An Algorithm to Extract Rules from Artificial Neural Networks

by

S. M. Kamruzzaman

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

BUET

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY
March 29, 2005

[illegible]

The thesis titled **"An Algorithm to Extract Rules from Artificial Neural Networks"** submitted by **S. M. Kamruzzaman,** Roll No. 040005001P, Session April 2000 has been accepted as satisfactory in partial fulfillment of the requirements for the degree of **Master of Science in Computer Science and Engineering.** Examination held on March 29, 2005. Approved as to style and contents by:

M. M. Islam

1. Dr. Md. Monirul Islam (Supervisor) Chairman
Associate Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

[illegible] 29/3/05

2. Dr. Md. Shamsul Alam Member (Ex-officio)
Professor and Head
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

M Masroor Ali.

3. Dr. Muhammad Masroor Ali Member
Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

[illegible]

4. Dr. Md. Mostofa Akbar Member
Assistant Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

[illegible]

5. Dr. Mohammad Zahidur Rahman Member (External)
Associate Professor
Department of Computer Science and Engineering
Jahangirnagar University, Savar, Dhaka, Bangladesh

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.

[illegible]

S. M. Kamruzzaman
(Candidate)

To my Parents and Wife
for their support and encouragement

Acknowledgements

This thesis would not exist without the support and guidance of many others. Happily, I can now thank some of the people who played an important role in my graduate carrier.

First and foremost, I would like to thank Dr. Md. Monirul Islam, Associate Professor, Department of Computer Science and Engineering, BUET, my supervisor, for his endless patience, scholarly guidance, constant and energetic supervision, valuable advice, suggestions and encouragement at all stages have made it possible to complete this thesis.

Special gratitude to Dr. Md. Shamsul Alam, Professor and Head, Department of CSE, BUET for his kind cooperation in this research work. I would also like to thank the other members of my thesis committee: Dr. Muhammad Masroor Ali, Dr. Md. Mostofa Akbar, and the external member Dr. Mohammad Zahidur Rahman for their kind suggestions.

I would like to express my profound gratitude to my former supervisor Prof. Dr. Chowdhury Mofizur Rahman for his continuous support, indispensable advices and encouragement during my study, specially, in the initial stage of my thesis work.

I am very grateful to Mr. Mohammad Mahfuzul Islam, my course teacher for his valuable discussion and suggestions. I would like to thank to Mr. Ahmed Ryadh Hasan of Independent University Bangladesh, my younger brother, for his continuous support and encouragement.

Many thanks to my friends and colleagues of International Islamic University Chittagong, Mr. Emdadul Haque, Mr. Shamsul Alam, Mr. Abul Hasan and Mr. Ilias Amin for making my graduate life enjoyable and memorable. I would also like to thank my parents and brothers for their love and support, as well as the continuous contribution for my higher studies. Most importantly, I would not have been able to accomplish my goals in graduate carrier without the love, patience, support, and joy that my life partner Jesmin and my daughter Rifa have provided.

Abstract

A new rule extraction algorithm, called rule extraction from artificial neural networks (REANN) is proposed and implemented to extract symbolic rules from ANNs. A standard three-layer feedforward ANN is the basis of the algorithm. A four-phase training algorithm is proposed for backpropagation learning. In the first phase, the number of hidden nodes of the network is determined automatically in a constructive fashion by adding nodes one after another based on the performance of the network on training data. In the second phase, the ANN is pruned such that irrelevant connections and input nodes are removed while its predictive accuracy is still maintained. In the third phase, the continuous activation values of the hidden nodes are discretized by using an efficient heuristic clustering algorithm. And finally in the fourth phase, rules are extracted by examining the discretized activation values of the hidden nodes using a rule extraction algorithm, REx. Extensive experimental studies on several benchmarks classification problems, such as breast cancer, iris, diabetes, wine, season, golf-playing, and lenses classification problems, demonstrate the effectiveness of the proposed approach with good generalization ability.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

The last two decades have seen a growing number of researchers and practitioners applying artificial neural networks (ANNs) to solve a variety of problems such as pattern classification and function approximation or regression problem. The outputs, and often the inputs as well, are discrete for classification problems. On the other hand, function approximation or regression problems have continuous outputs, and the function or regression may be nonlinear. In many applications, it is desirable to extract knowledge from trained ANNs for the users to gain a better understanding how the ANNs solve the problems. The classification concept represented as rules is certainly more comprehensible to human users than a collection of ANNs weights.

While the predictive accuracy obtained by ANNs is often higher than that of other methods or human experts, it is generally difficult to understand how ANNs arrive at a particular conclusion due to the complexity of the ANNs architectures. It is often said that an ANN is practically a "black box". Even for an ANN with only single hidden layer, it is generally impossible to explain why a particular pattern is classified as a member of one class and another pattern as a member of another class.

The rules generated from ANNs should be simple enough for human users to understand. Unlike a collection of ANN weights, symbolic rules can easily be interpreted and verified by human experts. They can also provide new insights about application problems and the corresponding data. Getting an explanation about the reasoning of the ANN is not very easy. This is mainly because the learned knowledge is represented by the topology of the network and by the weight and bias values. These are usually not meaningful for humans.

Lack of explanation capability is one of the most important reasons why ANNs do not get the necessary interest in the industry. Users want to know the reasoning behind the conclusion of a learning system in most of the real world applications such as safety critical applications. It is therefore necessary that an ANN should be able to explain itself. This can be done in several ways: extracting if-then rules, converting ANNs to decision trees are some of them.

Extracting if-then rules is usually accepted as the best way of extracting the knowledge represented in the ANN. Not because it is an easy job, but because the rules created at the end are more understandable for humans than any other representation (i.e. decision trees). And the rules extracted form the trained network can be used in other systems, like expert systems. Extracting rules from ANNs has some other advantages:

i) Knowledge Discovery: ANNs are very powerful in discovering previously unknown dependencies and relationships in the data sets. But these discoveries are coded as weights within the network. Weights do not make much sense to human users. By extracting rules these discoveries can be unrevealed.

ii) Knowledge Acquisition for Expert Systems: One of the biggest bottlenecks of the expert systems is the difficulty in knowledge acquisition. Rules generated from a trained ANN can solve the knowledge acquisition problem.

iii) Training Data Validation: An expert system can validate the data used in training by looking at the rules created and can choose a less noisy sample from the data.

iv) User Explanation Capability: Absence of explanation capability in ANNs limits the realization of full potential of such systems. Users need to explain about the results in safety critical real life applications. Rules generated from an ANN will make the realization.

v) Improving Generalization Ability of ANNs: Cross validation is widely used for improving the generalization ability of ANNs. However, this may even fail in some cases. If an experienced user knows the rules represented within the ANN he/she can try to find under which cases the generalization fails. He/she can also find the regions, which are not represented properly in the training set.

vi) Understanding of How Symbolic and Connectionist Approaches Intcgrated: Rule extraction from ANNs can help us to understand how one can profitably integrate symbolic and connectionist approaches.

Rule extraction algorithms can be categorized as "decompositional", "pedagogical" and "eclectic". Decompositional rule extraction algorithms focus on extracting rules at the level of hidden and output nodes. Pedagogical techniques see the ANN as a black box. The techniques see the rule extraction as a learning task. They try to extract rules that map inputs to outputs. Eclectic techniques try to incorporate elements of both techniques.

There are four criteria to determine the quality of rules. These are: accuracy, fidelity, consistency, and comprehensibility.

- Rule set is considered **accurate** if it can correctly classify unseen cases.
- If the rule set can mimic ANNs it has a high level of **fidelity**.
- Rule sets are **consistent** if the rule sets extracted from the ANNs, which are trained under different conditions, create the same classification.
- The **comprehensibility** is measured by the size of the rule set and by the number of antecedents in each rule.

1.2 Literature Review

A number of works exists in the literature that extracts rules from trained ANNs [1-2]. Two methods for extracting rules from ANN are described in Towell and Shavlik [3]. The first method is the subset algorithm [4], which searches for subsets of connections to a node whose summed weight exceeds the bias of that node. The major problem with Subset algorithms is that the cost of finding all subsets grows as the size of the power set of the links to each node.

The second method, the M-of-N algorithm [5], is an improvement of the Subset method that is designed to explicitly search for M-of-N rules from knowledge based ANNs. Instead of considering an ANN connection, groups of connections are checked for their contribution to a node's activation. This is done by clustering the ANN connections. The weights of the connections in a cluster are then replaced by their average weights. Cluster with small average and a few connections are checked for possible elimination since their removal are not likely to have any effect on the network classification. A rule is formed for each hidden and output node. This rule consists of a threshold and the weighted antecedents of the remaining connections.

M-of-N approach only works well on knowledge-based networks, created from domain knowledge rules (like KBANN or KBCNN). It cannot extract rules from classical networks. This technique has some weaknesses. M-of-N algorithm makes the assumption that each hidden node corresponds to a meaningful concept. Hidden nodes are assigned symbolic names and the relationship between nodes and names should not change during the learning. Preventing the change is very difficult in real world applications.

Craven and Shavlik [6] proposes a method that uses sampling and queries. Instead of searching for rules from the ANN, the problem of rule extraction is viewed as a learning task. The target concept is the function computed by the network and the ANN input features are the inputs for the learning task. Conjunctive rules are extracted from the ANN with the help of two oracles.

H. Liu and S. T. Tan [7] proposes X2R, a simple and fast algorithm that can be applied to both numeric and discrete data, and generate rules from datasets. X2R can generate concise rules from raw data sets. It only calculates first order information in generating rules. It can generate perfect rules in the sense that the error rate of the rules is not worse than the inconsistency rate found in the original data. The rules generated by X2R, are order sensitive, i.e, the rules should be fired in sequence.

R. Setiono and H. Liu [8] presents a novel way to understand an ANN. Understanding an ANN is achieved by extracting rules with a three phase algorithm: first, a weight decay backpropagation network is built so that important connections are reflected by their bigger weights; second, the network is pruned such that insignificant connections are deleted while its predictive accuracy is still maintained; and last, rules are extracted by recursively discretizing the hidden node activation values.

Thrun [9] describes a rule extraction algorithm, which analyzes the input-output behavior of a network using Validity Interval Analysis. VI-Analysis divides the activation range of each network's node into intervals, such that all network's activation values must lie within the intervals. The boundary of these intervals are obtained by solving linear programs. Two approaches for generating the rule conjectures, specific-to-general and general-to-specific, are described. The validity of these conjectures are checked with VI-analysis.

R. Setiono [10] proposes a rule extraction algorithm for extracting rules from pruned ANNs for breast cancer diagnosis. The author describes how the activation values of a hidden node can be clustered such that only a finite and usually small number of discrete values need to be considered while at the same time maintaining the network accuracy. A small number of different discrete activation values and a small number of connections from the inputs to the hidden nodes will yield a set of compact rules for problem.

R. Setiono proposes a rule extraction algorithm named NeuroRule [11]. This algorithm extracts symbolic classification rule from a pruned network with a single hidden layer in two steps. First, rules that explain the network outputs are generated in terms of the discretized activation values of the hidden nodes. Second, rules that explain the discretized hidden node activation values are generated in terms of the network inputs. When two sets of rules are merged, a DNF representation of network classification is obtained. Under DNF representation, the classification concept is expressed as the disjunction of one or more subconcepts.

Ismail Taha and Joydeep Ghosh [12] proposes three rule extraction techniques for knowledge Based Neural Network (KBNN) hybrid systems and presents their implementation results. The suitability of each technique depends on the network type, input nature, complexity, the application nature, and the requirement transparency level. The first proposed approach (BIO-RE) is categorized as Black-box Rule Extraction (BRE) technique, while the second (Partial-RE) and third techniques (Full-RE) belong to Link Rule Extraction (LRE) category.

Binarized Input-Output Rule Extraction (BIO-RE) technique extracts a set of binary rules from any ANN regardless of its kind. Partial-RE extracts partial rules of most important embedded knowledge in MLP. The idea underlying Partial-RE algorithm is that it first sorts both positive and negative incoming links to all hidden and output nodes in descending order into two different sets based on their weight values. Starting from the highest positive weight (say i), it searches for individual incoming links that can cause a node j (hidden/output) to be active regardless of the other input links to this node. If such link exists, it generates a rule: If $Node_i \xrightarrow{cf} Node_j$, where cf represents the measure of belief in the extracted rule and is equal to the activation value of node j.

Like the Partial-RE approach, Full-RE falls in the LRE category. It is notable because: (i) It extracts rules with certainty factors from trained feedforward ANNs. (ii) It extracts all possible rules that present in the semantic interpretation of the internal structure of the trained ANN that they were extracted from. (iii) it is universal since there is no restriction on the values that any input feature can take. (iv) it is applicable to any ANN node with a monotonically increasing activation function.

Huan Liu [13] reports a new rule induction method that handles noise effectively. One of the purposes of a rule induction method is to find compact rules that generalize well on the data. The single-pattern-based-rule induction method works as follows: starting with a pattern, find a rule that differentiate the pattern from patterns of other classes, remove the patterns covered by that rule from the training sample, induce the next rule, and repeat this process until no patterns remain.

Andrzej Lozowski et al. [14] presents symbolic knowledge extraction from mapping/extrapolating ANNs. An algorithm to obtain crisp rules in the form of logical implications which roughly describe the ANN mapping is introduced. The number of extracted rules can be selected using an uncertainty margin parameter as well as by changing the precision of the soft quantization of the inputs. Crisp linguistic terminology and input partitioning can be used to provide finer resolution of input variables. However, a fuzzy methodology for handling various degrees of membership of objects in classes now becomes necessary.

R. Setiono [15] proposes a rule extraction (RX) algorithm to extract rules from a pruned ANN. The network is a standard feedforward backpropagation network with a single hidden layer that has been trained to meet a prespecified accuracy requirement. The process of extracting rules from a trained ANN can be made much easier if the complexity of the ANN has first been removed. The pruning process attempts to eliminate as many connections as possible from the ANN, while at the same time tries to maintain the prespecified accuracy rate. It is expected that less connections will result in more concise rules. No initial knowledge of the problem domain is required. Relevant and irrelevant attributes of the data are distinguished during the pruning process. Those that are relevant will be kept, others will be automatically discarded.

M. W. Craven and J. W. Shavlik develop an algorithm called TREPAN [16], for extracting comprehensible, symbolic representations from trained ANNs. Given a trained ANN, TREPAN produces a decision tree that approximates the concept represented by the ANN. The tree extracted by TREPAN nearly matches the accuracy of the ANN itself, and is fewer complexes than tree produced by conventional decision-tree induction algorithms running directly on the training data

Huan Liu [17] describes a family of rule generators that can be used to extract classification rules in various applications. It includes versions that can handle noise in data, produce perfect rules, and can induce order independent or dependent rules. The basic idea of the proposed algorithm is simple: using first order information in the data to determine shortest sufficient conditions in a pattern that can differentiate the pattern from patterns belonging to other classes. The sole use of first order information avoids the combinatorial complexity in computation, although it is well known that using higher order information may provide better results.

R. Setiono and W. K. Leow [18] proposes a method, Fast Extraction of Rules from Neural Networks (FERNN), for extracting symbolic rules from trained feedforward ANNs with a single hidden layer. The method does not require network pruning and hence no network retraining is necessary. Given a fully connected trained feedforward ANN with single hidden layer, FERNN first identifies the relevant hidden nodes by computing their information gains. For each relevant hidden node, its activation values is divided into two subintervals such that the information gain is maximized. FERNN finds the set of relevant ANN connections from the input nodes to the hidden nodes by checking the magnitudes of their weights. The connections with larger weights are identified as relevant. Finally, FERNN generates rules that distinguish the two subintervals of the hidden node activation values in terms of the network inputs.

R. Setiono [19] presents MofN3, a new method for extracting M-of-N rules from ANNs. The topology of the ANNs is the standard three-layered feedforward networks. Nodes in the input layer are connected only to the nodes in the hidden layer, while nodes in the hidden layer are also connected to nodes in the output layer. Given a hidden node of a trained ANN with N incoming connections, show how the value of M can be easily computed. In order to facilitate the process of extracting M-of-N rules, the attributes of the dataset have binary values –1 or 1.

R. Setiono, W. K. Leow and Jack M. Zurada [20] describes a method called rule extraction from function approximating neural networks (REFANN) for extracting rules from trained ANNs for nonlinear regression. It is shown that REFANN produces rules that are almost as accurate as the original networks from which the rules are extracted. For some problems, there are sufficiently few rules that useful knowledge about the problem domain can be gained. REFANN works on a network with a single layer and one linear output node.

Zhi-Hua Zhou et al. [21] proposes REFNE (Rule Extraction from Neural Network Ensemble) which is designed to extract symbolic rules from trained NN ensembles that perform classification tasks. REFNE utilizes trained ensembles to generate a number of instances. It could gracefully breaks the ties made by individual ANNs in prediction. Instead of discretizing all the continuous attributes at the beginning of the extraction of symbolic rules, REFNE adopts a specific discretization scheme so that different continuous attributes can be discretized to different number of intervals and unnecessary discretization can be avoided.

1.3 Problems of Existing Works

The problems of existing works are summarized as follows:

i) Use predefined and fixed number of hidden nodes that require human experience and prior knowledge of the problem to be solved.

ii) Clustering algorithms used to discretize the output values of hidden nodes are not efficient.

iii) Computationally expensive.

iv) Could not produce concise rules.

1.4 Objective of the Thesis

Multilayer feedforward ANNs trained by using the backpropagation-learning algorithm is limited to searching for a suitable set of weights in an a priori fixed network topology. This mandates the selection of an appropriate network topology for the learning problem on hand. Too small networks are unable to learn the problem well while overly large networks tend to overfit the training data, and consequently result in poor generalization performance.

This thesis proposes a hybrid approach with both constructive and pruning components for automatic determination of simplified ANN architectures. The objective of the thesis are summarized as follows:

i) To develop an efficient algorithm for extracting rules from ANNs to explain the functionality of ANNs.
ii) To find an efficient method for clustering the outputs of hidden nodes.
iii) To extract concise rules with high predictive accuracy.

