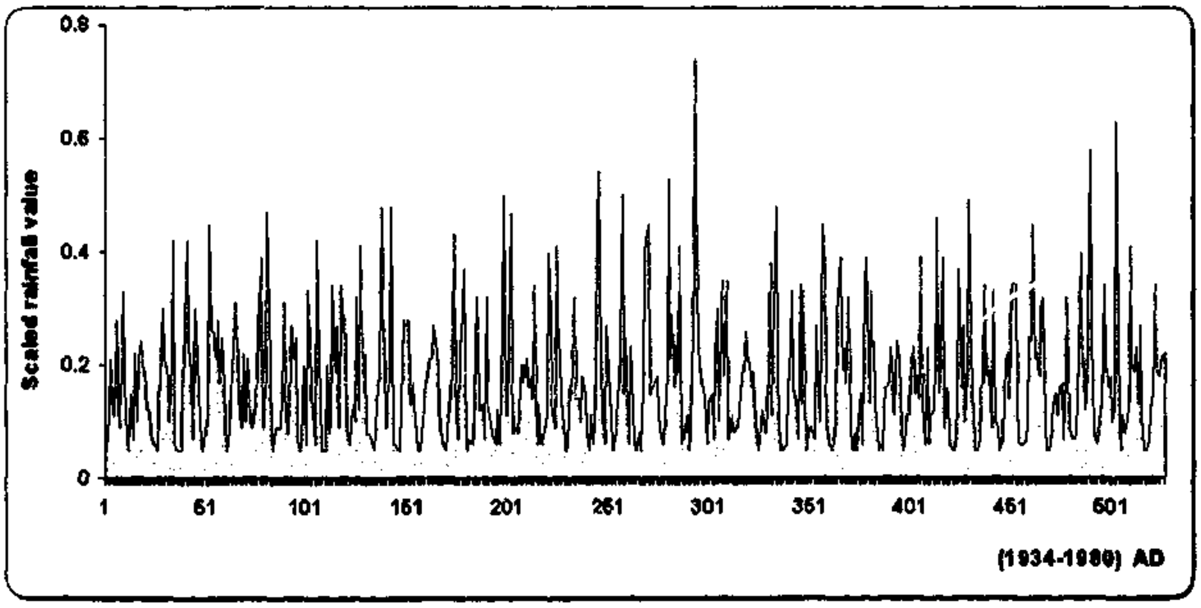


Figure 6.7. Complexity of the rainfall database: Average monthly rainfall from (1934-1980) AD (test data)

- **Experimentation Setup for Training and Performance Evaluation**

The rainfall data was scaled (0-1) and we divided the data from 1893-1933 as training set and data from 1934-1980 as test set. While the proposed neuro-fuzzy system is capable of adapting the a hitecture according to the problem we had to perform some initial experiments to decide .he architecture of the neural network. Since rainfall has a yearly periodicity, we started with a network having 12 input nodes. Further experimentation showed that it was not necessary to include information corresponding to the whole year, but 3-month information centered over the predicted month of the fifth year in each of the 4 previous years would give good generalization properties. Thus, based on the information from the four previous years, the network would predict the amount of rain to be expected in each month of the fifth year. We used the same architecture for all the three learning neural network learning algorithms. To have a performance comparison of the different learning techniques, the training was terminated after 1000 epochs. The training was repeated three times after re-initializing the networks. Test data was presented to the network and the output from the network was compared with the actual data in the time series. The worst observed errors are reported. Following are the details of network training:

Neuro-fuzzy training

We used 5 membership functions for each input variable and the following evolving parameters: sensitivity threshold *Sthr*=0.999, error threshold *Errthr*=0.001.

ANN Training

For neural networks using BP, backpropagation with variable learning rate (BP-VLR) and SCGA, we used 1 input layer, 2 hidden layers and an output layer [12-12-12-1]. Input layer consists of 12 neurons corresponding to the input variables. The first and second hidden layer consists of 12 neurons. For the ABFNN network, we used only I hidden layer with 7 neurons. Training errors (RMSE) achieved are reported in Table 6.3. To have a performance evaluation between the 4 learning algorithms, we also trained a neural network (12-7-1) with one hidden layer containing 7 neurons and the training was terminated after 1000 epochs for all the four learning methods. Figure 6.8 shows the training performance and convergence of the four neural network algorithms.

Test Results

Table 6.3 summarizes the comparative performance of EFuNN and ANN learning algorithms. Figure 6.9 depicts the comparative performance between the different soft computing models.

Table 6.3. Test results and performance comparison of rainfall forecasting

	EFuNN	ANN (BP)	ANN (VLR)	ANN (SCGA)	ANN (ABF)
Learning epochs	1	10000	10000	600	1000
Training error (RMSE)	0.0006	0.0954	0.0875	0.0780	0.0800
Testing error (RMSE)	0.0901	0.0948	0.0936	0.0923	0.0930
Computational load (in billion flops)	0.065	8.82	8.75	1.26	-

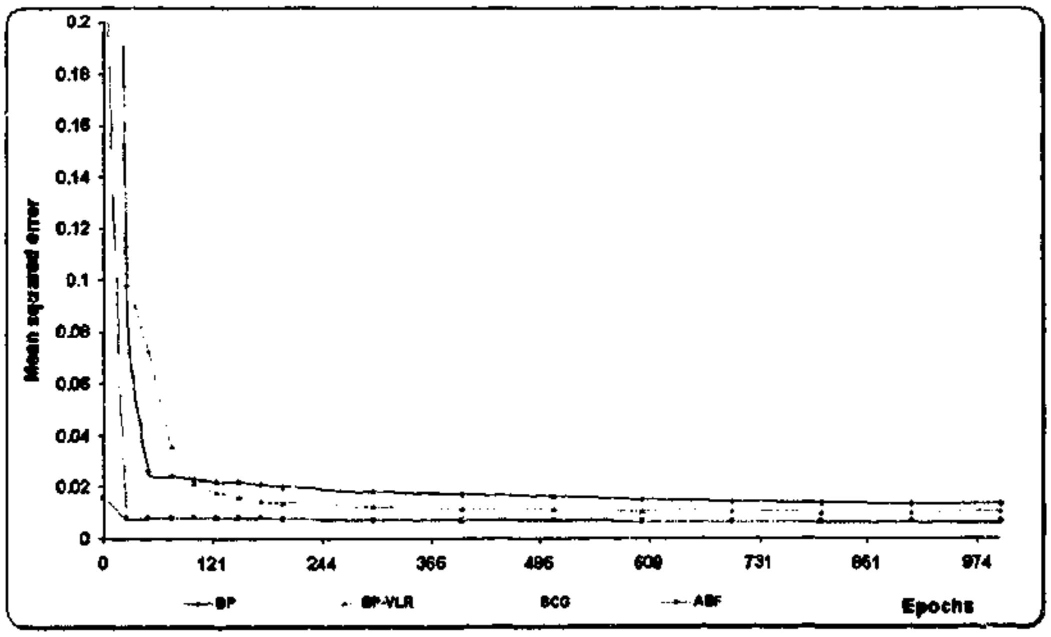


Figure 6.8. Convergence of neural network learning algorithms (for 1000 epochs).

As the RMSE values on test data are comparatively less, the prediction models are reliable. As evident from Figure 6.9, there have been few deviations of the predicted rainfall value from the actual. In some cases it is due to delay in the actual commencement of monsoon, El-Nino Southern Oscillations (ENSO) resulting from the pressure oscillations between the tropical Indian Ocean and the tropical Pacific Ocean and their quasi periodic oscillations [69]. The integrated neuro-fuzzy technique outperformed neurocomputing techniques with the lowest RMSE test error and performance time. Compared to pure BP and BP-VLR, ABFNN and SCGA converged much faster. Alternatively, BP training needs more epochs (longer training time), to achieve better performance. Compared to ANN, an important advantage of neuro-fuzzy model is its reasoning ability (*if-then* rules) of any particular state. As climate and rainfall prediction involves tremendous amount of imprecision and uncertainty, neuro-fuzzy technique might warrant the ideal prediction model.

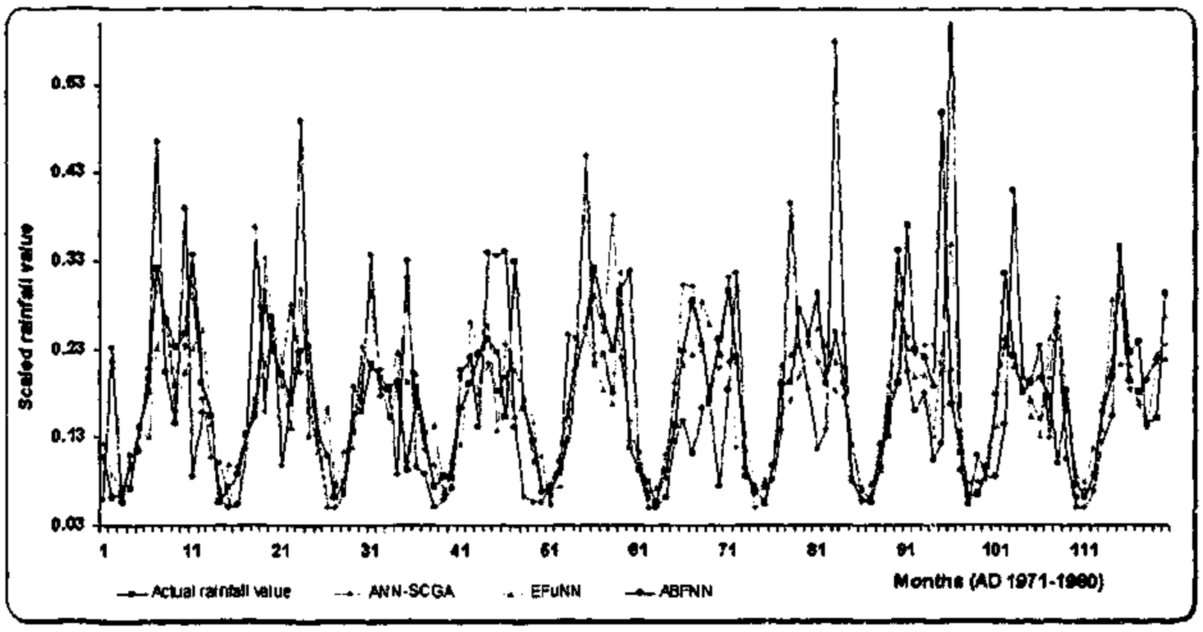


Figure 6.9. Test results showing monthly prediction of rainfall for 10 years using the different connectionist models

The proposed prediction models based on soft computing on the other hand are easy to implement and produces desirable mapping function by training on the given data set. A network requires information only on the input variables for generating forecasts. In our experiments, we used only 40 years training data to evaluate the learning capability. Network performance could have been further improved by providing more training data. Moreover, the considered connectionist models are very robust, capable of handling the noisy and approximate data that are typical in weather data, and therefore should be more reliable in worst situations.

6.4. Conclusions

In this Chapter, we have presented the learning mechanisms of ANFIS and EFUNN. We have also demonstrated how integrated neuro-fuzzy systems could be used for solving practical problems. In all the 3 applications presented, neuro-fuzzy systems have outperformed neural networks using different learning algorithms.

However, an important disadvantage of integrated neuro-fuzzy system is the careful determination of the network parameters like number and shape of membership functions for each input variable, learning rates and an efficient technique to determine the initial rule base, fuzzy operators etc. Even though EFuNN constructs the rule base automatically, the performance of the network still depends on the careful selection of sensitivity threshold, error threshold learning rates etc. Even though a trial and error approach is practical, when the problem becomes complicated (large number of input variables) determining the optimal parameters to build an optimal network will be a difficult task.

In Chapter 7, we will discuss the issues related to modeling neuro-fuzzy systems and how it could be overcomed using evolutionary computation.

Chapter 7: Evolutionary Design of Neuro-Fuzzy Systems

7.0 Introduction

As described in Section 5.3, an integrated neuro-fuzzy system is a combination of artificial neural network and fuzzy inference system in such a way that neural network learning algorithms are used to determine the parameters of fuzzy inference system. A wide variety of neural network learning algorithms and different types of fuzzy inference systems are available and it might be a puzzling task to determine which combination might be the optimal for solving a particular problem.

Potential interactions between connectionist learning systems, fuzzy inference systems and evolutionary search procedures have attracted considerable research work recently [12] [9] [10]. We have already discussed evolutionary neural networks and evolutionary fuzzy systems in Chapter 3 and Chapter 4 respectively. In an integrated neuro-fuzzy model there is no guarantee that the neural network-learning algorithm converges and the tuning of fuzzy inference system will be successful. Success of evolutionary search procedures for optimal design of neural networks and fuzzy inference system are well proven and established in many application areas. In this chapter, we will explore how the integration of neural networks and fuzzy inference systems could be optimized using evolutionary search procedures. We present the theoretical frameworks and some experimental results to demonstrate the efficiency of the proposed technique.

7.1.Integration of Neural networks, Fuzzy Inference systems and Evolutionary Computation

Artificial neural networks and fuzzy inference systems are both very powerful soft computing tools for solving a problem without having to analyze the problem itself in detail. Natural intelligence is a product of evolution. Therefore, by mimicking biological evolution, we could also simulate high-level intelligence. The evolutionary approach to artificial intelligence is based on the computational models of natural selection and genetics. Evolutionary computation works by simulating a population of individuals, evaluating their performance, and evolving the population a number of times until the required solution is obtained.

The drawbacks pertaining to neural networks and fuzzy inference systems seem complementary and evolutionary computation could be used to optimize the integration to produce the best possible synergetic behavior to form a single system. The integrated architecture share data structures and knowledge representations. The parameters of the fuzzy inference system will be fine-tuned using evolutionary algorithms and neural network learning techniques. In such an integrated learning occurs at two levels: evolutionary learning (global optimization) and a local search by conventional neural network algorithm (gradient descent). Evolutionary algorithms could also be used to determine the optimal learning parameters of the gradient descent technique.

Why Optimize Neuro-Fuzzy Systems?

Mamdani type fuzzy inference system possess a high degree of freedom to select the most suitable fuzzification and defuzzification interface components as well as the inference mechanism itself. Mamdani type FIS provides a highly flexible means to formulate knowledge, while at the same it remains interpretable. Some disadvantages of Mamdani type fuzzy inference system are the following:

- Lack of flexibility (rigid partitioning of input-output spaces). When the input variables are mutually dependent it becomes very difficult to form a good partition of the input space. Sometimes uniform partitioning (usually adopted method) is inefficient and does not scale well.
- The size of the rule base increases rapidly with the number of variables and linguistic terms in the system.

To accommodate the deficiencies mentioned above, researchers have proposed variants of the Mamdani inference system [168].

The main advantage of Takagi-Sugeno type fuzzy inference method is that it presents a set of compact system equations that allows the rule consequent parameters (4.17) to be estimated using classical methods, which facilitates the design process. This could be also interpreted as a main drawback as this inference system does not provide a natural framework for representing expert knowledge that is afflicted with uncertainty and thus not able to fully utilize the fuzzy logic capability. Due to the rule consequents, incorporating expert knowledge often becomes very difficult. Takagi-Sugeno fuzzy systems are more difficult to interpret since the overall output depends on the simultaneous activation of the rule antecedents, and on the function in the rule consequent that depends on the crisp inputs as well rather than being a constant.

