

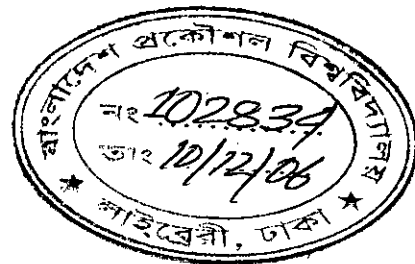
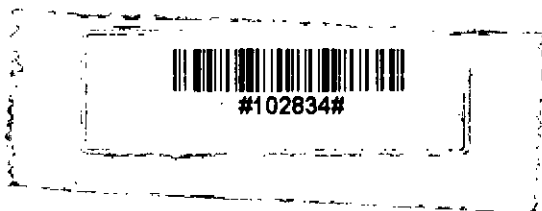
A New Algorithm to Design Multiple Hidden Layer Artificial Neural Networks

By

Mohammad Iqbal Bin Shahid

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

July 2006



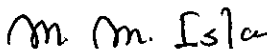
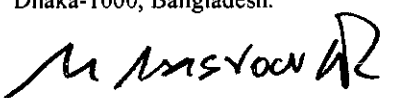
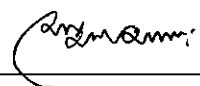
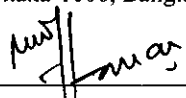

Submitted to

Bangladesh University of Engineering and Technology

In partial fulfillment of the requirements for
M. Sc. Engg. (Computer Science and Engineering) degree.

The thesis titled "A New Algorithm to Design Multiple Hidden Layer Artificial Neural Networks" submitted by **Mohammad Iqbal Bin Shahid**, Roll No. 040305002P, Session April 2003 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of Master of Science and Engineering in Computer Science and Engineering (M.Sc. Engg. in CSE) held on July 01, 2006.

BOARD OF EXAMINERS

1. 
Chairman
Dr. Md. Monirul Islam
(Supervisor)
Associate Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology (BUET)
Dhaka-1000, Bangladesh.
2. 
Member
Dr. Muhammad Masroor Ali
Professor and Head
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology (BUET)
Dhaka-1000, Bangladesh.
3. 
Member
Dr. Md. Mostofa Akbar
Assistant Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology (BUET)
Dhaka-1000, Bangladesh.
4. 
Member
Dr. Masud Hasan
Assistant Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology (BUET)
Dhaka-1000, Bangladesh.
5. 
Member (External)
Dr. Hafiz Muhammad Hasan Babu
Associate Professor
Department of Computer Science and Engineering
Dhaka University
Dhaka-1000, Bangladesh.

CANDIDATE'S DECLARATION

It is hereby declared that this thesis or any part of it has not been submitted elsewhere for the award of any degree or diploma.

M. M. Islam 01.07.2006

Signature of the Supervisor

Dr. Md. Monirul Islam

M. I. Bin Shahid 01.07.2006

Signature of the Candidate

Mohammad Iqbal Bin Shahid

To
My Beloved Mother, My Sister
&
Dear Wife

Without whom any achievement in life would be meaningless

89

ACKNOWLEDGEMENT

At first, I want to thank almighty Allah. All praises go to Allah. Almighty gave me enough hardworking capability and patience which made me to complete the thesis work.

I performed the thesis work under the supervision of my respectable teacher Dr. Md. Monirul Islam. I am grateful to him for his constant guidance, helpful suggestions and precious assistance. He helped me by giving his invaluable time throughout the thesis work. His encouragement and motivations helped me to overcome all difficulties.

I pay my gratitude to Dr. Muhammad Masroor Ali, Dr. Md. Abul Kashem Mia, Dr. A. S. M. Latiful Hoque, Dr. Md. Saidur Rahman and Dr. Md. Mostofa Akbar for their positive criticisms. These helped me a lot to practice achieving perfection.

I also want to thank Dr. Md. Bashir Uddin and Dr. Md. Shahid Ullah, Professors of the Department of EEE, Dhaka University of Engineering and Technology (DUET), Gazipur, for giving me enough flexibilities and lab facilities to complete my thesis work. I am also grateful to Mr. Mashud Hyder, my colleague, for sharing thoughts and ideas.

At last, I want thank my family and especially my beloved wife for continuously inspiring me to complete my thesis work smoothly.

ABSTRACT

This thesis presents a new constructive algorithm called multilayered constructive architecture (MCA) for designing and training multiple hidden layered artificial neural networks (ANNs). Unlike most previous constructive algorithms, MCA puts emphasis on both simplicity and generalization ability of designed ANNs. In order to maintain simplicity, MCA uses a minimum number of user specified parameters in designing ANNs. The use of both layered and cascaded architecture in MCA increases the generalization ability of designed ANNs. MCA has been tested extensively on a number of benchmark problems in machine learning and neural networks, including Australian credit card assessment, breast cancer, diabetes, glass and heart disease. The experimental results show that MCA can produce compact ANNs with good generalization ability.

Contents

		Acknowledgement	i
		Abstract	ii
		Contents	iii
		List of Figures	v
		List of Tables	vi

CHAPTER 1	1.1	Introduction	01
INTRODUCTION	1.2	Historical Survey	02
	1.2.1	<i>Single Hidden Layered ANNs</i>	03
	1.2.2	<i>Multiple Hidden Layered ANNs</i>	06
	1.3	Objectives	08
	1.4	Organization of Chapters	09

CHAPTER 2	2.1	Introduction	10
BACKGROUND	2.2	Human Brain	10
	2.3	Biological Basis of Neural Networks	11
	2.4	Model of a Neuron	12
	2.5	Learning Methods	15
	2.5.1	<i>Supervised Learning</i>	16
	2.5.2	<i>Unsupervised Learning</i>	16
	2.5.3	<i>Reinforced Learning</i>	16
	2.6	Characteristics of ANNs	17
	2.7	Backpropagation Learning Algorithm	17
	2.8	Constructive Algorithms	21
	2.9	Cascade Correlation Learning Architecture (CasCor)	21
	2.9.1	<i>Reasons Behind the Development of CasCor</i>	21
	2.9.2	<i>CasCor</i>	22
	2.10	CNNDA	25
	2.10.1	<i>Description of the algorithm</i>	25
	2.11	Four Layered Vs Three Layered ANNs	29
	2.11.1	<i>Construction of a Four Layered ANN</i>	29
	2.12	Conclusion	34

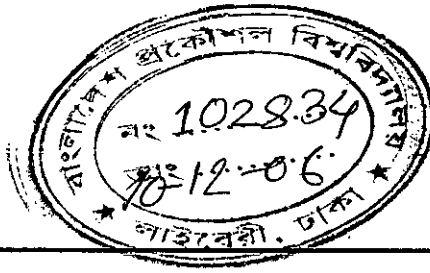
CHAPTER 3	3.1	Introduction	35
MULTILAYERED CONSTRUCTIVE ARCHITECTURE	3.2	MCA	35
	3.2.1	<i>Modified Single Layer BP Algorithm</i>	40
	3.2.2	<i>The Correlation Maximization Process</i>	41
	3.2.3	<i>The Negative Correlation Criterion</i>	41
	3.3	Advantages of MCA	43
	3.4	Differences with Existing Algorithms	45
	3.5	Conclusion	46
CHAPTER 4	4.1	Introduction	47
SIMULATION RESULTS	4.2	Description of Datasets	47
	4.2.1	<i>The Breast Cancer Problem</i>	47
	4.2.2	<i>The Australian Credit Card Problem</i>	48
	4.2.3	<i>The Diabetes Problem</i>	48
	4.2.4	<i>The Glass Problem</i>	48
	4.2.5	<i>The Heart Disease Problem (General Dataset)</i>	49
	4.2.6	<i>The Heart Disease Problem (Cleveland Dataset)</i>	49
	4.3	Experimental Setup	50
	4.4	Results and Analysis	51
	4.5	Comparison	61
	4.6	Discussion	64
	4.7	Conclusion	66
CHAPTER 5	5.1	Concluding Remarks	67
CONCLUSION	5.2	Future Scopes	68
REFERENCES			69

List of Figures

Figure No.	Figure Title	Page No.
2.1	Block diagram representation of human nervous system	10
2.2	Sketch of a biological neuron showing components	11
2.3	Model of a neuron	12
2.4	Threshold function	14
2.5	Piece-wise linear function.	14
2.6	Sigmoid function	15
2.7	Hyperbolic tangent function	15
2.8	The Cascade architecture of CasCor	23
2.9	A two hidden layer multilayer perceptron (MLP)	26
2.10	Three-layered subnetwork	29
2.11	Units A and B	31
2.12	Input space and separating hyperplane (two-dimensional case)	32
2.13	Complete four-layered feedforward ANN	33
3.1	Flowchart of MCA	36
3.2	Initial ANN	37
3.3	Add new neuron in the $H1$ layer	38
3.4	ANN with $H1$ layer completed	38
3.5	First neuron in cascade above $H1$ layer	39
3.6	Final ANN with $H1$ hidden layer and two neurons in cascade above it	39
3.7	Initial ANN	42
3.8	Only first hidden layer neurons are considered after adding $n2$	42
3.9	More neurons ($3...l$) are added until SN crosses zero	43
3.10	Comparison of propagation delay (bold lines) between CasCor and MCA	44
4.1	Training process of MCA for <i>cancer</i> problem	55
4.2	Training process of MCA for <i>card</i> problem	56
4.3	Training process of MCA for <i>diabetes</i> problem	57
4.4	Training process of MCA for <i>glass</i> problem	58
4.5	Training process of MCA for <i>heart</i> problem	59
4.6	Training process of MCA for <i>heartc</i> problem	60

List of Tables

Table No.	Table Title	Page No.
4.1	Characteristics of Datasets	50
4.2	Results of MCA on Different Classification Problems	52
4.3	Comparison Among MCA, CasCor, aCasper and CNNDA in Terms of Average Number of Hidden Neurons Required	62
4.4	Comparison Among MCA, CasCor, aCasper and CNNDA in Terms of Classification Errors	62
4.5	Comparison Between MCA and CNNDA in Terms of Training Epochs	63



Chapter 1

INTRODUCTION

1.1 Introduction

An artificial neural network (ANN) is a massively parallel distributed processor made up of simple processing units, which have a natural propensity of storing experiential knowledge and making it available for use. These units are generally referred to as neurons. ANN resembles the brain in two respects— first, knowledge is acquired by it from the environment through a learning process and second, interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge. The procedure used to perform the learning process is called a learning algorithm. The function of learning algorithm is to modify the synaptic weights of ANNs in orderly fashion to solve a given problem. Learning algorithm is also referred to as training algorithm.

The performance of any ANN is defined mainly in terms of generalization ability and training time. Training time is referred to as the time required to complete the learning process. It depends on the ANN architecture, type of data used and the learning algorithm. Generalization refers to the ANN producing reasonable outputs for inputs not encountered during training. Both of these performance measures depend greatly on the architecture of ANN.

ANNs can roughly be categorized into two types depending on architecture. The first type is single layered ANN that consists of an input layer and an output layer. Another type is the multilayered ANN that consists of additional layer(s) referred to as the hidden layer(s). Multilayered ANNs can be categorized into two types— single hidden layered ANNs and multiple hidden layered ANNs. Single hidden layered ANNs are generally referred to as three layered ANNs. In most of the cases, multiple hidden layered ANNs consist of two hidden layers, resulting in four layered ANNs. Although three layered ANNs can solve any real world problems, four layered ANNs perform better. It is still unsolved if more than four layers of neurons can perform better or not.

Multilayered ANNs with fixed topology trained using standard backpropagation (BP) algorithm [1] based on gradient descent are the most common use of ANN models. These ANNs are only useful with the appropriate architecture which is found by trial and error in BP. The optimal architecture is the one that can generalize well with concise network size. Moreover, the architecture should be formed in such a manner that the training time should be as low as possible.

All algorithms that determine the ANN architecture have to start with an initial architecture. The initial architecture is determined by the nature of the algorithm used. In constructive approach, the algorithm starts with a small network and constructs the network by adding neurons and connections [2]. The simplest ANN consists of no hidden neurons. In problems where prior information exists, the initial state can be different. In pruning approach [3], in contrast to the constructive approach, the algorithm starts with a large network and removes the unnecessary neurons and connections. The search must be terminated when the generalization performance of the ANN begins to decrease. Some algorithms continue until all training examples are correctly classified but these algorithms fail to learn noisy data and generate larger ANNs.

1.2 Historical Survey

The modern era of ANN began with the pioneering work of McCulloch and Pitts (1943). In their classic paper, they describe a logical calculus of neural networks that united the studies of neurophysiology and mathematical logic. They formed an ANN consisting of some basic neurons and synaptic connections which can compute any computable function. The discipline of ANN and artificial intelligence was born with this work.

The next major development was in 1949, when Hebb's book "*The Organization of Behavior*" was published. Hebb's approach depends mainly on the simultaneous activity of two neurons connected at the two ends of any synaptic connection. However, the book had almost no impact on the engineering community though it was immensely influential among psychologists. Some major developments of ANN were achieved from 1958 to 1986. Among them, Rosenblatt's perceptron (1958), Widrow's Adaline (1960), Widrows Madaline (1962) and Hopfield network (1982) were prolific.

In 1986, the most popular learning algorithm i.e. BP was conceived by Rumelhart, Hilton and Williams. Up to now, most ANNs are trained using BP learning algorithm or some form of its variations.

The automated design of ANNs is an important issue for any learning task. There have been many attempts to design ANNs automatically, such as various evolutionary and non-evolutionary algorithms. Depending on the architecture, both evolutionary and non-evolutionary algorithms can be divided into two major types—

- Single hidden layered ANNs
- Multiple hidden layered ANNs

In 2005, Xiang, Ding and Lee published a paper on geometrical interpretation and architecture selection of MLP [4]. Some general guidelines for selecting the architecture of the MLP, i.e., the number of the hidden neurons and the hidden layers, are proposed based upon this interpretation and the controversial issue of whether four-layered MLP is superior to the three-layered MLP is also carefully examined.

1.2.1 Single Hidden Layered ANNs

Single hidden layered ANNs consist of three layers— one input, one output and one hidden layer. Hidden layer consists of a certain number hidden neurons. The design criteria for three layered ANNs are rather simple. Thus, several algorithms for designing these types of ANNs are proposed. The major algorithms for designing three layered ANNs are—

- a. Constructive Algorithms
- b. Pruning Algorithms
- c. Evolutionary Algorithms

a. Constructive Algorithms: The concept of constructive algorithm was first proposed by Ash in 1989 [5]. It was named DNC- Dynamic Node Creation. DNC starts with an initial ANN and incrementally adds hidden neurons to the network until a satisfactory solution is found. Hidden neurons are added once at a time to the same hidden layer when the average error curve begins to flatten too quickly. The whole ANN is retrained after the addition of each hidden neuron.

The Upstart algorithm [6] is a constructive algorithm for binary classification problems. The algorithm starts without hidden neurons and tries to separate the data. If separation is not possible, then corrector nodes are added. The generated ANN is very much similar to an ANN with one hidden layer by defining all the corrector nodes as hidden neurons.

The Grow and Learn (GAL) [7] algorithm is a bit modified variation of constructive algorithm. This algorithm has two phases- *active* and *sleep*. In the *active* phase, the network grows while learning class definitions. In the *sleep* phase of the algorithm, the neurons those are no longer necessary are removed to reduce the complexity.

Constructive algorithm by node splitting was proposed in 1993 by Mike Wynn-Jones [8]. In this method, new neurons are added to the hidden layer not by adding and training, but by “splitting” an existing neuron. An algorithm that adds, deletes neurons and layers was proposed by Nabhan and Zomaya [9]. The algorithm applies an intelligent generate and test procedure, explores different alternatives and selects the most promising one.

A relatively new algorithm, Constructive Algorithm for Real Valued Examples (CARVE), was proposed by Young and Downs. CARVE [10] uses convex hull methods for the determination of ANN weights. The algorithm starts with an empty hidden layer into which thresholds units are added one at a time until the layer is complete.

Construction Using Cross Validation [11] uses cross validation for adding neurons to a single hidden layered ANNs. The ANNs with more hidden neurons is only accepted if the total accuracy on training and cross validation samples is higher than that of the previous ANN.

Finally in 2005, a new algorithm for variable selection and architecture definition in multilayer perceptron was published by Eleuteri, Tagliaferri and Milano [12]. In this paper, a novel information geometric-based variable selection criterion for multi-layer perceptron networks is described. It is based on projections of the Riemannian manifold defined by a multi-layer perceptron network on sub manifolds defined by multi-layer perceptron networks with reduced input dimension.

b. Pruning Algorithms: Pruning algorithm is another type of architecture optimization algorithm. As different constructive algorithms became popular in the course of time,

many researchers became interested in a new strategy, which is deletion of neurons and connections from an initial large ANN.

Node pruning in backpropagation networks was developed by Chung and Lee and published in 1992 [13]. In this paper, starting from the viewpoint of pattern classification theory, a study on the characteristics of hidden neurons in oversized ANNs is reported. It shows that four categories of excessive hidden neurons can be found in an oversized ANN. A new node pruning algorithm to attain appropriate sized BP networks is then proposed by detecting and removing those excessive neurons.

Iterative pruning algorithm for feedforward ANN was proposed in 1997 [14]. In this method, the key idea is to iteratively eliminate neurons and to adjust the remaining weights. This is done in such a way that the ANN performance does not worsen over the entire training set. The algorithm also provides a simple criterion for choosing the neurons to be removed, which has proved to work well in practice.

A new pruning algorithm is the bottom-up decision tree pruning algorithm with near optimal generalization [15]. It was published in 1998. In this work only a single pass through the tree is required and a strong performance guarantee is proven for the generalization error of the resulting pruned tree.

c. Evolutionary Algorithms: In 1990, a new algorithm was conceived in the field of architecture optimization, known as evolutionary algorithm. In this algorithm, any ANN actually 'evolves' rather than being constructed [16]. After this, the use of evolutionary algorithm in ANNs is increasing rapidly. However, almost all the evolving ANNs consist of a single hidden layer.

Yao, in 1993, proposed the idea of evolutionary artificial neural networks (EANNs) [17] using genetic algorithm. This algorithm distinguishes among three levels of evolution in EANNs, i.e. the evolution of connection weights, architectures and learning rules. Interactions between different levels of evolution were consulted in that work. It was also argued that the evolution of learning rules and its interactions with other levels of evolution play a vital role in EANNs.

Yao and Liu developed EPNet for evolving ANNs in 1997 [18]. This paper presents a new evolutionary system, i.e., EPNet, for evolving ANNs. The evolutionary algorithm used in EPNet is based on Fogel's evolutionary programming (EP). Unlike most previous studies on evolving ANN's, this paper puts its emphasis on evolving ANN's behaviors. This is one of the primary reasons why EP is adopted. Five mutation operators proposed in EPNet reflect such an emphasis on evolving behaviors. Close behavioral links between parents and their offspring are maintained by various mutations, such as partial training and node splitting. EPNet evolves ANN's architectures and connection weights (including biases) simultaneously in order to reduce the noise in fitness evaluation. The parsimony of evolved ANN's is encouraged by preferring neuron/connection deletion to addition. EPNet can produce very compact ANN's with good generalization ability in comparison with other algorithms.

1.2.2 Multiple Hidden Layered ANNs

In this type of ANN, the number of hidden layers is more than one. To optimize the architecture consisting more than three layers some algorithms have already been proposed. Most of them are constructive in nature, while some of them incorporate pruning also. There is no such algorithm which uses pruning or evolutionary programming for designing this type of ANN. Compared to three layered ANN designing algorithm, very few algorithms exist for designing four or more layered ANNs. In the previous subsection some three layered ANN designing algorithms are consulted but there are many more algorithms to design such type of ANNs. Whereas, cascade correlation learning architecture [19], modified versions of cascade correlation learning architecture [20-25], CNNDA [26] and MOST [27] are the prolific designing algorithms for multiple hidden layered ANNs. There are no other important algorithms to design these types of ANNs. So, the field of designing these types of ANNs is still wide opened.

