BUILDING EFFICIENT SYSTEMS FROM DATA IN A COMPUTATIONAL INTELLIGENCE FRAMEWORK

Debrup Chakraborty
Electronics and Communication Sciences Unit
Indian Statistical Institute
Kolkata - 700108
India.

A thesis submitted to the *Indian Statistical Institute*
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
August 2004

ACKNOWLEDGEMENTS

This work was done under the supervision of Prof. Nikhil R. Pal. I would like to record my gratitude to him for his constant encouragement, guidance and inspiration. He has been a friend and guide to me for the last few years, and I profited immensely in his company. I am grateful to him for the time he spent on discussing our research and scrutinizing it critically and with painstaking care; and also for allowing our joint works to be included in this thesis.

I would like to thank Prof. B.N. Chatterjee of IIT Kharagpur along with the other members of the Research Fellow Advisory Committee. Their critical comments during my annual reviews have been very helpful.

I shared an office with Mr. D.P. Muni for the last four years. I am thankful to him for his companionship and encouragement. He along with with Arindam, Achintya, Bishwadeep, Sanjaya have been great friends and they have helped me a lot during my stay in ECSU.

I would like to thank Mr. P.P. Mohanta, who has been very friendly and who performed the difficult duty of maintaining software and machines in our laboratories.

This thesis is written in LaTeX. Most of the codes for simulation were compiled using *gcc*, the illustrations were done by *xfig*. These along with many others are *free* software. I wish to use this opportunity to acknowledge my indebtedness to everyone who has been a part of the free software movement.

The Institute's library staff, inmates of R.S. hostel and everyone in the Electronics and Communication Sciences Unit (ECSU) have always been co-operative and friendly, which helped a lot. ECSU provided a great environment for research, which made everyday worth looking forward to.

I would like to thank Indian Statistical Institute for providing me financial support to carry out this research.

ISI, Kolkata
August 2004. (Debrup Chakraborty)

Contents

List of Figures

List of Tables

List of Important Abbreviations

ARD	Automatic Relevance Detection
BIC	Boundary Indicator Component
BNN	Biological Neural Network
BVN	Boundary Vigilance Network
CBF	Component Basis Function
CI	Computational Intelligence
EBP	Error Backpropagation
FA	Function Approximation
FCM	Fuzzy c Means
FQI	Feature Quality Index
GA	Genetic Algorithm
GFS	Group Feature Selection
GFSMLP	Group Feature Selecting Multilayer Perceptron
GFSRBF	Group Feature Selecting Radial Basis Function
k-nn	k- Nearest Neighbor
LDA	Linear Discriminant Analysis
MLP	Multilayer Perceptron
MST	Minimal Spanning Tree
OFS	Online Feature Selection
PAC	Probably Approximately Correct
PCA	Principal Component Analysis
PI	Performance Index
PR	Pattern Recognition
RBF	Radial Basis Function
RVN	Receptive Field Vigilance Network
SBS	Sequential Backward Selection
SFFS	Sequential Floating Forward Selection
SFS	Sequential Forward Selection
SSE	Sum of Square Error
SVD	Singular Value Decomposition

Chapter 1

Introduction and Scope of the Thesis

1.1 Introduction

There are problems of interest for which precise mathematical or physical understanding is yet to come. Let us illustrate this with an example. Suppose, for a mining operation one needs to blast a certain portion of the soil/rock using some specific explosives. Before the blast is made, the miners want to know the intensity of vibration that would be produced at a certain distance from the site of blasting. The intensity of vibration may depend on several factors. For example, it will depend on the characteristics and quantity of the explosive used, the rock characteristics of the region, the distance between the blasting site and the point where vibration is measured. But, we do not know precisely how these factors control the intensity of vibration. There are physics based models which model the gross scenario, but they are based on assumptions which are sometimes too simple to be useful. There are numerous such problems for which we do not yet have precise "physics based" models. For such problems, it is known that the outputs bear certain relationship with the possible inputs. But the underlying process governing the relation between the inputs and outputs is believed to be so complex that it is difficult to formulate physical models to explain it to our satisfaction. For such problems it is possible to obtain past data in the form of input-output observations. We can assume that there is some unknown function which maps the relevant input space to the output space. Based on the available input-output data our task is to construct a computational system which can act as the function to perform this transformation. Such systems built from data attempt to mimic the original system, and can be used for future predictions. This thesis deals with identification of

such systems from input-output data.

Systems which have been developed using input-output examples are sometimes called learning systems. Such systems implicitly or explicitly learn some function from data. These can be classified into two categories based on the type of output they produce. Systems in the first category produce numerical values as outputs (these are termed as function approximation type systems). For example, one may train a system to predict the price of a stock based on available data. Such a system will produce a numerical output, which in this case is the predicted price of a stock. Systems of the second kind do not produce numerical values but class labels or some decision (such a system can be called a classifier type system). For example, a classifier can be designed to decide whether a pixel denotes land or water from the gray level characteristic of a remotely sensed image. Note that, the numerical output of a system may also be interpreted as a decision.

Building systems which learn from input-output examples has been a problem of interest for a long time. This problem primarily falls under the broad field of pattern recognition. Pattern recognition (PR) deals with algorithms and methods to find regularities in data and consequently using them for the task of recognition. There are three main tasks involved in pattern recognition: classification, clustering and feature analysis. Pattern recognition often become an integral part of designing non PR systems. For example, clustering is extensively used for designing control systems [36, 37, 38, 159, 167].

Pattern recognition has been an active field of research for the last few decades [44, 48, 61]. In its formative years pattern recognition was done through statistical methods [90, 96]. The traditional statistical methods are always not well suited for analyzing complex data which arise in today's applications. With increased need for more specialized methods for complex problems various other sophisticated methods have come into being. There have been a tremendous advancement both in terms of statistical theory and techniques to deal with complex data of high dimensionality and large size [76, 232].

Another set of methods which has evolved to solve the problem of learning is called the *computational intelligence* (CI) methods. These methods are quite useful and popular to build "intelligent" systems. There is a long history of attempts to make machines

intelligent. It has been recognized, that an important facet of human intelligence comes from its ability to recognize patterns based on past experience. Thus, there are families of methods which are biologically motivated, and try to imitate some of the characteristics of the computation that goes in the human brain. Artificial neural networks form an important category of such methods. There are a number of artificial neural architectures which can learn from input-output examples. It has been proved that some of these architectures have the universal learning ability, i.e., they can learn any "reasonable" non-linear function [77, 84]. In addition, these systems are robust to noise, fault tolerant and have capability to generalize [77, 253]. Moreover, unlike the traditional statistical techniques neural networks work without any assumption on the distribution of the data. Consequently, neural networks have been quite successful in providing reasonable solutions to many learning problems [77].

A desirable property of decision making systems would be to have reasoning ability as human beings do. Also, it would be better if linguistic rules from domain experts can be incorporated into such systems. The theory of fuzzy sets and fuzzy logic help to a great extent to achieve these. The paradigm of fuzzy logic is very apt to handle certain types of uncertainties which are inherent in real systems. Fuzzy logic deals with reasoning with linguistic variables in an approximate sense. Thus, fuzzy systems have more flexibility, can handle uncertainty and the outputs produced by such systems are often more interpretable and useful. Fuzzy logic has long been successfully used to design control systems, classifiers and other other application systems designed from data [12, 46, 117, 157, 180].

Neural networks have excellent learning and generalization abilities but they usually lack interpretability and work as a black box. On the other hand, fuzzy systems are well known for their interpretability. Thus an integration of these two important components of CI produces systems with advantages of both [137]. So, there have been many attempts to integrate neural networks and fuzzy systems. Such systems are called neuro-fuzzy systems [104, 137, 183].

This thesis deals with building *efficient* systems from data using neural networks and neuro-fuzzy systems. Efficiency of a system designed from data is not only dependent on the accuracy with which it can predict but also on certain other attributes. Some such attributes are: readability, low complexity, generalization ability, ability to do feature analysis, ability to say "don't know", incremental learnability etc. Not all sys-

tems possess all these properties, and there are increased efforts in all directions to make systems which possess these properties. The emphasis of the proposed methodologies in this thesis is to incorporate some of these desirable properties in the systems. Particularly we have concentrated on two attributes of efficiency, namely, capability to do *feature analysis* and *generalization ability*. Ability to say "don't know" is related to generalization ability, as it deals with the problem of identifying the areas in the input space where a system should generalize. We have developed systems which are capable of saying "don't know" when appropriate. These systems also have incremental learning ability.

It is known that for a given problem all features that characterize a data point may not be equally important. Some features may be redundant while others may even be *bad* features. Feature analysis techniques aim to reduce the number of features to a small but adequate (adequate for the task at hand) subset of the available features, i.e., they aim to discard the bad/ irrelevant features from the available set of features. This reduction may improve the performance of classification, function approximation and other pattern recognition systems in terms of speed, accuracy and simplicity. Most methods of feature analysis act as a preprocessing step on the data, and they precede the main learning task. It has been long noticed that the suitability of the features depends on the learning algorithm [43, 44, 118]. Hence, it would be best if the learning machine can be equipped with an inherent feature selection capability. We have proposed systems which have a feature selection component inherent in them. We call such feature selection techniques as *online* techniques. In this thesis we develop neuro-fuzzy systems for function approximation and classification which can select the relevant features from data when they get trained. Consequently, such schemes give rise to less complex systems with enhanced generalization abilities. We also propose an online scheme for doing sensor selection. This problem of sensor selection has been addressed using modified versions of two traditional neural network architectures, namely, the Multilayer Perceptron (MLP) and the Radial Basis Function (RBF) networks.

Another important problem that we address here is on generalization. We provide methodologies to train neural networks by a specialized technique so that they can learn the relevant areas of the input space from which the training data might have come, and they do not respond to test points which lie outside the "boundary" of the training data. In other words, our method of training enables neural networks

to identify areas in the input space for which it should produce a response. We call this as *strict generalization*. This method can be easily used for incremental learning of the neural networks, i.e., one can incorporate new knowledge in a trained neural network without hampering its previous memories. We also deal with the problem of overfitting in MLP networks. A serious problem with neural network training is that it is trained with a finite training sample, and a finite training sample can easily result in an overfitted neural network which generally produces bad generalization. We propose a method to overcome this problem too. Our method relies upon a scheme to expand the given training set to any desired size, and it produces consistent generalizations.

In this chapter, in the sections to follow, we first formally describe the problem of learning from data. Then, in Section 1.3 we discuss the computational intelligence paradigm, and in some detail we describe neural networks and fuzzy systems. These tools would be primarily used throughout the thesis. In fact, all schemes described in this thesis are connectionist systems, and some are neuro-fuzzy hybrids. In Section 1.4 we discuss some of the desirable characteristics of systems built from data, and finally in Section 1.5 we discuss the scope of the thesis, and provide *summary* of the chapters that follow.

1.2 Systems Built from Data : A Formal Look

Let $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N\} \subset \Re^s$ and $Y = \{\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_N\} \subset \Re^t$ and let there be an unknown function $\boldsymbol{S} : \Re^s \Rightarrow \Re^t$ such that $\boldsymbol{y}_k = \boldsymbol{S}(\boldsymbol{x}_k)\ \forall k = 1, ..., N$. In other words, there is an unknown function $\boldsymbol{S}$ which transforms $\boldsymbol{x}$ to $\boldsymbol{y}$. $\boldsymbol{S}$ can represent many types of systems. $\boldsymbol{S}$ can be an *algebraic system*, where $\boldsymbol{S}$ does not evolve with time, and thus can be represented by a set of algebraic equations. On the other hand, $\boldsymbol{S}$ can be a dynamical system, which evolves with time, and hence differential equations are required to characterize them. Again, $\boldsymbol{S}$ can be characterized by the type of output it produces. The output may be continuous numerical valued or they can be decisions as in a classifier.

Finding $\boldsymbol{S}$ from the input-output data (X, Y) is often called *system identification*. $\boldsymbol{S}$ can be of two types: regression or function approximation (FA) type when the elements in Y are continuous, and classifier type when Y contains class labels (or

categorical decisions). In both cases it is assumed that the input-output data (X, Y) are generated from a time invariant but unknown probability distribution. The data set $T = (X, Y) = \{(\boldsymbol{x}_i, \boldsymbol{y}_i) : i = 1, ..., N\}$ that is used to find $\boldsymbol{S}$ is called a training set. During training, $\boldsymbol{x}_i, (i = 1, 2, ..., N)$ is used as the input and the outputs $\boldsymbol{y}_i, (i = 1, 2, ..., N)$ acts as a teacher. The design methodology uses this training set T to learn (build) a system with a hope that the obtained system will work well not only on the training inputs but also on data points which are not present in the training set. The performance of such a system is measured by its performance on future data, commonly called test data. The test data set is independent of the training set, and it is also assumed that the test data follow the same probability distribution as that of the training data.

There are numerous ways to find $\boldsymbol{S}$. $\boldsymbol{S}$ can be determined explicitly through regression or other statistical methods [76]. Similarly, a classifier function may be an analytic function as in case of a Bayes' classifier [48]. These traditional tools make certain assumptions about the data. In case of parametric regression one needs to have some knowledge about the functional form of $\boldsymbol{S}$, which is often difficult to get for many real life applications. Similarly, the optimal Bayes' classifier needs knowledge of the apriori probabilities and class-conditional density functions of the data [48]. Obtaining them for real data sets is quite difficult. Also there are nonparametric regression methods which can work without a parametric assumption on the data [76]. On the other hand, one may obtain *implicit* solutions of such problems through computational transforms like neural networks, fuzzy systems and other hybrid systems. These type of systems do not require prior knowledge regarding the distribution of the data. These tools (neural networks, fuzzy logic, evolutionary computation etc.) are collectively known as computational intelligence (soft computing) tools. In this thesis our main aim is to develop "efficient" systems using some computational intelligence tools. In our view efficiency of a system is just not a function of better generalization. For example, if a system uses less number of features (sensors) then also it is efficient in terms of design cost and complexity. We next provide a brief overview of computational intelligence along with a detailed discussion of specific computational intelligence tools that would be used through out the thesis.

1.3 The Computational Intelligence Framework

There have been several attempts to define *Computational Intelligence (CI)*. Bezdek [11], in 1994, was the first to introduce this term (in the present context): "... A system is computationally intelligent when it: deals with only numerical (low-level) data, has a pattern recognition component, does not use knowledge in the artificial intelligence sense; and additionally when it (begins to) exhibit (i) computational adaptivity; (ii) computational fault tolerance; (iii) speed approaching human-like turnaround, and (iv) error rates that approximate human performance."

In 1995, Fogel [58] summarized CI as "... These technologies of neural, fuzzy and evolutionary systems were brought together under the rubric of Computational Intelligence, a relatively new term offered to generally describe methods of computation that can be used to adapt solutions to new problems and do not rely on explicit human knowledge."

Irrespective of the way CI is defined, CI tools should have sufficient ability to solve practical problems. It should be able to learn from experience and capable of self-organizing. Consequently, it is reasonable to assume that the major constituents of a computational intelligence system are artificial neural networks, fuzzy sets, rough sets, evolutionary computation, and immuno-computing.

The major tools of CI are also encompassed by another widely used term, *soft computing*, which refers to a collection containing neural networks, fuzzy logic, evolutionary computation etc. Soft-computing is defined as a consortium of different computing tools that can exploit our tolerance for imprecision and uncertainty to achieve tractability, robustness and low cost [249]. We prefer the term " computational intelligence".

Two very popular and widely used tools of computational intelligence are neural network and fuzzy logic. These tools have been successfully used in various domains to make human-like systems. We shall next present a brief overview of these tools.

1.3.1 Neural Networks

It is well established that the site for intelligence in human beings or other living beings is the brain. The characteristics of the nerve cells which constitute the brain

are different from that of the other somatic cells in our body. It is believed that the computation done in the nerve cells is the precursor of intelligence. There have been many attempts to built feasible models for the nerve cells and the connectionist systems of today are a product of these research. These systems are also called artificial neural systems or artificial neural networks. These terms are used interchangeably and often for brevity the term artificial is dropped.

Biological neural networks (BNN) are a massive interconnection of nerve cells. Each nerve cell performs some simple operations and all of them perform similar operations. The group dynamics of the ensemble of the nerve cells, i.e., of the network is quite different from individual dynamics of the nerve cells. BNNs are highly robust and fault tolerant. They are capable of learning and generalization. These properties of BNNs are shared by most artificial neural models. And these properties make them attractive for use in many diverse problem areas.

Research in neural networks started from the search of a feasible computational model of the nervous system. In 1943, McCulloch and Pitts proposed a computational model of a neuron [150]. In order to describe how neurons in the brain might work, they modeled a simple neural network using electrical circuits. In 1949, Hebb [78] pointed out the fact that neural pathways are strengthened each time they are used, a concept fundamentally essential to the ways in which a human learns. These two works probably were the pioneers in neural network research. In the 1950's, Rosenblatt's work [200] resulted in a two-layer network, the perceptron, which was capable of learning certain classification tasks by adjusting connection weights. Although the perceptron was successful in classifying certain patterns, it had some limitations. The perceptron was not able to learn the classic XOR (exclusive or) problem. Consequently, interest in neural networks was at a low profile for quite sometime. However, the perceptron had laid foundations for later work in neural computing. Interest in neural networks again revived in the eighties with some path breaking works like self organizing feature map [119], the hopfield network [83], multilayer perceptron [204], adaptive resonance theory [23] etc.

The historical perspective of neural networks was in making biologically plausible systems. The objective was more towards building a model for the nervous system. But in last decade, it was observed that neural networks, both the traditional architectures, and their variants, have great potential to solve real life problems even though they

may not mimic computation in the brain. The thrust of research for the last ten years were more to develop systems which can be used to solve practical problems than to look for biological plausibility. Of course, there have been many advances towards modeling the biological nervous systems and thereby gaining new insights into the process of brain functioning. These issues are now-a-days dealt within the subject of computational neuroscience [52, 228]. We shall not be venturing into those issues.

Neural networks are lucrative for their interesting properties of learning and generalization. They can be applied without any assumptions regarding the distributions and other statistical characterizations of data. Theoretical issues about learning have been extensively studied. Valiant [231] proposed a mathematical theory of learning that addressed the issues of learnability and computational complexity of learning in a restricted framework called the "Probably approximately correct" (PAC) learning [231]. The PAC framework provides a reasonable setup to study the theoretical basis of learning. There have been a variety of theoretical studies encompassing learning in general and in particular learning in neural networks [2, 107, 232, 234]. All these studies helped to gain a better understanding of the learning and generalization process. Consequently these have helped to develop better learning systems.

Artificial neural networks have been successfully used in pattern recognition and many other fields. There are specific neural architectures which can perform the various pattern recognition tasks [15, 144]. The popular feed-forward networks like the multilayer perceptrons, radial basis function networks can learn any non-linear input-output mapping under a fairly general set of conditions [77, 84, 189]. They have been successfully used as classifiers and function approximation tools for variety of applications. The self-organizing feature map, linear vector quantization and adaptive resonance theory can be used for clustering [79]. The Hopfield network can be used as an associative memory, which can store and recall patterns [79, 253]. In addition Hopfield nets have been successfully used for various computationally hard combinatorial optimization problems [79]. There are numerous work in the literature which discusses variants of these networks and use of them for various practical problems. A glimpse of these voluminous research can be found in many books related to neural networks [15, 77, 79, 144, 234, 253].

A new paradigm for learning from data called Support Vector Machines (SVM) [232, 259, 260, 261, 262] is of great interest in recent times. Support vector machines are

more closely related to statistical learning than to CI, but a mention of this paradigm may be done here. SVM is primarily a tool for binary classification problems.

In SVM a linear separating hyperplane is constructed to distinguish between two classes. Since, most real life data are not linearly separable, in SVM often the input data points are implicitly mapped to a new high dimensional space F, and then, a linear separating hyperplane is constructed in F which classifies the mapped input points. The hyperplane is constructed by maximizing the *margin* of separation between the two classes. Such a hyperplane is called the *optimal hyperplane*. There are several theoretical arguments supporting the good generalization properties of the optimal hyperplane [260]. The learning procedure involved in identifying the optimal hyperplane is a constrained quadratic optimization problem which minimizes the weighted sum of two terms. The first term is related to the reciprocal of the margin and the other term involves the sum of classification errors. Though the optimization problem involved is a quadratic programming problem, when the associated data set is large it may get computationally expensive. Many attempts have been made to solve the optimization problem efficiently [263, 265, 266].

SVMs when used as classifiers work for two-class problems only. There are a few proposals which discuss the use of SVMs for multi-class problems. Each of these solves more than one binary classification problem and then aggregates the results obtained by those binary classifiers. One against one, one against all [267], directed acyclic graph SVM [268] are some of the methods for solving multi-class problems. This framework can also be used for regression type problems [270, 269].

Support vector machines have been successfully used in diverse applications including handwritten numeral recognition [232], object recognition [277, 279], face recognition [278, 280], bioinformatics [271, 272, 273, 274, 276], text categorization [264, 275].

1.3.2 Fuzzy Systems

Fuzzy sets were introduced in 1965 by Zadeh [250] with a view to reconcile mathematical modeling and human knowledge in engineering sciences. Since then, a considerable body of literature has blossomed around the concept of fuzzy sets in an incredibly wide range of areas, from mathematics and logics to traditional and advanced engineering

methodologies. Applications are also found in many areas including medicine, finance, control, and consumer products.

A fuzzy subset A of a (crisp) set X is characterized by assigning to each element x of X the degree of membership of x in A. For example, if X is a group of people, then the set of *Tall* people defines a fuzzy set on X. The set *Tall* has no precise class boundary and a person can belong to the *Tall* set to some extent. Mathematically the set *Tall* can be defined through a membership function μ_{Tall}, defined as

$$\mu_{Tall} : X \rightarrow [0, 1].$$

The function μ_{Tall} assigns to each element in X a value in [0,1]. This value quantifies the degree to which an element belongs to the set *Tall*. This is a very suitable way to model concepts which are inherently ambiguous and can only be understood by linguistic descriptions like tall, short, handsome, rich etc. Fuzzy logic attempts to model the way human beings reason with imprecision. It can accommodate or model imprecision which is inherent in any physical system. Thus, it has been a very successful tool for reasoning under uncertainty. This has led to development of applications which use the concept of fuzzy logic for approximate reasoning [117, 180]. These are called fuzzy systems. Among all fuzzy systems, fuzzy controllers [46, 129, 130] are probably the most widely used ones as they do not require the strict mathematical model of the system to decide on the control actions but depend on linguistic rules, which can be easily provided by domain experts. Fuzzy rules can also be extracted from input-output data [36, 37, 38, 105, 159, 167]. Specific hardware implementation of fuzzy controllers and other kinds of fuzzy systems have also been done [242, 243] which has helped a lot for use of fuzzy systems in fielded applications and consumer products.

For practical pattern recognition problems, imprecision/fuzziness is often unavoidable. For example, a designer always desire to have a classifier which can produce crisp decisions. But the numerical representation of the training data may provide overlapped class boundaries. Thus, in such cases a crisp decision is not always possible. Rather a fuzzy decision, which tells about the degree of belonging of a point to a class, is more interpretable and useful. Fuzzy logic can handle such scenarios. Consequently there is tremendous scope of fuzzy logic in developing pattern recognition systems. Fuzzy techniques for numerical pattern recognition is quite mature. There are numerous methods of fuzzy classification and clustering designed for numerous applications

[9, 10, 12, 13, 109, 111, 173, 174, 181].

1.4 Some Desirable Characteristics of Systems Built from Data

An efficient system built from data is expected to have certain attributes. Here we list some of the desirable characteristics for systems designed from data. Of course this may not be an exhaustive list, and there could be many other characteristics which are desirable given any specific application.

1.4.1 Readability

This property refers to the interpretability of the system. A good system should be easily interpretable, and one should be able to attach meanings to each parameter of the system. Consequently, for a readable/interpretable system one can find the reasons for a specific decision made by the system.

Most neural architectures lack readability. The internal parameters of a trained neural network, say an MLP, are not interpretable by any easy means. There have been numerous attempts to make neural networks more readable [137]. Attempts have also been made to extract symbolic or other rules from a trained network [47, 214, 216, 224, 225, 227]. An useful and popular way to make neural networks interpretable is through its integration with fuzzy systems, which are well known for their readability. We have considered a class of neuro-fuzzy systems called the *neural fuzzy systems* which are fuzzy systems built on neural networks [102, 103, 104, 105, 136, 137, 255]. These types of networks are readable, each internal parameter associated with these networks has some meaning associated with it. Thus, by reading these parameters from such networks we can arrive at simple linguistic rules which govern the decision making process.

1.4.2 Generalization Ability

Generalization ability is probably the most important attribute of any system learned from data. A system learned from a finite training set $T = \{(\boldsymbol{x}_i, \boldsymbol{y}_i) : i = 1, ..., N\}$

should not only perform well with points in T but also for points which are not included in the training set. This property of a learning machine to produce "good results" for future data (data not included in the training set) is called generalization ability. There are many methods in the literature to enhance generalization ability of learning systems and particularly of neural networks. In neural networks generalization ability can be enhanced in different ways including: early stopping [3], complexity control [81, 213], expanding the training set [82, 100] and ensemble methods [19, 252]. In Chapter 2 we provide a brief survey on some of these methods to enhance the generalization ability in neural networks.

1.4.3 Low Complexity

There are many model selection criteria which aim to find a model with better generalization properties [1, 5, 120, 199, 223, 233]. It has been observed that less complex systems perform better generalizations. The minimum description length criteria [5, 199] aims to find that model which can be described by the least number of bits. Out of all models which have equal performance on training data, the model with the minimum description length is expected to have the best generalization capabilities. This is an interesting manifestation of the old rule "the simplest model is the best one", which is popularly known as the Occam's Razor.

As mentioned above, a less complex system is related to better generalization. But, from a practical view point, a less complex system also has other advantages to offer. It has less running time. It may reduce the complexity of the hardware through which it gets implemented. Thus they have less setup and running costs. And it is obvious that a less complex model has a better readability.

1.4.4 Managing the Curse of Dimensionality

One of the important attributes for any learning system should be to identify and use only the necessary features. More features sometimes help to bring out more characteristics of an object and thus may help the task of building the system. But, always more features are not necessarily good. In the available set of features some may be redundant and some may be bad too. These features add on to the dimensionality

of the problem without doing any real good. Thus an increased number of features means a complex model, which is more likely to give bad generalization. Also the learning complexity and computational overhead of a system increase with the increase in the number of features. So, feature selection or dimensionality reduction is a very important step for any pattern recognition/system identification task. There have been many attempts to address this problem. Some good surveys discussing feature selection methods are [8, 18, 42, 161]. We also provide a brief survey of the feature selection methods in Chapter 2.

Feature selection is generally looked upon as a preprocessing task, which is applied on the data prior to the learning process. But, a good system designed from data should have a feature selection component embedded in it. If a system is able to detect the good features and discard the bad ones then it consequently gives rise to less complex systems, which are expected to have other good attributes like low complexity, better readability and better generalization abilities.

1.4.5 Ability to Say "Don't Know"

For most systems which are built from data, a difficult problem is to determine the operative region of that system. In other words, finding out the region in the input space where it should perform meaningful generalization. It is known, that for any system designed from data, it is not wise to extrapolate too much beyond the training data. For example, a trained classifier is not expected to perform good if a test point is *far* from the "boundary" of the training set. Thus a good property of a system would be its ability to say "don't know" for cases which are not typical to its training sample. This characteristic is very important for very precision applications like in medical diagnosis.

This property is related to the generalization ability. In formal terms this problem aims to find the sampling window from which the training data are generated, and an ideal system should not respond to points beyond the sampling window. As the true input distribution is never known, a reasonable approximation of the sampling window can be the "boundary" of the training set. Thus, a good system should generalize only in or around the boundary of the training set. We call this "strict generalization".

Multilayer Perceptrons or other feedforward neural systems do not possess this property of strict generalization. Our experiments in Chapter 6 show that an MLP may produce strong response even for points which are really very far away from the sampling window of the input distribution. In most cases this is not desirable. In the literature, adequate emphasis has not been given to address this problem. MacKay in [147] discusses a scheme for neural networks which gives low response for test points which come from areas of input space that are sparsely represented in the training data. But MacKay does not consider the areas of the input space which are not represented by training data.

1.4.6 Incremental Learnability

A system built from data should be able to learn new knowledge without losing its previous memory. This is a very important attribute of any learning system, as information about a system may come in an incremental fashion. If one collects a few data points characterizing a system, the characteristics of the system may change with time, also new facets of the systems may get discovered which should be incorporated in the model.

There are several attempts to address this problem of incremental learning [57, 60, 210]. In the neural network framework there are two distinct paradigms to achieve this. One, through bounded updates of the weights of a trained network when it faces a new training set [60], and the other by using both the new and old data points to retrain a network [177, 245]. Both these are not optimal. The first alternative may lead to incomplete learning, if the new training data are not very similar to the past data. The second alternative is computationally expensive, and the past data which it rely on may not always be available.

A system equipped with one or more of the above six characteristics will be called as an "efficient" system.

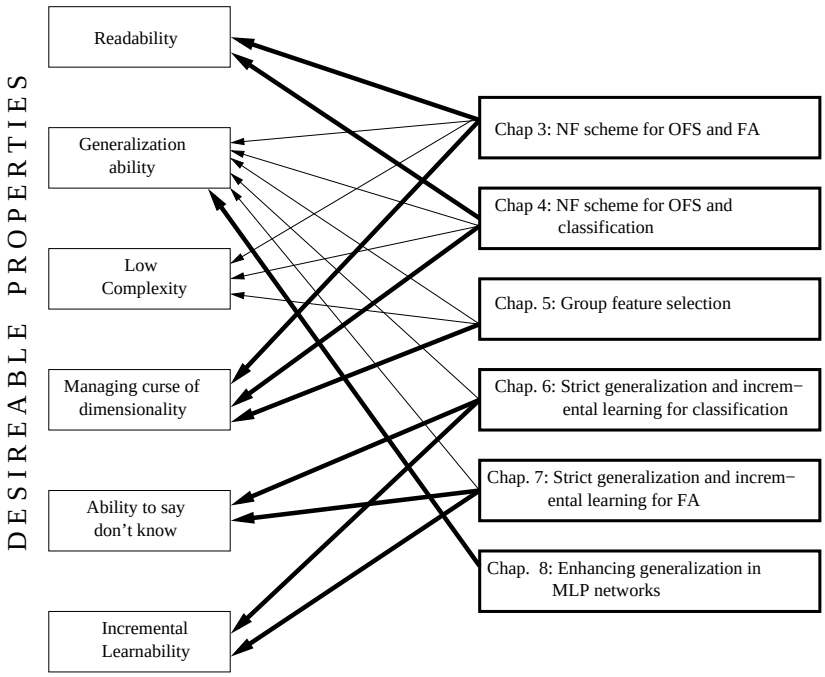


Figure 1.1: The organization

1.5 Scope of the Thesis

In this thesis we describe some new methods to build efficient systems from data. We use neural networks and neuro-fuzzy systems as tools. We described a few desirable properties for systems designed through data in the previous section. In this thesis we deal with two important properties in the list, namely managing the curse of dimensionality (Chapters 3-5)and enhancing generalization ability (Chapters 6-8). Incremental learnability is also achieved for MLP networks (Chapters 6-7).

Figure 1.1 gives a pictorial description of the theme of each of the main chapters of the thesis. On the left we have the various desirable properties of systems built from data and on the right we have the six chapters which form the main body of the thesis. There are two kinds of arrows linking chapters with the properties. A bold arrow signifies the main attribute of the system that is achieved in a chapter and a weak arrow links with an attribute/property that is also achieved as a by-product.

The feature selection/dimensionality reduction schemes that we deal with in this thesis are unique in the sense that they are integrated with the main learning algorithm. We

call these as *online feature selection* techniques. We have used these online methods to develop two types of neuro-fuzzy systems: one for function approximation and the other for classification. We have also demonstrated that online feature selection can be suitably used for selecting useful sensors using modified Radial Basis Function networks and Multilayer perceptron networks.

A system built from data should be equipped with the ability to say "don't know" when it faces unfamiliar examples. In other words, it should not respond to test points which are far away from the boundary of its training set. We term this property as *strict generalization*. We have extensively studied the generalization properties of Multilayer perceptrons both in the context of classification and function approximation and found that ordinary MLP does not possess the property of strict generalization. We have devised methods for training MLPs so that they can perform strict generalization. Also, the MLPs trained by our method can be made to learn incrementally, i.e., a trained MLP may be easily augmented with new knowledge contained in a new data set without effecting its previous memory.

Another problem that we deal in the thesis is of avoiding overfitting in MLP. Generally, neural networks are trained with a finite training sample, and hence in each epoch the network faces the same set of points and thus tends to "memorize" those points which may lead to overfitting and consequently such networks may produce poor generalization. We have devised a scheme to avoid overfitting in MLP.

In the following subsections we describe in brief the contents and main contributions in the chapters to follow.

1.5.1 Literature Survey

The thesis deals with theories and methodologies for building systems which have the capability for selecting important features and have better generalization abilities. In Chapter 2, we give a brief survey of feature selection techniques and methods to enhance generalization abilities in neural networks.

We begin with a brief survey of literature on feature analysis. We discuss the important paradigms of feature selection both using the classical and CI frameworks. We introduce the concept of *online feature selection* and discuss its relationship with and

advantages over other existing feature selection algorithms. We provide motivations behind using the online feature selection methodology. Next, we discuss some previous work on enhancing generalization abilities of neural networks. Also, we introduce *strict generalization* and discuss its utility.

1.5.2 Online Feature Selection and Function Approximation Type System Design in a Neuro-Fuzzy Paradigm [24, 164]

In Chapter 3 we begin with a short survey of previous works in neuro-fuzzy systems and then we discuss our proposed neuro-fuzzy system for integrated feature analysis and function approximation. We present a systematic approach to build a neuro-fuzzy system from multiple-input-multiple-output data. The novelty of the system lies in the fact that it can simultaneously do feature selection and system identification in an integrated manner. We call it online feature selection. The network can be trained to learn the input-output mapping and additionally the importance of the input features can be determined from the network parameters. The neuro-fuzzy system is realized by a five layer network, each layer performing a different task. The second layer of the net is the most important one, which along with fuzzification of the input also learns a modulator function for each input feature. This enables *online* selection of important features by the network. The system is so designed that learning maintains the non-negative characteristic of certainty factors of rules. To get an "optimal" network architecture and to eliminate conflicting rules, methodologies for pruning nodes and links are also proposed. The pruned network represents a small set of rules but it retains almost the same level of performance as that of the original one. The rules governing the input-output mapping can be easily extracted from the final trained network. Thus, the network is fully *readable* in terms of linguistic rules. The proposed network is tested on both synthetic and real data sets and the performance is found to be quite satisfactory.

1.5.3 Online Feature Selection and Classifier Design in a Neuro-Fuzzy Paradigm [25, 27]

In Chapter 4 we modify the network described in Chapter 3 for the classification task. As both methods in Chapters 3 and 4 are neuro-fuzzy techniques, to be more precise, they are neural-fuzzy systems, the crux of the method lies in the structure of the fuzzy rules that are used to explain the data. The structure of the fuzzy rules for a classification system is quite different from that of a function approximation system, and hence the network structure used in Chapter 4 is different from that of the network described in Chapter 3. Unlike a five layered network, the classifier network is a four layered network. The network for function approximation in Chapter 3 uses center of area type defuzzification, but such type of defuzzification cannot be applied for the classification network. The classifier network uses *max* as the defuzzification operator. Also, due to the change of the defuzzification strategy, the operator for intersection has also been changed. In the function approximation network, we use product as the operator for intersection while in the classifier we use an approximate but differentiable version of the *min* function, which we call *softmin*. To suite the classification task at hand the strategies to optimize the network are also changed. Additionally, we discuss about the tuning of the input membership functions of the network. We also deal with the issue of initialization of the neuro-fuzzy system. This problem has not been adequately discussed in literature. A blind initialization may lead to a huge network for data sets of high dimensionality. We provide guidelines for selecting the initial network for a specific data set in high dimension. Numerous simulation results on benchmark data sets are reported in this chapter which demonstrates the effectiveness of the network for online feature selection and classification.

1.5.4 Online Sensor Selection Using Feed-Forward Networks [28, 31]

Chapter 5 deals with the feature selection problem in a different setting. We try to minimize the required number of *sensors*, where each sensor can be responsible for several features. The motivation for this problem comes from the fact that many applications rely on data from multiple sensors. For example, in an intelligent welding

inspection system the sensors could be radiograph, acoustic emission, thermograph, eddy-current detector etc. The raw data obtained from these sensors are seldom used, but numerous features are extracted from each sensory data. Thus, the total number of features so obtained can be grouped into several subsets based on their sensory origin. We try to find the importance of a group of features not of an individual feature. Thus, if we can detect a less important (or redundant) group of features then we can discard the associated sensor from that application. This reduces the complexity of the hardware, system design complexity and time, computation time and running cost. Note that, sensor selection is more general than feature selection. Treating each feature as a sensor, a sensor selecting network can be used for feature selection. But, the converse is not true. This problem has not been addressed in literature. We propose two models for feature group (sensor) selection. These models are variants of the conventional RBF and MLP networks. We call them *Group Feature Selecting Radial Basis Function* (GFSRBF) network and *Group Feature Selecting Multilayer Perceptron* (GFSMLP) network. We have shown experimentally that these networks can do the task of sensor selection and classification/function approximation quite efficiently. We have also shown that GFSRBF and GFSMLP have universal approximation properties.

1.5.5 Strict Generalization and Incremental Learning in Multilayer Perceptron Networks: Classification [26, 29]

In Chapter 6 we discuss *strict generalization* in MLP networks for classification tasks. Our experiments show that the response of a multilayer perceptron (MLP) network on points which are far away from the boundary of its training data is generally not reliable. Ideally a network should not respond to a data point which lies far away from the boundary of its training data. To realize this, we propose a new training scheme for MLPs when used as classifiers. Our scheme involves training subnets for each class present in the training data. Each subnet can decide whether a data point belongs to a certain class or not. Training each subnet requires data from the class which the subnet represents along with some points outside the boundary of that class. For this purpose we first introduce the concept of an approximate boundary of a class and propose an easy method to generate points outside the "boundary" of a pattern class. The trained subnets are then merged to solve the multi-class classification problem.

We show through simulations that an MLP trained by our method does not respond to points which lie outside the boundary of its training sample. Also, our network can deal with overlapped classes in a better manner than conventional MLP. In addition, this scheme enables incremental training of an MLP, i.e., the MLP can learn new knowledge without forgetting the old knowledge. The philosophy used is quite general in nature and can be used with other learning machines also.

1.5.6 Strict Generalization and Incremental Learning in Multilayer Perceptron Networks: Function Approximation [32]

In Chapter 7 we extend the methodology developed in Chapter 6 for function approximation problems. Since the method in Chapter 6 involves training a separate subnet for each class, a direct application of the method cannot be made for function approximation problems. In Chapter 7 we propose a novel training scheme for MLPs which realizes strict generalization for function approximation type problems. The methodology consists of training two networks to perform two different tasks. The first network, called the *mapping network*, learns the usual input-output mapping present in the data and the other network, which is called the *vigilance network*, learns a decision boundary and decides the points the mapping network should respond to. The design of the vigilance network has been addressed in two ways. The first approach depends on a methodology to generate points outside the boundary of the training set. The points in the training set are assumed to belong to one class and the generated points in the other class. Thus, the vigilance net learns a two class problem, and it can detect whether a test data point falls within the boundary of the training set or not. For high dimensional data sets this methodology becomes computationally expensive because we need to generate too many points to characterize the boundary of the training data. Hence, we propose another simple method to train the vigilance network, which does not need any additional training examples. This method breaks up the training data into hyperspherical clusters using a clustering algorithm. And it builds local receptive fields centered at each cluster using bell shaped functions as in RBF type networks. The parameters of the bell shaped functions are decided using the clustering results. For a test point, if the responses from all receptive fields are low then we conclude that

the point lies outside the boundary of the training sample. The strict generalization capability of the network is demonstrated using both approaches. In this system also we achieve incremental learning as a byproduct.

1.5.7 Enhancing the Generalization Ability of Multilayer Perceptron Networks [30]

Chapter 8 deals with the problem of overfitting in MLP. Typically an MLP is trained with a finite training set. Since the right size of the network is not known, often the network overfits the training data and thus produces poor generalizations. There are many previous attempts to reduce overfitting in MLP. Some of the main paradigms through which this is achieved are complexity regularization, weight pruning, early stopping etc. Here we devise an easy scheme to expand the training set to any desirable size, such that the problem of overfitting is partially removed. Our scheme uses a k-nearest neighbor heuristic to generate additional training examples which implicitly act as constrains on the training and help to reduce overfitting. Our method is tested for both classification and function approximation problems, and the results demonstrate that the proposed method performs quite well in comparison to other similar methods for enhancing the generalization ability in MLP networks.

1.5.8 Conclusions and Future Work

We conclude the thesis in Chapter 9. We summarize our findings, identify some of the limitations of our methods and provide discussions on further scopes and directions of research.

Chapter 2

Literature Survey

In this chapter we provide a brief survey of some of the important works on two important attributes of systems designed from data, namely, ability to do feature analysis and generalization ability.

2.1 Feature Analysis

All features that are present in a data set may not be useful for the task at hand. Some features present may be redundant and some may be bad too. Thus selecting the most relevant and useful features from a given set of features is useful as it helps in building systems with low complexity and may save computational time and future data collection efforts. Also it has been shown that reducing the number of features may lead to an improvement of the prediction accuracy due to finite sample size effects [89]. Moreover, a system built with a smaller number of features is more readable, has less parameters and is expected to have better generalization abilities.

Feature analysis deals with finding a transform $\Phi : \Re^s \rightarrow \Re^q$ using a criterion J on a training set $T = (X, Y) = \{(\boldsymbol{x}_i, \boldsymbol{y}_i) | \boldsymbol{x}_i \in \Re^s, \boldsymbol{y}_i \in \Re^t, i = 1, 2, ..., N\}$. Typically, J is related to the problem that we intend to solve using T. The transform Φ is said to perform dimensionality reduction if $q < s$.

Feature analysis can be divided into three basic types: feature extraction, feature selection and feature ranking. Feature extraction is a method to generate a q dimensional vector from a given s dimensional input vector. For feature extraction it is not necessary that $q < s$. There may be applications where one needs to find a richer description

of objects by increasing the number of features. Extracting additional features from a given feature vector is common in many image processing and other signal processing applications, but we shall not be dealing with those methods. We are only interested in transformations which perform a reduction in the dimensionality of the input space.

The feature extraction process projects the original features in a different space of lower dimensionality following some criteria. The reduced set of features may be computed as linear/nonlinear combination of the original ones which may not bear the meanings of the original ones. A special case of feature extraction is feature selection where the components of $\Phi(\boldsymbol{x})$ are some components of $\boldsymbol{x}$, i.e., in this process some components of the original vectors are discarded and others are retained as they are. Thus, in feature selection a subset of the original set of features is selected using some criteria.

Feature ranking methods aim to rank features according to their suitability for the task at hand. The ranking algorithm assigns some real number to each of the features, and one can order the features according to these real numbers. Feature selection may then be performed based on these ranks. But ranking individual features is not the best way to look at the problem because two correlated features may get high rank, while only one of them is required for the task.

Feature selection algorithms can be characterized based on two attributes namely the feature evaluation criterion function J and the search technique. Popular choices of J include measures of class separability, information, the classification error rate etc. [42]. Once the criteria J gets fixed, the problem of feature selection reduces to a search problem. An exhaustive search would require examining $\binom{s}{q}$ possible q subsets of the original feature set. The number of possibilities grow exponentially making exhaustive search impractical even for moderate values of s. To cope with this problem there are numerous heuristics to arrive at a near optimal feature set [44]. Also there are many methods which do not explicitly use a search technique, but use a transformation on the available features to obtain a reduced set of features.

Kohavi and John [118] classified feature selection techniques as wrapper methods and filter methods. Filter methods are independent of the main learning algorithm (commonly known as the induction algorithm in machine learning literature). And wrapper methods are dependent on the learning algorithm. In other words, the wrapper methods evaluate the feature subsets based on their error rates on a specific learning

machine. Wrapper methods are found to be better than the filter methods as it has been observed that suitability of a feature subset is dependent on the learning algorithm [43, 118]. But, wrapper approaches are computationally much more expensive, as one needs to build different learning machines with different feature sets and then evaluate the feature set with that machine. For example, if the learning machine is a feed-forward neural network (like a multilayer perceptron (MLP)), one has to train an MLP with each feature subset that is considered. Obviously, this is very expensive in terms of computation time.

Feature selection can also be classified according to the tools that are used to arrive at the optimal feature set. There are many methods that use statistical measures of class separability etc. as the feature selection criteria and employ different search heuristics to select suitable feature subsets [44]. Another popular paradigm called the Computational Intelligence (CI) paradigm has also been used for feature selection. Neural networks, fuzzy logic and genetic algorithms constitute the primary tools of CI. There are many methods which use these tools for feature selection [43, 161, 195, 213].

Feature selection is a huge field of study and there is a huge body of work which addresses this problem. Some good surveys on feature selection algorithms include [8, 18, 42]. Not all types of feature selection algorithms are useful for all types of data. Classical feature selection algorithms were designed for data sets with small dimensionality, but the data sets generated by today's applications are large both in terms of dimensionality and size. There are specialized methods to deal with the problem of feature selection with large data sets. A glimpse of some recent feature selection methods meant for large data sets, typical in applications like text categorization, can be found in a recent special issue on feature selection in *Journal of Machine Learning Research* [67].

In this brief study, we discuss some of the main paradigms of feature selection. Our study does not cover all methods, but explores some of the prevalent paradigms. Also we do not attempt to present a strict taxonomy of feature selection algorithms. In this thesis we have used a special type of feature selection technique to build classifiers and function approximation systems, which we call *Online Feature Selection* (OFS) method. Before discussing existing OFS methods, for completeness, we review a few other prevalent methods for feature analysis in both classical and CI framework. We start with feature extraction algorithms which perform dimensionality reduction. The

two methods that we consider here are the Principal Component Analysis(PCA) and Sammon's non-linear projection method. Next we discuss classical feature selection methods under two different headings, namely, the evaluation criteria and the search techniques. This is followed by a discussion on some feature selection methods using the CI tools. Finally, we introduce OFS, we justify the need for online selection methods and discuss some existing methods in the family.

2.1.1 Feature Extraction

Feature selection through Orthogonal Transforms

There have been many studies on feature analysis using orthogonal transformation methods mainly through approaches like the principle component analysis (PCA) [45, 64, 93]. PCA is a linear orthogonal transform from s-dimensional space to q-dimensional space ($q \leq s$), such that the features of the data in the new q-dimension are uncorrelated and maximal amount of variance of the original data is preserved only by a small number of features.

Let us consider a data set $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N\}$, where $\boldsymbol{x}_i = (x_{i1}, x_{i1}, ..., x_{is})^T \in \Re^s$. Now, the objective is to find a suitable transform which maps each $\boldsymbol{x}_i$ from a s dimensional space to a q dimensional space where $q < s$. In the transformed space, a pattern is represented by its projection

$$\boldsymbol{x}'_i = \boldsymbol{W}^T \boldsymbol{x}_i, \quad i = 1, 2, ..., N \tag{2.1}$$

where $\boldsymbol{W} = (\boldsymbol{w}_1, \boldsymbol{w}_2, ..., \boldsymbol{w}_q) \in \Re^{s \times q}$ is an orthogonal matrix. To find the transform matrix $\boldsymbol{W}$, PCA begins with the total scatter matrix $\boldsymbol{S}_x$, which is defined as

$$\boldsymbol{S}_x = \sum_{i=1}^{N} (\boldsymbol{x}_i - \boldsymbol{m}_x)(\boldsymbol{x}_i - \boldsymbol{m}_x)^T \tag{2.2}$$

where $\boldsymbol{m}_x$ is the sample mean in the original input space, i.e.,

$$\boldsymbol{m}_x = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{x}_i. \tag{2.3}$$

The scatter matrix in the transformed space is thus

$$\boldsymbol{S}_y = \sum_{i=1}^{N} (\boldsymbol{x}'_i - \boldsymbol{m}_y)(\boldsymbol{x}'_i - \boldsymbol{m}_y)^T$$

$$= \sum_{i=1}^{N} \boldsymbol{W}^T(\boldsymbol{x}_i - \boldsymbol{m}_x)(\boldsymbol{x}_i - \boldsymbol{m}_x)^T\boldsymbol{W}$$
$$= \boldsymbol{W}^T\boldsymbol{S}_x\boldsymbol{W} \tag{2.4}$$

where $\boldsymbol{m}_y$ is the sample mean in the transformed space. The orthogonal matrix is chosen to maximize the determinant of the scatter matrix of the projected samples. Suppose $\boldsymbol{W}^* = [\boldsymbol{w}_1^*, \boldsymbol{w}_2^*, ..., \boldsymbol{w}_q^*]$ maximizes $|S_y|$. Then $\{\boldsymbol{w}_1^*, \boldsymbol{w}_2^*, ..., \boldsymbol{w}_q^*\}$ is the set of eigenvectors corresponding to the q largest eigenvalues of $\boldsymbol{S}_x$.