It is interesting to note that Takagi-Sugeno-type fuzzy systems are high performers (more accuracy) but often requires complicated learning procedures and computational expensive. On the other hand, Mamdani-type fuzzy systems can be modeled using faster heuristics but with a compromise on the performance (high RMSE). Hence, there is always a compromise between performance and computational time. Most of the integrated neuro-fuzzy systems currently available are based on either Mamdani-type or Takagi-Sugeno-fuzzy inference system. Hence selection of a good inference system itself becomes complicated when the dimensionality and complexity of the input-output mapping increases. As evident from Chapter 6, for a successful design of a neuro-fuzzy design, the user has to specify the shape and quantity of the membership function for each input/output variable, fuzzy operators, defuzzification method, fuzzy inference mechanism etc. The user also has to specify the rule base (except EFuNN and NEFCON) and the learning technique that will fine-tune the membership functions and other tunable parameters. We are familiar with "trapped in local minima" whenever we refer to local search techniques. Since the neuro-fuzzy systems use gradient descent method, there is no guarantee that global optima would be obtained and the parameters are fine-tuned. Evolutionary algorithms are popular for obtaining a global optimal solution but not often well in local searches. Integrating evolutionary computation (a global optimization technique) with a local search technique might help to explore the solution space more effectively.

7.2. Generic Architecture of Evolving Neuro-Fuzzy Systems (EvoNF)

In this section, we define the architecture of EvoNF, a computational framework to optimize fuzzy inference systems using evolutionary computation and neural network learning algorithms. The proposed framework could adapt to Mamdani, Takagi-Sugeno or other fuzzy inference systems. The architecture and the evolving mechanism can be considered as general framework for adaptive neuro-fuzzy systems, that is an integrated neuro-fuzzy model that can change their membership functions (quantity and shape), rule base (architecture), fuzzy operators and learning parameters according to different environments without human intervention.

Figure 7.1 shows how a Mamdani or Takagi-Sugeno fuzzy inference system could be adapted in an integrated framework. Architecture details of Mamdani and Takagi-Sugeno fuzzy inference system have been provided earlier in Sections 5.3.1 and 5.3.2. Solving multiobjective scientific and engineering problems is, generally, a very difficult goal. In these particular optimization problems, the objectives often conflict across a high-dimension problem space and may also require extensive computational resources. We propose a 5-tier evolutionary search procedure wherein the membership functions, rule base (architecture), fuzzy inference mechanism (T-norm and T-conorm operators), learning parameters and finally the type of inference system (Mamdani, Takagi-Sugeno etc.) are adapted according to the environment.

Figure 7.2 illustrates the interaction of various evolutionary search procedures. Referring to Figure 7.2, for every fuzzy inference system, there exist a global search of learning algorithm, inference mechanism, rule base and membership functions in an environment decided by the problem. Thus the evolution of the fuzzy inference system will evolve at the slowest time scale while the evolution of the quantity and type of membership functions will evolve at the fastest rate. This hierarchical evolution is very much similar to the MLEANN framework proposed in Section 3.3.

The function of the other layers could be derived similarly. For every learning algorithm (parameter), there is the global search of inference mechanisms, rule base and membership functions that proceed on a faster time scale in an environment decided by the fuzzy inference system and the problem. For every inference mechanism (fuzzy operators) there is the global search of rule base and membership functions that proceeds on a faster time scale in an environment decided by the learning algorithm, fuzzy inference system and the problem.

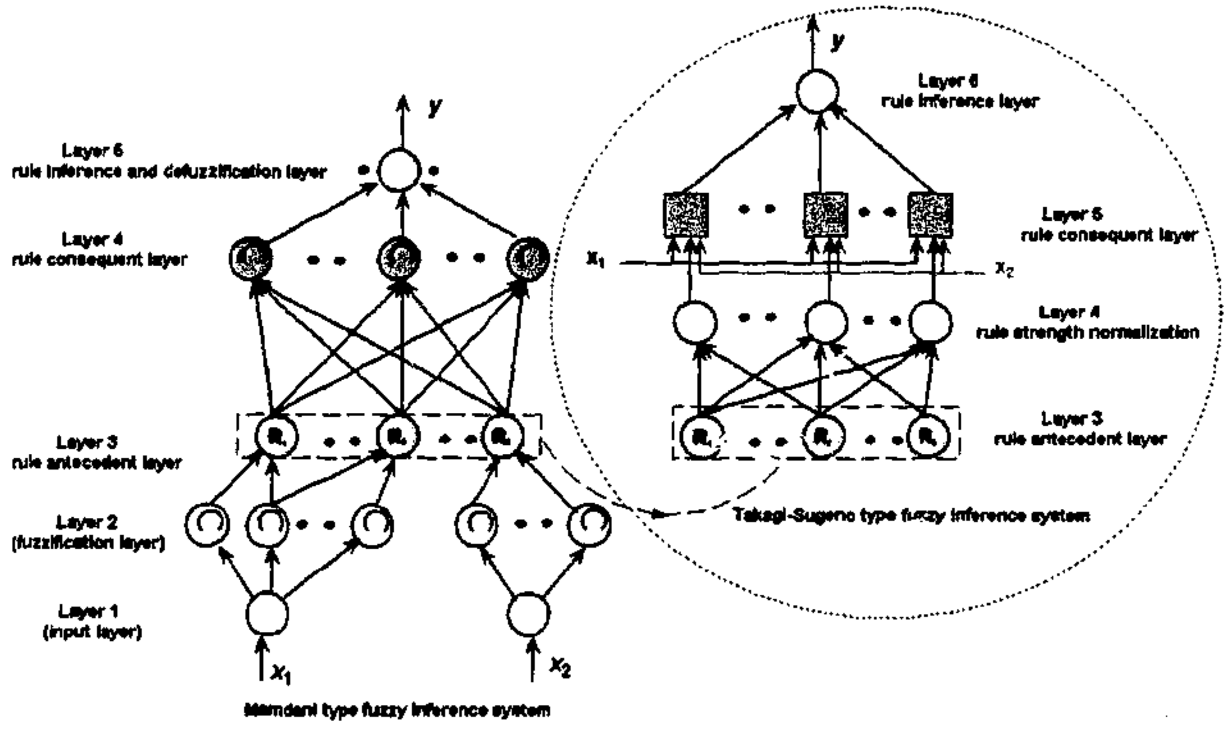
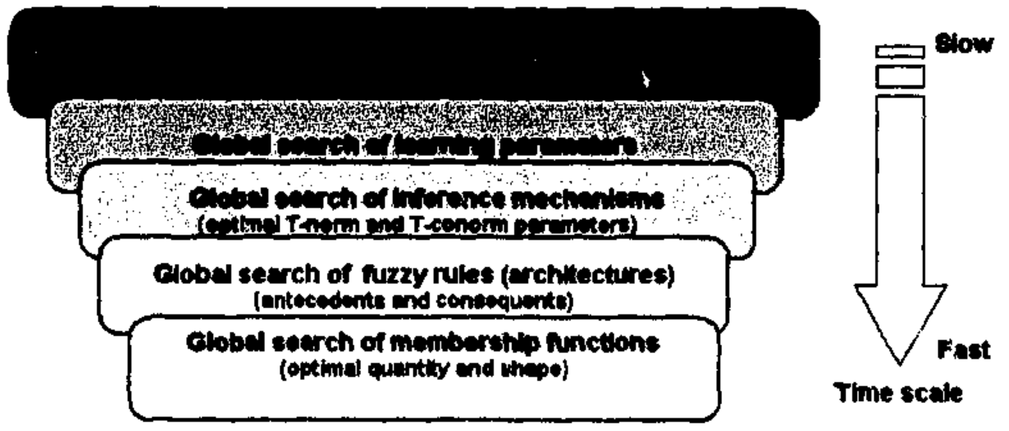


Figure 7.1 Architecture of self-constructing EvoNF model



Global search of learning parameters
Global search of inference mechanisms
(optimal T-norm and T-conorm parameters)
Global search of fuzzy rules (architectures)
(antecedents and consequents)
Global search of membership functions
(optimal quantity and shape)
Slow
Fast
Time scale

Figure 7.2. General framework for EvoNF

Likewise for every architecture, evolution of membership function parameters proceeds at the faster time scale in an environment decided by the inference mechanism, learning algorithm, type of fuzzy inference system and the problem. Hierarchy of the different adaptation layers (procedures) will rely on the prior knowledge. For example, if there is more prior knowledge about the architecture than the inference mechanism then it is better to implement the architecture at a higher level. If we know that a particular fuzzy inference system will suit best for the problem, we reduce the computational task by minimizing the search space.

7.3.Parameterization of the Inference System

For fine-tuning the integrated neuro-fuzzy model all the nodes functions are to be parameterized. Evolutionary search of optimal inference procedure could only be formulated if all the node functions are parameterized. The parameters could be further fine tuned by evolutionary learning or any neural network learning algorithm or a combination of both.

- **Parameterization of Membership Functions**

Fuzzy inference system is completely characterized by its membership function. A generalized bell MF is specified by three parameters (p, q, r) and is given by:

$$Bell\,(x, p, q, r) = \frac{1}{1 + \left|\frac{x - r}{p}\right|^{2q}}$$

Figures 7.3 (a-d) shows the effects of changing *p, q* and *r* in a bell membership function. Similar parameterization can be done with most of the other membership functions.

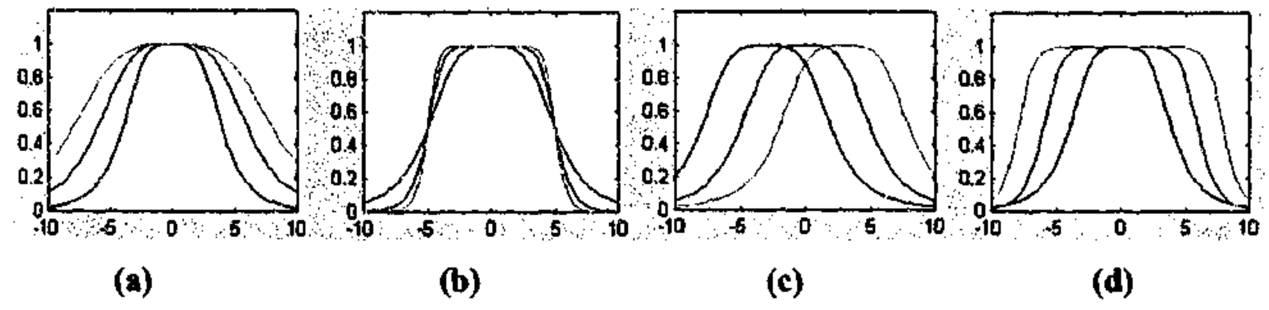


Figure 7.3. (a) Changing parameter *p* **(b)** changing parameter *q* **(c)** changing parameter *r (d)* changing *p* and *q*

- **Parameterization of T-norm and T-conorm operators**

T-norm is a fuzzy intersection operator, which aggregates the intersection of two fuzzy sets A and B while T-conorm operators compute fuzzy union of two fuzzy sets A and B. The Schweizer and Sklar's T-norm and T-conorm operator can be expressed as:

$$T(a,b,p) = \left[max\left\{0, (a^{-p} + b^{-p} - 1)\right\}\right]^{-\frac{1}{p}} \tag{7.1}$$

$$S(a,b,p) = 1 - \left[max\left\{0, ((1-a^{-p}) + (1-b^{-p}) - 1)\right\}\right]^{-\frac{1}{p}} \tag{7.2}$$

It is observed that

$$\begin{aligned} lim_{p\to 0}\ T(a,b,p) &= ab \\ lim_{p\to\infty}\ T(a,b,p) &= min\{a,b\} \end{aligned} \tag{7.3}$$

which correspond to two of the most frequently used T-norms in combining the membership values on the premise part of a fuzzy if-then rule [215]. To give a general idea of how the parameter *p* affects the T-norm and T-conorm operators, Figure 7.4 (a) shows two fuzzy sets *A* and *B* and Figure 7.4 (b) and Figure 7.4 (c) are $T_{(a,b,p)}$ and $S_{(a,b,p)}$ respectively.

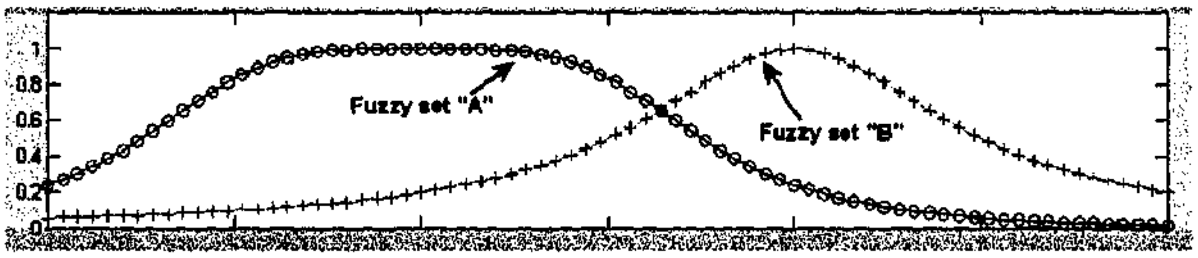


Figure 7. 4(a). Bell Membership functions for fuzzy set *A* and *B*

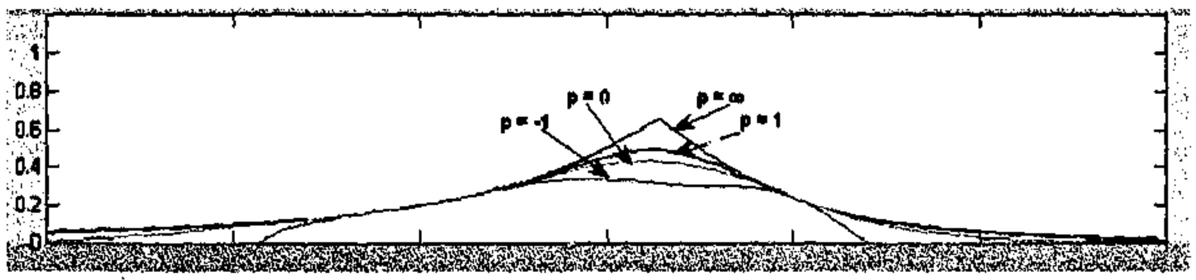


Figure 7. 4(b). Effects of changing parameters of T-norm operators

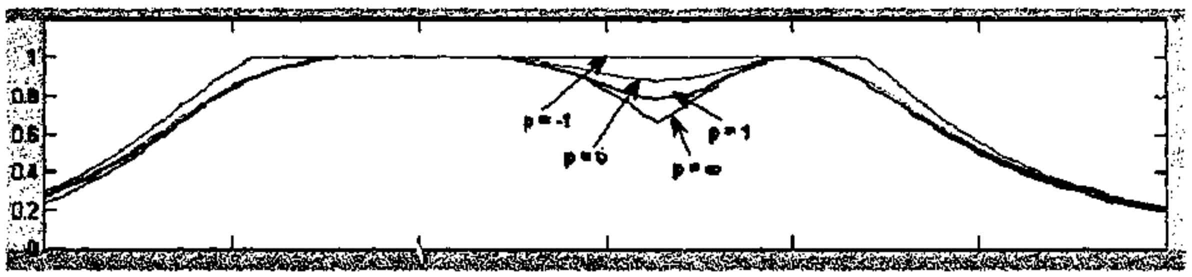


Figure 7. 4(c). Effects of changing parameters of T-conorm operators

7.4.Chromosome Modeling and Representation Issues

The antecedent of a fuzzy rule defines a local region, while the consequent describes the behavior within the region via various constituents. Basically, the antecedent part remains the same regardless of the inference system used. Different consequent constituents result in different fuzzy inference systems. For applying evolutionary algorithms, problem representation (chromosome) is very important as it directly affects the proposed algorithm. Referring to Figure 7.2, each layer (from fastest to slowest) of the hierarchical evolutionary search process has to be represented in a chromosome for successful modeling of an integrated neuro-fuzzy system using evolutionary computation. A typical chromosome of the EvoNF mode! would appear as shown in Figure 7.5 and the detailed modeling process (refer to Figure 7.2) is as follows.