1.5 Thesis Overview

The remaining chapters of this thesis are organized as follows:

- Chapter 2 provides background material for the rest of the thesis. Historical backgrounds of ANNs, human brain, biological basis of the ANNs, model of a neuron are first elaborated. ANN architectures, learning methods, characteristics of ANNs, and some application domains of ANNs are discussed next. Finally, the backpropagation training algorithm, which was used for training the ANNs are presented.
- Chapter 3 presents the proposed algorithm REANN: Rule Extraction from Artificial ANNs, which is the main contribution of this thesis. Descriptions of the REANN are first elaborated; detailed descriptions of different components used in REANN are then described.
- Chapter 4 presents a detailed experimental evaluation of REANN. In the reported experiments, REANN is applied to ANNs that were trained to solve classification problems. This chapter evaluates REANN performance on several well-known benchmark classification problems. Experimental details, results, comparison, and discussion are described.
- Finally, in chapter 5, the contributions and limitations of the research are presented. The directions of future works to remove the limitations of present work are proposed.

Chapter 2

Background

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.

Various learning mechanisms exist to enable the ANN acquire knowledge. ANN architectures have been classified into various types based on their learning mechanisms and other features. Some classes of ANN refer to this learning process as training and the ability to solve a problem using the knowledge acquired as inference.

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 Systems (ANS) technology or Artificial Neural Networks or simply ANNs. In the literature, this technology is also referred to as Connectionist Networks, Neurocomputers, Parallel

Distributed Processors etc. Also neurons are referred to as neurodes, Processing Elements (PEs), and nodes.

2.2 Historical Backgrounds of ANNs

The modern era of ANNs began with the pioneering work of McCulloch and Pitts [22]. McCulloch and Pitts describe a logical calculus of ANNs that united the studies of neuropsychology and mathematical logic. McCulloch and Pitts showed that a network so constituted would, in principle, compute and computable function. This was a very significant result and with it, it is generally agreed that the disciplines of ANNs and of artificial intelligence were born.

In 1948, Wiener's famous book Cybernetics [23] was published, describing some important concepts for control, communications, and statistical signal processing. The second edition of this book was published in 1961, adding new material on learning and self-organization. In chapter 2 of both editions of this book, Wiener appears to grasp the physical significance of statistical mechanics in the context of the subject matter, but it was left to Hopfield (more than 30 years latter) to bring the linkage between statistical mechanics and learning systems to full fruition.

The next major development in ANNs came in 1949 with the publication of Hebb's book [24] *The Organization of Behavior*, in which an explicit statement of a psychological learning rule for synaptic modification was presented for the first time. Specially, Hebb proposed that the connectivity of the brain is continually changing as organism learners differing functional tasks, and that neural assemblies are created by such changes. Hebb's book was immensely influential among psychologists, but unfortunately it had little or no impact on the engineering community.

In 1952, Ashby's book, *Design for a Brain: The Origin of Adaptive Behavior*, was published, which is just as fascinating to read today, as it must have been then. The book was concerned with the basic notion that adaptive behavior is not inborn but rather learned, and that through learning the behavior of an animal (system) usually changed for the better. The book emphasized the dynamic aspects of the living organism as a machine and the related concept of stability.

In 1954, Miskey wrote a "neural network" doctorate thesis at Princeton University, which was entitled "Theory of Neural-Analog Reinforcement System and Its Application to the Brain Model Problem." In 1961, an excellent early paper by Minsky on AI entitled "Steps Toward Artificial Intelligence," was published; this latter paper contains a large section what is now termed ANNs. In 1967 Minsky's book, *Computation: Finite and Infinite Machines,* was published. This clearly written book extended the 1943 results of McCulloch and Pitts and put them in the context of automata theory and theory of computation.

Some 15 years after the publication of McCulloch and Pitt's classic paper, a new approach to the pattern recognition problem was introduced by Rosenblatt [25] in his on the perceptron, a novel method of supervised learning. The crowning achievement of Rosenblatt's work was the so-called perceptron convergence theorem, the first proof for which was outlined by Rosenblatt [26]; proofs of the theorem also appeared in Novikoff and others. In 1960, Widrow and Hoff [27] introduced the least mean square (LMS) algorithm and used it to formulate the Adaline (adaptive linear element). The difference between the perceptron and the Adaline lies in the training procedure. One of the earliest trainable-layered ANNs with multiple adaptive elements was the Madaline (multiple Adeline) structure proposed by Widrow and his students [28]. In 1967, Amari [29] used the stochastic gradient method for adaptive pattern classification. In 1965, Nilsson's book [30], *Learning Machines*, was published, which is still the best-written exposition of linearly separable patterns in hypersurfaces. During the classical period of the perceptron in the 1960s, it seemed as if ANNs could do anything. But then came the book by Minsky and Papert [31], who used mathematics to demonstrate that there are fundamental limits on what single layer perceptron can compute. In a brief section on multiplayer perceptrons, they stated that there was no reason to assume that any of the limitations of single layer perceptrons could be overcome in the multiplayer version.

An important activity that did emerge in the 1970s was self-organizing maps using competitive learning. The computer simulation work done by von der Malsburg [32] was perhaps the first to demonstrate self-organization. In 1976 Willshaw and von der Malsburg [33] published the first paper on the formation of self-organizing maps, motivated by topological ordered maps in the brain.

In 1980s major contributions to the theory and design of ANNs were made on several fronts, and with it there was a resurgence of interest in ANNs. Grossberg [34], building on his earlier work on competitive learning [35-36], established a new principle of self-organization known as adaptive resonance theory (ART). Basically, the theory involves a bottom up recognition layer and a top down generative layer. If the input pattern and learned feedback pattern match, a dynamical state called "adaptive resonance" (i.e., amplification and prolongation of neural activity) takes place. This principle of forward/backward projections has been rediscovered by other investigators under different guises.

An important development in 1982 was the publication of Kohonen's paper on self organizing maps [37] using a one-or two-dimensional lattice structure, which was different in some respects from the earlier work by Willshaw and von der Malsburg. Kohonen's model has received far more attention in an analytic context and with respect to applications in the literature, than the Willshaw-von der Malsburg model, and has become the benchmark against which other innovations in this field are evaluated.

In 1983, Kirkpatrick, Gelatt, and Vecchi [38] described a new procedure called simulated annealing, for solving combinatorial optimization problems. Simulated annealing is rooted in statistical mechanics. It is based on a simple technique that was first used in computer simulation by Metropolis et al. [39]. The idea of simulated annealing was later used by Ackley, Hinton, and Sejnowski in the development of a stochastic machine known as Boltzmann machine, which was the first successful realization of a multiplayer ANN.

In 1986 the development of the back-propagation algorithm was reported by Rumelhart, Hinton, and Williams [40]. In that same year, the celebrated two-volume book, Parallel Distributed Processing: Explorations in Microstructure of Cognition [41], edited by Rumelhart and McClelland, was published. This latter book has been a major influence in the use of back-propagation learning, which has emerged as the most popular learning algorithm for the training of multiplayer perceptrons. In fact, back propagation learning was discovered independently in two other places about the same time. After the discovery of the back-propagation algorithm in the mid-1980s, it turned out that the algorithm had been described earlier by Werbos in his Ph.D. thesis at Harvard University in August 1974 [42]; Werbos's Ph.D. thesis was first documented description of efficient reverse-mode gradient computation that was applied to general network models with ANNs arising as a special case. The basic idea of back-propagation may be traced further back to the book Applied Optimal Control by Bryson and Ho [43]. In section 2.2 entitled "Multistage Systems" of the book, a derivation of back-propagation using a Lagrangian formalism is described. In the final analysis, however, much of the credit for the back-propagation algorithm has to be given to Rumelhart, Hinton, and Williams for proposing its use for machine learning and for demonstrating how it could work.

In 1988, Broomhead and Lowe [44] described a procedure for design of layered feedforward networks using radial basis functions (RBF), which provide an alternative to multilayer perceptrons. The basic idea of radial basis functions goes back at least to the method of potential functions that was originally proposed by Bashkirov, Braverman, and Muchnik [45], and the theoretical properties of which were developed by Aizeman, Braverman, and Rozonoer [46-47]. A description of the method of potential functions is presented in the class book, Pattern Classification and Scene Analysis, by Duda and Hart [48]. Nevertheless, the paper by Broomhead and Lowe has led to a great deal of research effort linking the design of ANNs to an important area in numerical analysis and also linear adaptive filters. In 1990, Poggio and Girosi [49] further enriched the theory of RBF networks by applying Tikhonov's regularization theory.

In 1989, Mead's book, Analog VLSI and Neural Systems [50], was published. This book provides an unusual mix of concepts drawn from neurobiology and VLSI technology. Above all, it includes chapters on silicon retina and silicon cochlea, written by Mead and coworkers, which are vivid examples of Mead's creative mind.

In the early 1990s, Vapnik and coworkers invented a computationally powerful class of supervised learning networks, called support vector machines, for solving pattern recognition, regression, and density estimation problem [51-54]. This new method is based on results in the theory of learning with finite sample sizes. A novel feature of support vector machines is the natural way in which the Vapnik-Chervonekis (VC) dimension is embodied in their design. The VC dimension provides a measure for the capacity of a ANN to learn form a set of examples [55-56].

2.3 Human Brain

The human nervous system may be viewed as a three-stage system, as depicted in the block diagram of Fig. 2.1 [57]. 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 arrow pointing from right to left signify the presence of feed-back in the system. 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.

The struggle to understand the brain has been made easier because of the pioneering work of Ramón y Cajál [58], 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.

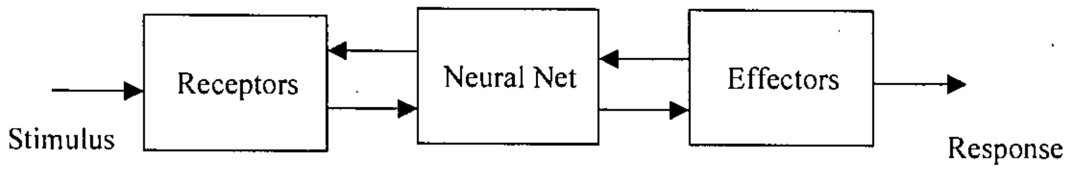


Fig. 2.1 Block diagram representation of human nervous system.

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 [59]. The net result is that there is an enormously efficient structure. The energetic efficiency of the brain is approximately 10^{-16} Jules/operation/second, whereas the best computers used today is about 10^{-6} Jules/operation/second [60].

2.4 Biological Basis of ANNs

The human brain is a very complex system capable of thinking, remembering and problem solving. There have been many attempts to emulate brain functions with computer models, and although there have been some rather spectacular achievements coming from these efforts, all of the models developed to date pale into oblivion when compared with the complex functioning of the human brain.

A neuron is the fundamental unit 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.

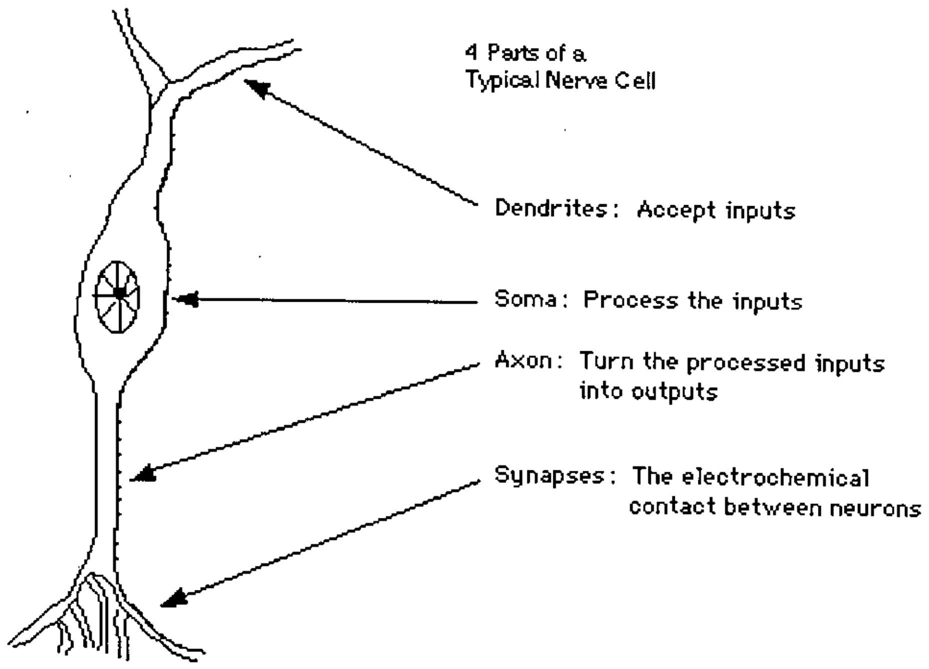

4 Parts of a
Typical Nerve Cell
Dendrites: Accept inputs
Soma: Process the inputs
Axon: Turn the processed inputs into outputs
Synapses: The electrochemical contact between neurons

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.

2.5 Model of a Neuron

A neuron is an information-processing unit that is fundamental to the operation of an 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 discuss below:

i) *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}. It is important to make a note of a manner in which the subscripts of the synaptic weight w_{kj} are written. The first subscript refers to the neuron in question and the second subscript refers to the input end of the synapse to which the weight refers. 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.

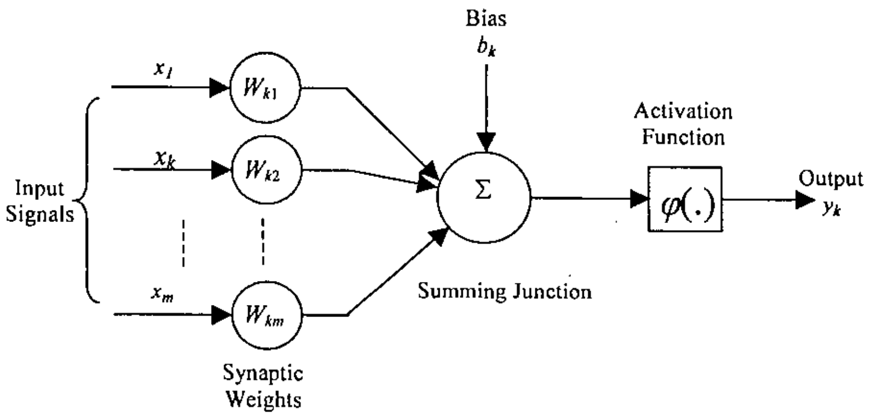


Fig. 2.3 Model of a neuron.

ii) *An adder* for summing the input signals, weighted by the respective synapse of the neuron; the operations described here constitute a linear combiner.

iii) *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:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{2.1}$$

and

$$y_k = \varphi(u_k + b_k) \tag{2.2}$$

where $x_1, x_2, \ldots, x_m$ are the input signals; $w_{k1}, w_{k2}, \ldots, w_{km}$ are the synaptic weights of neuron k; u_k is the linear combiner due to the input signals; b_k is the bias; $\varphi(.)$ 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

$$v_k = u_k + b_k \tag{2.3}$$

In particular, depending on whenever the bias b_k is positive or negative, the relationship between the induced local field or activation potential v_k of neuron k and the linear combiner output u_k is modified. The bias b_k is an external parameter of artificial neuron k. Formulate the combinations of Eqs. (2.1) to (2.3) as follows:

$$v_k = \sum_{j=0}^{m} w_{kj} x_j \tag{2.4}$$

and

$$y_k = \varphi(v_k) \tag{2.5}$$

2.5.1 Types of Activation Function

The activation function, denoted by φ(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:

2.5.1.1 Threshold Function

For this type of activation function, described in Fig. 2.4 we have

$$\varphi(v) = \begin{cases} 1 & if\, v \geq 0 \\ 0 & if\, v < 0 \end{cases} \tag{2.6}$$

In engineering literature, this type of threshold function is commonly referred to as Heaviside function. Correspondingly, the output of neuron k employing such a threshold function is expressed as

$$y_k = \begin{cases} 1 & if\, v_k \geq 0 \\ 0 & if\, v_k < 0 \end{cases} \tag{2.7}$$

where v_k is the induced local field of the neuron; that is

$$v_k = \sum_{j=1}^{m} w_{kj} x_j + b_k \tag{2.8}$$

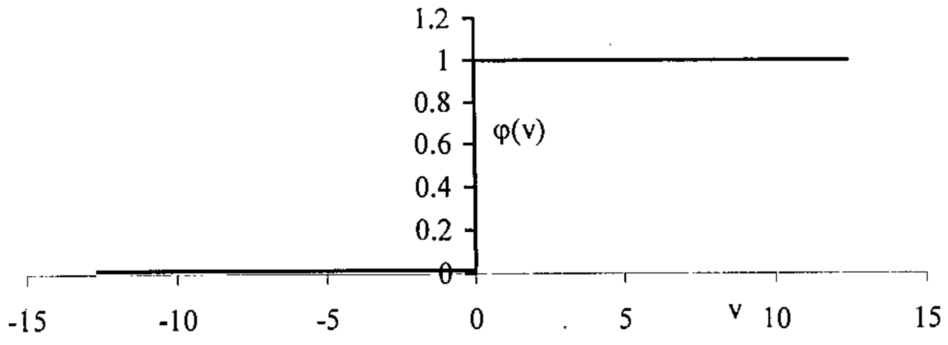


Fig. 2.4 Threshold function.

2.5.1.2 Piecewise-Linear Function

For the piece-wise linear function described in Fig. 2.5 we have

$$\varphi(v) = \begin{cases} 1, & v \geq +\frac{1}{2} \\ v, & +\frac{1}{2} > v > -\frac{1}{2} \\ 0, & v \leq -\frac{1}{2} \end{cases} \tag{2.9}$$

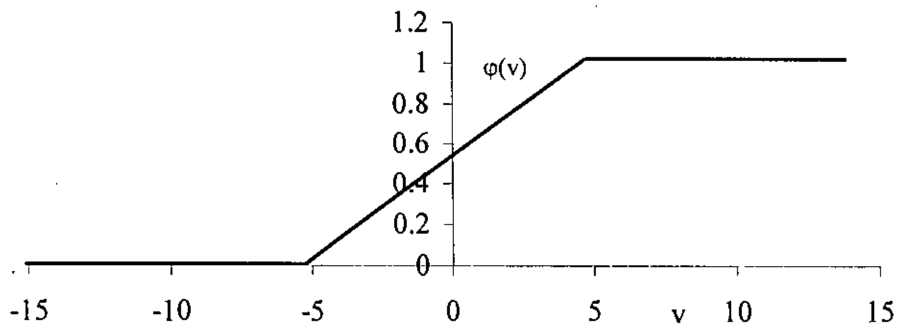


Fig. 2.5 Piece-wise linear function.