For a fixed number of input vectors, PCA can be performed in a batch mode. It involves computation of the matrix $\boldsymbol{S}_x$ and then eigenvalue decomposition of $\boldsymbol{S}_x$. Many approaches are available for eigenvalue decomposition [64, 178]. The conventional approaches of computing the matrix $\boldsymbol{W}$ are computationally expensive, especially when s is very large. Also, the batch mode cannot be applied when the input vectors form an infinite sequence. This problem can be addressed through adaptive methods like neural networks for PCA [45, 158, 201, 202].

PCA networks are able to realize only linear input-output mappings and they cannot usually separate independent subsignals from their linear mixture. To overcome these drawbacks PCA networks containing non-linear units have also been proposed [99]. In another form of PCA, called kernel PCA [211], the computations are done in a space different from the input space. In kernel PCA, input vectors are projected to a new space which is nonlinearly related to the input space, and the computations are performed in this new space.

The main drawback of the PCA and related methods is that the transformed features do not bear the true meanings of the original feature set. Thus PCA and related techniques are feature *extraction* techniques not feature selection techniques. Mao in [148] showed that the Gram-Schmidt (GS) orthogonal transform can be used for feature selection. Unlike the PCA, the transformed features in the GS space can be linked back to the original set of features.

PCA and related techniques are by nature unsupervised, i.e., they operate on the input vectors and do not consider the class labels (for classification) or the output vectors (for function approximation). On the other hand, the linear discriminant analysis (LDA) is a supervised technique which takes into account the outputs. In this case the transformed features are derived to maximize the separation between class means

relative to the covariances of the classes [48].

The PCA, LDA and related techniques do not consider the specific classifier or the induction algorithm which will use the features. Thus these methods can be classified as filter techniques, which filter out irrelevant information from the data before they are used for further processing.

Sammon's Nonlinear Projection Method

Sammon [209] proposed a simple yet very useful nonlinear projection algorithm that attempts to preserve the structure of the input space ($X \subset \Re^s$) in the transformed space ($X' \subset \Re^q$) by finding N points in q-dimensional space such that inter-point distances in the q-dimensional space approximate the corresponding inter-point distances in the original s-dimensional space.

Let $X = \{\boldsymbol{x}_k | \boldsymbol{x}_k = (x_{k1}, x_{k2}, ..., x_{ks})^T, k = 1, 2, ..., N\}$ be the set of N input vectors and let $X' = \{\boldsymbol{x}'_k | \boldsymbol{x}'_k = (x'_{k1}, x'_{k2}, ..., x'_{kq})^T, k = 1, 2, ..., N\}$ be the unknown vectors to be found.

Let $d^*_{ij} = d(\boldsymbol{x}_i, \boldsymbol{x}_j)$, $\boldsymbol{x}_i, \boldsymbol{x}_j \in X$ and $d_{ij} = d(\boldsymbol{x}'_i, \boldsymbol{x}'_j)$, $\boldsymbol{x}'_i, \boldsymbol{x}'_j \in X'$, where $d(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is the Euclidean distance between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. Sammon suggested looking for X' minimizing the error function E, where

$$E = \frac{1}{\sum_{i<j} d^*_{ij}} \sum_{i<j} \frac{(d^*_{ij} - d_{ij})^2}{d^*_{ij}} \quad . \tag{2.5}$$

Minimization of E is an unconstrained optimization problem in the Nq variables $x'_{ij}, i = 1, 2, ..., N;\ j = 1, 2, ..., q$. Sammon used the method of steepest descent for (approximate) minimization of E. Let $\boldsymbol{x}'_i(n)$ be the estimate of $\boldsymbol{x}'_i$ at the n-th iteration. Then $\boldsymbol{x}'_i(n+1)$ is given by

$$x'_{ij}(n+1) = x'_{ij}(n) - \alpha \left[\frac{\frac{\partial E(n)}{\partial x'_{ij}(n)}}{\left| \frac{\partial^2 E(n)}{\partial x'_{ij}(n)^2} \right|} \right] \tag{2.6}$$

where α is a non-negative scalar constant called the step size for gradient search.

With this method one cannot get an explicit mapping function governing the relationship between patterns in X and corresponding patterns in X'. Therefore, it is not

possible to project new points. Hence, with every additional point, it is necessary to redo the optimization with all data points. Every step within an iteration requires computation of $\frac{N(N-1)}{2}$ distances and for large N the computation time becomes high. Finally, there are many local minima on the error surface and it is usually unavoidable for the algorithm to get stuck in some local minimum. When N is large, getting a good solution may be difficult and one may need to try several initializations.

A number of neural networks have been proposed for feature extraction and multivariate data projection [149, 169] which augment Sammon's algorithm with prediction capability. Another related work in [170] describes a fuzzy system which can do Sammon's projection. The utility of these methods lies in he fact that they can predict projections of a new data point which is different from the set of points used for training the system. Again, the projected features found by the Sammon's algorithm or its connectionist or other relatives are not the original features. An attempt to do feature selection following Sammon's algorithm can be found in [33].

2.1.2 Feature Selection

The methods discussed so far were feature extraction methods, which used two different criteria for projecting data points to a lower dimensional space. Now we shall concentrate on feature selection algorithms. As stated earlier, a feature selection algorithm is characterized by two attributes: the feature evaluation criteria and the search technique employed. Here we review some of the important feature evaluation criteria and search techniques

Feature Evaluation Criteria

An optimal subset is always relative to certain evaluation function (i.e., an optimal subset chosen using one evaluation function may not be the same as that which uses another evaluation function). Typically, an evaluation function tries to measure the ability of a feature or a feature subset to distinguish between different classes. Some of the evaluation criteria are: distance, information (or uncertainty), dependence and classifier error rate. We next discuss in brief these categories of evaluation functions. A more detailed survey regarding evaluation functions can be found in [8, 42].

Distance Measures

Distance measures are used to compute measures of separation between pattern classes. Since a class can be characterized by a probability distribution, a distance measure can be a measure of (dis)similarity between two probability distributions. Such measures are also known as separability, divergence, or discrimination measure. For a two-class problem, a feature x_1 is preferred to another feature x_2, if x_1 induces a greater difference between the two-class conditional probabilities than x_2; if the difference is zero, then x_1 and x_2 are indistinguishable.

There can be many types of distance measures. A popular measure of class separability is defined as $S = trace(S_b^{-1}S_w)$ [48], where, S_w and S_b are the within class and between class scatter matrices respectively. They are defined as:

$$S_w = \sum_{j=1}^{c} \pi_j E\{(\boldsymbol{x} - \boldsymbol{m}_j)(\boldsymbol{x} - \boldsymbol{m}_j)^T | \omega_j\} \tag{2.7}$$

$$S_b = \sum_{j=1}^{c} (\boldsymbol{m}_j - \bar{\boldsymbol{x}})(\boldsymbol{m}_j - \bar{\boldsymbol{x}})^T \tag{2.8}$$

where

$$\bar{\boldsymbol{x}} = E\{\boldsymbol{x}\} = \sum_{j=1}^{c} \pi_j \boldsymbol{m}_j \tag{2.9}$$

Here, c is the number of classes, π_j is the prior probability that a pattern belongs to class ω_j, $\boldsymbol{x}$ is the feature vector; and $\boldsymbol{m}_j$ is the sample mean vector of points in class ω_j. $\bar{\boldsymbol{x}}$ is the mean of the entire data set. A lower value of S ensures that the classes are well separated.

There are also other distance measures like the Mahalanobis distance, Bhattacharya Distance, Divergence which can be defined for two class classification problems [44]. For using these distances one needs to know the class conditional probability density functions or one have to compute a reasonable estimate of them.

Information Measures

These measures typically determine the information gain obtained from a feature. The information gain from a feature x is defined as the difference between the prior uncertainty and expected posterior uncertainty using x. Feature x_1 is preferred to feature x_2 if the information gain from feature x_1 is greater than that from feature x_2. Entropy,

mutual information etc. are examples of such measures. If the prior probabilities associated with c classes $\omega_1, \omega_2, ...\omega_c$ be $\pi_1, \pi_2, ..., \pi_c$, then the initial uncertainty about a class can be measured by the Shannon's entropy as

$$H(C) = -\sum_{i=1}^{c} \pi_i log(\pi_i) \tag{2.10}$$

The average uncertainty after knowing the data (the input vectors) X is the conditional entropy $H(C|X)$. The amount by which the uncertainty about the class decreases due to the knowledge of the data is called the mutual information $I(C;X)$, where

$$I(C;X) = H(C) - H(C|X). \tag{2.11}$$

Suppose the data initially consist of s features, so, $X \subset \Re^s$. Let $F = \{f_1, f_2, ..., f_s\}$ denote the set of features. Let $S \subset F$, and we denote the input vectors in X which contains only the features contained in S by X_S. Similarly X_{f_i} will denote the data set with only the feature f_i. In light of equation (2.11), the problem of feature selection can be formulated as: given an initial set $F = \{f_1, f_2, ..., f_s\}$ of s features, find the subset $S \subset F$ with q features that minimizes $H(C|X_S)$. Minimization of $H(C|X_S)$ is equivalent to maximization of the mutual information $I(C;X_S)$, as the class uncertainty $H(C)$ is fixed, and determined by the prior probabilities of the classes. A variety of methods using this evaluation criteria are described in [8]. We discuss two recent methods next.

Battiti [6] presents a method for feature selection using the mutual information criteria. Let $F = \{f_1, f_2, ..., f_s\}$ be the original set of features and the aim is to select $q < s$ features. Let S denote the set of selected features. The algorithm starts with an empty S. In each step one feature gets added to S. The first feature f is chosen so that $I(C;X_f)$ is maximized. X_f denotes the data set with only feature f included. For selection of a feature f in a later step which is not already selected, that f is considered which maximize $I(C;X_f) - \beta \sum_{s \in S} I(X_f;X_s)$. This is continued till $|S| = q$. Battiti considers the mutual information of a feature with the class and also the mutual information of that feature with the already selected features. Thus, features which have a high mutual information with the already selected features are penalized. The parameter β regulates the relative importance of the mutual information between the candidate feature and the already selected features with respect to the mutual information with the output class.

Kwak and Choi [127] provides a modification of Batitti's method [6]. They show that the feature selection criterion used in [6] does not work well for for non-linear problems. In their method too, they start with an empty S. They select the first feature f which maximizes $I(C; X_f)$ and in the later steps, they chose a feature f_i maximizing $I(C; X_{f_i}, X_S)$. $I(C; X_{f_i}, X_S)$ is the mutual information between the feature f_i plus the already selected features in S and the output classes in C.

Dependence Measures

The dependence of an input feature with the output can be used as a criteria to rank individual features. The correlation coefficient is a classical dependence measure [68, 42]. Let $Cor(x, y)$ denote the correlation coefficient of the feature x with the output y. If $Cor(x_1, y) > Cor(x_2, y)$, then feature x_1 is considered more suitable than x_2. This is suitable as a feature *ranking* criteria if the targets y are continuous valued. But, can also be used for two class classification problems where each class label is mapped to a given value of y, say ± 1. The correlation criteria can only detect linear dependencies between a feature and a target. The correlation coefficient can also be used to measure the dependence of a feature with other features. Dependency of a feature with the other features can help to quantify the degree of redundancy of the features [42].

Error Rate Measures

Error rate measures consider the performance of a classifier or function approximation system to evaluate features. The methods using this type of evaluation function are called wrapper methods As the features are selected using the classifier/function approximation system that later on uses these selected features in predicting the class labels/outputs for unseen instances, the quality of features obtained by such criteria is very high. But the computational costs of such methods are also more.

Search Techniques

In feature selection, once the evaluation criterion for feature subsets is fixed, it reduces to a search problem. One has to search for the subset that optimizes the feature selection criterion. There are a variety of search techniques for feature selection. Among them a family of methods known as the *sequential* methods is very popular. These methods dynamically change the number of features in the set of relevant features in

each step. These methods can be divided into two categories: the bottom up method, which starts with an empty set and adds features to it and the top down method, where it starts with all features and discards features as it proceeds. As these algorithms do not consider all subsets, the optimal subset may not be obtained by them.

Sequential Forward Selection (SFS) is a bottom up search procedure where one feature at a time is added to the current feature set until the required number of features are obtained. At each stage, the feature to be included in the feature set is selected from among the remaining available features so that the new enlarged set of features yields an optimum value of the criterion function used. The *Sequential Backward Selection* (SBS) is a top down counterpart of the SFS method. Here, starting from the complete set of features, features are discarded sequentially until the desired number of features is obtained. Both these methods are suboptimal and suffer from the so called "nesting effect". It means that in case of SFS a feature once selected cannot be discarded even if at a later stage this feature becomes redundant due to the inclusion of some other feature. Similarly, for SBS a discarded feature cannot be reselected. This problem renders the sequential methods suboptimal. Details of the SFS, SBS and other related methods can be found in [44]. In [116] Kittler provides a comparative study of these algorithms.

Pudil *et al.* in [191] introduce the *sequential floating selection* methods which are modifications over the SFS and SBS algorithms. The *Sequential Floating Forward Selection* (SFFS) method is a bottom up method which includes new features following the SFS strategy, but it is also equipped to discard features from the set of selected features. Similarly, its top down counterpart (Sequential Floating Backward Selection) can include features which it has discarded in a previous step. This modification helps to get rid of the nesting effect which occurs with SFS and SBS. The floating search methods also do not consider all feature subsets hence they also cannot yield optimal feature sets.

Narendra and Fukunaga in [152] describe the *branch and bound algorithm* to find an optimal set of features faster than an exhaustive search. This method is a top down search but with a backtracking facility which allows all possible combination of features to be examined. The computational efficiency of the process lies in an effective organization of the search method. By virtue of this organization detailed evaluation of many candidate feature sets can be avoided without undermining the optimality of the

feature selection procedure. But, this method works on an monotonicity assumption on the feature selection criteria. In other words, if $S_1, S_2, ..., S_k$ be nested feature subsets such that

$$S_1 \supset S_2 \supset ... \supset S_k,$$

then the criterion function J must satisfy

$$J(S_1) \geq J(S_2) \geq ... \geq J(S_k).$$

By a straightforward application of this property, many combination of features can be rejected from the set of candidate feature sets. The required monotonicity property often restricts the use of the branch and bound method for many practical applications. Some of the widely used criterion functions like the Bhattacharyya distance, Divergence follow the monotonicity property [44]. But the classification error rate which is considered the best evaluation criteria for feature selection is not monotonic. It is known that for a finite sample size, with increase in the number of features, the classification accuracy of a classifier initially increases, but it starts to decrease after a certain point [86]. In [71], Hamamoto *et al.* present some experiments on feature selection for classification problems using the branch and bound technique [152]. They use classification error rate as the criteria for feature evaluation. Experiments in [71] show that the branch and bound algorithm can perform equivalent to an exhaustive search even if classification error rate is used as the evaluation criteria. Foroutan and Sklansky in [55] introduce a concept of *approximate monotonicity*. They propose a methodology for classification using piecewise linear functions. They state four conditions under which the classification error rates produced by a piecewise linear classifier would be monotonic. They also observe that in most real life cases, a piecewise linear classifier would not follow all the four conditions. Thus, they term the error rates produced by piecewise liner classifiers as approximately monotonic or mildly monotonic. They further demonstrate that classification error rates of a trained piecewise linear classifier can be used as a feature evaluation criteria in a modified branch and bound technique to arrive at an optimal set of features. In [91] Jain *et al.* provide a comparative study of feature selection algorithms which employ different search techniques.

The search techniques described till now are all deterministic techniques, but there are also probabilistic search techniques, which result in a optimal feature set with a certain probability. The Las Vegas filter and Las Vegas wrapper [143] are examples

of the probabilistic feature selection techniques. These are iterative techniques. In each iteration a feature subset S is sampled without replacement at random from the possible 2^s subsets and S is evaluated by certain evaluation criteria. In each round, the best set obtained is retained.

Genetic algorithms (GA) and Simulated annealing are also powerful stochastic search techniques. They have also been applied for the problem of feature selection. We consider GA based feature selection in the next section.

2.1.3 Feature selection by Computational Intelligence Tools

The methodologies that we discussed so far are mainly statistical in nature. They employ statistical measures for evaluating features and apply classical search techniques for generating the feature subsets. Now we shall discuss some methods of feature selection which are performed using three main tools of computational intelligence, i.e., genetic algorithms, neural networks and fuzzy logic.

Feature Selection by Genetic algorithms

Genetic algorithm (GA) has established itself as a reliable and efficient tool for solving various types of optimization problems. The problem of dimensionality reduction can be posed as an optimization problem. Given a set of s-dimensional input patterns, the task of the GA is to find a transformed set of patterns in a q-dimensional space ($q < s$) that optimizes a criterion. Typically, the transformed patterns are evaluated based upon their dimensionality, and either class separation or classification accuracy in the reduced dimension.

Use of GAs for feature selection was first introduced by Siedlecki and Sklansky [218]. In their work, they use GA to find an optimal binary vector, where each bit is associated with a feature. If the i^{th} bit of this vector is 1, then the i^{th} feature is allowed to participate in the classification; if the bit is 0, then the corresponding feature does not participate. Each resulting subset of features is evaluated according to its classification accuracy on a set of test data points using a nearest neighbor classifier [12]. This technique was later expanded to allow linear feature extraction by Punch *et al.* [192] and independently by Kelly and Davis [114]. In these methods the single bit used for

a feature was expanded to real valued coefficients allowing independent linear scaling of each feature, while maintaining the ability to remove features by assigning a weight of zero. Raymer *et al.* [195] use GAs to optimize a vector of feature weights along with a binary mask which decides the feature to be included. Additionally they encode the classifier parameter in the chromosome. For a k-nn classifier they encode k in the chromosome and optimize its value along with the feature weights. As these methods consider the classification error rate for the evaluation of a feature set, they are wrapper approaches. Other methods of feature selection using GAs can be found in [22, 65].

Neural Network Based Feature Selection

There are of methods which use neural networks for dimensionality reduction. We have already mentioned about neural networks for feature extraction and data projection which implement variants of PCA and Sammon's algorithm. Here we discuss certain neural networks which do feature selection following some other criteria.

There are a variety of methods for feature selection which is based on pruning of redundant nodes and links in a neural network. Pruning input nodes of a neural network is equivalent to removal of a feature. In [196], Reed provides a survey of neural network pruning algorithms. Some of the methods discussed in [196] can be used for feature selection. Pruning based methods start with all features, and subsequently the input nodes gets pruned to obtain a network which learns with a reduced number of features [215, 221].

Setiono and Liu in [215] used a three-layered feedforward neural network to select important features. They used a cost function involving the cross entropy and an additional penalty term on the weights of the neural network. From a trained network, the values of the weights connecting the input layer and the hidden layer are used to decide suitability of features and features are removed accordingly. The penalty function is so designed that the weights between the input and hidden layers, which bear low magnitudes are forced to take near zero values. Thus, if all links from a input node bear near zero values, then the feature associated with that node can be removed. They start with all features and features are removed based on the performance of the network. After removal of a feature the network is retrained , and the selection process is repeated until no feature meets the criterion for removal. This method requires

training of multiple networks and is thus computationally expensive. In another work, Steppe *et al.* [221] use a likelihood ratio test statistics to prune a trained neural network in a sequential fashion to arrive at the "best" model and also the best set of features. They sequentially remove input nodes in a neural network and test the null hypothesis that the reduced model is equivalent to the full model. The hypothesis is accepted on the basis of a likelihood ratio test statistic.

Ruck *et al.* [203] were the first to propose use of sensitivity of output of a network to its input for feature ranking. They used a multilayer perceptron architecture. They defined a feature saliancy criteria as

$$\Lambda_j = \sum_{\boldsymbol{x} \in T} \sum_{k} \sum_{x_j \in D_j} \left| \frac{\partial o_k}{\partial x_j} \right|, \tag{2.12}$$

where D_j is the set of values for the j^{th} feature that will be sampled and and o_k is the output of the k^{th} output node. T is the training set. Therefore, $\Lambda_j > \Lambda_i$ is assumed to indicate that the importance of the j^{th} feature is more than that of the i^{th} feature.

Zurada *et al.* in [254] compute a sensitivity matrix for a trained MLP using the partial derivatives of the outputs with respect to the inputs. The mean square average sensitivity matrix for the network is then computed using all training data. The values of the average sensitivity matrix are then used to prune redundant features. Engelbrecht in [54] too uses the partial derivatives of the outputs with respect to the input to define the sensitivity of each feature, and uses these values for network pruning. Yeung and Sun in [248] use the variance of the output error with respect to perturbation in input to define a sensitivity measure. Zeng and Yeung in [251] use the expected value of the output error with respect to the input change.

In [43] a different feature quality index (FQI) is proposed. Using a trained MLP, for each feature r they compute FQI_r. For each training data point $\boldsymbol{x}_i$, the r-th component is set to zero. If $\boldsymbol{x}_i^{(r)}$ denotes the modified data point, then except for the r-th component the other components of $\boldsymbol{x}_i^{(r)}$ are the same as $\boldsymbol{x}_i$. Let $\boldsymbol{o}_i$ and $\boldsymbol{o}_i^{(r)}$ denote the output vectors obtained from the neural network with inputs $\boldsymbol{x}_i$ and $\boldsymbol{x}_i^{(r)}$, respectively. If the r-th feature is not important, then the difference between $\boldsymbol{o}_i$ and $\boldsymbol{o}_i^{(r)}$ should be small. Therefore, the FQI_r is defined as

$$FQI_r = \frac{1}{N} \sum_{i=1}^{N} ||\boldsymbol{o}_i - \boldsymbol{o}_i^{(r)}||, \tag{2.13}$$

where N is the size of the training sample. The features can be ranked according to importance as $r_1 \geq r_2 \geq ... \geq r_s$ if $FQI_{r_1} \geq FQI_{r_2} \geq ... \geq FQI_{r_s}$.

In [7], Belue and Bauer propose a method which introduces noise as an extra feature in a neural network. Then the features are ranked from least significant to most significant by comparing the saliency of each of the true features with that of the noise feature. Grandvalet in [66] discusses another method of anisotropic noise injection for determination of relevance of inputs.

MacKay [145, 146, 147] considers neural network learning in a Bayesian framework. MacKay and Neal propose a feature selection mechanism in the Bayesian learning framework called automatic relevance detection (ARD) [154]. In ARD model, each input variable is associated with a hyperparameter that controls the magnitude of the weights on connections out of that input unit. The significance of an input variable is determined according to the posterior distributions of these hyperparameters.

Fuzzy Set Theoretic and Neuro-Fuzzy Methods

Fuzzy set theory has also been used for feature selection. In [171] the indices of fuzziness, entropy etc. are used to define an index of feature evaluation based on inter and intra class distances. In [131] fuzzy entropy is employed to evaluate the quality of features. Rezaee *et al.* in [197] discuss a feature selection strategy for fuzzy valued features using the sequential and exhaustive search methods. In [229], Tsang *et al.* discuss the complexity issues of feature selection algorithms. They prove that optimal fuzzy valued feature subset selection (OFSS) is NP-hard, they also provide a heuristic method for OFSS.

There are also a few feature selection strategies meant for fuzzy systems. In [141] Lin *et al.* use fuzzy curves and fuzzy surfaces to determine relevant inputs. Linkens and Chen [142] use a correlation based input selection for fuzzy models. Gaweda *et al.* [62] propose an input selection method by sensitivity analysis of a Takagi Sugeno fuzzy model.

Neuro-fuzzy paradigm has also been used for feature selection [108, 110, 125, 126, 172]. In [125, 126] Krishnapuram and Lee developed a neural network for classification which uses fuzzy aggregation functions as activation functions. On completion of training,

the redundant links can be identified and removed from the net. If all links emanating from an input node are removed, then the corresponding feature is redundant and hence eliminated. Similar type of networks are also discussed in [108, 110, 123, 124].

Feature Selection Based on Support Vector Machines

Several attempts are made to address the problem of feature selection using SVMs. As discussed in Chapter 1 the support vector machine looks for an optimal hyperplane in a high dimensional space [259, 232, 262]. Suppose we have the training data $\{(\boldsymbol{x}_k, y_k) \in \Re^s \times \{-1, 1\}, k = 1, 2, ..., N\}$ where $\boldsymbol{x}_k$ is a training example and y_k is its class label. The method consists of computing a decision function of the form:

$$f(\boldsymbol{x}) = \boldsymbol{w}^T.\boldsymbol{x} + b,$$

by maximizing the distance between the set of points $\{\boldsymbol{x}_k, k = 1, 2, ..., N\}$ and the hyperplane parameterized by $(\boldsymbol{w}, b)$ while being consistent with the training set. Here $\boldsymbol{w} \in \Re^s$ and $b \in \Re$. The class label of $\boldsymbol{x}$ is obtained by considering the sign of $f(\boldsymbol{x})$. As the decision function is a hyperplane, it cannot classify data sets with non linear class boundaries. In such cases the input points are implicitly projected to a high dimensional space where the hyperplane is constructed. An explicit projection of the training examples in the new space is not required but computation of the dot products of the training vectors in the transformed space is enough to compute the decision function. The dot products can be computed using a class of special type of functions called Mercer Kernels [259, 260, 262].

The SVM Recursive feature elimination (SVM-RFE) algorithm proposed by Guyon et al. [274] aims to find a subset of features of size r among d features which maximizes the performance of the SVM predictor. The method is based on sequential backward selection. It starts with all the features and removes chunks of features at each step until r features are left. For the linear case, in each step those features are removed whose removal minimizes the change in $||\boldsymbol{w}||^2$. This method is similar to those employed in neural networks [254] in the sense that the removal criteria is the sensitivity of $||\boldsymbol{w}||^2$ with respect to a feature. The method in [274] can also be modified to do feature selection in the nonlinear case where the feature removal criteria is based on the change of the cost function which is being minimized. For the linear case, the cost function

and the norm of the weight vector $||\boldsymbol{w}||^2$ differs only by a constant. Note that, for such methods, the set of selected features may change with the kernel function that is being used.

SVMs are provided with many statistics that allow to estimate their generalization performance from bounds on the *leave-one-out* error L. One of the most common L error bounds for SVMs is the radius/margin bound [232]:

$$L \leq 4R^2 ||\boldsymbol{w}||^2$$

where R is the radius of the smallest sphere which contains all the mapped data in the new space. In many papers, the L error bound is used for model selection [282]. In [283] Weston et al. use the margin bound for feature selection using a gradient descent algorithm. Similar work is reported in [285] also.

Bi et al. [284] use a sparse support vector machine for dimensionality reduction. They construct a series of sparse linear (without kernels) SVMs to generate linear models that can generalize well and use a subset of the non-zero weighted variables found by the models to produce the final nonlinear model. In [281] Grandvalet and Canu introduce scaling factors in the input variables and the scaling factors are learnt along with the other parameters of the hyperplane.

We have so far discussed some of the main paradigms of feature selection. Of course, this is not an exhaustive list of feature selection techniques; there exist many more methods and many variations of those discussed above. A large part of this thesis deals with the problem of feature selection. As mentioned earlier, we deal with computational intelligence techniques, and the feature selection methods proposed here are realized through neural networks. The family of methods which we consider in this thesis is unique in the sense that they are integrated with the main learning algorithm. We call these methods as *online* methods for feature selection. Next we discuss online feature selection.

2.1.4 Online Feature Selection

All methods that we have considered so far are offline in nature, i.e., in these methods feature selection precedes the main learning algorithm. By online feature selection (OFS) we mean a feature selection method which is inherent in the learning algorithm.

It has been recognized that the suitability of a feature subset depends on the problem being solved and also on the tool that is being used to solve the problem [43, 118]. Hence it is desirable that a learning system has a feature selection component inherent in it. The methods that we develop in Chapters 3-5 of this thesis deal with this problem of developing learning systems which can simultaneously learn an input-output mapping and the importance of the features present in the data.

There are some methods in the literature where simultaneous feature selection is done along with the development of the classifier. Decision trees like ID3 [193], C4.5 [194], CART [20] etc. do feature selection along with the tree building process. Such methods are termed *embedded* methods [18]. Adding and removing features form the core of these methods; in addition, they also involve routines for combining features into richer descriptions.

The LINKON proposed by Bhattacharyya *et al.* [16] is also an OFS method. Here the feature selection problem is formulated as a linear programming problem, which builds a classifier and discards the irrelevant features simultaneously. LINKON is applicable only to two class problems. A hyperplane $\boldsymbol{w}^T\boldsymbol{x} + b = 0$ can classify linearly separable pattern classes. They aim to find a "sparse hyperplane" where most of the components of $\boldsymbol{w}$ are zero. Thus, they formulate a linear programming problem, where the cost function is taken as the sum of the absolute values of the components of $\boldsymbol{w}$. In their scheme a feature x_i gets automatically eliminated, if its corresponding w_i is nearly zero. Also, the cost function forces the vector $\boldsymbol{w}$ to have as many zero components as possible.

Mackay and Neal's automatic relevance detection scheme [154] discussed in Section 2.1.3 is also a related method, but it requires the knowledge of the prior distributions of the hyperparameters associated with each feature. Raymer's genetic algorithm based method for feature selection [195] also encodes the parameters of the classifier within the chromosome along with the feature weights. The classifier considered by Raymer is a k-nn classifier.

Perkins and Theiler [188] term their method as online. But their usage of the term "online" is different from our usage. They consider a scenario where all features are not known at a time but features arrive one at a time. The learner's task is to select a subset of features and construct a model at each time step which is as good as

possible given the features seen so far. They use a technique called *grafting* [187] to accomplish the task. Grafting too considers feature selection as an integral part of the learning process, it builds a model in an incremental iterative fashion. In each iteration, a fast gradient based heuristic is used to access which feature is most likely to improve the existing model, and that feature is then added to the model. The model is incrementally optimized through gradient descent.

Pal and Chintalapudi in [166] introduced a method based on attenuation functions to do feature selection in an MLP. They associated a function, called feature attenuator function, with each input node of an MLP. Every input feature is multiplied by the corresponding attenuator function prior to its entry into the network. The attenuator functions are designed to take values between 0 and 1. The parameters of the attenuator functions were learned by the error backpropagation (EBP) learning scheme. At the onset of training each attenuator function value is set to almost zero. After training, for a bad or indifferent feature, the attenuator function acquires a value close to 0 and for a good feature a value close to 1. This simple method can learn the importance of the features along with other parameters of the MLP. Thus it is an online method. This attenuation based feature selection scheme has been applied to several real life applications [85, 163].

The methods of feature selection proposed in this thesis (Chapters 3-5) are motivated by the work in [166]. For each of the neural/neuro-fuzzy architectures that we have considered, we assume that there is a feature gate associated with each input node of a network. These feature gates restrict the entry of bad features into the network but allow the good features to get in. These gates are implemented through functions, which we call as feature modulators or attenuators. There is a learnable parameter associated with each feature modulator. The value of the attenuation parameter decides whether the associated gate is closed, open or partially open. These parameters are learned along with other parameters of a network and by interpretation of the values of these parameters from a trained network we can comment about the importance of different features. Further, according to the importance of the features, the network developed can be pruned to obtain a neural/neuro-fuzzy network with lesser free parameters and such a network is expected to have better generalization capabilities.

Our methods are quite different from other methods in the literature in the sense that our method do learning and feature selection simultaneously. Though in decision trees

selection of features is done along with the creation of the tree, but decision trees normally use certain other feature evaluation criteria. Our methods are connectionist in nature and they perform gradient based search over the parameter space to find parameters which can approximate the unknown input-output relation along with importance of each feature. And the learning of all parameters including the parameters associated with the feature gates is done through the minimization of the error in prediction on the training data. Thus our methods are wrapper methods but they have some additional good characteristics. In a wrapper approach one has to build a different classifier (or other learning system) with each of the feature subset and evaluate each feature set according to the prediction accuracy of the classifier constructed with each feature subset. Where as our methods try to build the "optimal" system along with the "optimal" subset of the features.

2.2 Enhancing Generalization Ability

Generalization ability is probably the most important attribute of any system learned from data. A model learned from a finite training set $T = \{(\boldsymbol{x}_i, \boldsymbol{y}_i) : i = 1, ..., N\}$ should not only perform well on points in T but also for points which are not included in the training set. This property of a learning machine to produce good results for future data (data not included in the training set) is called generalization ability. It can be assumed that the training data are generated from a time invariant joint probability distribution $P(\boldsymbol{x}, \boldsymbol{y})$. Suppose the inputs and outputs are related by a (unknown) function f and using T we have arrived at a predictor (the learning machine) ϕ. Our goal would be to find such a ϕ that minimizes the risk functional

$$R(\phi) = \int \mathcal{L}(\phi(\boldsymbol{x}), \boldsymbol{y}) dP(\boldsymbol{x}, \boldsymbol{y}), \tag{2.14}$$

using the data set T. Here, $\mathcal{L}(.,.)$ measures the loss incurred for incorrect prediction. $\mathcal{L}$ can be the number of misclassifications in case the predictor is a classifier; similarly, $\mathcal{L}$ can be the sum of squared deviations for FA type problems. Now, $P(\boldsymbol{x}, \boldsymbol{y})$ which generates the training data is unknown, hence an approximation of eq. (2.14) can be obtained by replacing the integral by a sum on the available data as

$$R_{emp}(\phi) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(\phi(\boldsymbol{x}_i), \boldsymbol{y}_i). \tag{2.15}$$

This is called the empirical risk. A minimization of the empirical risk does not always mean a minimization of the true risk. If the function class from which the predictor ϕ is selected has too many degrees of freedom, then ϕ will tend to overfit the training data and will thus produce bad generalization.

The current literature provides a rich understanding of the process of generalization. There are numerous methods which are applied to avoid overfitting in learning systems. Particularly, for neural networks the gain in generalization ability is obtained through four ways: (1) early stopping (2)complexity control (3) expanding the training set, and (4) by ensemble methods. Next we discuss work related to each of this broad categories.

2.2.1 Early Stopping

It has been observed that typically the generalization error decreases in an early period of training, reaches a minimum and then increases as training goes on, but the training error monotonically decreases. Thus, it is considered better to stop training after an adequate time of training. This is called early stopping. Early stopping is achieved through the validation method. The available training set T is divided into three disjoint sets and they are named as the training set Tr, the validation set V and the test set Te. The network is trained using the points in Tr, and after each epoch the performance of the network is tested using the validation set V. Note that, the points in the validation set act as unseen points to the trained network. The stopping criteria is decided on the performance of the network on the validation set V. When the performance starts degrading, training is stopped. The test set Te is then used to judge the final performance of the network, after training is complete. Among many others, early stopping has been considered in detail in [3].

2.2.2 Complexity Control

When number of parameters in a network (the weights and biases) increases, the network may employ more freedom to learn the intricacies of the training data and thus it may end up in "memorizing" the training data. This leads to overfitting and consequently bad generalization. On the other hand, a network with too few parameters

will not have enough power to represent the the data accurately. Thus, the best generalization is obtained by trading off the training error and the network complexity.

One of the techniques to reach this tradeoff is to minimize a cost function composed of two terms: the ordinary training error, plus some measure of the network complexity. The statistical inference literature has several such schemes. These are based on different criteria like Akaike's Information Criteria [1], Minimum Description Length [5, 199], Bayesian Information Criteria [120, 223], Vapnik's guaranteed risk minimization [233], etc. Various other complexity measures for learning systems have also been proposed [2, 199, 233]. It appears that an inexact but useful measure for complexity of a network will be the number of free parameters the network possesses. Thus, there are many techniques to prune network parameters to arrive at a less complex network [196]. We discuss some of these methods next.

Let $\boldsymbol{W} = (w_1, w_2, ..., w_{n_w})^T$ be the vector containing all free parameters of a network $\mathcal{N}$ and $\boldsymbol{o}_i$ be the output of $\mathcal{N}$ for an input $\boldsymbol{x}_i$. Then the cost function E (the empirical risk) on the basis of which the network is trained is given by

$$E = \frac{1}{N} \sum_{i=1}^{N} ||\boldsymbol{o}_i - \boldsymbol{y}_i||^2. \tag{2.16}$$

Note that $\boldsymbol{o}_i$ and hence E is a function of the network parameters $\boldsymbol{W}$. Training $\mathcal{N}$ means finding $\boldsymbol{W}$ that minimizes E. This is an unconstrained optimization problem and the components of $\boldsymbol{W}$ can take any value. Hinton [81] proposed a method of modifying the cost function E by adding an extra penalty term involving the weights. The modified cost function is given by

$$E_m = \frac{1}{N} \sum_{i=1}^{N} ||\boldsymbol{o}_i - \boldsymbol{y}_i||^2 + P_1, \tag{2.17}$$

where, P_1 is a penalty term, defined as

$$P_1 = \frac{\epsilon}{2} ||\boldsymbol{W}||^2. \tag{2.18}$$

In eq. (2.18) ϵ is a constant known as the decay constant. The added penalty in the cost function involves the magnitudes of the weights in the network. Thus a minimization of E_m will penalize weights with high values, and the trained network will tend to have weights with low magnitudes. He called this method as *weight decay*.

There are some drawbacks in using the penalty term P_1. P_1 causes all weights to decay exponentially to zero [73] at the same rate, hence it does not encourage any particular weight to take near zero values which can be pruned after training. But low magnitude of weights is, of course, a desired feature of a trained network. A modified penalty function P_2 is suggested in [236]:

$$P_2 = \frac{\epsilon}{2} \sum_{i=1}^{n_w} \frac{w_i^2}{1 + w_i^2}. \tag{2.19}$$

Note that $\frac{w_i^2}{1+w_i^2}$ takes near zero values for very small values of w_i and values near one for large weights. Thus, this can be considered a measure of the number of non zero weights in the network. Training a network with this penalty term will tend to produce a network with small number of non-zero weights, and the weights with near zero magnitude can be pruned after training. Setiono in [213] uses a combination of both penalties P_1 and P_2 for pruning a network. A combination of both P_1 and P_2 serves two purposes: the network will tend to have the minimum number of non zero weights and the penalty P_1 will discourage the network to have any weight with high magnitude.

In the *optimal brain damage* method of le Cun *et al.* [41], once the network is trained, the importance of each weight w_i is determined by computing a saliency s_i defined as

$$s_i = \left(\frac{\partial^2 E}{\partial w_i^2} \right) \frac{w_i^2}{2}.$$

The weights with small values of saliency have very little influence on the error and are hence removed.

Kanjilal and Banerjee [98] proposed a method of pruning hidden nodes and links of a trained neural network using the singular value decomposition (SVD) technique [64]. They consider a trained network $\mathcal{N}$ with say r hidden nodes. For each of the N training data points they compute the r outputs of the hidden nodes, thereby generating a $N \times r$ matrix B. SVD is performed on B to obtain $B = U_B \Sigma_B V_B^T$. Where, U_B and V_B have orthogonal columns and Σ_B is a diagonal matrix. The number of dominant singular values in Σ_B (say $l \leq r$) indicates the number of hidden nodes which are to be retained. The specific hidden nodes which are to be retained are found by QR decomposition with column pivoting [64] transformation on V_B^T.

The *regularization* framework originally introduced by Tikhonov [226], also adds an extra penalty term to the empirical risk term, which generally is a smoothness constraint. The model is built through minimization of this modified risk function. Regularization, as applied to neural networks, has been extensively discussed in [63, 189, 190]. The popular techniques for complexity control in neural networks, like weight decay [81], weight sharing [155], pruning [196, 213], training with noise [14] have been shown to be variants of regularization [34].

2.2.3 Expanding the Training Set

The problem of bad generalization and overfitting also arise due to the fact that neural networks are trained with finite training samples. The available training data are reused in every epoch and as a result, the neural network "concentrates" more and more on these points and often results in a bad generalization. A probable solution to this problem would be to have an infinitely large training set which is seldom realizable in practice. Hence there are methods which aim to expand the training set with additional points to cope with the finite sample effect.

Generating additional data points can be best done if the joint input-output probability distribution of the data points is known. But, this is never known apriori. So the methods for generating points aim at estimating the probability distribution from data and then draw additional training sample using the estimated density.

In [82] authors add noise to the training set. They consider adding white noise independently to the input and output vectors to generate new training samples. The noisy samples are generated following a kernel density estimate of the training data. For function approximation problem, they consider an estimate of the joint distribution of the input and output vectors and for classification problem they consider the class conditional densities.

Karistinos and Pados [100] suggest another procedure for random expansion of a given training set. They propose a locally most entropic estimate of the true joint input-output probability density function and use it to generate new training samples. They argued that the method in [82] is an extreme special case of their method.

2.2.4 Ensemble Methods

Ensemble methods have gained much importance in the current days. An ensemble method creates multiple predictors for the same task and an ensemble of these predictors is used for the final prediction. A predictor for which a small change in input does not necessarily produce a small change in the output is called unstable. Unstable predictors have large variance and they produce bad generalizations. Aggregating the prediction of multiple unstable predictors helps to bring down the variance and consequently the ensemble produces better generalization.

Usually neural networks are unstable [19]. In the context of neural networks, an ensemble is a collection of a finite number of neural networks trained for the same task. Typically, a neural network ensemble is constructed in two steps, i.e., training a number of component neural networks and then their predictions are combined. For training component neural networks the most prevailing strategies are Bagging [19] and Boosting [59]. Bagging is based on bootstrap sampling [53]. It generates several bootstrap samples from the original training set and a component neural network is trained with each bootstrap sample. In boosting a series of component networks are trained and the training set of a network is decided by the performance of its preceding network. The training samples which are wrongly predicted by the preceding networks will play more important role in training the latter networks. There are also other approaches of training component neural networks. Hampshire and Waibel [72] utilize different objective functions to train different component neural networks. Cherkauer [35] trains component networks with different hidden nodes. Yao and Liu [246] consider all individuals in an evolved population of neural networks as component networks.

All these methods try to get component networks which perform more or less the same on the training data but have variabilities in their parameters. Generally, all the component networks generated by the different methods are aggregated to get the final prediction. In [252] a genetic algorithm based method is proposed to select suitable component networks which yield better generalization.

Once the component networks have been trained, the next issue of importance is how to aggregate their outputs. The most common methods of aggregation are simple or weighted average for function approximation and majority voting for classification problems [19]. There can be other methods for combining predictors too. For example,

Wolpert [238] uses learning systems to combine component predictions. Ueda [230] exploits optimal linear weights to combine component predictions.

2.2.5 Constraining the Learning

In Section 1.4.5 it was noted that one of the desirable properties of any system designed from data is its ability to say "don't know" for points which are not typical to the training sample. This property is related to generalization. It addresses the important question: what regions of the input space should a learning machine generalize in? Typically the training samples, both in case of classification and function approximation, are restricted to a small region in the input space. The learning machine has no information about the nature of the function beyond the region where the training points lie. A good system should generalize only on test points which lie in the region represented by the training points; for other points, the system should not produce any response. We call this as *strict generalization*. Multilayer perceptrons do not possess this property of strict generalization. They produce some output for any test point irrespective of its position with respect to the training set. Experiments show (please refer to Chapters 6 and 7) that the response produced by an MLP for points which lie far away from the boundary of its training samples are generally not reliable. In Chapters 6 and 7 we study this problem and provide some solutions by implicitly constraining the learning process. This problem has not been addressed adequately in the literature.

Chapter 3

Online Feature Selection and Function Approximation Type System Design in a Neuro-Fuzzy Paradigm[1]

3.1 Introduction

The success of any system designed from data depends on the quality of features that are used to build the system. But most methods of function approximation (FA) either ignore feature analysis or perform it in a separate phase offline from the main learning task. In Section 2.1.4 we defined online feature selection as a feature selection method integrated with the main learning task. Most feature selection methods available in literature are offline in nature (refer Section 2.1.4). The suitability of a feature set depends on the task at hand as well as the learning method. Hence, it is expected that a system identification scheme with a built in feature selection component will result in systems with better prediction accuracy. It is needless to mention that a design methodology equipped with the capability of discarding redundant or bad features will ultimately lead to a less complex system. And it will consequently have less run-time complexity, less hardware cost, more readability and increased generalization ability. In this chapter we propose a neuro-fuzzy system with online feature selection capability for fuzzy rule based FA.

Let $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N\} \subset \Re^s$ and $Y = \{\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_N\} \subset \Re^t$ and let there be an unknown function $\boldsymbol{S} : \Re^s \Rightarrow \Re^t$ such that $\boldsymbol{y}_k = \boldsymbol{S}(\boldsymbol{x}_k)\ \forall k = 1, ..., N$. In other words,

[1] The contents of this chapter have been published in [24, 164].

there is an unknown function $\boldsymbol{S}$ which transforms $\boldsymbol{x}$ to $\boldsymbol{y}$. The problem of FA is to find $\boldsymbol{S}$, given X and Y. Function approximation appears in various fields of science and engineering. And there are also various methods to solve this problem. There are statistical methods which give explicit solutions to it; also there are computational transforms which provide implicit solutions. Neural networks, fuzzy rule based systems etc. are such computational transforms. Fuzzy rule based systems can be built based on rules supplied by experts. Also, rules extracted from data using exploratory data analysis or other tools may be used to build fuzzy systems [36, 37, 38, 159, 167]. Thus, if one needs to model a system for which the explicit equations governing the input-output relation are not known, but experts' opinion is available in terms of imprecise rules, then one can use such rules to build a fuzzy model of that system. But, our objective here is a bit different. We are interested in building models from input-output data. Thus, we aim to extract rules governing the input-output relationship of a system using the input-output data pairs. We develop a neuro-fuzzy system to do this. The proposed system learns human interpretable fuzzy rules only from input-output data. Along with the input-output mapping, our system also learns the importance of the various features present in the data. After completion of training the system gets rid of the redundant/bad features and also prunes the network accordingly to arrive at an "optimal" structure of lower complexity.

In this chapter we first provide a brief introduction to neuro-fuzzy systems in Section 3.2, where we justify the use of neuro-fuzzy systems for FA, also we refer to some earlier work on neuro fuzzy systems. Next we discuss our methodology for building a neuro-fuzzy system with integrated feature analysis capability. In Section 3.3 we discuss the structure of our neuro-fuzzy system, in Section 3.4 we derive the learning rules, in Section 3.5 we discuss the pruning strategies, and in Section 3.7 we discuss some simulation results. This chapter is concluded in Section 3.8.

3.2 Neuro-fuzzy Systems: Motivation and Earlier Works

It is known that neural networks can act as universal approximators for a large class of non-linear functions, hence the choice of neural networks for FA is quite justified and has been proved to be successful [77]. Neural Networks are usually robust, possess parallelism and good generalizing capabilities but they usually do not have readability and

work as a black box. Neural networks are capable of learning input-output mapping through minimization of a suitable error function. The structure of the various networks enables them to learn the mapping in an easy manner. But, as such they are not capable of learning or representing "knowledge" *explicitly*, which a fuzzy system can do. Here we are making a clear distinction between identifying a mapping and extraction of readable or intelligible knowledge. The underlying relation in a system, which has been approximated by a neural network, cannot be easily understood from a trained network by any easy means but there are some specialized methods which aims to extract symbolic or other kinds of rules from a trained network [47, 214, 216, 224, 225, 227]. On the other hand, fuzzy rule-based systems which have also been used for FA are highly interpretable in terms of linguistic rules. As fuzzy if-then rules can be easily understood by human beings and often an initial rule-base can be provided by an expert, there is no problem of readability. However, fuzzy rule based systems, as such are not capable of learning. So to extract rules from data one has to depend on techniques like clustering or other tools of exploratory data analysis [36, 37, 38, 167], or an initial rule base is supplied by an expert, which is then tuned using data. Thus, judicious integrations of neural networks and fuzzy logic are expected to result in systems with merits of both paradigms. Several attempts have been made to integrate fuzzy systems and neural networks with a view to achieving systems which are interpretable, robust and have learning abilities [39, 103, 136, 137, 138, 151, 174, 175, 182, 184, 186].

The various neuro fuzzy unification schemes developed till date can be classified into three major groups :

- Neural Fuzzy Systems
- Fuzzy Neural Systems
- Co-operative Systems

Neural fuzzy systems are fuzzy systems implemented by neural networks [112, 113, 136, 137, 160, 176]. Fuzzy neural systems are neural networks, capable of handling fuzzy information [75, 138]. The inputs, outputs and weights of fuzzy neural networks could be fuzzy sets, often fuzzy numbers or membership values. The Co-operative systems are those, which use different paradigms (neuro and fuzzy) to solve various facets of the

same problem [176]. All these three paradigms taken together is known as neuro-fuzzy computing.

Lee and Lee [133] were the first to study the concept of fuzzy neurons. They introduced the theory of fuzzy sets to the conventional McCulloch and Pitts model and finally analyzed fuzzy neural networks based on principles of neural networks and the mechanism of fuzzy automata [97, 133]. Since then there has been a variety of work which has addressed the problem of hybridizing the two useful paradigms. The last few years have seen a tremendous advancement of this new paradigm to solve a wide range of problems [21, 137, 174, 183]. There have also been attempts to realize these systems through specialized hardware [244], which have enabled use of these systems in a variety of fielded applications including consumer products. A detailed list of references on neuro-fuzzy systems can be found in [183]. Here we give a glimpse of some of the recent important contributions in neuro-fuzzy paradigm; we are particularly interested in neural-fuzzy systems.