Layer 1: The simplest way is to encode the number of membership functions per input variable and the parameters of the membership functions. Figure 7.7 depicts the chromosome representation of *n bell* membership functions specified by its parameters *p*, *q* and *r*. The optimal parameters of the membership functions located by the evolutionary algorithm will be later fine tuned by the neural network-learning algorithm. Similar strategy could be used for the output membership functions in the case of a Mamdani fuzzy inference system. Experts may be consulted to estimate the MF shape forming parameters to estimate the search space of the MF parameters. Ascold et al defined a second order fuzzy set [173] to determine the upper and the lower boundaries of the MF parameters.

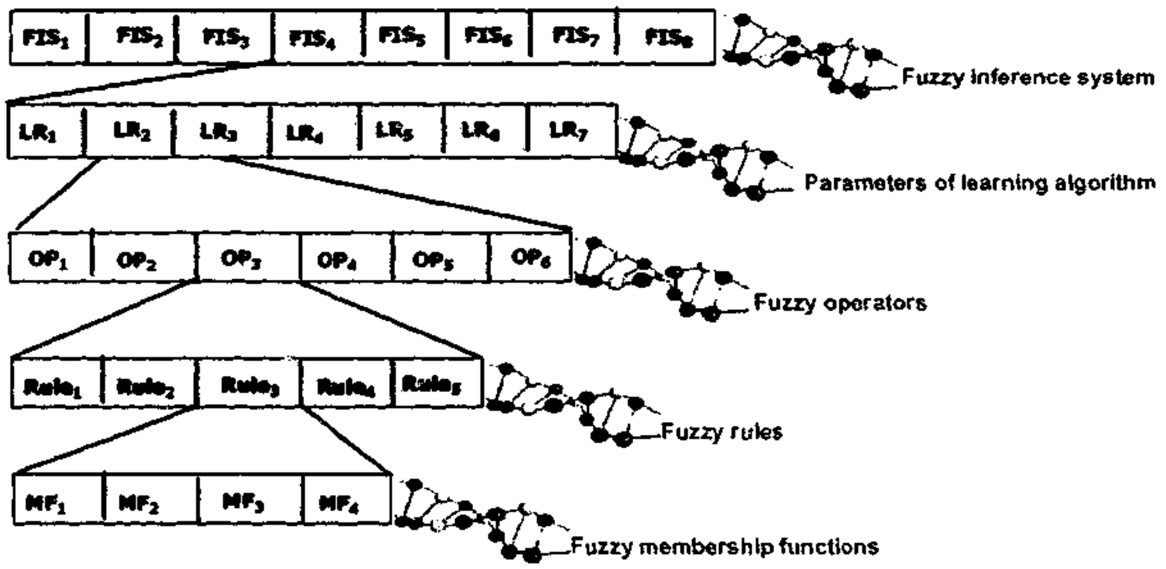


Figure 7.5. Chromosome structure of the EvoNF model

We used the angular coding method proposed by Herrera et al [71] for representing the rule consequent parameters of the Takagi-Sugeno inference system. Rather than directly coding the consequent parameters, the "transformed" parameters represent the direction of the tangent $\alpha_i = arctan\ p_i$. The range for the parameters α_i is the interval (-90^0, +90^0), such that the parameters p_i can assume any real value. A single input Takagi-Sugeno system $Y = p_1 X + p_0$ defines a straight line. The real value p_1 is simply the gradient between this line and the X-axis. Parameter p_0 determines the offset of the straight line (intercept) along the Y-axis. Angular coding is advantageous, since the value of p_0 varies between different rules and it is difficult to use some fixed interval to exploit the search space. The procedure is illustrated in Figure 7.6.

Layer 2. This layer is responsible for the optimization of the rule base. This includes deciding the total number of rules, representation of the antecedent and consequent parts. The number of rules grow rapidly with an increasing number of variables and fuzzy sets. The simplest way is that each gene represents one rule, and "1" stands for a selected and "0" for a non-selected rule. Figure 7.8 displays such a chromosome structure representation. To represent a single rule a position dependent code with as many elements as the number of variables of the system is used. Each element is a binary string with a bit per fuzzy set in the fuzzy partition of the variable, meaning the absence or presence of the corresponding linguistic label in the rule. For a three input and one output variable, with fuzzy partitions composed of 3,2,2 fuzzy sets for input variables and 3 fuzzy sets for output variable, the fuzzy rule will have a representation as shown in Figure 7.9.

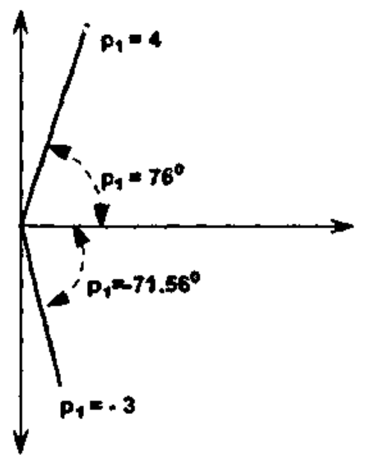


Figure 7.6. Angular coding technique of rule consequent parameters of Takagi Sugeno inference system

	MF_1				MF_n	
p_1	q_1	r_1	....	p_n	q_n	r_n

Figure 7.7. Chromosome representing *n m*embership functions for every input/output variable coding the parameters of a bell shape MF

1	2	3		m-2	m-1	m
0	1	1	....	0	1	0

Figure 7.8. Chromosome representing the entire rule base consisting of m fuzzy rules

input variables							output variable		
1	0	1	1	1	0	1	1	0	1

Figure 7.9. Chromosome representing an individual fuzzy rule (3 input variables and 1 output variable)

Layer 3. In this layer, a chromosome represents the different parameters of the T-norm and T-conorm operators. Real number representation is adequate to represent the fuzzy operator parameters. The parameters of the operators could be even fine- tuned using gradient descent techniques.

Layer 4. This layer is responsible for the selection of optimal learning parameters. Performance of the gradient descent algorithm directly depends on the learning rate according to the error surface. We used real number representation to represent the learning parameters. The optimal learning parameters decided by the evolutionary algorithm will be used to tune the membership functions and the inference mechanism.

Layer 5. This layer basically interacts with the environment and decides which fuzzy inference system (Mamdani type and its variants, Takagi-Sugeno type, Tsukamoto type etc.) will be the optimal according to the environment.

Once the chromosome representation, *C*, of the entire neuro-fuzzy model is done, the evolutionary search procedure could be initiated as follows:

7) *Generate an initial population of N numbers of C chromosomes. Evaluate the fitness of each EvoNF depending on the problem.*
8) *Depending on the fitness and using suitable selection methods reproduce a number of children for each individual in the current generation.*
9) *Apply genetic operators to each child individual generated above and obtain the next generation.*
10) *Check whether the current model has achieved the required error rate or the specified number of generations has been reached. Go to Step 2.*

End

7.5. Experimentation Setup Using EvoNF

We have applied the proposed technique to the three time series mentioned in Section 2.2. Fitness value is calculated based on the RMSE achieved on the test set. We have considered the best-evolved neuro-fuzzy model as the best individual of the last generation. We also explored three different learning methods:

Type 1: Evolutionary learning of membership functions, T-norm and T- conorm operators, rule base, consequent parameters and fine tuning of the membership functions using gradient descent method. The evolutionary algorithm further optimizes the learning rates of the gradient descent technique. This method could be considered as a meta learning approach.

Type 2: Evolutionary learning of membership functions, T-norm operator, rule base and consequent parameters. No gradient descent learning is used. This is equivalent to pure evolutionary design of fuzzy inference systems.

Type 3: Evolutionary learning of membership functions, rule base and consequent parameters with fixed T-norm (min) operator. The MFs are fine-tuned using gradient descent method and the evolutionary algorithm is used to further optimize the learning rates. This experiment is to demonstrate how important is the tuning of fuzzy operators.

We reduced the search space by incorporating the following priori knowledge

- Takagi-Sugeno fuzzy inference system was selected
- The initial rule base was generated using a grid partitioning method and the rule base was further optimized using the evolutionary algorithm. This approach seems to work faster than building up the rule base from scratch.
- only Gaussian and Bell shaped MF's was used.

The genotypes were represented by real coding using floating-point numbers and the initial populations were randomly created based on the parameters shown in Table 7.1. We used a special mutation operator, which decreases the mutation rate as the algorithm greedily proceeds in the search space. If the allelic value x_i of the i-th gene ranges over the domain a_i and b_i the mutated gene x_i' is drawn randomly uniformly from the interval $[a_i, b_i]$ [72].

$$x_i' = \begin{cases} x_i + \Delta(t, b_i - x_i), if\ \omega = 0 \\ x_i + \Delta(t, b_i - x_i), if\ \omega = 1 \end{cases} \qquad (7.4)$$

where ω represents an unbiased coin flip $p(\omega = 0) = p(\omega = 1) = 0.5$, and

$$\Delta(t,x) = x\left[1 - \gamma^{\left(1-\frac{t}{t_{max}}\right)^{b}}\right] \tag{7.5}$$

defines the mutation step, where γ is the random number from the interval $[0,1]$ and t is the current generation and t_{max} is the maximum number of generations. The function Δ computes a value in the range $[0,x]$ such that the probability of returning a number close to zero increases as the algorithm proceeds with the search. The parameter b determines the impact of time on the probability distribution Δ over $[0,x]$. Large values of b decrease the likelihood of large mutations in a small number of generations.

The parameters mentioned in Table 7.1 were decided after a few trial and error approaches. Experiments were repeated 3 times for the three time series and the worst performance measures are reported.

Table 7.2 summarizes the quantity of membership functions and the rule base before and after Type 1 learning. Test results showing the RMSE for the three time series are presented in Table 7.3. Figures 7.10 and 7.11 depicts of the different learning methods for gas furnace series. Learning convergence for waster water series and Mackey glass time series are plotted in Figures 7.12 - 7.15.

Table 7.1. Parameters used for evolutionary design of neuro-fuzzy systems

Population size	30
Fuzzy inference system	Takagi Sugeno
Rule antecedent membership functions Rule consequent parameters	2 - 4 membership functions per input variable parameterized Gaussian angular coding
T-norm operators	Parameterized Schweizer and Sklar's operator
Learning rate	0.05 – 0.20
Learning epochs	100 epochs of gradient descent algorithm for all the 3 time series
Ranked based selection	0.50
Elitism	5 %
Starting mutation rate	0.70

Iterations

Mackey Glass	Type 1	60	**Gas Furnace**	Type 1	60	**Waste Water**	Type 1	65
	Type 2	90		Type 2	135		Type 2	180
	Type 3	60		Type 3	60		Type 3	65

Table 7.2. Comparison of membership functions and fuzzy rules before and after learning (Type 1)

	Before learning						**After learning**					
	MFs					No. of rules	MFs					No. of rules
	Input				Output		Input				Output	
	I_1	I_2	I_3	I_4	O_1		I_1	I_2	I_3	I_4	O_1	
Mackey Glass	4	4	4	4	linear	256	3	3	4	3	linear	94
Gas Furnace	3	3	-	-	linear	9	3	4	-	-	linear	12
Waste Water	4	4	4	4	linear	256	4	3	4	3	linear	112

Table 7.3. Comparison of EvoNF, ANFIS and MLEANN

Time series	Learning algorithm	Neuro-fuzzy				MLEANN	
		EvoNF		ANFIS			
		RMSE		RMSE		RMSE	
		Training	Test	Training	Test	Training	Test
Mackey Glass	Type 1	0.0007	0.0008				
	Type 2	0.0009	0.0009	0.0019	0.0018	0.0004*	0.0004*
	Type 3	0.0009	0.0009				
Gas Furnace	Type 1	0.0093*	0.0110*				
	Type 2	0.0111	0.0154	0.0137	0.0570	0.0110	0.0210
	Type 3	0.0101	0.0256				
Waste Water	Type 1	0.0150*	0.0310*				
	Type 2	0.0190	0.0342	0.0530	0.0810	0.0425	0.0521
	Type 3	0.0180	0.0330				

**Lowest RMSE values*

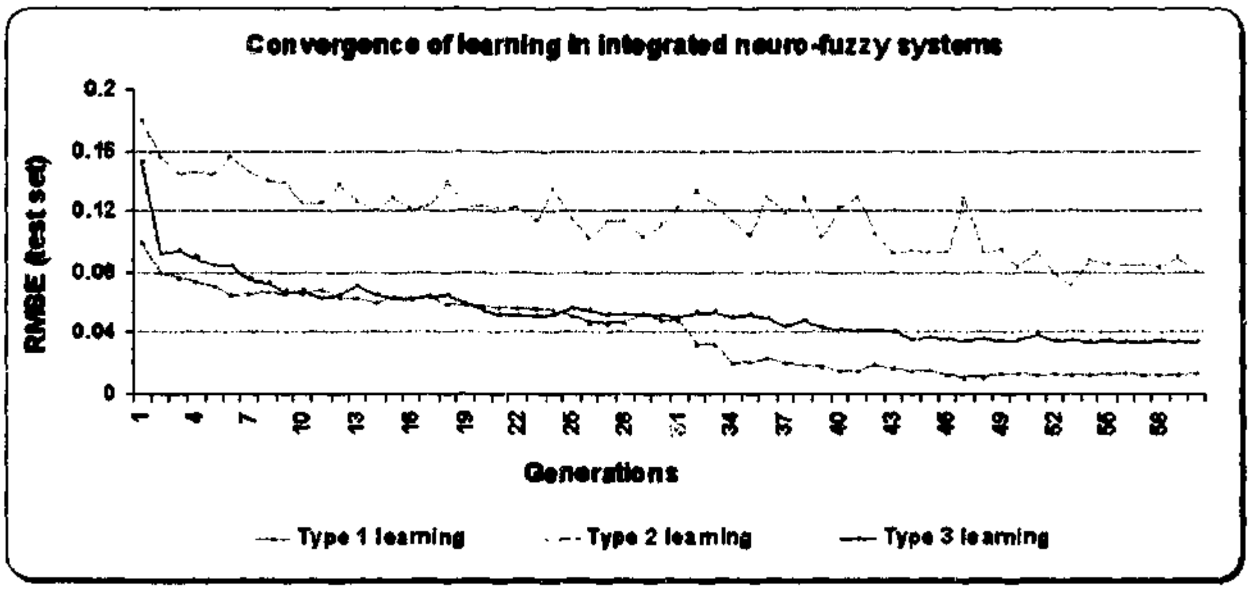


Figure 7.10. Gas furnace series: convergence of the different learning methods after 60 generations