The cascade correlation learning architecture (CasCor) was the first algorithm to design multiple hidden layer ANNs [19]. It was proposed by Fahlman and Lebiere in 1991. Instead of just adjusting the weights in an ANN of fixed topology, CasCor begins with a minimal network, then automatically trains and adds new hidden neurons one by one in cascade, creating a multilayered architecture. Newly added hidden neurons are trained by

maximizing the correlation between it and the overall residual error. Most of the multiple hidden layered ANN designing algorithms are based on CasCor.

In 1991, Sjøgaard and Yeung, separately tried to simplify the CasCor algorithm by adding the neurons in a single layer rather than in cascade [20, 21]. Both of them became succeeded with the ANN acting upon very simple problems. But both of their algorithms failed to solve complex problems.

In 1992, Littman and Ritter designed an algorithm, which constructs the ANN in a cascaded fashion but uses error minimization rather than maximizing correlation [22]. The performance of the ANN is achieved by making several layers of nonlinear neurons those are trained in a strictly feedforward manner and adding one after another. The influence of objective function is also observed in this work.

Simplified CasCor learning was proposed in 1995 by Lehtokangas, *et al.* [23]. In this work they tried some modifications on the conventional CasCor by removing the shortcut connections and adding the neurons in layer rather than cascading. It was proven with simulations that the shortcut connections do not increase the performance of a CasCor network. The most important feature of CasCor algorithm lies in the training process.

In 1999, Lehtokangas proposed an algorithm for initializing weights in CasCor learning [24]. It is based on the concept of stepwise regression. Lehtokangas, again in 2000, proposed a modification of conventional CasCor [25]. In his paper, he emphasized on the optimal hyperplane constraints. Using these constraints, he achieved better generalization, smaller ANN size and faster learning.

One major improvement of CasCor was the cascade neural network design algorithm (CNDA) [26]. It was conceived by Islam and Murase in 2000. It has an architecture consisting of four layers. There are no direct connections from input layer to output layer in this architecture. In order to improve the generalization ability, CNDA uses a combination of constructive and pruning algorithms and bounded fan-ins for hidden neurons.

In 2003, Aran and Alpaydin proposed a constructive algorithm with Multiple Operators using Statistical Test (MOST) for determining the architecture [27]. The ANNs that are constructed by MOST can have multiple hidden layers with multiple hidden neurons in each layer. The algorithm uses neuron removal, addition and layer addition and determines the number of neurons in layers by heuristics. It applies a statistical test to compare different architectures.

1.3 Objectives

ANNs can be incorporated mainly with two tasks— classification and regression. Our target is to design a constructive ANN which can perform classification tasks easily. Although significant number of constructive algorithms and variations have already been proposed for classification, most of them suffer from one or more of the following problems—

- ◆ Generalization problem
- ◆ Complex architecture
- ◆ Confined architecture
- ◆ Lot of user defined parameters
- ◆ Large propagation delay
- ◆ Slow convergence
- ◆ ANNs designed for complex problems cannot solve easier problems properly and vice versa.

In [28], it has been proven mathematically that an ANN consisting of four or more layers is much preferable over a three layer ANN in terms of number of parameters needed for the training and generalization ability. Besides, cascading the hidden layer causes the ANN to be too complex to solve the real world problems. On the other hand, ANNs with strictly layered architecture can show poor performance for complex problems. A combination of both is necessary. Considering all the above mentioned problems and findings we have decided our objectives as—

- ◆ To develop a new constructive algorithm that automatically creates an ANN combined of both layered and cascaded architecture.

- ◆ To reduce the complexity of a general cascaded architecture created by CasCor and its variants.
- ◆ To determine the performance of new algorithm by applying it to real world data sets.
- ◆ To compare the performance of the new algorithm with some existing algorithms.

1.4 Organization of Chapters

The remaining chapters of this thesis are organized as follows:

- ◆ *Chapter 2* provides background material for the rest of the thesis. Human brain, biological basis of the ANNs, model of a neuron is first elaborated. ANN architectures, learning methods, characteristics of ANNs, and some application domains of ANNs are discussed next. The BP training algorithm, which is used at different stages of training, is presented then. A brief discussion on CasCor and CNNDA is given. Finally, the analysis on the superiority of four layered ANN over three layered ANN is provided.
- ◆ *Chapter 3* presents the proposed algorithm MCA: *Multilayered Constructive Architecture*, which is the main contribution of this thesis. Algorithm of MCA is first elaborated; detailed descriptions of different processes and methods used in MCA are then described. Advantages of MCA over other similar approaches are enlisted then. This chapter ends with the statements of difference between MCA and other algorithms.
- ◆ *Chapter 4* presents a detailed experimental evaluation of MCA. In the reported experiments, MCA is applied to solve classification problems. This chapter evaluates MCA's performance on several well-known benchmark classification problems. Types and sources of data, experimental details, results, analysis and comparison with other algorithms are described.
- ◆ *Chapter 5* presents the conclusive remarks on the research and proposed future research tasks based on MCA.

2.1 Introduction

ANNs are simplified models of the biological neuron system. They are massively parallel distributed processing system made up of highly interconnected computing elements, neuron, that have the ability to learn and thereby acquire knowledge and make it available for use.

ANNs are simplified imitations of the central nervous system, and obviously therefore, have been motivated by the kind of computing performed by the human brain. The structural constituents of a human brain termed neurons are the entities, which perform computations such as cognition, logical inference, pattern recognition, and so on. Hence the technology, which has been built on a simplified imitation of computing by neurons of a brain, has been termed Artificial Neural Networks (ANNs) or simply Neural Networks.

2.2 Human Brain

The human nervous system may be viewed as a three-stage system, as depicted in the block diagram of Fig. 2.1. Central to the system is the brain, represented by the neural (nerve) net, which continually receives information, perceives it, and makes appropriate decisions. Two sets of arrows are shown in the figure, pointing from left to right indicate the forward transmission of information bearing signals through the system. The arrows, which point from right to left, represent the presence of feedback. The receptors convert stimuli from the human body or the external environment into electrical impulses that convey information to the neural net (brain). The effectors convert electrical impulses generated by the neural net into discernible responses as system outputs.

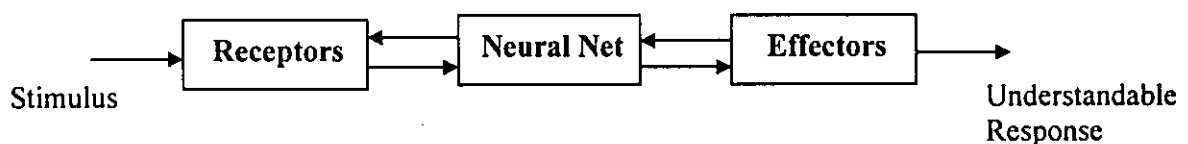


Fig. 2.1 Block diagram representation of human nervous system.

The struggle to understand the brain has been made easier because of the pioneering work of Ramon y Cajal [29], who introduced the idea of neurons as structural constituents of the brain. Typically neurons are five to six orders of magnitude slower than silicon logic gates, events in a silicon chip happen in the nanosecond (10^{-9} s) range, whereas neural events happen in the millisecond (10^{-3} s) range. However, the brain makes up for the relatively slow rate of operation of a neuron by having a truly staggering number of neurons (nerve cells) with massive interconnections between them. It is estimated that there are approximately 10 billion neurons in the human cortex and 60 trillion synapses or connections [30]. The net result is that there is an enormously efficient structure. The energetic efficiency of the brain is approximately 10^{-16} Jules per operation per second, whereas the best computers used today is about 10^{-6} Jules per operation per second [31].

2.3 Biological Basis of Neural Networks

The human brain is a very complex system capable of thinking, remembering and problem solving. A neuron is the fundamental node of the brain's nervous system. It is a simple processing element that receives and combines signals from other neurons through input paths called dendrites. If the combined input signal is strong enough, the neuron 'fires', producing an output signal along the axon that connects to the dendrites of many other neurons. Fig. 2.2 is a sketch of neuron showing the various components. Each signal coming into a neuron along dendrites passes through a synapse or synaptic junction. This junction is an infinitesimal gap in the dendrites that is filled with neurotransmitter fluid that either accelerates or retards the flow of electrical charges.

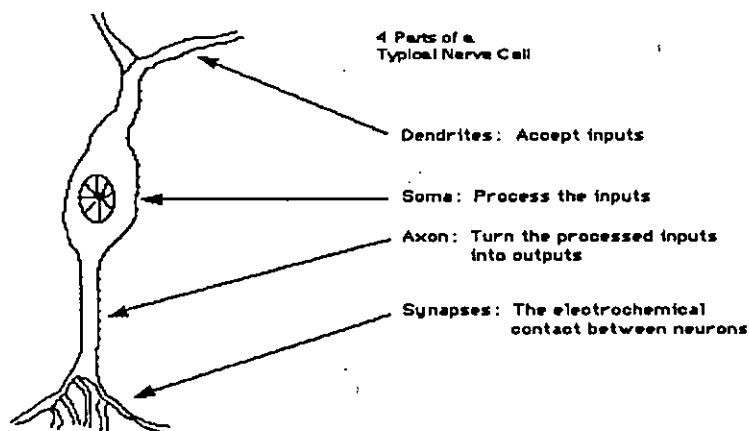


Fig. 2.2 Sketch of a biological neuron showing components.

The fundamental actions of the neurons are chemical in nature, and this neurotransmitter fluid produces electrical signals that go to the nucleus or soma of the neuron. The adjustment of the impedance or conductance of the synaptic gap is a critically important process. Indeed, these adjustments lead to memory and learning. As the synaptic strengths of the neuron are adjusted, the brain “learns” and stores information.

2.4 Model of a Neuron

A neuron is an information-processing node that is fundamental to the operation of a ANN. The block diagram of Fig. 2.3 shows the model of a neuron, which forms the basis for designing artificial ANNs. The three basic elements of the neural model are discussed below:

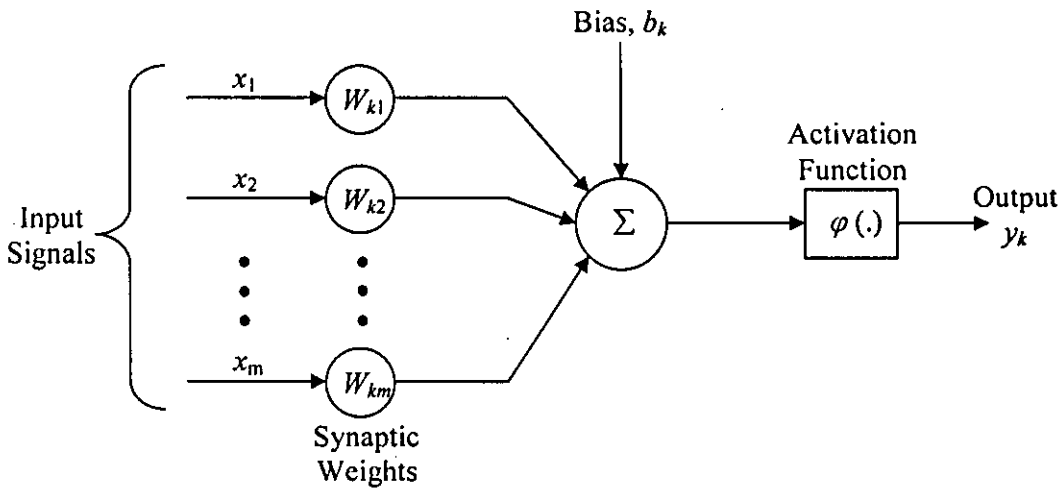


Fig. 2.3 Model of a neuron

- A. *A set of synapse or connecting links*, each of which is characterized by a weight. Specifically, a signal x_j at the input of synapse j connected to neuron k is multiplied by the synaptic weight w_{kj} . Unlike a synapse in the brain, the synaptic weight of artificial neuron may lie in the range that includes negative as well as positive values.
- B. *An adder* for summing the input signals, weighted by the respective synapse of the neuron; the operations described here constitutes a linear combiner.
- C. *An activation function* for limiting the amplitude of the output of a neuron. The activation function is also referred to as a squashing function in that it squashes

(limits) the permissible amplitude range of output signal to some finite value. Typically, the normalized amplitude range of the output of a neuron is written as the closed node interval $[0, 1]$ or alternatively $[-1, 1]$.

The neural model of Fig. 2.3 also includes an externally applied bias denoted by b_k . The bias b_k has the effect of increasing or lowering the net input of the activation function, depending on whenever it is positive or negative, respectively.

In mathematical terms, a neuron k may describe by writing the following pairs of equations:

$$u_k = \sum_{j=1}^m w_{kj} x_j \quad (2.1)$$

and

$$y_k = \varphi(u_k + b_k) \quad (2.2)$$

where x_1, x_2, \dots, x_m are the input signals; $w_{k1}, w_{k2}, \dots, w_{km}$ are the synaptic weights of neuron k ; u_k is the linear combiner due to the input signals; b_k is the bias; $\varphi(\cdot)$ is the activation function; and y_k is the output signal of the neuron. The use of bias b_k has the effect of applying an affine transformation to the output u_k of the linear combiner in the model of Fig. 2.3, as shown by

$$v_k = u_k + b_k \quad (2.3)$$

The bias b_k is can be considered as an internal parameter of artificial neuron k . Considering this, Eqs. (2.1 – 2.3) can be formulated as follows:

$$v_k = \sum_{j=0}^m w_{kj} x_j \quad (2.4)$$

and

$$y_k = \varphi(v_k) \quad (2.5)$$

The activation function, denoted by $\varphi(v)$, defines the output of a neuron in terms of the induced local field v . The three basic types of activation functions are describe below:

A. Threshold Function: For this type of activation function, the output of any neuron y is expressed as -

$$y = \varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases} \quad (2.6)$$

Where, v is the induced local field of the neuron; that is

$$v = \sum wx + b \quad (2.7)$$

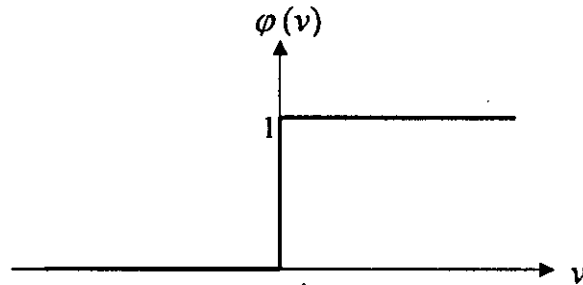


Fig. 2.4 Threshold function.

B. Piece-wise Linear Function: The piece-wise linear function described in Fig. 2.5 is as follows –

$$\varphi(v) = \begin{cases} 1, & v \geq +\frac{1}{2} \\ v, & +\frac{1}{2} > v > -\frac{1}{2} \\ 0, & v \leq -\frac{1}{2} \end{cases} \quad (2.8)$$

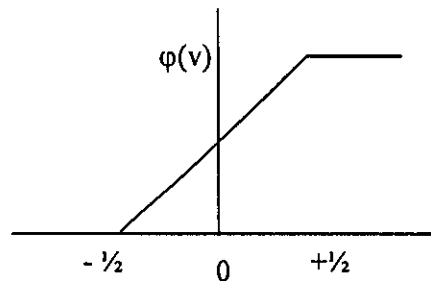


Fig. 2.5 Piece-wise linear function.

C. Sigmoid Function: The sigmoid function, whose graph is s-shaped, is by far the most common for activation function used in the construction of artificial ANNs. It is defined as a strictly increasing function that exhibits a graceful balance between linear and nonlinear behavior. An example of sigmoid function is logistic function, defined by –

$$\varphi(v) = \frac{1}{1 + \exp(-av)} \quad (2.9)$$

Where a is the slope parameter of the sigmoid function. The sigmoid function is continuous and differentiable.

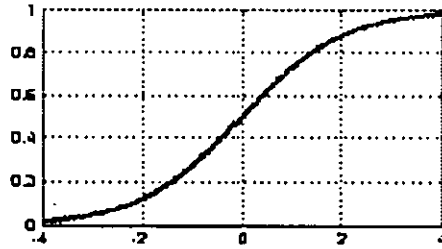


Fig. 2.6 Sigmoid function.

The activation functions defined by Eqs. (2.6), (2.8) and (2.9) range from 0 to +1. It is sometimes desirable to have the activation function range from -1 to +1. In that case, the activation function assumes an antisymmetric form with respect to the origin. For the corresponding form of a sigmoid function, hyperbolic tangent function described in Fig. 2.7 is defined by

$$\varphi(v) = a \tanh(bv) = a \frac{e^{bv} - e^{-bv}}{e^{bv} + e^{-bv}} \quad (2.10)$$

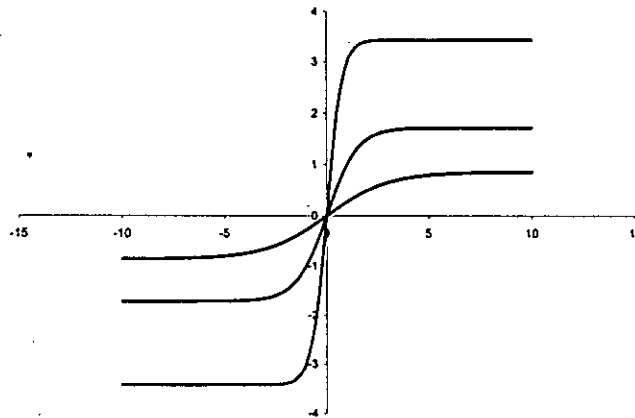


Fig. 2.7 Hyperbolic tangent function

2.5 Learning Methods

In the context of ANNs, learning can be defined as:

“Learning is a process by which the free parameters of a neural network are adapted through a process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes takes place.” [32].

This definition of learning process implies the following sequence of events:

- ◆ The ANN is *stimulated* by the environment.
- ◆ The ANN *undergoes changes* in its free parameters as a result of this stimulation.
- ◆ The ANN *responds in a new way* to the environment because of the changes that have occurred in its internal structure.

Learning methods in ANNs can be broadly classified into three basic types: supervised, unsupervised, and reinforced.

2.5.1 Supervised Learning

In supervised learning both inputs and outputs are provided. The ANN then processes the inputs and compares the outputs against the desired outputs. Errors are then propagated through the system, causing the system to adjust the weights, which control the ANN. This process occurs over and over as the weights are continually tweaked. The set of data which enables the training is called the “training set”. During the training of an ANN, the same set of data is processed many times as the connection weights are ever refined. A very popular example of supervised learning is the BP algorithm.

2.5.2 Unsupervised Learning

The other type of training is called unsupervised learning. In unsupervised learning, the ANN is provided with inputs but not with desired outputs. The system itself must then decide what features it will use to group the input data. This is often referred to as self-organization. At the present time unsupervised learning is not well understood. This adoption to the environment is the promise, which would enable science fiction types of robots to continually learn on their own as they encounter new situations and new environments.

2.5.3 Reinforced Learning

In this method, a teacher though available, does not present the expected answer but only indicates if the computed output is correct or incorrect. The information provided helps the ANN in its learning process. A reward is given for a correct answer computed and a penalty for a wrong answer. But, reinforced learning is not one of the popular forms of learning.

2.6 Characteristics of ANNs

ANNs have profound strengths and weaknesses, and these must be recognized if they are to be used properly. The characteristics of ANNs are described below:

- ◆ ANNs exhibit mapping capabilities, that is, they can map input patterns to their associated output patterns.
- ◆ ANNs learn by examples. Thus, ANN architectures can be 'trained' with known examples of a problem before they are tested for their 'inference' capability on unknown instances of the problem. They can, therefore, identify new objects previously untrained.
- ◆ ANNs possess the capability to generalize. Thus, they can predict new outcomes from past trends.
- ◆ ANNs are robust systems and are fault tolerant. They can, therefore, recall full patterns from incomplete, partial or noisy patterns.
- ◆ ANNs can process information in parallel, at high speed, and in a distributed manner.

2.7 Backpropagation Learning Algorithm

There are some algorithms for training multilayered ANNs. Among them the backpropagation (BP) algorithm is the most prolific one. In MCA, BP algorithm is used in a modified manner. So, this algorithm is described in detail in this section.

The BP learning algorithm involves two phases. During the first phase the input is presented and propagated forwarder through the ANN to compute the output value for each node. This output is then compared with the targets to generate an error signal δ for each of the output node. The second phase involves a backward pass through the ANN during which the error signal δ is passed to each node in the ANN and the appropriate weight changes are made. This second backward pass allows the recursive computation of δ as indicated above. The first step is to compute δ for each of the output nodes. This simply the difference between the actual and desired output values times the derivative of the squashing function. Then the weight changes for all connections that feed into the final layer can be computed. After this is done, then compute δ 's for all nodes in the

penultimate layer. This propagates error back one layer and same process can be repeated for every layer.