Two of the important works in neural fuzzy systems are Jang's adaptive-network based fuzzy inference systems (ANFIS) [92] and Lin and Lee's neural network based fuzzy logic control and decision system [136]. Lin and Lee's work [136] describes a multilayered feedforward connectionist model designed for fuzzy logic control and decision making. A hybrid two step learning scheme that combined self-organized (unsupervised) and supervised learning algorithms for selection of fuzzy rules and tuning of membership functions were developed. They used Kohonen's self-organizing feature map [119] for finding the centers of the membership functions. After selection of the rule set, i.e., when the network architecture is established, the second step of supervised learning begins. Some heuristic guidelines for rule reduction and combination were also provided. There are many variants of this work, like Lee *et al.* [132] proposed a neural network model for fuzzy inferencing. They developed an algorithm for adjusting (tuning) the membership functions of antecedent linguistic values of the rule set by error backpropagation (EBP), where the consequent parts were considered fixed. Li and Wu [134] proposed a neuro-fuzzy hierarchical system with if-then rules for pattern classification problem. A five layer network was also presented by Yao *et al.* [247], where the parameters of the net are identified using evolutionary programming and the tuned network is then pruned to extract a small set of rules. Pedrycz and Reformat [185] used both the gradient descent algorithm and genetic programing for

structure and parameter learning of a neuro-fuzzy system. Shann and Fu [217] presented a layered network for selection of rules. Initially, the network was constructed to contain all possible fuzzy rules. After training through error backpropagation (EBP), the redundant rules were deleted by a rule pruning process for obtaining a concise rule base. The architecture of Shann and Fu is similar to that of Lin and Lee in several respects. Pal and Pal [168] discussed some limitations of the scheme by Shann and Fu and provided a better rule tuning and pruning strategy. The network proposed by Kim and Kasabov [115] also consists of two phases, one of rule generation from data and a rule tuning phase by error backpropagation. Lin and Cunningham [140] developed a layered network for system identification. They used fuzzy curves for feature selection, but this phase was a part of preprocessing on the data before the data get into the network. Wu and Er [240] proposed a dynamic fuzzy neural network implementing Takagi-Sugeno-Kang fuzzy systems based on extended radial basis function network. Lin and Chung [135] developed a neuro-fuzzy combiner based on reinforcement learning for multiobjective control. Figureiredo and Gomide [56] proposed a neural fuzzy system which encodes the knowledge learned in the form of fuzzy if-then rules and processes data using fuzzy reasoning principles. After learning, linguistic rules can be easily extracted from the network.

In [94] Juang and Lin discussed a self-constructing neural fuzzy inference network (SONFIN) with online learning ability. The SONFIN is a modified Takagi-Sugeno-Kang type fuzzy rule-based model possessing neural network's learning ability. There are no rules initially in the SONFIN. They are created and adapted as the learning proceeds via simultaneous structure and parameter identification. In [95] RSONFIN is proposed which is a recurrent version of SONFIN. SONFIN and RSONFIN have been used in several applications like word boundary detection [239], speech segmentation [139], object tracking [49] and radar pulse compression [50]. A variety of other real life applications have also been developed using neuro-fuzzy systems.

None of the methods cited above considers the problem of feature analysis explicitly. Though it has been identified that feature analysis is an important phase for designing any system from data. Next, we propose a novel neuro-fuzzy architecture which has a feature selection component inherent in it.

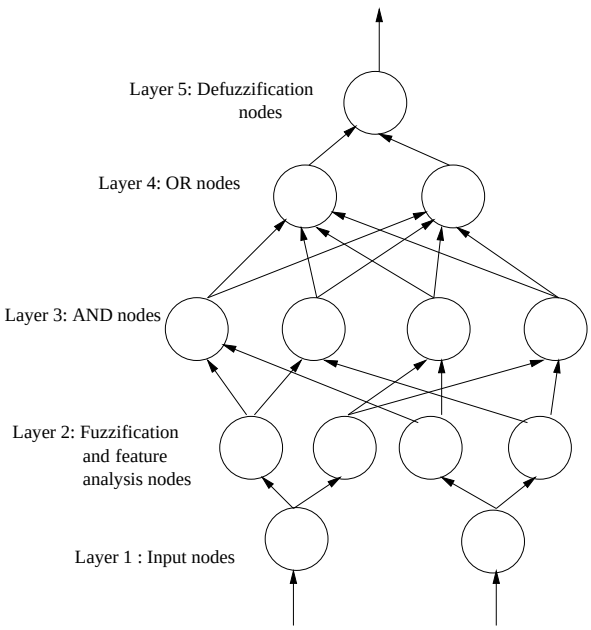


Figure 3.1: The network structure.

3.3 The Network Structure

For the FA problem that we consider here, let there be s input features $(x_1, x_2, ..., x_s)$ and t outputs $(y_1, y_2, ..., y_t)$. The proposed neural-fuzzy system will deal with fuzzy rules of the form, R_i : If x_1 is A_{1i} and x_2 is A_{2i} ..... and x_s is A_{si} then y_j is B_{ji}. Here A_{ji} is the i-th fuzzy set defined on the domain of x_j and B_{ji} is the i-th fuzzy set defined on the domain of y_j.

From our notation one might think that for each rule we are using a different set of antecedent linguistic values (fuzzy sets) but that is not necessarily true; in fact, for every feature only a few fuzzy sets are defined and hence some of the $A_{ij} = A_{ik}$ for some j and k. Similar is the case for the linguistic values defined on the output variables.

The neural-fuzzy system is realized using a five layered network as shown in Fig. 3.1. The node functions with its inputs and outputs, are discussed layer by layer. We use suffixes p, n, m, l, k to denote respectively the suffixes of the nodes in layers 1 through 5 in order. The output of each node is denoted by z. For example, z_p will indicate output from a node in layer 1, similarly, z_n will denote output from a node in layer 2 etc.

Layer 1: Each node in layer 1 represents an input linguistic variable of the network and is used as a buffer to transmit the input to the next layer, that is to the membership function nodes representing its linguistic values. Thus, the number of nodes in this layer is equal to the number of input features in the data. If x_p denotes the input to any node in layer 1 then the output of the node will be

$$z_p = x_p. \tag{3.1}$$

Layer 2: Each node in layer 2 represents the membership function of a linguistic value associated with an input linguistic variable. Moreover, this layer also does the feature analysis task. The output of these nodes lies in the interval [0,1] and represents the membership grades of the input with respect to different linguistic values. Therefore, the nodes in this layer act as fuzzifiers. The most commonly used membership functions are triangular, trapezoidal and bell shaped. Although any one of these choices may be used, we consider bell shaped membership functions. All connection weights between the nodes in layer 1 and layer 2 are unity. If there are N_i fuzzy sets associated with the i^{th} feature and if there are s input features then the number of nodes in this layer would be $N^2 = \sum_{i=1}^{s} N_i$. The output of a node in layer 2 is denoted by

$$\bar{z}_n = exp\left\{-\frac{(z_p - \mu_n)^2}{\sigma_n^2}\right\}. \tag{3.2}$$

In eq. (3.2) the subscript n denotes the n-th term (fuzzy set) of the linguistic variable x_p. μ_n and σ_n represent the mean and spread respectively of the bell shaped function representing a term of the linguistic variable x_p associated to node n. We deliberately put a - (bar) on z_n because, this is not the output of this layer. The actual output for this layer will be a modified value of $\bar{z}_n$.

For the purpose of feature selection, the output of this layer needs to be modified so that every indifferent/bad feature x_p gets eliminated. If a linguistic variable x_p is not important (or is indifferent) for describing the system behavior, i.e., for defining the input-output relation, then the values of x_p should not have any effect on the firing strength of the rules involving that input variable. This is our main guiding principle for feature analysis. Since for any $T - norm$, $\mathcal{T}(1, \alpha) = \alpha$, $0 \leq \alpha \leq 1$, this can be realized if an indifferent feature always generates a membership of unity or almost unity. This may appear impossible at the first sight. Note that for an indifferent feature, all of its terms (i.e., all of its linguistic values) should have no effect on the firing strength. Next we explain how this can be realized.

Let us associate a function f_n with each node n in layer 2. We call f_n a modulator function. For an indifferent (or bad) feature we want all linguistic values defined on that feature to result in a modulated membership value of almost 1. In other words, for a bad/redundant feature, we want to have $f_n.\bar{z}_n \approx 1$. To achieve this we model f_n as :

$$f_n = exp\left[\lambda_p \; ln\left(\frac{1}{\bar{z}_n}\right)\right]. \tag{3.3}$$

Here $\lambda_p \in [0,1]$ is a parameter associated with a particular *linguistic variable* x_p of which node n is a term. From eq. (3.3) we see that when λ_p is nearly 1 then f_n is nearly $\frac{1}{\bar{z}_n}$, and when λ_p is nearly 0 then f_n is nearly 1. So for bad features λ_p should get large values (close to 1) and small values (close to 0) for good features. Thus, for a bad feature, the modulated membership value would be $f_n.\bar{z}_n \approx \bar{z}_n.\frac{1}{\bar{z}_n} \approx 1$ irrespective of the value of x_p. Similarly, for a good feature, the modulated membership value would be $f_n.\bar{z}_n \approx 1.\bar{z}_n \approx \bar{z}_n \approx$ the actual membership value. Since λ_p must take values between zero and one, we model λ_p by $e^{-\beta_p^2}$. Thus, the activation function of any node n in layer 2 becomes :

$$z_n = \bar{z}_n \; exp\left[e^{-\beta_p^2} ln\left(\frac{1}{\bar{z}_n}\right)\right], \tag{3.4}$$

which can be simplified to

$$z_n = \bar{z}_n^{\left(1-e^{-\beta_p^2}\right)}. \tag{3.5}$$

In eq. (3.5) $\bar{z}_n$ is computed using eq. (3.2). The parameter β_p can be learnt by back-propagation technique. We see that when β_p^2 takes a large value then z_n tends to $\bar{z}_n$ and for small values of β_p^2, z_n tends to 1, thereby making the feature indifferent. Therefore, our objective would be to make β_p^2 take large values for good features and small values for bad ones through the process of learning. Layer 2 can be better realized using two layers of neurons, the first one for computation of the membership value, $\bar{z}_n$ and second layer for the modulated output using eq. (3.5).

Layer 3: This layer is called the AND layer. Each node in this layer represents an IF part of a fuzzy rule. There are many operators ($T-norms$) for fuzzy intersection [117]. Here we choose product as the operator for intersection. The number of nodes in this layer is $N^3 = \prod_{i=1}^{s} N_i$. The output of the m-th node in the layer is

$$z_m = \prod_{n \in P_m} z_n \tag{3.6}$$

where P_m is the set of indices of the nodes in layer 2 connected to node m of layer 3.

Layer 4: This is the OR layer and it represents the THEN part (i.e. the consequent) of the fuzzy rules. The operation performed by the nodes in this layer is to combine the fuzzy rules with the same consequent. The nodes in layers 3 and 4 are fully connected. Let w_{lm} be the connection weight between node m of layer 3 and node l of layer 4. The weight w_{lm} represents the certainty factor of a fuzzy rule, which comprises the AND node m in layer 3 as the IF part and the OR node l in layer 4 representing the THEN part. These weights are adjustable while learning the fuzzy rules. If there are M_i fuzzy sets associated with the i^{th} output variable and there are t outputs then the number of nodes in this layer is $N^4 = \sum_{i=1}^{t} M_i$. For simplicity let us assume that there is only one output variable and M linguistic values are defined on it. So the fourth layer has $N^4 = M$ nodes. For each output linguistic value there are exactly N^3 rules having that value as the consequent. Every node of this layer picks up only one rule from among the associated N^3 rules based on the maximum agreement with facts (in terms of the product of firing strength and certainty factor) for computation of the defuzzified output. When all certainty factors are equal, the rules are selected based on the maximum firing strength. This rule selection is viewed as an OR operation and realized by the max operator. Thus, like Shann and Fu [217] and Pal and Pal [168], the output of the node l in layer 4 is computed by

$$z_l = max_{m \in P_l}(z_m w_{lm}), \tag{3.7}$$

where P_l represents the set of indices of the nodes in layer 3 connected to the node l of layer 4. Since the learnable weights w_{lm}s are interpreted as certainty factors, each w_{lm} should be non-negative, preferably should lie in [0,1]. The EBP algorithm or any other gradient based search algorithm does not guarantee that w_{lm} will remain non-negative, even if we start the training with non-negative weights. Since the defuzzification uses a normalization scheme, we can ignore the constraint that $w_{lm} \in [0, 1]$. We model w_{lm} by g_{lm}^2. The g_{lm} is unrestricted in sign but the effective weight $w_{lm} = g_{lm}^2$ will always be non-negative. Therefore, the output (activation function) of the l^{th} node in layer 4 will be

$$z_l = max_{m \in P_l}(z_m g_{lm}^2). \tag{3.8}$$

Layer 5: This layer is the defuzzification layer. Each node of layer 5 represents an output linguistic variable and performs defuzzification, taking into consideration the

effects of all membership functions of the associated output linguistic variable. The number of nodes in this layer is equal to the number of outputs. Here we use a centroid type defuzzification scheme, and a node in this layer computes the output as :

$$z_k = \frac{\sum_{l \in P_k} z_l a_l c_l}{\sum_{l \in P_k} z_l a_l}. \tag{3.9}$$

In eq. (3.9) P_k is the set of indices of the nodes in layer 4 connected to node k in layer 5 and a_l, c_l are the spread and mean of the membership function representing node l in layer 4. The weights of the links connecting nodes in layer 4 and layer 5 are unity.

3.4 Learning of Feature Modulators and Rules

We now derive the learning rules for the neural-fuzzy system with the activation or node functions described in the previous section. In the training phase, the concept of backpropagation is used to minimize the error function

$$e = \frac{1}{2} \sum_{i=1}^{N} E_i = \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{t} (y_{ik} - z_{ik})^2, \tag{3.10}$$

where t is the number of nodes in layer 5 and y_{ik} and z_{ik} are the target and actual outputs of node k in layer 5 for input vector $\boldsymbol{x}_i$; $i = 1, 2, ..., N$. The method for adjusting the learnable weights in layer 4 and the parameters β_p in layer 2 are based on gradient descent search. We use online update scheme and hence derive the learning rules using the instantaneous error function E_i. Without loss we drop the subscript i in our subsequent discussions.

The delta value, δ, of a node in the network is defined as the influence of the node output with respect to E. The derivation of the delta values and the adjustment of the weights and the parameters β_p are presented layer wise next.

Layer 5: The output of the nodes in this layer is given by eq. (3.9) and δ values for this layer, δ_k, will be

$$\delta_k = \frac{\partial E}{\partial z_k}.$$

Thus,

$$\delta_k = -(y_k - z_k). \tag{3.11}$$

Layer 4: The delta for this layer would be

$$\delta_l = \frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial z_l}.$$

In other words,

$$\delta_l = \delta_k \frac{a_l(c_l - z_k)}{\sum_{l' \in P_k} z_{l'} a_{l'}}, \tag{3.12}$$

where k is a node in layer 5 with which node l in layer 4 is connected.

Layer 3: The delta for this layer would be

$$\delta_m = \frac{\partial E}{\partial z_m} = \frac{\partial E}{\partial z_l}\frac{\partial z_l}{\partial z_m}.$$

Hence, the value of δ_m will be

$$\delta_m = \begin{cases} \sum_{l \in Q_m} \delta_l \; g_{lm}^2 & \text{if } z_m \; g_{lm}^2 = \max_{m'}\{z_{m'} g_{lm'}^2\} \\ 0 & \text{otherwise.} \end{cases} \tag{3.13}$$

Here Q_m is the set of indices of the nodes in layer 4 connected with node m of layer 3.

Layer 2: Similarly, the δ_n for layer 2 would be

$$\delta_n = \frac{\partial E}{\partial z_n} = \frac{\partial E}{\partial z_m}\frac{\partial z_m}{\partial z_n}.$$

Hence,

$$\delta_n = \sum_{m \in R_n} \delta_m \left(\frac{z_m}{z_n}\right). \tag{3.14}$$

In eq. (3.14) R_n is the set of indices of nodes in layer 3 connected with node n in layer 2.

With the δ calculated for each layer now we can derive the weight updating equation and the equation for updating β_p.

$$\frac{\partial E}{g_{lm}} = \frac{\partial E}{\partial z_l}\frac{\partial z_l}{\partial g_{lm}},$$

or,

$$\frac{\partial E}{\partial g_{lm}} = \begin{cases} \sum_{l \in Q_m} 2 \; \delta_l \; z_m \; g_{lm} & \text{if } z_m \; g_{lm}^2 \; = \max_{m'}\{z_{m'} g_{lm'}^2\} \\ 0 & \text{otherwise.} \end{cases} \tag{3.15}$$

Similarly, we calculate

$$\frac{\partial E}{\partial \beta_p} = \frac{\partial E}{\partial z_n}\frac{\partial z_n}{\partial \beta_p}.$$

or

$$\frac{\partial E}{\partial \beta_p} = -\sum_{n \in R_p} \delta_n \left(2\ \beta_p\ e^{-\beta_p^2}\ z_n\right) \left(\frac{z_p - \mu_n}{\sigma_n}\right)^2. \tag{3.16}$$

Where, R_p is the set of indices of nodes in layer 2 connected to node p of layer 1. Hence, the update equations for weights and β_p are

$$g_{lm}(n+1) = g_{lm}(n) + \eta \left(-\frac{\partial E}{\partial g_{lm}}\right) \tag{3.17}$$

and

$$\beta_p(n+1) = \beta_p(n) + \nu \left(-\frac{\partial E}{\partial \beta_p}\right). \tag{3.18}$$

In eq. (3.17) and eq. (3.18) η and ν are learning coefficients.

The network learns the weights of the links connecting layers 3 and 4 and also the parameters associated with nodes in layer 2, which do the feature selection. The initial values of β's are so selected that no feature gets into the network in the beginning. This is realized by assigning very low positive values (say 0.0001) to each β_p. Thus in the beginning of learning every node in layer 2 produces a value which is nearly equal to one, and consequently, all features are considered unimportant. As learning proceeds, the values of βs gets updated in such a way that the important features, i.e., the features which can reduce the error rapidly, only pass through the network. This happens because, gradient descent will modify those β_p more which can reduce the error faster.

The importance of a feature is determined by two factors: the error reducing capability of the feature and the learning machine used. The values of β_p which are obtained after training will depend not only on the data but also on other parameters of the network. For example, different choice of input and output fuzzy sets and different learning rates may give different values of β_p. Thus, the features selected by the method, are dependent on both the problem that is being solved and the learning machine that is being used. However, if there is a necessary feature its β value is likely to get appropriately updated in different trials.

The importance of a feature is determined by two factors: the error reducing capability of the feature and the learning machine used. The values of β_p which are obtained after

training will depend not only on the data but also on other parameters of the network. For example, different choice of input and output fuzzy sets and different learning rates may give different values of β_p. Thus, the features selected by the method, are dependent on both the problem that is being solved and the learning machine that is being used. However, if there is a necessary feature its β value is likely to get appropriately updated in different trials.

3.4.1 Implicit Tuning of Membership Functions

The feature modulator not only helps us to select good features but also has an interesting side effect. It tune the membership functions to some extent. The output of a layer 2 node is:

$$z_n = (\bar{z}_n)^{\gamma_p},$$

where

$$\bar{z}_n = exp\left\{-\frac{(z_p - \mu_n)^2}{\sigma_n^2}\right\},$$

and $\gamma_p = 1 - e^{-\beta_p^2}$.

Note that, z_n is also a Gaussian function with mean μ_n and spread σ'_n because

$$\left[exp\left\{-\frac{(x - \mu_n)^2}{{\sigma_n}^2}\right\}\right]^{\gamma_p} = exp\left\{-\frac{(x - \mu_n)^2}{{\sigma'_n}^2}\right\} \tag{3.19}$$

where

$$\sigma'_n = \frac{\sigma_n}{\sqrt{\gamma}_p}. \tag{3.20}$$

The feature modulators, thus, tune the spread of the input membership functions and retain their Gaussian structure.

At this point a natural question may come: why are we not tuning the parameters of the membership functions? There are two reasons. First, tuning of spreads of the membership functions although can reduce the total error, it cannot do the task of feature selection. Tuning of membership function parameters refines each rule to reduce the error, while for elimination of a feature the spreads of *all* membership functions defined on that feature need to be modified. Second, simultaneous tuning of modulator functions and the parameters like mean and spread of input membership functions is not desirable. Because, tuning of modulator function looks at all membership functions

defined on a feature as a whole, while the tuning of membership function parameters tries to improve the performance of a rule, i.e., tune parameters of the membership functions considering each rule separately. As a result, if both modulator functions and membership function parameters are tuned, the learning process may become unstable. Therefore, tuning of parameters of input membership functions should be done after feature elimination (i.e., after tuning of the modulator functions). Consequently, we do not discuss tuning of membership function parameters here but we discuss it for the classification network in Chapter 4 (refer Section 4.5.3). But, here too one can apply the same methodology for tuning the membership functions.

Next we discuss strategies to prune the network to get an "optimal" readable network.

3.5 Optimizing the Network

We started with a network which represented all possible rules given a set of input and output fuzzy sets. But all possible rules usually are never needed to represent a system. Moreover, the modulator functions associated with the 2nd layer, may decide that all of the features are not important. Hence, some of the nodes present in the network may be redundant, and presence of these redundant nodes will decrease the readability/interpretability of the network. We know that a system identification task can be easily handled by a conventional Multilayer Perceptron (MLP) network, but we have used a neural fuzzy system for the purpose of FA to increase the readability of the network, so that we can understand the relation between the inputs and outputs in terms of linguistic rules. Thus, to make the network optimal and more readable, we need to prune it removing redundant nodes and incompatible rules. We next discuss what we mean by redundant nodes and incompatible rules, and how to remove them.

3.5.1 Pruning Redundant Nodes

Let us consider a FA problem with s input features. So layer 1 of the network will have s nodes. Let the indices of these nodes be denoted by p ($p = 1$ *to* s). Let $\mathcal{N}_p$ be the set of indices of the nodes in layer 2 which represents the fuzzy sets on the feature represented by node p of layer 1 and let $|\mathcal{N}_p| = N_p$. We also assume that c ($c < s$) of the s features are indifferent/bad as dictated by the training. Let R be the set of

indices of the nodes which represents the c indifferent/bad features. Hence, any node with index p in layer 1 such that $p \in R$ is redundant. Also any node n in layer 2, where $n \in \mathcal{N}_p$ and $p \in R$ is also redundant. In our network construction, a node in layer 3, can be uniquely identified by its connections with the nodes in layer 2. We can indicate a node m in layer 3 as $S_m = [x_{m1}, x_{m2}, ..., x_{ms}]$ where $x_{mp} \in \mathcal{N}_p$. Now for any $p \in R$ we can group the nodes in layer 3 into N_p many groups, we call them G_{pr}, where $r = 1,\ 2, ...,\ N_p$. Every node in the r^{th} group is connected to the r^{th} fuzzy set on the p^{th} feature. Let S_m be a node in layer 3 which belongs to the r^{th} group, i.e., $S_m = [x_{m1}, x_{m2}, ..., x_{ms}] \in G_{pr}$. Then for every group $G_{p\kappa}$, $\kappa \neq r$, $s = 1, 2, ..., N_p$, there exists exactly one node $S_q = [x_{q1}, x_{q2}, ..., x_{qs}]$, such that $x_{qj} = x_{mj}$, $\forall j \neq p$, $j = 1, 2, ..., s$, where $p \in R$ is a bad feature.

Thus, every group of nodes has identical connection structure with the nodes of layer 2 except for its connection to a node corresponding to the redundant feature p, and as per our construction that particular node produces an output membership value of 1, for all feature values. Hence, in layer 3 it is enough to keep only one of the N_p groups and the other $N_p - 1$ groups of nodes are redundant.

To elucidate the concept of redundant nodes, let us consider an FA task with two input features x_1 and x_2. So, layer 1 of the network for this task will have two nodes, we name them as X_1 and X_2 [Fig. 3.2]. We also assume that input feature x_1 has three fuzzy sets associated with it and the feature x_2 has two fuzzy sets associated with it. Hence, layer 2 will have three nodes X_{11}, X_{12}, X_{13} connected with X_1, and two nodes X_{21} and X_{22} connected to X_2. The nodes in layer 3 are named using their connections to nodes in layer 2, for example, a layer 3 node connected to X_{11} and X_{22} will be denoted by $X_{11}X_{22}$ [Fig. 3.2]. Now, if, training dictates that feature x_1 is redundant then irrespective of the values of x_1, each of the nodes X_{11}, X_{12} and X_{13} will produce an output of unity. In this case we can group the nodes in layer 3 into three subsets, which are shown by white nodes, gray nodes and black nodes in Fig. 3.2. Since X_{11} produces an output of unity, the gray group has 2 nodes representing two antecedent clauses "x_2 is X_{21}" and "x_2 is X_{22}". Similarly, each of the white and black groups also represents the same two antecedent clauses as the outputs of both X_{12} and X_{13} are 1. Hence, it is enough to retain any one of the 3 groups. Note that, in this case if two group of nodes are pruned, then the 3rd layer looses its importance, as it really does not do any AND-ing operation. But such a situation, will rarely occur where out

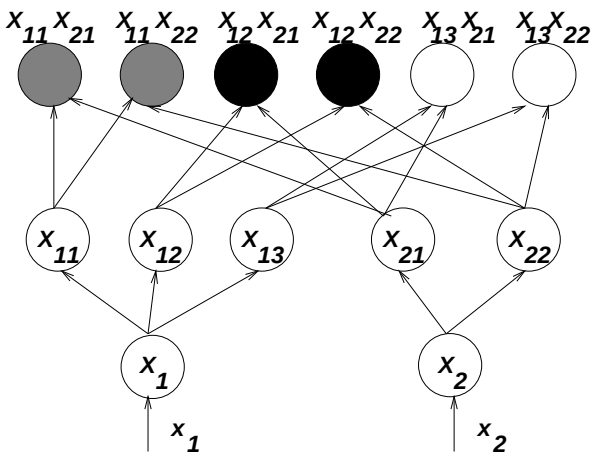


Figure 3.2: Subnet to illustrate redundant nodes.

of only two input features one is redundant. If it happens, then the 3rd layer simply transfers its input to the next layer.

The redundant nodes are not required for the FA task, but they add to the computational overhead of the network. So removal of these nodes is necessary to get an optimal network. The important part of this method is determination of the set of redundant nodes in layer 1. For this we use the value of $1 - e^{-\beta_p^2}$ (we call it γ_p) as an indicator. We have seen earlier, that for good features γ_p takes values close to 1 and for bad features it is close to 0. So we fix a small positive threshold th such that $p \in R$ if $\gamma_p < th$. The method for removal of redundant nodes (here the removal of a node also means removal of its incoming and outgoing links) is summarized in Table 3.1.

Selection of the Threshold th

Now we present a guideline for selecting the threshold th. We have used Gaussian membership functions for the input fuzzy sets (also for output fuzzy sets). Hence, as per our formulation the output of the nodes in layer 2 can be represented by

$$z_n = (\bar{z}_n)^{\gamma_p},$$

where

$$\bar{z}_n = exp\left\{-\frac{(z_p - \mu_n)^2}{\sigma_n^2}\right\},$$

Table 3.1: Pruning of redundant nodes

Algorithm: **Pruning of redundant nodes**

begin
  $R = \phi$
  for each p in layer 1
    if $(\gamma_p < th)$
      $R = R \cup \{p\}$;
      remove node p in layer 1;
      remove the nodes in layer 2
        connected to p;
    end if
  end for
  do while$(R \neq \phi)$
    let $i \in R$;
    $R = R - \{i\}$;
    find G_{ij}, $j = 1, 2, ..., N_i$;
    remove nodes in $G_{i2}, G_{i3}, ..., G_{iN_i}$;
  end do
end

and $\gamma_p = 1 - e^{-\beta_p^2}$. If we consider $\sigma_n = \sqrt{2}\sigma_n{}'$, then we have,

$$\bar{z}_n = exp\left\{-\frac{(z_p - \mu_n)^2}{2\sigma'^2_n}\right\} \tag{3.21}$$

and,

$$z_n = \left[exp\{-\frac{(z_p - \mu_n)^2}{2\sigma'^2_n}\}\right]^{\gamma_p}. \tag{3.22}$$

We know that 99% of the area under the membership function in eq. (3.21) lies over the interval $[\mu_n - 3\sigma'_n, \mu_n + 3\sigma'_n]$. Consequently, the value of $\bar{z}_n$, beyond this interval would be negligibly small. For a bad/indifferent feature we want the modulated membership value z_n to be almost unity over the entire interval $[\mu_n - 3\sigma'_n, \mu_n + 3\sigma'_n]$. Therefore, we can safely choose that value of γ_p as the threshold th, which makes $z_n = c$ ($c \approx 1$) at $z_p = \mu_n - 3\sigma'_n$ and at $\mu_n + 3\sigma'_n$. Thus, from eq. (3.22) we obtain the threshold $th = -\frac{ln(c)}{4.5}$. Note that, for such a choice if $z_p \in (\mu_n - 3\sigma'_n, \mu_n + 3\sigma'_n)$, then $z_n \geq c$. If we consider c=0.8, then we obtain $th = 0.05$, which we use in the simulations reported in this thesis.

3.5.2 Pruning Incompatible Rules

According to our construction of the network, the links between layer 3 and layer 4 represent the rules, and the weights associated with the links can be interpreted as the certainty factor of the rules. But as the nodes in layer 3 and layer 4 are fully connected, initially, all fuzzy rules are considered. If there are τ linguistic values for an output linguistic variable then there are τ rules with the same antecedent but different consequents, which are inherently inconsistent. Let us consider the subnet in Fig. 3.3, which shows only the connections used for selecting the most relevant rule corresponding to the antecedent clause (IF part) represented by the node m in layer 3. Fig. 3.3 corresponds to the following incompatible rules:

If $(antecedent)_m$ then y_k is $T_{l,z_k}(w_{k_r m})$, l=1,2,...,τ.

Where, $(antecedent)_m$ is the antecedent clause represented by node m of layer 3, T_{l,z_k} is the l^{th} fuzzy set on the k^{th} output variable y_k. The certainty factors $w_{k_r m}$ of the rules are shown in parenthesis.

For rule pruning the centroid of the set of incompatible rules is calculated considering

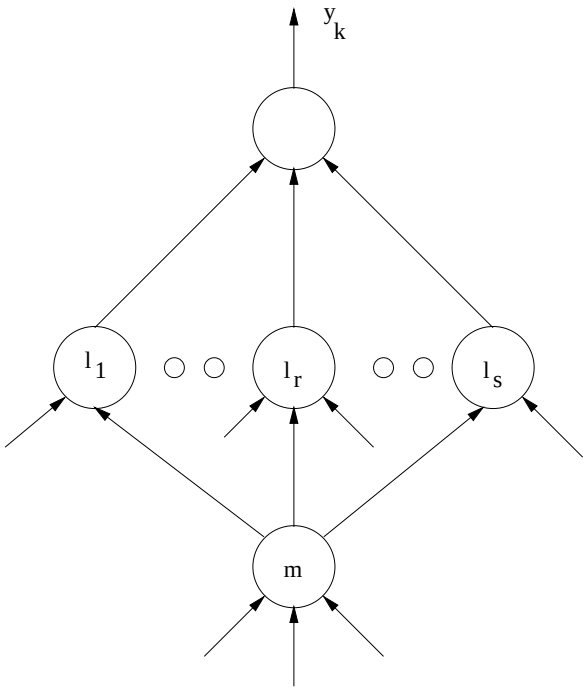


Figure 3.3: Incompatible rules

the connections in Fig. 3.3 as

$$c_{km} = \frac{\sum_{l \in P_k} z_l a_l c_l}{\sum_{l \in P_k} z_l a_l}. \tag{3.23}$$

Since, $z_l = z_m g_{lm}^2$,

$$c_{km} = \frac{\sum_{l \in P_k} z_m g_{lm}^2 a_l c_l}{\sum_{l \in P_k} z_m g_{lm}^2 a_l}. \tag{3.24}$$

Hence,

$$c_{km} = \frac{\sum_{l \in P_k} g_{lm}^2 a_l c_l}{\sum_{l \in P_k} g_{lm}^2 a_l}. \tag{3.25}$$

c_{km} in eq. (3.25) can be viewed as a centroid of the set of incompatible fuzzy rules which corresponds to Fig. 3.3 with certainty factor g_{lm}^2 for the rule with antecedent node m and consequent node l. We calculate the membership values of c_{km} in all consequent fuzzy sets of the incompatible rules. Then the rule which has the highest membership value for c_{km} is selected and the other rules are deleted.

3.6 Training Phases

The training of the system takes place in 3 phases. Phase 1 is called the feature selection phase, where the training is done on the initial network with all the possible nodes and links. The Phase 1 training is considered to be over once the modulator functions stabilize, i.e., when

$$\frac{||\mathbf{\Gamma}(n) - \mathbf{\Gamma}(n+1)||}{s} < \epsilon,$$

where $\mathbf{\Gamma}(n) \in \Re^s$ is the vector of γ_p values after the n^{th} epoch, $||.||$ is the Euclidean norm and ϵ is a small positive constant. After Phase 1 training is over, based on the values of the parameter γ, the pruning of the redundant *nodes* is done. After pruning, the output of the second layer nodes would be as

$$z_n = \bar{z}_n$$

as the modification of the membership value for the purpose of feature selection will no longer be required. After pruning, the network is retrained for a few epochs to adapt its weights in its new reduced architecture, and this phase is called Phase 2 of training. Finally, the incompatible rules (*links*) are pruned and again the network is allowed to learn in its new architecture, which is termed as Phase 3 of training. Let $\boldsymbol{W}(n)$ denote the vector of the weights of all the links connecting layer 3 and layer 4 after the n^{th} epoch in Phase 2. The Phase 2 training can now be stopped when

$$\frac{||\boldsymbol{W}(n) - \boldsymbol{W}(n+1)||}{|\boldsymbol{W}(n)|} < \epsilon,$$

where $|\boldsymbol{W}(n)|$ gives the number of components in $\boldsymbol{W}(n)$. Phase 3 tuning can also be terminated based on the same criteria. Note that for Phase 3, the number of components of $\boldsymbol{W}(n)$ will be less than that in Phase 2. However in the present simulations we have arbitrarily chosen the number of epochs. After the Phase 3 training is over, we obtain a network which is readable, and the rules that describe the input-output relation can be easily retrieved from the final architecture of the network.

3.7 Results

The methodology developed is tested on two data sets taken from [222] and the performance is found to be quite satisfactory. We first describe the data sets and then in

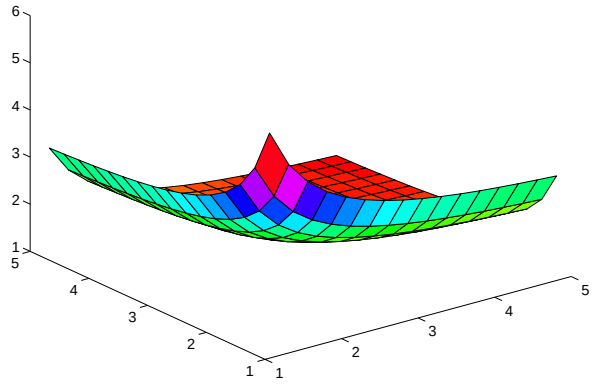


Figure 3.4: Plot of Hang.

two separate subsections we present the results obtained on them.

Of the two data sets one is synthetically generated and the other is a real life one. The first one is named **Hang** which is generated by the equation:

$$y = (1 + x_1^{-2} + x_2^{-1.5})^2, \quad 1 \leq x_1, x_2 \leq 5. \tag{3.26}$$

The graph of eq. (3.26) is shown in Fig. 3.4. Equation (3.26) represents a nonlinear system with two inputs x_1 and x_2 and a single output y. We randomly generate 50 points from $1 \leq x_1, x_2 \leq 5$ and obtain 50 input-output data points according to eq. (3.26). To illustrate the feature analysis capability of the proposed net, we add two random variables x_3 and x_4, in the range [1,5] as dummy inputs. It is expected that features x_3 and x_4 would be indifferent to the output of the system.

The second data set is called **Chem**. This is a data set on operator's control of a chemical plant for producing a polymer by polymerization of some monomers. There are five input features, which a human operator may refer to for control and one output, that is his/her control. The input variables are monomer concentration (u_1), change of monomer concentration (u_2), monomer flow rate (u_3), two local temperatures inside the plant (u_4 and u_5). The only output (y) is the set point for monomer flow rate. In [222] there is a set of 70 data points obtained from an actual plant operation. We name this data set as Chem and use as our training data. In [222] it has been reported that the two local temperatures inside the plant, i.e., u_4 and u_5 *do not significantly*

contribute to the output.

One of the most important issues for rule based system identification is to determine the input and output fuzzy sets. We do not use any sophisticated technique in this regard. We found out the domain of each input and output component and picked up a number of fuzzy sets to span the whole range with considerable overlap between adjacent fuzzy sets. As stated earlier we used fuzzy sets with Gaussian membership functions.

We measure the performance of our system by the sum of squared errors (SSE) and maximum deviation (MD) of the output from the target. Lin and Cunningham [140] defined a performance index PI as:

$$PI = \frac{\sqrt{\sum_{k=1}^{N}(z_k - y_k)^2}}{\sum_{k=1}^{N}|y_k|}, \tag{3.27}$$

where z_k denotes the output for the k^{th} point and y_k denotes the desired output for the same point. But in [206] it was pointed out that this performance index is monotonically decreasing with $N^{-1/2}$, i.e., it is possible to obtain a very small PI just by increasing N. Still we evaluated the performance of our system based on PI in eq. (3.27) for an easy comparison with results reported in the literature.

3.7.1 Results on Hang

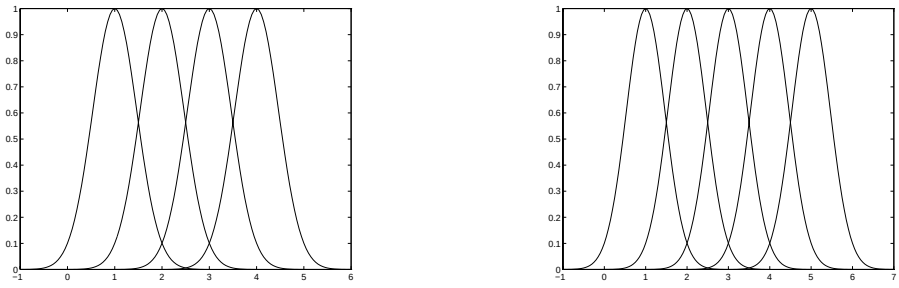
Here we used four input fuzzy sets for each input feature and five output fuzzy sets for the output linguistic variable. The input and output fuzzy sets are shown in Fig. 3.5. Hence, the initial architecture for this problem is as described in Table 3.2.

The network was trained using the data set, Hang with learning parameters $\eta = 0.1$ and $\mu = 0.1$ for 1000 epochs in Phase 1, 500 epochs in Phase 2 and 3500 epochs in Phase 3. The SSE was reduced from 57.907 to 1.513. The PI was equal to 0.01, which is comparable to the result obtained by Sugeno and Yasukawa [222], who obtained a PI of 0.01. Using this data Lin and Cunningham [140] obtained a PI of 0.003, but in their case they used only the good features, i.e., only features x_1 and x_2. Moreover, we did not tune the membership functions defined on the input and output variables which could improve the results further.

The values of β_p for the various features and the corresponding values of $1 - e^{-\beta_p^2}$

Table 3.2: Architecture of the neural fuzzy system used for Hang.

layer no.	no. of nodes
1	4
2	16
3	256
4	5
5	1



(a) (b)

Figure 3.5: Membership functions used for Hang: (a) Input membership functions, (b) Output membership functions.

Table 3.3: Value of β_p for different input features for Hang.

	x_1	x_2	x_3	x_4
β_p	2.53	2.54	0.00	0.00
$1 - e^{-\beta_p^2}$	0.99	0.99	0.00	0.00

after the Phase 1 training are given in Table 3.3. Table 3.3 clearly shows that the network is able to indicate features x_3 and x_4 as not important and eliminate their effect completely on the output. In this case, as Table 3.2 shows, we started with 256 nodes in layer 3, i.e., 256 antecedent clauses. Also as layer 4 contains 5 nodes, the initial architecture represented $256 \times 5 = 1280$ rules. But Phase 1 of training indicates that two features are redundant/bad. Before Phase 2 training, the network is pruned of the redundant nodes, which reduces the antecedent clauses to 16, hence, the number of rules gets reduced to 16×5, i.e., 80. Since after Phase 2 incompatible rules are removed, the total number of rules represented by the final architecture is 16. Thus, here we obtain a 99.75% reduction in the number of rules in the final architecture.

We also investigated the generalizing capability of the network. A mesh of 256 points in the range $1 \leq x_1, x_2 \leq 5$ was considered. The network then results in a SSE of 17.07 and a PI of 0.008. The mean square error on the test set was 0.06. The maximum deviation of the desired output from the obtained output was 0.79. This proves that the network also has good generalizing capabilities. The difference of the correct surface and the surface produced by our system is shown in Fig. 3.6.

3.7.2 Results on Chem

As described before, this data set has 5 input features namely u_1, u_2, u_3, u_4, and u_5 and a single output y. The number of input and output fuzzy sets considered are shown in Table 3.4, and the initial number of nodes in the different layers are depicted in Table 3.5. The membership functions of the various fuzzy sets used for this data set are depicted in Fig. 3.7.

For Chem, the learning parameters were $\eta = 0.0001$ and $\mu = 0.00001$ and the training was continued for 1000 epochs in Phase 1, 500 epochs in Phase 2 and 3500 epochs

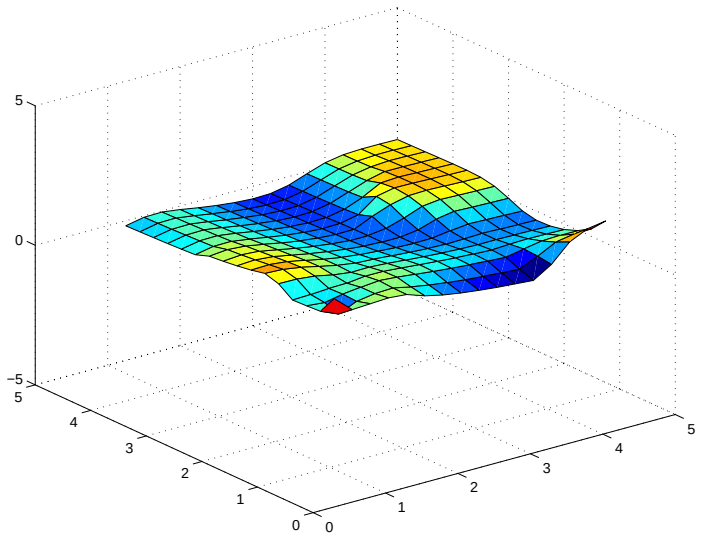


Figure 3.6: Difference surface for Hang.

Table 3.4: No of Fuzzy sets for different features used with Chem

Features	No of Fuzzy Sets
x_1	4
x_2	2
x_3	4
x_4	2
x_5	2
y	7

Table 3.5: Initial architecture of the Neural Fuzzy System used for Chem

layer no.	no. of nodes
1	5
2	14
3	128
4	7
5	1

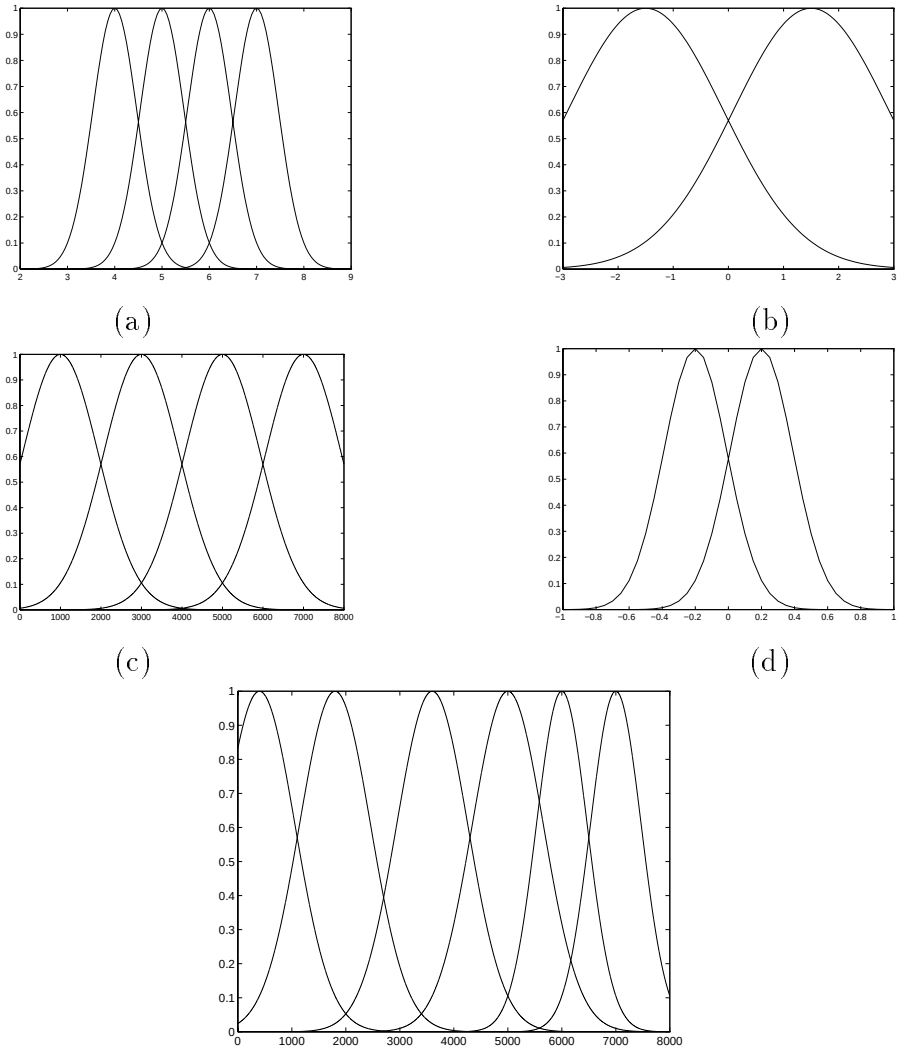


Figure 3.7: Membership function used for different features of Chem data: (a) Membership function for u_1 (b) Membership function for u_2 (c) Membership function for u_3 (d) Membership function for u_4 and u_5 (e) Membership function for y

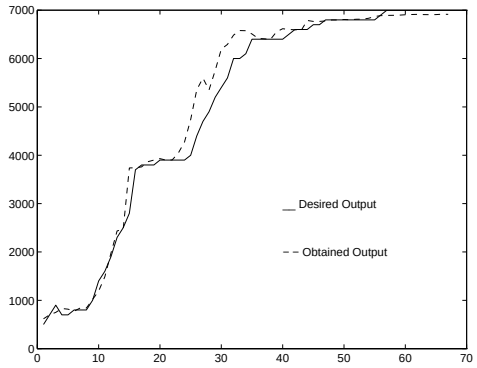


Figure 3.8: Performance comparison of the proposed system for Chem data

Table 3.6: Values of β_p for different input features.

	u_1	u_2	u_3	u_4	u_5
β_p	-2.53	1.98	1.39	0.21	0.20
$1 - e^{-\beta_p^2}$	0.99	0.98	0.85	0.04	0.04

in Phase 3. The PI was equal to 0.0021 after Phase 3. Lin and Cunningham [140] obtained a PI of 0.0022. Sugeno and Yasukawa [222] do not provide any performance measure of their system on this data. The performance of our system is compared with that of the real output in Fig. 3.8, which exhibits a good match.

The values of β_p and $1 - e^{-\beta_p^2}$ for the various features after the Phase 1 training are given in Table 3.6. Table 3.6 again establishes the capability of the proposed system in identifying the features that are not important. It clearly shows that u_4 and u_5 do not contribute significantly to the output of the system - thus they are indifferent or bad features. This result conforms to the findings of Sugeno and Yasukawa [222], who also found that features u_1 to u_3 are the only important ones. In this case, the number of antecedent clauses at the beginning of Phase 1 training was 128 (Table 3.5) and the number of nodes in layer 4 was 7. Thus the initial architecture represented $128 \times 7 = 896$ rules. At the end of Phase 1 two features were identified as not important, and thus, pruning of redundant nodes yielded 32 antecedent clauses resulting in $32 \times 7 = 224$ rules. After pruning of the incompatible rules, the final architecture represents 32 rules. So in this case we obtain 96.42% reduction in the number of rules.

The generalization ability of the network for this data could not be measured as we could not get any data to do so.