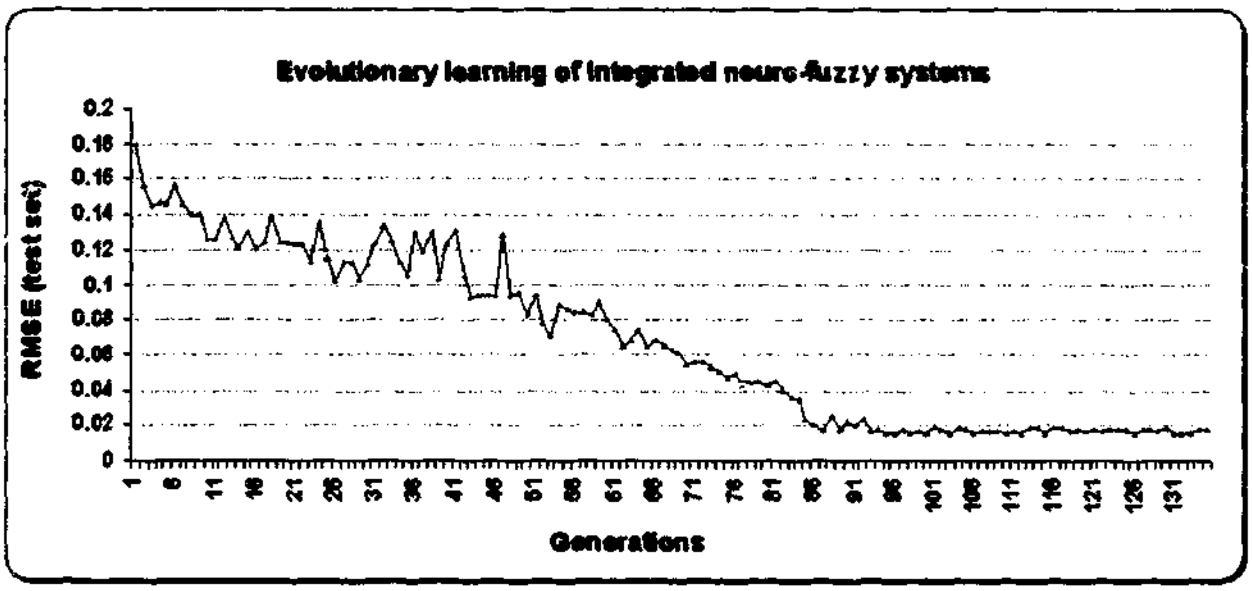


Figure 7.11. Gas furnace series: convergence of the evolutionary algorithm after 135 generations

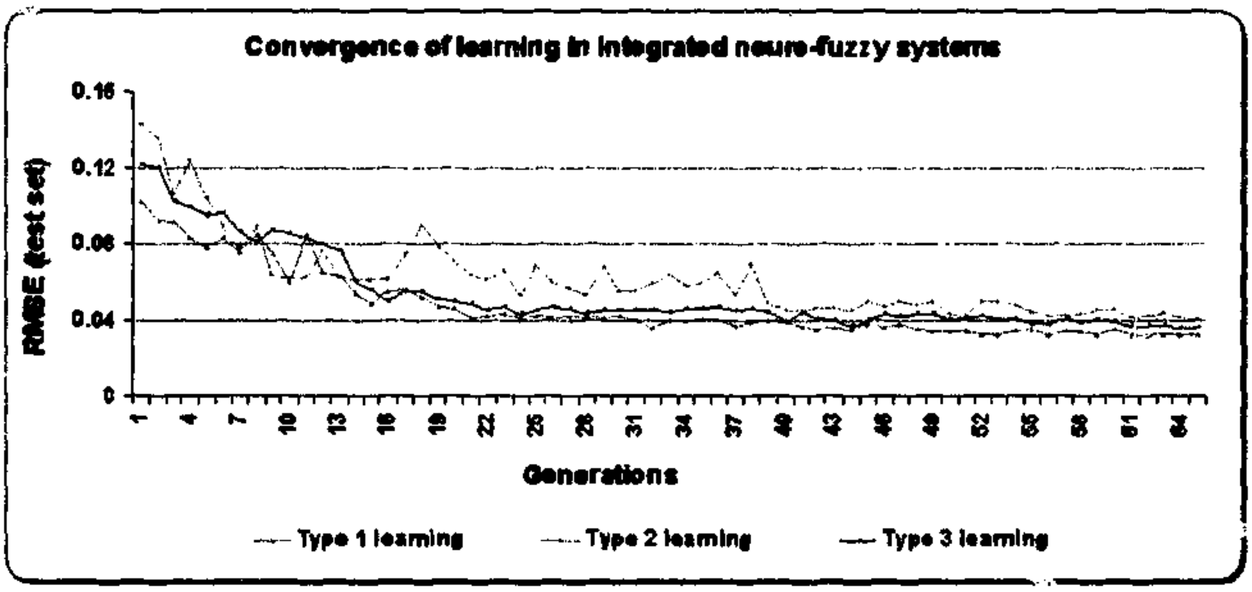


Figure 7.12. Waste water series: convergence of the different learning methods after 65 generations

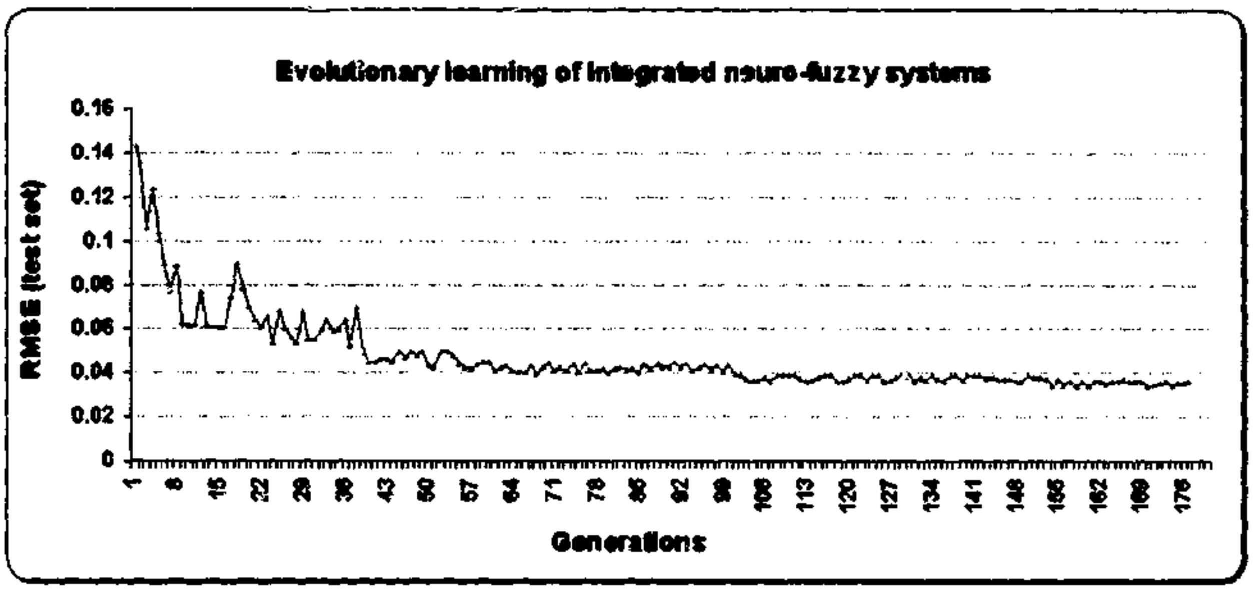


Figure 7.13. Waste water series: convergence of the evolutionary algorithm after 180 generations

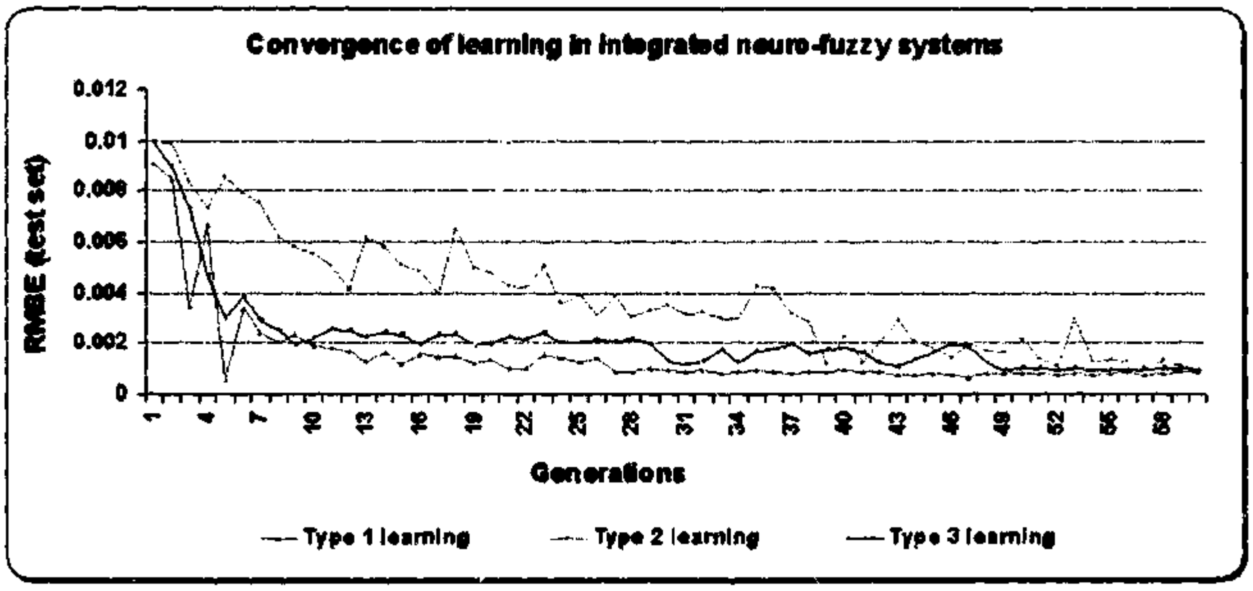


Figure 7.14. Mackey Glass series: convergence of the different learning methods after 60 generations

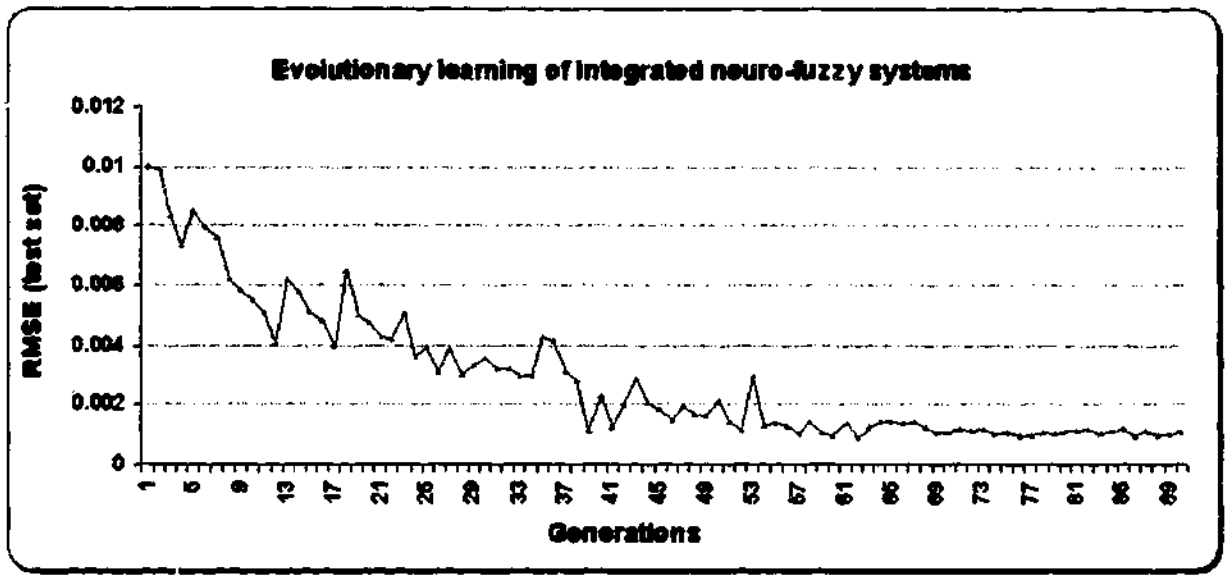


Figure 7.15. Mackey Glass series: convergence of the evolutionary algorithm after 90 generations

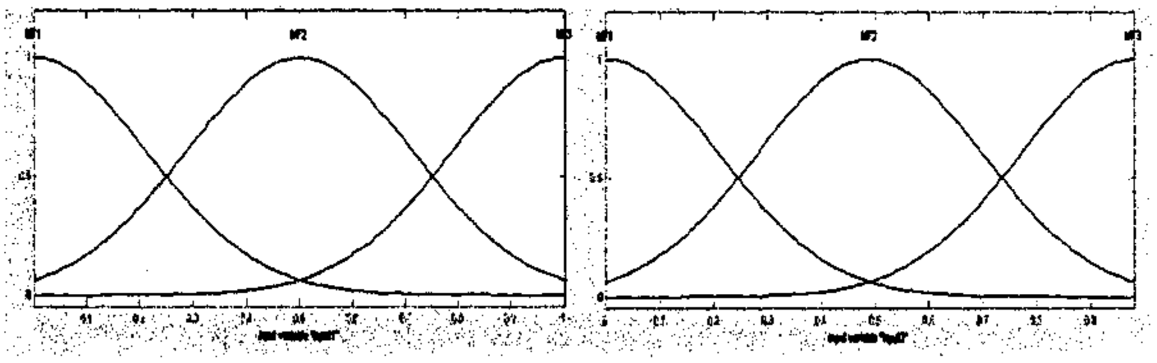


Figure 7.16. Gas furnace time series: membership functions for input variable 1 and 2 (before Type 1 learning)

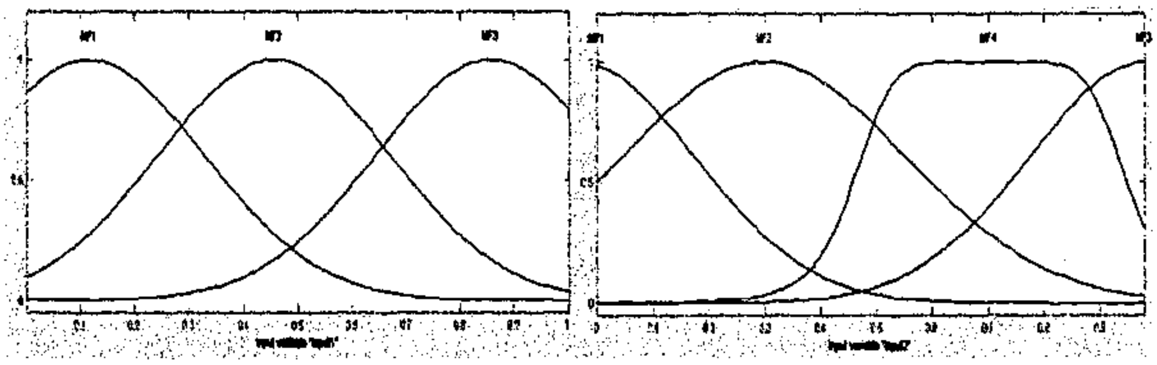


Figure 7.17. Gas furnace time series: membership functions for input variable 1 and 2 (after Type 1 learning)

7.6.Discussions and Conclusions

We have explored the three different learning mechanisms on an integrated neuro-fuzzy system implementing a Takagi-Sugeno fuzzy inference system. In the Type 1 learning an evolutionary learning method and gradient descent method was used to fine tune the performance. The fuzzy operators were also learned in the Type 1 method. Type 3 method is similar to Type 1 except the fuzzy operators. The fuzzy operators were fixed for the Type 3 learning. Type 2 learning does not use the gradient descent method and all the parameterized node functions are learned using evolutionary computation.