Let us consider an input vector $X_p = (x_1, x_2, \dots, x_n)$, is applied to the input layer of the ANN. The “p” subscript refers to the pth training vector. The input nodes distribute the values to the hidden layer nodes. The net input to the j^{th} hidden node is,

$$net_{pj}^h = \sum_{i=1}^N w_{ji}^h x_{pi} + \theta_j^h \quad (2.11)$$

Where, w_{ji}^h is the weight of the connection from i^{th} input node to j^{th} hidden node and θ_j^h is the bias term. The “h” subscript refers to the quantity on the hidden layer. Assuming that activation of the node is equal to the net input; then the output of the node is,

$$i_{pj} = f_j^h(net_{pj}^h) \quad (2.12)$$

The equations of the output nodes are,

$$net_{pk}^o = \sum_{j=1}^L w_{kj}^o i_{pj} + \theta_k^o \quad (2.13)$$

$$o_{pk} = f_k^o(net_{pk}^o) \quad (2.14)$$

◆ Update of Output-Layer Weights

The error at a single output node is defined as $\delta_{pk} = y_{pk} - o_{pk}$, where the subscript “p” refers to the p^{th} training vector, and “k” refers to the k^{th} output node. In this case y_{pk} is the desired output and o_{pk} is the actual output of the k^{th} node. The error to be minimized is the sum of the squares of the errors for all output nodes:

$$E_p = \frac{1}{2} \sum_{k=1}^M \delta_{pk}^2 \quad (2.15)$$

To determine the direction in which the change of weights, the negative of the gradient of E_p , ∂E_p , with respect to weights, w_{kj} is calculated. Then the values of the weights can be adjusted such that the total error can be reduced. It is often usual to think of E_p as a surface in the weight space.

From Eq. (2.15) and the definition of δ_{pk}

$$E_p = \frac{1}{2} \sum_{k=1}^M (y_{pk} - o_{pk})^2 \quad (2.16)$$

$$\frac{\partial E_p}{\partial w_{kj}^o} = -(y_{pk} - o_{pk}) \frac{\partial f_k^o}{\partial (net_{pk}^o)} \frac{\partial (net_{pk}^o)}{\partial w_{kj}^o} \quad (2.17)$$

Where, Equation (2.16) is used for the output value, o_{pk} , and the chain rule for partial derivatives. The last factor of Equation (2.17) is,

$$\frac{\partial (net_{pk}^o)}{\partial w_{kj}^o} = \frac{\partial}{\partial w_{kj}^o} \sum_{j=1}^l w_{kj}^o i_{pj} + \theta_k^o = i_{pj} \quad (2.18)$$

Combining Equations (2.17) and (2.18), the negative gradient,

$$-\frac{\partial E_p}{\partial w_{kj}^o} = (y_{pk} - o_{pk}) f_k^{\prime o} (net_{pk}^o) i_{pj} \quad (2.19)$$

As far the magnitude of the weight change is concerned, it has been taken to be proportional to the negative gradient. Thus the weights on the output layer are updated according to

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \Delta_p w_{kj}^o(t) \quad (2.20)$$

Where,

$$\Delta_p w_{kj}^o = \eta (y_{pk} - o_{pk}) f_k^{\prime o} (net_{pk}^o) i_{pj} \quad (2.21)$$

The factor η is called the learning rate parameter. If the sigmoid function is used then the weight update equation for output node is,

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \eta (y_{pk} - o_{pk}) o_{pk} (1 - o_{pk}) i_{pj} \quad (2.22)$$

By defining output layer error term,

$$\delta_{pk}^o = \eta (y_{pk} - o_{pk}) f_k^{\prime o} (net_{pk}^o) = \delta_{pk} f_k^{\prime o} (net_{pk}^o) \quad (2.23)$$

By combining equations (2.22) and (2.23) the weight update equation becomes,

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \eta \delta_{pk}^o i_{pj} \quad (2.24)$$

◆ Update of Hidden-Layer Weights

The error of the hidden layer is given by,

$$E_p = \frac{1}{2} \sum_k (y_{pk} - o_{pk})^2$$

$$\begin{aligned}
&= \frac{1}{2} \sum_k (y_{pk} - f_k^o(\text{net}_{pk}^o))^2 \\
&= \frac{1}{2} \sum_k (y_{pk} - f_k^o(\sum_j w_{kj}^o i_{pj} + \theta_k^o))^2
\end{aligned} \tag{2.25}$$

The gradient of E_p with respect to hidden layer weights,

$$\begin{aligned}
\frac{\partial E_p}{\partial w_{ji}^h} &= \frac{1}{2} \sum_k \frac{\partial}{\partial w_{ji}^h} (y_{pk} - o_{pk})^2 \\
&= - \sum_k (y_{pk} - o_{pk}) \frac{\partial o_{pk}}{\partial(\text{net}_{pk}^o)} \frac{\partial(\text{net}_{pk}^o)}{\partial i_{pj}} \frac{\partial i_{pj}}{\partial(\text{net}_{pj}^h)} \frac{\partial(\text{net}_{pj}^h)}{\partial w_{ji}^h}
\end{aligned} \tag{2.26}$$

Each of the factors of Equation (2.26) can be calculated explicitly from previous equations. The result is,

$$\frac{\partial E_p}{\partial w_{ji}^h} = - \sum_k (y_{pk} - o_{pk}) f_k^{\prime o}(\text{net}_{pk}^o) w_{kj}^o f_j^{\prime h}(\text{net}_{pj}^h) x_{pi} \tag{2.27}$$

The hidden layer weights update in proportion to negative of the Equation (2.27):

$$\Delta_p w_{ji}^h = \eta f_j^{\prime h}(\text{net}_{pj}^h) x_{pi} \sum_k (y_{pk} - o_{pk}) f_k^{\prime o}(\text{net}_{pk}^o) w_{kj}^o \tag{2.28}$$

By using Equation (2.23),

$$\Delta_p w_{ji}^h = \eta f_j^{\prime h}(\text{net}_{pj}^h) x_{pi} \sum_k \delta_{pk}^o w_{kj}^o \tag{2.29}$$

Every weight update on the hidden layer depends on all error terms, δ_{pk}^o , on the output layer. The known errors on the output layers are propagated back to the hidden layer to determine the appropriate weight changes on that layer. By defining hidden layer error term,

$$\delta_{pj}^h = f_j^{\prime h}(\text{net}_{pj}^h) \sum_k \delta_{pk}^o w_{kj}^o \tag{2.30}$$

So the weight update equation becomes analogous to those for the output layer:

$$w_{ji}^h(t+1) = w_{ji}^h(t) + \eta \delta_{pj}^h x_{pi} \tag{2.31}$$

The value of η is commonly chosen between 0.25 and 0.75 by the ANN user, and usually reflects the *rate of learning* to ensure that the ANN will settle to a solution.

2.8 Constructive Algorithms

Although BP algorithm is the most widely used learning algorithm, it has a severe drawback, that is, the number of hidden layers and the number of nodes in each hidden layer should be predefined. To overcome this difficulty, many algorithms that construct a ANN dynamically have been proposed. Constructive (or generative) algorithms offer an attractive framework for the incremental construction of near-minimal ANN architectures. These algorithms start with a small network (usually a single hidden node) and dynamically grow the network by adding and training nodes as needed until a satisfactory solution is found. The most well known constructive algorithms are dynamic node creation (DNC), the cascade correlation (CasCor) algorithm and the cascade neural network design algorithm (CNND). The last two involves training of multiple hidden layered ANNs and has some similarities with the MCA. The following sections consist of short description of these algorithms.

2.9 Cascade Correlation Learning Architecture (CasCor)

Cascade-Correlation is a new architecture and supervised learning algorithm for artificial neural networks. Instead of just adjusting the weights in a ANN of fixed topology, Cascade-Correlation begins with a minimal network, then automatically trains and adds new hidden units one by one, creating a multi-layer structure. Once a new hidden unit has been added to the ANN, its input-side weights are frozen. This unit then becomes a permanent feature-detector in the ANN, available for producing outputs or for creating other, more complex feature detectors.

2.9.1 Reasons behind the development of CasCor

The Cascade-Correlation learning algorithm was developed in an attempt to overcome certain problems and limitations of the popular back-propagation (BP) learning algorithm. The most important of these limitations is the slow pace at which BP learns from examples. Even on simple benchmark problems, a back-propagation network may require many thousands of epochs to learn the desired behavior from examples. There are two major problems that contribute to the slowness. These are called the *step-size problem* and the *moving target problem*.

◆ *The step-size problem*

The step-size problem occurs because the standard back-propagation method computes only $\partial E / \partial w$, the partial first derivative of the overall error function with respect to each weight in the ANN. Given these derivatives, one can perform a gradient descent in weight space, reducing the error with each step. In a practical learning system, however, taking infinitesimal steps is not feasible; for fast learning, possible largest steps should be taken. Unfortunately, if the step size is too large; the ANN will not reliably converge to a good solution.

To overcome this problem one can incorporate “momentum” term in the learning or can use the “Quickprop” algorithm [33]. The CasCor algorithm uses the latter.

◆ *The moving target problem*

A second source of inefficiency in back-propagation learning is called the moving target problem. Briefly stated, the problem is that each unit in the interior of the ANN is trying to evolve into a feature detector that will play some useful role in the ANN’s overall computation, but its task is greatly complicated by the fact that all the other units are changing at the same time. The hidden units in a given layer of the net cannot communicate with one another directly; each unit sees only its inputs and the error signal propagated back to it from the ANN’s outputs. The error signal defines the problem that the unit is trying to solve, but this problem changes constantly. Instead of a situation in which each unit moves quickly and directly to assume some useful role, a complex dance among all the units is seen that takes a long time to settle down.

One way to combat the moving-target problem is to allow only a few of the weights or units in the ANN to change at once, holding the rest constant. The cascade-correlation algorithm uses an extreme version of this technique, allowing only one hidden unit to evolve at any given time.

2.9.2 CasCor

Cascade-Correlation combines two key ideas: The first is the *cascade architecture*, in which hidden units are added to the ANN one at a time and do not change after they have been added. The second is the learning algorithm, which creates and installs the new

hidden units. For each new hidden unit, the magnitude of the *correlation* between the new unit's output and the residual error signal is maximized.

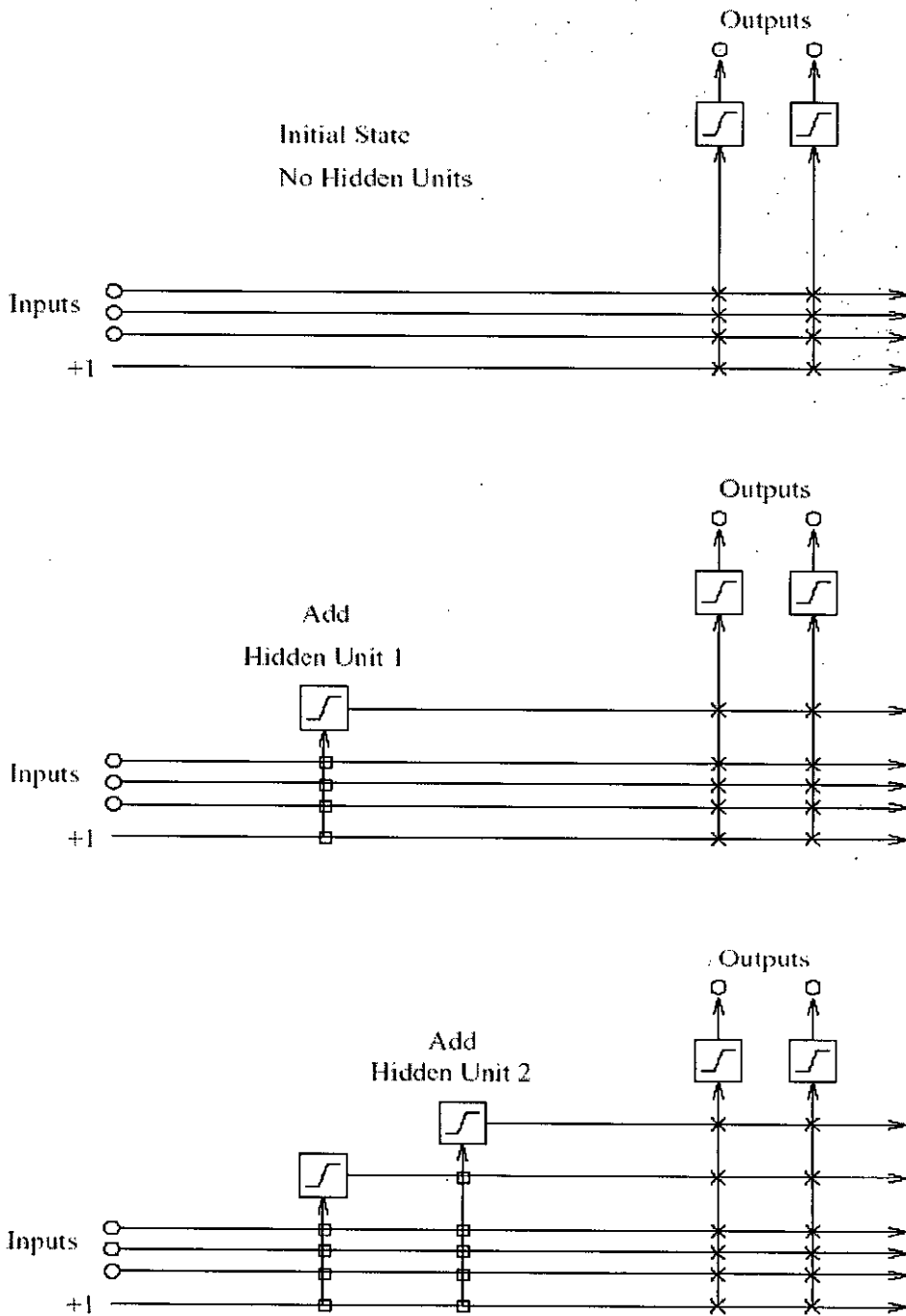


Fig. 2.8 The Cascade architecture of CasCor, initial state and after adding two hidden units. The vertical lines sum all incoming activation. Boxed connections are frozen, X connections are trained repeatedly.

The cascade architecture is illustrated in Fig. 2.8. It begins with some inputs and one or more output units, but with no hidden units. The number of inputs and outputs is dictated by the problem and by the I/O representation the experimenter has chosen. Every input is connected to every output unit by a connection with an adjustable weight. There is also a *bias* input, permanently set to +1.

Hidden units are added to the ANN one by one. Each new hidden unit receives a connection from each of the ANN's original inputs and also from every pre-existing hidden unit. The hidden unit's input weights are frozen at the time the unit is added to the net; only the output connections are trained repeatedly. Each new unit therefore adds a new one-unit "layer" to the ANN; unless some of its incoming weights happen to be zero.

The learning algorithm begins with no hidden units. The direct input-output connections are trained as well as possible over the entire training set. With no need to back-propagate through hidden units, one can use the Widrow-Hoff or "delta" rule, the perceptron learning algorithm, or any of the other well-known learning algorithms for single-layer networks. Here, the quickprop algorithm is used to train the output weights. With no hidden units, quickprop acts essentially like the delta rule, except that it converges much faster.

At some point, this training will approach an asymptote. When no significant error reduction has occurred after a certain number of training cycles, the ANN is run one last time over the entire training set to measure the error. If the ANN's performance is satisfactory, training is stopped; if not, there must be some residual error that can be reduced further. This is achieved by adding a new hidden unit to the ANN, using the unit-creation algorithm described below. The new unit is added to the net, its input weights are frozen, and all the output weights are once again trained using quickprop. This cycle repeats until the error is acceptably small.

To create a new hidden unit, begin with a *candidate unit* that receives trainable input connections from all of the ANN's external inputs and from all pre-existing hidden units. The output of this candidate unit is not yet connected to the active ANN. A number of passes is run over the examples of the training set, adjusting the candidate unit's input weights after each pass. The goal of this adjustment is to maximize S , the sum over all

(Fig. 2.9). In such architecture, the first hidden ($H1$) layer receives only ANN inputs (I), while the second hidden ($H2$) layer receives I plus the outputs (X) of the $H1$ layer. The output layer receives signals from the $H1$ and $H2$ layers.

The major steps of CNNDA are as follows:

Step 1. Create an initial ANN architecture. The initial architecture has three layers, i.e. an input, an output, and a hidden layer. The number of nodes in the input and output layers is the same the number of inputs and outputs of the problem. Initially, the hidden layer contains only one node. Randomly initialize connection weights between input layer to hidden layer and hidden layer to output layer within a certain range.

Step 2. Train the ANN using the BP. Stop the training process if the training error E does not significantly reduce in the next few iterations. The assumption is that the ANN has inappropriate architecture. The training error E is calculated according to the following equation:

$$E = 100 \frac{O_{\max} - O_{\min}}{NS} \sum_{s=1}^S \sum_{i=1}^N (Y_i(s) - Z_i(s))^2 \quad (2.36)$$

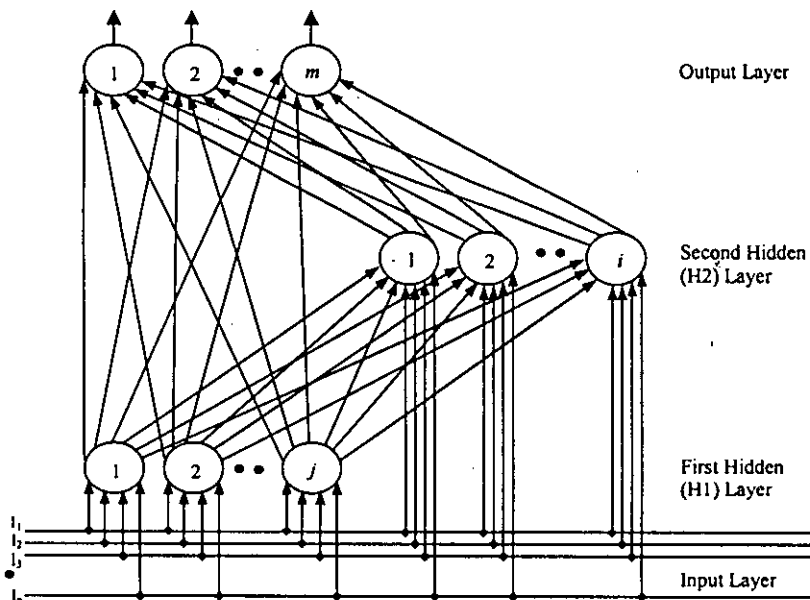


Fig: 2.9 A two hidden layer multilayer perceptron (MLP)

where o_{\max} and o_{\min} are the maximum and minimum values of the target outputs in the problem representation, N is the number of output nodes, S is the total number of examples in the training set, $Y_s(s)$ and $Z_s(s)$ are, respectively, the actual and desired outputs of node i for training data s . The advantage of the above error equation is that it is less dependent on the size of the training set and the number of output nodes.

Step 3. If the value of E is acceptable, go to step 12. Otherwise, continue.

Step 4. Compute the number of hidden layers in the ANN. If this number is two (i.e. the user-defined maximum-hidden-layer number), go to step 6. Otherwise, continue.

Step 5. Create a new hidden layer, i.e. a second hidden ($H2$) layer, with only one node. Initialize connection weights of that node in the same way as described in step 1. Go to step 2.

Step 6. Compute the contribution C and the number of fan in connections f of each node in the $H2$ layer. The $C_i(n)$ of the i -th node at any iteration n is E_i/E . Here E_i is the error of the ANN excluding node i and computed according to Eq. (2.39).

Step 7. Compare the values of $C_i(n)$ and $f_i(n)$ with their previous values $C_i(n - \tau_1)$ and $f_i(n - \tau_1)$, respectively. If $C_i(n) \leq C_i(n - \tau_1)$ and $f_i(n) - f_i(n - \tau_1) = M$, continue. Otherwise, go to step 9. Here τ_1 and M are user-defined positive integer numbers.