3.8 Conclusion and Discussions

In this chapter we proposed a novel scheme for simultaneous feature selection and system identification in a neuro-fuzzy framework. It is a five layer network, which can realize a fuzzy rule based inferencing system and at the same time can find out the features which are not important. We also proposed methodologies for pruning the redundant nodes and incompatible rules to obtain a more readable network. The proposed system has been implemented on several data sets and the results obtained are quite good.

There are a few issues that have not been considered in the present work. We have not given any guidelines to decide on the number of input and output fuzzy sets and their definitions, which are important for designing a good system. We did not tune the parameters of different membership functions used. Tuning of the membership functions is expected improve the performance further. We consider tuning of the input membership functions for the classification network described in the next chapter.

Our pruning strategy removes redundant nodes and eliminates incompatible rules but the system begins with all possible antecedent clauses, which may not always be required. There may also be some rules which are never fired or are fired only by a few data points. In the FA architecture we have not considered pruning of such less used rules. Pruning of such rules may further bring down the complexity of the architecture. In the next chapter we have considered pruning of less used rules in the context of classification problems. The same technique can be applied to the FA network described in this chapter.

The main thrust of this chapter was to demonstrate the effectiveness of the proposed network for simultaneous (online) feature analysis and systems identification and it is found to do a good job as revealed by the simulation results. Now, a natural question arises: can we use the network proposed here for classification type problems? The nature of the outputs for a classification system restricts the use of this network, as is, for such problems. In the next chapter we propose a modification of this network

to enable it to solve classification problems. Also we address some limitations of the function approximation network.

Chapter 4

Online Feature Selection and Classifier Design in a Neuro-Fuzzy Paradigm[1]

4.1 Introduction

A classifier is a function $\boldsymbol{S} : \Re^s \rightarrow N_{pt}$, where N_{pt} is a set of label vectors of dimension t defined as

$$N_{pt} = \left\{ \boldsymbol{y} \in \Re^t : y_i \in [0,1] \forall i \ \ \& \ \ \exists i, y_i > 0 \right\} = [0,1]^t - \{\mathbf{0}\}. \tag{4.1}$$

In eq. (4.1), $[0,1]^t$ represents all t dimensional vectors whose each component lies in the interval $[0,1]$ and $\mathbf{0}$ represents the t dimensional vector with all zero components. Thus, a classifier is a function which takes as input an object data, i.e., a feature vector in $\Re^s$ and assigns a class label to it. More specific types of label vectors can be used for specific types of classifiers as

$$N_{ft} = \left\{ \boldsymbol{y} \in N_{pt} : \sum_{i=1}^{t} y_i = 1 \right\}; \tag{4.2}$$

$$N_{ht} = \left\{ \boldsymbol{y} \in N_{ft} : y_i \in \{0,1\} \forall i \right\}. \tag{4.3}$$

$\boldsymbol{S}$ is called a crisp classifier, if it assigns the object to one of the classes without any ambiguity, i.e., if $\boldsymbol{S}[\Re^s] = N_{ht}$. The classifier is fuzzy or probabilistic if $\boldsymbol{S}[\Re^s] = N_{ft}$, while $\boldsymbol{S}[\Re^s] = N_{pt}$ gives possibilistic classifiers. All these classifiers together are sometimes called soft classifiers [12]. Designing classifiers means to find a good $\boldsymbol{S}$. $\boldsymbol{S}$ may be an analytical function (like the Bayes classifier) or it can be a computational

[1] The contents of this chapter have been published in [25, 27].

transform which does classification implicitly. Fuzzy systems, neural networks or other hybrid systems are examples of such computational transforms.

Fuzzy systems built on fuzzy rules have been successfully applied to various classification tasks [12, 38, 70, 87, 101, 106, 153, 156, 174, 179, 207]. Fuzzy systems depend on linguistic rules which are provided by experts or the rules are extracted from a given training data set using methods like exploratory data analysis, evolutionary algorithms etc. [167, 205, 207, 208, 235].

In Chapter 3 we discussed a methodology for online feature analysis and function approximation in a 5-layered neuro-fuzzy network. In this chapter we describe a neuro-fuzzy system for online (simultaneous) feature selection and classification. The feature selection strategy which we use in this chapter is the same as in Chapter 3. But, unlike the network in Chapter 3 this is a four layered network. Also the methods used to optimize the network have been modified to suite the classification process. In this regard a few *new concepts* have also been introduced for pruning of the network and hence the rule-base.

Our network is trained in three phases. In phase-I, starting with some coarse definition of initial membership functions, the network selects important features and learns the initial rules. In the phase-II, the redundant nodes as detected by the feature attenuators are pruned, and the network is re-tuned to gain performance in its reduced architecture. In phase-III, the architecture is further reduced by pruning incompatible rules, zero rules and less used rules. After pruning, the network represents the final set of rules. The membership functions which constitute the final rules are then tuned to achieve better performance.

In the next section we describe the structure of fuzzy rules for classification and also the network architecture to realize them. In Section 4.3 we derive the learning rules for the feature modulators and certainty factors of the fuzzy rules. In Section 4.4 we discuss a pruning algorithm to get rid of the redundant nodes. Section 4.5 discusses two more pruning strategies along with a scheme for tuning of the membership function parameters. Section 4.6 includes the simulation results on a synthetic data set and three real data sets. Finally, this chapter is concluded in Section 4.7.

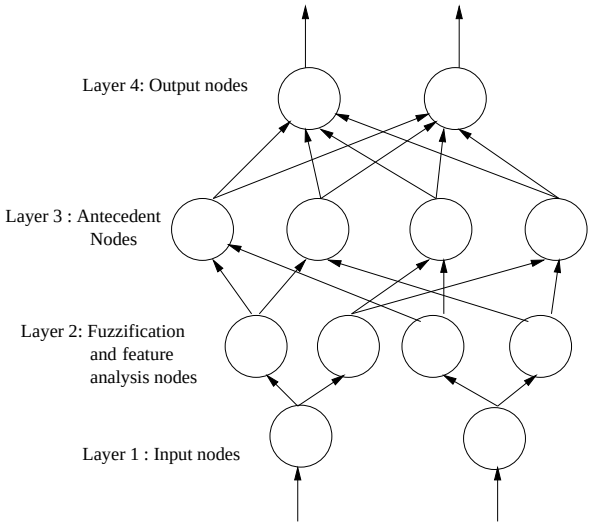


Figure 4.1: The structure of the classification network.

4.2 The Classification Network

Let there be s input features $(x_1, x_2, ..., x_s)$ and t classes $(\omega_1, \omega_2, ..., \omega_t)$. Given a $\boldsymbol{x} \in \Re^s$ the proposed neural-fuzzy system deals with fuzzy rules of the form, R_i : If x_1 is A_{1i} and x_2 is A_{2i} ..... and x_s is A_{si} then $\boldsymbol{x}$ belongs to class ω_l with a certainty d_l, $(1 \leq l \leq t)$. Here A_{ji} is the i-th fuzzy set defined on the domain of x_j.

Note that, the structure of the rules used here is quite different from the structure of the rules used in Chapter 3. Here, the classification system is realized using a four layered network as shown in Fig. 4.1. The first layer is the input layer, the second layer is the membership function and feature selection layer, the third layer is called the antecedent layer and the fourth layer is the output layer. As in the previous chapter we use suffixes p, n, m, l to denote respectively the suffixes of the nodes in layers 1 through 4. The output of each node is denoted by z.

The first two layers of the network bear the same meanings and perform the same task as the function approximation network in Chapter 3. The output of the p^{th} node of layer 1 (the input layer) is

$$z_p = x_p. \tag{4.4}$$

The output of any node n in layer 2 (fuzzification and feature analysis layer) is

$$z_n = \bar{z}_n^{(1-e^{-\beta_p^2})}. \tag{4.5}$$

where

$$\bar{z}_n = exp\left\{-\frac{(z_p - \mu_n)^2}{\sigma_n^2}\right\}. \tag{4.6}$$

Here too we assume bell shaped (Gaussian) membership functions. μ_n and σ_n are the center and spread respectively of the bell shaped function representing a term of the linguistic variable x_p associated to node n (n indicates the n-th term (fuzzy set) of the linguistic variable x_p). The justification behind eq. (4.5) can be found in Section 3.3. Up to this the functioning of the present network is similar to that in Chapter 3.

In layer 3 (the antecedent layer) each node represents the IF part of a fuzzy rule. The nodes in this layer performs an intersection operation. In Chapter 3 we used product as the operator for intersection and got fairly good results. But we know that for any *T-norm* $\mathcal{T}$, $\mathcal{T}(x, y) \leq min(x, y)$. So use of product as the intersection operator is counter-intuitive. To elaborate it, consider two propositions with truth values a and b. It is not natural to assume that they will produce a firing strength less than $min(a, b)$. Let us consider a rule :
If x_1 is A_1 *and* x_2 is A_2 *and* ... x_s is A_s then the class is K.
Suppose for an input $\boldsymbol{x} = (x_1, x_2, ..., x_s)^T$ each x_i, $i = 1, 2, .., s$, has a membership of 0.9 in the respective fuzzy set A_i, $i = 1, 2, ..., s$. Thus, for this input if *product* is used as the operator for intersection then the firing strength of the rule will be $(0.9)^s$. So, the firing strength decreases exponentially with s. Consequently, for a reasonably big s, the firing strength reduces almost to zero, though each of the input components have a high membership of 0.9 in the corresponding fuzzy sets. Therefore, the use of product as an operator for intersection is not intuitively appealing. One might wonder why did we (others also) use and obtain good results using product in Chapter 3. The answer lies in the defuzzification process. For example, in the height method of defuzzification, the defuzzified value is computed as a weighted sum of the peaks of the output fuzzy sets, where the weights are the normalized values of the firing strengths. The output is then computed as a convex combination of the peaks of the output fuzzy sets. But in this case such defuzzification methods cannot be applied. Note that, in our classification rules the consequents are class labels which are categorical quantities and a convex combination of classes is not meaningful. Since, crisp class labels are

categorical variables, defining fuzzy sets on them is also not meaningful. Hence, in such a case center of area type defuzzification is not applicable. Consequently, here we use min as the operator for intersection. As min is not differentiable, for ease of computation many softer versions of min that are differentiable have been previously used [4, 162]. We use the following soft version of min which we call $softmin$:

$$softmin(x_1, x_2, ..., x_s, q) = \left(\frac{x_1^q + x_2^q + ... + x_s^q}{s} \right)^{\frac{1}{q}}.$$

As $q \to -\infty$, $softmin$ tends to the minimum of all x_i's, $i = 1, 2, .., s$. For our purpose we use $q = -12$ in all results reported. Note that, $softmin$ is not a *T-norm* as it does not satisfy the associativity property. For our feature selection task, the intersection operator must satisfy the *identity* property, which $softmin$ does possess for $q \to -\infty$. So, $softmin$ is compatible with our feature selection strategy though it is not a *T-norm*. Thus, the output of the m-th node in layer 3 is

$$z_m = \left(\frac{\sum_{n \in P_m} z_n^q}{|P_m|} \right)^{\frac{1}{q}}, \tag{4.7}$$

where P_m is the set of indices of the nodes in layer 2 connected to node m of layer 3 and $|P_m|$ denotes cardinality of P_m.

In layer 4 (the output layer), each node represents a class. So, if there are t classes then there will be t nodes in layer 4. The nodes in this layer perform an OR operation, which combine the antecedents of layer 3 with the consequents. According to the structure of the fuzzy rules that we are concerned about, the consequent of a rule is a class with a degree of certainty. The nodes in layers 3 and 4 are fully connected. Let w_{lm} be the connection weight between node m of layer 3 and node l of layer 4. The weight w_{lm} represents the certainty factor of a fuzzy rule, which comprises the antecedent node m in layer 3 as the IF part and the output node l in layer 4 representing the THEN part. These weights are adjustable while learning the fuzzy rules. The OR operation is performed by some *s-norm* [117]. We use here the max operator. Thus the output of node l in layer 4 is computed by

$$z_l = max_{m \in P_l}(z_m w_{lm}), \tag{4.8}$$

where P_l represents the set of indices of the nodes in layer 3 connected to the node l of layer 4. Thus, the output of node l in this layer represents the certainty with which

a data point belongs to class l. We classify a point $\boldsymbol{x}$ to a class q if $z_q = max_l(z_l)$. Since w_{lm}s are interpreted as certainty factors, each w_{lm} should be non-negative and it should lie in $[0, 1]$. The error backpropagation algorithm or any other gradient based search algorithm does not guarantee that w_{lm} will remain non-negative, even if we start the training with non-negative weights. Hence, we model w_{lm} by $e^{-g_{lm}^2}$. The g_{lm} is unrestricted in sign but the effective weight $w_{lm} = e^{-g_{lm}^2}$ will always be non-negative and lie in [0,1]. Therefore, the output (activation function) of the l-th node in layer 4 is

$$z_l = max_{m \in P_l}(z_m e^{-g_{lm}^2}). \tag{4.9}$$

Since it is enough to pick up the node with the maximum value of z_l, the product of firing strength and certainty factor, for applications it is not necessary to maintain w_{lm} in $[0, 1]$. The non-negativity alone would be enough. So we use $w_{lm} = g_{lm}^2$. This also reduces the computational overhead. Consequently, eq. (4.9) can be modified to

$$z_l = max_{m \in P_l}(z_m g_{lm}^2). \tag{4.10}$$

Note that, in layer 4 we use the usual max operator instead of a differentiable soft version of max. As max is not differentiable the update equations that we derive in the next section have to be split for different conditions of the network. We could have used a differentiable (soft) version of max here too. If we use both soft version of min and soft version of max as activation functions in two layers of the network, the update equations would become complicated. Hence, we choose to use the usual max in this layer and a differentiable version of min in layer 3.

4.3 Learning Phase I: Feature Selection and Rule Extraction

We now derive the learning rules for the neural-fuzzy classifier with the activation or node functions described in the previous section. In the training phase, the concept of backpropagation is used to minimize the error function

$$e = \frac{1}{2}\sum_{i=1}^{N} E_i = \frac{1}{2}\sum_{i=1}^{N}\sum_{l=1}^{t}(y_{il} - z_{il})^2, \tag{4.11}$$

where t is the number of nodes in layer 4 and y_{il} and z_{il} are the target and actual outputs of node l in layer 4 for input data $\boldsymbol{x}_i$; $i = 1, 2, ..., N$. The method for adjusting

the learnable weights in layer 4 and the parameters β_p in layer 2 is based on gradient descent search. We use online update scheme and hence derive the learning rules using the instantaneous error function E_i. Without loss we drop the subscript i in our subsequent discussions.

The delta value, δ, of a node in the network is defined as the influence of the node output on E. The derivation of the delta values and the adjustment of the weights and the parameters β_p are presented layer wise next.

Layer 4: The output of the nodes in this layer is given by equation (4.10) and the δ values for this layer, δ_l, will be

$$\delta_l = \frac{\partial E}{\partial z_l}.$$

Thus,

$$\delta_l = -(y_l - z_l). \tag{4.12}$$

Layer 3: The delta for this layer is

$$\delta_m = \frac{\partial E}{\partial z_m} = \frac{\partial E}{\partial z_l}\frac{\partial z_l}{\partial z_m}.$$

Hence, the value of δ_m will be

$$\delta_m = \begin{cases} \sum_{l \in Q_m} \delta_l g_{lm}^2 & \text{if } z_m g_{lm}^2 = \max_{m'}\{z_{m'} g_{lm'}^2\} \\ 0 & \text{otherwise.} \end{cases} \tag{4.13}$$

Here Q_m is the set of indices of the nodes in layer 4 connected with node m of layer 3.

Layer 2: Similarly, the δ_n for layer 2 is

$$\delta_n = \frac{\partial E}{\partial z_n} = \frac{\partial E}{\partial z_m}\frac{\partial z_m}{\partial z_n}.$$

Hence,

$$\delta_n = \sum_{m \in R_n} \delta_m \left(\frac{z_m z_n^{q-1}}{\sum_{n \in P_m} z_n^q} \right). \tag{4.14}$$

In (4.14) R_n is the set of indices of nodes in layer 3 connected with node n in layer 2.

With the δ calculated for each layer now we can write the weight update equation and the equation for updating β_p.

$$\frac{\partial E}{\partial g_{lm}} = \frac{\partial E}{\partial z_l}\frac{\partial z_l}{\partial g_{lm}},$$

or

$$\frac{\partial E}{\partial g_{lm}} = \begin{cases} \sum_{l \in Q_m} 2\delta_l z_m g_{lm} & \text{if } z_m g_{lm}^2 = \max_{m'}\{z_{m'} g_{lm'}^2\} \\ 0 & \text{otherwise.} \end{cases} \tag{4.15}$$

Similarly, we calculate

$$\frac{\partial E}{\partial \beta_p} = \frac{\partial E}{\partial z_n}\frac{\partial z_n}{\partial \beta_p},$$

or

$$\frac{\partial E}{\partial \beta_p} = -\sum_{n \in R_p} \delta_n \left(2\ \beta_p\ e^{-\beta_p^2}\ z_n\right) \left(\frac{z_p - \mu_n}{\sigma_n}\right)^2. \tag{4.16}$$

Here, R_p is the set of indices of nodes in layer 2 connected to node p of layer 1. Hence, the update equations for weights, g_{lm} and β_p are

$$g_{lm}(n+1) = g_{lm}(n) - \eta \left(\frac{\partial E}{\partial g_{lm}(n)}\right) \tag{4.17}$$

and

$$\beta_p(n+1) = \beta_p(n) - \nu \left(\frac{\partial E}{\partial \beta_p(n)}\right). \tag{4.18}$$

In eq. (4.17) and eq. (4.18) η and ν are learning coefficients, which are usually chosen by trial and error or one can use methods described in [77, 137] for better choices.

In learning phase-I the network learns the weights of the links connecting layers 3 and 4 and also the parameters associated with nodes in layer 2, which do the feature selection. The initial values of β's are so selected that no feature gets into the network in the beginning and the learning algorithm will pass the features, which are important, i.e., the features that can reduce the error rapidly.

Explicit tuning of the membership function parameters are not performed in this phase, the reason behind this is discussed in Section 3.4.1. Here too, the modulator functions implicitly modify the input membership functions in the same way as discussed in Section 3.4.1.

Next we discuss the second phase of learning which prunes the network and results in a reduced architecture representing a small but adequate rule-base.

4.4 Leaning Phase II: Pruning Redundant Nodes and Further Training

For the classification network also a number of pruning strategies can be applied to get a reduced rule-base. The modulator functions associated with the second layer nodes, may decide that all features present are not important. Hence, some of the nodes present in the network may be redundant. Also as per our formulation each node in layer 3 is connected with all nodes in layer 4, which gives rise to *incompatible rules* that need to be removed. Further, here we used fuzzy sets which covered the total domain of each feature, thus the antecedents cover the entire hyperbox that bounds the data. The training data may (usually will) not be distributed even over the smallest hyperbox containing the data. Consequently, there may be some rules which are never fired by the training data. Such rules which are not supported by the training data could be harmful. The certainty factors of such rules may not be meaningful and can lead to bad generalization. We call such rules as *less used rules*. In the function approximation network of Chapter 3 we did not consider pruning such rules. But, such rules can appear in case of function approximation also. So, it is necessary to get rid of *redundant nodes*, *incompatible rules* and *less used rules*.

4.4.1 Pruning Redundant Nodes

The definition of redundant nodes for the classification network remains exactly the same as that of redundant nodes for the function approximation system. Redundant nodes arises as the network may totally discard some of the features present in the data. Pruning redundant nodes involves pruning nodes and links associated with the first two layers of the network. As these two layers have exactly the same structure and function as that of the function approximation network, hence the same pruning strategy as discussed in Section 3.5 can be used here. After pruning the redundant nodes, the certainty factor of the rules are further tuned using eq. (4.17). Phase II training ends, once the values of w_{lm} stabilize.

4.5 Learning Phase III: Pruning Incompatible Rules, Less used Rules and Zero Rules and Further Training

In phase III, the network is pruned further and the certainty factors of the rules are again tuned. In this last phase of training the parameters of the membership functions are also tuned. The details are presented in the following subsections.

4.5.1 Pruning Incompatible Rules

As per construction of our network, the nodes in layer 3 and layer 4 are fully connected and each link corresponds to a rule. The weight associated with each link is treated as the certainty factor of the corresponding rule. If there are t classes then layer 4 will have t nodes and there will be t rules with the same antecedent but different consequents, which are inherently inconsistent.

Suppose layer 3 has N^3 nodes (i.e., N^3 antecedents) and layer 4 has N^4 nodes (thus, N^4 consequents or classes). Each node m in layer 3 is then connected to N^4 nodes in layer 4. The link connecting node m of layer 3 and node l of layer 4 has a weight w_{ml} associated with it, which is interpreted as the certainty factor of the rule represented by the link. For each node m in layer 3 we retain only one link with a node in layer 4 that has the highest certainty factor associated with it. The removal of incompatible rules is described more explicitly in the algorithm presented in Table 4.1.

4.5.2 Pruning Zero Rules and Less Used Rules

After removal of the incompatible rules each node in layer 3 is connected with only one node in layer 4. Suppose node m in layer 3, which is connected to a node l in layer 4, has a very low weight $w_{lm} < \omega_{low}$ (we take $\omega_{low} = 0.001$). Thus, the rule associated with the node pair m and l has a very low certainty factor and it does not contribute significantly in the classification process. We call such rules as *Zero rules*. These rules can be removed from the network. In fact such rules should be removed from the network as we do not like to make any decision with a very low confidence. Removal of a zero rule means removing a node in layer 3 along with its links.

Table 4.1: Pruning incompatible rules

Algorithm: **Pruning of incompatible rules**

begin
  for each node m in layer 3
    Find j such that $w_{mj} = max_l\{w_{ml}\}$;
    Retain the link between node m
      and node j;
    Remove all other links connecting
      node m to layer 4;
  end for
end

Further, as discussed earlier, our network starts with all possible antecedents which cover a hyperbox bounding the data. Hence, there may (usually will) be rules which are never fired or fired by only a few data points. Such rules are not well supported by the training data and will result in bad generalization. We call such rules as *less used rules* and they are also removed.

Consider an antecedent node m in layer 3. We count the number of training data points n_t for which the firing strength of the antecedent clause represented by node m is greater than a threshold $\alpha \in (0, 1)$. If n_t is less than τ then we can consider node m as well as the rule represented by it as inadequately supported by the training data. Every such node m in layer 3 along with its links can be removed. This will help us to avoid bad generalization. In our simulations the threshold α is selected as 0.1 and τ as 3. Note that, the choice of τ is related to the definition of outliers. If a rule represents outliers, we should delete it. In this study unless a rule represents at least three training points, we take it as a less-used rule and delete it.

4.5.3 Tuning Parameters of the Reduced Rule Base

The network (rule base) obtained after pruning of the redundant nodes, less used rules and the zero rules is considerably smaller than the initial network with which we began. The parameters of the membership functions of this reduced rule base are now tuned to get a better classifier performance. The update equations for the centers and the spreads of the membership functions can be written as

$$\mu_n(t+1) = \mu_n(t) - \lambda \frac{\partial E}{\partial \mu_n(t)} \tag{4.19}$$

and

$$\sigma_n(t+1) = \sigma_n(t) - \lambda \frac{\partial E}{\partial \sigma_n(t)}. \tag{4.20}$$

Here

$$\frac{\partial E}{\partial \mu_n} = 2\delta_n \gamma_p z_n \frac{(z_p - \mu_n)}{\sigma_n^2} \tag{4.21}$$

and

$$\frac{\partial E}{\partial \sigma_n} = 2\delta_n \gamma_p z_n \frac{(z_p - \mu_n)^2}{\sigma_n^3}, \tag{4.22}$$

where z_p is the feature related to node n of the 2nd layer, λ is a predefined learning constant, and $\gamma_p = 1 - e^{\beta_p^2}$.

The update eqs. (4.19) and eq. (4.20) are applied iteratively along with eq. (4.17) till there is no further decrement in the error defined by eq. (4.11). We emphasize that in this phase of training the β values are *not* updated. As discussed earlier, tuning of the feature modulators along with the membership function parameters may make the training unstable.

4.6 Results

4.6.1 The Data Sets

The methodology is tested on a few data sets and the results obtained on them are quite satisfactory. We use four data sets: a synthetic data set named Elongated and three real data sets named Iris, Phoneme and RS-Data.

Elongated [149] has three features and 2 classes, the scatterplots of features 1-2, 2-3 and 1-3 are shown in Fig. 4.2. These plots show that features 1 and 2 or features 2

and 3 are enough for the classification task, thus any one of these two combinations is enough for the classification task. Iris [17] is a data set with four features and 3 classes. It is well known that for Iris, features 3 and 4 are enough for the classification task [12]. The Phoneme data set contains vowels coming from 1809 isolated syllables (for example: pa, ta, pan,...) in French and Spanish language [122, 256]. Five different attributes are chosen to characterize each vowel. These attributes are the amplitudes of the five first harmonics AHi, normalized by the total energy Ene (integrated on all frequencies), AHi/Ene. Each harmonic is signed positive when it corresponds to a local maximum of the spectrum and negative otherwise. The Phoneme data set has two classes, nasal and oral. The RS-Data [121] is a satellite image of size 512×512

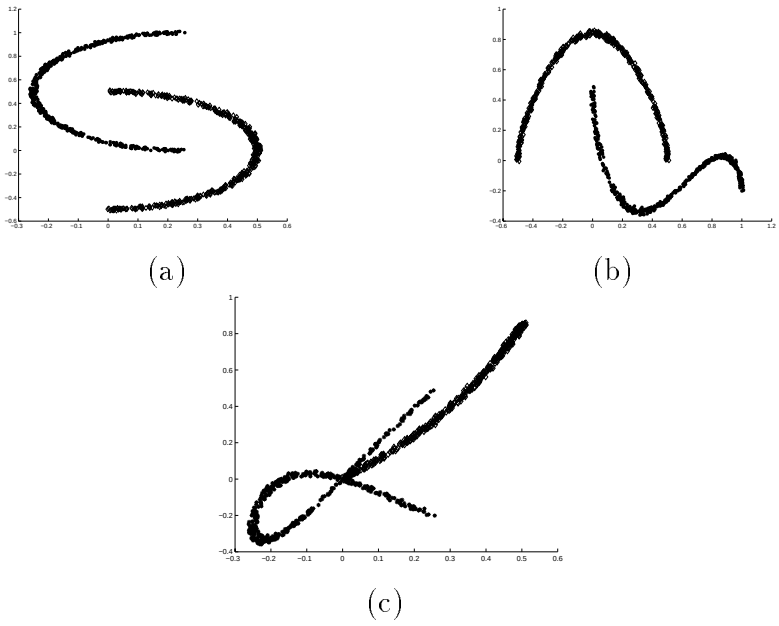


Figure 4.2: Plot of Elongated: (a) features 1-2 (b) features 2-3 (c) features 1-3

pixels captured by seven sensors operating in different spectral bands from Landsat-TM3. Each of the sensors generates an image with pixel values varying from 0 to 255. The 512×512 ground truth data provide the actual distribution of classes of objects captured in the image. From these images we produce the labeled data set with each pixel represented by a 7-dimensional feature vector and a class label. Each dimension of a feature vector comes from one channel and the class label comes from the ground truth data.

We divide each data set X into training (X_{tr}) and test (X_{te}) sets, such that $X_{tr} \cup X_{te} =$

X and $X_{tr} \cap X_{te} = \phi$. For Elongated, Iris and Phoneme the training and test divisions were made randomly. For RS-Data we created a training sample containing exactly 200 points randomly selected from each class. The summary of the data sets used is given in Table 4.2.

Table 4.2: Summary of the data sets

Name	Total Size	Trng. Size	Test Size	No. of classes	No. of features
Elongated	1000	500	500	2	3
Iris	150	100	50	3	4
Phoneme	5404	500	4904	2	5
RS-Data	262144	1600	260544	8	7

4.6.2 The Implementation Details

Before we present the results we discuss a few implementational issues. The network contains the feature modulators (β_p's), the certainty factor of the rules (g_{lm}'s) and the membership function parameters (μ_n and σ_n) as free parameters. The initial values of β_p's are set to 0.001; so, initially the network considers all features to be equally *unimportant*. The initial values of the certainty factor of the rules, i.e., the weights of the links between layer 3 and layer 4 are all set to 1.0, which signifies that initially all rules have the *same* certainty factor. For the initial values of the membership function parameters, for each feature we choose an arbitrary number of equidistant fuzzy sets with considerable overlap between adjacent ones. These fuzzy sets span the entire domain of the feature. The choice of fuzzy sets may have considerable influence on the performance of the classifier. For one data set with complex class structures we use exploratory data analysis to get the initial network structure. It is possible to design more elaborate methods using clustering and cluster validity indices to decide the initial membership functions. For some such methods, readers can refer to [159, 170]. Since the thrust of this chapter is to establish the utility of integrated feature selection and classifier design in a neuro-fuzzy framework, we do not pursue the issue of network initialization further. Table 4.3 depicts the list of the free parameters with their meanings and initial values.

Table 4.3: Free parameters of the network

Parameter	Meaning	Initial Value	Updating phase and update equations
β_p	Modulator function for feature p	0.001	Phase I eq. (4.18)
g_{lm}	Certainty factor of a rule represented by antecedent node m and output node l	1.0	Phases I,II and III eq. (4.17)
μ_n	Center of the membership function denoted by node n of layer 2	As discussed	Phase III eq. (4.19)
σ_n	Spread of the membership function denoted by node n of layer 2	As discussed	Phase III eq. (4.20)

Table 4.4: User defined parameters

Parameter	Meaning	Suggested Value(s)
η	Learning constant for certainty factors of rules	Decided according to data
ν	Learning constant for feature modulators	Decided according to data
λ	Learning constant for membership function parameters	Decided according to data
τ	Threshold on the number of data points for choosing *less used* rules	3
th	Threshold for choosing redundant nodes	0.05
ω_{low}	Threshold for choosing *zero* rules	0.001

Table 4.5: Number of fuzzy sets for each feature for Elongated

Feature 1	3
Feature 2	4
Feature 3	3

Other than the free parameters the network also contains some user defined parameters. For different data sets different values of the learning parameters η, ν and λ are chosen. For all the data sets, as explained earlier, τ is taken as 3, th as 0.05 and ω_{low} as 0.001. Table 4.4 gives the values of all user defined parameters of the network. Other than the learning coefficients, users can use the values suggested in Table 4.4. For the learning coefficient, as done in all gradient based learning schemes, either one can use a trial and error method or a scheme like the ones discussed in [77, 88, 137]. A simple workable solution is to keep a snap-shot of all learnable parameters before an epoch starts and use a high value (say 1) of the learning coefficient. If the average error after an epoch is found to increase, then the learning constant is decreased by an amount, say 10%, of the present value and the learning is continued after resetting the learnable parameters using the snap-shot.

4.6.3 Experimental Results

Results on Elongated

The number of fuzzy sets used for each feature for this problem is shown in Table 4.5 and the actual fuzzy sets used are depicted in Fig. 4.3. The initial architecture is shown in Table 4.6.

After only 100 iterations the number of misclassifications on the training set were reduced to 0. The values of $1 - e^{-\beta_p{}^2}$ after 100 iterations are depicted in Table 4.7. Table 4.7 shows that the network selects features 1 and 2 and rejects the 3rd feature. Consequently the network is pruned for redundant nodes. After pruning the network retains its performance, i.e., produces a misclassification of 0 on X_{tr}.

Initially the network had 36 antecedent nodes. Hence it had $36 \times 2 = 72$ rules. After

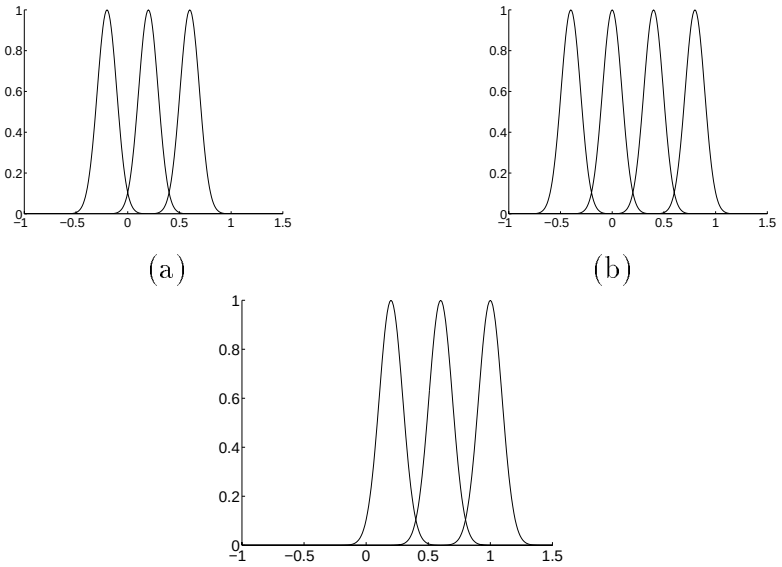


Figure 4.3: Fuzzy sets used for Elongated: (a) feature 1 (b) feature 2 (c) feature 3

Table 4.6: Initial architecture of the network used to classify Elongated

Layer 1	3
Layer 2	10
Layer 3	36
Layer 4	2

Table 4.7: Value of $1 - e^{-\beta_p{}^2}$ for different input features for Elongated

Features	1	2	3
$1 - e^{-\beta_p^2}$	0.51	0.44	0.01

pruning of the redundant nodes the number of antecedent nodes becomes 12; hence, at this stage the number of rules is $12 \times 2 = 24$. Next, the incompatible rules are pruned to obtain 12 rules. Finally, one rule is found to be *less used* and hence removed. Thus, the final architecture represents 11 rules as shown in Fig. 4.4. In Fig. 4.4 the 11 ellipses represent the 11 antecedent clauses. The co-ordinates of the center of an ellipse correspond to the centers of the two bell shaped fuzzy sets that form the antecedent clause. The major and minor axes of the ellipses are equal to twice the spreads of the respective fuzzy sets. For antecedent clauses of rules representing a particular class we use a particular type of line to draw the corresponding ellipses. For example, in Fig. 4.4, the continuous line is used to represent class 1 and the dotted line for class 2. The linguistic rules read from the network are shown in Table 4.8.

The final network produces a misclassification of 0 on X_{tr} as well as on X_{te}.

Results on Iris

Here we used 3 fuzzy sets for each of the four features. The fuzzy sets are shown in Fig. 4.5. The initial architecture for the network is shown in Table 4.9.

After 300 epochs the β_p values stabilized and the number of misclassifications produced on the training set X_{tr} was 3. The values of $1 - e^{-\beta_p{}^2}$ after 300 iterations are shown in Table 4.10, which suggests that only features 3,4 are important. Hence, we prune the redundant nodes. After pruning, the network *still* produces 3 misclassifications on X_{tr}.

In this case the initial network had 81 antecedent nodes resulting in $81 \times 3 = 243$ rules. After pruning of the redundant nodes the number of antecedent nodes becomes 9, hence at this stage the number of rules is $9 \times 3 = 27$. Next, the incompatible rules are pruned to obtain 9 rules. For Iris we found 4 *less used* rules and we removed them.

Table 4.8: The linguistic rules for Elongated.

Rule no.	Rule
1	if x_1 is CLOSE TO -0.2 and x_2 is CLOSE TO -0.4 then class 1
2	if x_1 is CLOSE TO -0.2 and x_2 is CLOSE TO 0.0 then class 2
3	if x_1 is CLOSE TO -0.2 and x_2 is CLOSE TO 0.4 then class 2
4	if x_1 is CLOSE TO -0.2 and x_2 is CLOSE TO 0.8 then class 2
5	if x_1 is CLOSE TO 0.2 and x_2 is CLOSE TO -0.4 then class 1
6	if x_1 is CLOSE TO 0.2 and x_2 is CLOSE TO 0.0 then class 2
7	if x_1 is CLOSE TO 0.2 and x_2 is CLOSE TO 0.4 then class 1
8	if x_1 is CLOSE TO 0.2 and x_2 is CLOSE TO 0.8 then class 2
9	if x_1 is CLOSE TO 0.6 and x_2 is CLOSE TO -0.4 then class 1
10	if x_1 is CLOSE TO 0.2 and x_2 is CLOSE TO 0.0 then class 1
11	if x_1 is CLOSE TO 0.2 and x_2 is CLOSE TO 0.4 then class 1

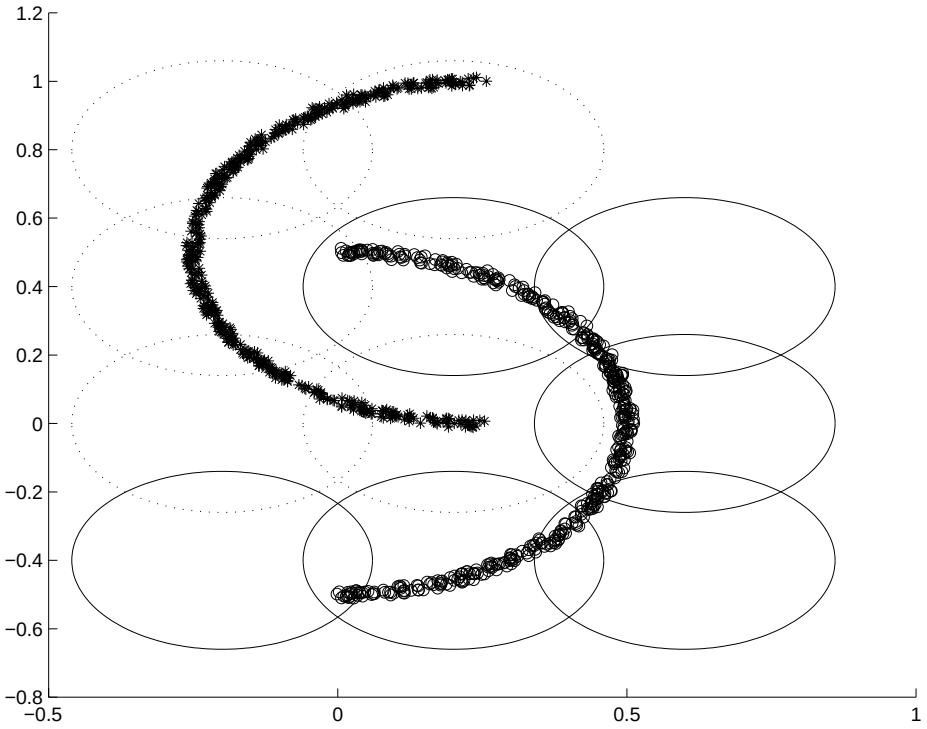


Figure 4.4: The rules for classifying Elongated.

Table 4.9: Initial architecture of the network used for Iris

Layer 1	3
Layer 2	12
Layer 3	81
Layer 4	3

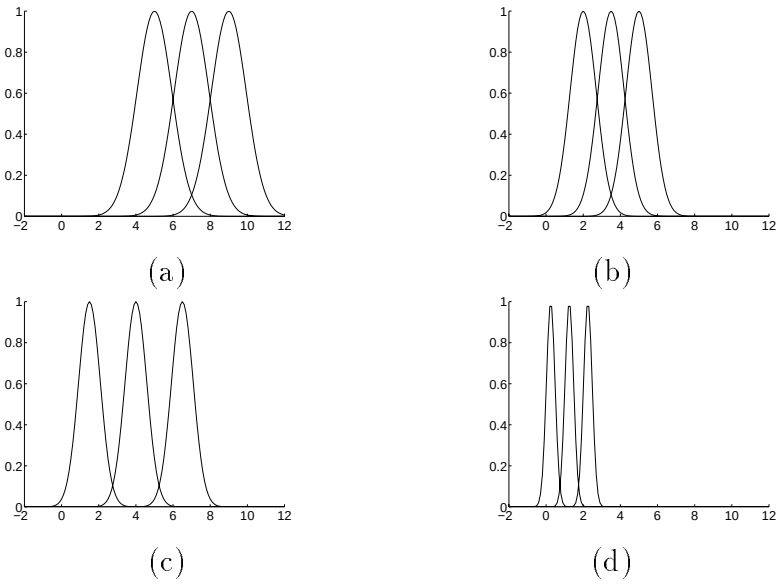


Figure 4.5: Fuzzy sets used for Iris: (a) feature 1 (b) feature 2 (c) feature 3 (d) feature 4

Table 4.10: Value of $1 - e^{-\beta_p{}^2}$ for different input features for Iris.

Features	1	2	3	4
$1 - e^{-\beta_p^2}$	0.00	0.00	0.52	0.51

The final architecture represented only 5 rules that are depicted in Fig. 4.6 and the corresponding linguistic rules are shown in Table 4.11. In Iris data features 3 and 4 represent petal length and petal width of iris flowers respectively. In Table 4.11 pl and pw represent the petal length and petal width. The final network again produces a misclassification of 3 on the training set X_{tr} and 1 on the test set X_{te}. This clearly suggests that pruning neither degrades the performance on the training data nor the generalization capability of the system.

We also trained our network with all the 150 points. In this case too we obtained just the same results in terms of features selected and misclassification, i.e., we obtained a misclassification of 4 on the entire data set.

There are many results available on Iris data in the literature. Table 4.12 (adopted from [179])shows the best resubstitution accuracy for some rule based classifiers. From Table 4.12 it is clear that FuGeNeSys and SuPFuNIS exhibit the best results using 5 rules. We obtain a resubstitution accuracy of 97.3% with 5 rules which is better than all others in Table 4.12 leaving out FuGeNeSys and SuPFuNIS. Note that, all results in Table 4.12 are obtained by using all 4 features, but our classifier uses only 2 features. In [38] Chiu uses only two features, i.e., features 3 and 4 to develop a classifier. The features considered by Chiu are the same as those selected by our classifier. Chiu reports a training error of 3 and a test error of 0 on a training-test partition of 120+30 using only three rules. This is of course better than the result obtained by us, but we do not know the training-test partition used in [38].

Results on Phoneme

Like Iris, here also we used three fuzzy sets for each feature. Figure 4.7 shows the membership functions used. In this case for each feature we used the same membership functions. The initial architecture of the network is presented in Table 4.13.

Table 4.11: Linguistic rules for Iris data.

Rule no.	Rule
1	if pl is CLOSE TO 1.5 and pw is CLOSE TO 0.25 then class1
2	if pl is CLOSE TO 4.5 and pw is CLOSE TO 1.25 then class2
3	if pl is CLOSE TO 6.5 and pw is CLOSE TO 1.25 then class3
4	if pl is CLOSE TO 4.5 and pw is CLOSE TO 2.25 then class3
5	if pl is CLOSE TO 6.5 and pw is CLOSE TO 2.25 then class3

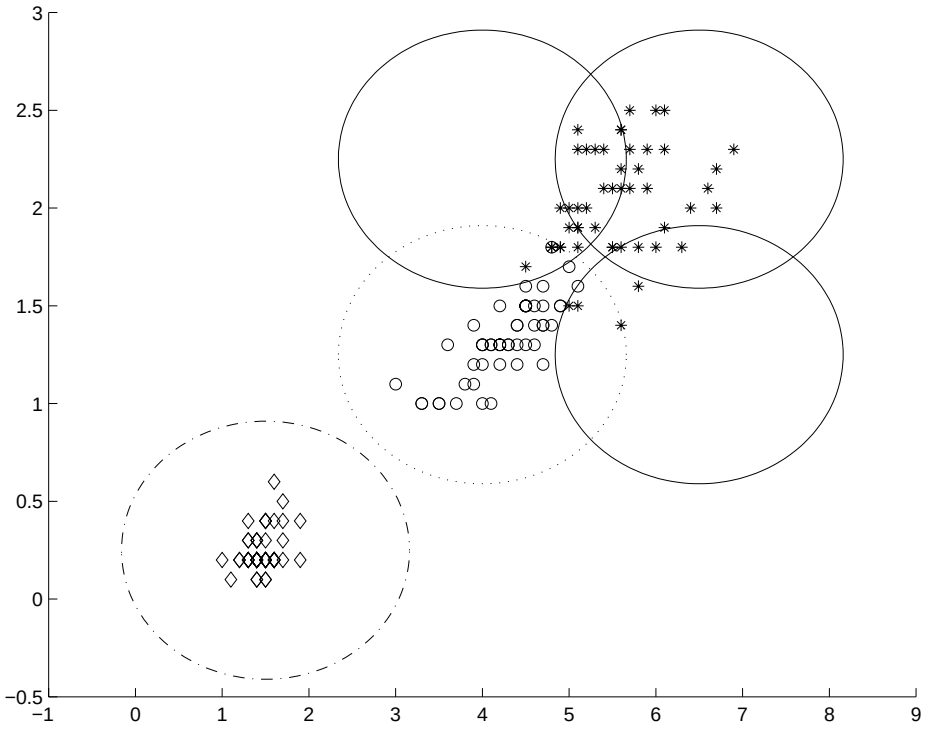


Figure 4.6: The rules for classifying Iris.

Table 4.12: Best resubstitution accuracy for Iris data for different rule based classifiers

Method	Rules	Resubstitution accuracy (%)
FuGeNeSys [207]	5	100
NEFCLASS [153]	7	96.7
ReFuNN [101]	9	95.3
EFuNN [106]	17	95.3
FeNe-I [70]	7	96.0
SuPFuNIS [179]	5	100

Table 4.13: Initial architecture of the network used to classify Phoneme

Layer 1	5
Layer 2	15
Layer 3	243
Layer 4	2

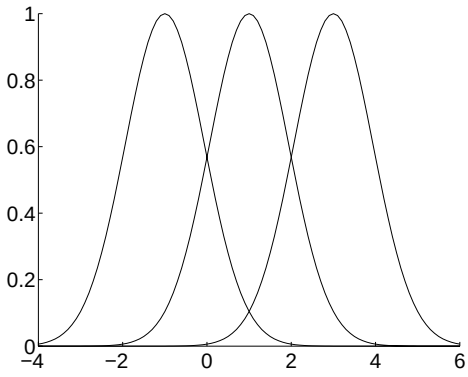


Figure 4.7: The fuzzy sets used for all the 5 features of Phoneme

Table 4.14: Value of $1 - e^{-\beta_p{}^2}$ for different input features for Phoneme.

Features	1	2	3	4	5
$1 - e^{-\beta_p^2}$	0.66	0.68	0.98	0.93	0.00

On termination of phase 1, the number of misclassifications on X_{tr} were reduced to 85. The values of $1 - e^{-\beta_p{}^2}$ after phase I training (Table 4.14) reveal that the network rejects only the 5th feature. The network is accordingly pruned for redundant nodes. In this case too, the network can retain its performance after pruning, i.e., produce the same misclassification of 85 on X_{tr}.

Pruning of redundant nodes reduces the number of antecedent clauses to 81. The removal of *incompatible rules* consequently yields 81 rules. Among these 81 rules 32 rules were *zero rules* and there was no *less used rules*. Therefore, the final network represented 49 rules. After phase 3 training the number of misclassifications on X_{tr} was 75(15%). This shows that tuning of membership function parameters enhances the performance of the classifier in this case. The misclassifications produced on X_{te} by the final network was 898(18.3%).

This data set has been extensively studied in [122] and the average misclassifications reported there on this data set using MLP are 15.40% on the training data and 19.63 % on the test data. Using Radial Basis Function networks, the average training and test error were 19.9% and 22.48% respectively [122]. Thus our results are quite comparable (in fact a little better) than the previously reported results.

Results on RS-Data

In all the previous results reported, we selected the initial architecture of the network arbitrarily. Such initial networks may not yield good results in case of complex class structures. Also blind selection of initial networks may make the network too large and learning on such networks may become computationally very expensive. This problem becomes more severe when the number of features and number of classes are large. We demonstrate this problem in this example. We shall also discuss a methodology for deciding on the initial network. Most neuro-fuzzy techniques reported in literature do

Table 4.15: Value of $1 - e^{-\beta_p{}^2}$ for different input features for RS-Data.

Features	1	2	3	4	5	6	7
$1 - e^{-\beta_p^2}$	0.99	0.00	0.99	0.41	0.28	0.00	0.98

not pay adequate attention on the setting up of the initial network. The methodology that we discuss here is not a very general one, but it worked well with the present data set. More investigations are required in this area to evolve a general guideline for choosing the initial network.

The data set in question has 7 input features and 8 classes. With 3 fuzzy sets for each input feature, the number of antecedent nodes becomes 2187. After 100 iterations the values of the feature modulators almost stabilized and the values (as shown in Table 4.15) suggest that features 2 and 6 are not important.