As evident from Table 7.3, in terms of RMSE error, the evolutionary design of neuro-fuzzy systems could outperform the conventional design of neuro-fuzzy systems using deterministic techniques. For interest empirical results of EvoNF was compared with ANFIS (Adaptive Neuro-Fuzzy Inference System) implementing a Takagi-Sugeno fuzzy inference system. For all the three time series considered, the evolutionary design approach gave the best results on training and test sets.

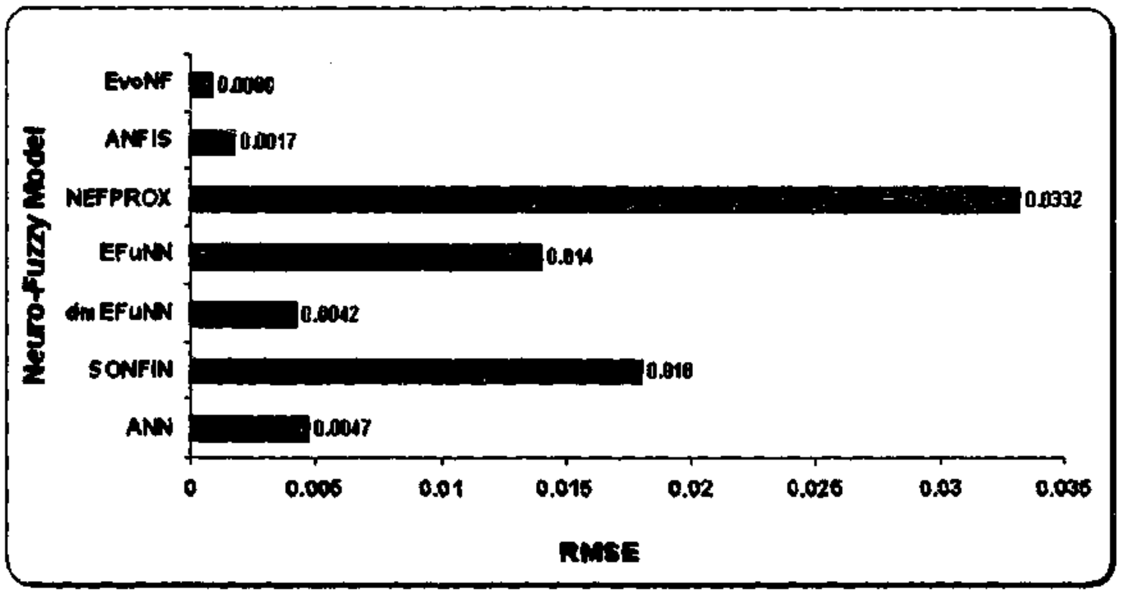


Figure 7.18. Comparison of EvoNF, popular neuro-fuzzy models and neural network

Our experiments using the three different learning strategies also reveal the importance of fine-tuning the global search method using a local search method (integrated learning). Type-2 learning method took longer time for the convergence and the final RMSE values obtained were higher than the integrated learning approach. The empirical results obtained from Type 1 and Type 3 learning clearly illustrates the role of fuzzy operator tuning. For the gas furnace series, the test results obtained using the Type 1 learning were almost 100% less than that of Type 3 learning.

We have also compared the empirical results of EvoNF model with MLEANN. While EvoNF could outperform MLEANN for gas furnace and waste water series, for Mackey glass series MLEANN performed marginally better. Perhaps the chaotic behavior of the Mackey Glass series could not be well represented using a Takagi Sugeno fuzzy inference system. Neuro-fuzzy performance could have been improved if we used a first order Takagi-Sugeno model or other learning methods. Figure 7.18 illustrates the comparison of EvoNF with different integrated neuro-fuzzy models and an artificial neural network trained using the backpropagation algorithm for predicting the Mackey Glass time series.

In this Chapter we have presented how the optimal design of integrated neuro-fuzzy systems could be achieved using a 5-tier evolutionary search process. However, the real success in modeling such systems will directly depend on the genotype representation of the different layers. All prior knowledge available about the problem domain / system design are to be encoded into the system to minimize the search space by the evolutionary algorithms.

Hierarchical evolutionary search processes attract considerable computational effort. Fortunately, evolutionary algorithms work with a population of independent solutions, which makes it easy to distribute the computational load among several processors using parallel algorithms. Hence, for complicated problems, parallel evolutionary algorithms might prove to be very useful [62]. As a guideline, for NF systems to be highly intelligent some of the major requirements are fast learning (memory based - efficient storage and retrieval capacities), on-line adaptability (accommodating new features like inputs, outputs, nodes, connections etc), achieve a global error rate and robust. The data acquisition and preprocessing training data is also quite important for the success of optimization of fuzzy inference systems.

Chapter 8: Conclusions

8.0. Introduction

The integration of different intelligent technologies is an active area of research in artificial intelligence. While James Bezdek [41] defines intelligent systems in a frame called computational intelligence, Lotfi Zadeh [255] explains the same using the soft computing framework. Integration issues range from the different types of techniques, theories of computation to problems of exactly how best to implement hybrid systems [12].

In this thesis, we explored the hybrid integration of neural networks - evolutionary algorithms, fuzzy systems - evolutionary algorithms, neural networks – fuzzy systems and neural networks - fuzzy systems - evolutionary algorithms. We have applied the different hybrid combinations to some practical applications and some popular chaotic time series, which has been widely used by several researchers working with connectionist models. In this Chapter, we summarize the main results obtained in the thesis. The scientific importance of the research work is pointed out on theoretical and application study parts. In addition, the topics for future research are discussed.

8.1. Main Results

For designing artificial neural networks, our experimentations using 4 different learning algorithms on three different chaotic time series clearly demonstrates that there is no best algorithm that will give optimal performance for all the problems. The initialization of weights, node transfer functions, architecture of the neural network, learning algorithm parameters etc play an equal role in optimization of the network performance. MLEANN framework is able to adapt the architecture (connectivity, number of neurons and node transfer functions), connection weights and learning algorithms and its parameters according to the problem.

Empirical results clearly demonstrate the efficiency of the proposed technique when compared to the best possible design using conventional approaches. The Evolutionary approach also shows superior performance when compared to a deterministic hybrid search technique using cutting angle method. In MLEANN, our work was mostly concentrated on the evolutionary search of optimal learning algorithms.

We have applied neuro-fuzzy systems to three practical applications and the empirical results clearly demonstrate the efficiency of the hybrid approach. Performance of neuro-fuzzy systems will depend on the careful selection of quantity and shape of membership functions for each input/out variable, an algorithm to determine the rule base and other network parameters etc. As demonstrated in Chapter 4, our various experiments also point out the need for adaptation of the various components (membership functions and parameters, knowledge base, fuzzy operators, inference system etc.) of a fuzzy inference system.

In Chapter 7, we presented the integration of neural networks, fuzzy inference systems and evolutionary algorithms. This could be viewed as an adaptive computational framework for automatic learning and optimization of fuzzy inference systems. In terms of RMSE error, the EvoNF could outperform the conventional design of neuro-fuzzy systems using deterministic techniques. For comparison purposes the empirical results of conventional design of neuro-fuzzy systems was compared with ANFIS (Adaptive Neuro-Fuzzy Inference System) implementing a Takagi-Sugeno fuzzy inference system. For all the three time series considered, the evolutionary design approach gave the best results on training and test sets. Experiment results also reveal the importance of evolutionary learning and fine-tuning by a local search method. Using a pure evolutionary learning method, the algorithm requires more iteration to converge. In spite of more iteration using a pure EA, very often the results are not as good as the integrated learning approach (type 1). Fuzzy operator tuning also plays a major role to optimize the performance of the integrated system.

Referring to the empirical results of the EvoNF approach, for Mackey glass series evolutionary neural networks performed marginally better. Perhaps the chaotic behavior of the Mackey Glass series could not be well represented using a Takagi Sugeno fuzzy inference system. Neuro-fuzzy performance might be improved if we used a higher order Takagi-

Sugeno model or more efficient learning methods. This also agrees with the equivalence between fuzzy inference systems and neural networks as defined by Hayashi et al [113].

8.2. Future Research Directions

We used a fixed chromosome structure (direct encoding technique) to represent the various layers in the connectionist models. As size of the chromosome grows, computational complexity also increases. Cellular configuration might be helpful to explore the representation more efficiently. For evolutionary neural networks, Gutierrez et al [107] has shown that their cellular automata technique performed better than direct coding. For the evolutionary search of architectures (layers/connectivity for neural networks and rule base for fuzzy inference system), it will be interesting to model as co-evolving [75] sub-networks instead of evolving the whole network. Further, it will be worthwhile to explore the whole population information of the final generation for deciding the best solution.

Hierarchical evolutionary search process attracts enormous computational complexity. Fortunately, evolutionary algorithms work with a population of independent solutions, which makes it easy to distribute the computational load among several processors. As we all know, the problems of the future will be more complicated in terms of complexity and data volume attracting more computational load. Hence, more research is to be diverted to design suitable message passing interfaces and implement the different hybrid algorithms in a parallel environment [18]. The design of parallel evolutionary algorithms involves choices such as using one population or multiple populations. In both cases, the size of population or populations must be determined carefully, and when multiple populations are used, one must decide how many to use. In addition, the populations may remain isolated or they may communicate by exchanging individuals. Communication involves extra costs and additional decision on topologies, on how many individuals are exchanged, and on the frequency of communications [62].

The use of fuzzy logic to translate and improve heuristic rules has also been applied to manage the resource of evolutionary algorithms (population size, selection pressure etc.) as the algorithm greedily explores and exploits the search space. The technique proposed by Lee

[156] to perform a run-time tuning of population size and reproduction operators based on the fitness measures might be helpful to improve the computational run-time efficiency of the evolutionary search process.

In this thesis, we have investigated only the hybrid combinations involving neural networks, fuzzy systems and evolutionary algorithms. It will be interesting to investigate hybrid combinations of other intelligent techniques like support vector machines [235], artificial immune systems [76], Bayesian methods [135], rough sets [163] and popular hard computing techniques like CART [56], MARS [92] etc.

References

[1] Abe A and Lan M S (1995), A Method for Fuzzy Rules Extraction Directly from Numerical Data and its Application to Pattern Classification, IEEE Transactions on Fuzzy Systems, 3(1): pp. 18-28.

[2] Abe S and Lan M S (1995), Fuzzy Rule Extraction Directly from Numerical Data for Function Approximation, IEEE Trans. Systems, Man and Cybernetics, 25: pp. 119-129.

[3] Abraham A and Nath B (1999), Artificial Neural Networks for Intelligent Real Time Power Quality Monitoring Systems, First International Power & Energy Conference, INT-PEC'99, CD ROM Proceeding, M Isreb (Editor), Australia.

[4] Abraham A and Nath B (1999), Failure Prediction of Critical Electronic Systems in Power Plants Using Artificial Neural Networks, In Proceedings of First International Power and Energy Conference, Isreb M (Editor), Australia.

[5] Abraham A and Nath B (2000), Optimal Design of Neural Nets Using Hybrid Algorithms", In proceedings of 6th Pacific Rim International Conference on Artificial Intelligence (PRICAI 2000), pp. 510-520.

[6] Abraham A and Nath B (2000), A Soft Computing Approach for Fault Prediction of Electronic Systems, In Proceedings of The Second International Conference on Computers in Industry, ICCI 2000, Published by The Bahrain Society of Engineers, Majeed A Karim et al (Eds), Bahrain, pp. 83-91.

[7] Abraham A and Nath B (2000), An Evolving Fuzzy Neural Network Model Based Reactive Power Control, In Proceedings of the Second International Conference on Computers in Industry, ICCI 200, Published by the Bahrain Society of Engineers, Majeed A Karim et al (Eds), Bahrain, pp. 247-253.

[8] Abraham A and Nath B (2000), Connectionist Models for Intelligent Reactive Power Control, In Proceedings of The Australasian MATLAB Users Conference 2000, Published by Ceanet, Australia (Proceedings Online), Melbourne, Australia.

[9] Abraham A and Nath B (2000), Designing Optimal Neuro-Fuzzy Systems for Intelligent Control, In proceedings of the Sixth International Conference on Control Automation Robotics Computer Vision (ICARCV 2000), Singapore.

[10] Abraham A and Nath B (2000), Evolutionary Design of Fuzzy Control Systems - An Hybrid Approach, The Sixth International Conference on Control, Automation, Robotics and Vision, (ICARCV 2000), Singapore.

[11] Abraham A and Nath B (2000), Evolutionary Design of Neuro-Fuzzy Systems - A Generic Framework, In Proceedings of The 4-th Japan-Australia Joint Workshop on Intelligent and Evolutionary Systems, Published by the National Defence Academy (Japan) and University of New South Wales (Australia) , Akira Namatame et al (Editors), ISBN 07317 0504 1, Japan, pp. 106-113.

[12] Abraham A and Nath B (2000), Hybrid Intelligent Systems: A Review of a Decade of Research, Gippsland School of Computing and Information Technology, Monash University, Technical Report Series (5/2000), pp 1-54.

[13] Abraham A and Nath B (2000), IT Impact On New Millennium Manufacturing, In Proceedings of 5th International Conference on Computer Integrated Manufacturing, Computer Integrated Manufacturing, Singh J , Lew S C and Gay R (Editors),Singapore, pp. 321-332.

[14] Abraham A and Nath B (2001), A Neuro-Fuzzy Approach for Forecasting Electricity Demand in Victoria, Applied Soft Computing Journal, Elsevier Science, Volume 1 (2), pp.127-138.

[15] Abraham A and Nath B (2001), ALEC -An Adaptive Learning Framework for Optimizing Artificial Neural Networks, Computational Science, Springer-Verlag Germany, Vassil N Alexandrov et al (Editors), San Francisco, USA, pp. 171-180.

[16] Abraham A and Nath B (2001), Meta-Learning Evolutionary Artificial Neural Networks, Elsevier Science, Neuro Computing Journal (under review).

[17] Abraham A and Steinberg D (2001), MARS: Still an Alien Planet in Soft Computing?, Computational Science, Springer-Verlag Germany, Vassil N Alexandrov et al (Editors), ISBN 3-540-42233-1, San Francisco, USA, pp. 235-244.

[18] Abraham A, Buyya R and Nath B (2000), Nature's Heuristics for Scheduling Jobs in Computational Grids, In Proceedings of 8th IEEE International Conference on Advanced Computing and Communications, Sinha P S and Gupta R (Eds), Tata McGraw-Hill Publishing Co. Ltd, New Delhi, India, pp. 45-52.

[19] Abraham A, Nath B and Mahanti P K (2001), Hybrid Intelligent Systems for Stock Market Analysis, Computational Science, Springer-Verlag Germany, Vassil N Alexandrov et al (Editors), San Francisco, pp. 337-345.