2.5.1.3 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

$$\varphi(v) = \frac{1}{1+\exp(-av)} \tag{2.10}$$

where a is the slope parameter of the sigmoid function. By varying the parameter a, obtain sigmoid functions of different slopes. In the limit, as the slope parameter approaches infinity, the sigmoid function becomes simply a threshold function. Whereas a threshold function assumes the value 0 or 1, a sigmoid function assumes a continuous range of values from 0 to 1. More important point is that the sigmoid function is differentiable, whereas the threshold function is not.

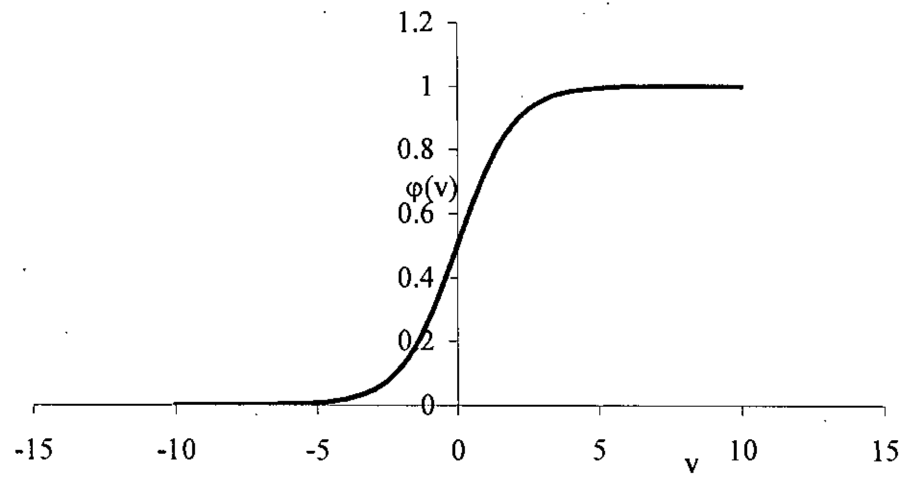


Fig. 2.6 Logistic function.

The activation functions defined by Eqs. (2.6), (2.9), and (2.10) range from 0 to +1. It is sometimes desirable to have the activation function range from −1 to +1, in which case the activation function assumes an antisymmetric form with respect to the origin; that is, the activation function is an odd function of the induced local field. Specifically, the threshold function of Eqs. (2.6) is now defined as

$$\varphi(v) = \begin{cases} 1 & \textit{if } v > 0 \\ 0 & \textit{if } v = 0 \\ -1 & \textit{if } v < 0 \end{cases} \tag{2.11}$$

which is commonly referred to as the signum function. For the corresponding form of a sigmoid function hyperbolic tangent function described in Fig. 2.7 may use, defined by

$$\varphi(v) = \tanh(v) = \frac{e^{av} - e^{-av}}{e^{av} + e^{-av}} \tag{2.12}$$

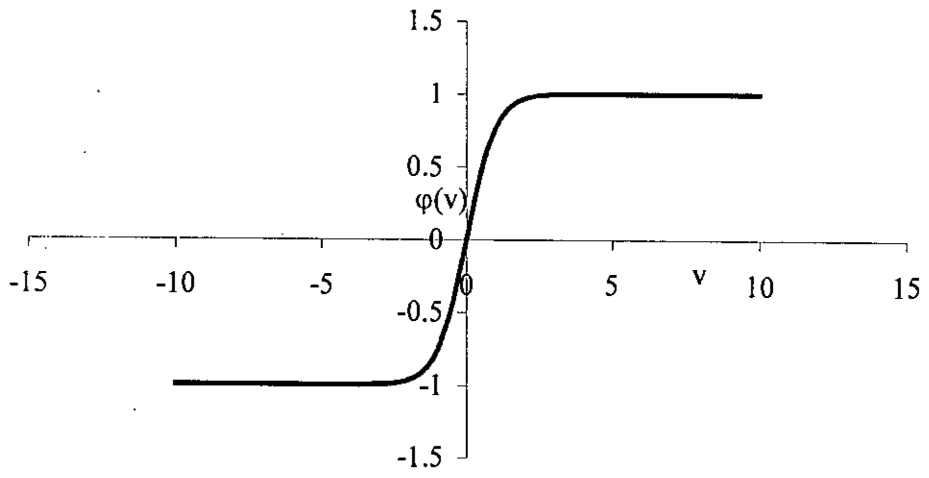


Fig. 2.7 Hyperbolic tangent function.

2.6 ANN Architectures

The manner in which the nodes of a network are structured is intimately linked with the learning algorithm used to train the network. Three fundamental different classes of network architectures are describe below:

2.6.1 Single Layer Feedforward Networks

In a layered ANN the nodes organized in the form of layers. In the simplest form of a layered network, we have an input layer of source node that projects onto an output layer of nodes, but not vice versa. In other words this network is strictly a feedforward or acyclic type. It is illustrated in the Fig. 2.8 for the case of four nodes in both the input and output layers. Such a network is called single layer network, with the designation "single layer" referring to the output layer of computation nodes (neurons).

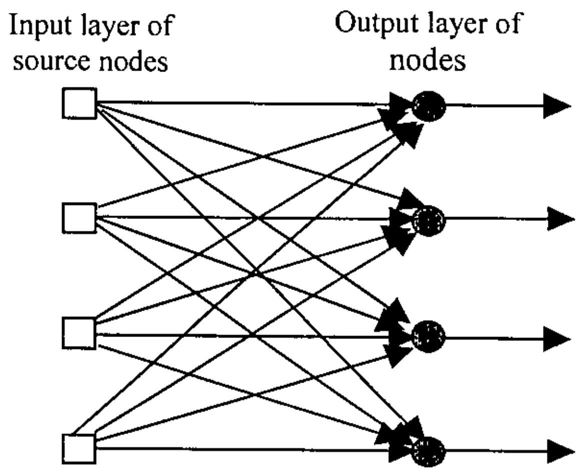


Fig. 2.8 Single layer feedforward network.

2.6.2 Multilayer Feedforward Networks

The second class of feedforward network distinguishes itself by the presence of one or more hidden layers, whose computation nodes are correspondingly called hidden nodes. The function of hidden nodes is to intervene between the external input and the network output in some useful manner. By adding one or more hidden layers, the network is enabled to extract higher order statistics.

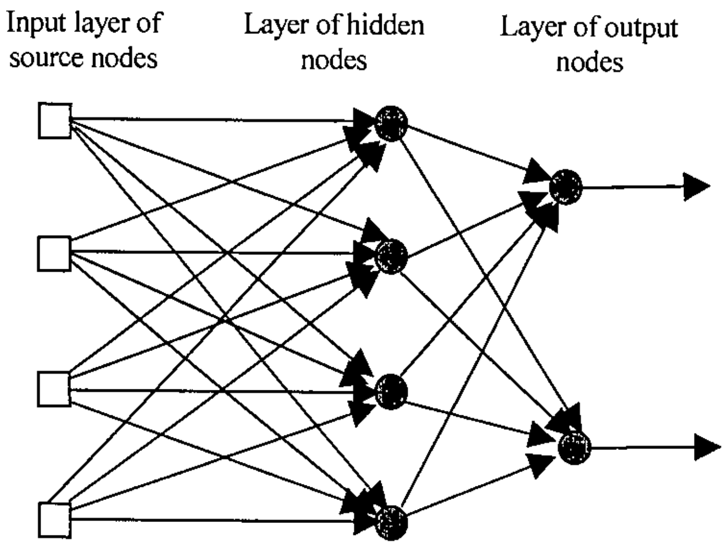


Fig. 2.9 Multilayer feedforward network.

2.6.3 Recurrent Networks

A recurrent network distinguishes itself from a feedforward network in that it has at least one feedback loop. For example, a recurrent network may consist of a single layer of nodes with each node feeding its output signals back to the inputs of all other nodes as illustrated in the architecture graph in Fig. 2.10. In the structure depicted in this figure there are no self-feedback loops in the network; self-feedback refers to a situation where the output of a node is fed back into its own input.

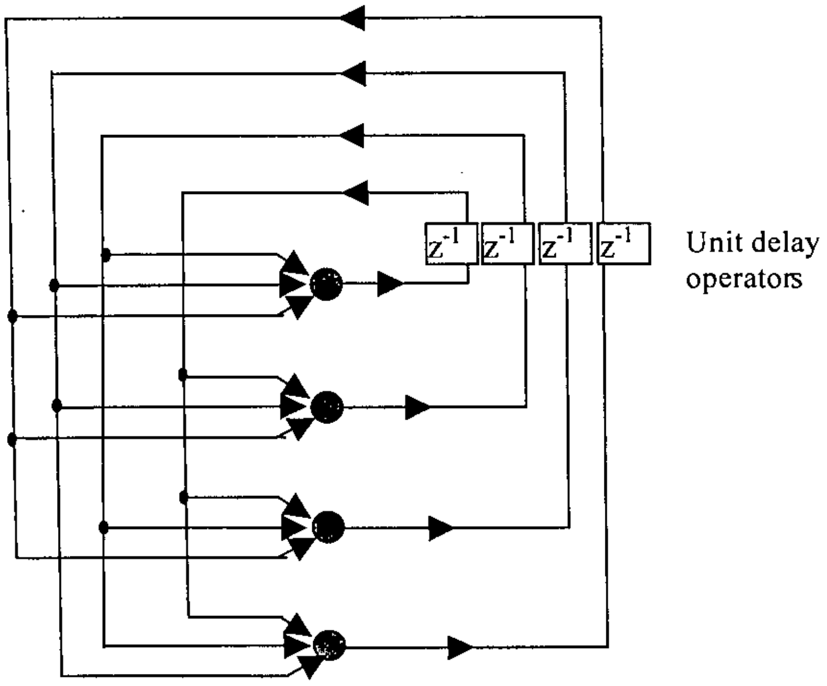


Fig. 2.10 Recurrent network with no self-feedback loops.

2.6.4 Architecture Determination

The architecture of an ANN depends on the number of nodes in the input layer, hidden layer, and output layer. The number of nodes in the input and output layers is the same as the number of inputs and outputs of the problem. Generally, the number of nodes in the hidden layer is determined based on previous experience in traditional backpropagation.

It has been known that the performance of ANNs is greatly dependent on their architecture. Too small networks are unable to learn the problem well while overly large networks tend to overfit the training data, and consequently result in poor generalization performance. In practice, a variety of architectures are tried out and the one that appears best suited to the given problem is picked. Such a trial-and-error approach is not only computationally expensive but also does not guarantee that the selected network architecture will be close to optimal or will generalize well. This suggests the need for algorithms that learn both the network topology and the weights.

The automated design of ANNs is an important issue for any learning task. There have been many attempts to design ANNs automatically, such as constructive, pruning, and evolutionary algorithm. The important parameters of any design algorithms are the consideration of generalization ability and of training time of ANNs.

The main difficulty of evolutionary algorithm is that they are quite demanding in both time and user defined parameters. In contrast, constructive algorithm requires much smaller amounts of time and used defined parameters. The most well known constructive and pruning algorithms to determine ANNs architecture automatically are described in the next chapter.

2.7 Learning Methods

Learning is a process by which the free parameters of an ANN are adapted through a process of stimulation by the environment is which the network is embedded. The method of learning is determined by the manner in which the parameter changes take place. Learning methods in ANNs can be broadly classified into three basic types: supervised, unsupervised, and reinforced.

2.7.1 Supervised Learning

In supervised learning both inputs and outputs are provided. The network 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 network. 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 a network 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 backpropagation algorithm.

2.7.2 Unsupervised Learning

The other type of training is called unsupervised learning. In unsupervised learning, the network 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.

2.7.3 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 network 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.

2.8 Characteristics of ANNs

ANNs have profound strengths and weaknesses, and these must be recognized if they are to be used properly. Although ANNs are sometimes called neural computers, they are infact not computers; but rather, they are basically memories that memorize results, just as human brain memories certain results. ANNs use memory-based storage of information in ways that are different and more flexible than simple storage in a look-up table. In the ANN, as in the brain, the storage of information is distributed through the network. Although this makes it hard to keep things separate that should be kept separate, it does give rise to the ANNs ability to make generalizations that are so important to the practical applications of ANNs. The characteristics of ANNs are described as follows:

i) The ANNs exhibit mapping capabilities, that is, they can map input patterns to their associated output patterns.

ii) The 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.

iii) The ANNs process the capability to generalize. Thus, they can predict new outcomes from past trends.

iv) The ANNs are robust systems and are fault tolerant. They can, therefore, recall full patterns from incomplete, partial or noisy patterns.

v) The ANNs can process information in partial, at high speed, and in a distributed manner.

2.9 Some Application Domains for ANNs

Using ANNs can solve many computationally intensive problems in pattern recognition and classification. These problems include character recognition, speech recognition, speaker recognition, medical diagnosis, financial data analysis, finger print identification, photographic image identification, image and signal restoration, chromosome classification, and robot movement and manipulation.

2.10 Backpropagation Algorithm

The backpropagation (BP) network is a layered, feedforward network that is fully interconnected by layers. Thus, there are no feedback connections and connections that bypass one layer to go directly to a latter layer. Although only three layers are used in the discussion, more than one hidden layer is permissible.

The BP learning algorithm involves two phases. During the first phase the input is presented and propagated forward through the network to compute the output value o_{pk} for each node. This output is then compared with the targets, resulting an error signal for each output node. The second phase involves a backward pass through the network during which the error signal δ_{pk} is passed to each node in the network 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.

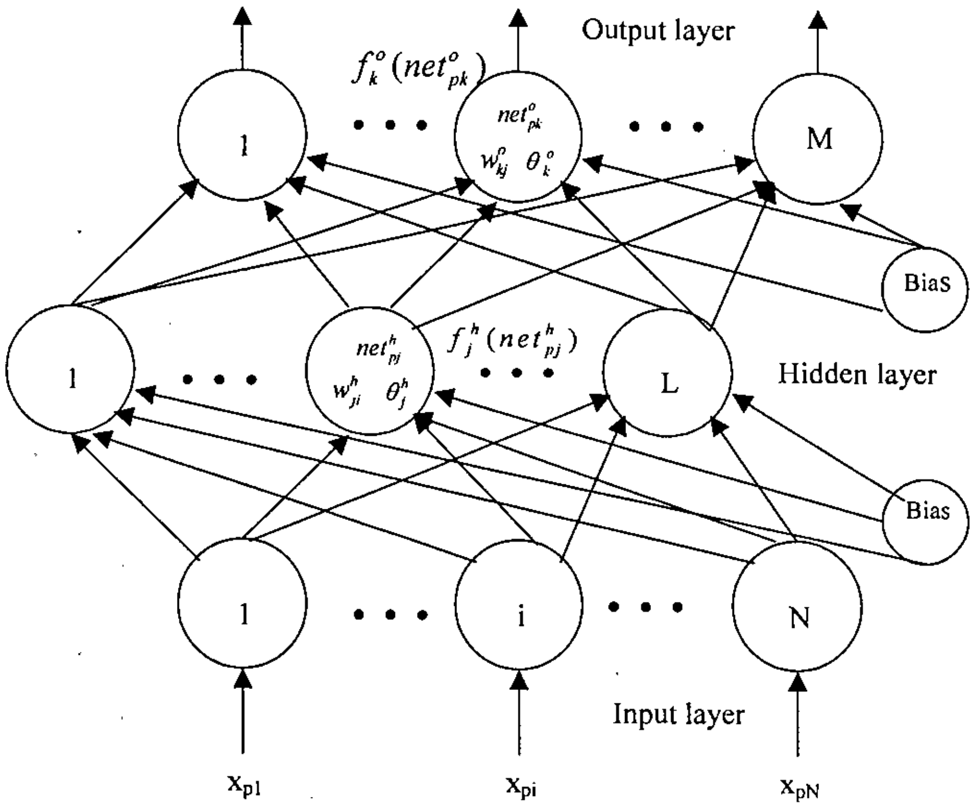


Fig. 2.11 The three-layer BP network architecture.

The significance of the process is that, as the network trains, the nodes in the intermediate layers organize themselves such that different nodes learn to recognize different features of the total input space. After training, when presented with an arbitrary input pattern that is noisy or incomplete, the nodes in the hidden layer of the network will respond with an active output if the new input contains a pattern that resembles the feature the individual node learns to recognize during training.

Let us consider an input vector, $x_p = (x_1, x_2, \ldots, x_{pn})^t$, is applied to the input layer of the network. The "p" subscript refers to the p training vector. The input nodes distribute the values to the hidden layer nodes. The net input to the jth hidden node is,

$$net^h_{pj} = \sum_{i=1}^{N} w^h_{ji} x_{pi} + \theta^h_j \tag{2.13}$$

Where w^h_{ji} is the weight on the connection from ith input node to jth hidden node, and θ^h_j 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,

$$i_{pj} = f^h_j(net^h_{pj}) \tag{2.14}$$

Where the function $f^h_j(net^h_{pj})$ is referred to as an activation function.

The equations of the output nodes are,

$$net^0_{pk} = \sum_{j=1}^{L} w^o_{kj} i_{pj} + \theta^o_k \tag{2.15}$$

$$o_{pk} = f^o_k(net^o_{pk}) \tag{2.16}$$

Where "o" superscript refers to quantities on the output layer.