So, we discard these two unimportant features. Note that, the network is still quite big and also use of only three fuzzy sets for each feature may not be adequate. So we now try to exploit some tools of exploratory data analysis to get a better initial network. The rest of the analysis is done on the remaining 5 features, we call this data set as Reduced RS-Data. For each class of the data we run the fuzzy c means (FCM) algorithm [12] with fuzzifier 2 and number of clusters 3 (the FCM algorithm is discussed in an appendix, at the end of this chapter). The number of clusters is determined in an ad hoc manner, one may use some cluster validity index [12] to determine the "optimal" number of clusters. Thus, we obtain 3 prototypes for each class (in total 24 prototypes) in $\Re^5$. Let $\boldsymbol{p}_i = (p_{i1}, p_{i2}, p_{i3}, p_{i4}, p_{i5})^T$, where i=1 to 24, be the prototypes. Let $X_i \subset X_{tr} \subset \Re^5$ be the set of data points represented by the prototype $\boldsymbol{p}_i$. Here X_i is obtained from the final partition matrix of the FCM algorithm. Let $\boldsymbol{x}_j^i = (x_{j1}^i, ..., x_{j5}^i)^T$; $j = 1$ to $|X_i|$ be the points in X_i . For each X_i and $k = 1$ to 5, we calculate

$$\sigma_{ik} = \frac{1}{|X_i|} \left[\sum_{j=1}^{|X_i|} (x_{jk}^i - p_{ik})^2 \right]^{\frac{1}{2}}. \tag{4.23}$$

For each feature k (k=1 to 5), we take p_{ik} (i=1 to 24), as the center and σ_{ik} as the spread of a bell shaped fuzzy membership function. In this way, for each feature we obtain 24 fuzzy sets. For each feature k, if $|p_{ik} - p_{jk}| \leq 2.0, i \neq j$ and if $\sigma_{ik} \geq \sigma_{jk}$,

Table 4.16: Number of fuzzy sets for each feature for Reduced RS-Data

Feature 1	9
Feature 2	8
Feature 3	11
Feature 4	16
Feature 5	11

we discard the fuzzy set with center p_{jk} otherwise we discard the fuzzy set with center p_{ik}. This is quite a natural choice as the peaks of two adjacent fuzzy sets should be at least two gray levels apart. The final number of fuzzy sets for each feature that we arrived at after this process is shown in Table 4.16. Now we retain only the 132 antecedents which are supported by more than 3 data points. To investigate the effect of the choice of τ, we also consider the net where each node is supported by at least 5 data points. In this case we get 78 antecedents.

We train both these networks: one with 132 antecedents (i.e., $132 \times 8 = 1056$ rules) and the other with 78 antecedents (i.e., $78 \times 8 = 624$ rules), discarding all other antecedents at the onset of training. After pruning the incompatible rules, with 132 rules we obtain a training error of 20% and a test error of 16% and with 78 rules we obtain a training error of 22% and a test error of 19%. None of the rules in both cases were *Zero rules*, also pruning of *less used rules* was not applicable in this case, as the less used rules were already discarded during the setting up of the initial network. RS-Data was used by Kumar et al. [121] in a comparative study of different classification methods. The best results obtained by them using a fuzzy integral based scheme showed a misclassification of 21.85% on the test set. Our results with a reduced set of features out perform their results.

Table 4.17 summarizes the results on the various data sets used. It clearly shows good performance of the proposed system.

Table 4.17: Performance and rule reduction of the proposed system

Data set	Initial no. of rules	Final no. of rules	Misclassification on X_{tr}	Misclassification on X_{te}
Elongated	72	11	0%	0%
Iris	243	5	3%	2%
Phoneme	486	49	15%	18%
RS-Data	1056	132	20%	16%
	624	78	22%	19%

4.7 Conclusions and Discussion

In this chapter we proposed an extension of the scheme described in Chapter 3 for designing fuzzy rule based *classifier* in a neuro-fuzzy framework. The novelty of the system lies in its capacity to to select good features online. The network described here is also completely readable, i.e., one can easily read the rules required for the classification task from the network. The network starts with all possible rules and in the training process it only retains the rules required for classification, thus resulting in a smaller architecture of the final network. The final network has a lower running time than the initial network. The proposed method is tried on four data sets and in all four cases the network could select the good features and extract a small but adequate set of rules for the classification task. For one data set (Elongated) we obtained zero misclassification on both training and test sets and for all other data sets the results obtained are comparable to the results reported in the literature.

The proposed methodology is different in several respects from the network described in Chapter 3 for FA type problems. The main philosophy of feature selection here is almost the same as that for the FA network. But the learning rules and the pruning schemes have been modified to suite the structure of the classification rules. Additionally, a scheme for tuning the input membership functions have been proposed here; this modification can be applied in case of the FA network also.

Here too no specific guideline is given for selection of the initial input fuzzy sets. For further development of the methodology, some specialized tools of exploratory data

analysis may be used to decide upon the number and definition of the input fuzzy sets. To some extent we did it in the case of RS-Data. But the method suggested to analyze RS-Data needs further refinement and modifications to make it general in nature.

In the next chapter we present a methodology to do group feature selection using group feature modulators. The method is well suited for sensor selection.

Appendix: The Fuzzy c-Means Algorithm

Given a data set $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N\}$, a $c - partition$ of X is a $c \times N$ matrix $U = [\boldsymbol{u}_1 \boldsymbol{u}_2 ... \boldsymbol{u}_N] = [u_{ik}]_{c \times N}$ where $\boldsymbol{u}_k$ denotes the k^{th} column of U and u_{ik} denotes the membership of $\boldsymbol{x}_k$ to the i^{th} cluster. There can be three sets of c-partitions whose columns correspond to the three types of label vectors discussed in eqs. (4.1)-(4.3). The three types of $c - partitions$ are:

$$M_{pcN} = \left\{ U \in \Re^{cN} : \boldsymbol{u}_k \in N_{pc} \forall k; 0 < \sum_{k=1}^{n} u_{ik} \forall i \right\} \tag{4.24}$$

$$M_{fcN} = \{ U \in M_{pcN} : \boldsymbol{u}_k \in N_{fc} \forall k \} \tag{4.25}$$

$$M_{hcN} = \{ U \in M_{fcN} : \boldsymbol{u}_k \in N_{hc} \forall k \} \tag{4.26}$$

The fuzzy c-means (FCM) algorithm finds out a fuzzy $c - partition$ of X, i.e., it finds a $U \in M_{fcN}$. The FCM algorithm minimizes the following objective function:

$$J = \sum_{i=1}^{c} \sum_{k=1}^{N} u_{ik}^{m} ||\boldsymbol{x}_k - \boldsymbol{v}_i||^2. \tag{4.27}$$

In eq. (4.27) $\boldsymbol{v}_i \in \Re^s$ is the i^{th} point prototype. $m > 1$ is called the fuzzifier that controls the degree of fuzziness in U, and $||.||$ is an inner product induced norm, commonly, the Euclidean norm. The FCM algorithm aims to find that $\boldsymbol{V} = [\boldsymbol{v}_1 ... \boldsymbol{v}_c]$ and $U = [u_{ik}]$ which minimize J subject to the constraint that $U \in M_{fcN}$.

The most popular technique to solve this optimization problem is through grouped coordinate descent or *alternating optimization* [12]. The first order necessary conditions for U and $\boldsymbol{V}$ at a local minima of J are

$$u_{ik} = \left[\sum_{j=1}^{c} \left(\frac{||\boldsymbol{x}_i - \boldsymbol{v}_i||}{||\boldsymbol{x}_i - \boldsymbol{v}_j||} \right)^{\frac{2}{m-1}} \right]^{-1}, \quad \forall i, k \tag{4.28}$$

and

$$\boldsymbol{v}_i = \frac{\sum_{k=1}^{N} u_{ik}^m \boldsymbol{x}_k}{\sum_{k=1}^{N} u_{ik}^m}, \quad \forall i. \tag{4.29}$$

The FCM algorithm begins with a valid fuzzy c-partition U and using eq. (4.29) a new set of prototypes is computed. Again using these new prototypes a new partition matrix is computed using eq. (4.28). This is continued till the entries of the partition matrix stabilize. The same procedure can be carried out by initializing the prototypes instead of the partition matrix.

We mentioned earlier that the fuzzifier m controls the degree of fuzziness in the partition. As $m \to 1^+$, the algorithm tends to produce a hard partition, i.e., the columns of U becomes hard labels. And $\lim_{m\to\infty} u_{ik} = 1/c$ [9]. Thus, as the value of m increases the algorithm produces more fuzzy partitions.

Chapter 5

Online Sensor Selection Using Feed-Forward Networks[1]

5.1 Introduction

This chapter addresses the problem of feature selection in a different setting. Here we assume that the features available can be divided into a few groups. The motivation of the problem comes from the fact that now-a-days for a given problem we often obtain data from multiple sensors. For example, in an intelligent welding inspection system the sensors could be radiograph, acoustic emission, thermograph, eddy-current detector etc. The sensory information obtained from various sensors in the raw form may not always be useful. Hence, from a single sensory information one may generate/extract several features. For example, using the two dimensional histogram of the radiograph we can compute several features. If we use all these sensors, then the design cost and complexity of the hardware may be high. Moreover, the learning task also will become more difficult. So, the designer tries to reduce the number of sensors without hampering the system performance. Thus, the problem is selection of useful sensors where each sensor generates a set of features. Conventional feature selection methods select good features from all available features generated from all these sensors. But, our objective here is to discard the features obtained from redundant *sensors*, if any. In other words, we aim to discard sensors which are not necessary for solving a given problem.

[1] Parts of this chapter have been published in [28] and the whole of it have been communicated in [31].

This problem is different from feature selection. We call this problem as Group Feature Selection (GFS). Sensor selection is a special type of GFS, where each feature group corresponds to a sensor. This kind of grouping results in a natural partition of the total set of features according to their sensory origin. In this case, selecting good feature groups is equivalent to the selection of the good (relevant) sensors. Such group feature selection can thus help us to discard redundant sensors and consequently we can design systems with low hardware and computational costs. Sometimes, it can reduce the time required to make decisions, which is very important for many applications including medical diagnosis. There may exist other natural groupings among features too. For example, given an image, there could be features based on co-occurance matrix [74] and wavelet analysis. In this case the set of co-occurance based features can form one group while the wavelet based features can give another group. Here we consider the problem of feature group selection, instead of individual feature selection. Of course, individual feature selection is a special case of this group feature selection methodology. *To our knowledge this problem has not been addressed in the literature.*

In this study we use two connectionist schemes to deal with the problem of group feature selection. The first scheme uses a modified radial basis function network which we call Group Feature Selecting Radial Basis Function (GFSRBF) network and the other one involves a modified multilayer perceptron called the Group Feature Selecting Multilayer Perceptron (GFSMLP). In both methods, the user needs to specify the groupings that exist between the features. The networks are designed to discard the effect of the redundant/bad groups.

We emphasize that we are restricting ourselves to applications where data can be collected under controlled environment, like the intelligent welding system that we used to motivate sensor selection problem. In application areas where the environmental conditions can change drastically, the method is still applicable, but one needs to be careful about the data used for sensor selection. For example, consider the problem of channel selection for satellite imagery. The channel which is best for a cloudy day may not necessarily be the best when used in a sunny day. Our method can still select relevant sensors if the training set contains data captured in both cloudy and sunny conditions. However, collection of data generated under all conditions may be a difficult task depending on the application.

The rest of the chapter is organized as follows: In Section 5.2 we describe the GFSRBF

network where we discuss its structure, the learning rules and its universal approximation properties. Then in Section 5.3 we discuss the GFSMLP. Finally, in Section 5.4 we present results on some well known classification and function approximation problems. In Section 5.5 the chapter is concluded.

5.2 The Group Feature Selecting Radial Basis Function (GFSRBF) Network

Given an input data set $X = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\} \subset \Re^s$, a radial basis function network computes the function

$$\boldsymbol{F}^*(\boldsymbol{x}) = \sum_{i=1}^{n} \boldsymbol{w}_i \phi_i(\boldsymbol{x})$$

where the ϕ_i's are the basis functions. If we assume Gaussian type basis functions then

$$\phi_i(\boldsymbol{x}) = exp\left\{-\frac{||\boldsymbol{x} - \boldsymbol{\mu}_i||^2}{\sigma_i^2}\right\}. \tag{5.1}$$

In eq. (5.1) $\boldsymbol{\mu}_i$ and σ_i are the parameters related to the i^{th} basis function, commonly known as the center and spread respectively and $||.||$ is the Euclidean norm. Let us assume $\boldsymbol{x} = [x_1\ x_2\ ...\ x_s]^T$ and $\boldsymbol{\mu}_i = [\mu_{i1}\ \mu_{i1}\ ...\ \mu_{is}]^T$. Then we have

$$\phi_i(\boldsymbol{x}) = \prod_{j=1}^{s} exp\left\{-\frac{(x_j - \mu_{ij})^2}{\sigma_i^2}\right\}. \tag{5.2}$$

We assume that our data are generated by l sensors and from each of the sensors i we generated n_i $(i = 1, 2, ..., l)$ features, so $\sum_{i=1}^{l} n_i = s$. Let the features from each sensor i be denoted by a vector $\boldsymbol{a}^i$. Hence, we can say that $\boldsymbol{x} = [\boldsymbol{a}^1\ \boldsymbol{a}^2\ ...\boldsymbol{a}^l]^T$. Similarly, we can write the vector representing the center as $\boldsymbol{\mu}_i = [\boldsymbol{m}_i^1\ \boldsymbol{m}_i^2\ ...\ \boldsymbol{m}_i^l]$, where $\boldsymbol{m}_i^j$ and $\boldsymbol{a}^j$, $j = 1, 2, .., l$, have the same dimensionality. Equation (5.1) can now be rewritten as

$$\phi_i(\boldsymbol{x}) = \prod_{j=1}^{l} exp\left\{-\frac{||\boldsymbol{a}^j - \boldsymbol{m}_i^j||^2}{\sigma_i^2}\right\}. \tag{5.3}$$

In eq. (5.3) each basis function ϕ_i is represented as the product of component Gaussian functions $C_i^j,\ j = 1, 2, ..., l$,where

$$C_i^j = exp\left\{-\frac{||\boldsymbol{a}^j - \boldsymbol{m}_i^j||^2}{\sigma_i^2}\right\}. \tag{5.4}$$

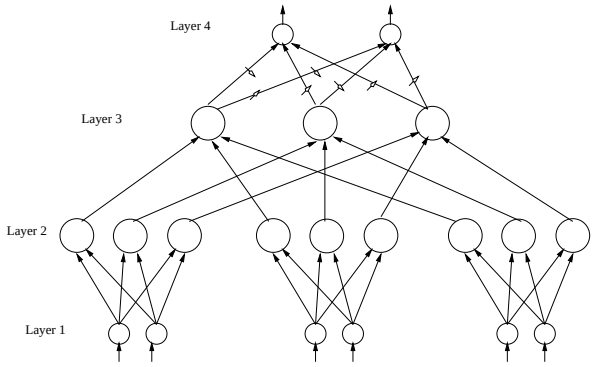


Figure 5.1: The GFSRBF network structure

Thus, each C_i^j takes as input the vector representing the features from a specific sensor j. So, the output of C_i^j is related to the inputs obtained from the j^{th} sensor. Our objective is to eliminate the effect of the features generated from bad/redundant sensors. For the time being let us *pretend* that we know the bad or redundant groups. Hence, we aim to design the component functions in such a manner that a component function C_i^k corresponding to a bad/redundant group will always take the value of unity (1) irrespective of the input $\boldsymbol{a}^k$. If we can do so, then C_i^k will *never* contribute anything to the total process i.e., to $\phi_i(\boldsymbol{x})$. To achieve this we design the the component functions as:

$$C_i^j = \left[exp\left\{ -\frac{||\boldsymbol{a}^j - \boldsymbol{m}_i^j||^2}{\sigma_i^2} \right\} \right]^{1-e^{-\beta_j^2}} . \tag{5.5}$$

Clearly, in eq. (5.5), if we set $|\beta_j| \approx 0$, then $C_i^j \approx 1$, thereby it can eliminate the effect of the sensor j in each of the basis function ϕ_i irrespective of values of $\boldsymbol{a}^j$. On the other hand, if $|\beta_j|$ is very large then C_i^j in eq. (5.5) reduces to C_i^j in eq. (5.4) resulting in no change of the role of the basis functions. But, how do we know which group is good and which is bad? In other words, how do we set the values of the β_js. The solution lies in the training process. We treat each β_j as an adjustable parameter and learn its appropriate value along with other parameters through training. With these preliminaries, we next discuss the network structure of GFSRBF.

5.2.1 The Network Structure

GFSRBF network is a four layer feed-forward network as shown in Fig. 5.1. The network in Fig. 5.1 is designed for data obtained from three sensors, where two features are computed from each sensor. Also it assumes three basis functions and two output nodes. The use of three basis functions has nothing to do with the number of sensors. We denote our training data set as $T = \{(\boldsymbol{x}, \boldsymbol{y}) | \boldsymbol{x} \in \Re^s, \boldsymbol{y} \in \Re^t\}$. Each point $\boldsymbol{x}$ has s features which can be grouped into l groups. The division could be made based on sensors or some other criteria. To avoid ambiguity, here we deviate from our earlier notation of network outputs. In our subsequent discussions we denote the output of the i^{th} layer by $z^{(i)}$. We now discuss the general structure of the network layer by layer.

Layer 1: This layer is called the *input layer*. The number of nodes in this layer is equal to the dimensionality of the input data, here it is s.

Layer 2: This is called the *component function layer*, and this layer is responsible for the feature selection task. If the network contains m basis functions then this layer will contain $l \times m$ nodes. Thus, this layer contains the component function for each basis function for all feature groups. Let $z_{ij}^{(2)}$ denote the output of the component function related to the i^{th} basis function and the j^{th} group - the superscript (2) denotes the layer number. Then we have

$$z_{ij}^{(2)} = \left[exp\left\{ -\frac{||\boldsymbol{a}^j - \boldsymbol{m}_i^j||^2}{\sigma_i^2} \right\} \right]^{1-e^{-\beta_j^2}} . \tag{5.6}$$

In eq. (5.6) β_j is an adjustable parameter related to the j^{th} feature group. We call $\gamma_j = 1 - e^{-{\beta_j}^2}$, as the feature group modulator for the j^{th} feature group. When $\gamma_j \to 0$, $z_{ij}^{(2)} \to 1$. Thus, for a bad group of features *if* $\gamma_j \to 0$, then the effect of the j^{th} group gets eliminated. The training procedure (to be discussed later) will start with very low values of γ_j, i.e., with very small values of β_j^2 for all j and thereby making all feature groups unimportant. As the training process continues, the network allows features from only those groups which can lower the sum of square error significantly.

Layer 3 : This layer is called the *basis function layer*, the number of nodes in this layer depends on the number of basis functions used (required) for solving the problem. The

output of the i^{th} basis function is

$$z_i^{(3)} = \prod_{j=1}^{l} z_{ij}^{(2)}. \tag{5.7}$$

Layer 4 : This is called the *output layer*. The number of nodes in this layer is equal to the number of classes present in the data or the dimensionality of the output vector. The nodes in this layer are fully connected to the nodes of layer 3. The connection between node i in layer 4 and node j in layer 3 bears a learnable weight w_{ij}. Like a conventional RBF network the output of the i^{th} node in this layer is given by

$$z_i^{(4)} = \sum_{j=1}^{m} w_{ij} z_j^{(3)} \tag{5.8}$$

where m is the number of nodes in layer 3. When the network is used for classification problems then the target output of an output node lies in [0,1]. And, this is true for all kinds of class labels that the data may have (probabilistic, possibilistic, fuzzy or hard). But eq. (5.8) shows that $z_i^{(4)}$ is unbounded as the learnable weight $w_{ij}, \forall i, j$ can take any value. Consequently, for classification tasks we modify the output of this node by adding a standard sigmoidal nonlinearity to this node function, so that the learning process becomes more simple. The output of node i in this layer is then computed as

$$z_i^{(4)} = \frac{1}{1 + exp(-\sum_{j=1}^{m} w_{ij} z_j^{(3)})}. \tag{5.9}$$

So, for regression (function approximation) type of applications, nodes in the 4^{th} layer use eq. (5.8) while for classifier applications eq. (5.9) is used. Next we discuss the parameter updating strategies for both cases.

5.2.2 The Learning Rules

We assume that there are t outputs and the training data contain points in $\Re^s$ along with its associated output in $\Re^t$. In case of classifiers the output $\boldsymbol{y}$ is a label vector in $[0,1]^t$. Let the output associated with a data point $\boldsymbol{x}$ be $\boldsymbol{y} = [y_1\ y_2\ ...\ y_t]^T$. Thus we can define the instantaneous error for a data point $\boldsymbol{x}$ as

$$E_{\boldsymbol{x}} = \frac{1}{2} \sum_{i=1}^{t} (z_i^{(4)} - y_i)^2. \tag{5.10}$$

For our further discussion without loss of generality we omit the subscript $\boldsymbol{x}$ and call the error term as E. The error function depends on the weights, w_{ij}s, connecting nodes of layer 2 and layer 3, the parameters of the basis functions and the group feature modulators β_js. We shall consider fixed parameters for the basis functions, i.e., basis functions with fixed centers and spreads. We use the gradient descent technique to update the weights and the feature group modulators β_js. Thus, the update equations for w_{ij} and β_j are

$$w_{ij}(n+1) = w_{ij}(n) - \eta \left(\frac{\partial E}{\partial w_{ij}(n)} \right) \tag{5.11}$$

and

$$\beta_j(n+1) = \beta_j(n) - \nu \left(\frac{\partial E}{\partial \beta_j(n)} \right). \tag{5.12}$$

Here η and ν are predefined learning rates. For the *classification* network, i.e., for the network with sigmoidal activation functions in the output units, we get

$$\frac{\partial E}{\partial w_{ij}} = z_j^{(3)} z_i^{(4)} (z_i^{(4)} - y_i)(1 - z_i^{(4)}), \tag{5.13}$$

and for the *regression* network we get

$$\frac{\partial E}{\partial w_{ij}} = (z_i^{(4)} - y_i) z_j^{(3)}. \tag{5.14}$$

For both networks

$$\frac{\partial E}{\partial \beta_j} = -2\beta_j e^{-\beta_j^2} \sum_{k=1}^{m} \delta_k^{(3)} z_k^{(3)} \left(\frac{||\boldsymbol{a}^j - \boldsymbol{m}_k^j||^2}{\sigma_k^2} \right) \tag{5.15}$$

where,

$$\delta_k^{(3)} = \frac{\partial E}{\partial z_k^{(3)}} = \sum_{i=1}^{c} (z_i^{(4)} - y_i) w_{ik}. \tag{5.16}$$

Note that, along with w_{ij} and β_j, the other parameters $\boldsymbol{m}_k^j$ and σ_k could also be learnt using gradient descent technique. Since our objective here is to demonstrate the feature (sensor) selection ability of the proposed scheme, we do not update $\boldsymbol{m}_k^j$ and σ_k, but we use judicious choices for them as discussed next.

5.2.3 Selection of Centers and Spreads

Selecting the parameters for the radial basis functions forms an important part in designing RBF networks. Generally there are two common strategies used in practice: (1) The parameters for the basis functions are chosen apriori and are kept fixed. Only the weights between the hidden and output layers get updated during learning. (2) All parameters are optimized by a gradient descent (or a similar) technique. As discussed earlier we follow the first strategy here. Initial centers and spreads are determined by the Fuzzy c Means clustering algorithm (FCM) [12] (discussed in the appendix of Chapter 4). We use the fuzzifier $m = 2$ in all reported results. Once we obtain the cluster centers $\boldsymbol{\mu}_i$ by the FCM algorithm, we calculate the spread σ_i of the i^{th} basis function as $||\boldsymbol{\mu}_i - \boldsymbol{\mu}_j||$, where $\boldsymbol{\mu}_j$ is the center of the basis function nearest to $\boldsymbol{\mu}_i$. In other words

$$\sigma_i = min_{j \neq i} ||\boldsymbol{\mu}_i - \boldsymbol{\mu}_j||. \tag{5.17}$$

5.2.4 A Threshold for the Feature Attenuators

The final values of the group feature attenuators (γ_j) of a trained GFSRBF network represent certain scaling parameters for each sensor. Based on these values, one can decide the importance of the sensors. A low value of γ_j, indicates that sensor j is less important and a high value indicates a high importance of the sensor. In the limit, $\gamma_j = 0$, represents that the sensor j is totally redundant while, $\gamma_j = 1$ suggests that sensor j is very important. But, as γ_js are modeled and updated, it can take any value in [0,1]. Here we try to find a threshold th for γ_j, such that if γ_j takes values less than th we can discard sensor j.

From eq. (5.6), we get

$$z_{ij}^{(2)} = exp\left[-\gamma_j \frac{||\boldsymbol{a}^j - \boldsymbol{m}_i^j||^2}{{\sigma_i}^2}\right]. \tag{5.18}$$

We call a group of features redundant/bad, if all component basis functions (CBF) related to that feature group produce a response almost equal to one for all points. Specifically, we consider a feature group to be bad, if all CBFs produce a response greater than 0.95 even for points which are as far as 2σ distance away from the center of the associated CBF. Thus, we select that value of γ as the threshold which makes

the right hand side of eq. (5.18) equal to $0.95(\approx 1)$ when $||\boldsymbol{a}^j - \boldsymbol{m}_i^j||$ is replaced by 2σ. So, we obtain

$$e^{-4\gamma_j} = 0.95. \tag{5.19}$$

Equation (5.19) gives $\gamma_j = 0.0128$. Hence we can safely discard a feature group with attenuation (γ) less than 0.01.

5.2.5 Universal Approximation Property of GFSRBF

The universal approximation property of RBF is well known. If $\gamma_j = 1$ then GFSRBF reduces to RBF. But during the training process, γ_j usually takes values in [0,1]. Therefore, it is necessary to check the universal approximation property of GFSRBF. Unless, GFSRBF has the universal approximation property, it may not be able to learn the input-output relation for all functions and thus may not be able to do the group feature selection task. So, we check this property here.

We consider the GFSRBF network for function approximation (i.e., the one without the sigmoidal non-linearity in the output node) with a single output. The proof can be easily extended for the multiple output case.

Definition: Let $X \subset \Re^s$ and G be a family of functions on X with values in $\Re$. Suppose that for all $\boldsymbol{x}_1, \boldsymbol{x}_2 \in X$, such that $\boldsymbol{x}_1 \neq \boldsymbol{x}_2$, there is an $f \in G$ such that $f(\boldsymbol{x}_1) \neq f(\boldsymbol{x}_2)$, then we say that G is a separating family of functions on X [80].

Let $\boldsymbol{x} = (\boldsymbol{a}^1, \boldsymbol{a}^2, ..., \boldsymbol{a}^l)$. We define a function family Φ as

$$\Phi = \left\{ \prod_{j=1}^{l} \left[exp\left\{ -\frac{||\boldsymbol{a}^j - \boldsymbol{m}^j||^2}{\sigma^2} \right\} \right]^{\gamma_j} \middle| \boldsymbol{a}^j, \boldsymbol{m}^j \in \Re^{n_j}, \sum_{j=1}^{l} n_j = s, \gamma_j \in [0,1], \sigma \in \Re \right\}.$$

Next we prove a few lemmas concerning the function family Φ.

Lemma 1: Φ is a separating family.

Proof : For any $\boldsymbol{x}_1 = (\boldsymbol{a}_1^1, \boldsymbol{a}_1^2, ..., \boldsymbol{a}_1^l)$ and $\boldsymbol{x}_2 = (\boldsymbol{a}_2^1, \boldsymbol{a}_2^2, ..., \boldsymbol{a}_2^l) \in X$, if $\boldsymbol{x}_1 \neq \boldsymbol{x}_2$ then there exists an i, $0 < i \leq l$ such that $\boldsymbol{a}_1^i \neq \boldsymbol{a}_2^i$. Without loss of generality we assume $\boldsymbol{a}_1^1 \neq \boldsymbol{a}_2^1$. Pick any $\phi \in \Phi$ with $\gamma_j \neq 0$. If $\phi(\boldsymbol{x}_1) \neq \phi(\boldsymbol{x}_2)$ then done. If $\phi(\boldsymbol{x}_1) = \phi(\boldsymbol{x}_2)$, let

$$\phi(\boldsymbol{x}_1) = \left[exp\left\{ -\frac{||\boldsymbol{a}_1^1 - \boldsymbol{m}^1||^2}{\sigma^2} \right\} \right]^{\gamma_1} \prod_{j=2}^{l} \left[exp\left\{ -\frac{||\boldsymbol{a}_1^j - \boldsymbol{m}^j||^2}{\sigma^2} \right\} \right]^{\gamma_j}$$

and

$$\phi(\boldsymbol{x}_2) = \left[exp\left\{-\frac{||\boldsymbol{a}_2^1 - \boldsymbol{m}^1||^2}{\sigma^2}\right\}\right]^{\gamma_1} \prod_{j=2}^{l}\left[exp\left\{-\frac{||\boldsymbol{a}_2^j - \boldsymbol{m}^j||^2}{\sigma^2}\right\}\right]^{\gamma_j}.$$

If,

$$\left[exp\left\{-\frac{||\boldsymbol{a}_1^1 - \boldsymbol{m}^1||^2}{\sigma^2}\right\}\right]^{\gamma_1} = \left[exp\left\{-\frac{||\boldsymbol{a}_2^1 - \boldsymbol{m}^1||^2}{\sigma^2}\right\}\right]^{\gamma_1},$$

then $\boldsymbol{m}^1$ is equidistant from both $\boldsymbol{a}_1^1$ and $\boldsymbol{a}_2^1$. Let $\hat{\boldsymbol{m}}^1$ be such that it is not equidistant from $\boldsymbol{a}_1^1$ and $\boldsymbol{a}_2^1$. As, $\boldsymbol{a}_1^1 \neq \boldsymbol{a}_2^1$, such an $\hat{\boldsymbol{m}}^1$ always exists. In ϕ, replace $\boldsymbol{m}^1$ by by $\hat{\boldsymbol{m}}^1$ and call the new function $\hat{\phi}$. Then, $\hat{\phi}(\boldsymbol{x}_1) \neq \hat{\phi}(\boldsymbol{x}_2)$.

If,

$$\left[exp\left\{-\frac{||\boldsymbol{a}_1^1 - \boldsymbol{m}^1||^2}{\sigma^2}\right\}\right]^{\gamma_1} \neq \left[exp\left\{-\frac{||\boldsymbol{a}_2^1 - \boldsymbol{m}^1||^2}{\sigma^2}\right\}\right]^{\gamma_1},$$

then, select $\hat{\boldsymbol{m}}^1$ such that it is equidistant from both $\boldsymbol{a}_1^1$ and $\boldsymbol{a}_2^1$ so that, $\hat{\phi}(\boldsymbol{x}_1) \neq \hat{\phi}(\boldsymbol{x}_2)$. Thus, Φ is a separating family.

Lemma 2: Φ contains the function 1.

Proof: A function $\phi_1 \in \Phi$ with $\gamma_j = 0, \forall j$ is the function 1.

Lemma 3: If $\phi \in \Phi$ then $\phi^n \in \Phi$ for any $n \in R$.

Proof: Consider

$$\phi(\boldsymbol{x}) = \prod_{j=i}^{l}\left[exp\left\{-\frac{||\boldsymbol{a}_1^j - \boldsymbol{m}^j||^2}{\sigma^2}\right\}\right]^{\gamma_j}$$

then,

$$(\phi(\boldsymbol{x}))^n = \prod_{j=i}^{l}\left[exp\left\{-\frac{||\boldsymbol{a}_1^j - \boldsymbol{m}^j||^2}{\frac{\sigma^2}{n}}\right\}\right]^{\gamma_j}.$$

Hence, $(\phi(\boldsymbol{x}))^n \in \Phi$.

We shall now prove the universal approximation property of GFSRBF using Stone-Wierstrass theorem.

Stone-Wierstrass Theorem [80]: Let X be a non-void compact set and G be a separating family of functions on X containing the function 1, then for any continuous function $f(\boldsymbol{x})$ defined on X, and for any $\epsilon > 0$, there exists a polynomial $p(\mathbf{x}) = \sum_{j=1}^{n} w_j(g_j(\boldsymbol{x}))^{n_j}$ where $g_j(x) \in G$, such that

$$sup\{|f(\boldsymbol{x}) - p(\boldsymbol{x})||\boldsymbol{x} \in X\} < \epsilon.$$

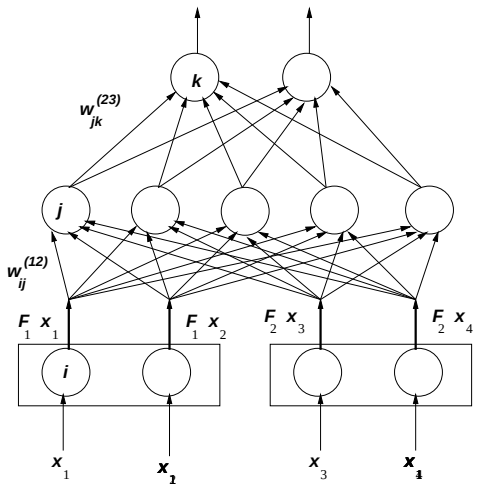


Figure 5.2: MLP with group feature selection (GFSMLP)

Let us denote the network output by $GFSRBF(\boldsymbol{x})$, for an $\boldsymbol{x} = (\boldsymbol{a}^1, \boldsymbol{a}^2, ..., \boldsymbol{a}^l) \in \Re^s$. In other words

$$GFSRBF(\boldsymbol{x}) = \sum_{i=1}^{n} w_i \prod_{j=1}^{l} \left[exp\left\{ -\frac{||\boldsymbol{a}^j - \boldsymbol{m}_i^j||^2}{\sigma_i^2} \right\} \right]^{\gamma_j}.$$

Theorem : For any given continuous function f on the compact set $X \subset \Re^s$ and arbitrary $\epsilon > 0$,

$$sup\left\{ |f(\boldsymbol{x}) - GFSRBF(\boldsymbol{x})| \,\middle|\, \boldsymbol{x} = (\boldsymbol{a}^1, \boldsymbol{a}^2, ..., \boldsymbol{a}^l) \in X \right\} < \epsilon$$

Proof: It directly follows from Lemmas 1-Lemma3, and the Stone Wierstrass theorem.

5.3 A Group Feature Selecting Multilayer Perceptron

A multilayer perceptron (MLP) can also be modified to do group feature selection. A feature selecting MLP was proposed by Pal and Chintalapudi [166] which used sigmoidal attenuation functions in the input layer. We generalize their idea to propose GFSMLP, a modified form of MLP for group feature selection.

Here the philosophy is different from that used in case of radial basis function network. Let F_l be an attenuator function attached to the l^{th} group of features. Each feature x_i of the l^{th} group gets multiplied by the attenuator function F_l before it gets into the

network. If $F_l = 0$, then no feature of the l^{th} group will get into the network; while if $F_l = 1$, then every feature of the l^{th} group enters the network unattenuated. For intermediate values of F_l, transformed values of the features enter into the network. Figure 5.2 shows the architecture of a group feature selecting MLP (GFSMLP) with just one hidden layer, and two groups of features with attenuation functions F_1 and F_2. Each attenuation function F_l should be such that, it has a tunable parameter β_l and $F_l \in [0, 1]$. To facilitate the learning of β_l, F_l should be differentiable. Moreover, F_l should be such that over a reasonably large interval (a, b), as β_l goes from a to b, F_l should either monotonically increase from 0 to 1, or monotonically decrease from 1 to 0. There can be many choices for F_l, we take $F_l = e^{-\beta_l^2}$. β_l is a parameter related to the l^{th} group of features. If $\beta_l \to 0$, $F_l \to 1$ and if $\beta_l \to \pm\infty$, $F_l \to 0$. The objective here is to select appropriate values of β_ls through training such that $F_l \to 1$, if l is associated with a useful group of features and $F_l \to 0$, if the l^{th} group is a *bad* or *redundant* group. The parameters β_l can be learnt by the backpropagation algorithm along with other parameters of a multilayer perceptron.

Let us consider a network with sigmoidal activation functions and a single hidden layer. For each input vector $\boldsymbol{x} = (x_1, x_2, ..., x_s)^T \in \Re^s$, let S_l denote the set of features which belongs to the l^{th} group. Let $z_i^{(1)}$, $z_i^{(2)}$, $z_i^{(3)}$ be the outputs of the i^{th} nodes of the input, hidden, output layers, respectively. Thus, for input $\boldsymbol{x}$, if $x_i \in S_l$, the output of the i^{th} node in the input layer would be

$$z_i^{(1)} = x_i F_l. \tag{5.20}$$

Let $w_{jk}^{(23)}$ be the weight of the link connecting the j^{th} node of the hidden layer with the k^{th} node of the output layer. Similarly, $w_{ij}^{(12)}$ denotes the weight connecting the i^{th} node of the input layer with the j^{th} node of the hidden layer. Thus,

$$z_j^{(2)} = \frac{1}{1 + e^{-\sum_i z_i^{(1)} w_{ij}^{(12)}}} \tag{5.21}$$

and

$$z_k^{(3)} = \frac{1}{1 + e^{-\sum_j z_j^{(2)} w_{jk}^{(23)}}}. \tag{5.22}$$

For an input $\boldsymbol{x}$ if the target is $\boldsymbol{y}$, then we define an error E as

$$E = \frac{1}{2} \sum_k (z_k^{(3)} - y_k)^2. \tag{5.23}$$

We define

$$\delta_i^{(q)} = \frac{\partial E}{\partial z_i^{(q)}}, \qquad q = 1, 2, 3. \tag{5.24}$$

Thus, the update equations for the two sets of weights can be derived as

$$w_{jk}^{(23)}(n+1) = w_{jk}^{(23)}(n) - \eta \delta_k^{(3)} z_j^{(2)} z_k^{(3)} (1 - z_k^{(3)}) \tag{5.25}$$

and

$$w_{ij}^{(12)}(n+1) = w_{ij}^{(12)}(n) - \eta z_j^{(2)} (1 - z_j^{(2)}) z_i^{(1)} \delta_j. \tag{5.26}$$

In the above equations, η is a predefined learning constant. The update equation for the feature attenuator of the l^{th} group, β_l, can be derived as

$$\beta_l(n+1) = \beta_l(n) + \nu \sum_{i \in S_l} 2\delta_i^{(1)} z_i^{(1)} \beta_l(t). \tag{5.27}$$

Here, ν is also a predefined learning constant. The β_ls are so initialized that at the beginning of training no feature group is important (i.e., no feature gets into the network). As training continues, the β_ls are changed in such a manner that for each important group $F_l \rightarrow 1$.

Although, we have shown the derivation for an MLP with only one hidden layer, its extension to MLPs with more than one hidden layer is straightforward. Next we discuss that the universal approximation property of MLP is retained in GFSMLP.

5.3.1 Universal Approximation Property of GFSMLP

The universal approximation property of MLP with sigmoidal activation functions is well known [84]. Here we show that adding attenuation functions in the input layer does not affect the universal approximation property of an ordinary MLP.

For convenience we assume one feature in one group. The net input to neuron j in the hidden layer is

$$net_j = \sum_i w_{ij}(x_i F_i) \tag{5.28}$$

where F_i is the attenuator for feature i. Note that, if the groups contain more than one feature then the attenuators for features in the same group will be the same. We

can consider the attenuators as a part of the weights connecting the input to the first hidden layer. Thus we can say

$$W_{ij} = w_{ij} F_i. \tag{5.29}$$

and

$$net_j = \sum_i W_{ij} x_i \tag{5.30}$$

So the net input to a node in the first hidden layer remains of the same form as that of a conventional MLP. But, the weight W_{ij} is composed of two parts w_{ij} and F_i where both w_{ij} and F_i are adjustable, w_{ij} is unrestricted in sign and magnitude and $F_i \in [0, 1]$. For any trained MLP if we consider W_{ij}s to be the weights connecting the input and the first hidden layers then a decomposition as in eq. (5.29) is always possible, the trivial one being $F_i = 1, \forall i$. Thus the GFSMLP is equivalent to an ordinary MLP, and hence the universal approximation property would be retained. But finding such a decomposition of weights for feature selection from a trained ordinary MLP would be difficult; so we impose a nice structure on the weights through the concept of attenuators to facilitate learning and feature selection.

5.4 Results

We provide here experimental results on five data sets: **Chem**, **Iris**, **RS-Data**, **Wine**, **Breast-Cancer**.

The Chem data [222] set is used to test the function approximation capability of the network. The description of this data set can be found in Section 3.7.

The remaining data are on classification problem. Here, in addition to the Iris data and RS-Data (described in Section 4.6.1) we use two more data sets, Wine and Breast-Cancer. The Wine data set [17] consists of 178 points in 13 dimension distributed in 3 classes. 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. The Breast-Cancer data [17] consist of 684 points in 9 dimension distributed in 2 classes (malignant and benign).

In the following subsections we discuss, in detail, the results obtained by GFSRBF and

Table 5.1: Values of γ_j in GFSRBF for Chem data (considering 3 groups of features)

Group 1 $\{u_1, u_2\}$	Group 2 $\{u_3\}$	Group 3 $\{u_4, u_5\}$
0.00	1.00	0.00

GFSMLP networks on these data sets. As stated earlier, the attenuation parameters for each network are initialized with the aim, that at the onset of training the network considers all feature groups to be unimportant. Thus, for GFSRBF we set $\beta_j = 0.001$ which makes $\gamma_j \approx 0, \forall j$, where j represents a feature group. And in GFSMLP we set $\beta_j = 3$ thus making $F_j = 0.0001, \forall j$. Both for GFSRBF and GFSMLP we consider 0.01 as a threshold for the feature attenuators, i.e., if the value of γ_j corresponding to feature group j is less than 0.01 then we discard that feature group.

5.4.1 Chem

In this example we demonstrate the feature selection capability of our network for the function approximation task. As stated earlier, the data set consists of 5 input features. The five input features can be divided into 3 groups with respect to the type of information. The monomer concentration (u_1) and change of monomer concentration (u_2) can constitute one group, the monomer flow rate (u_3) as the second group and the temperature parameters (u_4 and u_5) can form the third group. With 15 basis functions we find that the GFSRBF network accepts only the second group of features, i.e., only u_3 is important for the task (see Table 5.1). Figure 5.3 shows the plot of the output y with u_3. Figure 5.3 reveals a very strong correlation between u_3 and y(correlation coefficient = 0.9984!). The Chem data set was used by Lin and Cunningham in [140]. To evaluate a system they used a performance index PI defined in eq. (3.27). They got a PI of 0.0022 on Chem by using features u_1, u_2, u_3. The PI in our case was 0.0043. In Chapter 3, using the neuro-fuzzy system we obtained a PI of 0.0021.

A close look into the Chem data shows that the values of feature 3 numerically dominate all other features. Table 5.2 shows the ranges of the input features as well as of the output. Since, a radial basis function type network computes the Euclidean norm, it is

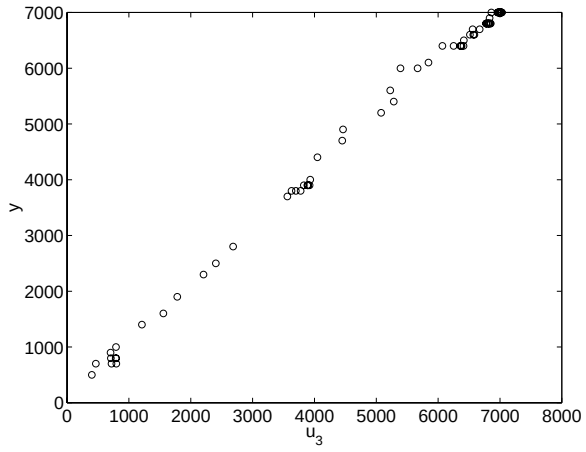


Figure 5.3: Plot of y and u_3

Table 5.2: Range of feature values for Chem data

Features	Minimum	Maximum
u_1	4.47	6.80
u_2	-0.29	0.17
u_3	401.00	7032.00
u_4	-0.40	0.20
u_5	-0.10	0.30
output	500.00	7000.00

quite natural that features with larger numerical values will dominate the output of the basis functions. Therefore, u_3 will have the strongest influence on the network behavior. Moreover, u_3 has a strong positive correlation with the output y. Consequently, the network picks up u_3. But previously it has been reported that the features u_1 and u_2 also have some effect on the output [222]. Our network cannot detect that, and as a result we get a reasonable (but not very good) performance as suggested by the PI value. This is not a problem of the model or of the philosophy being used, but is due to very wide variance of different features. To establish this fact we normalize feature 3 (u_3) and the output (y) so that each of these two lies in [0,1]. We call this new data set as Normalized-Chem.

With Normalized-Chem we run GFSRBF with different number of basis functions and different initializations of the FCM algorithm. The FCM outputs are used to compute the centers and spreads of the basis functions. For a fixed number of basis functions, various FCM initializations do not significantly change the feature attenuators and the performance. Table 5.3 gives the average performance of GFSRBF on Normalized-Chem data for different number of basis functions. With a fixed architecture (using a given number of basis functions), 5 independent runs are made with different initializations, and the average value of PI and the number of groups selected are shown in Table 5.3. The frequency of each of the groups selected in these 30 runs are shown in Fig. 5.4. From Fig. 5.4, we see that the network rejects the third group for most of the runs. From Table 5.3 we see that the best performance is obtained by using 3 basis functions and 15 basis functions. In case of 3 basis functions the network considers the first and the second group of features, but for 15 basis functions it considers only the second group. In both cases the networks result in almost the same average performance. This proves that changing the number of basis functions changes the learning machine, so the importance of the features may also vary. The best performance obtained by GFSRBF is better than that obtained by Lin and Cunningham [140] and also our neuro fuzzy system described in Chapter 3.

The performance of GFSMLP on Normalized-Chem data for different hidden nodes is shown in Table 5.4. Here too the average performance in terms of PI and the number of groups selected are shown for 5 independent runs for each architecture. Table 5.4 shows that the GFSMLP selects two groups for all runs. Also the two groups selected are the first and the second groups. The PI value suggests that GFSMLP can also do

Table 5.3: Performance of GFSRBF on Normalized-Chem (considering 3 groups of features)

No. of basis functions	PI		no. of groups selected	
	Mean	Std. Dev.	Mean	Std. Dev.
2	0.0023	0.0000	2.0	0.00
3	0.0020	0.0000	1.8	0.44
5	0.0037	0.0001	1.0	0.00
7	0.0029	0.0002	2.0	0.00
10	0.0025	0.0005	2.0	0.00
15	0.0020	0.0000	1.0	0.00

Table 5.4: Performance of GFSMLP on Normalized-Chem (considering 3 groups of features)

No. of hidden nodes	PI		No. of groups selected	
	Mean	Std. Dev.	Mean	Std. Dev.
5	0.002360	0.0000	2.0	0.0
7	0.002295	0.0000	2.0	0.0
10	0.002258	0.0000	2.0	0.0
15	0.002022	0.0000	2.0	0.0

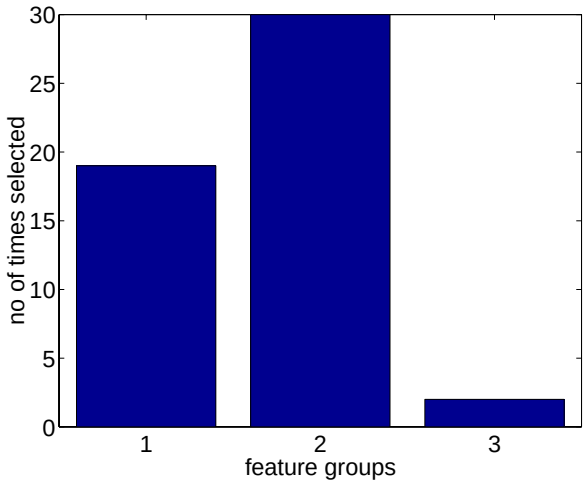


Figure 5.4: Bar diagram showing number of times each feature group gets selected for Chem data using GFSRBF

the function approximation job with a good accuracy. The best PI value obtained for 15 hidden nodes is better than both Lin and Cunningham's [140] system and our neuro-fuzzy system in Chapter 3. Note that, GFSMLP gives relatively more importance on the first group than that by GFSRBF. This emphasizes the fact that importance of a feature (or of a group of features) is a function of the tool being used to solve a problem.

5.4.2 Iris

For this data we select 100 points randomly as the training data. First we assume that the four features present in Iris form four groups with one feature in each group. So we obtain one feature modulator for each feature. We use a network with six basis functions. The values of the modulator functions for all features are shown in Table 5.5. Table 5.5 clearly shows that the network does not accept the first and the second features. This result is consistent with the well known fact that the third and fourth features of Iris data are enough for classification. The number of misclassifications obtained on the training data is 3 and on the whole data (150 points) is 5. This performance is quite comparable with other classifiers [12].