[20] Abraham A, Nath B and Saratchandran P (2001), Soft Computing and Computer Aided Reliability Engineering - A Synergism for Online Monitoring of Electronic Systems, Applied Soft Computing Journal, Elsevier Science (under review).

[21] Abraham A, Philip N S and Joseph K B (2001), Will We Have a Wet Summer? Soft Computing Models for Long Term Rainfall Forecasting, 15th European Simulation Multiconference (ESM 2001), Modeling and Simulation 2001, Kerckhoffs E J H and Snorek M (Eds), Prague, Czech Republic, pp. 1044-1048.

[22] Abraham A (2001), Neuro-Fuzzy Systems: State-of-the-Art Modeling Techniques, Connectionist Models of Neurons, Learning Processes, and Artificial Intelligence, Springer-Verlag Germany, Jose Mira and Alberto Prieto (Eds.), Granada, Spain, pp. 269-276.

[23] Akbarzadeh M. R., Kumbla K. K., and Jamshidi M. (1995), Genetic algorithms in learning fuzzy hierarchical control of distributed parameter systems. In Proceedings of the IEEE Conference on Systems, Man and Cybernetics, Volume 5, pp. 4027-4032.

[24] Alamo T., Gordillo F., and Aracil J. (1995), Robust fuzzy control using genetic algorithms. In Proceedings of the Third European Conference on Intelligent Techniques and Soft Computing (EUFIT'95), pp. 781-785.

[25] Alba E, Cotta C., and Troya J. M. (1996), Type-Constrained Genetic Programming for rule-base definition in fuzzy logic controllers. In Proceedings of GP'96 Conference, pp. 255-260.

[26] Aliev R A and Aliev R R (2001), Soft Computing and its Applications, World Scientific Publishing CO, Singapore.

[27] Andlinger P and Reichl E R (1991), Fuzzy-Neunet: A Non Standard Neural Network, In Prieto et al., pp. 173-180.

[28] Angeline P J, Saunders G B and Pollack J B (1994), An Evolutionary Algorithm that Evolves Recurrent Neural Networks, IEEE Transactions on Neural Networks, Vol 5:1, pp. 54-65.

[29] Arao M, Tsutsumi, Fukuda T and Shimokima K (1995), Flexible Intelligent System based on Fuzzy Neural Networks and Reinforcement Learning, In proceedings of IEEE International Conference on Fuzzy Systems, Vol. 5(1), pp 69-70.

[30] Auer P, Herbster M and Warmuth M (1996), Exponentially Many Local Minima for Single Neurons, D Touretzky et al (Eds.), Advances in Neural Information Processing Systems, MIT Press, Cambridge, MA, Vol 8, pp. 316-322.

[31] Baffles P T and Zelle J M (1992), Growing layers of Perceptrons: Introducing the Exentron Algorithm, Proceedings on the International Joint Conference on Neural Networks, Vol 2, pp. 392-397.

[32] Balakrishnan K and Honawar V (1996), Some Experiments in Evolutionary Synthesis of Robotic Neurocontrollers. Proceedings of World Congress on Neural Networks. pp 1035-1040.

[33] Bastian A. (1996), A genetic algorithm for tuning membership functions, In Proceedings of the Fourth European Conference on Intelligent Techniques and Soft Computing (EUFIT'96), pp. 494-498.

[34] Baxter J (1992), The evolution of learning algorithms for artificial neural networks, Complex systems, IOS press, Amsterdam, pp. 313-326.

[35] Beliakov G and Abraham A (2002), Global Optimization of Neural Networks Using a Deterministic Hybrid Approach, Proceedings of the First International Workshop on Hybrid Intelligent Systems, Adelaide, Australia, Springer-Verlag Germany, pp. 79-92. (in press).

[36] Benitez J J, Castro J L and Requena I (1997), Are Artificial Neural Networks Black Boxes?, IEEE Transactions on Neural Networks, Volume 8, pp. 1156-1164.

[37] Berenji H R (1992), A Reinforcement Learning-Based Architecture for Fuzzy Logic Control, International. Journal of Approximate Reasoning, 6: pp. 267 –292, 1992.

[38] Berenji H R and Khedkar P (1992), Fuzzy Rules for Guiding Reinforcement Learning, In International. Conference. on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'92), pp. 511-514.

[39] Berenji H R and Khedkar P (1992), Learning and Tuning Fuzzy Logic Controllers through Reinforcements, IEEE Transactions on Neural Networks, Vol (3), pp. 724-740.

[40] Bersini H, Nordvik J P and Bonarini A (1993), A Simple Direct Adaptive Fuzzy Controller Derived from its Neural Equivalent, In Proceedings IEEE International. Conference on Fuzzy Systems, pp. 345-350.

[41] Bezdek J C (1996), Computational Intelligence Defined – By Everyone!, Computational Intelligence: Soft Computing and Fuzzy-Neuro Integration with Applications, Okay Kaynal et al (Eds), Springer Verlag, Germany.

[42] Bezdek J C and Pal S K (1992), Fuzzy Models for Pattern Recognition, IEEE Press, New York.

[43] Bishop C M (1995), Neural Networks for Pattern Recognition, Oxford Press.

[44] Boers E J W, Borst M V and Sprinkhuizen-Kuyper I G (1995), Artificial Neural Nets and Genetic Algorithms, DW Pearson et al, (Eds.); Proceedings of the International Conference in Ales, France, Springer Verlag, NY, , pp. 333-336.

[45] Boers E J W, Borst M V and Sprinkhuizen-Kuyper I G (1995); Evolving artificial neural networks using the Baldwin effect, In D.W. Pearson et al (Eds.), Artificial Neural

Nets and Genetic Algorithms, Proceedings of the International Conference in Alès, France, pp. 333-336, Springer-Verlag, New York.

[46] Boers E J W, Kuiper H, Happel B L M, and Sprinkhuizen-Kuyper I G (1993); Designing modular artificial neural networks, In: H.A. Wijshoff (Ed.); Proceedings of Computing Science in The Netherlands, pp. 87-96.

[47] Bonarini A. (July 1996), Fuzzy sets and evolutionary learning: Motivations and issues for integrated systems. In Proc. Thirteenth International Conference on Machine Learning (ICML'96), pages 30-37.

[48] Bonissone P. P., Khedkar P. S., and Chen Y. (1996), Genetic algorithms for automated tuning of fuzzy controllers: A train handling application. In Proc. of the Fifth IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'96), Volume 1, pp. 674-680.

[49] Booker L B, Goldberg D E and Holland J H (1989), Classifier Systems and Genetic Algorithms, Artificial Intelligence, Volume 40, pp. 235-282.

[50] Bourlard H.A and Morgan N. (1994), Connectionist Speech Recognition: A Hybrid Approach, Boston, Kluwer Academic Publishers.

[51] Box G E P and Jenkins G M (1970), Time Series Analysis, Forecasting and Control, San Francisco: Holden Day.

[52] Branke J, Kohlmorgen U and Schmeck H (1995), A Distributed Genetic Algorithm Improving the Generalization Behavior of Neural Networks, Proceedings of the European Conference on Machine Learning, N Lavrac et al (eds.), pp. 107-112.

[53] Braun H (1995), On Optimizing Large Neural Networks (Multilayer Perceptrons) by Learning and Evolution, in Proceedings of the Third International Congress on Industrial and Applied Mathematics, ICIAM.

[54] Braun H and Weisbrod J (1993), Evolving Neural Networks for Application Oriented Problems, in D.B Fogel (Ed.), Proceedings of the second Conference on Evolutionary Programming, USA.

[55] Braun H and Zagorski P (1994), ENZO-M - a Hybrid Approach for Optimizing Neural Networks by Evolution and Learning, in Y Davidor et al (Eds.), Proceedings of the third Int. Conference on Parallel Problem Solving from Nature, Israel.

[56] Breiman, L., Friedman, J., Olshen, R., and Stone, C.J. (1984). Classification and Regression Trees, Chapman and Hall, New York.

[57] Brown M, Bossley K and Mills D (1995), High Dimensional Neurofuzzy Systems: Overcoming the Course of Dimensionality, In Proceedings. IEEE International. Conference on Fuzzy Systems, pp. 2139-2146.

[58] Buckley J J and Feuring T (1999), Fuzzy and Neural: Interactions and Applications, Studies in Fuzziness and Soft Computing, Heidelberg, Germany, Physica Verlag.

[59] Buckley J J and Hayashi Y (1993), Hybrid neural nets can be fuzzy controllers and fuzzy expert systems, Fuzzy Sets and Systems, 60: pp. 135-142.

[60] Buckley J J and Hayashi Y (1995), Neural Networks for Fuzzy Systems, Fuzzy Sets and Systems, 71: pp. 265-276.

[61] Bunke H and Kandel A, (2000), Neuro-Fuzzy Pattern Recognition, World Scientific Publishing CO, Singapore.

[62] Cantu-Paz E (1999), Designing Efficient and Accurate Parallel Genetic Algorithms, Technical Report No. 99017, Illinois Genetic Algorithms Laboratory, UIUC, USA.

[63] Carpenter G A, Grossberg S, Markuzon N, Reynolds J H, and D. B. Rosen (1992), Fuzzy ARTMAP: A Neural Network Architecture for Incremental Supervised Learning of Analog Multidimensional Maps, IEEE Transactions Neural Networks, 3(5), pp 698-712.

[64] Castillo P A, Merelo J J, Prieto A, Rivas V and Romero G (2000), G-Prop: Global optimization of multilayer perceptrons using GAs, Neurocomputing, 35, pp. 149-163.

[65] Castro J. L., Herrera F., and Delgado M. (1993), A learning method of fuzzy reasoning by genetic algorithms, In Proceedings of the First European Congress on Fuzzy and Intelligent Technologies (EUFIT'93), volume 2, pp. 804-809.

[66] Chalmers D J, The Evolution of Learning: An Experiment in Genetic Connectionism", In Touretzky D S et al (Eds), Proceedings of the 1990 Connectionist Models Summer School, Morgan Kaufmann, CA, pp. 81-90.

[67] Cherkassky V (1998), Fuzzy Inference Systems: A Critical Review, Computational Intelligence: Soft Computing and Fuzzy-Neuro Integration with Applications, Kayak O, Zadeh LA et al (Eds.), Springer, pp.177-197.

[68] Chong, E. K.P. and Zak, S.H. (1996). An Introduction to Optimization. John Wiley and Sons, Inc. New York.

[69] Chowdhury A and Mhasawade S V (1991), Variations in Meteorological Floods during Summer Monsoon Over India, Mausam, 42, 2, pp. 167-170,

[70] Cichocki A. and Unbehauen R. (1993), Neural Networks for Optimization and Signal Processing. NY: John Wiley & Sons

[71] Cordon O and Herrera F (1997), Evolutionary Design of TSK Fuzzy Rule Based Systems Using (μ , λ) Evolution Strategies, In Proceedings of the Sixth IEEE International Conference on Fuzzy Systems, Spain, Volume 1, pp. 509-514.

[72] Cordón O Herrera F Hoffmann F and Magdalena L (2001), Genetic Fuzzy Systems: Evolutionary Tuning and Learning of Fuzzy Knowledge Bases, World Scientific Publishing Company, Singapore.

[73] Cordón O, Herrera F Peregrín A (2000), Searching for Basic Properties Obtaining Robust Implication Operators in Fuzzy Control. Fuzzy Sets and Systems, Vol 111, pp. 237-251.

[74] Czogala E and Leski J (2000), Fuzzy and Neuro-Fuzzy Intelligent Systems, (Studies in Fuzziness and Soft Computing), Springer Verlag, Germany.

[75] Darwen P. J (1996), Co-evolutionary Learning by Automatic Modularisation with Speciation' PhD Thesis, University of New South Wales.

[76] Dasgupta D (Ed) (1999), Artificial Immune Systems and Their Applications, Springer Verlag, Berlin.

[77] De Souza Flavio, Vellasco M M R, Pacheco M A C (2001), The Hierarchical Neuro-Fuzzy BSP Model: An Application in Electric Load Forecasting, Connectionist Models of Neurons, Learning Processes and Artificial Intelligence, Jose Mira et al (Editors), LNCS 2084, Springer Verlag Germany.

[78] Duch W and Korczak J (1999), Optimization and global minimization methods suitable for neural networks, Neural Computing Surveys.

[79] Dumitrescu D, Lazzerini B and Jain L C (Eds) (2001), Fuzzy Sets and their Application to Clustering and Training, CRC Press, USA.

[80] Eppler W (1990), Pre-structuring of neural networks with fuzzy rules, In Neuro-Nimes'90. 3rd International Workshop on Neural Networks and their Applications, pp 227-241.

[81] Esogbue A O (1993), A Fuzzy Adaptive Controller Using Reinforcement Learning Neural Networks. In Proceedings. IEEE International. Conference on Fuzzy Systems, pp. 178-183.

[82] Fahlman S E and Lebiere C (1990), The Cascade – Correlation Learning architecture, Advances in Neural Information Processing Systems, D.Tourretzky (Ed.), Morgan Kaufmann, pp. 524-532.

[83] Feng J C and Teng L C (1998), An Online Self Constructing Neural Fuzzy Inference Network and its Applications, IEEE Transactions on Fuzzy Systems, Vol 6, No.1, pp. 12-32.

[84] Fine T L (1999), Feedforward Neural Network Methodology, Springer Verlag, New York.

[85] Fogel D B (1995), Evolutionary Computation: Towards a New Philosophy of Machine Intelligence, 2nd Edition, IEEE Press.

[86] Fogel D B (1997), The Advantages of Evolutionary Computation, Bio-Computing and Emergent Computation, D. Lundh, B. Olsson, and A. Narayanan (Eds.), Sköve, Sweden, World Scientific Press, Singapore, pp. 1-11.

[87] Fogel D B (2001), Blondie24: Playing at the Edge of AI, Morgan Kaufmann Publishers, USA.

[88] Fontanari J F and Meir R (1991), Evolving a learning algorithm for the binary perceptron, Network, vol.2, pp. 353-359.

[89] Forti M (1996), A Note on Neural Networks With Multiple Equilibrium Points, IEEE Transactions on Circuits and Systems-I: Fundamental Theory, 43, pp. 487 (5).

[90] Frean M (1990), The upstart algorithm: a method for constructing and training feed forward neural networks, Neural computations, Volume 2, pp.198-209.

[91] Freisleben B and Kunkelmann T (1993), Combining Fuzzy Logic and Neural Networks to Control an Autonomous Vehicle, In Proceedings IEEE International. Conference on Fuzzy Systems, pp 321-326.

[92] Friedman, J. H. (1991), Multivariate Adaptive Regression Splines, Annals of Statistics, 19, pp. 1-141.

[93] Fukuda T and Shibata M (1994), Fuzzy-Neuro-GA Based Intelligent Robotics, In Zurada J M et al (Eds), Computational Intelligence Imitating Life, IEEE Press, pp. 352-362.

[94] Fuller R, Introduction to Neuro-Fuzzy Systems (2000), (Studies in Fuzziness and Soft Computing), Springer Verlag, Germany.

[95] Fullmer B and Miikkulainen R (1992), Using Marker-Based Genetic Encoding of Neural Networks To Evolve Finite-State Behaviour, FJ Varela and P Bourgine (Eds), Proceedings of the First European Conference on Artificial Life, France), pp.255-262.