2.10.1 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 pth training vector, and "k" refers to the kth output node. In this case

y_{pk} is the desired output and o_{pk} is the actual output of the kth node. The error to be minimized is the sum of the squares of the errors for all output nodes:

$$E_p = \frac{1}{2}\sum_{k=1}^{M} \delta_{pk}^2 \tag{2.17}$$

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 adjust 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. (2.17) and the definition of δ_{pk}

$$E_p = \frac{1}{2}\sum_{k=1} (y_{pk} - o_{pk})^2 \tag{2.18}$$

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

Where Eq. (2.16) is used for the output value, o_{pk}, and the chain rule for partial derivatives. The last factor of Equation (2.19) is,

$$\frac{\partial (net_{pk}^o)}{\partial w_{kj}^o} = \frac{\partial}{\partial w_{kj}^o} \sum_{j=1}^{L} w_{kj}^o i_{pj} + \theta_k^o = i_{pj} \tag{2.20}$$

Combining Eqs. (2.19) and (2.20), the negative gradient,

$$-\frac{\partial E_p}{\partial w_{kj}^o} = (y_{pk} - o_{pk}) f_k^{o\prime} (net_{pk}^o) i_{pj} \tag{2.21}$$

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

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \Delta_p w_{kj}^o(t) \tag{2.22}$$

Where,

$$\Delta_p w_{kj}^o = \eta (y_{pk} - o_{pk}) f_k^{o\prime} (net_{pk}^o) i_{pj} \tag{2.23}$$

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

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \eta(y_{pk} - o_{pk})o_{pk}(1 - o_{pk})i_{pj} \tag{2.24}$$

By defining output layer error term,

$$\delta_{pk}^o = \eta(y_{pk} - o_{pk})f_k^{o'}(net_{pk}^o) = \delta_{pk} f_k^{o'}(net_{pk}^o) \tag{2.25}$$

By combining Eqs. (2.24) and (2.25) the weight update equation becomes,

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \eta\delta_{pk}^o i_{pj} \tag{2.26}$$

2.10.2 Update of Hidden-Layer Weights

The error of the hidden layer is given by,

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

$$= \frac{1}{2}\sum_k (y_{pk} - f_k^o(net_{pk}^o))^2$$

$$= \frac{1}{2}\sum_k (y_{pk} - f_k^o(\sum_j w_{kj}^o i_{pj} + \theta_k^o))^2 \tag{2.27}$$

The gradient of E_p with respect to hidden layer weights,

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

$$= -\sum_k (y_{pk} - o_{pk}) \frac{\partial o_{pk}}{\partial(net_{pk}^o)} \frac{\partial(net_{pk}^o)}{\partial i_{pj}} \frac{\partial i_{pj}}{\partial(net_{pj}^h)} \frac{\partial(net_{pj}^h)}{\partial w_{ji}^h} \tag{2.28}$$

Each of the factors of Eq. (2.28) can be calculated explicitly from previous equations. The result is,

$$\frac{\partial E_p}{\partial w_{ji}^h} = -\sum_k (y_{pk} - o_{pk}) f_k^{o'}(net_{pk}^o) w_{kj}^o f_j^{h'}(net_{pj}^h) x_{pi} \tag{2.29}$$

The hidden layer weights update in proportion to negative of the Eq. (2.29)

$$\Delta_p w_{ji}^h = \eta f_j^{h'}(net_{pj}^h) x_{pi} \sum_k (y_{pk} - o_{pk}) f_k^{o'}(net_{pk}^o) w_{kj}^o$$

By using Equation (2.25),.

$$\Delta_p w_{ji}^h = \eta f_j^{h'}(net_{pj}^h) x_{pi} \sum_k \delta_{pk}^o w_{kj}^o \tag{2.30}$$

Every weight update on the hidden layer depends on all error terms, δ^o_{pk}, 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,

$$\delta^h_{pj} = f^{h'}_j(net^h_{pj}) \sum_k \delta^o_{pk} w^o_{kj} \tag{2.31}$$

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

$$w^h_{ji}(t+1) = w^h_{ji}(t) + \eta \delta^h_{pj} x_{pi} \tag{2.32}$$

The amount of weight adjustment depends on three factors: δ, η, x. The size of the weight adjustment is proportional to δ, the error value of the node. Thus a larger error value of that node results in the larger adjustments to its incoming weights.

The weight adjustment is also proportional to x, the output value for that originating node. If this output value is small, then the weight adjustment is small. If the output value is large, then the weight adjustment is large. Thus a higher activation value for incoming node results in a larger adjustment to this outgoing weight.

The value of η is commonly between [0.1, 1.0] is chosen by the network user, and usually reflect the rate of learning of the network to ensure that the network will settle to a solution. A small value of η means that the network will have make a large number of iterations, but that is the price to be paid. It is often possible to increase the size of η as learning proceeds. Increasing η as the network error decreases will often help to speed convergence by increasing the step size as the error reaches a minimum, but the network may bounce around too far from the actual minimum value if η gets too large.

Chapter 3

Rule Extraction from ANNs (REANN)

3.1 Introduction

It is becoming increasingly apparent that without some form of explanation capability, the full potential of ANNs may not be realized. The rapid and successful proliferation of applications incorporating ANNs in many fields, such as commerce, science, industry, medicine etc., offers a clear testament to the capability of ANN paradigm. Extracting rules from trained ANN is one of the promising areas that are commonly used to explain the functionality of ANNs. The aim of this chapter is to introduce a new algorithm to extract rules from trained ANNs. The new algorithm is known as rule extraction from ANNs (REANN). Detailed description of REANN and its different components are presented in this chapter.

3.2 The REANN Algorithm

A standard three-layer feedforward ANN is the basis of the proposed algorithm REANN. The hyperbolic tangent function $\delta(y) = \frac{e^y - e^{-y}}{e^y + e^{-y}}$, which can take any value in the interval [-1, 1], is used as the hidden node activation function. Rules are extracted from near optimal ANN by using a new rule extraction algorithm, REx. The aim of REANN is to search for simple rules with high predictive accuracy. REANN uses first order information of training examples to find a smaller number of conditions that can differentiate one examples of a particular class with other examples of different classes. The reason for using first order information is to avoid the combinatorial complexity in computation. However, the use of second or higher order information may provide better results with high computational cost.

In comparison with other existing algorithms in the literature, the major advantages of REANN include i) it determines the near optimal architecture automatically by using a constructive-pruning strategy ii) it uses an efficient method to discretize the output values of hidden nodes iii) it is computationally inexpensive and iv) the extracted rules are concise, comprehensible, order insensitive and highly accurate.

The major steps of REANN 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, an output, and a hidden layer. The number of nodes in the input and output layers is the same as the number of inputs and outputs of the problem. Initially, the hidden layer contains only one node. The number of nodes in the hidden layer is automatically determined by using a basic constructive algorithm. Randomly initialize all connection weights within a certain small range.

Step 2 Remove redundant input nodes, and connections between input nodes and hidden nodes and between hidden nodes and output nodes by using a basic pruning algorithm. When pruning is completed, the ANN architecture contains only important nodes and connections. This architecture is saved for the next step.

Step 3 Discretize the outputs of hidden nodes by using an efficient heuristic clustering algorithm. The reason for discretization is that the outputs of hidden nodes are continuous, thus rules are not readily extractable from the ANN.

Step 4 Generate rules that map the inputs and outputs relationships. The task of the rule generation is accomplished in three phases. In the first phase, rules are generated by using rule extraction algorithm, REx, to describe the outputs of ANN in terms of the discretized output values of the hidden nodes. In the second phase, rules are generated by REx, to describe the discretized output values of the hidden nodes in terms of the inputs. Finally in the third phase, rules are generated by combining the rules generated in first and second phase.

Step 5 Prune redundant rules generated in Step 4. Replace specific rules with more general ones.

Step 6 Check the classification accuracy of the network. If the accuracy falls below an acceptable level, i.e. rule pruning is not successful then stop. Otherwise go to Step 5.

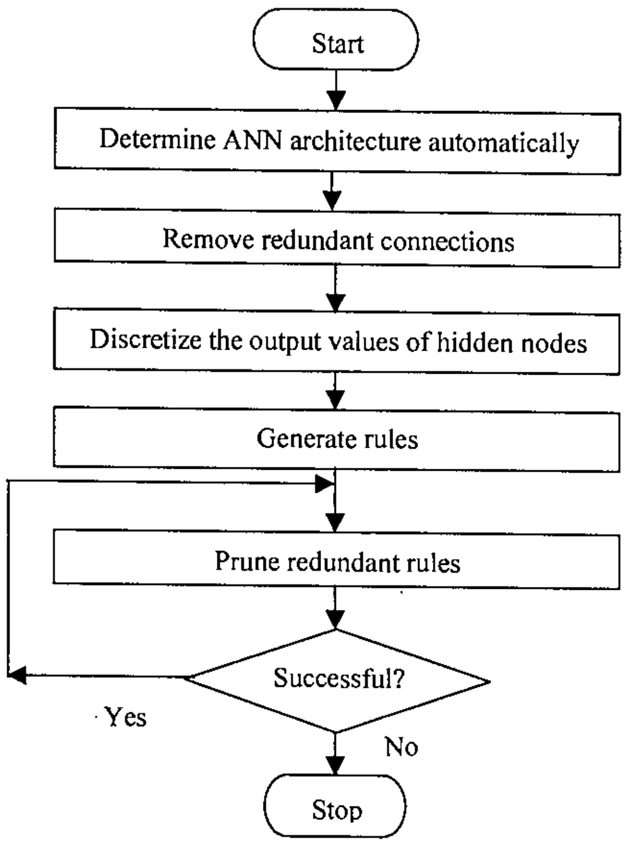

Start
Determine ANN architecture automatically
Remove redundant connections
Discretize the output values of hidden nodes
Generate rules
Prune redundant rules
Successful?
Yes
No
Stop

Fig. 3.1 Flow chart of the REANN algorithm.

The rules extracted by REANN are compact and comprehensible, and do not involve any weight values. The accuracy of the rules from pruned networks is high as the accuracy of the original networks. The important features of REANN are the rule generated by REx is recursive in nature and is order insensitive, i.e, the rules need not be required to fire sequentially.

3.2.1 Constructive Algorithm

One drawback of the traditional backpropagation algorithm is the need to determine the number of nodes in the hidden layer prior to training. To overcome this difficulty, many algorithms that construct a network dynamically have been proposed [61-63]. The most well known constructive algorithms are dynamic node creation (DNC) [64], feedforward neural network construction (FNNC) algorithm and the cascade correlation (CC) algorithm [65].

The constructive algorithm used in REANN is based on the feedforward neural network construction (FNNC) algorithm proposed by Rudy Setiono and Huan Liu [66]. In FNNCA the training process is stopped when the classification accuracy on the training set is 100% [67]. However, it is not possible to get 100% classification accuracy for most of the benchmark classification problems. In addition, higher classification accuracy on the training set does not guarantee the higher generalization ability i.e. classification accuracy on the testing set. Thus a validation set is used in this study to stop the training of the network.

The major steps of constructive algorithm used in REANN are summarized in Fig. 3.2, which are explained further as follows:

Step 1 Create an initial ANN consisting of 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 as the number of inputs and outputs of the problem. Initially the hidden layer contains only one node i.e. h=1. Randomly initialize all connection weights within a certain range.

Step 2 Train the network on the training set by using BP algorithm until the error is almost constant for a certain number of training epochs, τ, is specified by the user.

Step 3 Compute the ANN error E on validation set. If E is found unacceptable (i.e., too large), then assume that the ANN has inappropriate architecture, and go to the next step. Otherwise stop the training process. The ANN error E is calculated according to the following equations:

$$E(w,v) = \frac{1}{2}\sum_{i=1}^{k}\sum_{p=1}^{C}(S_{pi} - t_{pi})^2 \tag{3.1}$$

where, k is the number of patterns, C is the number of output nodes, and t_{pi} is the target value for pattern x_i at output node p. S_{pi} is the output of the network at output node p.

$$S_{pi} = \sigma(\sum_{m=1}^{h}\delta((x_i)^T w_m)v_{pm}) \tag{3.2}$$

h is the number of hidden nodes in the network, x_i is an n-dimensional input pattern, i=1, 2, ..., k, w_m is an p-dimensional vector weights for the arcs connecting the input layer and the m-th hidden node, m=1, 2, ..., h, v_m is a C-dimensional vector of weights for the arcs connecting the m-th hidden node and the output layer. The activation function for the output layer is sigmoid function $\sigma(y) = 1/(1+e^{-y})$ and for the hidden layer is hyperbolic tangent function $\delta(y) = (e^y - e^{-y})/(e^y + e^{-y})$.

Step 4 Add one hidden node to hidden layer. Randomly initialize the weights of the arcs connecting this new hidden node with input nodes and output nodes. Set h = h+1 and go to step 2.

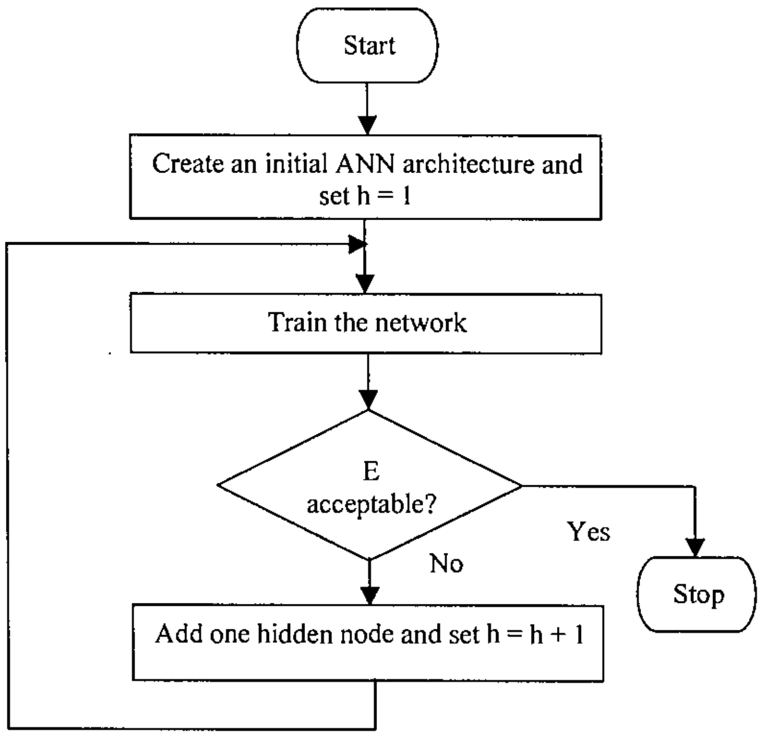

Start
Create an initial ANN architecture and set h = 1
Train the network
E acceptable?
Yes
No
Stop
Add one hidden node and set h = h + 1

Fig. 3.2 Flow chart of the constructive algorithm used in REANN.

Although other architecture determination algorithms, such as pruning and evolutionary algorithms could be used in REANN, the reasons for using constructive algorithm are fourfold. First, it is straightforward in constructive algorithms to specify an initial network, while it is problematic in pruning algorithms, one does not know in practice how big the initial network should be. Second, constructive algorithms always search for small network solutions first. They are thus computationally more efficient than pruning algorithms, in which the majority of the training time is spent on networks larger than necessary. Because of smaller solutions, the ANN is less likely to overfit the training data and, thus, more likely to generalize better. Third, the strong convergence of a constructive algorithm follows directly from its universal approximation ability. Fourth, a constructive approach usually requires a relatively small number of user specified parameters. The use of many user specified parameters requires a user to know rich prior knowledge, which often does not exist for complex real world problems.

3.2.2 Pruning Algorithm

Pruning offers an approach for dynamically determining an appropriate network topology. Pruning techniques begin by training a larger than necessary network and then eliminate weights and nodes that are deemed redundant [68-69].

As the nodes of the hidden layer are determined automatically by constructive algorithm in REANN, the aim of this pruning algorithm used here is to remove, as many unnecessary connections as possible. A node is pruned if all the connections to and from the node are pruned. Typically, methods for removing weights from the network involve adding a penalty term to the error function. It is hope that by adding a penalty term to the error function, unnecessary connections will have small weights, and therefore pruning can reduce the complexity of the network significantly. The simplest and most commonly used penalty term is the sum of the squared weights.

Given a set of input patterns $x_i \in \Re^n$, i = 1, 2, ..., k. Let w_m is an p-dimensional vector weights for the arcs connecting the input layer and the m-th hidden node. The weight of the connection from the l-th input node to the m-th hidden node is denoted by w_{ml}, v_m is a c-dimensional vector of weights for the arcs connecting the m-th hidden node and the output layer. The weight of the connection from the m-th hidden node to the p-th output node is denoted by $v_{pm.}$ It has been suggested that faster convergence can be achieved by minimizing the cross entropy function instead of squared error function [70]. The backpropagation algorithm is applied to update the weights (w, v) and minimize the following function:

$$\theta(w, v) = F(w, v) + P(w, v), \tag{3.3}$$

Where F(w, v) is the cross entropy function

$$F(w,v) = -\sum_{i=1}^{k}\sum_{p=1}^{o}\left(t_{pi}\log S_{pi} + (1-t_{pi})\log(1-S_{pi})\right), \tag{3.4}$$

S_{pi} is the output of the network

$$S_{pi} = \sigma(\sum_{m=1}^{h}\delta((x_i)^T w_{ml})v_{pml}) \tag{3.5}$$

Where $(x_i)^T w_m$ denotes the scalar product of the vectors x_i and w_m, $\delta(.)$ is the hyperbolic tangent function and $\sigma(.)$ is the logistic sigmoid function.

P (w, v) is a penalty term used for weight decay

$$P(w,v)=\varepsilon_1\left(\sum_{m=1}^{h}\sum_{l=1}^{n}\frac{\beta(w_{ml})^2}{1+\beta(w_{ml})^2}+\sum_{m=1}^{h}\sum_{p=1}^{o}\frac{\beta(v_{pm})^2}{1+\beta(v_{pm})^2}\right)+\varepsilon_2\left(\sum_{m=1}^{h}\sum_{l=1}^{n}(w_{ml})^2+\sum_{m=1}^{h}\sum_{p=1}^{o}(v_{pm})^2\right) \tag{3.6}$$

The values for the weight decay parameters ε_1, $\varepsilon_2 > 0$ must be chosen to reflect the relative importance of the accuracy of the network verses its complexity. More weights may be removed from the network at the cost of a decrease in the network accuracy with larger values of these two parameters. They also determine the range of values where the penalty for each weight in the network is approximately equal to ε_1. The parameter $\beta > 0$ determines the steepness of the error function near the origin.

The pruning algorithm used in REANN is briefly described below. This pruning algorithm removes the connections of the ANN according to the magnitudes of their weights. As the eventual goal of REANN is to get a set of simple rules that describe the classification process, it is important that all unnecessary connections and nodes must be removed. In order to remove as many connections as possible, the weights of the network must be prevented from taking values that are too large [71]. At the same time, weights of irrelevant connections should be encouraged to converge zero. The penalty function is found to be particularly suitable for these purposes.