Physically the Iris features are the sepal length (f_1), sepal width (f_2), petal length

Table 5.5: Values of γ_j for Iris data (considering 4 groups of features) using GFSRBF

Group 1 (f_1)	Group 2 (f_2)	Group 3 (f_3)	Group 4 (f_4)
0.00	0.00	0.33	0.99

Table 5.6: Misclassifications and number of groups selected for Iris data with GFSRBF (considering 2 groups of features)

No. of basis functions	Trng Error (%)		Test Error (%)		no. of groups selected	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
3	3.0	0.0	4.0	0.0	1.0	0.0
5	3.0	0.0	4.0	0.0	1.0	0.0
7	3.0	0.0	4.0	0.0	1.0	0.0
10	3.0	0.0	2.8	1.08	1.0	0.0

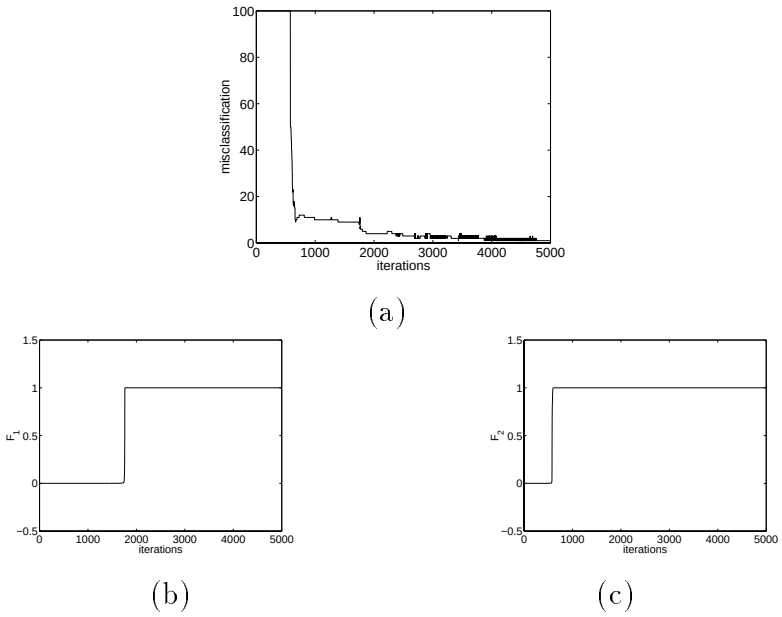


Figure 5.5: Variation of misclassifications and attenuator values with number of iterations for Iris data: (a) misclassification (b) attenuator values for group 1 features (c)attenuator values for group 2 features

(f_3)and petal width (f_4) of iris flower. Therefore, we can make two natural groups of features, i.e., one group characterizing the sepals and the other containing the petals. In other words, we consider features 1 and 2 as the first group and features 3 and 4 as the second group. With this grouping we ran GFSRBF for different basis functions and different FCM initializations. Table 5.6 shows the average misclassification and the average number of groups selected for 5 independent runs of GFSRBF for each architecture. Table 5.6 clearly suggests that our network selects only one group and is consistent with the change in basis functions. Also in all cases the network selected group 2 (i.e., f_3 and f_4) features.

GFSMLP also can select the relevant features for Iris data. We use a GFSMLP with 10 nodes in the hidden layer, each with a sigmoidal activation function. We use the same 100 samples for training as used for GFSRBF. Figure 5.5(a) gives the variation of the misclassifications with the number of iterations for a typical run. Figures 5.5(b) and 5.5(c) depict the feature attenuator values (F_l) for different iterations for group 1 and group 2 respectively for the same run. Figure 5.5(a) shows that the misclassification drops sharply from 100 to 10 at around 800 iterations. Figure 5.5(c) reveals that at that

Table 5.7: Values of γ_j in GFSRBF for RS-Data (considering 7 groups, 1 feature per group)

Group 1	Group 2	Group 3	Group 4	Group 5	Group 6	Group 7
0.42	0.00	0.84	0.99	0.41	0.99	0.25

time the features in the second group enters the network. As the training continues we find that again there is a sharp decrease in misclassification around 1800 iterations when the first group of features gets in the network (Fig. 5.5(b)). This behavior is consistent with most of the runs. We find that the 2nd group of features first gets in the network to give an average misclassification of 10 (averaged over 10 runs of the same network with different initializations). If training is continued, then the first group of features also gets in and reduces the misclassifications to 0. The final network produces a misclassification of 1 on the whole data. This clearly demonstrates the feature selection capability of the network. It says that feature 3 and 4 constitute a very important group of features for Iris, but the other group also have some discriminating ability that can facilitate learning in MLP. In fact this observation is consistent with the results reported elsewhere [166] which suggests that feature 1 and 3 have equally good discriminating power as that of feature 3 and 4. Here we like to emphasize the fact, that this network is primarily intended to select features. Once the features are selected one can remove the nodes and links associated with the bad/redundant features and retrain the net for a few more epochs for further improving the performance of the network. For example, in this particular case, after features 3 and 4 are selected, one can delete the links associated with features 1 and 2 and retrain the net to achieve a very good performance.

5.4.3 RS-Data

This data set contains 262144 points distributed in 8 classes. From each class we randomly select 200 points to get a training sample of 1600 points. For this data set too initially we consider each of the 7 features as constituting a group. Running GFSRBF with 30 basis functions we obtain a misclassification of 18.43% on the training

Table 5.8: Misclassifications and number of groups of features selected for RS14 data with GFSRBF

No. of basis functions	Trng Error %		Test Error %		no. of groups selected	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
10	24.72	0.23	17.55	0.28	5.6	0.89
20	20.61	0.71	16.42	0.34	6.0	0.00
25	19.93	1.11	14.96	0.10	5.4	0.54
30	18.96	0.73	15.21	0.75	5.2	0.44

Table 5.9: Misclassifications and number of groups of features selected for RS14 data with GFSMLP

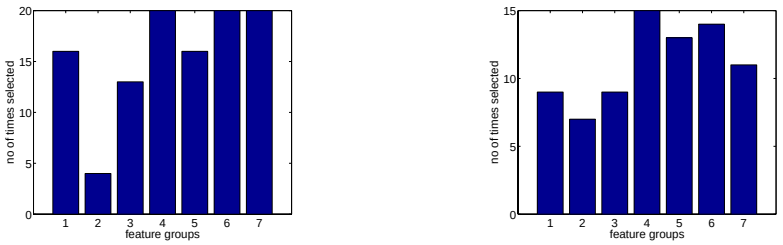
No. of hidden nodes	Trng Error %		Test Error %		no. of groups selected	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
10	26.52	4.25	20.88	2.53	3.4	0.54
20	21.82	2.67	20.76	3.12	3.8	0.44
25	18.02	2.26	17.96	1.38	4.2	0.44
30	17.63	2.74	18.86	1.65	4.0	0.70

data and 15.59% on the test data. The final values of the feature attenuators are shown in Table 5.7, which reveals that the network completely discards the second feature. In [121] a misclassification of 21.8% was obtained on the test data. In [128], a misclassification of about 14% on the test data was reported. The performance of our system is comparable to theirs though our system uses less number of features. In Chapter 4, using the neuro-fuzzy system we obtained a misclassification of 16% on test data using only 5 features.

In another experiment we generate an additional feature from each of the 7 channels of the image. For each pixel p in an image we consider its 8-neighborhood over a 3×3 window and compute the standard deviation of the 9 pixel values (the neighborhood of p and the pixel itself), we call it d_p. In the new data set, for each channel we take the gray value of p and d_p as features. So we have 14 features divided into 7 groups. We call this as the RS14 data. Table 5.8 shows the average misclassification (in percentage) on training and test data for 5 independent runs for different architectures. Figure 5.6(a) shows the number of times each feature gets selected for these 20 runs. From Fig. 5.6(a) we find that the features from the second sensor are selected the least number of times. This is consistent with the results described in Table 5.7 using 7 features.

No previous study regarding the goodness of features of RS-Data exists. We made a naive feature analysis to compare our results. We ran the k-nearest neighbor classifier [12], on this data with all possible combination of 6 features, i.e., in each run we left out one feature. Among the 7 possible combinations the feature set $\{1, 3, 4, 5, 6, 7\}$ results in the least number of misclassification. This clearly points out that feature 2 is a bad feature. The neuro-fuzzy classifier proposed in Chapter 4 also discards feature 2 as a bad feature but additionally it rejects the 6^{th} feature.

The GFSMLP selects fewer groups from RS14 data than GFSRBF. Table 5.9 shows the percentage of misclassification and the number of groups of features selected for different number of hidden nodes for GFSMLP. For each architecture (a fixed number of hidden nodes) average misclassifications and average number of features selected for 5 independent runs are reported in Table 5.9. As evident from Table 5.9, GFSMLP produces a poorer classification than GFSRBF. But, for all cases our results are better than the result reported in [121]. In [121] a misclassification of 21.8% is reported on the test data. Compared to the neuro-fuzzy system of Chapter 4, the performance of GFSMLP is poor. Figure 5.6(b) shows the frequencies with which the various groups



(a) (b)

Figure 5.6: Bar diagram showing the number of times each feature group gets selected for RS14 data: (a) GFSRBF (b) GFSMLP

are selected by GFSMLP. Here too the features from the second sensor are selected the least number of times. Hence this result is also consistent with the results of the previous experiments on RS-Data.

5.4.4 Wine

For Wine data a natural grouping of features is not possible. Thus, we consider 13 groups with one feature in each group. We use 100 randomly chosen points from the data set for training and the remaining 78 points for testing.

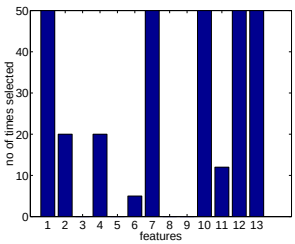
We train GFSRBF's with different number of basis functions. As shown in column 1 of Table 5.10 we consider five different architectures of GFSRBF. For each architecture we make 10 independent runs of the network with different initializations. Table 5.10 shows the average misclassifications along with their standard deviations for 10 independent runs of GFSRBF's with each architecture. Similarly for GFSMLP also we consider 5 different architectures as shown in column 1 of Table 5.11. For each architecture we make 10 independent runs. The summary of the runs by GFSMLP is included in Table 5.11. Tables 5.10 and 5.11 show that the number of features selected by GFSMLP is always less than that selected by GFSRBF. Figures 5.7(a) and 5.7(b) show the frequency distribution of the number of times each feature is selected over 50 runs of GFSRBF and GFSMLP respectively.

Table 5.10: Misclassifications and number of features selected for Wine data with GFSRBF

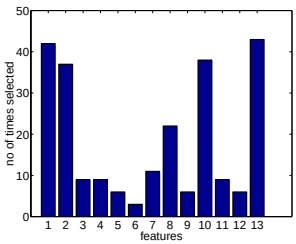
No of basis functions	Trng Error %		Test Error %		no. of features selected	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
3	4.5000	0.5270	5.1280	0	7.5000	0.5270
5	7.0000	0	4.8716	1.3240	6.0000	0
7	3.0000	0	1.2820	0	5.4000	0.8433
10	4.0000	0	2.5640	0	6.0000	1.0541
15	4.0000	0	2.5640	0	5.9000	0.3162

Table 5.11: Misclassifications and number of features selected for Wine data with GFSMLP

No.of hidden nodes	Trng Error %		Test Error %		Features Selected	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
3	4.3000	1.3375	8.4612	1.5048	5.8000	0.6325
5	1.9000	1.1972	4.6153	1.9301	4.9000	0.5676
7	1.9000	1.1972	4.4871	2.2814	5.2000	1.2293
10	4.4000	1.9551	3.5897	3.1287	4.1000	0.7379
15	2.4000	1.4298	7.1794	2.7826	4.1000	0.8756



(a)

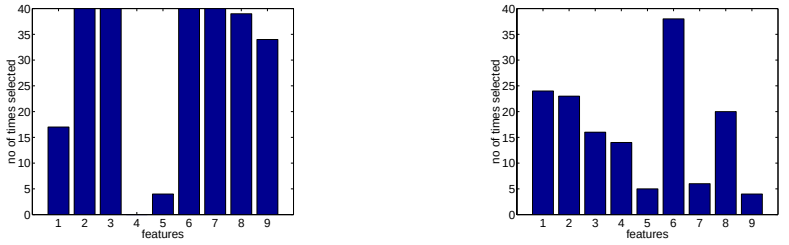


(b)

Figure 5.7: Bar diagram showing the number of times each feature gets selected for Wine data: (a) GFSRBF (b) GFSMLP

Table 5.12: Misclassifications and number of features selected for Breast-Cancer data with GFSRBF

No. of basis functions	Trng Error %		Test Error %		Features Selected	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
5	4.06	0.3134	1.004	0.75	6.3000	0.6749
7	4.14	0.2319	1.14	0.24	6.4000	0.5164
10	3.90	0.2867	1.14	0.24	6.7000	0.8233
15	3.96	0.2065	1.00	0.00	6.1000	0.3162



(a) (b)

Figure 5.8: Bar diagram showing the number of times each feature gets selected for Breast-Cancer data: (a) GFSRBF (b) GFSMLP

5.4.5 Breast-Cancer

For the Breast-Cancer data we randomly select 500 points from the 684 points to use them as the training set and the rest are used for testing. For this data set also we test the performance of GFSRBF using different architectures. We consider networks with 5, 7, 10 and 15 basis functions. Table 5.12 shows the mean misclassification and the standard deviation for 10 independent runs of GFSRBF for each architecture. It also shows the average number of features selected.

Table 5.12 reveals that the performance of GFSRBF on this data in terms of misclassification and the number of features selected does not vary much with the change in the number of basis functions. The bar diagram in Fig. 5.8(a) shows the number of times each feature is selected for the 40 runs.

Table 5.13: Misclassifications and number of features selected for Breast-Cancer data with GFSMLP

No. of hidden nodes	Trng Error %		Test Error %		Features Selected	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
5	4.32	0.54	1.15	0.33	3.5000	0.5270
7	4.16	0.40	1.80	0.91	3.9000	0.8756
10	3.56	0.47	1.25	0.54	4.1000	0.7379
15	3.86	0.48	0.95	0.59	3.6000	0.8433

Table 5.13 exhibits the performance of GFSMLP on Breast-Cancer data. Here too we follow the same protocol as in case of the GFSRBF network. We consider GFSMLPs with 5, 7, 10 and 15 hidden nodes. Here also we find that the performance is quite stable with the change in the number hidden nodes. Figure 5.8(b) depicts the frequency of the selected features over the 40 runs.

5.4.6 Evaluation of Features

We now try to evaluate the quality of features that are selected by GFSRBF and GFSMLP. We use the Wine and Breast-Cancer data for this purpose. We again demonstrate here that the suitability of the features depends not only on the task but also on the learning machine. We train conventional MLPs and conventional RBFs with all features present in the data and also with the features selected by GFSMLP and GFSRBF and compare their performance.

From Wine data a GFSRBF selects on the average 6.16 features. Thus, we take the six most frequently selected features as shown in Fig. 5.7(a) as the features selected by GFSRBF for Wine data. These selected features are: 1, 2, 7, 10, 12, 13. The GFSMLP selects 4.82 ($\approx$ 5) features on average for Wine data, thus the GFSMLP selected features for Wine data are: 1, 2, 8, 10, 13 (as evident from Fig. 5.7(b)). Similarly, we get the GFSRBF selected features for Breast-Cancer data as 1, 2, 3, 6, 7, 8, 9 and the GFSMLP selected features as 1, 2, 6, 8.

For the experiments we use the same training-test partition as used in the previous experiments. We use standard MATLAB implementations of RBF and MLP. For MLP we use sigmoidal activation functions and the *trainlm* algorithm for training. The MATLAB implementation of RBF takes the same spread for all basis functions. We experiment with spreads ranging from [0.1 10] with a difference of 0.1 and report the best result that we obtain.

Table 5.14 depicts the results of a conventional MLP when run with different hidden nodes on different sets of features of Breast-Cancer data. For each architecture (a fixed number of hidden nodes), 5 independent runs are made, and the average misclassification in percentage on training and test data along with the standard deviation are reported in Table 5.14. Table 5.14 shows that an ordinary MLP produces lesser test error on the GFSMLP features for all four different architectures that we tried. When MLPs are trained with features selected by GFSRBF the test result is slightly worse than what is achieved with all features. This again re-emphasizes the fact that features selected by GFSMLP are good, but they are best with MLP. Table 5.15 gives the results of a conventional RBF on the same set of features. Here too we see that the performance of an RBF network is better on the features selected by GFSRBF than on that selected by GFSMLP reconfirming the fact that utility of features depends on both the problem and the learning machine.

Tables 5.16 and 5.17 display the results on Wine data for MLP and RBF on different sets of features. Here too we notice that on average an ordinary MLP performs better with the GFSMLP features and an ordinary RBF performs better with the GFSRBF features.

5.5 Conclusions and Discussion

Many real life applications use data from several sensors for decision making. Intelligent systems for automatic inspection and controlling of welding, medical diagnosis, controlling of range safety for missile testing are some such examples. Typically, each sensor output is converted into a set of features. For example, X-ray radiograph may be used to compute a set of features for weld inspection. More sensors mean more cost, more processing time and sometimes more hazardous too (X-rays) and do not

Table 5.14: Misclassifications produced by an ordinary MLP with various sets of features on Breast-Cancer data

No. of hidden nodes	All Features				GFSMLP selected				GFSRBF selected			
	Trng. Error (%)		Test Error (%)		Trng Error (%)		Test Error (%)		Trng. Error (%)		Test Error (%)	
	Mean	S.D.	Mean	S.D.	Mean	S.D.	Mean	S.D.	Mean	S.D.	Mean	S.D.
5	1.00	1.44	2.80	1.78	1.40	0.51	1.2	0.83	1.00	0.24	2.3	0.97
7	0.76	0.86	1.90	0.74	0.84	0.33	1.6	0.82	0.84	0.45	2.3	0.70
10	0.36	0.54	2.30	1.03	2.40	0.56	2.1	0.65	0.12	0.19	3.3	0.27
15	0.20	0.24	2.30	1.25	2.20	0.35	1.9	1.24	0.20	0.34	2.4	1.29

Table 5.15: Misclassifications produced by an ordinary RBF network with various sets of features on Breast-Cancer data

No. of basis functions	All Features			GFSMLP selected			GFSRBF selected		
	Trng. Error (%)	Test Error (%)	spread	Trng Error (%)	Test Error (%)	spread	Trng. Error (%)	Test Error (%)	spread
5	3.80	1.50	6.4	4.00	1.50	2.7	3.60	1.00	6.2
7	3.80	1.00	6.4	3.80	1.00	4.9	3.60	1.00	6.0
10	3.60	0.00	8.8	3.60	1.00	4.8	3.40	1.00	5.7
15	3.40	0.50	6.7	3.40	0.00	2.7	3.60	0.50	5.6

Table 5.16: Misclassifications produced by an ordinary MLP with various sets of features on Wine data

No. of hidden nodes	All Features				GFSMLP selected				GFSRBF selected			
	Trng. Error (%)		Test Error (%)		Trng Error (%)		Test Error (%)		Trng. Error (%)		Test Error (%)	
	Mean	S.D.	Mean	S.D.	Mean	S.D.	Mean	S.D.	Mean	S.D.	Mean	S.D.
5	0.0	0.0	1.28	1.39	1.2	0.83	3.07	1.14	0.4	0.54	4.35	1.46
7	0.0	0.0	1.53	1.28	0.2	0.44	4.10	1.06	0.0	0.0	4.35	2.65
10	0.0	0.0	2.05	3.21	0.2	0.44	3.84	0.89	0.0	0.0	5.64	1.96
15	0.0	0.0	2.56	2.39	0.0	0.00	4.61	1.46	0.0	0.0	3.33	1.46

Table 5.17: Misclassifications produced by an ordinary RBF network with various sets of features on Wine data

No. of basis functions	All Features			GFSMLP selected			GFSRBF selected		
	Trng. Error (%)	Test Error (%)	spread	Trng Error (%)	Test Error (%)	spread	Trng. Error (%)	Test Error (%)	spread
5	1.00	3.84	1.4	5	3.84	0.5	2.00	1.28	0.5
7	1.00	2.56	1.1	5	3.84	0.5	2.00	3.84	0.4
10	1.00	3.84	0.8	4	2.56	0.4	2.00	3.84	0.3
15	0.00	1.28	0.9	2	7.69	0.4	1.00	2.56	0.4

necessarily lead to better performance. Therefore, if we can reduce the number of required sensors we can save cost, time, design complexity and sometimes minimize risks. This is a very important problem but not addressed in literature. In this chapter we provided some novel solutions to this problem. In particular we achieved the following:

1. We proposed two novel schemes for solving these problems. One scheme is based on RBF framework and the other uses MLP.
2. Both schemes are capable of selecting useful groups of features.
3. Both schemes can also select individual features.
4. We proved that both GFSRBF and GFSMLP retain the universal approximation property.
5. Our experimental results reconfirm that the importance of features depends on the tool used to solve a problem.
6. Our limited experiments show that GFSMLP needs lesser number of features to do a given task than GFSRBF.

In near future, we like to extend this concept to neuro-fuzzy framework.

With this we conclude our study of integrated feature selection methods in this thesis. In the next three chapters which follow, we discuss the generalization issue in MLP networks.

Chapter 6

Strict Generalization and Incremental Learning in Multilayer Perceptron Networks: Classification[1]

6.1 Introduction

As stated before, in this thesis we deal with two attributes of efficiency of systems built from data, namely, feature selection and generalization ability. In the three preceding chapters we considered a new paradigm of feature selection based on modulator functions which we called *online feature selection*. We applied the method for designing neuro-fuzzy systems for function approximation and classification to get "optimal" architectures which discard irrelevant features when they get trained. We also demonstrated that the concept of modulator functions can be extended to do sensor (group feature) selection with modified versions of RBF and MLP networks. In this chapter and in the following two chapters we shall consider generalization properties of multilayer perceptron networks.

Over the years multilayer perceptrons have been used to solve numerous problems in various domains. It has been proved that MLPs can act as universal approximators for a large class of nonlinear functions [77], further the learning and generalization properties of these networks have found many diverse applications. The generalization ability of MLPs is often over estimated. Users generally rely on the output of an MLP for any input without paying due consideration on the position of the input with

[1]The contents of this chapter have been published in [26, 29].

respect to the training data. Modern methods of training MLPs involve strict phases of training, validation and testing. But one generally depends on the data set at hand for these phases and good performance can be guaranteed only on the available data. The response of an MLP for a data point which lies well outside the "boundary" of the training data is usually erratic and hence not reliable. For example, even when a test data point is far away from the convex hull of the training data, an MLP will produce some output, sometimes this response (for classification) could be very high and can be totally useless and dangerous too. As MLPs are being used in critical areas like medical diagnosis, non destructive testing (NDT) [220], this may lead to serious problems. We will illustrate this issue latter with compelling examples.

MacKay [145, 146, 147] addresses more or less the same problem of designing classifiers which are intended to give low response in areas with sparse training data. In [147] MacKay elaborated that a network (classifier) designed with the most probable parameter (weight) vector $\boldsymbol{w}_{MP}$ (obtained using a training data D) will typically give more extreme, unrepresentative and overconfident output in areas with sparse training data - and this is *not* desirable. Hence, he suggested a strategy to moderate the output of a network with parameters $\boldsymbol{w}_{MP}$. His results show that the network with moderated output performs better. The moderated output is similar to the most probable output in the regions where the training data are dense. On the other hand, where the training data are sparse, the moderated output becomes significantly less certain than the most probable output. This is certainly an interesting approach. But, since MacKay's training scheme [147] does not consider the input space having no data, the moderated output could be high even in areas far away from the training data. The results in [147] reveal that this is indeed the case for a simple two class problem.

Another very important issue concerned with MLP is how to learn new knowledge without forgetting old knowledge. An MLP may (usually will) not retain its old knowledge when it is retrained by a new data set. Also, there is no easy means to augment an MLP to incorporate new knowledge. This issue of incremental learning has been addressed by a few researchers in [57, 60, 177, 210, 245]. This problem of training a neural network incrementally has been addressed in two distinct paradigms:

1. To use the old data points along with the new data points for training which yields a new system with both the old and the new knowledge [177, 245]. This

is computationally expensive and also the old training data may not be available in all cases.

2. While retraining with a new data set, only bounded modification of the network parameters is allowed, so that the network does not forget its old memories. This controlled update may not always guarantee complete learning in cases where the new training data are not similar to the old training data. The method described in [60] follows this philosophy.

In this chapter we address two issues. A network should learn only as much as dictated by the training data. When an MLP is used as a classifier, this means that for test data points which are away from the training data points, the class response for *every* class should be very low. We call such a generalization as *strict generalization*. The other issue deals with incremental learning. Given a trained network, if some new training data points come for which the current class responses for all classes are very low, then we should be able to augment the network so that it can learn the new data without forgetting the old ones. Note that, the old training data are not available any more.

Here we aim to build a classifier which will not make a class assignment for every data point. Thus, this classifier does not conform to the definition of a classifier that we provided in Chapter 4. In Chapter 4 we defined a classifier as a function $\boldsymbol{S} : \Re^s \rightarrow N_{pt}$, where N_{pt} are a set of label vectors defined as in eq. (4.1). According to this definition, for any point $\boldsymbol{x} \in \Re^s$ the classifier $\boldsymbol{S}$ should output a class label in N_{pt}, and the class label denotes the degree of belongingness of $\boldsymbol{x}$ in one or more classes. Note that, N_{pt} in eq. (4.1) excludes the all zero vector. Here, we want to design a classifier, that can also output a label vector representing that the point $\boldsymbol{x}$ does not belong to any of the classes. This can be suitably done by redefining the set of label vectors N_{pt} as N'_{pt}. Where N'_{pt} is defined as

$$N'_{pt} = \left\{ \boldsymbol{y} \in \Re^t : y_i \in [0,1] \forall i \right\} = [0,1]^t. \tag{6.1}$$

In eq. (6.1) N'_{pt} includes the all zero vector $\mathbf{0}$ also. This vector $\mathbf{0}$ can serve as a class label which indicates that a point does not belong to any of the classes. Now, we can define our classifier as $\boldsymbol{S} : \Re^s \rightarrow N'_{pt}$.

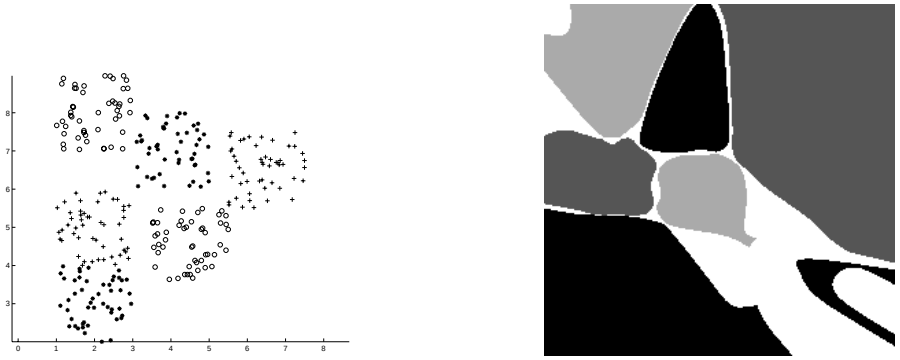
Rest of this chapter is organized as follows: First we discuss the concept of boundary

of a pattern class. Then we show by examples that the behavior of an MLP outside the boundary of the training data is erratic when it is used as a classifier. We then present a new training scheme which can achieve strict generalization. The proposed training scheme is also capable of incremental learning, i.e., a trained network can easily be augmented to incorporate new knowledge. Although in this chapter we concentrate on MLPs used as classifiers, with some modification the methodology can be made applicable to MLPs used for other applications like function approximation, system identification etc. which we consider in the next chapter.

6.2 Boundary of a Pattern Class

The data points from a pattern class usually will be generated following some distribution. The distribution function can give information about the spatial boundary of the pattern class. But in most real life cases such distribution functions are neither known apriori nor are easy to estimate from the data points representing different classes. One way to determine the class boundary may be to estimate the sampling window from which the points of that class are generated. The sampling window is generally defined as the convex compact support set of the distribution. The convex hull of the points in a class is considered to be a good estimate of its sampling window [198, 219]. If the points of a class are generated from a convex support set and form a cluster in the input space, then the convex hull of the points in the training data may be a good estimate of its boundary. But, in real life data sets, points from a class can take any shape - it may even form a number of separate clusters in the input space. And in these cases finding boundary of a pattern class poses a serious problem. In a recent work [212], Schölkopf *et al.* attempted to find the support of a multidimensional distribution using the support vector machine framework. This method can be suitably used to determine the boundary of a class, but here we take a simpler and intuitive definition of boundary.

We introduce the concept of "boundary" in terms of neighborhood. Suppose C is a set of points which belongs to a class. Any point $\boldsymbol{x}$ is considered to be a point outside the boundary of the class if the distance of $\boldsymbol{x}$ from its nearest neighbor in C is greater than a threshold th. The threshold th will depend on the shape and density of the points in the class represented by C. We continue with this naive definition of a boundary



(a) (b)

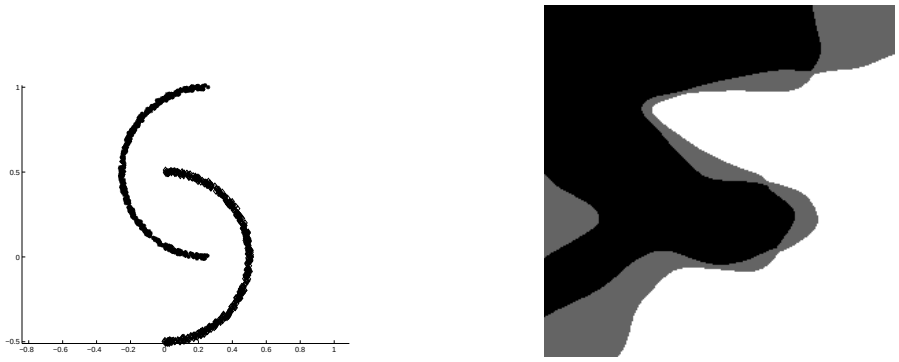
Figure 6.1: (a) Scatterplot of Scattered1 (b) Generalization by an MLP on Scattered1

and in Section 6.4 we shall device an algorithm to find points outside the boundary of a pattern class.

6.3 Improper Behavior of MLP Outside the Boundary of the Training Sample

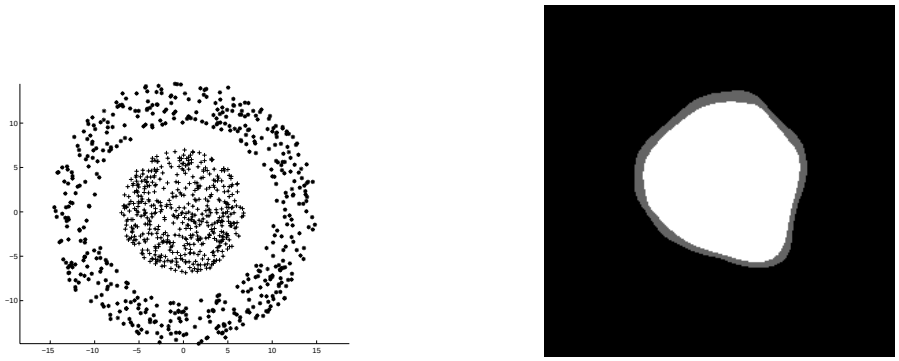
With the naive definition of "boundary" in mind, we show by examples that the behaviour of an MLP for a point which lies outside the "boundary" of its training sample is not predictable. For this purpose here we use three representative data sets, all in $\Re^2$. The data sets are named **Scattered1**, **Two-Shell** and **Dish-Shell**. Scattered1 has 3 reasonably well separated classes with 100 points in each class. The scatterplot of Scattered1 is shown in Fig. 6.1(a) where *, + and o represent the points from the three different classes. Both Two-Shell and Dish-Shell have two classes with 500 points in each class. Scatterplots of Two-Shell and Dish-Shell are shown in Figs. 6.2(a) and 6.3(a) respectively.

An MLP with 20 nodes in a the hidden layer with sigmoidal activation functions is used to classify all three data sets. There is no misclassifications on the training data for each of these data sets. This appears very encouraging. Let us now see, in each case how the network generalizes to points outside the "boundary" of the classes. For this we compute the smallest rectangle containing the data, with its sides parallel to the axes of co-ordinates. Then we increase each edge by 5% on each side. Next we generate



(a) (b)

Figure 6.2: (a) The scatterplot of Two-Shell (b) Generalization by a trained MLP



(a) (b)

Figure 6.3: (a) The scatterplot of Dish-Shell (b) Generalization by a trained MLP

an array of $256 \times 256 (= 65536)$ points uniformly covering the entire rectangle. And use these 65536 points as our test data. We consider a point to be classified to class k if the output of the k^{th} output node is *more than 0.8* and the output of all other nodes is *less than 0.2*. For Scattered1 the generalization realized by a trained MLP is shown in Fig. 6.1(b). The areas marked by the two shades of gray and black represent the three classes and white area represents the points for which the MLP could not make any decision. We draw the attention of the reader to the U-shaped black patch at the bottom right corner suggesting the class represented by * in Fig. 6.1(a). This is surely a very poor generalization, although the training set resulted in zero misclassification.

In Figs. 6.2(b) and 6.3(b) the generalization by a trained MLP on Two-Shell and Dish-Shell are shown. Here, black and white represent the two classes and gray represents the points for which no decision is made by the MLP. In these cases too the generalizations are not desirable. Note that, here we use a conservative approach to decide the class. Usually, a data point $\boldsymbol{x}$ is classified to class k if the k^{th} output node have the maximum response (ignoring how strong or weak the maximum response is or what the response to the other classes are). So, every point gets classified to one of the classes.

The above results clearly demonstrate that an MLP performs quite well on the training data and also on the points which lie inside the "boundary" of each pattern class. But its response on points outside the "boundary" of the training data may not follow any specific behavior - often it behaves in a strange manner!

6.4 A New Training Scheme

Here we discuss a new training scheme which takes care of the problems discussed above. Let us consider a classification problem of a data set $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n\} \subset \Re^s$. Suppose X consists of k classes and $X = \cup_{i=1}^{k} S_i$, $S_i \cap S_l = \phi$, $i \neq l$, such that $\boldsymbol{x}_j \in S_i$ is from class i. We assume that we have a mechanism to define the boundary of each S_i.

Let Ω_{small} be the smallest hypercube, which bounds X. By increasing each edge of Ω_{small} by $l\%$ on all sides, we inflate Ω_{small} to Ω, we call Ω as the inflated hyperbox of X. Our scheme will guarantee strict generalization on all test data points which lie within Ω. Thus Ω is the space from which the data points are expected to come.

We may have prior knowledge of Ω or we can compute Ω by inflating Ω_{small} to some extent. For our simulations we have used a specific inflation rate (l=5%). Now for each pattern class S_i, let Ξ_i denote a set of points generated uniformly *within* Ω, but *outside* the boundary of points in S_i. For the data set X we construct k training sets T_i, $i = 1, 2, ..., k$; where T_i includes points in S_i with label 1 and points in Ξ_i with label 0.

We train k MLPs with these T_is, $i = 1, 2, ..., k$; we call these networks M_i, $i = 1, ..., k$. The restriction on the architecture of each M_i is that its input layer consists of s nodes and the output layer consist of only 1 node. Thus each M_i learns a 2 class problem and can detect whether a data point is within the boundary of class i or not.

These k MLPs can be merged together, as shown in Fig. 6.4 to form a single network $\mathcal{M}$ which solves the required k class problem. Such merging of k subnets to identify k different classes is called *simple merging*. The s dimensional input is fed to each M_i, if it belongs to class $q, q \in \{1, 2, .., k\}$, then the response of M_q would be high and those of the rest would be low (Fig. 6.4).

If the training data from different classes do not overlap, the network can make unambiguous decision. But if the training data from more than one class overlap the network will be able to signal that. We shall discuss this later. We now provide a schematic description of the entire training process in algorithm TRAIN which is depicted in Table 6.1. In the algorithm we use $(m : n : l)$ to indicate an MLP with m input nodes, n hidden and l output nodes.

After the training is over we get a composite network, which, given a test input $\boldsymbol{x} \in \Re^s$ with unknown class label, will produce a k-dimensional output vector. The next issue is how to interpret the output of the composite network $\mathcal{M}$. To understand this, we need to consider the structure of the training data. Suppose there is no overlap between the training data from various classes. Thus for each training and test data points *at most* one of the subnets will produce high response. And consequently either the class label of $\boldsymbol{x}$ will be unambiguous or no decision can be made on it. If the training data from classes i and j are overlapped, and if the test data point $\boldsymbol{x} \in \Re^s$ is from the overlapped region, then output nodes corresponding to classes i and j of the composite network will be high. In such a case we should not assign $\boldsymbol{x}$ to one of i and j, but should make a decision that $\boldsymbol{x}$ can be in either of the two classes. This gives an additional information

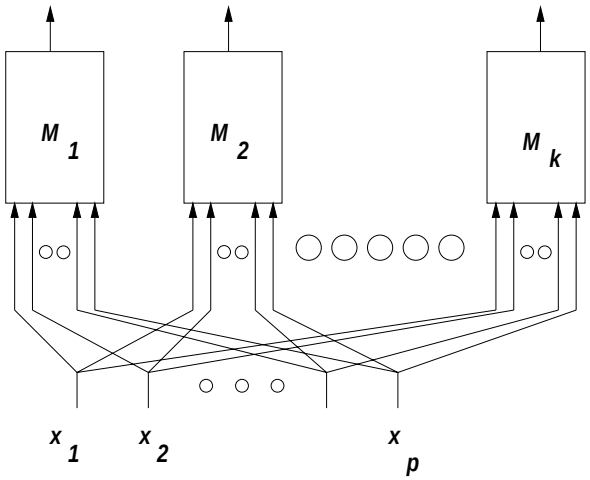


Figure 6.4: Simple Merging of k trained MLPs

Table 6.1: Algorithm Train

Algorithm **TRAIN**

Let $X = \cup_{i=1}^{k} S_i$, $S_i \subset \Re^s$ be the set of points in class i.

for $i = 1$ to k

Generate Ξ_i (the points outside the boundary of S_i)

$T_i = S_i \cup \Xi_i$ (S_i with class label 1 and Ξ_i with 0)

Train subnet M_i (architecture $s : n_i : 1$) with T_i.

end for

Simple merge $M_1, M_2, ..., M_k$ to get M

end

about $\boldsymbol{x}$, that it probably comes from an overlapped region.

Thus, by properly interpreting the output of our network we can get more knowledge about the points in an overlapped region, and also about the points on which our network may give wrong decisions. We say that a point belong to class l if the l^{th} output o_l of the composite network is greater than 0.8 and all other outputs are less than 0.2. If every output unit gives a response less than 0.2, (i.e., $o_j < 2, \forall j$) then we do not make any decision about the point. If $o_j > 0.8$ for $j = i_1, i_2, ..., i_l$ and $o_j < 0.2$ for all other classes, then $\boldsymbol{x}$ is probably in an overlapped area of the classes $i_1, i_2, ..., i_l$. In all other cases, responses of one or more classes are high, but none of them is high enough. In such cases we make a soft decision. We take the class label corresponding to the output units giving the maximum response. But in this case our network warns that this decision is a soft or weak decision, and it may be wrong. The test procedure is summarized in the algorithm TEST, shown in Table 6.2.

In conventional training of an MLP, whenever there are overlapped classes, the MLP will surely produce misclassifications. If an MLP learns overlapped classes without any misclassification, then it is a sort of overfitting on the data, which will produce bad generalization. The usual training of MLP is not designed in such a way that the outputs can be interpreted to detect whether a test point belongs to an area of overlap. We provide some simulation results in the results section in support of this. In case of our network we can obtain multiple class labels for a data point which lies in an area of overlap. Also when the class response for any class is not significantly high, our network gives a soft decision. This is an additional advantage of our method over the MLP.

Note that, the method for obtaining strict generalization assumes that the training data set is reliable and does not have outliers. It cannot get rid of effect of outliers in the training data. It is meant to verify whether a test point is an outlier with respect to the data used for training. Next we discuss how our method can be used for incremental learning.

Table 6.2: Algorithm Test

Algorithm **TEST**($\boldsymbol{x}$,M)
  Let M contain k output nodes;
  Input $\boldsymbol{x}$ to M;
  Let $U = \{1, 2, ..., k\}$(The set of classes);
  Let o_i, $i \in U$ be the output of the i^{th} output node;
  Let $I = \{i : o_i > 0.8, \forall i \in U\}$;
  If $o_j \leq 0.2, \forall j \in U - I$ *Then*
    if $|I| = 0$ *Then*
      class for $\boldsymbol{x}$ cannot be decided;
    else if $|I| = 1$ (let I=$\{c_1\}$) *Then*
      unambiguous decision, $\boldsymbol{x}$ belongs to class c_1;
    else (let I=$\{c_1, c_2, ..., c_l\}$, $1 < l \leq k$)
      ambiguous decision, $\boldsymbol{x}$ can belong to any of the
      classes $c_1, c_2, ..., c_l$;
    end if
  else
    $j = Argmax_{k \in U}\{o_k\}$;
    Soft decision, $\boldsymbol{x}$ belongs to class j;
  endif
end

6.4.1 Incremental Learning

The proposed training scheme can be easily used to augment a trained network with a new set of training data. We develop the method for incremental learning under the assumption that we have prior knowledge about the input space Ω, from which new training data may come. This is really not a strong assumption as one can start with a big enough Ω to account for future augmentation of the network.

Suppose, initially we have a data set $X_{init} = \cup_{i=1}^{k} S_i$, which represents k classes. Let us call the k classes as $1, 2, ..., k$, and S_i contains the points in the i^{th} class. We train k subnets, as discussed in the previous section to solve the k class problem. We name the subnets as M_{init}^{i}, $i = 1, 2, ..., k$. Then we merge these subnets by a simple merge to form a new network $\mathcal{M}_{init}$. $\mathcal{M}_{init}$ incorporates the knowledge in X_{init}.

Later suppose we obtain a new data set X_{new} which represents $m = k + n$ classes, where the first k classes are the same as the k classes in X_{init} and the other n classes are new ones. Let X_{new}^{O} denote the set of points in X_{new} which belongs to the first k classes and X_{new}^{N} denote the set of points which belongs to the new n classes, so $X_{new} = X_{new}^{O} \cup X_{new}^{N}$ and $X_{new}^{O} \cap X_{new}^{N} = \phi$. Usually the points in X_{new}^{N} will lie outside the boundary of the k classes in X_{init}. If so, then because of our specialized training, $\mathcal{M}_{init}$ will not produce high response for the points in X_{new}^{N}. But, if $\mathcal{M}_{init}$ produces high response for some points in X_{new}^{N}, then there is some overlap between the previous and the new classes. Irrespective of whether $\mathcal{M}_{init}$ produces high response for points in X_{new}^{N} or not, for each of these n new classes we construct n separate MLPs, $M_{additional}^{i}$, $i = k + 1, k + 2, ...k + n$, as discussed before and merge them by *simple merge* with $\mathcal{M}_{init}$ to produce a new network $\mathcal{M}_1$. Hence, $\mathcal{M}_1$ should be able to classify points in $X_{init} \cup X_{new}^{N}$ into $m = k + n$ classes.

Although the points in X_{new}^{O} represent the existing k classes, $\mathcal{M}_1$ may not correctly classify all points in X_{new}^{O}. If it can, then we are done and $\mathcal{M}_1$ is the desired network. Otherwise, we need to augment $\mathcal{M}_1$.

Let $X_{new}^{O} = X_{new}^{C} \cup X_{new}^{R}$, where X_{new}^{C} is the set of points which is correctly classified by $\mathcal{M}_1$, and X_{new}^{R} contains the points which are not classified correctly by $\mathcal{M}_1$. Note that, here by correct classification of points in X_{new}^{C} we mean that the network $\mathcal{M}_{init}$ produces high response for all points in X_{new}^{C} in one or more classes which includes the correct class. The points in X_{new}^{C} need not be considered for further training. The

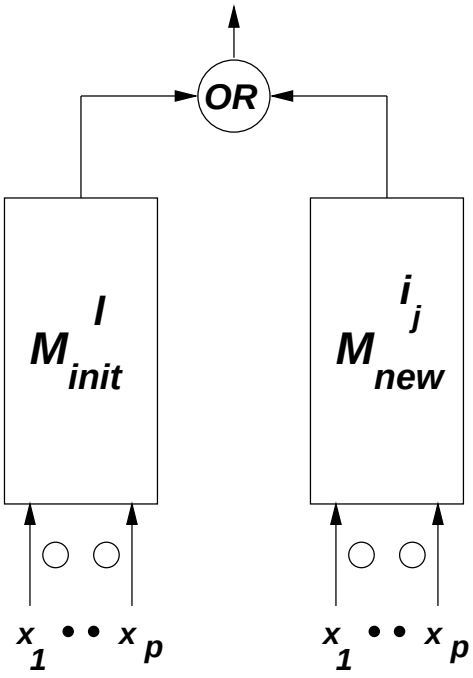


Figure 6.5: Compound Merge of two trained MLPs

points in X^R_{new} are either outside the boundary of all the classes present in X_{init} or there may be points representing class i but lying within the boundary of S_j, $j \neq i$. The latter case is a case of overlapped classes. In both cases points in X^R_{new} contain new information which is to be augmented with the initial network.

Suppose the set X^R_{new} represents $q \leq k$ classes. So we train q subnets, M^i_{new} $(i = i_1, i_2, ..., i_q)$, i_j denotes one of the q classes. $M^{i_j}_{new}$ is trained with the points which belong to class i_j in X^R_{new} along with points generated outside the "boundary" of the respective class. Note that, each of these q subnets detects (represents) one of the classes already represented by some subnet in $\mathcal{M}_{init}$ (and also in $\mathcal{M}_1$). Let us denote the initial subnets of $\mathcal{M}_{init}$ by M^i_{init} $(i = 1, 2, ..., k)$. Subnet M^l_{init} is merged with $M^{i_j}_{new}$, if $l = i_j$, by a *compound merge* (Fig. 6.5). In a compound merge the outputs of M^l_{init} and $M^{i_j}_{new}$ $(i_j = l)$ are combined by an OR operator to get the final output. For our case we use max as the OR operator. The network obtained by the compound merging of the subnets of $\mathcal{M}_1$ and $M^{i_j}_{new}$, $j = 1, 2..., q$, is denoted by $\mathcal{M}_{final}$. $\mathcal{M}_{final}$ should be able to classify all points in $X_{init} \cup X_{new}$.

For better clarity, we summarize the above method of augmenting a trained network by

the pseudo-code AUGMENT (shown in Table 6.3). The algorithm AUGMENT takes as input a pretrained network $\mathcal{M}_{init}$ which represents k classes and has k subnets $M^i_{init}, i = 1, ..., k$, and a new data set X_{new}. $X_{new} = X^O_{new} \cup X^N_{new}$, as discussed above. The algorithm AUGMENT creates the augmented network M_{final}.

The crucial step in the methodology described is generation of the training set for each class which contains the given training points of that class along with points which lie outside the boundary of that class. Thus, the success of the scheme will depend on the generation of points outside the boundary of a given pattern class. In the next subsection we define what we mean by the boundary of a class and a method to generate points outside the boundary of a pattern class.

6.4.2 Generating Points Outside the Boundary of a Pattern Class

Here we present a simple and approximate scheme to generate points outside the boundary of a pattern class. Let Ω to be the inflated hyperbox which bounds the data points in X (as discussed earlier). We define δ^i_{avg} as the average edge length of a minimal spanning tree (MST) which spans the points present in S_i, i.e., the training points for class i.

Definition: *Boundary of a pattern class*
A point $\boldsymbol{x}$ is outside the boundary of the pattern class C if the nearest neighbor of $\boldsymbol{x}$ in S_C is at a distance greater than $\alpha\delta^C_{avg}$, where $\alpha \geq 1$ is a predefined constant. Otherwise, it is inside the boundary of C.

The average edge length of an MST generally gives us some information about the sparseness of the data points present in the pattern class. If S_i forms a nice cluster then clearly δ^i_{avg} will give a good idea about the inter point distances. Even if S_i forms a few well separated clusters, and if the density of points in each cluster is almost the same then also δ^i_{avg} will give information about the sparseness of data points in each cluster. Thus δ^i_{avg} will not be much affected by the number of clusters and the "distance" between the clusters present in S_i, for large $|S_i|$. We state this more clearly in the next two propositions.