Step 8. Freeze the fan-in capacity of the i -th node. That means the i -th node will not receive any new input signals in the future when new nodes are added in the $H1$ layer. Notice that generally the i -th node and all other nodes in the $H2$ layer receive signals from all nodes in the $H1$ layer. Mark the i -th node with F .

Step 9. Compare the output $X(n)$ of each node in the $H1$ layer and F marked nodes in the $H2$ layer with their previous values $X(n - \tau_2)$. Here τ_2 is the user-defined positive integer number. If $X(n) \approx X(n - \tau_2)$, continue. Otherwise, go to step 11.

Step 10. Temporarily freeze input connection weights, keeping weight values fixed for some time, for any nodes in the $H1$ layer and F marked nodes in the $H2$ layer whose $X(n) \approx X(n - \tau_2)$.

Step 11. Add one node in the $H1$ or $H2$ layer. If adding a node in the $H1$ layer produces a larger size ANN (in terms of connections) than does adding a node in the $H2$ layer, then add one node in the $H1$ layer. Otherwise, add one node in the $H2$ layer. In the CNNDA, the addition of a node in any layer is done by splitting an existing node in that layer. Go to step 2.

Step 12. According to contribution C , delete the least contributory node from the ANN and unfreeze the input connections (if any connections are frozen) of one hidden node. Train the pruned ANN by the BP. If the ANN converges again, delete one node and unfreeze the input connections of one node. Continue this process until the ANN no longer converges.

Step 13. Decide the number of connections to be deleted by generating a random number between 1 and the user-defined maximum number. Calculate the approximate importance of each connection in the ANN by the nonconvergent method. According to calculated importance, delete a certain number of connections and unfreeze the same number of connections (if any connections are frozen) from the ANN. Train the pruned ANN by the BP. Continue this process until the ANN no longer converges. The last ANN before converge ends is the final ANN.

As a constructive algorithm, CNNDA offers some advantages over previous constructive algorithms. They are –

- ◆ Lower connections than CasCor architecture
- ◆ Four-layered ANN
- ◆ Greater generalization ability

The disadvantages are—

- ◆ Complex design procedure
- ◆ Layer size is restricted to four
- ◆ May create unnecessary complex structure for simple problems

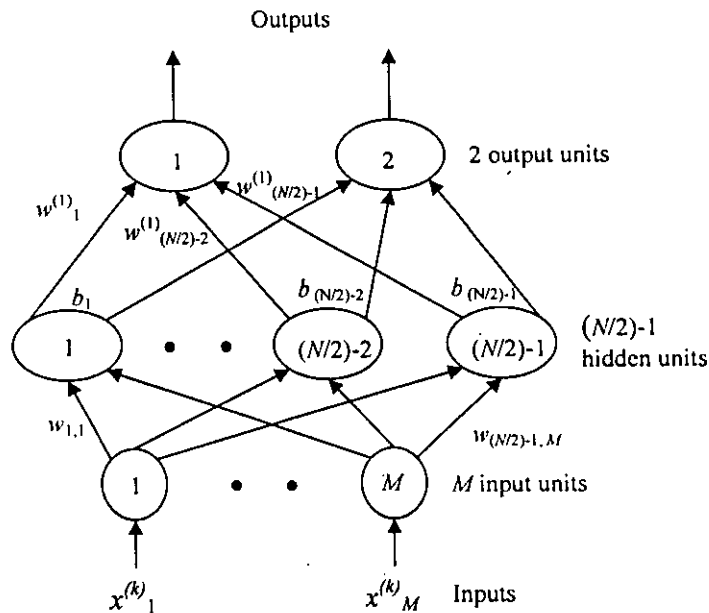
2.11 Four Layered versus Three Layered ANNs

Most of the constructive algorithms designed consist of a single hidden layer. Very few algorithms involve multiple hidden layers. It has been a controversial issue over time if multiple hidden layered ANNs require less parameter and if they work faster than single hidden layered ANNs. In this section, a mathematical proof is given to justify the superiority of four layered ANNs (two hidden layers) over three layered ANNs (single hidden layer).

It has already been proven in [34] that a three-layered feedforward ANN with $N-1$ hidden units can give any N input-target relations exactly. Based on this a four-layered ANN is constructed and is found to give any N input-target relations with a negligibly small error using only $(N/2) + 3$ hidden units.

2.11.1 Construction of a Four Layered ANN

For any N input-target pairs $(x^{(k)}, t^{(k)})$; $x^{(k)} \in R^M$, $t^{(k)} \in R$, it will be shown that, using significantly fewer hidden units than a three-layered ANN, a four-layered ANN can give the input-target relations with an arbitrarily small error. For simplicity, N is taken as



even.

Fig: 2.10 Three-layered subnetwork

Let us first consider the subnetwork shown in Fig. 2.10 and the following subtargets:

$$t^{(k)} = t^{(k)}/C + 0.5 \in (0, 1) \quad (k = 1, 2, \dots, N) \quad (2.32)$$

Where C is a positive constant and is determined so that $t^{(k)} \in (0, 1)$.

The subnetwork has M linear input units, $(N/2) - 1$ hidden units and two output units: The output units, in this case, are nonlinear units (i.e., each output unit has a sigmoid function). The output units are numbered one and two. The link weights between the input layer and the hidden layer are set using random numbers.

It is obvious that the finite N input vectors can be separated into two groups consisting of $N/2$ input vectors with a hyperplane in the input space, Let us denote these two input vector groups as V_1 and V_2 and assume that the index; k for the subtargets is reindexed so that $t^{(k)}$ ($k = 1, 2, \dots, N/2$) $\in V_2$. By adjusting the bias values of the hidden units in the subnetwork, it is possible for an $N/2 \times N/2$ matrix, O , formed by the hidden layer outputs with all of the vectors of V_1 being inputs to the subnetwork, to be made full-rank. Thus, one of the output units, for example, unit 1, can assign each input vector of V_1 to the corresponding subtarget value. The bias value of output unit 1 and the link weights between the hidden layer and output unit 1 are determined by solving the following equation:

$$OW^{(1)} = u, \quad u_k = s^{-1}(t^{(k)}) \quad (k = 1, 2, \dots, N/2) \quad (2.33)$$

Where, s^{-1} is the inverse function of a sigmoid function, $W_1^{(1)}$ is the bias value of output unit 1, and $W_i^{(1)}$ ($i = 2, 3, \dots, N/2$) are the link weights from the $(i - 1)$ th hidden unit to output unit 1.

Let us consider the V_2 inputs. Since the bias values of the hidden units are adjusted for V_1 inputs, there is no guarantee that another $N/2 \times N/2$ matrix, O' , which is formed by the hidden layer outputs with all of the vectors of V_2 being inputs to the subnetwork, will be full-rank, or invertible.

Let us write the bias values tuned for the V_1 inputs as follows:

$$B = (b_1, b_2, \dots, b_{(N/2)-1})^t \quad (2.34)$$

Assume that O' is not full-rank. In order to make O' full-rank, each bias value of the hidden units should be chosen from any interval of R . Hence, for an arbitrarily small $\epsilon > 0$, one can choose B' , which makes O' full-rank.

$$B' = (b'_1, b'_2, \dots, b'_{(N/2)-1})', \quad (2.35)$$

$$b'_i \in [b_i - e, b_i + e], \quad (i = 1, 2, 3, \dots, (N/2) - 1)$$

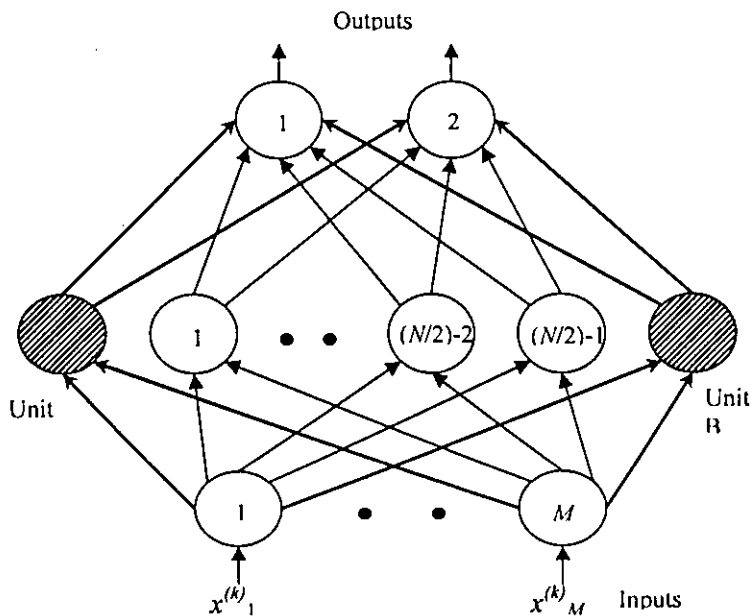


Fig: 2.11 Units A and B

Since the determinant of O is a continuous function of the bias values and is not zero at B , if e is set sufficiently small, the determinant of O at B' does not become zero. Thus, it has been shown that one can always choose bias values that make both O and O' full-rank at the same time. Hence, output unit 2 can also assign each input vector of V_2 to the corresponding subtarget value. The bias as value of output unit 2 and the link weights between the hidden layer and output unit 2 are determined in the same way. At this stage, output units 1 and 2 produce incorrect outputs for the V_2 inputs and the V_1 inputs, respectively.

Now let us add two hidden units to this subnetwork and denote these units as unit A and unit B, which are shown in Fig. 2.11. Like the other, hidden units, these units are fully connected from the input units. However, unit A is connected only to output unit 1 and unit B is connected only to output unit 2.

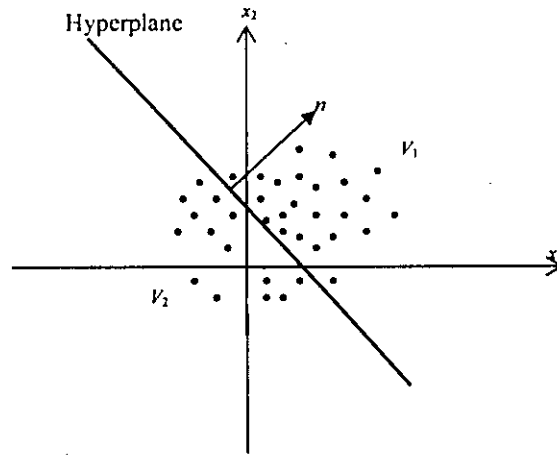


Fig: 2.12 Input space and separating hyperplane (two-dimensional case)

Let us consider unit A. By setting the link weight vector, W_A , which is the link weights from the input layer to unit A, to be equal to Tn , where n is a normal vector of the hyperplane described above and T is a large positive constant and by adjusting the bias value of Unit A so that outputs of unit A become 0.5 with vectors on the hyperplane being inputs to the subnetwork (Fig. 2.12), unit A can be made a detector for V_1 and V_2 (i.e., unit A produces almost zero and almost one for the V_1 inputs and the V_2 inputs, respectively). Here the direction of the normal vector should be interpreted appropriately. Setting the link weight from unit A to output unit 1 to be $-U$, where U is also a large positive constant, output unit 1 receives from unit A:

1. Small negative values, $-e_i$ ($e_i > 0$) for the V_1 inputs
($i = 1, 2, \dots, N/2$);
2. Large negative values, $-E_i$ ($E_i > 0$) for the V_2 inputs
($i = (N/2) + 1, (N/2) + 2, \dots, N$).

One can make E_i and e_i arbitrarily large and small, respectively, by setting T and U sufficiently large. E_i goes to infinity and e_i goes to zero. Thus, for an input vector of V_2 , unit A is able to send an arbitrarily large negative value to output unit 1, and all the other hidden units send fixed constant values. Since output unit 1 has a sigmoid function, the outputs of output unit 1 can be arbitrarily close to zero for the V_2 inputs. However, for the V_1 inputs, since unit A is able to send a value that is arbitrarily close to zero to output unit 1, the outputs of output unit 1 can be arbitrarily close to the corresponding subtarget values. In the same way, unit B can be tuned for output unit 2. A ANN is constructed whose performance is as follows.

1. *For the V_1 inputs:* The outputs of output unit 1 are arbitrarily close to the corresponding subtarget values and the outputs of output unit 2 are arbitrarily close to zero.
2. *For the V_2 inputs:* The outputs of output unit 1 are arbitrarily close to zero and the outputs of output unit 2 are arbitrarily close to the corresponding subtarget values.

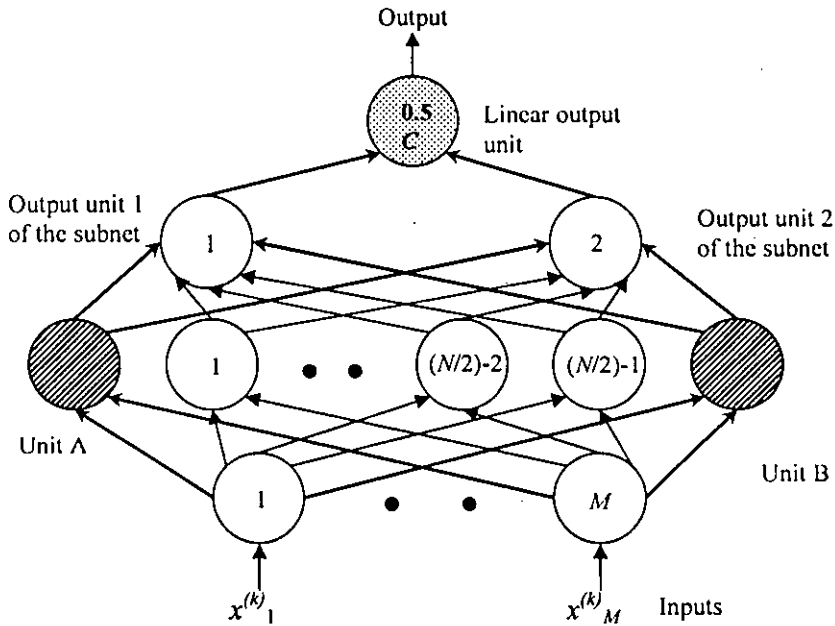


Fig: 2.13 Complete four-layered feedforward ANN

To complete the construction, let us add one more linear unit, i.e., the final output unit. The complete four-layered ANN is shown in Fig. 2.13. This output unit is fully connected from output units 1 and 2, of the subnetwork. The link weights from output units 1 and 2 to the output unit are set to C of (2.32), and the bias value of the output unit is set to $0.5C$ in order to restore the original target values. Accordingly, the four-layered ANN can produce almost the same corresponding target value for each of the N input vectors. The error between the target value and the actual ANN output can be made arbitrarily small. Hence, it has been shown that a four-layered feedforward ANN with $(N/2) + 3$ hidden units can give any N input-target relations with an arbitrarily small error.

A four-layered feedforward ANN is constructed with $(N/2) + 3$ hidden units and shown that such an ANN can give any N input-target relations with an arbitrarily small error.

This clearly shows the difference in capabilities between a four-layered feedforward ANN and a three-layered feedforward ANN and indicates the superiority of the four-layered ANN for the training of input-target pairs. As a result, it is always better to use an ANN consists of multiple hidden layer to optimize the performance. After this finding, the cascade correlation algorithm, which was proposed in 1991, became very famous and many variants of cascade correlation algorithm were developed.

2.12 Conclusion

In the above sections, some discussions were made about some prolific constructive algorithms and a mathematical proof on capabilities of four layered ANNs over three layered ANNs. All or many of them suffer from some problems, such as- ANN designed for highly nonlinear problems cannot solve less nonlinear problems accurately and vice versa, most of them are difficult for VLSI implementation, generalization abilities of most of them are not very convincing, designing algorithms are complicated in most of the cases and in several algorithms, many user define parameters are necessary.

Considering the above mentioned problems, the proposed algorithm (MCA) ought to be simple but efficient enough.

MULTILAYERED CONSTRUCTIVE ARCHITECTURE

3.1 Introduction

In the modern era of ANNs, finding the optimum architecture is an important issue. Different constructive algorithms have been developed during the last two decades to optimize the architecture of ANNs. Most of the algorithms were used to determine the optimum architecture for a three layered ANNs. Many of them are complex in nature and require a large number of user defined parameters. This chapter proposes a new constructive algorithm called multilayered constructive architecture (MCA) for designing and training single and multiple hidden layered ANNs. MCA uses incremental training, in association with negative correlation criterion, to design ANN architectures for classification tasks. It is the first algorithm, to our best knowledge, that creates ANNs combined of both layered and cascaded architectures. A unique hidden layer stopping criterion named negative correlation criterion is used in MCA to stop the growth of the first hidden layer. The following sections elaborately describe MCA, its advantages and difference with other existing algorithms.

3.2 MCA

MCA uses incremental training in association with negative correlation criterion to determine ANN architectures for classification tasks. Individual neurons are added to the hidden layer(s) one by one in a constructive fashion. MCA first tries to solve given the problem with a three layered i.e. single hidden layered ANN architecture by adding neurons in this layer by a strictly layered fashion. If necessary, then more neurons are added above the first hidden layer in a cascaded manner, creating a multiple hidden layered architecture.

The major steps of MCA are summarized in Fig. 3.1, which are explained further as follows.

Step 1. Create an initial ANN architecture. The initial architecture has three layers, i.e. an input layer, an output layer, and a hidden layer $H1$. The number of neurons in the input and output layers is the same as the number of inputs and outputs of the problem,

respectively. Initially, the hidden layer contains only one neuron. Randomly initialize connection weights between input layer to hidden layer and hidden layer to output layer within a certain range.

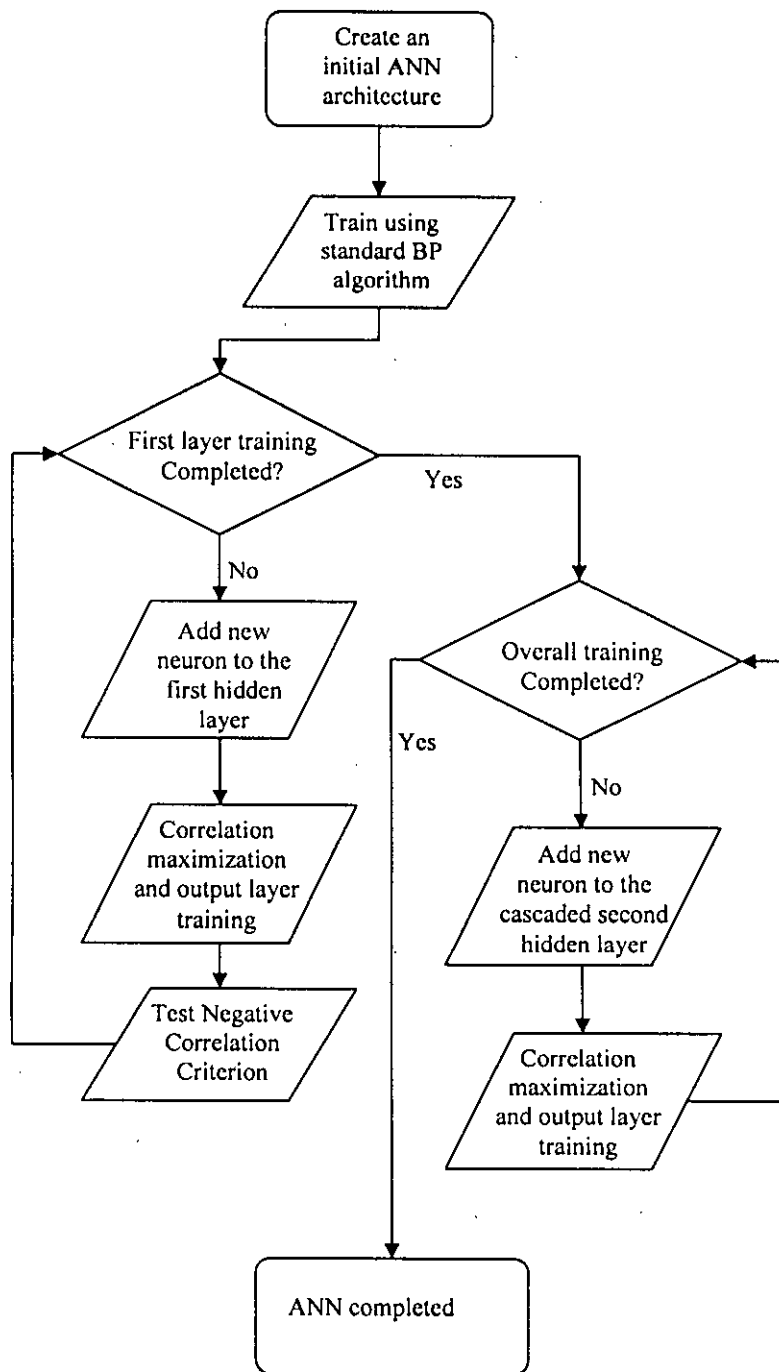


Fig 3.1 Flowchart of MCA

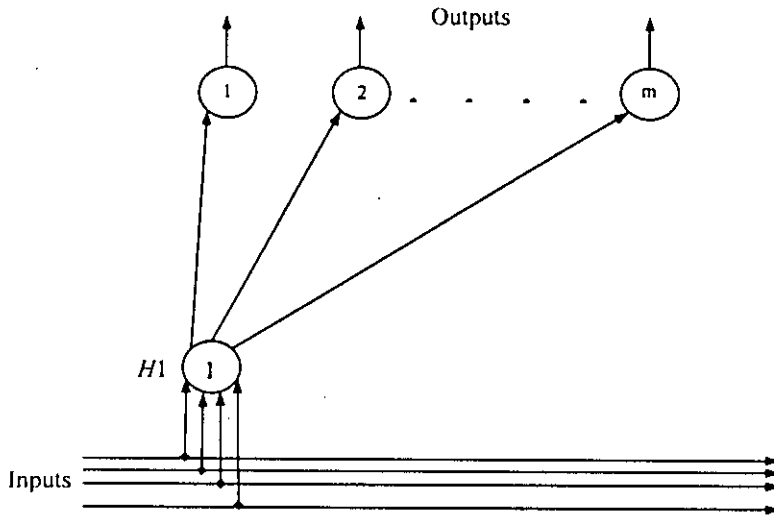


Fig 3.2 Initial ANN

Step 2. Train the network by using BP algorithm as described in Section 2.7. Stop the training process if the residual error E does not significantly reduce in the next few iterations. Freeze the fan-in capacity of the hidden neuron(s). This means that the neuron(s) will not receive any new input signals when new neurons are added in the future. In this work, the residual error E is calculated according to the following equation.

$$E = \frac{1}{2} \sum_{k=1}^m (y_k - o_k)^2 \quad (3.1)$$

Where, y_k and o_k are the desired output and the actual output of output neuron k , respectively. The total number of output neurons is denoted by m . If the value of E is acceptable, go to step 10. Otherwise, continue.