The steps of the weight-pruning algorithm are summarized in Fig. 3.3, which are explained further as follows:

Step 1 Train the network to meet a prespecified accuracy level with the following condition (3.7) satisfied by all correctly classified input patterns.

$$\max_p |e_{pi}| = \max_p |S_{pi} - t_{pi}| \le \eta_1,\ p = 1,2,\ldots,C. \tag{3.7}$$

Let η_1 and η_2 be positive scalars such that $\eta_1 + \eta_2 < 0.5$ (η_1 is the error tolerance, η_2 is a threshold that determines if a weight can be removed), where $\eta_1 \in [0, 0.5)$. Let (w, v) be the weights of this network.

Step 2 Remove connections between input nodes and hidden nodes and between hidden nodes and output nodes. This task is accomplished in two phases. In the first phase, connections between input nodes and hidden nodes are removed. For each w_{ml} in the network,

$$\text{if} \qquad \max_p |v_{pm} w_{ml}| \le 4\,\eta_2, \tag{3.8}$$

then remove w_{ml} from the network. In the second phase, connections between hidden nodes and output nodes are removed. For each v_{pm} in the network,

$$\text{if } |v_{pm}| \le 4\,\eta_2, \tag{3.9}$$

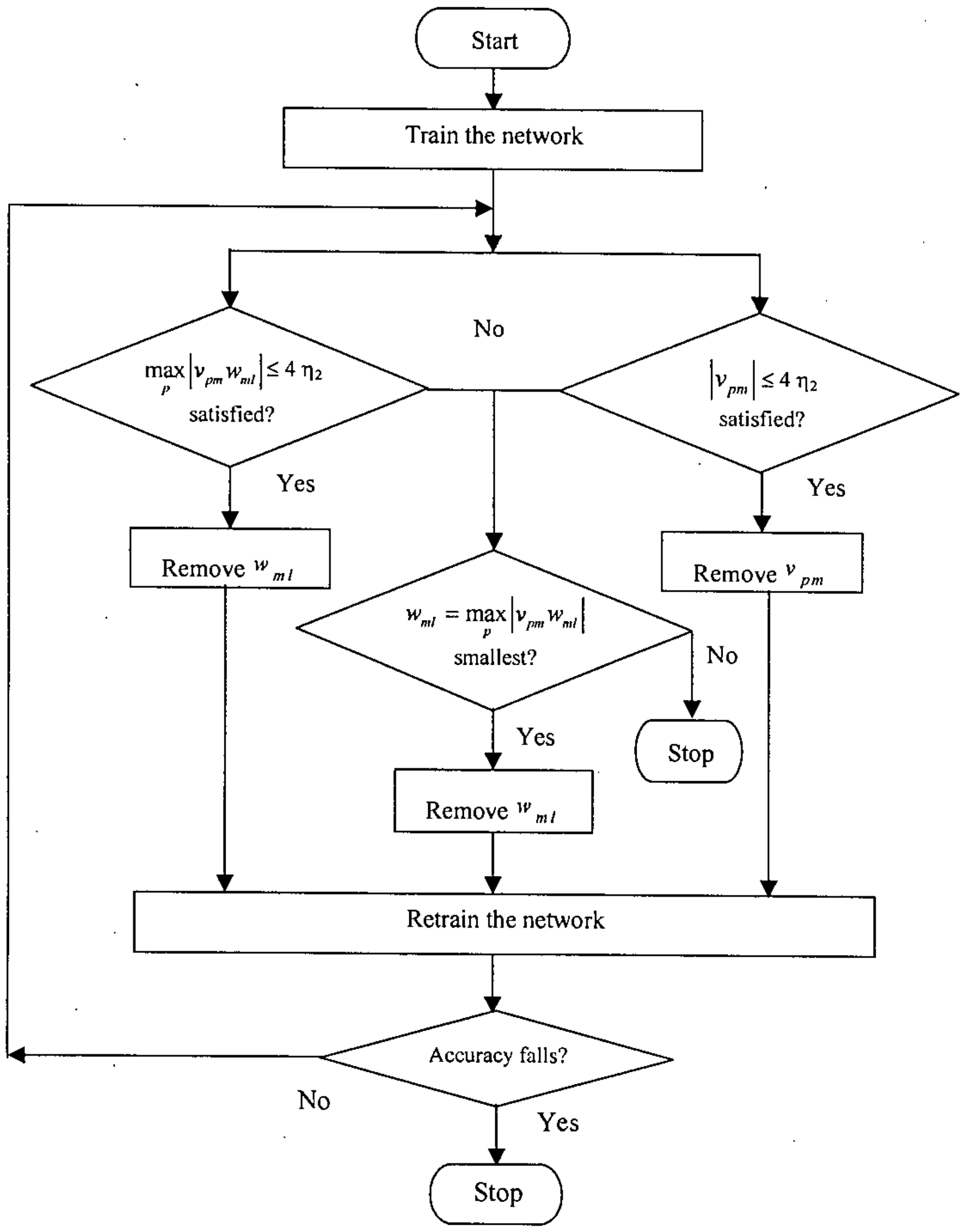
then remove v_{pm} from the network.

Step 3 Remove connections between input nodes and hidden nodes further. If no weight satisfies condition (3.8) or condition (3.9), then for each w_{ml} in the network, compute $w_{ml} = \max_p |v_{pm} w_{ml}|$. Remove w_{ml} with smallest w_{ml}. Continue, otherwise stop.

Step 4 Retrain the network and calculate the classification accuracy of the network.

Step 5 If classification rate of the network falls below an acceptable level, then stop and use the previous setting of network weights. Otherwise, go to Step 2.

The pruning algorithm used in REANN intended to reduce the amount of training time. Although it can no longer be guaranteed that the retrained pruned ANN will give the same accuracy rate as the original ANN, the experiments show that many weights can be eliminated simultaneously without deteriorating the performance of the ANN. The two conditions (3.8) and 3.9) for pruning depend on the weights for connections between input and hidden nodes and between hidden and output nodes. It



Start
Train the network
$\max_p |v_{pm} w_{ml}| \leq 4\,\eta_2$ satisfied?
No
$|v_{pm}| \leq 4\,\eta_2$ satisfied?
Yes
Yes
Remove w_{ml}
Remove v_{pm}
$w_{ml} = \max_p |v_{pm} w_{ml}|$ smallest?
No
Yes
Stop
Remove w_{ml}
Retrain the network
Accuracy falls?
No
Yes
Stop

Fig. 3.3 Flowchart of the pruning algorithm.

is imperative that during the training these weights be prevented from getting too large. At the same time, small weights should be encouraged to decay rapidly to zero.

3.2.3 Heuristic Clustering Algorithm

The process of grouping a set of physical or abstract objects into classes of similar objects is called clustering. A cluster is a collection of data objects that are similar within the same cluster and are dissimilar to the object in other clusters. A cluster of a data objects can be treated collectively as one group in many applications [72]. There exist a large number of clustering algorithms in the literature such as k-means, k-medoids [73-74]. The choice of clustering algorithm depends both on the type of data available and on the particular purpose and application.

After applying pruning algorithm in REANN, the ANN architecture produced by constructive algorithm contains only important connections and nodes. Nevertheless, rules are not readily extractable because the hidden node activation values are continuous. The discretization of these values paves the way for rule extraction.

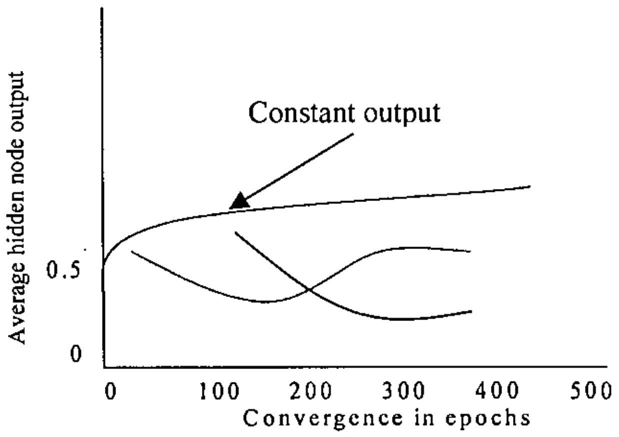


Fig. 3.4 Output of hidden nodes.

It is found that some hidden nodes of an ANN maintain almost constant output while other nodes change continuously during the whole training process [75]. Fig. 3.4 shows a hidden node maintains almost constant output after some training epochs. In REANN, no clustering algorithm is used when hidden nodes maintain almost constant output. If the outputs of hidden nodes do not maintain constant value, a heuristic clustering algorithm is used.

The aim of the clustering algorithm is to discretize the output values of hidden nodes. The algorithm places candidates for discrete values such that the distance between them is at least a threshold value ε. A very small ε will always guarantee that the network with discrete activation values will have the same accuracy as the original network with continuous activation values. The algorithm can then be run again with a larger value of ε to reduce the number of clusters.

The steps of the heuristic clustering algorithm are summarized in Fig. 3.5, which are explained further as follows:

Step 1 Let $\varepsilon \in (0, 1)$. D is the activation values in the hidden node. δ_1 is the activation value for the first pattern. The first cluster, $H(1) = \delta_1$, count = 1, and sum(1) = δ_1, set D = 1.

Step 2 For each pattern p_i $i = 1, 2, 3, \ldots k$. Checks whether subsequent activation values can be clustered into one of the existing clusters. The distance between an activation value under consideration and its nearest cluster, $|\delta - H(\bar{j})|$, is computed. If this distance is less than ε, then the activation value is clustered in cluster $\bar{j}$. Otherwise, this activation value forms a new cluster. Let δ be its activation value. If there exists an index $\bar{j}$ such that

$$|\delta - H(\bar{j})| = \min_{j \varepsilon \{1,2,\ldots\ldots D\}} |\delta - H(\bar{j})| \quad \text{and} \quad |\delta - H(\bar{j})| \leq \varepsilon$$

then set count($\bar{j}$):=count($\bar{j}$)+1, sum($\bar{j}$):=sum($\bar{j}$)+ δ else D = D+1

H(D) = δ, count(D) = 1, sum (D) = δ.

Step 3 Replace H by the average of all activation values that have been clustered into this cluster: H(j):=sum(j)/count(j), j=1, 2, 3,.....D.

Step 4 Once the activation values of all hidden nodes have been obtained, the accuracy of the network is checked with the activation values at the hidden nodes replaced by their discretized values. An activation value δ is replaced by $H(\bar{j})$, where index $\bar{j}$ is chosen such that $\bar{j} = \arg\min_j |\delta - H(j)|$. If the accuracy of the network falls below the required accuracy, then ε must be decreased and the clustering algorithm is run again, otherwise stop.

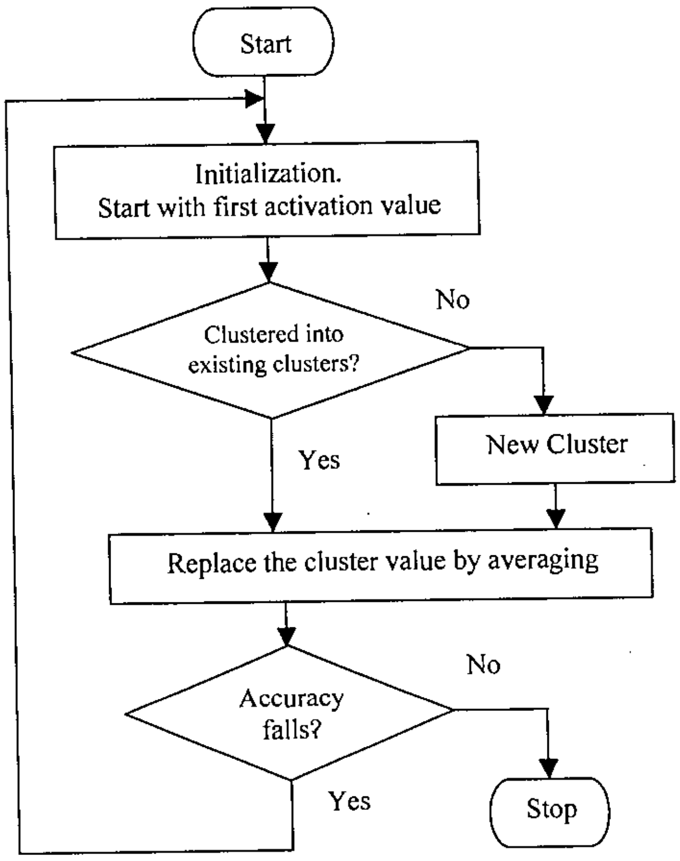

Start
Initialization.
Start with first activation value
Clustered into existing clusters?
No
New Cluster
Yes
Replace the cluster value by averaging
Accuracy falls?
No
Yes
Stop

Fig. 3.5 Flow chart of the heuristic clustering algorithm.

For a sufficiently small ε, it is always possible to maintain the accuracy of the network with continuous activation values, although the resulting number of different discrete activations can be impractically large.

The best ε value is one that gives a high accuracy rate after the clustering and at the same time generates as few clusters as possible. A simple way of obtaining an optimal value for ε is by searching in the interval (0, 1). The number of clusters and the accuracy of the network can be checked for all values of $\varepsilon = i\zeta$, $i = 1, 2, \ldots$, where ζ is a small positive scalar, e.g. 0.10. Note also that it is not necessary to fix the value of ε equal for all hidden nodes.

3.2.4 Rule Extraction Algorithm (REx)

Classification rules are sought in many areas from automatic knowledge acquisition [76-77] to data mining [78-79] and ANN rule extraction. This is because classification rules possess some attractive features. They are explicit, understandable and verifiable by domain experts, and can be modified, extended and passed on as modular knowledge. The REx is composed of three major functions:

i) Rule Extraction: this function iteratively generates shortest rules and remove/marks the patterns covered by each rule until all patterns are covered by the rules.

ii) Rule Clustering: rules are clustered in terms of their class levels and

iii) Rule Pruning: redundant or more specific rules in each cluster are removed.

A default rule should be chosen to accommodate possible unclassifiable patterns. If rules are clustered, the choice of the default rule is based on clusters of rules.

The steps of the Rule Extraction (REx) algorithm are summarized in Fig. 3.6, which are explained further as follows:

Step 1 Extract Rule:

```
i=0; while (data is NOT empty/marked){
generate Ri to cover the current pattern and differentiate it from patterns in other categories;
remove/mark all patterns covered by Ri ; i++}
```

The core of this step is a greedy algorithm that finds the shortest rule based on the first order information that can differentiate the pattern under consideration from the patterns of other classes. It then iteratively generates rules and removes the patterns covered by the rules.

Step 2 Cluster Rule:

Cluster rules according to their class levels. Rules generated in Step 1 are grouped in terms of their class levels. In each rule cluster, redundant rules are eliminated; specific rules are replaced by more general rules.

Step 3 Prune Rule:

replace specific rules with more general ones;

remove noise rules;

eliminate redundant rules;

Step 4 Check whether all patterns are covered by any rules. If yes then stop, otherwise continue.

Step 5 Determine a default rule:

A default rule is chosen when no rule can be applied to a pattern.

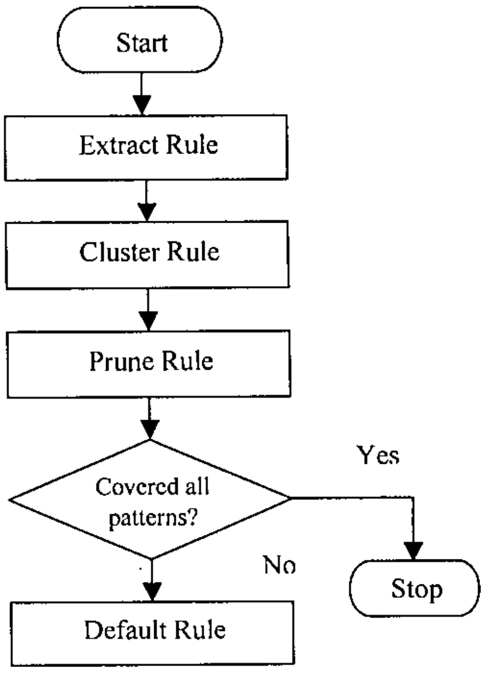


Fig. 3.6 Flow chart of the rule extraction (REx) algorithm.

REx exploits the first order information in the data and finds shortest sufficient conditions for a rule of a class that can differentiate it from patterns of other classes. It can generate concise and perfect rules in the sense that the error rate of the rules is not worse than the inconsistency rate found in the original data. The novelty of REx is that the rule generated by it is order insensitive, i.e, the rules need not be required to fire sequentially.

Chapter 4

Experimental Evaluation

4.1 Introduction

This chapter evaluates the performance of REANN on several well-known benchmark classification problems. These are the breast cancer, iris, diabetes, wine, season, golf playing, and lenses classification problems. They are widely used in machine learning and ANN research. The data sets representing all the problems were real world data and obtained from the UCI machine learning benchmark repository. Experimental details, results, comparisons with other works and discussion are described in this chapter.

4.2 Data Set Description

The following subsections briefly describe the data set used in this study. The characteristics of the data sets are summarized in Table 4.1. The detailed descriptions of the data sets are available at ics.uci.edu (128.195.11) in directory /pub/machine-learning-databases [80-81].

4.2.1 The Breast Cancer Data

The purpose of this problem was to diagnose a breast tumor as either benign or malignant based on cell descriptions gathered by microscopic examination. Input attributes were for instance the clump thickness, the uniformity of cell size and cell shape, the amount of marginal adhesion, and the frequency of bare nuclei.

The data set representing this problem contained 699 examples. Each example consisted of nine-element real valued vectors. This was a two-class problem. All inputs are continuous; 65.5% of the examples arc 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 Iris Data

This is perhaps the best-known database to be found in the pattern recognition literature. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are not linearly separable from each other.

Number of Instances: 150 (50 in each of three classes). Number of Attributes: 4 numeric (sepal length, sepal width, petal length and petal width). This was a three-class problem: Iris Setosa, Iris Versicolour and Iris Virginica. Class Distribution: 33.3% for each of 3 classes.

4.2.3 The Diabetes Data

The objective of this data set was diagnosis of diabetes of Pima Indians. Based on personal data, such as 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 was diabetes positive or not.

There were 768 examples in the data set, each of which consisted of eight-element real valued vectors. This was a two-class problem. All inputs are continuous. 65.1% of the examples are diabetes negative; entropy 0.93 bits per example. This dataset was created based on the "Pima Indians diabetes" problem dataset from the UCI repository of machine learning databases.