Let S_i consist of k clusters, S^l_i, $l = 1, 2, ..., k$. Let $\mathcal{T}_i$ be an MST on S_i and let T^l_i,

Table 6.3: Algorithm Augment

Algorithm **AUGMENT**($\mathcal{M}_{init}$, X_{new})

$X^N_{new} = S_{k+1} \cup S_{k+2} .... \cup S_{k+m}$;

(where S_{k+i}, contains points from class $k+i$)

for $i = k$ to $k+m$

train subnet $M^i_{additional}$ with S_i and points outside the boundary of S_i;

end for

simple merge $M^1_{init}, M^2_{init}, ..., M^k_{init}, M^{k+1}_{additional}, ..., M^{k+m}_{additional}$ to get $\mathcal{M}_1$;

$l(\boldsymbol{x}_i)$ be the class label of $\boldsymbol{x}_i$;

$U = \{1, 2, ..., k\}$ (the set of initial classes)

$I = \phi$;

$S_i = \phi, \forall i \in U$;

for all $\boldsymbol{x}_i \in X^O_{new}$;

if **TEST**($\mathcal{M}_1$,$\boldsymbol{x}_i$) cannot decide $\boldsymbol{x}_i$

put $\boldsymbol{x}_i$ in $S_{l(\boldsymbol{x}_i)}$;

if $l(\boldsymbol{x}_i) \notin I$

put $l(\boldsymbol{x}_i)$ in I;

end if;

end if;

end for;

if $I = \phi$ call $\mathcal{M}_1$ as M_{final} and *stop*;

for all $i \in I$

train subnet M^i_{new} with S_i and points outside the boundary of S_i;

end for;

$\forall i \in I$ Compound merge M^i_{new} with subnet M^i_{init} of $\mathcal{M}_1$ to get M_{final} ;

end

$l = 1, ..., k$, be the subsets of the edges in $\mathcal{T}_i$ that spans the points in cluster S_i^l. We assume that clusters are well separated, $|S_i|$ is large and number of clusters are much smaller than $|S_i|$. Under these assumptions we can safely say that $\mathcal{T}_i$ has only $k-1$ edges such that the two end points are in two different clusters. The remaining $|\mathcal{T}_i| - (k-1)$ edges, i.e., $|S_i| - k$ edges have both end points belonging to the same cluster. Then we can say:

Proposition 1: *T_i^l is the MST of points in S_i^l.*

Proof: If not, let τ_i^l be the MST on S_i^l. From the MST $\mathcal{T}_i$ we remove the subtree T_i^l and add τ_i^l to get $\mathcal{T}_i^{new}$. Thus,

$$\mathcal{T}_i^{new} = (\mathcal{T}_i - T_i^l) \cup \tau_i^l.$$

Now, $\mathcal{T}_i^{new}$ is a spanning tree of S_i. Also, if $W(T)$ denote the total weight of a tree T. Then by construction

$$W(\mathcal{T}_i^{new}) < W(\mathcal{T}_i),$$

as $W(\tau_i^l) < W(T_i^l)$. Hence, $\mathcal{T}_i$ is not an MST of S_i. Which is a contradiction.

Thus each T_i^l is a MST of the points in S_i^l. Let δ_{avg}^i be the average edge length of the MST $\mathcal{T}_i$ then we can arrive at the following proposition.

Proposition 2: *For large $|S_i|$, δ_{avg}^i is independent of the length of the MST edges connecting two different clusters.*

Proof : Let the average edge length of T_i^l be δ_l, and let the lengths of the $k-1$ edges which connect points from two different clusters be e_j, $j = 1, ..., k-1$. Let n_l be the number of points present in cluster l, $l = 1, 2, ..., k-1$. Hence the average edge length of $\mathcal{T}_i$ would be:

$$\delta_{avg}^i = \frac{1}{|S_i| - 1} \left\{ \sum_{l=1}^{k} (n_l - 1)\delta_l + \sum_{j=1}^{k-1} e_j \right\}. \tag{6.2}$$

Let δ^* be the average edge length of all the edges in $\mathcal{T}_i$ which connects points belonging to the same cluster, i.e., δ^* is the average edge length of all the MST's T_i^l, $l = 1, 2, ..., k$. Then

$$\delta^* = \frac{\sum_{l=1}^{k} (n_l - 1)\delta_l}{\sum_{l=1}^{k} (n_l - 1)} \tag{6.3}$$

$$= \frac{\sum_{l=1}^{k} (n_l - 1)\delta_l}{|S_i| - k}. \tag{6.4}$$

Hence we get,

$$\delta^i_{avg} = \frac{1}{|S_i| - 1} \left\{ (|S_i| - k)\delta^* + \sum_{j=1}^{k-1} e_j \right\} \tag{6.5}$$

$$= \frac{\delta^* \left(1 - \frac{k}{|s_i|}\right)}{\left(1 - \frac{1}{|s_i|}\right)} + \frac{\sum_{j=1}^{k-1} e_j}{|S_i| - 1} \tag{6.6}$$

Here, $k \ll |S_i|$ and $\sum_{j=1}^{k-1} e_j$ is finite. Thus,

$$\lim_{|S_i| \to \infty} \delta^i_{avg} = \delta^*. \tag{6.7}$$

Hence, for large $|S_i|$ the average edge length of $\mathcal{T}_i$ depends on the average length of the edges in each MST of the clusters present in S_i. Again if we assume, that the density of the points in each of the k clusters is same and hence the average edge lengths of the MST's of the points in each cluster is the same (say, δ). Then we have

$$\delta^* = \delta,$$

thus δ^i_{avg} is equal to the average edge length of each MST. Hence, for large $|S_i|$, the average edge length of an MST is independent of the number of clusters present in the set. So δ^i_{avg} can be used as a good measure of sparseness of the data even if S_i contains clusters in it.

Next we summarize the method of generation of points outside the boundary of a pattern class in procedure GENERATE (Table 6.4). GENERATE takes as input the inflated hyperbox (Ω) of the total data and a set of points S_C in the pattern class C.

The multiplier α in GENERATE controls the tightness of the boundary, *i.e.*, a smaller value of α will yield a tighter boundary than a larger value of α. To demonstrate the effect of α on the boundary we use a new data set called **Square**, which contains 500 points in $\Re^2$, generated randomly over a square. The scatterplot of Square is shown in Fig. 6.6. Figure 6.7 shows the points generated outside the boundary of Square for different values of α. Figure 6.8 shows the points generated outside the boundary of Dish-Shell, assuming that the entire data are from one class (scatterplot in Fig. 6.3). Figures 6.7 and 6.8 clearly exhibit that the algorithm can capture the concept of the boundary and its performance is quite good for $\alpha \in [1.5\ ,\ 2.5]$. So, in our simulations we use $\alpha = 2$.

Table 6.4: Algorithm Generate

Algorithm **GENERATE** (Ω, S_C)

begin

Select α;

Compute MST of the points in S_C;

Compute δ_{avg}^{C} ;

Select nCount; count =0 ;

while($count < nCount$)

Randomly generate $\boldsymbol{\xi}$ s.t. $\boldsymbol{\xi} \in \Omega$;

Find $\boldsymbol{x} \in S_C$ s.t. $||\boldsymbol{\xi} - \boldsymbol{x}|| = \min_{\boldsymbol{z} \in S_C} ||\boldsymbol{\xi} - \boldsymbol{z}||$;

if ($||\boldsymbol{\xi} - \boldsymbol{x}|| < \alpha \delta_{avg}$);

discard $\boldsymbol{\xi}$;

else

accept $\boldsymbol{\xi}$ as a point outside the boundary of C ;

$count = count + 1$;

endif

end while

end

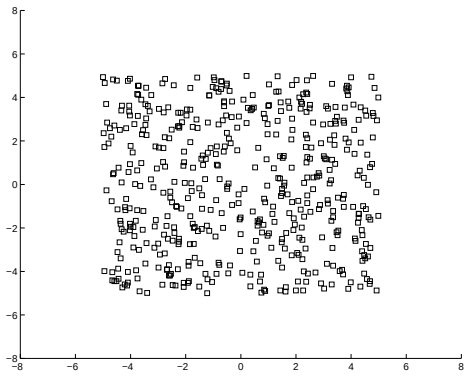


Figure 6.6: Scatterplot of Square

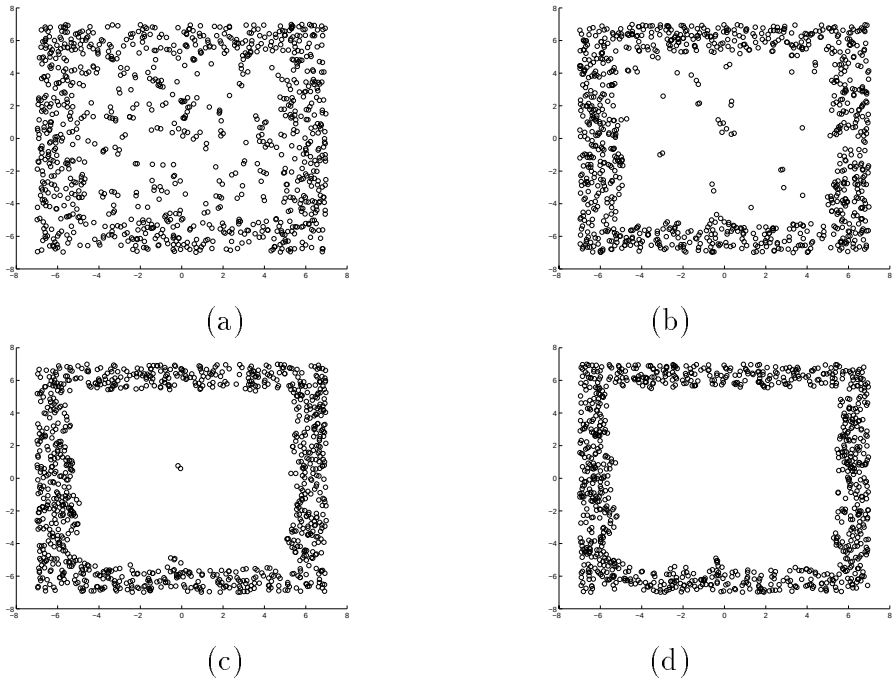


Figure 6.7: Points generated outside the boundary of Square for different values of α: (a) $\alpha = 1.0$ (b) $\alpha = 1.5$ (c) $\alpha = 2.0$ (d) $\alpha = 2.5$

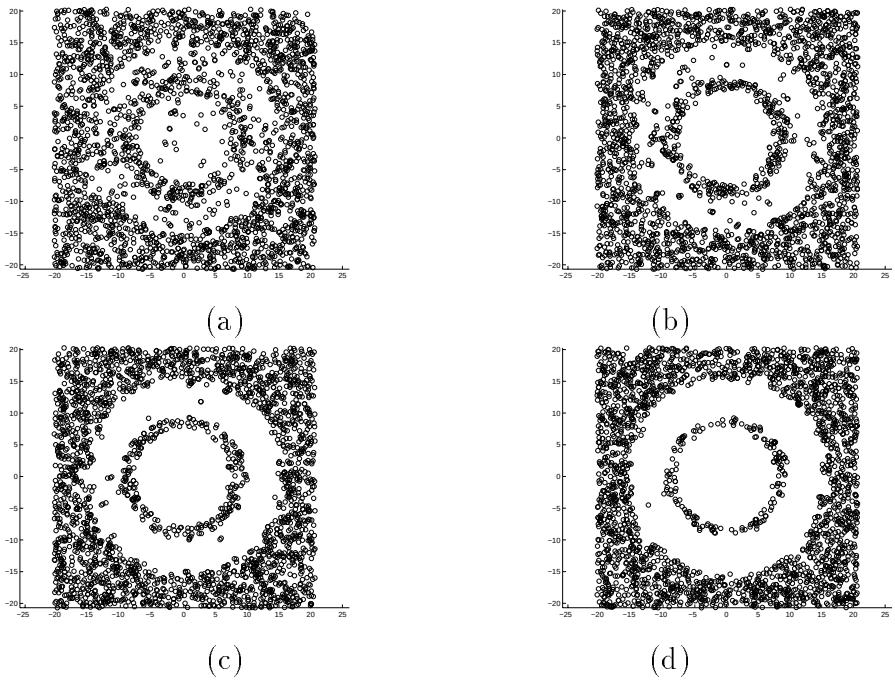


Figure 6.8: Points generated outside the boundary of Dish-Shell for various values of α: (a) $\alpha = 1.0$ (b) $\alpha = 1.5$ (c) $\alpha = 2.0$ (d) $\alpha = 2.5$

6.5 Results

We present here results on 5 data sets. The data sets are named **Dish-Shell**, **3D-Elongated**, **Cone-Torus**, **Sat-Image** and **Scattered**. We have some objectives in mind behind the choice of these data sets. The first two data sets are in low dimension and they have well separated classes. We use them to demonstrate the generalization ability of our scheme. And, as they are in low dimension we pictorially demonstrate the generalizations produced by our method. Cone-Torus and Sat-Image have considerable overlap between classes. These two data sets have been previously used by many researchers to evaluate different classifiers [122]. Hence, using Cone-Torus and Sat-Image we compare the performance of our method with that of a normal MLP. Also, we show that our method can handle overlapped classes in a better manner and can signal that a test point lies in the area of overlap. The results demonstrating strict generalization are presented in the next subsection. In a subsequent subsection we use the synthetic data set, Scattered, to demonstrate the incremental learning ability of our network.

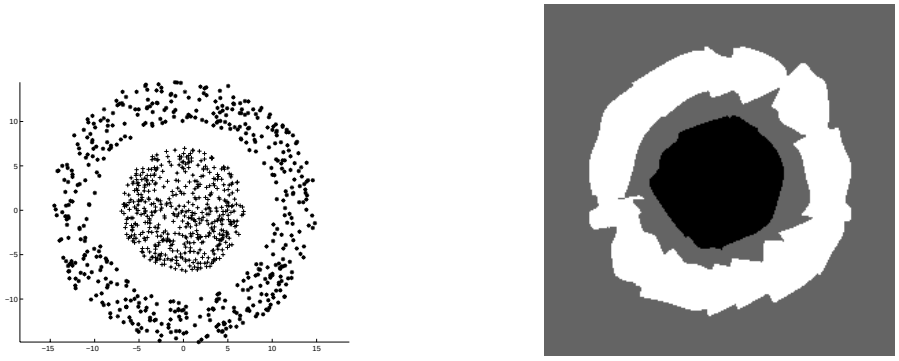
In all the simulations we use networks with sigmoidal node functions and use the Lavenberg Marquard algorithm [69] for training MLPs.

6.5.1 Demonstration of Good Generalization

Dish-Shell

This data set, as already stated, consists of 1000 points in $\Re^2$ equally distributed in two classes. The scatterplot of the data set is shown in Fig. 6.3(a), and for convenience it is again reproduced as Fig. 6.9(a), the points from the two classes are represented by + and *.

As Dish-Shell contains two classes we need to train two subnets. For each of the classes we generated 2500 points outside the boundary and trained two MLPs each with 20 nodes in the hidden layer and then they were merged by a simple merge. The generalization performed by the network is shown in Fig. 6.9(b). Comparison of Fig. 6.9(b) with Fig. 6.3(b) (generalization produced by conventional MLP on the same data set) reveals that the proposed method can do an excellent generalization.



(a) (b)

Figure 6.9: (a) Scatterplot of Dish-Shell (b) Generalization on Dish-Shell

3D-Elongated

3D-Elongated is a data set in $\Re^3$ having two classes. Each class has 500 points (this data set is similar to a data set used in [149]). Figure 6.10 shows the projection of the data on various planes.

We generate 3500 points outside the boundary of each class and train two MLPs, one for each class. Then these two nets are merged by a simple merge to obtain the final network. To test the network we randomly generate 100000 data points in the hyperbox bounding the data. We test the trained network with these data points. For 8848 data points the response of the network is high and is significantly low for other points. The plot of those 8848 points are shown in Fig. 6.11. Comparing Fig. 6.10 with Fig. 6.11 we see that the proposed scheme results in an excellent generalization.

Cone-Torus

Cone-Torus has 400 points in $\Re^2$ in both the training and test sets [122, 258]. There are three classes each representing three shapes, namely, a cone, a half torus and a Gaussian. We train 3 subnets each with 5 hidden nodes using the data in each class along with 400 points generated outside the boundary. The scatterplot of this data (Fig. 6.12) shows that two of the classes have considerable overlap. Table 6.5 shows the results of our network with this data set as interpreted by our procedure TEST. In Table 6.5, column 2 lists the number of cases for which the network unambiguously

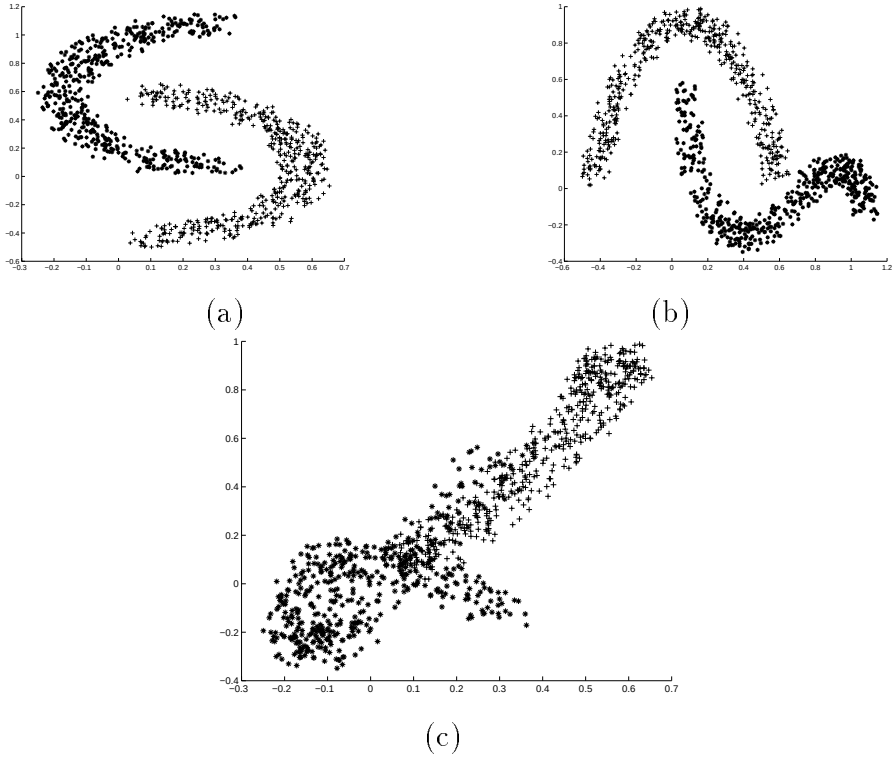


Figure 6.10: Scatterplot of 3D-Elongated (a) projected on 1-2 (b) projected on 2-3 (c) projected on 1-3

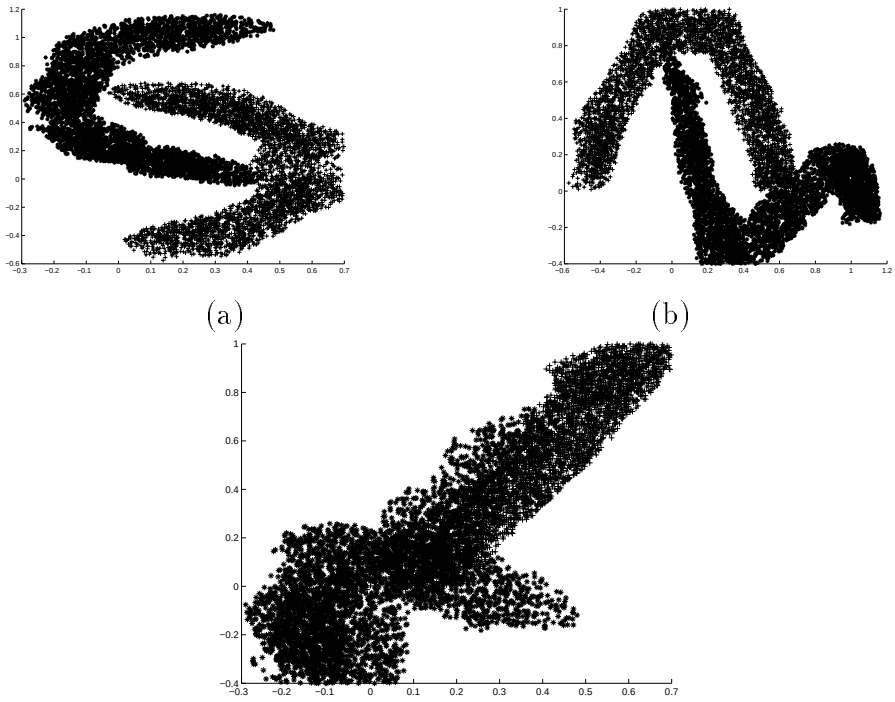


Figure 6.11: Generalization on 3D-Elongated (a) 1-2 (b) 2-3 (c) 1-3

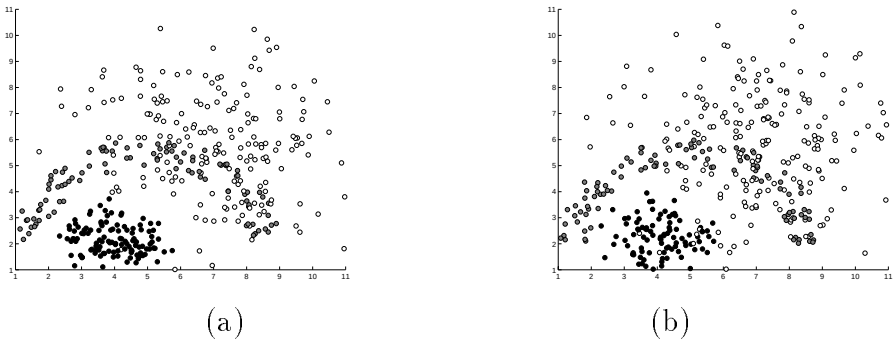


Figure 6.12: Scatterplot of Cone-Torus (a) Training Data (b) Test Data

detects the correct class, column 3 shows the number of instances for which the network suggests two classes including the correct class. In other words, column 3 represents the data points that are suspected to be in the overlapped area. Column 4 suggests that there is no area in the input space where all three classes overlap. This indeed matches with the class structure shown in Fig. 6.12. The column labeled "soft correct" depicts the number of data points that are correctly classified but the decisions are soft. Column 6 gives the number of points on which the network could not make any decision and column 7 gives the total number of points on which soft decisions were made. The last column gives the total number of misclassifications when the decisions are made like a conventional MLP, i.e., we take the maximum output of the network to decide the class label of a test point. Comparing column 8 of Table 6.5 with Table 6.6 (which gives the performance of ordinary MLP on Cone-Torus as reported in [122]) we find that our network can perform as good as ordinary MLP, in addition it can provide a deeper insight into the data. For example, of 59 misclassifications on the test data, 19 are declared as undecided indicating that probably these 19 points are not in the vicinity of the training data. If we take the maximum response of the output units to decide the class we find that out of these 19 undecided points only 6 are classified correctly. Thus of the 59 misclassifications 13(19-6) points were declared as undecided by our network. Of the remaining 46 (59-13) points 8 (39-31) are wrongly classified by the soft decision and the other 38 points probably fall in the overlapped region where the network suggested two possible classes as output.

Table 6.7 shows the results obtained by a conventional MLP with 15 nodes in the hidden layer, but the network outputs are interpreted by our TEST procedure. Comparing Tables 6.5 and 6.7 we see that our method gives high single class response to more points than the MLP for both training and test data. Table 6.7 fails to point out the overlapped cases and as expected many of the soft decisions are incorrect. For example, of the 238 soft decisions 65 are incorrect for the training data; similarly, out of 167 soft decisions for the test data, 58 are wrong. The number of undecided cases are also lower than that by our network. This may be attributed to the fact that when the network is confronted with similar data points but from different classes, it may learn one of the two classes (it cannot learn both classes). Thus NOT for a SINGLE data point we find high response for more than one class.

Table 6.5: Results on Cone-Torus by our method

	1 class	2 class	3 Class	Soft Correct	Undecided	Tot. Soft	Miss
Training	258	104	0	35	2	36	50 (12.5 %)
Test	241	101	0	31	19	39	59 (14.75 %)

Table 6.6: Results on Cone-Torus using conventional MLP (reported in [122])

Architecture	Trg error	Test error
2:10:3	15.25 %	14.25%
2:15:3	13.50 %	12.0%

Table 6.7: Results on Cone-Torus using conventional MLP interpreted by TEST

	1 class	2 class	3 Class	Soft Correct	Undecided	Tot. Soft	Mis
Training	161	0	0	173	1	238	66 (16.5%)
Test	228	0	0	109	5	167	63 (15.75 %)

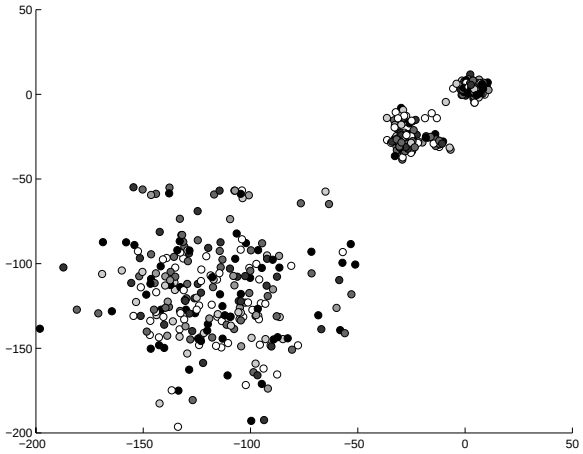


Figure 6.13: Scatterplot of Sat-Image along the two most significant components

Table 6.8: Results on Sat-Image using conventional MLP (reported in [122])

Architecture	Trg error	Test error
4:20:6	79.2 %	75.92%
4:65:6	24.4 %	23.08%

Sat-Image

The Sat-Image data set is generated from Landsat Multi-Spectral Scanner image data [122, 128, 256, 257]. The present data set covers an area of 82×100 pixels portion of the whole image. Each feature vector has 4 components containing the gray value of a pixel captured by 4 sensors operating in different spectral regions. The data set has 6 classes representing different kinds of ground covers. The training set has 500 points and the test set has 5935 points. In the literature there are other studies which use only these 4 features [128].

In this data set the classes have significant overlap as evident from Fig. 6.13, which shows the scatterplot of the data along the two most significant principal components (the various shades of gray shows the various classes). Some results on this data using conventional MLP can be obtained in [122], we summarize these results in Table 6.8. We also ran a conventional MLP with this data with various architectures. We got better results than what was reported in [122]. Table 6.9 shows the results obtained

Table 6.9: Results on Sat-Image using conventional MLP obtained by us

Architecture	Trg error	Test error
4:65:6	73 (14.6 %)	1012 (17.0%)
4:50:6	155 (31.0 %)	1917 (32.3%)
4:30:6	171 (34.0 %)	2160 (36.39%)
4:15:6	308 (61.6 %)	3718 (62.64%)

Table 6.10: Results on Sat-Image using conventional MLP interpreted by TEST

Arch.		1-class	2-class	3-class	4-class	Undecided	Soft correct	Mis
4:65:6	Trng	320	0	0	0	8	107	73
	Test	3750	0	0	0	198	1168	1012
4:50:6	Trng	207	0	0	0	90	136	155
	Test	2390	0	0	0	1054	1540	1917
4:30:6	Trng	57	0	0	0	14	272	171
	Test	537	0	0	0	167	3231	2160
4:15:6	Trng	0	0	0	0	0	192	308
	Test	537	0	0	0	167	2217	3718

by us with conventional MLP's on this data set. Table 6.10 shows the results of these MLPs as interpreted by our procedure TEST.

We shall describe two different runs with our network which we call Run-I and Run-II respectively. In Run-I, we generate 5000 points in $\Re^4$ in the boundary of each class C_i and additionally we take the points from the classes $C_j, j \neq i$ which lie *outside* the boundary of C_i. Notice that, as the dimensionality of the input space increases, more and more data points are to be generated outside the boundary to properly represent the geometric structure of a class and consequently training of the subnets becomes more expensive computationally. An easy way to bypass this is to consider the points of the other classes which lie outside the boundary of the class in question. These data points are very important to determine the structure of the class. In Run-I we use $\alpha = 2$, for generating points outside the boundary and also for considering the

Table 6.11: $|\bar{C}_i|$ for Run I and Run II

	$\lvert\bar{C}_1\rvert$	$\lvert\bar{C}_2\rvert$	$\lvert\bar{C}_3\rvert$	$\lvert\bar{C}_4\rvert$	$\lvert\bar{C}_5\rvert$	$\lvert\bar{C}_6\rvert$
Run I	379	342	342	288	212	316
Run II	394	417	373	381	403	365

points of other classes. The results of our method for Run-I are shown in Table 6.12. Table 6.12 reveals that there is significant overlap between the classes, as we find many points with multiple class labels. To obtain a more specific result, i.e., to get more points classified with single class label, we consider Run-II. In Run-II, we generate 5000 points outside the boundary with $\alpha = 2$, and consider the data points from other classes with $\alpha = 1$. The results of Run-II are shown in Table 6.13.

We will analyze the results on Sat-Image in two parts. First we will explain why Table 6.12 is so different from Table 6.13. Then we will compare Table 6.13 with Table 6.10.

Let S_i be the set of training data points from class i and $\bar{C}_i$ be the set of training points from the remaining classes which lies outside the boundary of S_i. If the classes are not overlapped then $\bar{C}_i$ will be equal to $\cup_{j \neq i} S_j$. Table 6.11 compares Run I and Run II in terms of the number of points for $\bar{C}_i$ considered by our algorithm for two different values of α. Low value of α defines a tight boundary for each class. For example, in case of class 5, we find that for $\alpha = 2$, only 212 points in $\bar{C}_5$ are considered to be outside the boundary of class 5 and this is increased to 402 (it becomes almost double) for $\alpha = 1$. This tells us that either class 5 has some overlap with the remaining classes or the boundary of class 5 is probably "touching" the boundary of other classes. Consequently for this data set with $\alpha = 1$ we expect to get a better result. Next we shall see that, this is indeed the case. But before that it is worth mentioning that using two different values of α one can get some idea about whether different classes are well separated or not. Comparing Tables 6.12 and 6.13, we find that use of a tighter class boundary improves the performance of the system drastically. Tighter boundary also reduces ambiguous choices. For example, $\alpha = 2$ results in 79 cases for the training data where our system suggests 3 classes and this 79 is reduced to 0 for $\alpha = 1$. For the test data 864 3-class cases are reduced to just 5.

Table 6.12: Results on Sat-Image by our method (Run -I)

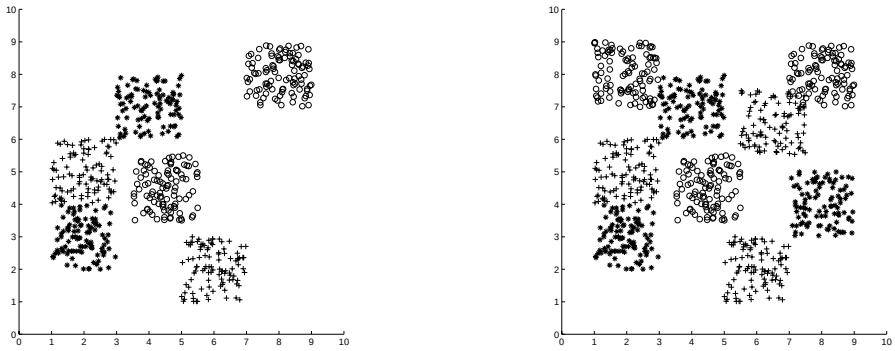
	1-class	2-class	3-class	4-class	5-class	Undecided	Soft correct	Total Soft	Miscl.
Trng	251	126	79	4	0	1	31	39	90(18%)
Test	2712	1353	864	14	2	420	413	570	1246(20.9 %)

Table 6.13: Results on Sat-Image by our method (Run -II)

	1-class	2-class	3-class	4-class	5-class	Undecided	Soft correct	Total Soft	Miscl.
Trng	384	82	0	0	0	13	16	21	70 (14.0 %)
Test	3865	960	5	0	0	826	205	279	1211 (20.4 %)

A tighter boundary increases the number of undecided cases (the system declines to classify a test point in an area not well supported by training data). Of the 1211 misclassifications that are obtained by taking the maximum output to decide the class, 826 cases are undecided. Of these 826 cases only 213 points are classified correctly if the decision is made based on the maximum response.

Analysis of Table 6.10 reveals that the use of TEST to interpret the output of a conventional MLP does not result in 2-class, 3-class cases - this behavior is exactly the same as that of Cone-Torus data set. The misclassifications even with a high architecture like (4:65:6), is comparable to that of our system. One may think, since we are using several networks probably our system has more free parameters - *this is just the opposite*. Note that, a (4:65:6) MLP has 650 learnable weights and 71 biases, i.e., a total of 721 parameters whereas our network has 300 weights and 66 biases, i.e., only 366 parameters - about half that of the net (4:65:6)!



(a) (b)

Figure 6.14: (a) Scatterplot of Scattered_P1 (b) Scatterplot of Scattered

6.5.2 Demonstration of Incremental Learning

Scattered

With this data set we shall demonstrate the incremental learning capabilities of the proposed scheme. Suppose Scattered is obtained in two phases. In the first phase the data set has 600 points equally distributed in 3 classes, we call this data set as Scattered_P1. Further 300 points are added to it to get the final data set Scattered. The scatterplot of both data sets are shown in Fig. 6.14. Figure 6.14(b) shows that the addition of 300 points changes the class structure drastically. Our objective is to train a network $\mathcal{M}_1$ which can classify the points in Scattered_P1 and then augment $\mathcal{M}_1$ to a new network $\mathcal{M}_2$ to classify all points in Scattered. The augmentation will be done using only the additional 300 points.

Since Scattered_P1 has 3 classes our first network $\mathcal{M}_1$ consists of 3 subnets. Each subnet is trained with the data points in each class of Scattered_P1 along with 2000 point generated outside the boundary of each class. The generalization produced by $\mathcal{M}_1$ on Scattered_P1 is shown in Fig. 6.15(a). Then we get the additional 300 points, 100 points in each of the existing 3 classes. We tried to classify them by $\mathcal{M}_1$. But none of the 300 points gets classified by $\mathcal{M}_1$. Hence, we conclude that the 300 points obtained later though belong to the set of classes already present in Scattered_P1, they lie outside the boundary of the classes represented by Scattered_P1. Figure 6.14(b) shows that this is indeed the case. So, we train 3 new subnets, each with 100 data points and 2000 points generated outside the boundary of each class. These subnets

(a) (b)

Figure 6.15: (a) Generalization on ScatteredP1 by $\mathcal{M}_1$ (b) Generalization by $\mathcal{M}_2$ on Scattered

are merged with the 3 subnets of $\mathcal{M}_1$ by a compound merge and we call the new network as $\mathcal{M}_2$. Figure 6.15(b) shows the generalization achieved by $\mathcal{M}_2$ on Scattered. Figure 6.15(b) exhibits an excellent performance of our scheme in terms of incremental learning ability and generalization capability. In the given example, the classes are well separated. In case the classes have overlap, the discussion made in Section 6.4 is equally applicable here.

6.6 Conclusions and Discussion

In this chapter we proposed a novel scheme to train an MLP so that it does not respond to data points which lie far from the training sample. The training scheme also equips the MLP with incremental learning capability. Unlike conventional MLP, it can detect data points that fall in the overlapped regions. Our method is based on training several nets with simpler tasks and then merging them for the complete task expected of the network. In this context we have proposed two merging schemes. The philosophy behind the proposed scheme is quite general and can be used with other learning machines also. The crucial point in the scheme is a method of generating points outside the "boundary" of a pattern class. As stated next, there are a few issues that have not been adequately addressed in this work.

We have given a naive definition of the boundary of a pattern class and proposed an algorithm to generate points outside the boundary of a pattern class represented

by a given set of points. The proposed algorithm although simple, works quite well. This problem of generating points outside the boundary of a pattern class itself is quite interesting but a difficult problem which needs further investigations. Another important problem, estimation of the number of points to be generated outside the boundary, for proper training has also not been addressed in the present work. The number of points to be generated has been chosen in an ad hoc manner considering the size of the inflated hyperbox of the data.

An important characteristic of the method is that it can detect points in the area of overlap of two or more classes and can thus deal with overlapped classes in a better way than the conventional MLP. Moreover, if we have a data set with two touching classes, a conventional MLP or any other classifier will be able to arrive at zero misclassification for all the training points. But in our method the points from either class which lie near the separating plane of the classes will be classified into both classes. In such cases, the classifier which produces zero misclassification on training data is very sensitive to the training data, because, if we give a little perturbation to the points near the separating plane, they will change classes. We think this is *not* desirable and is an indicator of bad generalization.

Generation of points outside the boundary of a class for high dimensional data sets may pose certain practical problems. As the dimensionality of the data increases, one has to generate more points to characterize the boundary of a class. This restricts the use of this method for reasonably high dimensional data sets. For the Sat-Image data we used a simple scheme to overcome this difficulty. We considered the points in other classes along with the points generated outside the boundary of a class. The results demonstrate that this scheme works quite well. In the next chapter we discuss strict generalization for MLP networks when used for function approximation problems. There we propose an alternative formulation to characterize the boundary of the training sample. This formulation does not require any additional points to be generated outside the boundary and so can be used easily for high dimensional data sets. The method formulated in the next chapter will also be demonstrated for classification task.

This chapter aimed at testing the feasibility of a new training scheme for MLPs which can realize strict generalization and support incremental learning. The simulation results demonstrate that our training scheme serves the purpose quite satisfactorily. In

the next chapter we discuss a modification of this scheme to solve function approximation type problems.

Chapter 7

Strict Generalization and Incremental Learning in Multilayer Perceptron Networks: Function Approximation[1]

7.1 Introduction

In the previous chapter we discussed strict generalization and incremental learning for MLP networks. The training methodology that we developed in Chapter 6 is applicable to classification problems only, as the scheme involves training subnets for each class. Here we extend the methodology for function approximation (FA) problems.

For function approximation problems, our method involves training a network for two different tasks: (a) to learn the input-output mapping present in the training set, and (b) to learn the boundary of the training set. In other words, the network performs the task of function approximation, additionally it learns a decision boundary as in classification problems. To realize this we use two MLP networks together. We call the first network which learns the input-output mapping as the *mapping network* and the other network which learns the decision boundary as the *vigilance network*. The vigilance network can be trained using the same scheme of generating points outside the boundary of the training sample (see Chapter 6). The points in the training set serve as positive examples (i.e., points inside the boundary) and the points generated outside the boundary are considered negative examples. Thus a trained vigilance network will have the capability to detect whether a test point falls within the boundary of the

[1]The contents of this chapter have been communicated in [32].

training set or not.

Generating points outside the boundary of the training set becomes quite expensive for reasonably high dimensional data sets. In such cases one has to generate a huge number of points outside the boundary to get a proper characterization of the boundary of the training set. Thus, this method becomes computationally very expensive for high dimensional data. We propose another method to train the vigilance network which does not require generation of additional points, but it involves decomposing the training sample into small subsets, and making the vigilance net learn the boundary of such sets. A vigilance network trained in either way can be combined with a mapping network to realize strict generalization for FA problems. The MLPs trained in this manner also offer *incremental learning* as a byproduct.

In the following sections we discuss our methods in details. In Section 7.2 we begin with a compelling example using a synthetic data set to motivate the problem. In Section 7.3 we discuss our schemes for realizing strict generalization and in Section 7.4 we describe the incremental learnability of the scheme. Section 7.5 provides some simulation results and Section 7.6 concludes the chapter.

7.2 The 3-Peaks Function: The Motivation

Let us consider the function:

$$y = 0.2e^{-\left(\frac{x-50}{10}\right)^2} + 0.4e^{-\left(\frac{x-25}{5}\right)^2} + 0.4e^{-\left(\frac{x-75}{5}\right)^2}. \tag{7.1}$$

We call this function as 3-Peaks. Fig. 7.1 depicts the function 3-Peaks. We sample a few points from the function in eq. (7.1) to train an MLP. Intentionally we sample points in such a manner that there remains a gap in the input space. Figure 7.2 shows the sampled points, we call this set of points as PT_1. The MLP trained with these sampled points are tested on a data set which contains 1000 equispaced points generated in [0,100]. Figure 7.3 shows the generalization done on the test data by four MLPs trained with different initializations. From Fig. 7.2 it is clear that the interval [40, 60] is not represented by any training data, so the MLP is not expected to perform well over this interval. Figure 7.3 shows some queer generalizations. Specially the generalization shown in Fig. 7.3(b).

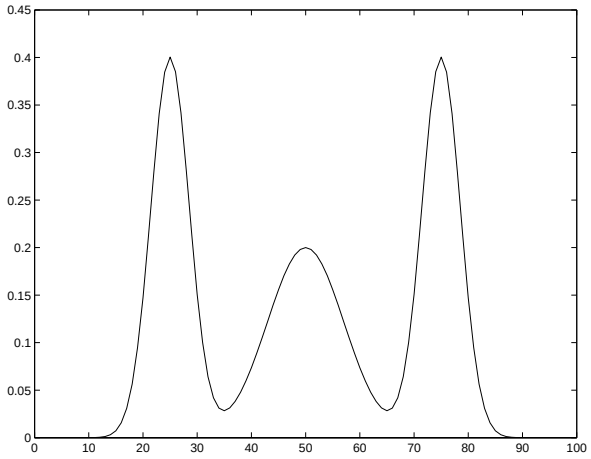


Figure 7.1: Plot of 3-Peaks

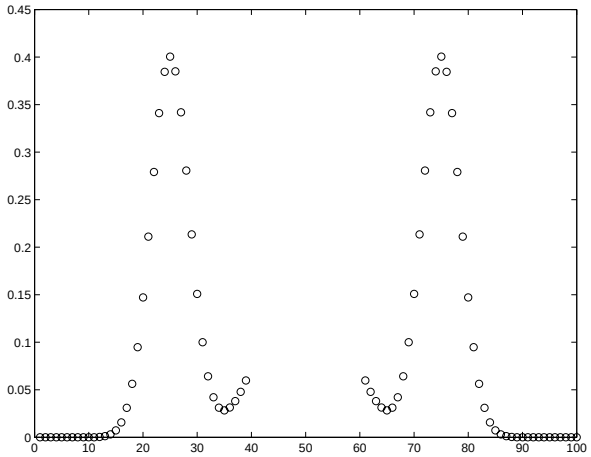


Figure 7.2: The points in 3-Peaks used for training

When training data are collected from a live process then there may remain areas in the input space which are not well represented by the training data or are not *at all* represented by the training data. For test points which lie in those areas, ideally, an MLP should not respond at all. But an MLP, as shown in Fig. 7.3, will always produce some output. When an MLP is used for classification, we may interpret the output of an output layer node as a confidence in favor of a class (although it may not produce desired results). But when an MLP is used for FA type applications we cannot do so. We device a mechanism here which can take care of this problem.

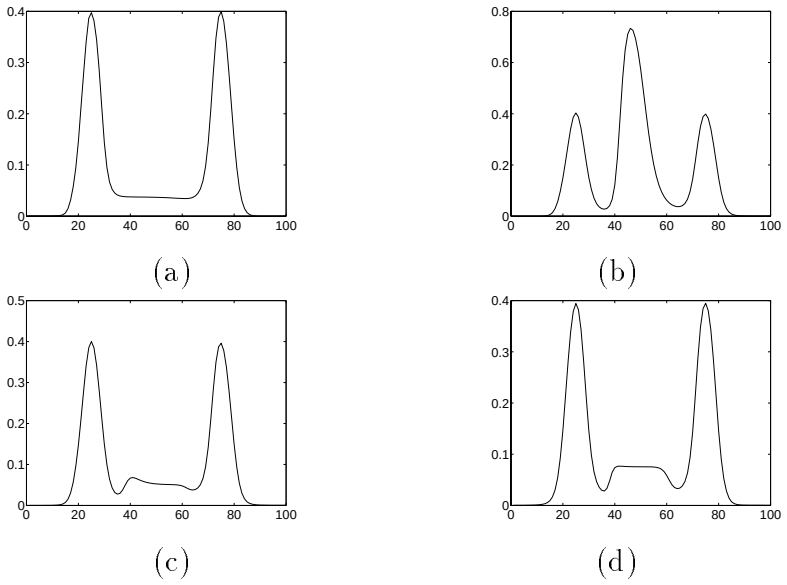


Figure 7.3: Generalization produced by an ordinary MLP trained with PT_1 for 4 different initializations

7.3 Training Scheme

Let $T = (X, Y) = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), (\boldsymbol{x}_2, \boldsymbol{y}_2), ..., (\boldsymbol{x}_N, \boldsymbol{y}_N)\}$ be our training set with N training samples where $\boldsymbol{x}_i \in \Re^s$ be an input vector and $\boldsymbol{y}_i \in \Re^t$ be the corresponding output vector. The task here is to learn the unknown input-output mapping that exists between $\boldsymbol{x}$ and $\boldsymbol{y}$. An ordinary MLP trained with conventional backpropagation or any other method can accomplish the task with a reasonable accuracy for almost all kinds of data. But we have an additional objective. We want to train an MLP in such a manner that it does not respond to test points which are away from the boundary of X. This can be realized if we can make the MLP learn the boundary of X along with the input-output mapping between $\boldsymbol{x}$ and $\boldsymbol{y}$. Thus, we want our network to learn a decision boundary as in case of classification problems. To realize this we use two networks. The first one is an usual MLP which learns the input-output mapping, we call this as the *mapping* network. The second network is called the *vigilance net* which decides whether the MLP should respond to a point or not. The final output for a test point is obtained by suitably combining the outputs of both networks. Training of the vigilance network can be done in two ways. The first approach requires additional training points generated outside the boundary of the training set. We call a vigilance

network trained in this manner as the *boundary vigilance network* (BVN). The second approach does not need any additional training examples but it builds receptive fields around clusters of data points and the responses of the receptive fields are aggregated to decide the position of a test point with respect to the training sample. A vigilance network trained in this fashion is called the *receptive field vigilance network* (RVN). In the following two subsections we discuss the construction of BVN and RVN in details.

7.3.1 Training the BVN: Training Vigilance Nets with Additional Examples Generated Outside the Boundary of the Training Set

We have at hand only the training samples, i.e., some positive examples of the instances that lie inside the boundary of X (the set X itself). But, to make the MLP learn the boundary of X, we need additional points which lie outside the boundary of X. To be more precise, the additional points should be outside but close to the boundary of X. Let Ω_{small} be the smallest hypercube, which bounds X. By increasing each edge of Ω_{small} by $l\%$ on all sides, we inflate Ω_{small} to Ω, we call Ω as the inflated hyperbox of X. Our scheme makes "strict generalization" on all test data points which lie within Ω. Thus Ω is the space from which the data points are expected to come. One may have initial information of Ω (the domain of the input space) or one can arrive at any desired Ω by choosing an appropriate measure of inflation. Let Ξ be a set of points generated randomly within Ω but outside the boundary of X(one can use the method described in Section 6.4.2 for this). These points in Ξ serve as examples outside the boundary of X. The *vigilance* network is trained with points in $X \cup \Xi$. Since the vigilance network is meant only to learn the boundary of the training data, the original outputs associated with each point in X are ignored for its training. The output for every $\boldsymbol{x} \in X$ is taken as 1 and the output for every $\boldsymbol{\xi} \in \Xi$ is taken as 0. Thus, if properly trained, the output of the vigilance network will be able to detect whether a test point lies within the boundary of the training set or not. The method for training a BVN is summarized in Table 7.1.

Table 7.1: Algorithm Train BVN

Algorithm **Train BVN**

Given training set $T = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), ..., (\boldsymbol{x}_N, \boldsymbol{y}_n)\} \subset \Re^s \times \Re^t$;
Let $X = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\} \subset \Re^s$;

Fix, Ω, the inflated hyperbox of X;
Fix n_{hv} = Number of hidden nodes for the vigilance net;
Generate $\Xi = \{\boldsymbol{\xi}_1, ..., \boldsymbol{\xi}_m\} \subset R^s$, such that, $\boldsymbol{\xi}_i \in \Omega$, but outside the boundary of X using Algorithm GENERATE (Table 6.4) ;

Let, $TV = \{(\boldsymbol{x}_1, 1), ..., (\boldsymbol{x}_N, 1), (\boldsymbol{\xi}_1, 0), ..., (\boldsymbol{\xi}_m, 0)\}$;

Train an ordinary MLP, $\mathcal{N}_v$ with architecture $(p : n_{hv} : 1)$ with TV, call it the vigilance net;

7.3.2 Training the RVN: Training Vigilance Nets with Receptive Fields Around Data Points

As discussed in the previous subsection, the vigilance net can be trained to estimate the sampling window of the input distribution by points generated outside the boundary of the training sample. But, generation of points outside the boundary of the training set becomes infeasible for high dimensional data, as in such cases we have to generate a large number of points outside the boundary to characterize the shape of the sampling window. Here we propose an alternative way to construct the vigilance network, which does not need any additional training examples.

We can assume that the input vectors of the training set X can be divided into a number of hyperspherical clusters $X_i, i = 1, 2, ..., n$, such that

$$\cup_{i=1}^{n} X_i = X$$

and

$$X_i \cap X_j = \phi, \quad \forall i, j, i \neq j.$$

Such decomposition into hyperspherical clusters can be done using any conventional clustering algorithm like the k-means [48], or the Fuzzy c-means [12]. The vigilance net is trained in such a manner that it can detect whether a test point falls in any of these clusters or not.