[96] Funabiki N, Kitamichi J and Nishikawa S (1998), An evolutionary Neural Network Approach for Module Orientation Problems, IEEE transactions on Systems, Man, And Cybernetics- Part B: Cybernetics, Vol.28, No.6, pp.849-855.

[97] Furuhashi T (1997), Development of If-Then Rules with the Use of DNA Coding, Fuzzy Evolutionary Computation, Pedrycz W (Ed.), Kluwer Academic Publishers, pp.107-125.

[98] Furuhashi T., Miyata Y., and Uchikawa Y. (1996), Pseudo-bacterial genetic algorithm and finding of fuzzy rules. In Proceedings of the Second Online Workshop on Evolutionary Computation (WEC2), pp. 65-68.

[99] Glorennec P Y (1990), Associating a Neural Network and Fuzzy Rules for Dynamic Process Control, 3rd Int. Workshop on Neural Networks and their Applications Neuro-Nimes'90, pp. 211-225.

[100] Gonzalez A and Herrera F (1997), Multi-Stage Genetic Fuzzy Systems Based on the Iterative Rule Learning Approach, Mathware and Soft Computing Vol 4, pp. 233-249.

[101] Goode P and Chow M Y (1994), A Hybrid Fuzzy/Neural System Used to Extract Heuristic Knowledge from a Fault Detection Problem, In Proceedings. IEEE International. Conference. on Fuzzy Systems, pp. 1731-1736.

[102] Grau F (1992), Genetic Synthesis of Boolean Neural Networks with a Cell Rewriting Developmental Process, In D Whitely and Schaffer J D., Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks, IEEE Computer Society Press, CA, pp. 55.74.

[103] Grefenstette J J (Ed) (1994), Genetic Algorithms for Machine Learning, Kluwer Academic Publishers, USA.

[104] Gupta M M (1992), Fuzzy Neural Computing Systems, In Proceedings of SPIE, Volume 1710, Science of Artificial Neural Networks, Volume 2, pp. 489-499.

[105] Gupta M M and Rao D H (1994), On the Principles of Fuzzy Neural Networks, Fuzzy Sets and Systems, 61: pp.1-18.

[106] Guptha M M (1991), Theory of T-norms and Fuzzy Inference Methods, Fuzzy Sets and Systems, Volume 40, PP. 431-450.

[107] Gutierrez G, Isasi P, Molina J M, Sanchis A and Galvan I M (2001), Evolutionary Cellular Configurations for Designing Feedforward Neural Network Architectures, Connectionist Models of Neurons, Learning Processes, and Artificial Intelligence, Jose Mira et al (Eds), Springer Verlag - Germany, LNCS 2084, pp. 514-521.

[108] Halgamuge S K and Glesner M (1992), The Fuzzy Neural Approach for Pattern Classification with the Generation of Rules Based on Supervised Learning, In Proceedings of Neuro-Nimes 92, pp. 167-173.

[109] Halgamuge S K, Glesner M (1994), Neural Networks in Designing Fuzzy Systems for Real World Applications, Fuzzy Sets and Systems, 65: pp. 1-12.

[110] Halgamuge SK, Mari A and Glesner M (1994), Fast Perceptron Learning by Fuzzy Controlled Dynamic Adaption of Network Parameters, Kruse. R et al (Eds.) Fuzzy Systems in Computer Science, pp 129-139.

[111] Harp S A, Samad T and Guha A (1989), "Towards the Genetic Synthesis of Neural Networks", In Schaffer J D (Editor), Proceedings of the Third International Conference on Genetic Algorithms and their Applications, Morgan Kaufmann, CA, pp. 360-369.

[112] Hayashi I, Nomura H, Yamasaki H and Wakami N (1992), Construction of Fuzzy Inference Rules by Neural Network Driven Fuzzy Reasoning and Neural Network Driven Fuzzy Reasoning With Learning Functions, International. Journal of Approximate Reasoning, 6: pp. 241-266.

[113] Hayashi Y and Buckley J J (1994), Approximations Between Fuzzy Expert Systems and Neural Networks, International Journal of Approximate Reasoning, Volume 10, pp.63-73.

[114] Hayashi Y and Imura A (1990), Fuzzy Neural Expert System with Automated Extraction of Fuzzy If-Then Rules from a Trained Neural Network, In First International. Symposium on Uncertainty Modeling and Analysis, pp 489-494.

[115] Herrera F., Lozano M and Veregay L (1995), Tuning Fuzzy Logic Control by Genetic Algorithms, International Journal of Approximate Reasoning, 12(3/4), pp. 299-315.

[116] Hoffmann F (1999), The Role of Fuzzy Logic in Evolutionary Robotics, Fuzzy Logic Techniques for Autonomous Vehicle Navigation, A. Saffiotti and D. Driankov (Ed.), Springer-Verlag.

[117] Hoffmann F (2000), Soft computing techniques for the design of mobile robot behaviors, Journal of Information Sciences, 122 (2-4) pp. 241-258.

[118] Hoffmann F and Pfister G (1994), Automatic Design of Hierarchical Fuzzy Controllers Using Genetic Algorithms, 2nd European Congress on Intelligent Techniques and Soft Computi g.

[119] Hofgen K U (1993), Computational limits on Training Sigmoidal Neural Networks, information Processing Letters, Volume 46, pp. 269-274.

[120] Hollatz J (1995), Neuro-Fuzzy in Legal Reasoning, In Proceedings. IEEE International. Conference on Fuzzy Systems, pp. 655-662.

[121] Höppner F, Klawonn F, and Kruse R (1996), Fuzzy-Clusteranalyse, Computational Intelligence, Vieweg, Braunschweig.

[122] Ichihashi H and Turksen I (1995), Neuro-Fuzzy Data Analysis and Its Future Directions, In Proceedings. IEEE International Conference on Fuzzy Systems, pp 1919-1925.

[123] Ishibuchi H, Okada H and Tanaka H (1992), Interpolation of fuzzy of fuzzy if-then rules by neural networks. In Proceedings, 2nd Int. Conf. on Fuzzy Logic and Neural Networks, pp. 337-340.

[124] Jacobsen H A (1998), A Generic Architecture for Hybrid Intelligent Systems, In Proceedings of the IEEE Conference on Fuzzy Systems.

[125] Jacobsen H A and Iordanova I (1994), Approach to Extraction of Fuzzy Production Rules from a Connectionist Component of a Hybrid Expert System, Proceedings of the 6th International Conference on AI: Methodology, Systems and Applications, Bulgaria, September 1994.

[126] Jager R (1995), Fuzzy Logic in Control, PhD Thesis, Technische Universiteit Delft, Netherlands.

[127] Jain L.C. and Jain R.K. (Eds) (1997), Hybrid Intelligent Engineering Systems, World Scientific Publishing Company, Singapore.

[128] Jain L.C. and Martin N.M. (Eds) (1999), Fusion of Neural Networks, Fuzzy Logic and Evolutionary Computing and their Applications, CRC Press USA.

[129] Jain L.C., Johnson R.P., Takefuji Y., and Zadeh L.A. (Eds) (1998), Knowledge-Based Intelligent Techniques in Industry, CRC Press USA, 1998.

[130] Jang J S R (1991), ANFIS: Adaptive Network Based Fuzzy Inference Systems, IEEE Transactions Systems, Man and Cybernetics.

[131] Jang J S R, Sun C T and Mizutani E (1997), Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence, Prentice Hall Inc, USA.

[132] Jang R (1992), Neuro-Fuzzy Modeling: Architectures, Analyses and Applications, PhD Thesis, University of California, Berkeley.

[133] Jones L (1997), The Computational Intractability of Training Sigmoidal Neural Networks, IEEE Transactions on Information Theory, Volume 43, pp 143-173.

[134] Judd S (1990), Neural Network Design and the Complexity of Learning, MIT Press, Cambridge, MA.

[135] Judea P (1997), Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann Publishers, USA.

[136] Kacprzyk J. (1996), Genetic algorithms in multistage fuzzy control. In Herrera F. and Verdegay J. (eds) Genetic Algorithms and Soft Computing, Physica Verlag, pp. 579-598.

[137] Kandel A, Zhang QY and Bunke H (1997), A genetic Fuzzy Neural Network for Pattern Recognition, In IEEE Transactions on Fuzzy Systems, pp. 75-78.

[138] Kasabov N (1996), Foundations of Neural Networks, Fuzzy Systems and Knowledge Engineering, The MIT Press.

[139] Kasabov N (1998), Evolving Fuzzy Neural Networks - Algorithms, Applications and Biological Motivation, In Yamakawa T and Matsumoto G (Eds), Methodologies for the Conception, Design and Application of Soft Computing, World Scientific, pp. 271-274.

[140] Kasabov N (1998), The ECOS Framework and the 'ECO' Training Method for Evolving Connectionist Systems, Journal of Advanced Computational Intelligence, Vol.2, No.6, pp.1-8.

[141] Kasabov N (1999), Evolving Connectionist and Fuzzy Connectionist Systems –Theory and Applications for Adaptive On-line Intelligent Systems, In: Neuro-Fuzzy Techniques for Intelligent Information Processing, Kasabov N and Kozma R, (Eds.), Physica Verlag, 1999.

[142] Kasabov N and Qun Song (1999), Dynamic Evolving Fuzzy Neural Networks with m-out-of-n Activation Nodes for On-line Adaptive Systems, Technical Report TR99/04, Department of information science, University of Otago, New Zealand.

[143] Kasabov N and Robert Kozma (Eds) (1999), Neuro-Fuzzy Techniques for Intelligent Information Systems, (Studies in Fuzziness and Soft Computing), Springer Verlag, Germany.

[144] Kasabov N K (1995), Hybrid Fuzzy Connectionist Rule-based Systems and the Role of Fuzzy Rules Extraction, In Proceedings. IEEE International. Conference on Fuzzy Systems, pp 49-56.

[145] Kasabov, N. and Woodford B (1999), Rule Insertion and Rule Extraction from Evolving Fuzzy Neural Networks: Algorithms and Applications for Building Adaptive, Intelligent Expert Systems, In Proceedings of the FUZZ-IEEE'99 International Conference on Fuzzy Systems, Seoul, Korea, pp. 1406-1411.

[146] Kaufmann A (1975), Introduction to the Theory of Fuzzy Subsets, New York, Academic Press.

[147] Kaynak O (1998), Computational Intelligence: Soft Computing and Fuzzy-Neuro Integration with Applications, Springer Verlag, Germany.

[148] Keller M, Yager R R and Tahani H (1992), Neural Network Implementation of Fuzzy Logic. Fuzzy Sets and Systems, 45: pp.1-12.

[149] Khan E and Venkatapuram P (1993), Neufuz: Neural Network Based Fuzzy Logic Design Algorithms, In Proceedings IEEE International Conference on Fuzzy Systems, pp. 647-654.

[150] Kim H B, Jung S H, Kim T G and Park K H (1996), Fast learning method for back-propagation neural network by evolutionary adaptation of learning rates, Neurocomputing, vol. 11, no.1, pp. 101-106.

[151] Kitano H (1990), Designing Neural Networks Using Genetic Algorithms with graph Generation System, Complex Systems, Volume 4, No.4, pp. 461-476.

[152] Kok J N, Marchiori E, Marchiori M and Rossi C (1996), Evolutionary Training of CLP-Constrained Neural Networks, 2nd Int. Conf. on Practical Application of Constraint Technology, pp.129-142.

[153] Kosko B (1992), Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence, Prentice Hall, Englewood Ccliffs, New Jersey.

[154] Kosko B (1997), Fuzzy Engineering, Upper Saddle River, NJ: Prentice Hall.

[155] Kulkarni A D (2001), Computer Vision and Fuzzy-Neural Systems, Prentice Hall Inc, USA.

[156] Lee M A (1994), Automatic Design and Adaptation of Fuzzy Systems and Genetic Algorithms Using Soft Computing Techniques, PhD thesis, University of California, Davis.

[157] Lee M.A. and Esbensen H (1997), Fuzzy / Multiobjective Genetic Systems for Intelligent Systems Design Tools and Components, Fuzzy Evolutionary Computation, Pedrycz W (Ed.), Kluwer Academic Publishers, pp.107-125.

[158] Li W (1994), Optimization of a fuzzy controller using neural network, In Proceedings IEEE International. Conference on Fuzzy Systems, pp 223-227.

[159] Li X H and Chen C L P (2000), The Equivalance Between Fuzzy Logic Systems and Feedforward Neural Networks, IEEE Transactions on Neural Networks, Volume 11, No. 2, pp. 356-365.

[160] Lin C T & Lee C S G (1991), Neural Network based Fuzzy Logic Control and Decision System, IEEE Transactions on Comput. (40(12): pp. 1320-1336.

[161] Lin C T and Lee C S G (1993), Reinforcement Structure/Parameter Learning for Neural-Network-Based Fuzzy Logic Control Systems. In Proceedings IEEE International. Conference. on Fuzzy Systems, pp. 88-93.

[162] Lin C T and Lee C S G (1996), Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems, Prentice Hall Inc, USA.

[163] Lin T Y, Cecrone N (Ed.) (1997), Rough Sets and Data Mining, Kluwer Academic Publishers, USA.

[164] Liu Y and Yao X (1996), Evolutionary design of artificial neural networks with different node transfer functions, Proceedings of the Third IEEE International Conference on Evolutionary Computation, Nagoya, Japan, pp.670-675.

[165] Liu Y and Yao X (1998), Towards designing neural network ensembles by evolution, Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature (PPSN-V), Lecture Notes in Computer Science, Vol. 1498, AE Eiben, M Schoenauer and HP Schwefel (Ed.), Springer-Verlag, Berlin, pp.623-632.

[166] Lotfi A (1995), Learning Fuzzy Inference Systems, PhD Thesis, Department of Electrical and Computer Engineering, University of Queensland, Australia.

[167] Mackey M C and Glass L (1977), Oscillation and Chaos in Physiological Control Systems, Science Vol 197, pp.287-289.

[168] Magdalena L. (1996), Adapting the Gain of an FLC with Genetic Algorithms, International Journal of Approximate Reasoning, 17(4), pp. 327-349.

[169] Mamdani E H and Assilian S (1975), An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller, International Journal of Man-Machine Studies, Vol. 7, No.1, pp. 1-13.

[170] Mascioli F and Martinelli G (1995), A constructive algorithm for binary neural networks: The oil Spot Algorithm, IEEE Transaction on Neural Networks, 6(3), pp 794-797.

[171] Masters T (1994), Signal and Image Processing with Neural Networks: A C++ Sourcebook, John Wiley and Sons, Inc., New York.

[172] Medsker L R (1995), Hybrid Intelligent Systems, Kluwer Academic Publishers.

[173] Melikhov A, Miagkikh V V and Topchy A P (1996), Optimization of Fuzzy and Neuro-Fuzzy Systems by means of Adaptive Genetic Search, In Proceedings of GA+SE'96 IC, Gursuf, Ukraine.

[174] Merril J W L and Port R F (1991), Fractally Configured Neural Networks, Neural Networks, Vol 4, No.1, pp 53-60.

[175] Meunier B B, Yager R R and Zadeh L A (Eds) (1995), Fuzzy Logic and Soft Computing, World Scientific Publishing Company, Singapore.

[176] Meunier B B, Yager R R, and Zadeh L A (Eds) (2000), Uncertainty in Intelligent and Information Systems, World Scientific Publishing Company, Singapore.