Table 4.1 Characteristics of data sets.

Data Sets	**No. of Examples**	**Input Attributes**	**Output Classes**
Breast Cancer	699	9	2
Iris	150	4	3
Diabetes	768	8	2
Wine	178	13	3
Season	11	3	4
Golf Playing	14	4	2
Lenses	24	4	3

4.2.4 The Wine Data

In a classification context, this is a well-posed problem with "well behaved" class structures. A good data set for first testing of a new classifier, but not very challenging. These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines. Number of instances 178, number of attributes 13. All attributes are continuous. This was a two-class problem.

4.2.5 The Season Data

The season data set contains discrete data only. There were 11 examples in the data set, each of which consisted of three-elements. These are weather, tree and temperature. This was a four-class problem.

4.2.6 The Golf Playing Data

The Golf playing data set contains both numeric and discrete data. There were 14 examples in the data set, each of which consisted of four-elements. These are outlook, temperature, humidity and wind. This was a two-class problem.

4.2.7 The Lenses Data

The data set contains 24 examples and are complete and noise free. The examples highly simplified the problem. The attributes do not fully describe all the factors affecting the decision as to which type, if any, to fit.

Number of Instances: 24. Number of Attributes: 4; age, spectacle prescription, astigmatic and tear production rate. All attributes are nominal. This was three-class problem: hard contact lenses, soft contact lenses and not contact lenses.

4.3 Experimental Setup

In all experiments, one bias node with a fixed input 1 was used for hidden and output layers. The learning rate was set between [0.1, 1.0] and the weights were initialized to random values between [-1.0, 1.0]. The number of training epochs τ was chosen between 5 and 20. Value of ε for clustering was set between [0.1, 1.0]. Values of weight decay parameters ε_1, ε_2, were set between [0.05, .5] and $[10^{-4}, 10^{-8}]$ and β was 10 for penalty function. Hyperbolic tangent function $\delta(y) = \frac{e^y - e^{-y}}{e^y + e^{-y}}$ is used as hidden node activation function and logistic sigmoid function $\sigma(y) = \frac{1}{1+e^{-y}}$ as output node activation function.

In this study, all data sets representing the problems are divided into two sets. One is the training set and the other is the testing set. Note that no validation set is used in this study. The numbers of examples in the training set and testing set are based on numbers in other works, in order to make comparison with those works possible. The sizes of the training and testing data sets used in this study are given as follows:

- **Breast cancer data set:** the first 350 examples are used for the training set and the rest 349 for the testing set.
- **Iris data set:** the first 75 examples are used for the training set and the rest 75 for the testing set.
- **Diabetes data set:** the first 384 examples are used for the training set and the rest 384 for the testing set.

- **Wine data set:** the first 89 examples are used for the training set and the rest 89 for the testing set.

4.4 Experimental Results

Tables 4.2-4.8 show ANN architectures produced by REANN and training epochs over 10 independent runs on seven benchmark classification problems. The initial architecture was selected before applying the constructive algorithm, which was used to determine the number of nodes in the hidden layer. The intermediate architecture was the outcome of the constructive algorithm, and the final architecture was the outcome of pruning algorithm used in REANN.

It is seen that REANN can automatically determine compact ANN architectures. For example, for the breast cancer data, REANN produces more compact architecture. The average number of nodes and connections were 6.8 and 5.8 respectively; in most of the 10 runs 5 to 6 input nodes were pruned. For the diabetes data one hidden node was pruned in some iterations, as all the connections to and from this node were pruned by pruning algorithm. The average number of nodes and connections were 12.5 and 19.4 respectively.

Table 4.2 ANN architectures and training epochs for **breast cancer** data. The results were averaged over 10 independent runs.

	Initial Architecture		Intermediate Architecture		Final Architecture		No. of Epoch
	No. of Node	No. of Connection	No. of Node	No. of Connection	No. of Node	No. of Connection	
Mean	12 (9-1-2)	11	12.7	18.1	6.8	5.8	233.2
Min	12 (9-1-2)	11	12	11	5	5	222
Max	12 (9-1-2)	11	14	33	10	9	245

Table 4.3 ANN architectures and training epochs for **iris** data. The results were averaged over 10 independent runs.

	Initial Architecture		Intermediate Architecture		Final Architecture		No. of Epoch
	No. of Node	No. of Connection	No. of Node	No. of Connection	No. of Node	No. of Connection	
Mean	8 (4-1-3)	7	9	14	8.8	10.2	196.7
Min	8 (4-1-3)	7	8	7	8	7	183
Max	8 (4-1-3)	7	10	21	10	14	217

Table 4.4 ANN architectures and training epochs for **diabetes** data. The results were averaged over 10 independent runs.

	Initial Architecture		Intermediate Architecture		Final Architecture		No. of Epoch
	No. of Node	No. of Connection	No. of Node	No. of Connection	No. of Node	No. of Connection	
Mean	11 (8-1-2)	10	13.2	30	12.5	19.4	302.6
Min	11 (8-1-2)	10	12	20	12	14	279
Max	11 (8-1-2)	10	14	40	13	24	326

Table 4.5 ANN architectures and training epochs for **wine** data. The results were averaged over 10 independent runs.

	Initial Architecture		Intermediate Architecture		Final Architecture		No. of Epoch
	No. of Node	No. of Connection	No. of Node	No. of Connection	No. of Node	No. of Connection	
Mean	17 (13-1-3)	16	18.5	40	18	26.5	213
Min	17 (13-1-3)	16	17	16	17	20	193
Max	17 (13-1-3)	16	20	64	19	43	237

Table 4.6 ANN architectures and training epochs for **season** data. The results were averaged over 10 independent runs.

	Initial Architecture		Intermediate Architecture		Final Architecture		No. of Epoch
	No. of Node	No. of Connection	No. of Node	No. of Connection	No. of Node	No. of Connection	
Mean	8 (3-1-4)	7	8.9	13.3	8.7	11.2	88.2
Min	8 (3-1-4)	7	8	7	8	9	73
Max	8 (3-1-4)	7	10	14	10	16	101

Table 4.7 ANN architectures and training epochs for **golf playing** data. The results were averaged over 10 independent runs.

	Initial Architecture		Intermediate Architecture		Final Architecture		No. of Epoch
	No. of Node	No. of Connection	No. of Node	No. of Connection	No. of Node	No. of Connection	
Mean	7 (4-1-2)	6	8.2	13.2	7.9	10.5	94.5
Min	7 (4-1-2)	6	7	6	7	6	86
Max	7 (4-1-2)	6	9	18	9	14	103

Table 4.8 ANN architectures and training epochs for **lenses** data. The results were averaged over 10 independent runs.

	Initial Architecture		Intermediate Architecture		Final Architecture		No. of Epoch
	No. of Node	No. of Connection	No. of Node	No. of Connection	No. of Node	No. of Connection	
Mean	8 (4-1-3)	7	9.1	14.7	8.9	12.1	109.2
Min	8 (4-1-3)	7	8	7	8	7	97
Max	8 (4-1-3)	7	10	21	10	17	128

Figs. 4.1-4.2 show the smallest of the pruned networks over 10 runs for breast cancer and diabetes problem. The pruned network for breast cancer problem has only 1 hidden node and 5 connections. The accuracy of this network on the training data and testing data were 96.275% and 93.429% respectively. In this example only three input attributes A_1, A_6 and A_9 were important and only three discrete values of hidden node activation's were needed to maintain the accuracy of the network. The discrete values found by the heuristic clustering algorithm were 0.987, -0.986 and 0.004. Of the 350 training data, 238 patters have the first value, 106 have the second value and rest 6 patterns have third value. The weight of the connection from the hidden node to the first output node was 3.0354 and to the second output node was –3.0354.

The pruned network for diabetes problem has only 2 hidden nodes. No input nodes were pruned by pruning algorithm. One hidden node was pruned, as all the connections to and from this node were pruned. The accuracy on the training data and testing data were 76.30% and 75.52% respectively. The weight of the connection from the first hidden node to the first output node was -1.153 and to the second output node was 1.153 and the weight of the connection from the second hidden node to the first output node was -32.078 and to the second output node was 32.084.

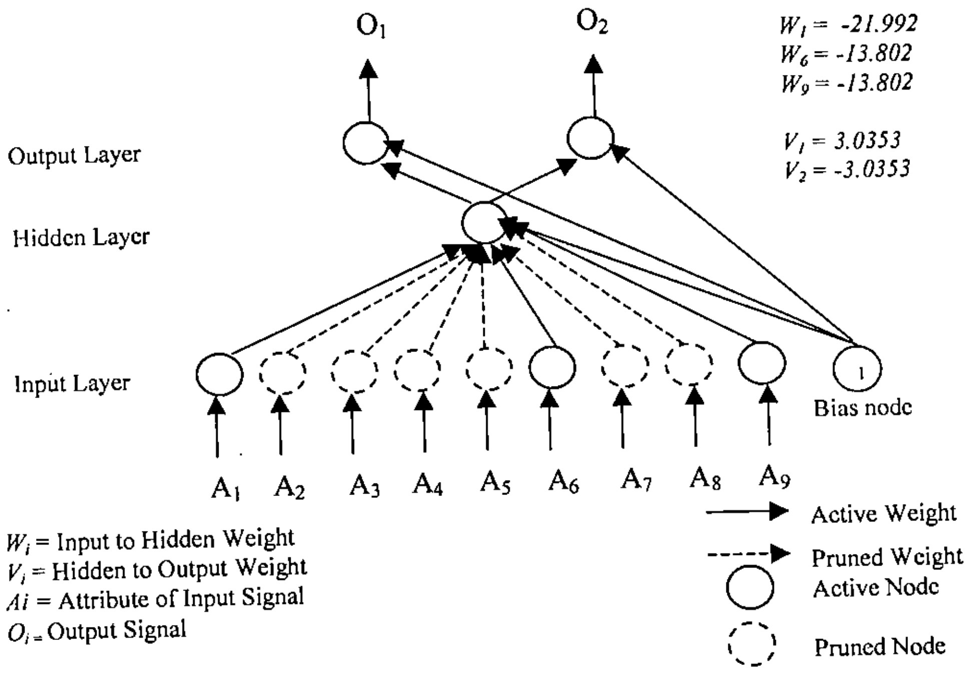


Fig. 4.1 A pruned network for breast cancer diagnosis problem. The accuracy on training and testing data sets were 96.275% and 93.429% respectively.

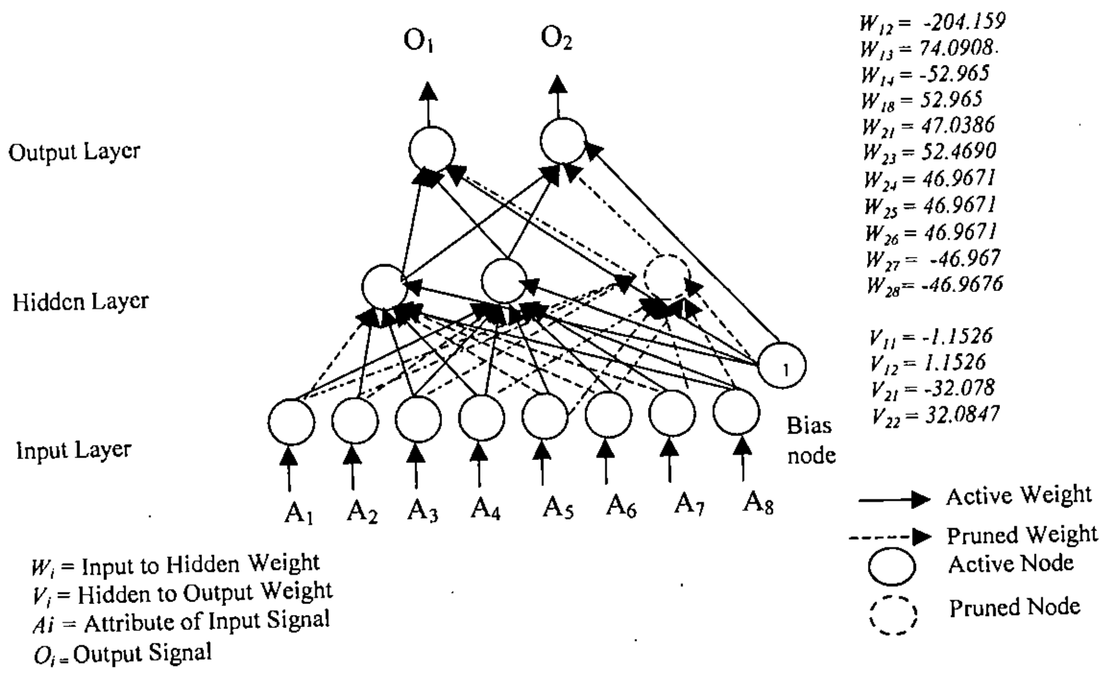


Fig. 4.2 A pruned network for diabetes problem.

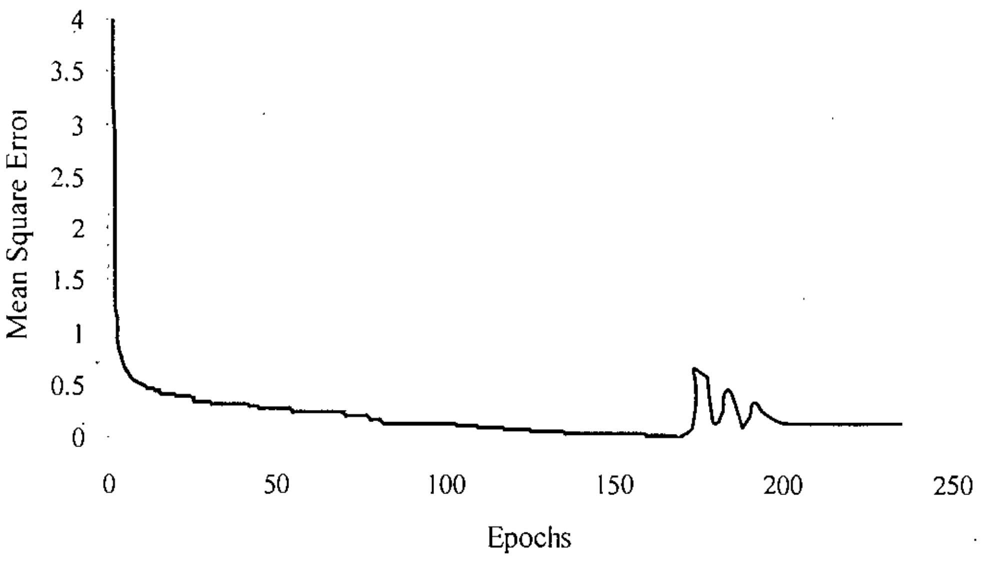


Fig. 4.3 Training time error for breast cancer data.

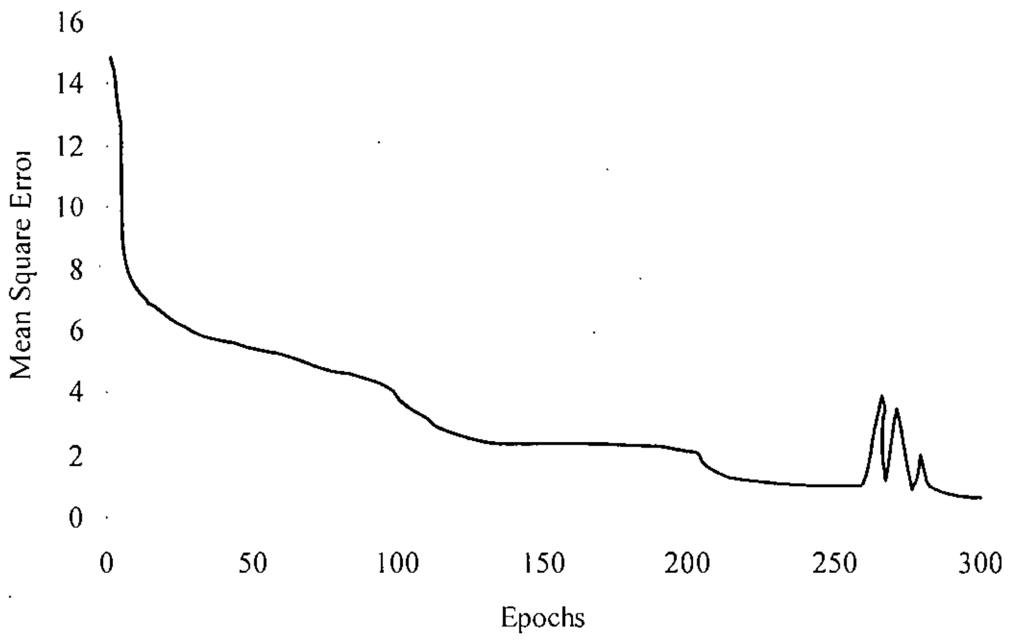


Fig. 4.4 Training time error for diabetes data.

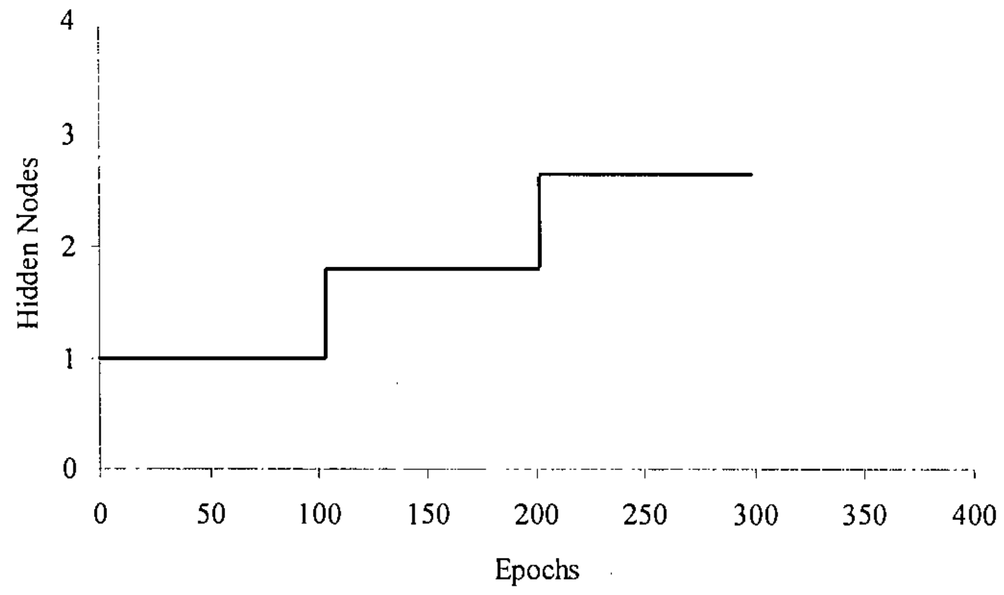


Fig. 4.5 Hidden node addition for the diabetes data.