Step 3. Insert a neuron in parallel to the first hidden neuron in the $H1$ layer as shown in Fig. 3.3. The hidden neuron receives a connection from each of the input layer neurons but its output side remains floating, i.e. not connected to the output layer neurons.

Step 4. Train the input side weights of the newly added neuron in a gradient ascent manner to maximize the correlation between its output and the overall difference error d calculated according to the following equation for the k th output neuron.

$$d_k = y_k - o_k \quad (3.2)$$

Stop the training of the newly added neuron when correlation is almost flattened i.e. does not increase significantly for some epochs. The correlation between any neuron's output and d is defined by

$$S = \sum_{k=1}^m \sum_{p=1}^P |(V_p - \bar{V})(d_{pk} - \bar{d}_k)| \quad (3.3)$$

Where, P denotes the total number of patterns in the training set. V_p is the output of the newly added hidden neuron and d_{pk} is the difference error of output neuron k for pattern p . \bar{V} and \bar{d}_k are the pattern averaged value of V_p and d_{pk} respectively.

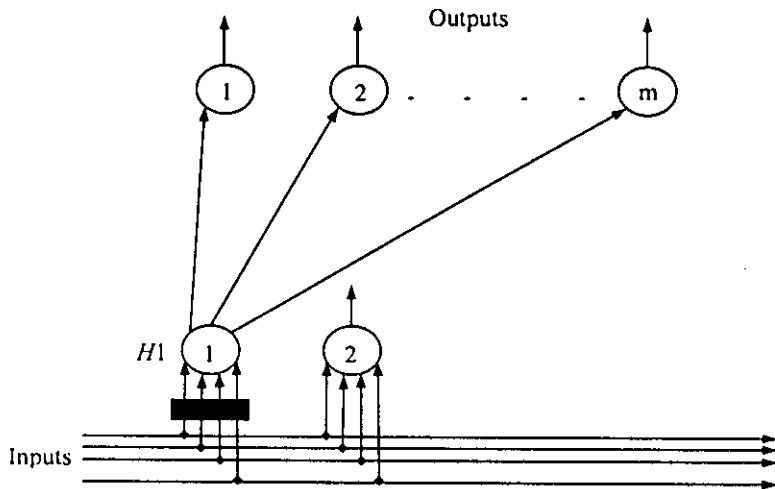


Fig 3.3 Add new neuron in the $H1$ layer (Black box indicates 'frozen' connections)

Step 5. Connect the output of the neuron with all the output layer neurons. Freeze the fan-in capacity of the neuron. Train all the output weights using modified single layer BP rule, as described in Section 3.2.1 to minimize the residual error E .

Step 6. If the added neuron does not reside in $H1$, go to step 8. Otherwise, continue.

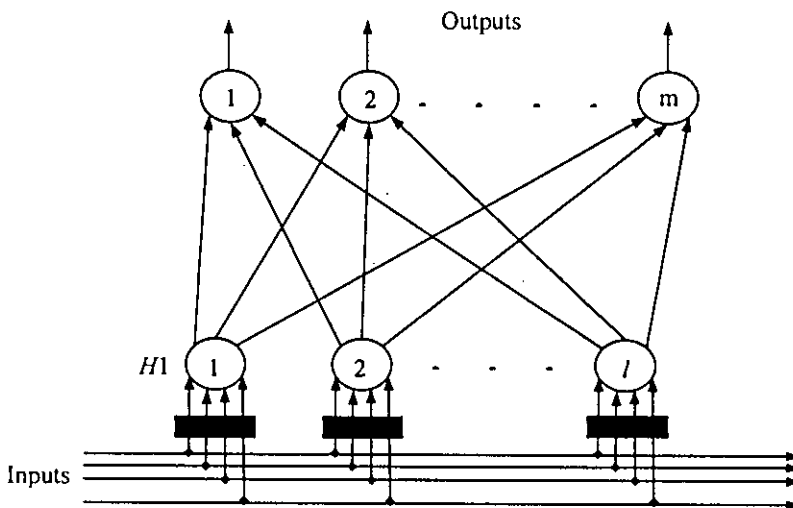


Fig. 3.4 ANN with $H1$ layer completed

Step 7. Test the growth of $H1$ layer by using the negative correlation criterion as per Section 3.2.3. If the criterion is met, the layer is completed (Fig. 3.4), go to step 8. Otherwise, go to step 3.

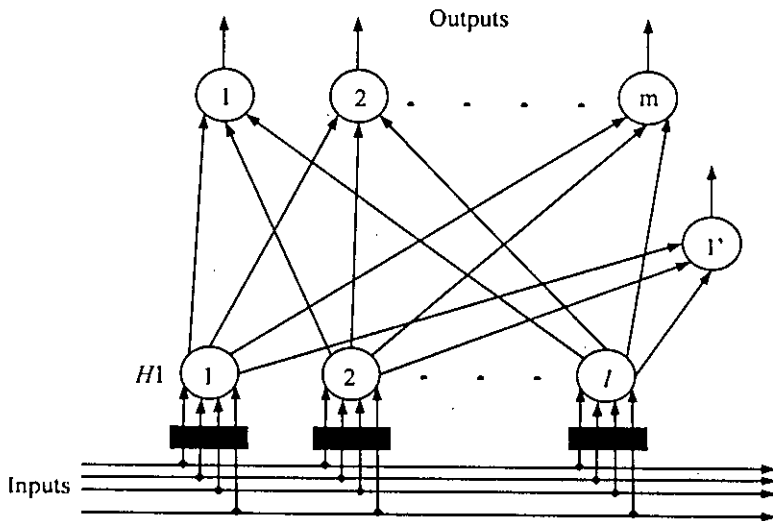


Fig. 3.5 First neuron in cascade above $H1$ layer

Step 8. If the value of E is acceptable, go to step 10. Otherwise, continue.

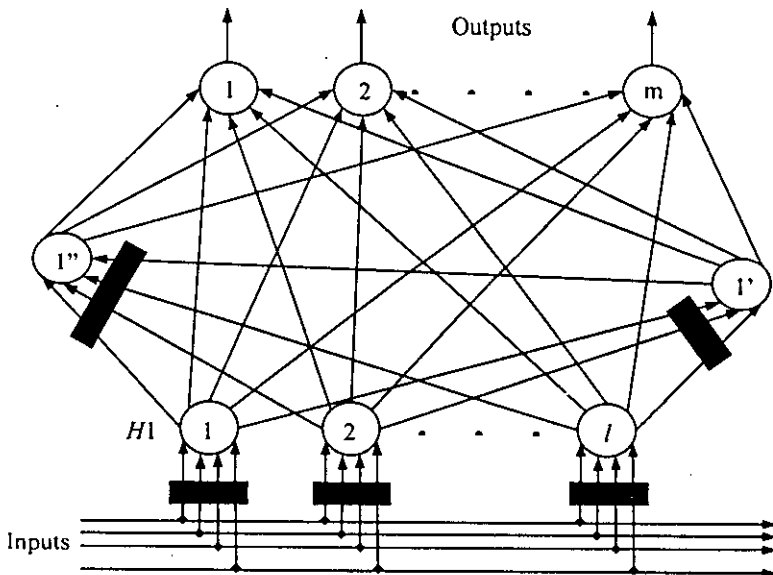


Fig. 3.6 Final ANN with $H1$ hidden layer and two neurons in cascade above it

Step 9. Construct a new hidden layer by adding a new neuron above the existing hidden layer(s) as per Fig. 3.5 and go to step 4. This neuron gets input connections from all of the previous hidden neurons whether they reside on the $H1$ or any other hidden layer.

This will create a cascaded hidden layer with a single neuron above the existing hidden layers. The output of the added neuron will be floating.

Step 10. The ANN obtained is the final ANN for a given problem (Fig. 3.6).

It is seen that the construction process of MCA is little bit complex than existing constructive algorithms in designing ANNs. However, the essence of MCA is that it can construct ANNs with both single and multiple hidden layers depending on the problem complexity. There are some processes and criteria those are incorporated in MCA at different stages. A modified single layer version of BP algorithm is used to update the output layer weights. MCA uses a unique layer stopping criterion to stop the growth of the first hidden layer. Correlation maximization process is used to add new hidden neurons. All these processes are described briefly in the following subsections.

3.2.1 Modified Single Layer BP Algorithm

In MCA, for updating all the output layer weights, a little modified version of this conventional BP rule is used. The conventional BP rule is described in Section 2.7. Some modifications like 'batch' processing in lieu of 'sequential' processing and use of a 'momentum' term are done to the single layer BP learning algorithm.

The output layer weight update equations (Eqns. 2.20, 2.21) of the BP learning algorithm is as follows—

Weight update factor—

$$\Delta_p w_{kj}^o = \eta (y_{pk} - o_{pk}) f_k^{\prime} (net_{pk}^o) i_{pj} \quad (3.4)$$

Updated weight—

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \Delta_p w_{kj}^o(t) \quad (3.5)$$

Where, the symbols indicate their usual meanings as per Section 2.7.

To modify the learning process, (3.4) is divided into two terms according to the following—

Weight update factor—

$$\Delta_h w_{kj}^o = \sum_p (y_{pk} - o_{pk}) f_k^{\prime} (net_{pk}^o) i_{pj} \quad (3.6)$$

In this equation (3.6), summation of all the weight updates for each pattern is taken. This cumulative weight update refers to as 'batch' processing.

Modified weight update factor—

$$\Delta w_{kj}^o(t) = \eta \Delta_n w_{kj}^o + \alpha \Delta w_{kj}^o(t-1) \quad (3.7)$$

Where, α is the momentum term, typically greater than unity. The equation for modified weight remains unchanged (3.5). It is therefore clear that a 'copy' of previous weight change is kept in memory in order to 'accelerate' the learning.

3.2.2 The Correlation Maximization Process

The correlation S between any neuron's output and overall d is defined by Eqn. (3.3). The input side weights of any newly added neuron is trained using the correlation maximization process. In order to maximize S , the partial derivative of S with respect to each of the candidate unit's incoming weights (from neuron i to neuron j) w_{ji} must be computed. Similar to the derivation of the BP algorithm, the derivative of S is

$$\frac{\partial S}{\partial w_{ji}} = \sum_{p,k} \sigma_k (d_{pk} - \bar{d}) f_p' X_{pi} \quad (3.8)$$

Where, σ_k is the sign of correlation between the candidate neuron and the k th output neuron, f_p' is the derivative of activation function f of the candidate neuron for pattern p .

A gradient ascent is performed to maximize S , after the computation of $\partial S / \partial w_{ji}$. Consequently, each weight of all the added hidden neuron is updated according to the following equation-

$$w_{ji}(t+1) = w_{ji}(t) + l_r \frac{\partial S}{\partial w_{ji}} \quad (3.9)$$

Where, l_r is the learning rate.

3.2.3 The Negative Correlation Criterion

An automatic stopping criterion is used in MCA to stop the growth of the first hidden layer. It is named negative correlation criterion and is based on the interneuron correlations among the outputs of the neurons in the same hidden layer. When a new neuron is installed in the hidden layer, its correlation with existing neuron(s) in same layer is measured and summed. It is then observed whether this summed value crosses

zero, i.e. changes the initial sign after the addition of new neurons in the same layer. The criterion can be clarified mathematically in the following way.

Correlation between two neuron's outputs is defined by the following equation.

$$N_{j1} = \sum_p (V_{ip} - \bar{V}_i)(R_{jp} - \bar{R}_j) \quad (3.10)$$

Where V_{ip} is the output of any existing neuron i and R_{jp} is the output of newly added neuron j in the layer for pattern p . \bar{V}_i and \bar{R}_j are the pattern averaged value of V_{ip} and R_{jp} respectively.

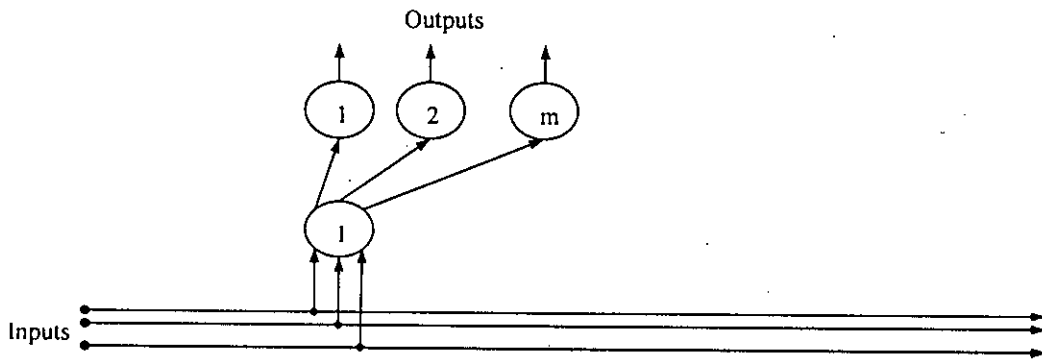


Fig. 3.7 Initial ANN

The summation of correlations SN between neuron 1 and all other neurons in the first hidden layer neuron is defined by-

$$SN = \sum_{j=2}^l N_{j1} \quad (3.11)$$

To explain the mechanism of this criterion, initial ANN with only a single hidden neuron $n1$ is considered (Fig 3.7). Theoretically, if any neuron $n2$ is added in the same ($H1$) layer (Fig. 3.8), the neuron should correlate negatively with $n1$. But in practical, two cases may occur.

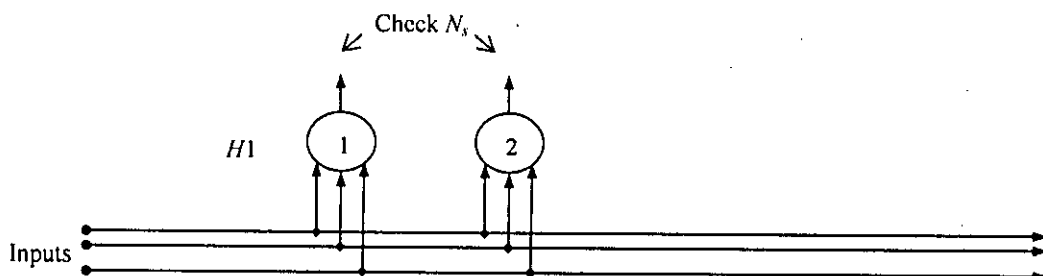


Fig. 3.8 Only first hidden layer neurons are considered after adding $n2$

- ◆ **Case 1:** *Neuron n_2 correlates positively with neuron n_1 .* Calculate the correlation between these neurons according to (3.10) and consider this value to be c_1 . At this instant, SN is calculated according to (3.11) and its value equals to c_1 , which is positive. Now, more neurons are added one by one in this layer as per Fig. 3.9 until SN crosses zero i.e. $SN \leq 0$.

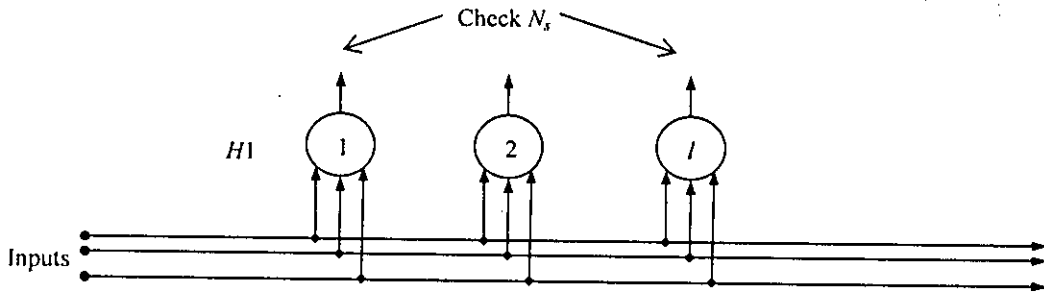


Fig. 3.9 More neurons ($3 \dots l$) are added until SN crosses zero

- ◆ **Case 2:** *Neuron n_2 correlates negatively with neuron n_1 .* Calculate the correlation between these neurons according to (3.10) and consider this value to be $-c_1$. At this instant, SN is calculated according to (3.11) and its value equals to $-c_1$, which is negative. Now, more neurons are added one by one in this layer as per Fig. 3.9 until SN crosses zero i.e. $SN \geq 0$.

The growth of the first hidden layer is stopped if the condition described in Case 1 or Case 2 is met. This means that the given problem can not be solved by single hidden layer. MCA therefore creates one or more cascaded hidden layer to solve the problem. In this manner, the layer stopping criterion can be used to stop the growth of the first hidden layer.

3.3 Advantages of MCA

MCA is an efficient algorithm to design multiple hidden layered ANN which offers the following advantages over other constructive or pruning or combined algorithms. Combined algorithms are those which involve both construction and pruning of network.

- ◆ MCA exhibits lower propagation delay than CasCor and most of its variants. Propagation delay is defined as the longest path that a signal must cross while passing through any ANN. CasCor and its variants like aCasper, modified CasCor, simplified CasCor create a completely cascaded architecture resulting in a long

propagation path, i.e. large propagation delay. On the other hand, MCA creates a combination of layered and cascaded architecture which creates less number of layers and lower propagation delay. The fact can be clarified with Fig. 3.10.