[177] Mezard M and Nadal J P (1989), Learning in feed forward layered networks: The Tiling algorithm, Journal of Physics A, Vol 22, pp. 2191-2204

[178] Miller G F, Todd P M and Hedge S U (1989), Designing Neural Networks Using Genetic Algorithms, Proceedings of the Third International Conference on Genetic Algorithms, JD Schaffer (Ed), pp. 379-384.

[179] Miller T J E (1982), Reactive Power Control in Electric Systems, Wiley– Interscience.

[180] Mitra S and Hayashi Y (2000), Neuro-Fuzzy Rule Generation: Survey in Soft Computing Framework', IEEE Transactions on Neural Networks, Volume II, No. 3. pp. 748-768

[181] Mitra S and Pal S (1995), Fuzzy Multi-Layer Perceptron: Inferencing and Rule Generation, IEEE Transactions Neural Networks, 6: pp. 51-63.

[182] Miyoshi T, Tano S, Kato Y, Arnould T (1993), Operator Tuning in Fuzzy Production Rules Using Neural networks, In Proceedings of the IEEE International Conference on Fuzzy Systems, San Francisco, pp. 641-646.

[183] Mizumoto M (1981), Fuzzy sets and their operations, Information and Control, vol. 48, pp. 30-48.

[184] Mizumoto M and Shi Yan (1997), A New Approach of Neurofuzzy Learning Algorithm, Intelligent Hybrid Systems: Fuzzy Logic, Neural Networks, and Genetic Algorithms, Ruan D (Ed.), Kluwer Academic Publishers, pp. 109-129.

[185] Mohammadian M and Stonier R J (1994), Generating Fuzzy Rules by Genetic Algorithms, Proceedings of 3rd IEEE International Workshop on Robot and Human Communication, Nagoya, pp.362-367.

[186] Moller A F (1993), A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning, Neural Networks, Volume (6), pp. 525-533.

[187] Moriarty D E and Miikkulainen R (1997), Forming Neural Networks through Efficient and Adaptive Co-evolution, Evolutionary Computation, Volume 5, pp. 373-399.

[188] Narazaki H and Ralescu A L (1991), A Synthesis Method for Multi-Layered Neural Network using Fuzzy Sets, International Workshop on Fuzzy Logic in Artificial Intelligence, pp. 54-66.

[189] Nasdaq Stock Market^{SM}: http://www.nasdaq.com

[190] Nath B and Nath M (2000), Using Neural Networks and Statistical Methods for forecasting Electricity Demand in Victoria, International Journal of Management and

Systems (IJOMAS), Special issue on Mathematics for Industry, Volume 16, No. 1, pp. 105-112.

[191] Nauck D and Kruse R (1992), A Neuro Fuzzy Controller Learning by Fuzzy Error Propagation, In Proceedings of Conference of the North American Fuzzy Information Processing Society NAFIPS '92, Mexico, pp. 388-397.

[192] Nauck D and Kruse R (1994), NEFCON-I: An X-Window Based Simulator for Neural Fuzzy Controllers. In Proceedings of the IEEE International Conference on Neural Networks, Orlando, pp. 1638-1643.

[193] Nauck D and Kruse R (1995), NEFCLASS: A Neuro-Fuzzy Approach for the Classification of Data, In Proceedings of ACM Symposium on Applied Computing, George K et al (Eds), Nashville, ACM Press, pp. 461-465.

[194] Nauck D and Kruse R (1997), A Neuro-Fuzzy Method to Learn Fuzzy Classification Rules from Data. Fuzzy Sets and Systems, 89, pp. 277-288.

[195] Nauck D and Kruse R (1997), Function Approximation by NEFPROX, In Proceedings of Second European Workshop on Fuzzy Decision Analysis and Neural Networks for Management, Planning and Optimization, Dortmund, pp. 160-169.

[196] Nauck D and Kruse R (1999), Neuro-Fuzzy Systems for Function Approximation, Fuzzy Sets and Systems, 101, pp. 261-271.

[197] Nomura H, Hayashi I and Wakami N (1992), A Learning Method of Fuzzy Inference Systems by Descent Method, In Proceedings of the First IEEE International conference on Fuzzy Systems, San Diego, USA, pp. 203-210.

[198] Nürnberger A, Nauck D and Kruse R (1999), Neuro-Fuzzy Control Based on the NEFCON-Model: Recent Developments. Soft Computing, 2(4), pp. 168-182.

[199] Omlin C W and Giles C L (1993), Pruning Recurrent neural networks for improved generalization performance, Tech. Report No 93-6, CS Department, Rensselaer Institute, Troy, NY.

[200] Osmera P. (1996), Optimization of parameters of fuzzy logic controllers by genetic algorithms. In Proc. Second Online Workshop on Evolutionary Computation (WEC2), pp. 17-20.

[201] Pal S K and Mitra S (1999), Neuro-Fuzzy Pattern Recognition: Methods in Soft Computing, John Wiley & Sons, Inc, USA.

[202] Pedrycz W (Editor) (1997), Fuzzy Evolutionary Computation, Kluwer Academic Publishers, USA.

[203] Pedrycz W and Card H C (1992), Linguistic Interpretation of Self Organizing Maps, In Prroceedings of the IEEE International Conference on Fuzzy Systems, San Diego, pp. 371-378.

[204] Pedrycz W, Fuzzy Sets Engineering, CRC Press, 1995.

[205] Petrovic-Lazerevic S, Coghill K and Abraham A (2001), Neuro-Fuzzy Support of Knowledge Management in Social Regulation, In Proceedings of Fifth International Conference on Computing Anticipatory Systems, CASYS 2001, Belgium (in press).

[206] Phansalkar V V and Thathachar M A L (1995), Local and Global Optimization Algorithms for Generalized Learning Automata, Neural Computation, 7, pp. 950-973.

[207] Philip N S and Joseph K B. (2001), Adaptive Basis Function for Artificial Neural Networks, Elsevier Neurocomputing Journal, (Accepted for publication).

[208] Polani D and Miikkulainen R (1999), Fast Reinforcement Learning Through Eugenic Neuro-Evolution. Technical Report AI99-277, Department of Computer Sciences, University of Texas at Austin.

[209] Porto V W, Fogel D B and Fogel L J (1995), Alternative neural Network training methods, IEEE Expert, volume 10, no.4, pp. 16-22.

[210] Procyk T J and Mamdani E H (1979), A Linguistic Self Organizing Process Controller, Automatica, Volume 15, pp. 15-30.

[211] Refenes A. (Ed.) (1995). Neural Networks in the Capital Markets, John Wiley and Sons, Inc., England.

[212] Russo M and Jain L C (Eds) (2001), Fuzzy Learning and Applications, CRC Press, USA

[213] Sanchez E, Shibata T and Zadeh L A (Eds) (1997), Genetic Algorithms and Fuzzy Logic Systems: Soft Computing Perspectives, World Scientific Publishing Company, Singapore.

[214] Schiffmann W, Joost M and Werner R (1993), Comparison of optimized backpropagation algorithms, Proceedings. Of the European Symposium on Artificial Neural Networks, Brussels, M. Verleysen (Ed.), de Facto Press, pp. 97-104.

[215] Schweizer B and Sklar A (1963), Associative Functions and Abstract Semi-groups, Publication of Math. Debrecen, 10, pp. 69-81.

[216] Sexton R, Dorsey R and Johnson J (1998), Toward Global Optimization of Neural Networks: A Comparison of the Genetic Algorithm and Backpropagation, Decision Support Systems, 22, pp. 171-185.

[217] Sexton R, Dorsey R and Johnson J (1999), Optimization of Neural Networks: A Comparative Analysis of the Genetic Algorithm and Simulated Annealing, European Journal of Operational Research, 114, pp. 589-601.

[218] Shang Y and Wah B W (1996), Global Optimization for Neural Network Training, Computer, 29, pp.45-55.

[219] Shukla K K and Raghunath (1999), An Efficient Global Algorithm for Supervised Training of Neural Networks, Computers and Electrical Engineering, 25, pp. 195(2).

[220] Simpson P (1992), Fuzzy Min-Max Neural Networks - Part 1: Classification, IEEE Transactions Neural Networks, 3: pp.776-786.

[221] Simpson P (1992), Fuzzy Min-Max Neural Networks - Part 2: Clustering, IEEE Transactions Fuzzy Systems, 1: pp. 32-45.

[222] Sorwar G, Abraham A and Dooley L (2001), Texture Classification Based on DCT and Soft Computing, The 10th IEEE International Conference on Fuzzy Systems, Melbourne, Australia (IEEE-FUZZ'01).

[223] Stepniewski S W and Keane A J (1997), Pruning Back-propagation Neural Networks Using Modern Stochastic Optimization Techniques, Neural Computing & Applications, Vol. 5, pp. 76-98.

[224] Sugeno M (1985), Industrial Applications of Fuzzy Control, Elsevier Science Pub Co.

[225] Sugeno M and Tanaka K (1991), Successive Identification of a Fuzzy Model and its applications to Prediction of a Complex System, Fuzzy Sets and Systems, Volume 28, pp. 315-334.

[226] Sulzberger SM, Tschicholg-Gurman NN, Vestli SJ (1993), FUN: Optimization of Fuzzy Rule Based Systems Using Neural Networks, In Proceedings of IEEE Conference on Neural Networks, San Francisco, pp 312-316.

[227] Takagi H (1990), Fusion Technology of Fuzzy Theory and Neural Networks - Survey and Future Directions. In Proceedings 1st International Conference on Fuzzy Logic & Neural Networks, pp. 13-26.

[228] Takagi H, Hayashi I (1991), NN-Driven Fuzzy Reasoning, International Journal of Approximate Reasoning, Vol.5, pp. 191-212.

[229] Takagi T and Sugeno M (1983), Derivation of Fuzzy Control Rules from Human Operators Control Actions, Proceedings of the IAFC Symposium on Fuzzy Information, Knowledge Representation and Decision Analysis, pp 55-60.

[230] Tano S, Oyama T, Arnould T (1996), Deep combination of Fuzzy Inference and Neural Network in Fuzzy Inference, Fuzzy Sets and Systems, 82(2) pp. 151-160.

[231] Teodorescu, H.N., Kandel, A., and Jain, L.C. (Eds) (1998), Fuzzy and Neuro-fuzzy Systems in Medicine, CRC Press USA.

[232] Topchy A P and Lebedko O A (1997), Neural Network training by means of cooperative evolutionary search, Nuclear Instruments & Methods In Physics Research,

Section A: accelerators, Spectrometers, Detectors and Associated equipment, Volume 389, no. 1-2, pp. 240-241.

[233] Tsukamoto Y (1979), An Approach to Fuzzy Reasoning Method, Gupta MM et al (Eds.), Advances in Fuzzy Set Theory and Applications, pp. 137-149.

[234] Van Rooij A., Jain L.C., and Johnson R.P. (1996), Neural Network Training Using Genetic Algorithms, World Scientific Publishing Company, Singapore.

[235] Vapnik V, Golowich S, and Smola A (1997), Support vector method for function approximation, regression estimation, and signal processing. In M.Mozer, M.Jordan, and T.Petsche, (Eds), Advances in Neural Information Processing Systems *9*, Cambridge, MA, 1997. MIT Press, pages 281-287,

[236] Vasilakos A V and Zikidis K C (1995), A Novel Neuro-Fuzzy Architecture for Fuzzy Computing Based on Functional Reasoning, In Proceedings IEEE International Conference on Fuzzy Systems, pp. 671-678.

[237] Vidhyasagar M M (1997), The Theory of Learning and Generalization, Springer-Verlag, New York.

[238] Von Altrock C (1995), Fuzzy Logic and Neuro Fuzzy Applications Explained, Prentice Hall, Iinc, USA.

[239] Wang L X (1994), Adaptive Fuzzy Systems and Control, Prentice Hall Inc, USA.

[240] Wang L X and Mendel J M (1992), Backpropagation fuzzy system as Nonlinear Dynamic System Identifiers, In Proceedings of the First IEEE International conference on Fuzzy Systems, San Diego, USA, pp. 1409-1418.

[241] Wang L X and Mendel J M (1992), Generating Fuzzy Rules by Learning from Examples, IEEE Transactions on Systems, Man and Cybernetics, Volume 22, No 6., pp. 1414-1427.

[242] Weigend A.S. and Gershenfeld N.A. Eds. (1994) Time Series Prediction: Forecasting the Future and Understanding the Past, Reading, MA: Addison-Wesley.

[243] Yager R (1988), On the Interface of Fuzzy Sets and Neural Networks, In International Workshop on Fuzzy System Applications, pp. 215-216.

[244] Yager R R and Filev D P (1992), Adaptive Defuzzification for Fuzzy System Modeling, In Proceedings of the Workshop of the North American Fuzzy Information Processing Society, pp. 135-142.

[245] Yao C C and Kuo Y H (1995), A Fuzzy Neural Network Model with Three-layered Structure, In Proceedings. IEEE International. Conference on Fuzzy Systems, pp. 1503-1510.

[246] Yao X (1995), Designing Artificial Neural Networks Using Co-Evolution, Proceedings of IEEE Singapore International Conference on Intelligent Control and Instrumentation, pp.149-154.

[247] Yao X (1999), Evolving Artificial Neural Networks, Proceedings of the IEEE, 87(9):1, pp. 423-1447.

[248] Yao X and Liu Y (1997), A new evolutionary system for evolving artificial neural networks, IEEE Transactions on Neural Networks, 8(3), pp. 694-713.

[249] Yao X and Liu Y (1998), Making use of population information in evolutionary artificial neural networks, IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics, 28(3): pp. 417-425.

[250] Yao X and Liu Y (1998), Making Use of Population Information in Evolutionary Artificial Neural Networks, IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics, 28(3): pp. 417-425.

[251] Yao X and Liu Y (1998), Towards designing artificial neural networks by evolution, Applied Mathematics and Computation, 91(1): pp. 83-90.

[252] Yoshinari Y, Pedrycz W, Hirota K (1993), Construction of Fuzzy Models Through Clustering Techniques, Fuzzy Sets and Systems, Volume 54, pp. 157-165.

[253] Zadeh L A (1965), Fuzzy Sets, Information and Control, Volume 8: pp. 338-353.

[254] Zadeh L A (1973), Outline of a New Approach to the Analysis of Complex Systems and Decision Processes, IEEE Transactions on Systems, Man and Cybernetics, 3(1): pp. 28-44.

[255] Zadeh LA (1998), Roles of Soft Computing and Fuzzy Logic in the Conception, Design and Deployment of Information/Intelligent Systems, Computational Intelligence: Soft Computing and Fuzzy-Neuro Integration with Applications, O Kaynak, LA Zadeh, B Turksen, IJ Rudas (Eds.), pp. 1-9.

[256] Zhang Q Y and Kandel A (1998), Compensatory Neuro-fuzzy Systems with Fast Learning Algorithms, IEEE Transactions on Neural Networks, Volume 9, No. 1, pp.83-105.

[257] Zhang X M and Chen Y Q (2000), Ray-guided global optimization method for training neural networks, Neurocomputing, 30, pp. 333-337.

[258] Zimmermann H G, Neuneier R, Dichtl H and Siekmann S (1996), Modeling the German Stock Index DAX with Neuro-Fuzzy, In Proceedings of the Fourth European Congress on Intelligent Techniques and Soft Computing (EUFIT'96).