Figs. 4.3-4.4 show the training time error for breast cancer and diabetes problem. For breast cancer problem, it was observed that the training error decreased and maintained almost constant for a long time after some training epochs and then fluctuates. The fluctuation was made due to the pruning process. As the network was retrained after completing the pruning process thus the training error again maintained almost constant value. For diabetes problem, it was observed that the training error decreased and maintained almost constant after some training epochs, it was further decreased when additional hidden nodes were added. The fluctuation was observed due to the connection pruning and finally maintained almost constant value in account of retraining the pruned network. Fig. 4.5 shows the effects of hidden node addition with increasing the training epochs for diabetes problems. It is seen that the number of hidden node was one at the beginning of the training process. As the training process progressed, the number of hidden nodes increases gradually. Finally the number of hidden nodes is 3 at 300 epochs.

Table 4.9 Number of extracted rules and rules accuracy for seven benchmarks problems.

Data Sets	**No. of Extracted Rules**	**Rules Accuracy on Training Set**	**Rules Accuracy on Testing Set**
Breast Cancer	2	96.28 %	93.43 %
Iris	3	98.67 %	97.33 %
Diabetes	2	76.56 %	72.14 %
Wine	3	91.01 %	83.15 %
Season	4	100 %	100 %
Golf Playing	3	100 %	100 %
Lenses	8	100 %	100 %

Table 4.9 shows number of extracted rules and rules accuracy for seven benchmark problems. In most of the cases REANN produces fewer rules with better accuracy. It was observed that two to three rules were sufficient to solve the problems. The accuracy was 100% for three data sets include season, golf playing, and lenses classification. These data sets having lower number of examples. The accuracy for other data sets was also encouraging and better compared to other works.

4.4.1 Extracted Rules

The number of rules extracted by REANN and the accuracy of the rules in training and testing data sets were described in Table 4.9. But the visualization of the rules in terms of the original attributes ware not discussed. The following subsections discussed the rules extracted by REANN in terms of the original attributes for breast cancer, iris, diabetes, wine, season, golf playing, and lenses classification problems. The number of conditions per rule and the number of rules extracted were also visualized here.

4.4.1.1 Breast Cancer Data

Rule 1: If Clump thickness (A_1) <= 0.6 and Bare nuclei (A_6) <= 0.5 and Mitosis(A_9) <= 0.3, then benign

Default Rule: malignant.

4.4.1.2 Iris Data

Rule 1: If Petal-length (A_3) <= 1.9 then Iris setosa

Rule 2: If Petal-length (A_3) <= 4.9 and Petal-width (A_4) <= 1.6 then Iris versicolor

Default Rule: Iris virginica.

4.4.1.3 Diabetes Data

Rule 1: If Plasma glucose concentration (A_2) <= 0.64 and Age (A_8) <= 0.69 then tested negative

Default Rule: tested positive.

4.4.1.4 Wine Data

Rule 1: If Input 10 (A_{10}) <= 3.8 then class 2

Rule 2: If Input 13 (A_{13}) >= 845 then class 1

Default Rule: class 3.

4.4.1.5 Season Data

Rule 1: If Tree (A_2) = yellow then autumn

Rule 2: If Tree (A_2) = leafless then autumn

Rule 3: If Temperature (A_3) = low then winter

Rule 3: If Temperature (A_3) = high then summer

Default Rule: spring.

200852

١٨

4.4.1.6 Golf Playing Data

Rule 1: If Outlook (A_1) = sunny and Humidity>=85 then don't play

Rule 2: Outlook (A_1) = rainy and Wind= strong then don't play

Default Rule: play.

4.4.1.7 Lenses Data

Rule 1: If Tear Production Rate (A_4) = reduce then no contact lenses

Rule 2: If Age (A_1) = presbyopic and Spectacle Prescription (A_2) = hypermetrope and Astigmatic (A_3) = yes then no contact lenses

Rule 3: If Age (A_1) = presbyopic and Spectacle Prescription (A_2) = myope and Astigmatic (A_3) = no then no contact lenses

Rule 4: If Age (A_1) = pre-presbyopic and Spectacle Prescription (A_2) = hypermetrope and Astigmatic (A_3) = yes and Tear Production Rate (A_4) = normal then no contact lenses

Rule 5: If Spectacle Prescription (A_2) = myope and Astigmatic (A_3) = yes and Tear Production Rate (A_4) = normal then hard contact lenses

Rule 6: If Age (A_1) = pre-presbyopic and Spectacle Prescription (A_2) = myope and Astigmatic (A_3) = yes and Tear Production Rate (A_4) = normal then hard contact lenses

Rule 7: If Age (A_1) = young and Spectacle Prescription (A_2) = myope and Astigmatic (A_3) = yes and Tear Production Rate (A_4) = normal then hard contact lenses

Default Rule: soft contact lenses.

Table 4.10 Performance comparison of REANN with other algorithms for **breast cancer** data.

Data Set	Feature	REANN	NN RULES	DT RULES	C4.5	NN-C4.5	OC1	CART
Breast Cancer	No. of Rules	2	4	7	-	-	-	-
	Avg. No. of Conditions	3	3	1.75	-	-	-	-
	Accuracy %	96.28	96	95.5	95.3	96.1	94.99	94.71

Table 4.11 Performance comparison of REANN with other algorithms for **iris** data.

Data Set	Feature	REANN	NN RULES	DT RULES	BIO RE	Partial RE	Full RE
Iris	No. of Rules	3	3	4	4	6	3
	Avg. No. of Conditions	1	1	1	3	3	2
	Accuracy %	98.67	97.33	94.67	78.67	78.67	97.33

Table 4.12 Performance comparison of REANN with other algorithms for **diabetes** data.

Data Set	Feature	REANN	NN RULES	C4.5	NN-C4.5	OC1	CART
Diabetes	No. of Rules	2	4				
	Avg. No. of Conditions	2	3				
	Accuracy %	76.56	76.32	70.9	76.4	72.4	72.4

Table 4.13 Performance comparison of REANN with other algorithms for **season** data.

Data set	Feature	REANN	RULES	X2R
Season	No. of Rules	5	7	6
	Avg. No. of Conditions	1	2	1
	Accuracy %	100.0	100.0	100.0

Table 4.14 Performance comparison of REANN with other algorithms for **golf playing** data.

Data set	Feature	REANN	RULES	RULES-2	X2R
Golf Playing	No. of Rules	3	8	14	3
	Avg. No. of Conditions	2	2	2	2
	Accuracy %	100.0	100.0	100.0	100.0

Table 4.15 Performance of REANN for **wine** data.

Data set	Feature	REANN
Wine	No. of Rules	3
	Avg. No. of Conditions	3
	Accuracy %	91.01

Table 4.16 Performance comparison of REANN with other algorithm for **lenses** data.

Data set	Feature	REANN	PRISM
Lenses	No. of Rules	8	9
	Avg. No. of Conditions	3	-
	Accuracy %	100.0	100.0

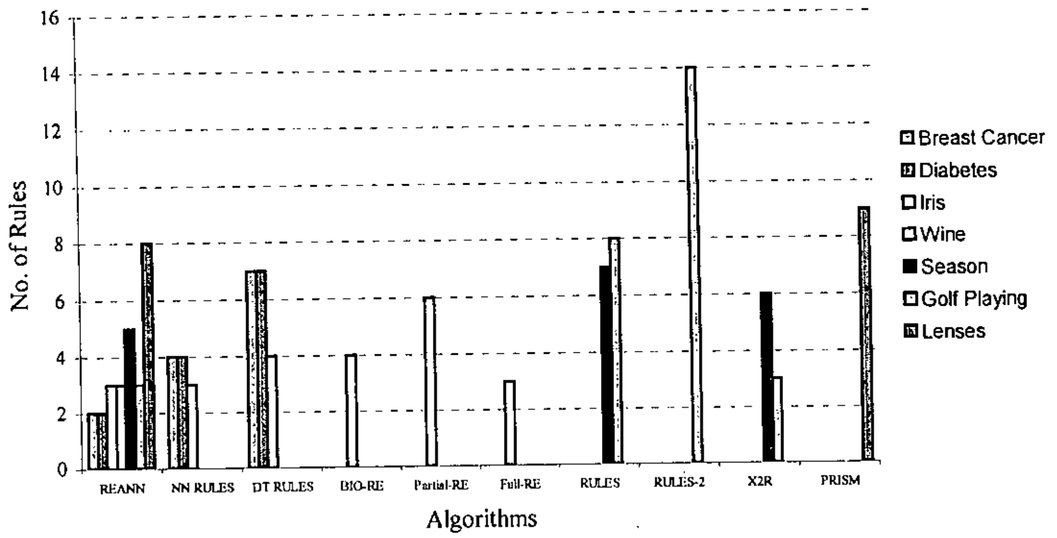


Fig. 4.6 Comparison of number of rules for various algorithms.

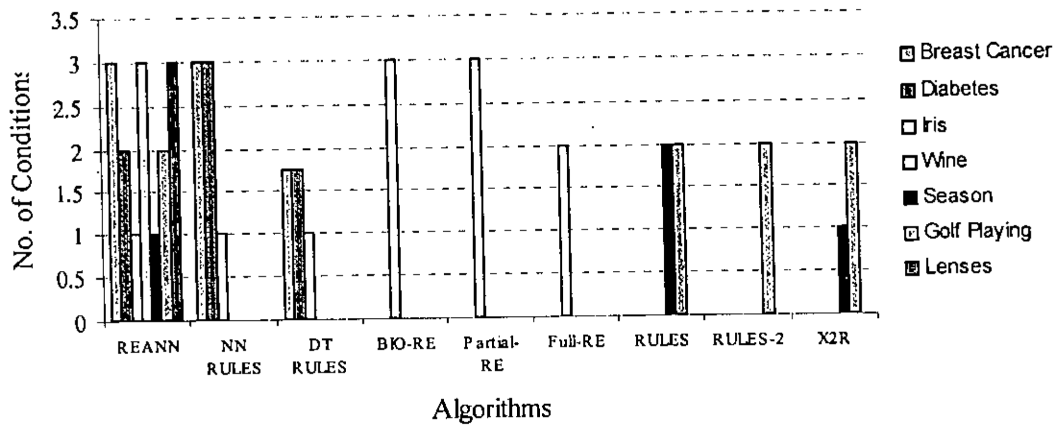


Fig. 4.7 Comparison of number of conditions per rule for various algorithms.

4.5 Comparison

This section compares experimental results of REANN with the results of other works. The primary aim of this work is not to exhaustively compare REANN with all other works, but to evaluate REANN in order to gain a deeper understanding of rule extraction.

Table 4.10 compares REANN results of breast cancer problem with those produced by NN RULES [11], DT RULES [11], C4.5 [76], NN-C4.5 [82], OC1 [82], and CART [83] algorithms. REANN achieved best performance although NN RULES was closest second. But number of rules extracted by REANN are 2 whereas these were 4 for NN RULES.

Table 4.11 compares REANN results of iris data with those produced by NN RULES, DT RULES, BIO RE [12], Partial RE [12], and Full RE [12] algorithms. REANN achieved 98.67% accuracy although NN RULES was closest second with 97.33% accuracy. Here number of rules extracted by REANN and NN RULES are equal.

Table 4.12 compares REANN results of diabetes data with those produced by NN RULES, C4.5, NN-C4.5, OC1, and CART algorithms. REANN achieved 76.56% accuracy although NN-C4.5 was closest second with 76.4% accuracy. Due to the high noise level, the diabetes problem is one of the most challenging problems in our experiments. REANN has outperformed all other algorithms.

Table 4.13 compares REANN results of season data with those produced by RULES [84] and X2R [7]. All three algorithms achieved 100% accuracy. This is possible because the number of examples is low. Number of extracted rules by REANN are 5 whereas these were 7 for RULES and 6 for X2R.

Table 4.14 compares REANN results of golf playing data with those produced by RULES, RULES-2 [85], and X2R. All four algorithms achieved 100% accuracy because the lower number of examples. Number of extracted rules by REANN are 3 whereas these were 8 for RULES and 14 for RULES-2.

Table 4.15 shows REANN results of wine data. REANN achieved 91.01% accuracy on wine data by extracting 3 rules. No detailed previous work found for showing comparison of this data set.

Table 4.16 compares REANN results of lenses data with those produced by PRISM [86]. Both algorithms achieved 100% accuracy because the lower number of examples. Number of extracted rules by REANN are 8 whereas these were 9 for PRISM.

Fig. 4.6 shows the comparison of number of rules graphically for various algorithms. It was found that number of rules extracted by REANN is lower in most of the cases for seven benchmark classification problems compared to other works. Fig. 4.7 shows the comparison of number of conditions per rule graphically for various algorithms. It was found again that number of conditions per rule is encouraging. REANN and NN RULES emphasis the use of parallel features; while DT RULES focus on individual feature, that's why number of conditions of rules extracted by REANN and NN RULES are greater than DT RULES.

4.6 Discussion

This thesis has shown how rules can be extracted from a trained ANN without making any assumptions about the network's activations or having initial knowledge about the problem domain. If some knowledge is available, however, it can always be incorporated into the network. For example, connections in the network from inputs thought to be not relevant can be given large penalty parameters during training, while those thought to be relevant can be given zero or small penalty parameters [87]. The REANN algorithm does not require threshold activation function to force the activation values to be zero or one [3-4], nor does it require the weights of the connections to be restricted in a certain range [88]. Network training and pruning is done via the simple and widely used backpropagation method [40].

REANN was capable of finding a satisfactory ANN architecture through the constructive process. It could dynamically add hidden nodes to ANN during training, depending on the performance of the training data. For example, the initial number of hidden node was one, which was small. REANN first added one node to ANN, when adding hidden nodes had not achieved the minimum mean square error, REANN started adding new nodes to the ANN.

It is known that constructive algorithms in some cases might produce larger sized networks than necessary [89], and that pruning algorithms are computationally expensive [90]. Network size and computational expense affect classification accuracy and training time, respectively. Thus, the synergy between constructive and pruning algorithms is suitable for producing simplified ANN architectures at a reasonable computational expense. The use of pruning algorithms in conjunction with constructive algorithms reduces the ANN size in terms of hidden nodes and/or connections. For example in breast cancer problem the network architecture determined by constructive algorithm was 9-1-2, after applying the pruning algorithm only three connections from input to hidden node were remaining. Hence 6 input nodes were pruned and the resulting simplified network contains 5 connections only.

The penalty function used in REANN consists of two components. The first component is to discourage the use of unnecessary connections and the second component is to prevent the weights of these connections from taking excessively large values. Simple criteria for eliminating connections are also given. The two components of penalty function have been used individually in the past. However, applying the approach that combines the penalty function and the magnitude based weight elimination criteria, pruning algorithm used in REANN is able to get smaller networks than those reported in the literature.

An efficient heuristic clustering algorithm is used in REANN for discretizing the continuous activation values of the hidden nodes. In this clustering algorithm, the first activation value forms the first cluster. The next step is to checks whether subsequent activation values can be clustered into one of the existing clusters. The distance between an activation value under consideration and its nearest cluster is computed. If this distance is less than ε, then the activation value is clustered in this cluster. Otherwise, this activation value forms a new cluster.

Once the discrete values of all hidden nodes have been obtained, the accuracy of the network is checked again with the activation values at the hidden nodes replaced by their discretized values. If the accuracy of the network falls below the required accuracy, then ε must be decreased and the clustering algorithm is run again. For a sufficiently small ε, it is always possible to maintain the accuracy of the network with continuous activation values, although the resulting number of different discrete activations can be impractically large.

The best ε value is one that gives a high accuracy rate after the clustering and at the same time generates as few clusters as possible. A simple way of obtaining an optimal value for ε is by searching in the interval (0, 1). The number of clusters and the accuracy of the network can be checked for all values of $\varepsilon = i\gamma$, $i = 1, 2, \ldots.$, where γ is a small positive scalar, e.g. 0.10.

This thesis also introduces a basic rule extraction (REx) algorithm. REx is composed of three major functions: rule extraction, rule clustering, and rule pruning. The rule extraction function iteratively generates shortest rules and remove/marks the patterns covered by each rule until all patterns are covered by the rules. On the other hand the rule clustering clustered the rules in terms of their class levels, and finally, redundant or more specific rules in each cluster are removed by rule pruning function. A default rule should be chosen to accommodate possible unclassifiable patterns. REx exploits the first order information in the data and finds shortest sufficient conditions for a rule of a class that can differentiate it from patterns of other classes.

REx can generate concise rules from raw data sets. It only calculates first order information in generating rules. It can generate perfect rules in the sense that the error rate of the rules is not worse than the inconsistency rate found in the original data. The novelty of REx is that the rule generated by it is order insensitive, i.e, the rules need not be required to fire sequentially.

Chapter 5

Conclusion

5.1 Introduction

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 a significant limitation; however, their learned hypotheses are usually incomprehensible. To address this limitation, a number of research groups have developed techniques for rule extraction. Rule extraction involves approximating the function represented by a trained network, such as symbolic inference rules, that facilitates better comprehensibility. The focus of this thesis has been the development of a rule-extraction algorithm, called REANN, which overcomes the significant limitations of previous algorithms.

In this concluding chapter, the contributions and limitations of the research are presented, and propose future research tasks aimed at addressing the limitations.