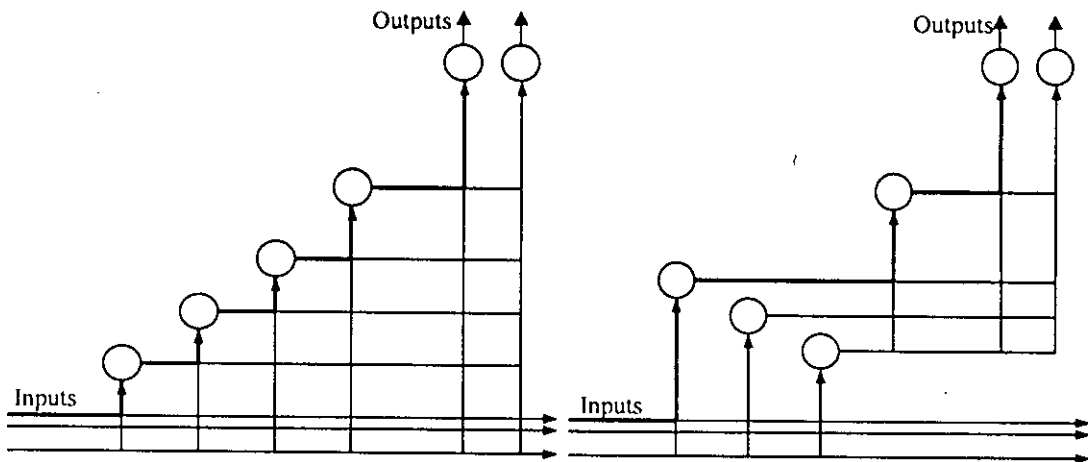


Fig. 3.10 Comparison of propagation delay (bold lines) between CasCor and MCA with 4 hidden neurons. (a) CasCor architecture with all 4 hidden neurons in cascade creating propagation delay path of length 5. (b) MCA with 3 hidden neurons in the same layer and 1 in cascade creating propagation delay path of length 3.

It can be logically proved that MCA exhibits lower propagation delay than CasCor. Let us consider an ANN with n hidden neurons. If the ANN is created by CasCor, these n neurons will create n hidden layers. Now, if the ANN is created by MCA, then the first hidden layer must contain at least 2 hidden neurons, leaving $n-2$ hidden neurons to form cascaded layers. These $n-2$ hidden neurons form $n-2$ hidden layers above the initial hidden layer. So, the resultant hidden layer number will be $n-2+1$, i.e. $n-1$. As the number of hidden layers in MCA is lower than the number of hidden layers in CasCor, the propagation delay is also lower.

- ◆ In most of the constructive algorithms, hidden neurons are added either in strictly layered or in strictly cascaded manner. As a result, the created ANN may cause underfitting or overfitting of data. MCA is independent of these problems due to its architecture combined of layered and cascaded hidden neurons.
- ◆ MCA is preferred to the pruning approach because specifying the initial network is easier in it. In pruning methods, one has to decide how big the initial network must be. Whereas, it is easy to found an initially small network. Since MCA starts with

smaller networks, the computation time is less and they are likely to find smaller networks.

- ◆ MCA uses the negative correlation criterion for stopping the first hidden layer which is very effective and very easy to implement. In other algorithms the architecture is either predefined or found by trial and error method.
- ◆ ANNs created by MCA are easier to implement in VLSI than CasCor and its variants because of lower number of cross connections.

3.4 Differences with Existing Algorithms

There are several algorithms for architecture determination in the realm of ANN. Most of them are constructive, pruning or combination of construction and pruning of network. However, most of these algorithms are used to design three layered ANNs. Very few algorithms are used to design four or more layered ANNs. MCA has some significant differences with these existing algorithms, such as—

- ◆ MCA is an algorithm that can design single or multiple hidden layered ANNs depending on a problem complexity. MCA first tries to solve a given problem with a single hidden layered ANN by adding a number of neurons in the hidden layer. If the problem remains unsolved, MCA then stops the growth of the first hidden layer and creates more hidden layers for solving the problem. Most existing algorithms can design either single or multiple hidden layered ANNs but not the both.
- ◆ The number of hidden layer is one in the case of most of the existing algorithms. The algorithms, which create ANN architecture consisting of two or more hidden layers, require the layer size, i.e. the number of neurons in each hidden layer to be predetermined. On the contrary, MCA neither need to predetermine the number of existing layers nor need to restrict the number of hidden neurons per layer.
- ◆ There is no effective way of stopping the hidden layer growth in the existing algorithms. As a result, either the number of hidden layers or the number of neurons in each hidden layer needs to be restricted. MCA involves a unique layer stopping criterion, named the negative correlation to restrict the number of neurons in the hidden layer.

- ◆ MCA uses 'batch' processing rather than the conventional 'sequential' processing in the single layer BP learning. In sequential processing, every time a pattern is passed through the ANN, the output error is calculated and the weight is updated. But in batch processing, all the patterns are passed through the ANN to calculate the cumulative output error. Although this process slows down the learning, but it can get rid of the 'herd effect' described in [19]. A 'momentum' factor is used to accelerate the batch processing.

3.5 Conclusion

This chapter mainly describes about the algorithm and analyzes the details of the algorithms. The advantages of MCA over existing algorithms, difference with existing algorithms are also stated in this chapter. Simulation and results of MCA on different datasets, comparison with other algorithms and analysis are described in the next chapter.

SIMULATION RESULTS

4.1 Introduction

This chapter evaluates the performance of MCA on several well-known benchmark problems. These problems have been subject of many studies in ANNs and machine learning. Six benchmark classification problems were used to evaluate the performance of MCA. These are the Breast Cancer problem, the Australian Credit Card problem, the Diabetes problem, the Glass problem and two sets of the Heart Disease problems.

The following section contains description of used datasets. Their origin and characteristics are also consulted in this section. Experimental setup and experimental results are discussed in the next section. The subsequent sections involve the analysis of experimental results, comparisons with other similar works, and discussion on comparative results.

4.2 Description of Datasets

The data sets representing all the problems were real world data and obtained from the UCI machine learning benchmark repository. The characteristics of the data sets are summarized in Table 4.1. The detailed descriptions of the data sets are available at <ftp://ics.uci.edu> (128.195.11) in directory/pub/machine-learning-databases [36].

4.2.1 The Breast Cancer Problem

This is a classification problem based on the diagnosis of breast cancer. It tries to classify a tumor as either benign or malignant based on cell descriptions gathered by microscopic examination. Input attributes are for instance the clump thickness, the uniformity of cell size and cell shape, the amount of marginal adhesion, and the frequency of bare nuclei.

This dataset consists of 9 inputs, 2 outputs and 699 examples. All inputs are continuous; 65.5% of the examples are benign. This makes for an entropy of 0.93 bits per example. This dataset was created based on the "breast cancer Wisconsin" problem dataset from the UCI repository of machine learning databases.

4.2.2 The Australian Credit Card Problem

This dataset predict the approval or non-approval of a credit card to a customer. Each example represents a real credit card application and the output describes whether the bank (or similar institution) granted the credit card or not. The meaning of the individual attributes is unexplained for confidence reasons.

There are 51 inputs, 2 outputs, 690 examples in the problem. This dataset has a good mix of attributes: continuous, nominal with small numbers of values, and nominal with larger numbers of values. There are also a few missing values in 5% of the examples. 44% of the examples are positive; entropy 0.99 bits per example. This dataset was created based on the "crx" data of the "Credit screening" problem dataset from the UCI repository of machine learning databases.

4.2.3 The Diabetes Problem

This is one of the most challenging dataset in the realm of classification problems. It deals with diagnosis of diabetes of Pima Indians. Based on personal data (age, number of times pregnant) and the results of medical examinations (e.g. blood pressure, body mass index, result of glucose tolerance test, etc.), try to decide whether a Pima Indian individual is diabetes positive or not.

The dataset contains 8 inputs, 2 outputs, 768 examples. All inputs are continuous. 65.1% of the examples are diabetes negative; entropy .0.93 bits per example. Although there are no missing values in this dataset according to its documentation, there are several senseless 0 values. These most probably indicate missing data. Nevertheless, this data is handled as if it was real, thereby introducing some errors into the dataset. This dataset was created based on the "Pima Indians diabetes" problem dataset from the UCI repository of machine learning databases.

4.2.4 The Glass Problem

The dataset describes of classification of glass types. The results of a chemical analysis of glass splinters (percent content of 8 different elements) plus the refractive index are used to classify the sample to be either float processed or non float processed building windows, vehicle windows, containers, tableware, or head lamps. This task is motivated by forensic needs in criminal investigation.

There are 9 inputs, 6 outputs, 214 examples in the problem. All inputs are continuous; two of them have hardly any correlation with the result. As the number of examples is quite small, the problem is sensitive to algorithms that waste information. The sizes of the 6 classes are 70, 76, 17, 13, 9, and 29 instances, respectively; entropy 2.18 bits per example. This dataset was created based on the "glass" problem dataset from the UCI repository of machine learning databases.

4.2.5 The Heart Disease Problem (General Dataset)

Prediction of heart disease is the goal of the dataset. It decides whether at least one of four major vessels is reduced in diameter by more than 50%. The binary decision is made based on personal data such as age, sex, smoking habits, subjective patient pain descriptions, and results of various medical examinations such as blood pressure and electro cardiogram results.

The dataset consists of 35 inputs, 2 outputs, and 920 examples. Most of the attributes have missing values, some quite many. For attributes 10, 12, and 11, there are 309, 486, and 611 values missing, respectively. Most other attributes have around 60 missing values. Additional boolean inputs are used to represent the "*missingness*" of these values. The data is the union of four datasets: from Cleveland Clinic Foundation, Hungarian Institute of Cardiology, V.A. Medical Center Long Beach, and University Hospital Zurich. The "heart" dataset has 45% patients with "no vessel is reduced" (entropy 0.99 bits per example).

4.2.6 The Heart Disease Problem (Cleveland Dataset)

This is an alternate version of the dataset heart, called *heartc*, which contains only the Cleveland data. This dataset represents the cleanest part of the heart data; it has only two missing attribute values overall, which makes the "value is missing" inputs of the ANN input representation almost redundant.

This dataset has same number of input and output as *heart* but it has only 303 examples. The *heartc* datasets have 54% patients with "no vessel is reduced" (entropy 1.00 bits per example). These two datasets were created based on the "heart disease" problem datasets from the UCI repository of machine learning databases.

TABLE 4.1
CHARACTERISTICS OF DATASETS

Dataset	Total number of Patters	Number of inputs	Number of outputs
<i>Cancer</i>	699	9	2
<i>Card</i>	690	51	2
<i>Diabetes</i>	768	8	2
<i>Glass</i>	214	9	6
<i>Heart</i>	920	35	2
<i>Heartc</i>	203	35	2

4.3 Experimental Setup

In the past, there were some criticisms toward the neural networks benchmarking methodology [36]–[38]. Suggestions for improvement have been put forward [36]–[38]. These suggestions are followed in this literature. All datasets are partitioned into three sets: a training set, a validation set, and a testing set. The validation set is used to stop the training when the overall residual error reaches a flatter region. The test set is used to evaluate the generalization performance of the trained ANN and is not seen by the ANN during the whole training process. It is known that the experimental results may vary significantly for different partitions of the same data collection, even when the number of examples in each set is the same [36]. It is necessary to know precise specification of the partition in order to accomplish an experiment or conduct fair comparisons. In the following experiments, each dataset are partitioned as follows:

- ◆ For the *breast cancer* dataset, the first 350 examples were used for the training set, the following 175 examples for the validation set, and the final 174 examples for the testing set.
- ◆ For the *credit card* dataset, the first 345 examples were used for the training set, the following 173 examples for the validation set, and the final 172 examples for the testing set.

- ◆ For the *diabetes* dataset, the first 384 examples were used for the training set, the following 192 examples for the validation set, and the final 192 examples for the testing set.
- ◆ For the *glass* dataset, the first 107 examples were used for the training set, the following 54 examples for the validation set, and the final 53 examples for the testing set.
- ◆ For the *heart* dataset, the first 460 examples were used for the training set, the following 230 examples for the validation set, and the final 230 examples for the testing set.
- ◆ For the *heartc* dataset, the first 152 examples were used for the training set, the following 76 examples for the validation set, and the final 75 examples for the testing set.

In all experiments, one bias node with a fixed input +1 is connected only to the first hidden layer. The learning rate η is set between [0.01, 0.6] and the weights were initialized to random values between [-0.5, +0.5]. The momentum α is ranged between [0.4, 0.6] depending on the characteristics of datasets. A hyperbolic tangent function is taken as the activation function— $f = a \tanh(bv)$, where, v is the output of the summing junction, $a = 1.7159$, $b = 2/3$. The output neurons act in “winner takes all” manner.

4.4 Results and Analysis

Table 4.2 shows the results of MCA over 20 independent runs on six different classification problems. *Total training epochs* can be divided into two parts. One is the number of epochs required for maximization of correlation and the other is the number of epochs required for minimization of residual error. *Residual error* column contains the value of error E when the training is stopped. E is measured according to Eqn. (3.1). The *training error rate* and the *test error rate* in the tables refer to the percentage of wrong classification produced by the trained ANN on the training patterns and test patterns respectively. The standard deviations (*StDev*) of different error rates are given next to the corresponding columns.

TABLE 4.2

RESULTS OF MCA ON DIFFERENT CLASSIFICATION PROBLEMS
THE RESULTS WERE AVERAGED ON 20 INDEPENDENT RUNS

A. THE BREAST CANCER PROBLEM

	Total Epochs	No. of Hidden Neurons	Residual Error	StDev (Residual Error)	Training Error Rate	StDev (Training Error)	Test Error Rate	StDev (Test Error)
<i>Mean</i>	90	2	0.05244		0.04		0.0057	
<i>Min</i>	84	2	0.0523	0.000126	0.04	0	0.0057	0
<i>Max</i>	96	2	0.0526		0.04		0.0057	

B. THE AUSTRALIAN CREDIT CARD PROBLEM

	Total Epochs	No. of Hidden Neurons	Residual Error	StDev (Residual Error)	Training Error Rate	StDev (Training Error)	Test Error Rate	StDev (Test Error)
<i>Mean</i>	106.8	2.1	0.09994		0.11706		0.12384	
<i>Min</i>	82	2	0.0984	0.000608	0.1101	0.004967	0.1221	0.002802
<i>Max</i>	153	3	0.1004		0.1246		0.1279	

C. THE DIABETES PROBLEM

	Total Epochs	No. of Hidden Neurons	Residual Error	StDev (Residual Error)	Training Error Rate	StDev (Training Error)	Test Error Rate	StDev (Test Error)
<i>Mean</i>	289.9	3.9	0.15277		0.22608		0.2099	
<i>Min</i>	186	3	0.1527	0.000067	0.224	0.0029518	0.1979	0.009231
<i>Max</i>	383	5	0.1529		0.2318		0.2188	

D. THE GLASS PROBLEM

	Total Epochs	No. of Hidden Neurons	Residual Error	StDev (Residual Error)	Training Error Rate	StDev (Training Error)	Test Error Rate	StDev (Test Error)
<i>Mean</i>	474.7	4	0.28064		0.3742		0.31699	
<i>Min</i>	347	4	0.2775	0.003605	0.3551	0.015341	0.3019	0.017324
<i>Max</i>	591	4	0.2898		0.4019		0.3396	

E. THE HEART DISEASE PROBLEM (GENERAL DATASET)

	Total Epochs	No. of Hidden Neurons	Residual Error	StDev (Residual Error)	Training Error Rate	StDev (Training Error)	Test Error Rate	StDev (Test Error)
<i>Mean</i>	235.1	3	0.10666		0.13609		0.18913	
<i>Min</i>	134	2	0.1014	0.002736	0.1261	0.058053	0.1783	0.004686
<i>Max</i>	307	4	0.1097		0.1435		0.1913	

F. THE HEART DISEASE PROBLEM (CLEVELAND DATASET)

	Total Epochs	No. of Hidden Neurons	Residual Error	StDev (Residual Error)	Training Error Rate	StDev (Training Error)	Test Error Rate	StDev (Test Error)
<i>Mean</i>	215.9	3.4	0.10007		0.13815		0.1867	
<i>Min</i>	171	3	0.0985	0.000678	0.1118	0.018874	0.1867	0
<i>Max</i>	284	4	0.1005		0.1579		0.1867	

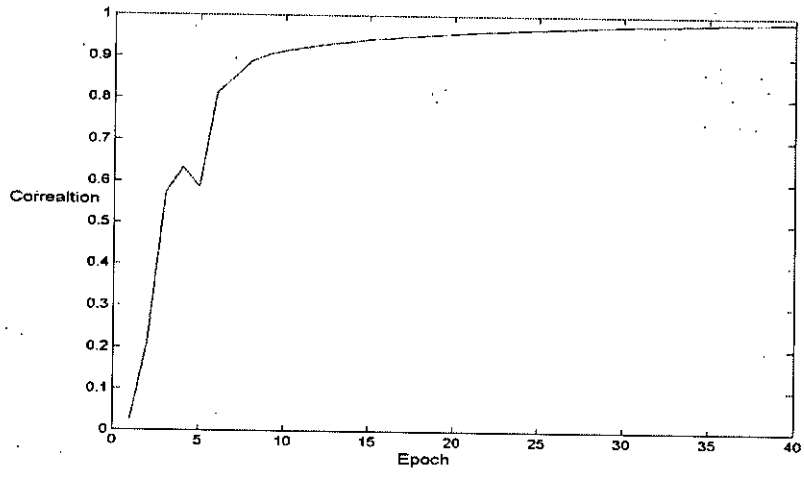
It is seen that MCA can produce compact ANN architecture with good generalization ability i.e. small test error rate. It also requires a small number of epochs in designing ANNs. For example, MCA performs very well on the *diabetes* problem (Table 4.2(C)), which is the most difficult problem in the realm of machine learning. In this case, the average number of epochs required for error minimization is around 171. The average number of epochs required to maximize the correlation in order to train the input side weights of the added hidden neurons is around 118. Three neurons are added on average

in the hidden layer(s). So, average number of epochs per hidden neuron is 39.33. Average residual error of 0.15277, training error rate of 0.2261 and test error rate of 0.2099 exhibits error minimizing capability of this ANN. These results also indicate the good generalization ability of MCA which is the inverse of the test error rate. Average test error rate of 0.2099 indicates that out of 192 test patterns, 40 patterns are incorrectly classified. This is a good result in the context of the critical *diabetes* problem. Moreover, it has standard deviations in the order 10^{-4} , 10^{-3} and 10^{-3} for residual error, training error rate and test error rate. These low standard deviations imply high *robustness* of MCA. Robustness is the consistency of the ANN under different initial conditions.

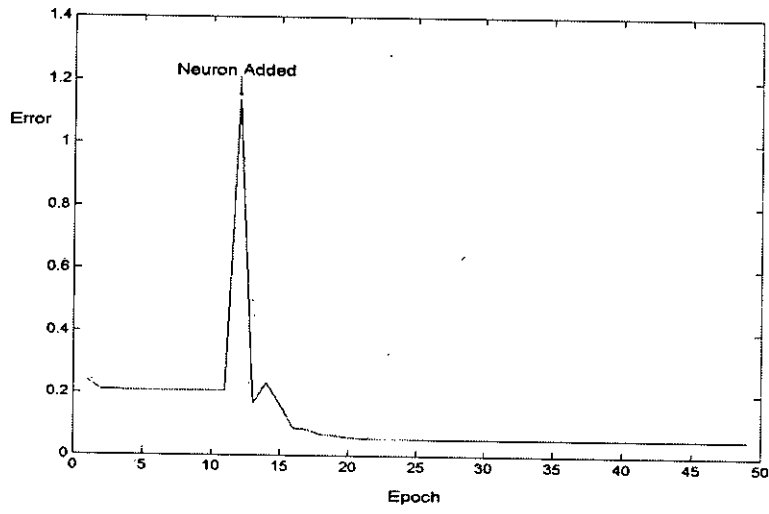
MCA exhibits very good results for not only for the *diabetes* problem, but also for the other benchmark problems. For the *cancer* problem (Table 4.2(A)), it shows the lowest test error rate (highest generalization), which is as low as 0.0057 on average. It means, among 174 test patterns, it incorrectly classifies only 1 pattern. On the other hand, it shows the highest error rate (lowest generalization) for the *glass* problem (Table 4.2(D)) which is 0.31699 on average. This is also good for the 'tiny' *glass* problem with the lowest pattern size. The lowest value of standard deviation for training patterns is 0 and it is achieved for the *cancer* problem. The highest standard deviation value for training patterns is 0.058053, produced by the *heart* problem. For the test patterns, the lowest standard deviation (0) is observed in two datasets, *cancer* and *heartc*. The highest standard deviation in this case is for the *glass* problem and it is only 0.017324. From table 4.2 one can figure out that for other classification problems, MCA exhibits excellent robustness as it has very low standard deviations.