5.2 Contributions

ANNs are often viewed as black boxes. While their predictive accuracy is high, one usually cannot understand why a particular outcome is predicted due to the complexity of the network. This thesis is an attempted to open up these black boxes by extracting rules from it through the proposed efficient rule extraction algorithm (REANN). Three factors make this possible. The first factor is that, the number of hidden nodes of the network is determined automatically in a constructive fashion by adding nodes one after another based on the performance of the network on training data. The second factor is a robust pruning algorithm. Using penalty function, it have been able to prune connections and nodes such that only very few input nodes, hidden nodes and connections left in the networks. By eliminating redundant weights, redundant input and hidden nodes are identified and removed from the networks. Removal of these redundant nodes significantly simplifies the process of rule extraction and the extracted rules themselves. The third factor is the clustering of the hidden nodes activation values. The fact that the number of distinct activation values at the hidden nodes can be made small enough enables to extract simple rules.

The REANN algorithm can extract concise rules from standard feedforward ANN. Network training and pruning is done via the simple and widely used backpropagation method. No restriction is imposed on the activation values of hidden nodes or output nodes. An important feature of rule extraction algorithm, REx, is its recursive nature. They are concise, comprehensible, order insensitive and do not involve any weight values. The accuracy of the rules from a pruned network is as high as the accuracy of the fully connected network.

Extensive experiments have been carried out in this thesis to evaluate how well REANN performed on several benchmark classification problems in ANNs including breast cancer, iris, diabetes, wine, season, golf playing, and lenses in comparison with other algorithms. In almost all cases, REANN outperformed the others. With the rules extracted by the method introduced here, ANNs should no longer be regarded as black boxes.

5.3 Limitations of REANN and Future Work

The REANN algorithm has some limitations that could be addressed in future work. Firstly, it is dependent on five user specified parameters. These are number of training epochs τ, threshold value for clustering ε, weight decay parameters ε_1 and ε_2, and penalty parameter β. The use of many user specified parameters requires a user to know rich prior knowledge, which often does not exist for complex real-world problems. Adaptive process could be used in future for making REANN less dependent to user specified parameters.

Secondly, REANN is not tested on classification problems having large number of output classes and regression problems. It would be interesting in the future to analyze REANN further on large classification problems. The analysis would help to find the strength and weakness of REANN on large classification and regression problems.

Thirdly, REANN is not applied to data mining problems. Classification is one of the data mining problems receiving great attention recently in the database community. Various classification algorithms have been designed to tackle the problem by researchers in different fields such as mathematical programming, machine learning, and statistics. Recently, there is a surge of data mining research in database community. In data mining, classification problem is re-examined under the context of large databases. Unlike researchers in other fields, database researchers pay more attention to the issue related to the volume of data. They are also concerned with the effective use of the available database techniques, such as efficient data retrieval mechanisms. With such concerns, most algorithms proposed are basically based on decision trees. The general impression is that the ANNs are not well suited for data mining. On the other hand, the use of ANNs in classification is not uncommon in machine learning community. In some cases, ANNs give a lower classification error rate than the decision trees but require longer learning time. In future, REANN could be applied for mining classification rules for large databases.

Fourthly, REANN is not considered the rule extraction technique for neuro-fuzzy network. A neuro-fuzzy network can be defined as a fuzzy system trained with some algorithm derived from ANNs. The integration of ANNs and fuzzy systems aims at the generation of a more robust, efficient and easily interpretable system where the advantages of each system are kept and their possible disadvantages are removed. Some ANN models such as the multilayer preceptron have been successfully applied to the training of neuro-fuzzy networks with back propagation algorithm to adjust the membership functions and connection weights of the processing nodes. In future, REANN could be applied for extracting rules for neuro-fuzzy network.

References

[1] R. Andrews, J. Diederich and A. B., Tickle, "Survey and critique of techniques for extracting rules from trained artificial neural networks," Knowledge Based System, vol. 8, pp. 373-389, 1995.

[2] Ashish Darbari, "Rule Extraction from Trained ANN: A Survey," Technical Report WV-2000-03, Knowledge Representation and Reasoning Group, Department of Computer Science, Dresden University of Technology, Dresden, Germany, 2000.

[3] G. G. Towell and J. W. Shavlik, "Extracting refined rules from knowledge-based neural networks," Machine Learning, vol. 13, pp. 71-101, 1993.

[4] L. Fu, "Rule learning by searching on adapted nets, " Proceedings of the Ninth National Conference on Artificial Intelligence, AAAI Press/The MIT Press, Menlo Park, CA, pp. 590-595, 1991.

[5] G. G., Towell and J. W., Shavlik, "Knowledge-based artificial neural networks," Artificial Intelligence, vol. 70, pp. 119-165, 1994.

[6] M.W., Craven and J. W., Shavlik, "Using sampling and queries to extract rules from trained neural networks," Proceedings of the Eleventh International Conference on Machine Learning, Morgan and Kaufmann, San Mateo, CA, 1994.

[7] H. Liu and S. T. Tan, "X2R: A fast rule generator," Proceedings of IEEE International Conference on Systems, Man and Cybernetics, Vancouver, CA, 1995.

[8] R. Setiono and Huan Liu, "Understanding neural networks via rule extraction," Proceedings of the 14th International Joint Conference on Artificial Intelligence, pp. 480-485, 1995.

[9] S. Thrun, "Extracting rules from artificial neural networks with distributed presentations, " Advances in Neural Information Processing Systems, vol. 7, The MIT Press, Cambridge, MA, 1995.

[10] R. Setiono, "Extracting rules from pruned neural networks for breast cancer diagnosis," Artificial Intelligence in Medicine, vol. 8, pp. 37-51, 1996.

[11] R. Setiono and H. Liu, "Symbolic presentation of neural networks," IEEE Computer, pp. 71-77, 1996.

[12] I. Taha and J. Ghosh, "Three techniques for extracting rules from feedforward networks," Intelligent Engineering Systems Through Artificial Neural Networks, vol. 6, pp. 23-28, ASME Press, St. Louis, 1996.

[13] Huan Liu, "Efficient rule induction from noise data," Expert Systems with Applications, vol. 10, pp. 275-280, 1996.

[14] Andrzej Lozowski, Tomasz J. Cholewo, and Jacek M. Zurada, "Symbolic rule representation in neural network models," Proceedings of the Second Conference on Neural Networks and their Applications, vol. 2, pp. 300-305, 1996.

[15] R. Setiono, "Extracting rules from neural networks by pruning and hidden-unit node splitting," Neural Computation, vol. 9, pp. 205-225, 1997.

[16] M. W. Craven and J. W. Shavlik, "Understanding time series networks: a case study in rule extraction," International Journal of Neural Systems, Special Issue on Noisy Time Series, 1997.

[17] Huan Liu, "A family of efficient rule generators," Encyclopedia of Computer Science and Technology, vol. 39, pp. 15-28, Marcel Dekker Inc., New York, 1998.

[18] R. Setiono and W. K. Leow, " FERNN: An algorithm for fast extraction of rules from neural networks," Applied Intelligence, vol. 12, pp. 15-25, 2000.

[19] R. Setiono, "Extracting M-of-N rules from trained neural networks," IEEE Transactions of Neural Networks, vol. 11, pp. 512-519, 2000.

[20] R. Setiono, W. K. Leow and Jack M. Zurada, "Extraction of rules from artificial neural networks for nonlinear regression," IEEE Transactions of Neural Networks, vol. 13, pp. 564-577, 2002.

[21] Z.-H. Zhou, Y. Jiang, and S.-F. Chen, "Extracting symbolic rules from trained neural network ensembles," AI Communications, vol. 16, pp. 3-15, 2003.

[22] W. S. McCulloch, and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," Bulletin of Mathematical Biophysics, vol. 5, pp. 115-133, 1943.

[23] N. Wiener, "Cybernetics: Or Control and Communication in the Animal and the Machine," New York: Wiley, 1948.

[24] D. O. Hebb, "The Organization of Behavior: A Neuropsychological Theory," New York: Wiley, 1949.

[25] F. Rosenblatt, "The Perceptron: A probabilistic model for information storage and organization in the brain," Psychological Review, vol. 65, pp. 386-408, 1958.

[26] F. Rosenblatt, "On the convergence of reinforcement procedures in simple perceptrons," Cornell Aeronautical Laboratory Report, VG-1196-G-4, Buffalo, NY, 1960.

[27] B. Widrow, and Jr. M. E. Hoff, "Adaptive switching circuits," IRE WESCON Conversion record, pp. 96-104, 1960.

[28] B. Widrow, "Generalization and information storage in networks of Adaline 'neurons'," M. C. Yovitz, G. T. Jacobi, and G. D., Goldstein, eds., Self Organizing Systems, Washington, D. C: Spartan Books, pp. 435-461, 1962.

[29] S. Amari, "A theory of adaptive pattern classifiers," IEEE Transactions on Electronic Computers, vol. EC-16, pp. 299-307, 1967.

[30] N. J. Nilsson, "Learning Machine: Foundations of Trainable Pattern-Classifying Systems," New York: McGraw-Hill, 1965.

[31] M. L. Minsky, and S. A. Papert, "Perceptron," Cambridge, MA: MIT Press, 1969.

[32] C. von der Malsburg, "Self organization of orientation sensitive cells in the striate cortex," kybernetik, vol. 14, pp. 85-100, 1973.

[33] D. J. Willshaw, and C. von der Malsburg, "How patterned neural connections can be set up by self-organization," Proceedings of the Royal Society of London Series B, vol. 194, pp. 431-445, 1976.

[34] S. Grossberg, "How does a brain build a cognitive code?," Psychological Review, vol. 87, pp. 1-51, 1980.

[35] S. Grossberg, "Neural expectation: Cerebellar and retinal analogs of cell fired by learnable or unlearned pattern classes," Kybernetik, vol. 10, pp. 49-57, 1972.

[36] S. Grossberg, "Adaptive pattern classification and universal recording: II. Feedback, expectation, olfaction, illusions," Biological Cybernetics, vol. 23, pp. 187-202, 1976.

[37] T. Kohonen, "Self organized formation of topologically correct feature maps," Biological Cybernetics, vol. 43, pp. 59-69, 1982.

[38] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," Science, vol. 220, pp. 671-680, 1983.

[39] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equations of statecalculations by fast computing machines," Journal of Chemical Physics, vol. 21, pp. 1087-1092, 1953.

[40] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations of back-propagation errors," Nature (London), vol. 323, pp. 533-536, 1986.

[41] D. E. Rumelhart, and J. L. McClelland, eds., "Parallel Distributed Processing: Exploration in the Microstructure of Cognition," vol. 1, Cambridge, MA: MIT Press, 1986.

[42] P. J. Werbos, "Beyond regression: New tools for prediction and analysis in behavioral science," Ph. D. Thesis, Harvard University, Cambridge, MA, 1974.

[43] A. E. Bryson, Jr., and Y. C. Ho, "Applied optimal control," Blaisdell. (Revised printing, 1975) Hemisphere Publishing, Washington, D. C, 1969.

[44] D. S. Broomhead, and D. Lowe, "Multivariable functional interpolation and adaptive networks," Complex Systems, vol. 2, pp. 321-355, 1988.

[45] O. A. Bashkirov, E.M. Braverman, and I. B. Muchnik, "Potential function algorithms for pattern recognition learning machines," Automation and Remote Control, vol. 25, pp. 629-631, 1964.

[46] M. A. Aizerman, E. M. Braverman, and L. I. Rozonoer, "Theoretical foundations of the potential function method in pattern recognition learning," Automation and Remote Control, vol. 25, pp. 821-837, 1964.

[47] M. A. Aizerman, E. M. Braverman, and L. I. Rozonoer, "The probability problem of pattern recognition learning and the method of potential functions," Automation and Remote Control, vol. 25, pp. 1175-1193, 1964.

[48] R. O. Duda and P. E. Hart, "Pattern Classification and Scene Analysis," New York: Wiley, 1973.

[49] T. Poggio, and F. Girosi, "Networks for approximation and learning," Proceedings of the IEEE, vol. 78, pp. 1481-1497, 1990.

[50] C. A. Mead, "Analog VLSI and Neural Systems," Reading, MA: Addison-Wesley, 1989.

[51] B. Boser, I. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," Fifth Annual Workshop on Computational Learning Theory, pp. 144-152, San mateo, CA: Morgan Kaufmann, 1992.

[52] C. Cortes, and V. Vapnik, "Support vector networks," Machine Learning, vol. 20, pp. 273-297, 1995.

[53] V. N. Vapnik, "The Nature of Statistical Learning Theory," New York: Springer-Verlag, 1995.

[54] V. N. Vapnik, "Statistical Learning Theory," New York: Wiley, 1998.

[55] V. N. Vapnik, and A. Ya. Chervonenkis, "On the uniform convergence of relative frequencies of events to their probabilities," Theoritical Probability and Its Applications, vol. 17, pp. 264-280, 1971.

[56] V. N. Vapnik, "Estimation of Dependencies Based on Empirical Data," Springer-Verlag, 1982.

[57] Simon Haykin, "Neural Neuworks-A Comprehensive Foundation," Second Edition, Pearson Education Asia, Third Indian Reprint, 2002.

[58] S. Ramón y Cajál, "Histologie duSystems Nerveus de l'homme et des vertebras," Paris, Maloine, 1911.

[59] 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.

[60] F. Faggin, "VLSI Implementation of Neural Networks," Tutorial Notes, International Joint Conference on Neural Networks, Seattle, 1991.

[61] T. Y. Kwok and D. Y. Yeung, "Constructive algorithms for structured learning in feedforward neural networks for regression problems," IEEE Transactions on Neural Networks, vol. 8, pp. 630-645, 1997.

[62] M. Monirul Islam, Xin Yao, and K. Murase, "A constructive algorithm for training cooperative neural network ensembles," IEEE Transactions on Neural Networks, vol. 14, pp. 820-834, 2003.

[63] R. Parekh, J.Yang, and V. Honavar, "Constructive neural network learning algorithms for pattern classification," IEEE Transactions on Neural Networks, vol. 11, 2000.

[64] T. Ash, "Dynamic node creation in backpropagation networks," Connection Science, vol. 1, pp. 365–375, 1989.

[65] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," Advances in Neural Information Processing System 2, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, pp. 524-532, 1990.

[66] R. Setiono and L.C.K. Hui, "Use of quasi-Newton method in a feedforward neural network construction algorithm", IEEE Transactions on Neural Networks, vol. 6, pp. 273-277, 1995.

[67] S. M. Kamruzzaman, Ahmed Ryadh Hasan, Abu Bakar Siddiquee, and Md. Ehsanul Hoque Mazumder, "Medical diagnosis using neural network," Proceedings of the International Conference on Electrical and Computer Engineering (ICECE-2004), BUET, Dhaka, pp. 537-540, 2004.

[68] M. Monirul Islam, M. A. H. Akhand, M. Abdur Rahman and K. Murase, "Weight freezing to reduce training time in designing artificial neural networks", Proceedings of 5th International Conference on Computer and Information Technology, EWU, Dhaka, pp. 132-136, 2002.

[69] J. Sietsma and R. J. F. Dow, "Neural net pruning-why and how?," Proceedings of IEEE International Conference on Neural Networks, vol. 1 (San Diego), pp. 325-333, 1988.

[70] A. van Ooyen and B. Nienhuis, "Improving the convergence of backpropagation algorithm," Neural Networks, vol. 5, pp. 465-471, 1992.

[71] R. Reed, "Pruning algorithms-A survey," IEEE Transactions on Neural Networks, vol. 4, pp. 740-747, 1993.

[72] Han Jiawei, Micheline Kamber, "Data Mining: Concepts and Techniques," Morgan Kaufmann Publisher: CA, 2001.

[73] L. Kaufman, P. J. Rousseeuw, "Finding Groups in Data: An Introduction to Cluster Analysis," John Wiley & Sons, 1990.

[74] T. Ng. Raymond, Jiawei Han, "Efficient and effective clustering methods for spatial data mining," VLDB Conference, Santiago, Chile, 1994.

[75] M. Monirul Islam and K. Murase, "A new algorithm to design compact two hidden-layer artificial neural networks", Neural Networks, vol. 4, pp. 1265–1278, 2001.

[76] J. R. Quinlan, "C4.5: Programs for Machine Learning," Morgan Kaufmann, San Mateo, CA, 1993.

[77] S. Russel and P. Norvig, "Artificial Intelligence: A Modern Approach," Prentice Hall, 1995.

[78] R. Agrawal, T. Imielinski, and A. Swami, "Database mining: A performance perspective," IEEE Transactions on Knowledge and Data Engineering, vol. 5, pp. 914-925, 1993.

[79] S-J Yen and A. L. P. Chen, "An efficient algorithm for deriving compact rules from databases," Proceedings of the Fourth International Conference on Database Systems for Advanced Applications, 1995.

[80] L. Prechelt, "Proben1-A Set of Neural Network Benchmark Problems and Benchmarking Rules", University of Karlsruhe, Germany, 1994.

[81] C. Blake, E. Keogh, and C. J. Merz, "UCI repository of of machine learning databases [http://www.ics.uci.edu/~mlearn/MLRepository.htm]," Department of Information and Computer Science, University of California, Irvine, CA, 1998.

[82] R. Setiono, "Techniques for extracting rules from artificial neural networks," Plenary lecture presented at the 5th International Conference on Soft Computing and Information Systems, Iizuka, Japan, October 1998.

[83] L. Breiman, J. Friedman, R. Olshen, and C. Stone, "Classification and Regression Trees," Wadsworth and Brooks, Monterey, CA, 1984.

[84] D. T. Pham and M. S. Aksoy, "Rules: A simple rule extraction system," Expert Systems with Applications, vol. 8, 1995.

[85] D. T. Pham and M. S. Aksoy, "An algorithm for automatic rule induction," Artificial Intelligence in Engineering, vol. 8, 1994.

[86] J. Cendrowska, "PRISM: An algorithm for inducting modular rules," International Journal of Man-Machine Studies, vol. 27, pp. 349-370, 1987.

[87] R. Setiono, "A penalty function approach for pruning feedforward neural networks," Neural Computation, vol. 9, pp. 185-204, 1997.

[88] R. Blassing, "GDS: Gradient decent generation of symbolic classification rules," Advances in Neural Information Processing Systems, vol. 6, pp. 1093-1100, Morgan Kaufmann, San Mateo, CA, 1994.

[89] Y. Hirose, K. Yamashita, and S. Hijiya, "Backpropagation algorithm which varies the number of hidden units," Neural Networks, vol. 4, pp. 61-66, 1991.

[90] M. Lehtokangas, "Modeling with constructive backpropagation," Neural Networks, vol. 12, pp. 707-716, 1999.