In order to show how the training process of MCA progresses, Figs. 4.1 -4.6 show the training process for six different classification problems. The training process of MCA can be thoroughly explained by taking into account the *diabetes* (Fig. 4.3) and the *glass* (Fig. 4.4) problems. Several observations can be made from these figures.

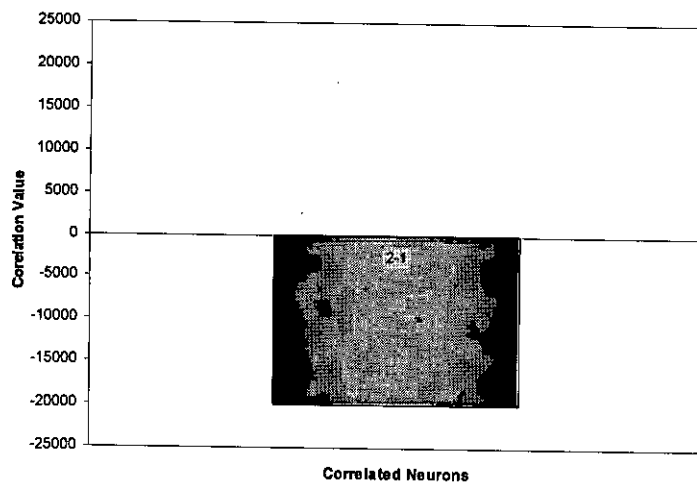
First, the correlation between a neuron and the residual error increases as the training process progresses, as shown in Figs. (4.3(a) & 4.4(a)). It is seen from these figures that the correlation does increase too much after some training epochs i.e. 40. MCA therefore stops the correlation maximization process after 40 epochs and add the neuron to the network.



(a)

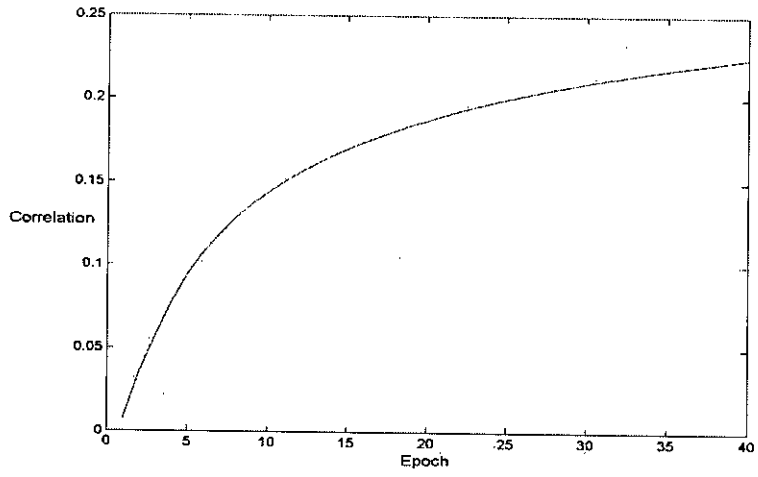


(b)

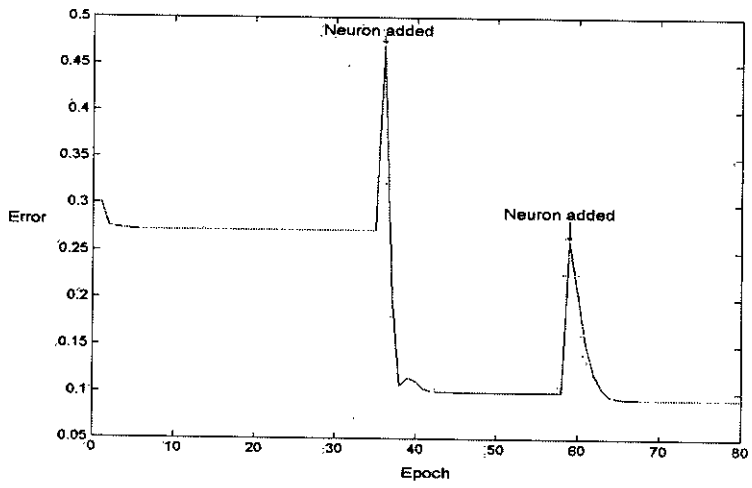


(c)

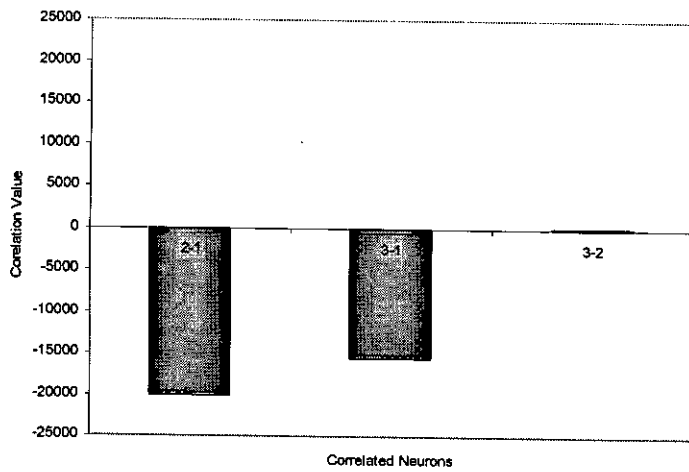
Fig 4.1 Training process of MCA for *cancer* problem



(a)



(b)



(c)

Fig 4.2 Training process of MCA for *card* problem

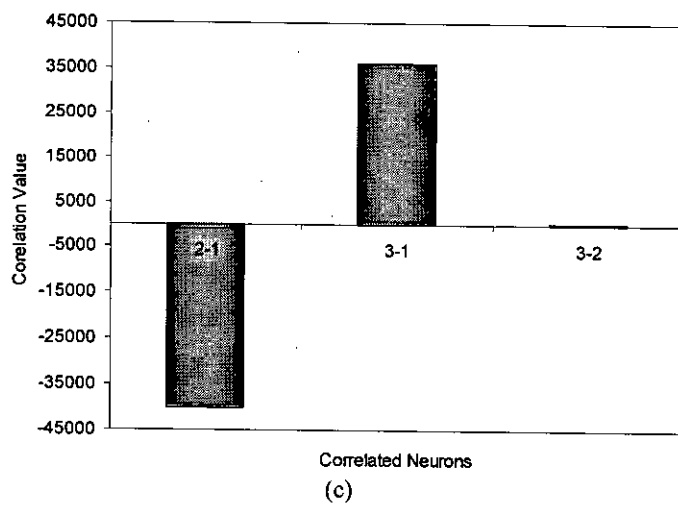
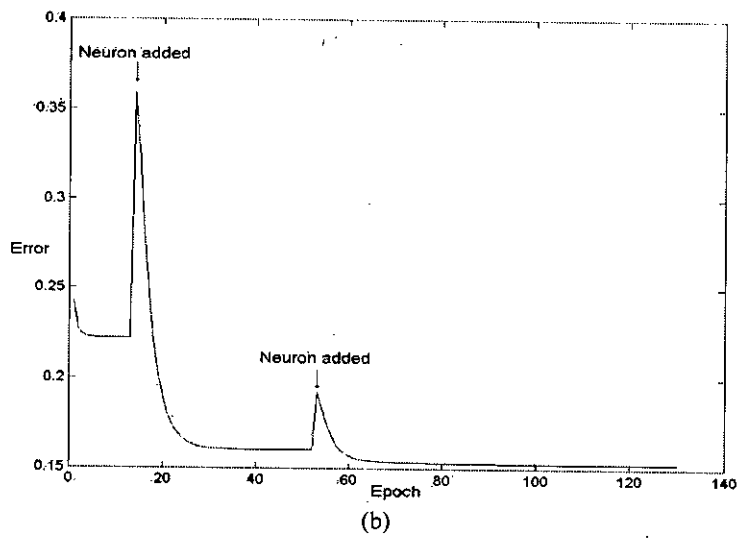
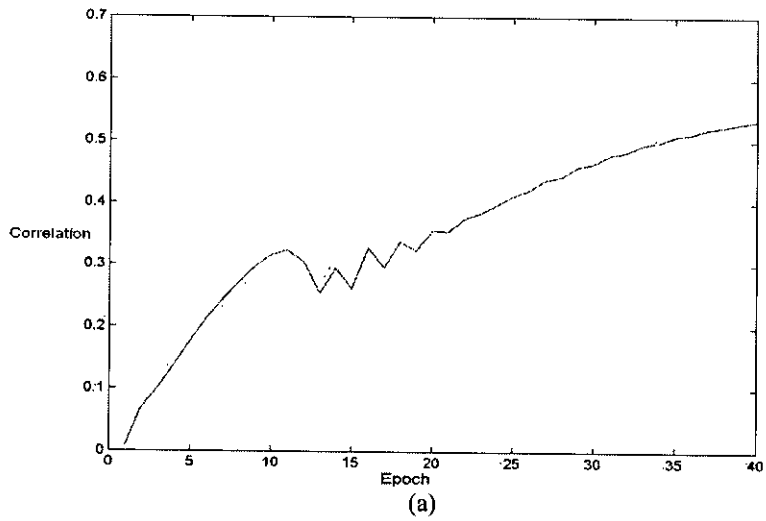


Fig 4.3 Training process of MCA for *diabetes* problem

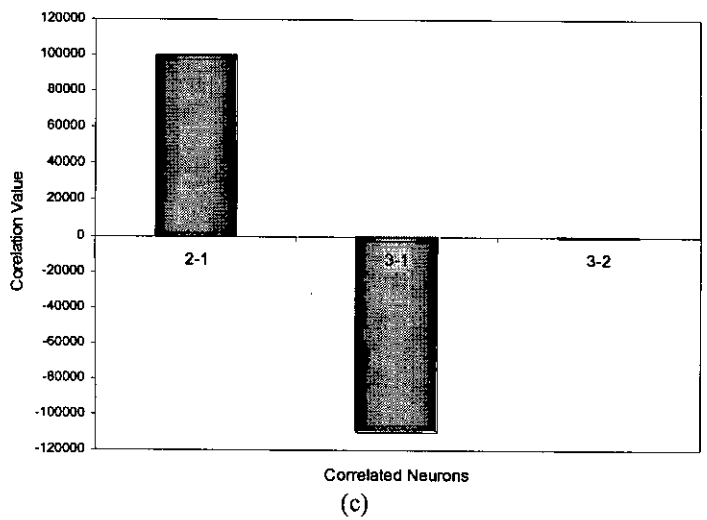
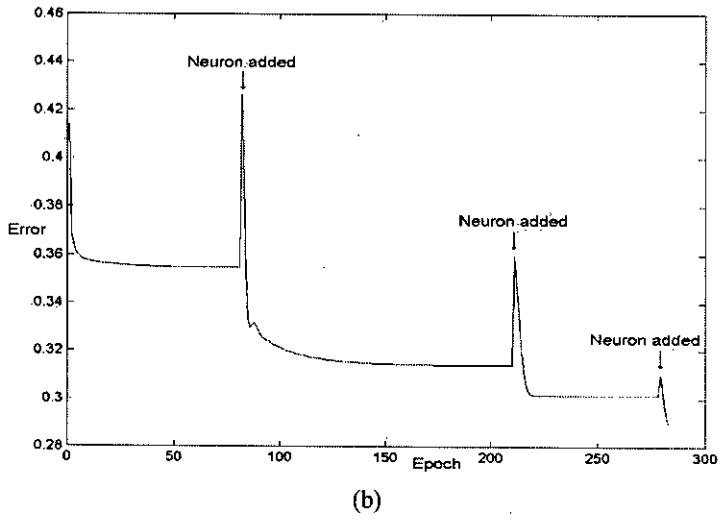
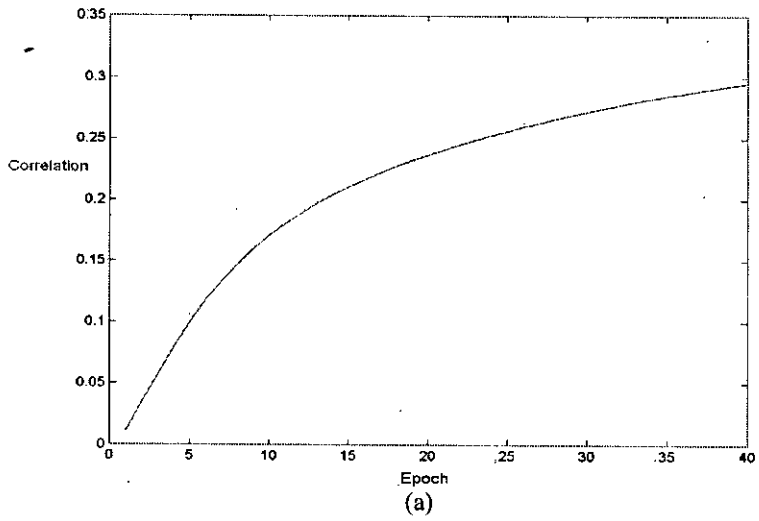
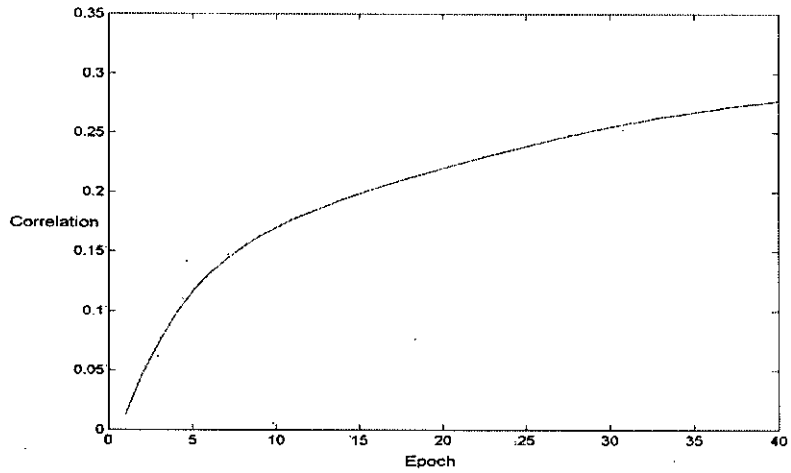
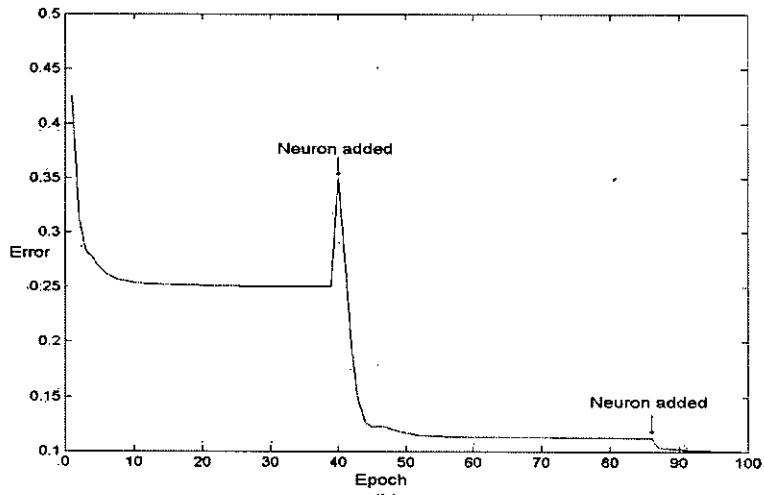


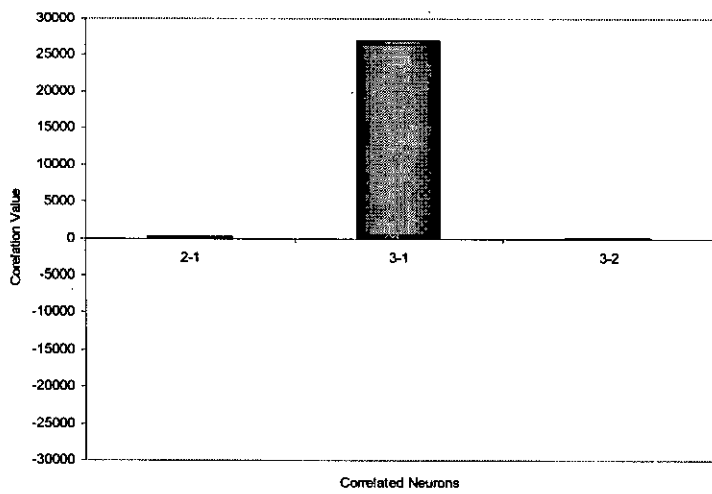
Fig 4.4 Training process of MCA for *glass* problem



(a)



(b)



(c)

Fig 4.5 Training process of MCA for *heart* problem

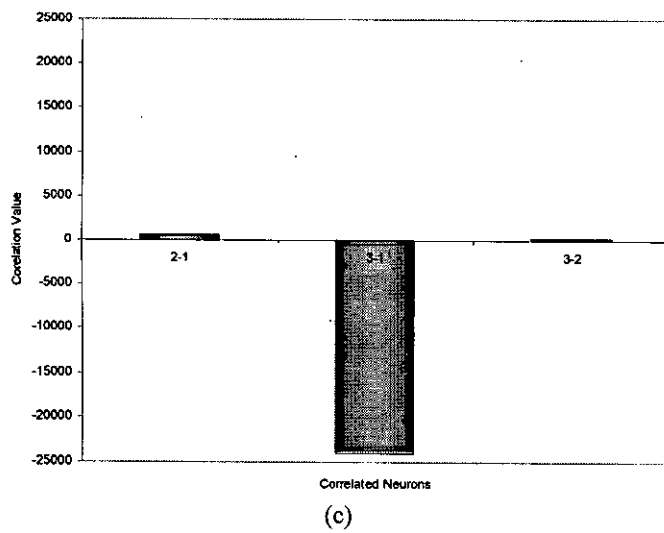
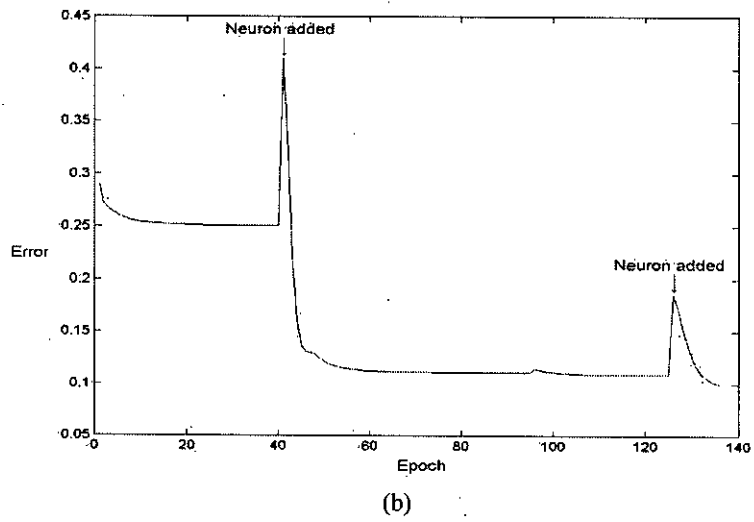
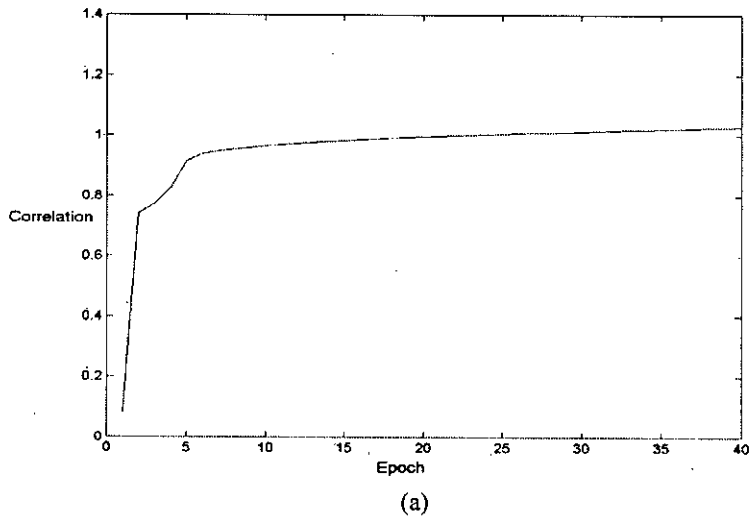


Fig 4.6 Training process of MCA for *heartc* problem

Second, it is observed from Figs. 4.3(b) and 4.4(b) that the error is minimized in very smooth manner except that there is a 'spike' when a new neuron is added. This happens due to two reasons. One is the initial random output weights of the added neuron and the other is the use of batch processing rather than sequential processing. Whenever a neuron is added to the ANN, its input side weights are already trained and frozen but the initial random weights of its output side are to be trained by using Eqns. (3.4 -3.6). These random weights may create a little spike on the curve. However, the spikes are quite large as seen from Figs. 3(b) and 4(b). This is because the weights are updated in MCA by using batch processing where the *cumulative* error for all the patterns is calculated. As a result, the error becomes too large before training.

Third, the training is stopped before the residual error comes to a flat region (Fig. 4.4(b)). This is due to the use of validation patterns. In MCA, the validation patterns were passed through the ANN and the error rate is calculated after every 30 epochs. When the error rate of the validation set reaches minimum, the training was stopped and the weights at that point were saved to evaluate different error rates.

Fourth, the negative correlation criterion can be observed from Figs. (4.3(c) & 4.4(c)). In Fig.(4.3(c)), newly added hidden neuron correlates negatively with the existing initial hidden neuron (*see column 2-1*) in the first hidden layer. So, the summation of correlations SN starts with a negative value i.e. $SN < 0$. When another neuron is added, although it correlates positively (*see column 3-1*) with the first neuron, SN remains negative. But no more neurons are added afterwards to meet the negative correlation criterion. This is due to the fact that the overall training is stopped at this point by validating the ANN with the validation patterns. Fig. 4.4(c) shows the perfect example of the negative correlation criterion. Here, SN starts with a positive value, but after the addition of the third hidden neuron, SN crosses zero, i.e. $SN \leq 0$. The growth of the first hidden layer is stopped then but the overall training is stopped after adding one neuron in cascade (*third spike on Fig. 4.4(b)*). As *glass* is a complex problem with a small number of training patterns, so the presence of cascaded layer is very necessary.

4.5 Comparison

The obtained results of MCA on six benchmark problems are compared with the results of different constructive and/or pruning algorithms in designing multiple hidden layered

ANNs. Three very well-known algorithms—the original cascade correlation (CasCor) [19], CNNDA [26] and aCasper [39] are taken into account. CasCor is the mother of all the constructive algorithms for designing multiple hidden layered ANNs. Very few algorithms could show better result than CasCor. One of these algorithms is the CNNDA. It uses both addition and deletion of neurons and connections. aCasper was approached in 1999 which is a modified regularized version of CasCor to obtain better generalization. The algorithm uplifted the performance of CasCor by a small degree.

Tables 4.3 - 4.5 show the comparisons among MCA and other algorithms on the basis of hidden neurons, classification error and training epochs.

TABLE 4.3
COMPARISON AMONG MCA, CASCOR, ACASPER AND CNNDA IN TERMS OF AVERAGE NUMBER OF HIDDEN NEURONS REQUIRED

DATA SET	MCA	CasCor	aCasper	CNNDA
<i>Cancer</i>	2	5.18	4.86	3.4
<i>Card</i>	2.1	1.07	0.12	-
<i>Diabetes</i>	3.9	9.78	3.02	4.3
<i>Glass</i>	4	8.07	4.18	-
<i>Heart</i>	3	2.64	0.1	-
<i>Heartc</i>	3.4	1.38	0.1	-

TABLE 4.4
COMPARISON AMONG MCA, CASCOR, ACASPER AND CNNDA IN TERMS OF CLASSIFICATION ERRORS

DATA SET		MCA	CasCor	aCasper	CNNDA
<i>Cancer</i>	<i>Mean</i>	0.0057	0.0195	0.0189	0.0116
	<i>StDev</i>	0	0.0038	0.008	-
<i>Card</i>	<i>Mean</i>	0.12384	0.1358	0.1372	-
	<i>StDev</i>	0.0028	0.0043	0.0059	-
<i>Diabetes</i>	<i>Mean</i>	0.2099	0.2453	0.2314	0.2087
	<i>StDev</i>	0.00923	0.0144	0.0126	-
<i>Glass</i>	<i>Mean</i>	0.31699	0.3476	0.3068	-
	<i>StDev</i>	0.017324	0.0588	0.0261	-
<i>Heart</i>	<i>Mean</i>	0.18913	0.1989	0.1921	-
	<i>StDev</i>	0.00468	0.0158	0.0044	-
<i>Heartc</i>	<i>Mean</i>	0.1867	0.1947	0.1885	-
	<i>StDev</i>	0	0.0128	0.0114	-

TABLE 4.5
COMPARISON BETWEEN MCA AND CNNDA IN TERMS OF TRAINING EPOCHS

<i>ALGORITHM ></i>	MCA	CNNDA
<i>DATA SET</i>		
<i>Cancer</i>	90	499.1
<i>Card</i>	106.8	-
<i>Diabetes</i>	289.9	335.4
<i>Glass</i>	474.7	-
<i>Heart</i>	235.1	-
<i>Heartc</i>	215.9	-

Observing tables 4.3 to 4.5, the comparative analysis for six classification problems among MCA and other eight algorithms are stated below—

- ◆ *The breast cancer problem:* For the *cancer* problem, MCA outperforms almost all the algorithms in each event. Number of hidden neurons required is lower than all of the algorithms. MCA shows the lowest classification error on average. The nearest low average classification error is offered by CNNDA which is 2.03 times larger than MCA. Even in the case of number of training epochs required, MCA outperforms CNNDA.
- ◆ *The Australian credit card problem:* For the *card* problem, number of hidden neurons required in MCA is higher than the algorithms CasCor and aCasper. These two algorithms have short-cut connections, making the architecture much more complex than MCA. In case of classification error, MCA shows the lowest error on average. The nearest low average classification error is offered by RPROP (LC) which is 1.07 times larger than MCA.
- ◆ *The Diabetes problem:* The *diabetes* problem is considered as one of the most challenging problems in the benchmarking realm. For the diabetes problem, number of hidden neurons required in MCA is lower than almost all the algorithms except the aCasper. In case of classification error, MCA shows lower error than all of the algorithms except CNNDA on average. CNNDA has a complex and strictly four layered architecture with lot of user defined parameters that helps it to generalize well in the case of diabetes problem. But MCA exhibits only 1.0057 time larger error rate, which is considerable with lower number of

102834

hidden neurons. In the case of number of training epochs required, MCA outperforms CNNDA.

- ◆ ***The Glass problem:*** The *glass* problem has the lowest number of patterns. So it is difficult to solve without multiple hidden layer. For the glass problem, number of hidden neurons required in MCA is the lowest among all the algorithms. In case of classification error, MCA shows lower error than all of the algorithms except aCasper on average. aCasper has a complex architecture with shortcut connections that helps it to generalize well in glass problem by increasing the feature space.
- ◆ ***The Heart disease problem (General Dataset):*** For the *heart* problem, number of hidden neurons required in MCA is higher than the algorithms CasCor and aCasper. In case of classification error, MCA shows lower error than all of the algorithms on average. The nearest low average classification error is offered by aCasper which is 1.0157 times larger than MCA.
- ◆ ***The Heart disease problem (Cleveland Dataset):*** For the *heartc* problem, number of hidden neurons required in MCA is higher than the algorithms CasCor and aCasper. In case of classification error, MCA shows lower error than all of the algorithms on average. The nearest low average classification error is offered by aCasper which is 1.0096 times larger than MCA.

4.6 Discussion

This section briefly explains why the performance of MCA is better than other constructive algorithms i.e. CasCor [19], CNNDA [26] and aCasper [39] for most classification problems we tested. There are three major differences that might contribute to better performance by MCA in comparison with other similar algorithms.

- ◆ The first is that MCA is an algorithm which designs an ANN consisting of single or more hidden layers, depending on the complexity of the problem. MCA starts with a minimal architecture, and then tries an ANN with single hidden layer to solve the problem. If the problem remains unsolved, then the hidden layer growth is stopped and more layers, consisting of a single neuron per layer, are constructed above it in cascade. This results in a compact ANN. Moreover, as

there is no need to define the initial layer number or layer size, chance of underfitting or overfitting is less. Other algorithms like CasCor or aCasper does not also need to define the initial architecture but they may create a very complex cascaded architecture for simple problems. In the case of CNNDAs hidden layer numbers are to be predefined.

- ◆ The second is that there is no effective way of stopping the hidden layer growth in the existing algorithms. As a result, either the number of hidden layers or the number of neurons in each hidden layer needs to be restricted. MCA involves a unique layer stopping criterion, named the negative correlation criterion to restrict the number of neurons in the first hidden layer. This process increases the classification ability. Although CasCor and aCasper does not need the hidden layer to be stopped until the problem is solved but again the problem of overfitting may occur due to the cascaded architecture.
- ◆ The third is the use of 'batch' processing rather than the conventional 'sequential' processing in the single layer BP learning. This results in excellent robustness of the ANN. In sequential processing, every time a pattern is passed through the ANN, the output error is calculated and the weight is updated. But in batch processing, all the patterns are passed through the ANN to calculate the cumulative output error. Although this process slows down the learning, but it can get rid of the 'herd effect' described in [14] and results in better generalization. An acceleration factor is used to speed up the batch processing.

The design process of MCA is observed and analyzed in every step. Some non-conventional approaches are used in MCA those are proven commendable. Use of momentum term became much easier when batch processing is used. In most of the cases the ANN need not run into the cascaded layer resulting in a strictly layered ANN. This is because every neuron is used up to its maximum capacity. When cascaded layer is formed (i.e. *glass* problem), only one or two neurons are required to solve the problem.

MCA shows a great stability as the standard deviation for classification error for MCA is lower than almost all the compared algorithms. So, MCA can be a very good alternative for CasCor and its variants for its simplicity and performance.

4.7 Conclusion

MCA is a very efficient but simple constructive algorithm. Although it is developed based on the concept of CasCor, it performs much better than CasCor in every media of performance measurement. ANN created by MCA not only generalizes well but also creates less complex architectures than existing constructive and/or pruning algorithms. As performance of any ANN is determined by its architecture and generalization ability, so it can be concluded that MCA is a high performance constructive algorithm.

5.1 Concluding Remarks

ANNs are one of the most widely used approaches to inductive learning. They have been applied to classification, regression, and reinforcement learning tasks and they have demonstrated good predictive performance in a wide variety of interesting problem domains. They suffer from some significant limitations. Most important of them is the prediction of ANN architecture before training. To address this limitation, a number of research groups have developed constructive architectures. The focus of this thesis has been the development of a constructive algorithm, called MCA, which overcomes the significant limitations of previous algorithms.

Constructive algorithms have been introduced to the ANN community for nearly 27 years. However, most constructive algorithms can only add neurons in the same layer, resulting in a single hidden layered architecture. Few algorithms exist that design automatically multiple hidden layered ANNs, e.g., the number of neurons in a hidden layer and the number of hidden layers. This paper proposes a new constructive algorithm to design as well as train multiple hidden layered ANNs. Neither the number of neurons in a hidden layer nor the number of hidden layers needs to be predefined and fixed. They are determined automatically in the learning process.

The approached algorithm, i.e. MCA uses both the layered and cascaded constructive approach to grow the neurons and hidden layers incrementally until the ANN performance reaches a satisfactory level. It starts with a simple ANN and then tries to solve the problem by increasing the number of hidden neurons in the same hidden layer. If the problem is too nonlinear to be solved with a single hidden layer, then MCA creates cascaded layer above the first hidden layer until the problem is solved. In MCA, a new method, *the negative correlation criterion* (NCC) is developed to determine the stopping point of the first hidden layer growth. Generalization is encouraged through the use of a validation set. MCA uses the correlation maximization process for training the input side weights of any neuron. According to [23] this is the reason behind accurate and fast performance of MCA. The short-cut connections from input to output are omitted for

simplicity and it is found that these connections contribute a little in the training of the ANN.

Extensive experiments have been carried out in this paper to evaluate how well MCA performed on different problems in comparison with other BP, constructive, pruning and evolutionary algorithms. In almost all cases, MCA outperformed the others. The low standard deviation of the classification error shows the great stability of this algorithm.

Although MCA has performed very well for almost all problems, experimental study appeared to have revealed a weakness of MCA in dealing with the *glass* problem. This is because the fact that MCA creates fewer connections as the input layer is connected only to the first hidden layer, resulting in a smaller feature space. There occurs a connectivity and performance tradeoff as described in [35]. ANNs, who outperform MCA at any problem, must have much more connections in their architecture, become difficult for VLSI implementation.

MCA with was compared with other similar algorithms but it is impossible to compare them fairly without re-implementing all the algorithms under the same experimental setup.

5.2 Future Scopes

MCA has some limitations which keeps open the field to study and improve the design methods of MCA. Speeding up the training can be good future work. Speeding up can be done by using any fast converging algorithm like 'quickprop' [33]. In MCA, the layer stopping criterion is used to stop the growth of the first hidden layer, but it can be used in any layers. It can also be used for training new hidden neurons. The cascaded hidden layers consist of only one neuron per layer, but multiple neurons can be added in the cascaded layers. Only classification problems were solved with MCA. With minor alterations, MCA can be used as a good predictor in regression tasks. When any new neuron is to be added to the hidden layer, one single candidate is chosen and trained; but one can also try the same thing with a 'pool' of candidates and select the one with highest correlation value.

REFERENCES

- [1] D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning internal representations by error propagation," *Parallel Distributed Processing: Explorations in the Microstructure Cognition, Chapter 8*, D. E. Rumelhart and J. L. McClelland (eds.), MIT Press, Cambridge, MA, and London, England, 1986.
- [2] T. Y. Kwok, and D. Y. Yeung, "Constructive algorithms for structure learning in feedforward neural networks for regression problems", *IEEE Transactions on Neural Networks*, Vol. 8, pp. 630–645, 1997.
- [3] R. Reed, "Pruning algorithms – a survey", *IEEE Transactions on Neural Networks*, Vol. 4, pp. 740–747, 1993.
- [4] C. Xiang, S. Q. Ding, and T. H. Lee, "Geometrical interpretation and architecture selection of MLP," *IEEE Transactions on Neural Networks*, Vol. 16, pp. 84 – 96, 2005.
- [5] T. Ash, "Dynamic node creation in backpropagation networks," *ICS Report 8901*, Institute of Cognitive Science, University of California, San Diego, 1989.
- [6] M. R. Freen, "The upstart algorithm: a method for constructing and training feedforward neural networks," *IEEE Transactions on Neural Networks*, Vol. 2, pp. 198–209, 1990.
- [7] E. Alpaydin, "GAL: networks that grow when they learn and shrink when they forget," *Technical Report 91-032*, International Computer Science Institute, Berkeley, CA, 1991.
- [8] Mike Wynn-Jones, "Node splitting: A constructive algorithm for feed-forward neural networks," *Neural Computing and Applications*, Vol. 1, pp. 17-22, 1993.
- [9] T. M. Nabhan and A. Y. Zomaya, "Toward generating neural network structures for function approximation," *Neural Networks*, Vol. 7, pp. 89–99, 1994.
- [10] S. Young, and T. Downs, "CARVE - a constructive algorithm for real valued examples," *IEEE Transactions on Neural Networks*, Vol. 9, pp. 1180-1190, 1998.
- [11] R. Setiono, "Feedforward neural network construction using cross validation," *Neural Computation*, Vol. 13, pp. 2865–2877, 2001.
- [12] A. Eleuteri, R. Tagliaferri, and L. Milano, "A novel information geometric approach to variable selection in MLP networks," *Neural Networks*, Vol. 18, pp. 1309 – 1318, 2005.

- [13]F. L. Chung and T. Lee, "A node pruning algorithm for backpropagation networks," *International Journal of Neural Systems*, Vol. 3, pp. 301- 314, 1992.
- [14]G. Castellano, A. M. Fanelli and M. Pellilo, "An Iterative Pruning Algorithm for Feedforward Neural Networks," *IEEE Transactions on Neural Networks*, Vol. 8, pp. 519-531, 1997.
- [15]M. Kearns and Y. Mansour, "A fast, bottom-up decision tree pruning algorithm with near optimal generalization," in *Proceedings 15th International Conference on Machine Learning* , pp. 269-277, 1998.
- [16]D. B. Fogel, L. J. Fogel, and V. W. Porto, "Evolving neural networks," *Biological Cybernetics*, Vol. 63, pp. 487-493, 1990.
- [17]X. Yao, "Evolutionary artificial neural networks," *International Journal on Neural Systems*, vol. 4, pp. 203-222, 1993.
- [18]X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Transactions on Neural Networks*, Vol. 8, pp. 694 - 713, 1997.
- [19]S. E. Fahlman, C. Lebiere, "The cascade-correlation learning architecture," *Technical Report CMU-CS-90-100*, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1991.
- [20]S. Sjøgaard, "A conceptual approach to generalization in dynamic neural networks," Ph.D. thesis, Aarhus University, Aarhus, Denmark, 1991.
- [21]D. Y. Yeung, "A neural network approach to constructive induction," *Machine Learning- Proceedings of the 8th International Workshop*, San Mateo, CA, Morgan Kaufman Publishers, pp. 228 - 232, 1991.
- [22]E. Littmann and H. Ritter, "Cascade network architectures," in *Proceedings International Joint Conference on Neural Networks*, Baltimore, Vol. 2, pp. 398-404, 1992.
- [23]M. Lehtokangas, J. Saarinen, K. Kaski, "Simplified cascade correlation learning," in *Proceedings European Symposium on Artificial Neural Networks*, Brussels, pp. 39-44, 1995.
- [24]M. Lehtokangas, "Fast initialization for cascade-correlation learning," *IEEE Transactions on Neural Networks*, Vol. 10, pp. 410 - 413, 1999.
- [25]_____, "Modified cascade-correlation learning for classification," *IEEE Transactions on Neural Networks*, Vol. 11, pp. 795 - 798, 2000.
- [26]M.M. Islam, K. Murase, "A new algorithm to design compact two-hidden-layer artificial neural networks", *Neural Networks*, Vol. 14, pp. 1265-1278, 2000.

- [27] O. Aran, E. Alpaydin, "An Incremental Neural Network Construction Algorithm for Training Multilayer Perceptrons", *ICANN'03*, Istanbul, 2003.
- [28] S. Tamura and M. Tateishi, "Capabilities of a four-layered feedforward neural network: four layers versus three," *IEEE Transactions on Neural Networks*, Vol. 8, pp. 251 – 255, 1997.
- [29] S. Ramon y Cajal, "Histologie du Systems Nerveus de l'homme et des vertebrae," Paris, Maloine, 1911.
- [30] G. M. Shepherd, and C. Koch, "Introduction to synaptic circuits," *The Synaptic Organization of the Brain*, G. M. Shepherd ed., New York, Oxford University Press, pp. 3-31, 1990.
- [31] F. Faggin, "VLSI implementation of neural networks," *Tutorial Notes, International Joint Conference on Neural Networks*, Seattle, 1991.
- [32] S. Haykin, "Neural Networks- A Comprehensive Foundation," *Pearson Education Inc.*, 1999.
- [33] S. E. Fahlman, "Faster-learning variations on back-propagation: an empirical study" in *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufmann, 1988.
- [34] S. Tamura, "Capabilities of a three layered feedforward neural network," in *Proc. IJCNN91*, pp. 2757- 2762, 1991.
- [35] D. S. Phatak, I. Koren, "Connectivity and performance tradeoffs in the cascade correlation learning architecture", *Technical Report TR-92-CSE-27*, ECE Dept., UMASS, Amherst, 1994.
- [36] L. Prechelt, "PROBEN1- A set of neural network benchmark problems and benchmarking rules," *Technical Report 21/94*, Faculty of Informatics, University of Karlsruhe, Germany, 1994.
- [37] _____, "A quantitative study of experimental evaluation of neural network learning algorithms," *Neural Networks*, Vol. 9, pp. 457–462, 1996.
- [38] _____, "Some notes on neural learning algorithm benchmarking," *Neurocomputing*, Vol. 9, pp. 343–347, 1995.
- [39] N. K. Treadgold, T. D. Gedeon, "Exploring constructive cascade networks," *IEEE Transactions on Neural Networks*, Vol. 10, pp. 1335 – 1350, 1999.