This RVN is a three layered network. It has s nodes in the input layer (if $X \subset \Re^s$), k nodes in the hidden layer and one node in the output layer. Each node in the hidden layer has two parameters $\boldsymbol{\mu}_i \in \Re^s$ and $\sigma_i \in \Re$ associated with it. For a input vector $\boldsymbol{x}$, the i^{th} hidden node computes

$$z_i = exp\left(-\frac{||\boldsymbol{x} - \boldsymbol{\mu}_i||^2}{\sigma_i^2}\right), \quad \forall i = 1, 2, ..., k. \tag{7.2}$$

The single output node in the third layer aggregates the outputs of the k hidden nodes to give a single response. If the output of the third layer node be denoted by b, then

$$b = \max_{i=1,2,...,k} \{z_i\}. \tag{7.3}$$

Each node in the hidden layer represents a cluster in the data set X. The parameters $\boldsymbol{\mu}_i$ and σ_i are decided using the FCM algorithm. If we decide k as the number of

hidden nodes then, we find out k clusters from X and denote $\boldsymbol{\mu}_i, \quad i = 1, 2, ..., k$ as the i^{th} cluster center. As discussed in the appendix of Chapter 4, FCM produces a set of centroids $\boldsymbol{V} = \{\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_k\}$, and a partition matrix $U = [u_{ij}]_{k \times N}$, where u_{ij} denotes the degree to which $\boldsymbol{x}_j$ belongs to the i^{th} cluster and $\boldsymbol{v}_i$ is the centroid of the i^{th} cluster. Here we take $\boldsymbol{\mu}_i = \boldsymbol{v}_i$. The fuzzy partition matrix obtained from FCM can be hardened using the maximum membership rule [12]. In other words, we can consider that a point $\boldsymbol{x}_i \in X$ belongs to cluster $c, 1 \leq c \leq k$, if

$$u_{ci} = \max_j \{u_{ji}\}.$$

So, the clustering algorithm partitions X into k disjoint sets $X_1, X_2, ..., X_k$. σ_i is chosen as the distance of $\boldsymbol{\mu}_i$, from the point which is farthest to $\boldsymbol{\mu}_i$ in X_i. Thus,

$$\sigma_i = \max_{\boldsymbol{x}_j \in X_i} \{||\boldsymbol{x}_j - \boldsymbol{\mu}_i||\}, \forall i = 1, 2, ..., k. \tag{7.4}$$

For a test point $\boldsymbol{x} \in \Re^s$ each hidden node in the vigilance network gives an output related to the distance of $\boldsymbol{x}$ from the cluster center that the node represents. Thus, if a test point lies in or around the boundary of the cluster that a hidden node represents, then the output of that hidden node will be high. Therefore, for a test point $\boldsymbol{x} \in \Re^s$, if b takes a high value then we conclude that $\boldsymbol{x}$ lies within or around some cluster of X; otherwise, it lies far from all k clusters of X. So, b can be used as an indicator of whether $\boldsymbol{x}$ lies in or around the boundary of X.

7.3.3 The Composite Network

Another network is trained along with the vigilance network. This second network is an ordinary MLP, which is trained with the points in X along with its associated output, i.e., with (X, Y). This network is called the *mapping* network (maps input to output). The vigilance network and the mapping network are combined together to a composite network which makes the final decision. Denoting the trained vigilance network as $\mathcal{N}_v$ and the mapping network as $\mathcal{N}_m$, the final trained network $\mathcal{N}$ is represented by the tuple $\mathcal{N} = (\mathcal{N}_m, \mathcal{N}_v)$. If the output dimension of the data is t, then the composite network will have $t+1$ output nodes. The first t output nodes correspond to the output of the mapping network $(\mathcal{N}_m)$ and the $(t+1)^{th}$ node corresponds to the output of the

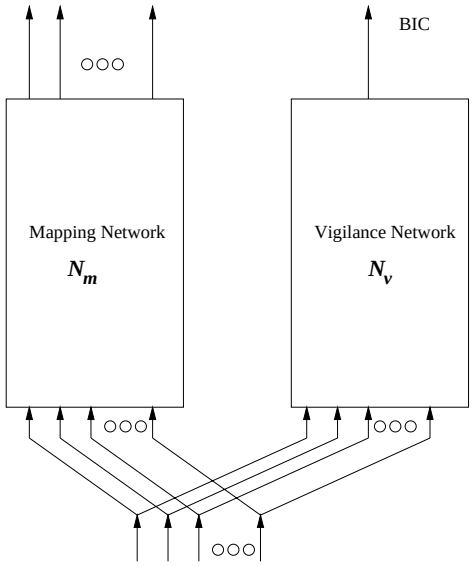


Figure 7.4: The composite network $\mathcal{N} = (\mathcal{N}_m, \mathcal{N}_v)$

vigilance network ($\mathcal{N}_v$). The output of $\mathcal{N}_v$ is called the *boundary indicator component* (BIC)(please refer to Fig. 7.4). A test point is fed to the composite network, and if the BIC gives a value greater than a threshold th, then the output of the test point corresponds to the output of the remaining t nodes. If the BIC bears a value lower than th, the network infers that the point is away from the boundary of the training set and hence the net may not produce a correct output (decision) for it. The algorithm for testing the composite network is given in Table 7.2. The algorithm is valid irrespective of the way the vigilance network is trained, i.e., the vigilance net can be either a BVN or a RVN. The threshold th is generally user defined. And, the threshold for different types of vigilance networks can be different. In our simulations we use $th = 0.5$ if the vigilance network is a BVN. In case the vigilance net is an RVN we use $th = e^{-1}$. In case of RVN, σ_i is the largest distance of a training point that belongs to the cluster associated with the i^{th} receptive field. So, it is reasonable to assume that the receptive field of a node is extended upto a distance equal to its σ or a little beyond that. Based on this idea we choose th equal to the response of a node at a distance σ, which is equal to e^{-1}.

A composite network trained using either kind of vigilance network can be applied for classification problems also. The method in Chapter 6 which was solely meant for classification has some added advantages. As in the method in Chapter 6 different

Table 7.2: Algorithm Test-FA

Algorithm **TEST-FA**
Given a trained network $\mathcal{N} = (\mathcal{N}_m, \mathcal{N}_v)$ and a test point $\boldsymbol{x}$;
Input $\boldsymbol{x}$ to both $\mathcal{N}_m$ and $\mathcal{N}_v$;
Call the output of $\mathcal{N}_v$ as the BIC;
if $(BIC < th)$
    Output "cannot be decided"
else
    Output of $\mathcal{N}_m$ gives the desired output for $\boldsymbol{x}$
endif

subnets are trained for different classes, and each subnet can detect the area in the input space corresponding to a class, hence areas of overlap can be detected using the output of the net. We have already given convincing picture of this fact in Section 6.5, where we used two data sets having significant overlap between classes and found that our network can handle such scenario in a better manner. But, a mapping network and vigilance network pair, when used for classification, will not have such advantages. If a BVN is used for classification, then the algorithm GENERATE (refer to Table 6.4) considers all points in the data set as a single class. The mapping network learns the classification problem as done by a conventional MLP. Hence, the output of a composite network cannot be interpreted to find points which lie in the area of overlapped classes. But, a composite network trained with RVN can be applied in case of high dimensional classification problems.

7.4 Incremental Learning

The methods discussed in the previous section can support incremental learning. Let us consider that the training points come in two phases. In the first phase we have a training sample $T_1 = (X_1, Y_1)$, and we have trained a network say $\mathcal{N}_1 = (\mathcal{N}_{m1}, \mathcal{N}_{v1})$ with T_1. At a later point of time we obtain a new set of data points say $T_2 = (X_2, Y_2)$. The points in T_2, i.e., X_2 may carry new information. If the points in X_2 are away

from the boundary of the points in X_1, then our network $\mathcal{N}_1$ is not expected to decide the output of the points in X_2. In such a case we want to augment $\mathcal{N}_1$ with the new knowledge contained in the training set T_2. Let us denote the BIC of $\mathcal{N}_1$, i.e., the output of $\mathcal{N}_{v1}$ by b_1. If for some points $\boldsymbol{x} \in X_2$, the BIC computed by $\mathcal{N}_1$ is high, we assume that those points are within the boundary of X_1 and we take the output of $\mathcal{N}_{m1}$ as the correct output. These points are not considered to provide new information. For points in X_2 which do not produce a high value of b_1 when tested with $\mathcal{N}_1$ are the points that carry new information and this information needs to be augmented with $\mathcal{N}_1$. Let R_{T_2} denote the set of points in T_2 which does not produce high values of BIC. Now, using R_{T_2} we train a new composite network $\mathcal{N}_2 = (\mathcal{N}_{m2}, \mathcal{N}_{v2})$. We then combine $\mathcal{N}_1$ and $\mathcal{N}_2$ to construct a new network having the knowledge of both training sets T_1 and T_2.

In the preceding discussion, if $\boldsymbol{x}^* \in X_2$ results in a high BIC in $\mathcal{N}_1$, we do not consider the associated input-output pair $(\boldsymbol{x}^*, \boldsymbol{y}^*) \in (X_2, Y_2)$ to provide new information irrespective of the value of $\boldsymbol{y}^*$. In this case if $\boldsymbol{y}^*$ is close to the output produced by $\mathcal{N}_1$ for $\boldsymbol{x}^*$, then it is fine; otherwise, one may argue to consider $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ as new information. We do not recommend that because of the following reasons:

1. When the network $\mathcal{N}_1$ was trained, some input-output pairs $(\boldsymbol{x}_k, \boldsymbol{y}_k)$ were used where $\boldsymbol{x}_k$ was close to $\boldsymbol{x}^*$, but $\boldsymbol{y}_k$ was not close to $\boldsymbol{y}^*$. In such a situation use of $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ is in direct conflict with some training points. So it is better to ignore.

2. There were some small gaps (hole) in the training set and our vigilance network was unable to detect those holes. In this situation, one may consider $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ as new information. But since we are not sure whether it corresponds to such a situation, we do not use $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ as new information.

3. The point $\boldsymbol{x}^*$ falls inside the boundary and the function rapidly changes around it. In this case surely the point $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ carries new information, but again we are not sure whether it corresponds to such a situation. Hence, here too we do not use $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ as new information.

The next issue is then how to aggregate the output of the networks to infer the output for a test point. We propose two schemes for the output aggregation. We call them max aggregation and average aggregation. We discuss these schemes next.

Table 7.3: Algorithm Max Aggregation

Algorithm **Max_Aggregation**
Given, two trained composite networks $\mathcal{N}_1 = (\mathcal{N}_{m1}, \mathcal{N}_{v1})$ and $\mathcal{N}_2 = (\mathcal{N}_{m2}, \mathcal{N}_{v2})$ and a test point $\boldsymbol{x}$;
feed $\boldsymbol{x}$ to both $\mathcal{N}_1$ and $\mathcal{N}_2$;
call the output of $\mathcal{N}_{m1}, \mathcal{N}_{v1}, \mathcal{N}_{m2}, \mathcal{N}_{v2}$ as $\boldsymbol{o}_1, b_1, \boldsymbol{o}_2, b_2$ respectively;
if$(b_1 < th \ and \ b_2 < th)$
    Output "cannot be decided";
elseif $(b_1 < th \ and \ b_2 \geq th)$
    Output $\boldsymbol{o}_2$;
elseif $(b_1 \geq th \ and \ b_2 < th)$
    Output $\boldsymbol{o}_1$;
else *Output* $\frac{1}{2}(\boldsymbol{o}_1 + \boldsymbol{o}_2)$;
endif

7.4.1 Max aggregation

We have two trained composite networks $\mathcal{N}_1 = (\mathcal{N}_{m1}, \mathcal{N}_{v1})$ and $\mathcal{N}_2 = (\mathcal{N}_{m2}, \mathcal{N}_{v2})$. These networks are trained by two different training sets T_1 and R_{T2}. R_{T2} contains only those points of T_2, for which the BIC values produced by $\mathcal{N}_1$ are low. In other words, the points in R_{T2} are not expected to lie within the boundary of T_1. For an arbitrary test point there can be four possibilities:

1. BIC of $\mathcal{N}_1$ is high and that of $\mathcal{N}_2$ is low.
2. BIC of $\mathcal{N}_2$ is high and that of $\mathcal{N}_1$ is low.
3. BIC of both $\mathcal{N}_1$ and $\mathcal{N}_2$ are low.
4. BIC of both $\mathcal{N}_1$ and $\mathcal{N}_2$ are high.

Note that, case 4 will seldom arise for any training point in $T_1 \cup T_2$, but this may arise for a test point, as the boundaries represented by the two networks *may* have

Table 7.4: Algorithm Average Aggregation

Algorithm **Avg_Aggregation**
Given, two trained composite networks $\mathcal{N}_1 = (\mathcal{N}_{m1}, \mathcal{N}_{v1})$ and $\mathcal{N}_2 = (\mathcal{N}_{m2}, \mathcal{N}_{v2})$ and a test point $\boldsymbol{x}$;
select q;

Feed $\boldsymbol{x}$ to both $\mathcal{N}_1$ and $\mathcal{N}_2$;

Call the output of $\mathcal{N}_{m1}, \mathcal{N}_{v1}, \mathcal{N}_{m2}, \mathcal{N}_{v2}$ as $\boldsymbol{o}_1, b_1, \boldsymbol{o}_2, b_2$ respectively;
if$(max(b_1, b_2) < th)$
    Output "cannot be decided";
else
    Output $\frac{b_1^q \boldsymbol{o}_1 + b_2^q \boldsymbol{o}_2}{b_1^q + b_2^q}$;

an overlap. Thus, for an arbitrary test point $\boldsymbol{x}$, we infer the output as the output of $\mathcal{N}_{m1}$ for case 1. Similarly, for case 2, the output of $\mathcal{N}_{m2}$ is taken as the output. For case 3 we infer that the output cannot be decided from the given network and for case 4 the output is computed as the average output of $\mathcal{N}_{m1}$ and $\mathcal{N}_{m2}$. In Table 7.3 we summarize the procedure for generating output from the composite network using the max aggregation.

7.4.2 Average Aggregation

In this case we calculate the aggregated output of the composite networks by an weighted average. Let for a test point $\boldsymbol{x}$, the output of $\mathcal{N}_{m1}$ and $\mathcal{N}_{m2}$ be $\boldsymbol{o}_1$ and $\boldsymbol{o}_2$ respectively and that of $\mathcal{N}_{v1}$ and $\mathcal{N}_{v2}$ be b_1 and b_2 respectively. If $max(b_1, b_2)$ is low, then we conclude that the output cannot be decided. Otherwise, we compute the aggregated final output, $\mathbf{o}$ as

$$\boldsymbol{o} = \frac{b_1^q \boldsymbol{o}_1 + b_2^q \boldsymbol{o}_2}{b_1^q + b_2^q}. \tag{7.5}$$

Here $q \geq 1$ is a constant. Thus the aggregated output is weighted according to the values of the BIC of the two networks. For a test point, if the BIC of a network is low then the output of that network has a negligible effect on the aggregated output. Note that, the choice of q is not very crucial here. $q = 1$ can be used. But for $q > 1$

$$\begin{array}{ll} \frac{b_1^q}{b_1^q + b_2^q} > \frac{b_1}{b_1 + b2} & if \quad b_1 > b_2 \\ \frac{b_1^q}{b_1^q + b_2^q} < \frac{b_1}{b_1 + b2} & if \quad b_1 < b_2. \end{array} \tag{7.6}$$

Thus a bigger value of q will lower the effect of the output of a network with lower BIC and will increase the effect of the output of a network with higher BIC. For our simulations we take $q = 2$. The procedure for average aggregation is summarized in Table 7.4.

7.5 Results

We present here results using various data sets to demonstrate the effectiveness of our network. Through the experiments presented in this section we aim to demonstrate two things:

1. A composite network $\mathcal{N} = (\mathcal{N}_m, \mathcal{N}_v)$ does meaningful generalizations, i.e., it produces good generalization in the vicinity of the training data and it does not respond to test points which come from areas in the input space that are not represented in the training data.

2. Out method has incremental learning ability.

We present the results in three different subsections. In Section 7.5.1 we demonstrate that our network can perform strict generalization using function approximation data sets. In Section 7.5.2 we show that for classification problems also a vigilance network and mapping network pair can be used. Finally, in Section 7.5.3 we show that our network can support incremental learning.

We use four function approximation data sets for demonstrating the effectiveness of our network. The data sets are 3-Peaks, Gabor, Normalized-Chem and Boston-Housing. For two data sets, 3-Peaks and Gabor, we can show the generalization properties pictorially and conclude that our network does a good job. But, for the two real life

data sets, Normalized-Chem and Boston-Housing, such a pictorial representation is not possible as these data sets are in high dimension. For these data sets we define some measures which help us to evaluate the performance of our network. Let $T = (X, Y) = \{(\boldsymbol{x}_i, \boldsymbol{y}_i) : i = 1, 2, ..., N\}$ be the training set and $X_{Te} = \{\boldsymbol{x}'_i : i = 1, 2, .., M\}$ be the input vectors of the test set. A trained composite network $\mathcal{N} = (\mathcal{N}_m, \mathcal{N}_v)$, will either respond to a test point $\boldsymbol{x}'_i$ or will nor respond to it. Thus, the set X_{Te} can be partitioned into two disjoint sets X^A_{Te} and X^R_{Te}. X^A_{Te} contains the points for which the composite network produces a response and X^R_{Te} includes the points for which the composite network does not produce any response. Now, for each test point $\mathbf{x}'_i$ we define a function Δ as:

$$\Delta(\boldsymbol{x}'_i) = \min_{\boldsymbol{x}_j \in X} ||\boldsymbol{x}'_i - \boldsymbol{x}_j||. \tag{7.7}$$

Hence, $\Delta(\boldsymbol{x}'_i)$ represents the distance of $\boldsymbol{x}'_i$ from its nearest neighbor in X. Let $\mu_{\Delta A}$ and $\mu_{\Delta R}$ respectively denote the mean Δ for points which are accepted by the vigilance network (i.e., points in X^A_{Te}) and the points which are rejected by the vigilance network (i.e., points in X^R_{Te}) respectively. Thus,

$$\mu_{\Delta A} = \frac{1}{|X^A_{Te}|} \sum_{\boldsymbol{x}'_i \in X^A_{Te}} \Delta(\boldsymbol{x}'_i), \tag{7.8}$$

and

$$\mu_{\Delta R} = \frac{1}{|X^R_{Te}|} \sum_{\boldsymbol{x}'_i \in X^R_{Te}} \Delta(\boldsymbol{x}'_i). \tag{7.9}$$

For a test set X_{Te} if $\mu_{\Delta A} < \mu_{\Delta R}$ then it is reasonable to assume that the network serves the intended purpose. As, $\mu_{\Delta A} < \mu_{\Delta R}$ implies that on average the distance of the points for which the composite network does not respond are away from the points in the training set.

In all the simulations that we report here we use sigmoidal node functions for the MLPs except for Gabor data, whose outputs lies in [-1 1]. For Gabor we use the *tanh* function in the output nodes of the mapping networks. We train the MLPs using the Lavenberg Marquardt algorithm [69]. Also as stated before, when a BVN is used we take the threshold $th = 0.5$, and in case a RVN is used we use $th = 0.368 \approx e^{-1}$.

7.5.1 Demonstration of Strict Generalization

3-Peaks

The 3-Peaks data set has been discussed in Section 7.2. We sample 80 points uniformly from the interval [0,100]-[40,60] and call them PT_1. We test the generalization capabilities of trained networks on a test set of 1000 equispaced points generated in the interval [0,100].

As PT_1 does not contain any point in the interval [40,60] (refer Fig. 7.2), an ordinary MLP is not expected to produce meaningful response for test points which lie in the interval [40,60]. In Fig. 7.3 we have already shown that this is indeed the case. Fig. 7.3 clearly reveals that an ordinary MLP cannot do meaningful generalizations in the areas of the input space which are not adequately represented by the training data.

A mapping network and vigilance network pair $NP_1 = (NP_{m1}, NP_{v1})$ trained with PT_1 produces better generalizations. Figure 7.5 shows the generalizations for 4 different composite networks trained with PT_1 when vigilance networks used are BVNs. Figure 7.6 shows the generalizations of 4 composite networks which use RVNs. Figures 7.5 and 7.6 reveal that the composite networks do not respond to test points which fall in the area not represented in the training set. Note, the response is plotted only when the BIC bears a value greater than or equal to $th = 0.5$ in case a BVN is used. And in case a RVN is used, the output is plotted only if the BIC ≥ 0.368.

Gabor

In this experiment we train the network with data generated from a 2-D Gabor function. The Gabor function that we use is:

$$h(x_1, x_2) = \frac{1}{2\pi(0.5)^2} e^{-\frac{x_1^2+x_2^2}{2(0.5)^2}} cos(2\pi(x_1 + x_2)). \tag{7.10}$$

A plot of the Gabor function is shown in Fig. 7.7.

We randomly generate 500 input output-pairs such that $(x_1, x_2) \in [-0.5, 0.5]\times[-0.5, 0] \cup [-.5, 0] \times [0, 0.5]$. We call this training set as GT_1. A scatterplot of these points is shown in Fig. 7.8. Additionally we generate 10201 points such that $(x_1, x_2) \in [-0.5, 0.5] \times [-0.5, 0.5]$, which we use as the test set. We call this test set as GTe.

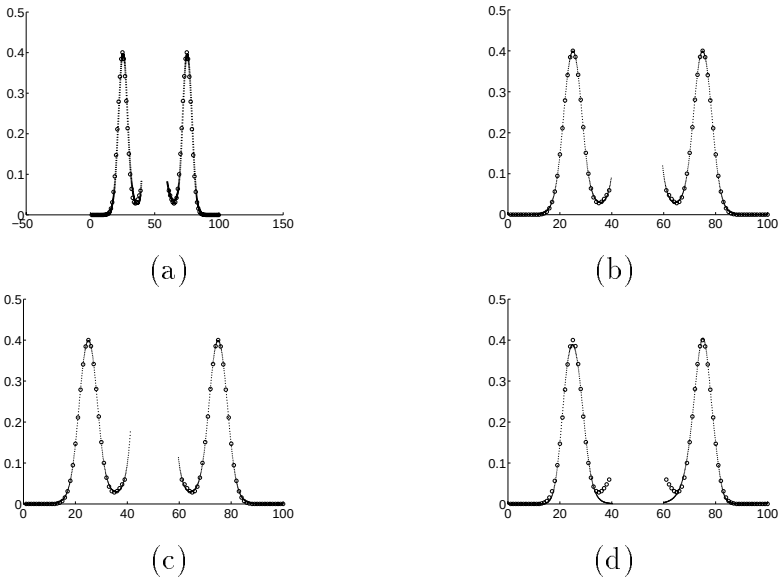


Figure 7.5: Generalizations produced by $NP_1 = (NP_{m1}, NP_{v1})$ (using BVN), when trained with PT_1 for various initializations (the large dots denotes the training points).

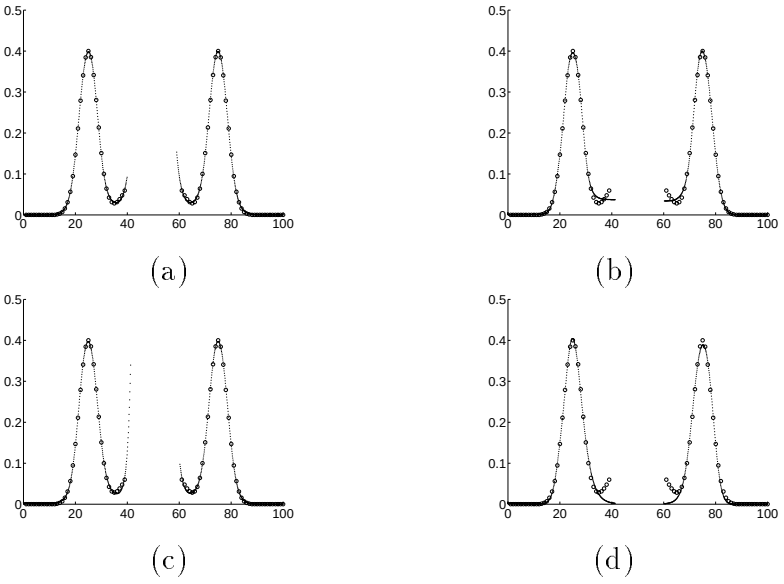


Figure 7.6: Generalizations produced by $NP_1 = (NP_{m1}, NP_{v1})$ (using RVN), when trained with PT_1 for various initializations (the large dots denotes the training points).

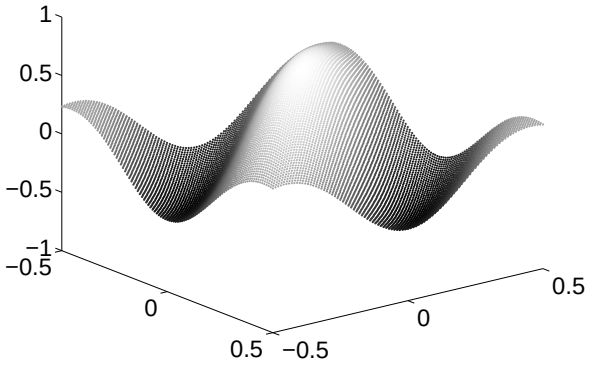


Figure 7.7: Plot of Gabor function

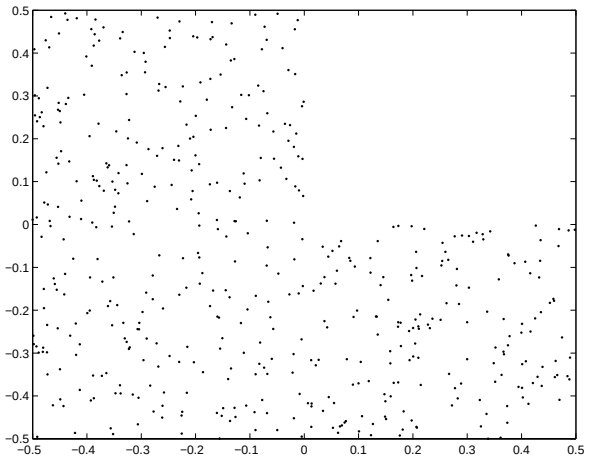


Figure 7.8: Scatterplot of the input vectors in GT_1

An ordinary MLP trained with GT_1, produces erratic generalizations on the unrepresented areas when tested with GTe. The generalization produced by ordinary MLP for 10 different initializations is shown in Fig. 7.9. In all the 10 cases, the right tail of the function is distorted. For a few cases, such as 7.9(b), 7.9(f), 7.9(j), the distortion is severe. With our method a composite network $NG_1 = (NG_{m1}, NG_{v1})$ trained with GT_1 produces better generalizations. Fig. 7.10 shows the generalizations produced by NG_1 when the vigilance networks are BVNs. In each of the cases shown in Fig. 7.10, we use the same architecture and initialization of the mapping network as the MLPs shown in Fig. 7.9. Figure 7.11 shows generalizations produced by 10 different composite networks trained using GT_1, when the vigilance nets are RVNs. Here too the same architecture and initialization are used for the mapping network as in case of the MLPs in Fig. 7.9. Figures 7.10 and 7.11 show that the composite networks refuse to make any decision for inputs coming from the area not supported by the training data.

Comparing Figs. 7.10 and 7.11 we see that, using either of RVN and BVN the intended purpose is served, but BVN gives better generalization. For all cases in Fig. 7.10 the boundaries of the function are smooth which is not the case when a RVN is used (Fig. 7.11). This occurs as the RVN partitions the training data set into hyperspherical clusters, hence, for a two dimensional data set it approximates the boundary with circular arcs. Figures 7.12(a) and (b) displays the functions in Fig. 7.10(a) and 7.11(a) as seen from top. Comparing Fig. 7.12(a) and 7.12(b) we see that in Fig. 7.12(b) the boundary is approximated with circular arcs and in Fig. 7.12(a) we get a more smooth boundary.

Boston-Housing Data and Normalized-Chem Data

Boston-Housing data set [17] contains 506 samples in 13 dimension and it contains only one output. We use a normalized version of this data set. We divide each input feature and the output by the respective maximum value so that they lie between 0 and 1. The Normalized-Chem data is discussed in Section 5.4.1. It has five input features and one output. The total number of samples present is 70.

For Boston-Housing data, we create a random training-test partition so that the the training and test set contains equal number of data points. We train 10 different composite networks with different initializations. For each run we use a mapping

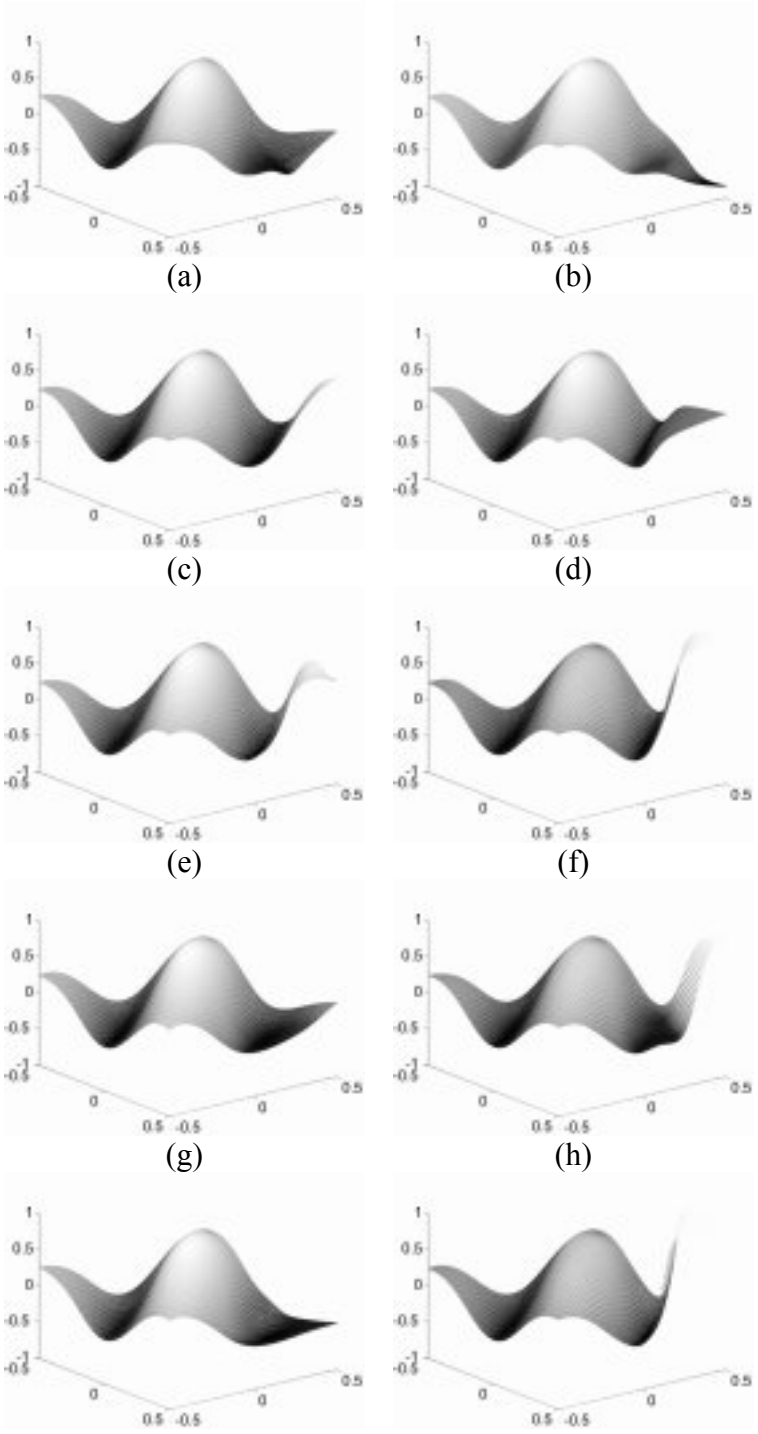


Figure 7.9: Generalization produced by an ordinary MLP when trained with GT_1 with different initializations

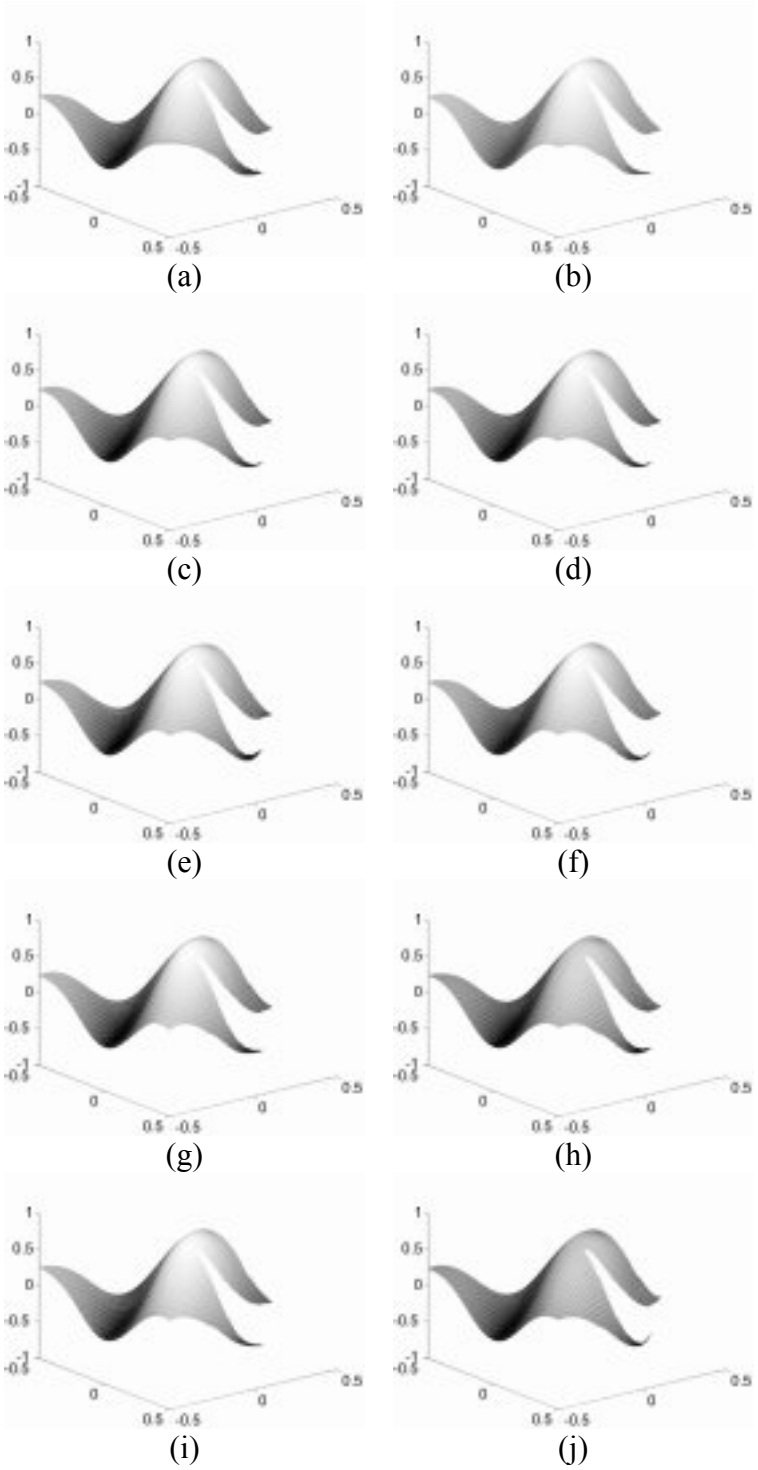


Figure 7.10: Generalization produced by 10 different initializations of NG_1 (using a BVN) when trained with GT_1.

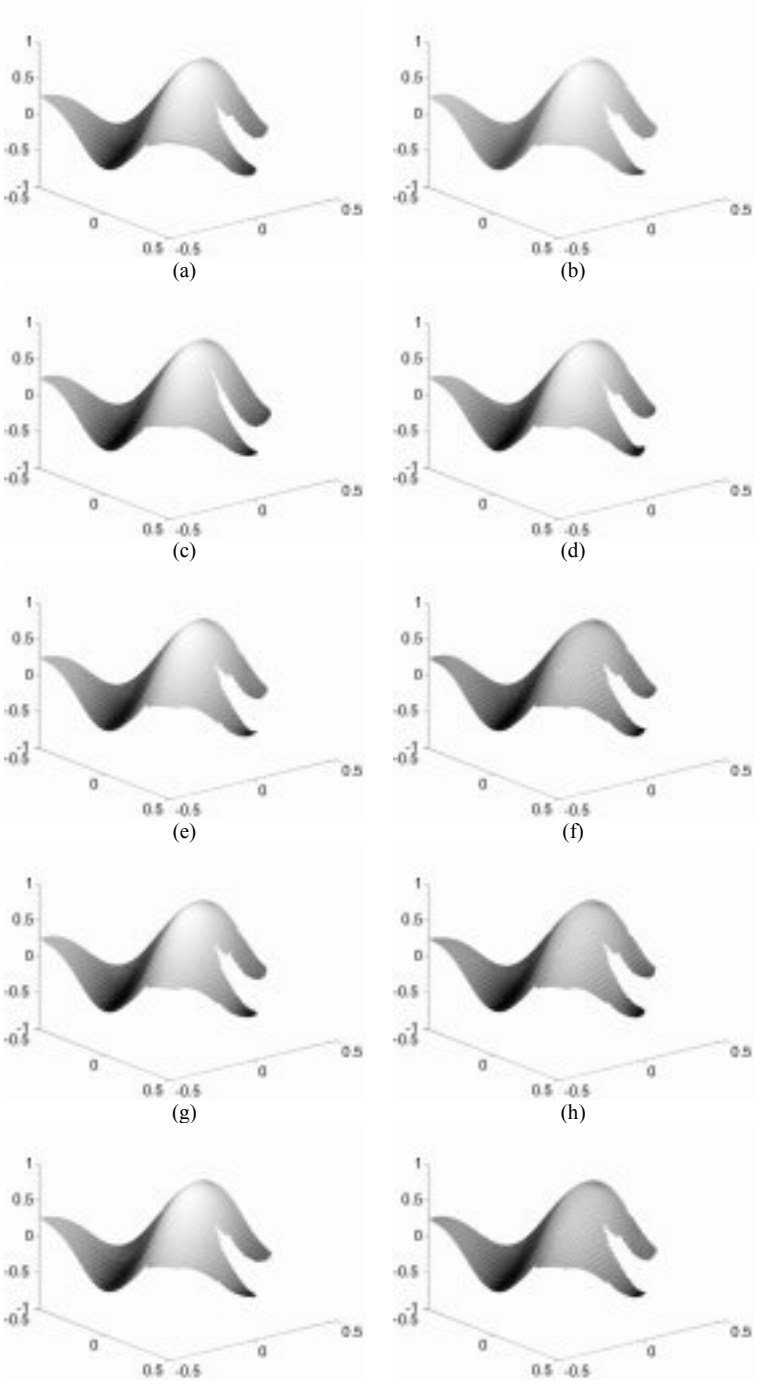


Figure 7.11: Generalization produced by 10 different initializations of NG_1 (using a RVN) when trained with GT_1.

(a) (b)

Figure 7.12: (a) Fig. 7.10(a) as seen from top (b) Fig. 7.11(a) as seen from top

network with 10 hidden nodes and a RVN with 10 receptive fields. Table 7.5 shows the results on the test sets for this data set. In Table 7.5, column 2 shows the number of points for which the composite network makes a decision, while column 3 gives the number of cases for which the network refuses to produce an output. Comparing column 4 with column 5 we see that for all cases $\mu_{\Delta A}$ is significantly lower than $\mu_{\Delta R}$, indicating that the points for which the network made predictions are in the vicinity of the training points. Columns 6 and 7 show the mean test error (the absolute deviation of the network response from the true output) for accepted and the rejected points. Comparing columns 6 and 7, we find that in all cases the network rejects those points which produces more error. Note that, the composite network do not respond to the rejected points (the points in X^R_{Te}), but in column 7 of Table 7.5 we report the deviations of the outputs of the mapping network for the rejected points ignoring the values of the BIC produced by the vigilance network. It is not expected that a trained network will produce good results for test points which are away from the training set, and comparing columns 6 and 7 we see that this is true for all runs with Boston-Housing data.

From Table 7.5 we see that the number of rejected points are very low. Thus, concluding that the average distances of the rejected test points from their nearest neighbors in the training set is always greater than those of the accepted points may not be wise. To validate that in average $\mu_{\Delta A} < \mu_{\Delta R}$, we perform another experiment. In this experiment, we use all 506 points as training examples and test the networks with 1000 additional points generated in the 10% inflated hyperbox containing the training data. Here too we train 10 different networks and test with different test sets each containing 1000 points. Table 7.6 shows the results for the 10 networks. From columns 2 and 3 of Table 7.6 we see that the number of points rejected is much more than

Table 7.5: Run statistics for Boston-Housing on 50% training-test partition

Run No.	$\lvert X^A_{Te}\rvert$	$\lvert X^R_{Te}\rvert$	$\mu_{\Delta A}$	$\mu_{\Delta R}$	Mean Test error for accepted points	Mean test error for rejected points
1	245	8	0.155	0.324	0.090	0.149
2	246	7	0.152	0.369	0.092	0.212
3	249	4	0.158	0.395	0.068	0.386
4	245	8	0.153	0.439	0.078	0.187
5	251	2	0.152	0.280	0.068	0.304
6	246	7	0.157	0.330	0.115	0.140
7	243	10	0.160	0.317	0.098	0.397
8	246	7	0.158	0.454	0.445	0.602
9	247	6	0.161	0.425	0.438	0.585
10	248	5	0.158	0.246	0.153	0.219

Table 7.6: Run statistics for Boston-Housing on artificially generated test data

Run No.	$\lvert X^A_{Te}\rvert$	$\lvert X^R_{Te}\rvert$	$\mu_{\Delta A}$	$\mu_{\Delta R}$
1	86	914	0.810	1.242
2	155	845	0.862	1.243
3	103	897	0.804	1.242
4	135	865	0.864	1.230
5	108	892	0.818	1.231
6	88	912	0.799	1.233
7	56	944	0.812	1.218
8	143	857	0.873	1.235
9	118	882	0.796	1.241
10	139	861	0.851	1.233

Table 7.7: Run statistics for Normalized-Chem on artificially generated test data

Run No.	$\lvert X^A_{Te}\rvert$	$\lvert X^R_{Te}\rvert$	$\mu_{\Delta A}$	$\mu_{\Delta R}$
1	258	742	0.238	0.485
2	249	751	0.250	0.464
3	166	834	0.224	0.467
4	159	841	0.237	0.449
5	208	792	0.232	0.468
6	252	748	0.245	0.492
7	152	848	0.225	0.455
8	150	850	0.229	0.458
9	199	801	0.231	0.464
10	177	823	0.235	0.458

the number of points accepted by the composite network. This is due to the fact that the input vectors are 13 dimensional, and we have only 506 training points. So, the training points occupy only a small part of the total hyperbox bounded by the data. And most of the artificially generated points fall outside the boundary of the training sample. The scenario was different in case of Table 7.5 as there it is expected that the test points follow the same probability distribution as that of the training points, and hence in Table 7.5 only a few points got rejected. Comparing column 4 and 5 of Table 7.6 we see that for all cases $\mu_{\Delta A} < \mu_{\Delta R}$, which shows that the networks respond only to points which are in the vicinity of the training points. As in this case the test data are artificially generated, we cannot measure the deviation of the network output from the true output.

Normalized-Chem contains only 70 points. So, we do not report the results on a training-test partition for this data but use artificial points to test the network. Here too we train 10 different composite networks, where the mapping networks use 10 hidden nodes and the RVN has 10 receptive fields. Each network is tested with a different test set containing 1000 points generated within the 10% inflated bounding box containing the training data. Table 7.7 shows the performance of the 10 networks. Comparing columns 4 and 5 of Table 7.7 we find that here too $\mu_{\Delta A} < \mu_{\Delta R}$ for all cases.

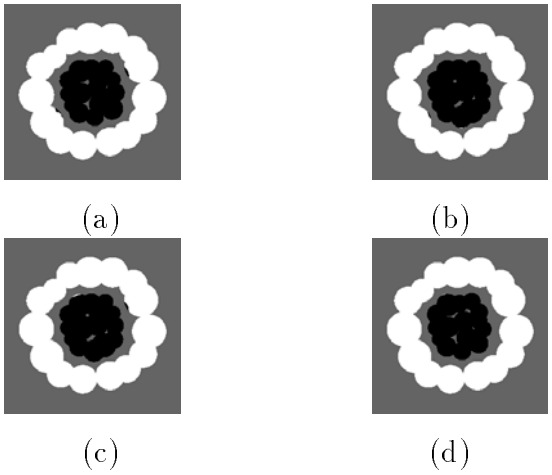


Figure 7.13: Generalizations on Dish-Shell using RVN (the four results are results obtained by 4 networks with different initializations)

7.5.2 Results on Classification

We report results on three classification data sets using a RVN and mapping network pair. The data sets we use are Dish-Shell (refer Section 6.3), Breast Cancer (refer Section 5.4) and Wine (refer Section 5.4).

The Dish-Shell consists of data points in $\Re^2$ divided into two classes. The scatter plot of Dish-Shell is shown in Fig. 6.9(a). The generalizations produced by 4 different mapping network-RVN pairs are shown in Fig. 7.13. The four networks whose outputs are shown in Fig. 7.13 use RVNs with 30 receptive fields and a mapping network with 10 hidden nodes. Here we have used an artificially generated array of $256 \times 256 (= 65536)$ points uniformly covering the entire (inflated) rectangle bounding the data as the test set (the same protocol was used in Section 6.3). In Fig. 7.13 black and white denote the two classes and gray denotes the points for which the composite network did not produce any response. The results in Fig. 7.13 clearly shows that the networks do not respond to the test points which are away from the training points. But, we notice that there are a few points within the dish (the black class) for which the network does not respond. This is probably due to the fact that those areas were not well represented by training data.

The performance of the composite network using RVN is also satisfactory for the 13 dimensional Wine data and 9 dimensional Breast-Cancer data. Tables 7.8 and 7.9

Table 7.8: Run statistics for Wine

Run No.	$\lvert X^A_{Te}\rvert$	$\lvert X^R_{Te}\rvert$	$\mu_{\Delta A}$	$\mu_{\Delta R}$
1	86	3	0.390	0.560
2	87	2	0.388	0.733
3	87	2	0.387	0.814
4	87	2	0.398	0.618
5	84	5	0.377	0.606
6	87	2	0.406	0.767
7	88	1	0.391	0.912
8	88	1	0.402	0.781
9	82	7	0.394	0.723
10	88	1	0.399	0.912

show the run statistics of 10 networks trained and tested for Wine and Breast-Cancer data respectively. For both these data sets we used equal number of points in the training and test sets. Also we used 10 hidden nodes in the mapping network and 10 receptive fields in the RVN. Tables 7.8 and 7.9 clearly show that for all networks $\mu_{\Delta R}$ is significantly greater than $\mu_{\Delta A}$, which means that the network refuses to respond to data points which are far from the boundary of the training sample.

7.5.3 Demonstration of Incremental Learning

Now we show the incremental learning capabilities of our network using the 3-Peaks and Gabor data sets. Here we assume that the data come in two phases. For 3-Peaks the training data set PT_1, generated in the interval [10,100]-[40,60], comes in phase 1. Also we sample 20 more points from [40,60], and call them PT_2. PT_2 forms the training data set for the second phase. To test the incremental learning capability we train two networks $NP_1 = (NP_{m1}, NP_{v1})$ and $NP_2 = (NP_{m2}, NP_{v2})$ using PT_1 and PT_2 respectively. The generalization performance of NP_1 and NP_2 is shown in Figs. 7.14(a) and (b) respectively. Figures 7.14(c) and (d) show the generalization by aggregating the outputs of NP_1 and NP_2 by max aggregation and average aggregation respectively. We find that for this data set both aggregation operators produce good

Table 7.9: Run statistics for Breast-Cancer

Run No.	$\lvert X^A_{Te}\rvert$	$\lvert X^R_{Te}\rvert$	$\mu_{\Delta A}$	$\mu_{\Delta R}$
1	341	1	2.160	8.307
2	341	1	1.999	5.916
3	341	1	2.054	9.165
4	340	2	2.016	7.083
5	337	5	2.081	8.517
6	334	8	2.016	7.058
7	338	4	1.923	7.854
8	341	1	2.131	5.916
9	341	1	2.083	9.165
10	338	4	2.119	6.730

generalization; perhaps the average aggregation operator generates a little more smooth output. Here we use BVN as the vigilance network. Figure 7.15 shows the same results when a RVN is used. In this case, clearly the average aggregation operator results in better generalization.

For Gabor data set we take $GT1_1$ that is used in Section 7.5.1 as the data set in phase 1. Also we generate another set of 200 input-output pairs such that $(x_1, x_2) \in [0, 0.5] \times [0, 0.5]$. We call this set as GT_2, the training set for the second phase. With GT_1 and GT_2, we train two networks $NG_1 = (NG_{m1}, NG_{v1})$ and $NG_2 = (NG_{m2}, NG_{v2})$ respectively. We use BVN as the vigilance network. The generalization produced by NG_1 and NG_2 on the test data is shown in Figs. 7.16 (a) and (b). Figures 7.16 (c) and (d) display the aggregated output of the two networks by Max aggregation and Average aggregation, on the same test set GTe. Both aggregation schemes produce good generalizations. Figure 7.17 shows the results when RVN is used as the vigilance network. Figures 7.16 and 7.17 demonstrate that in both cases the networks are capable of incremental learning.

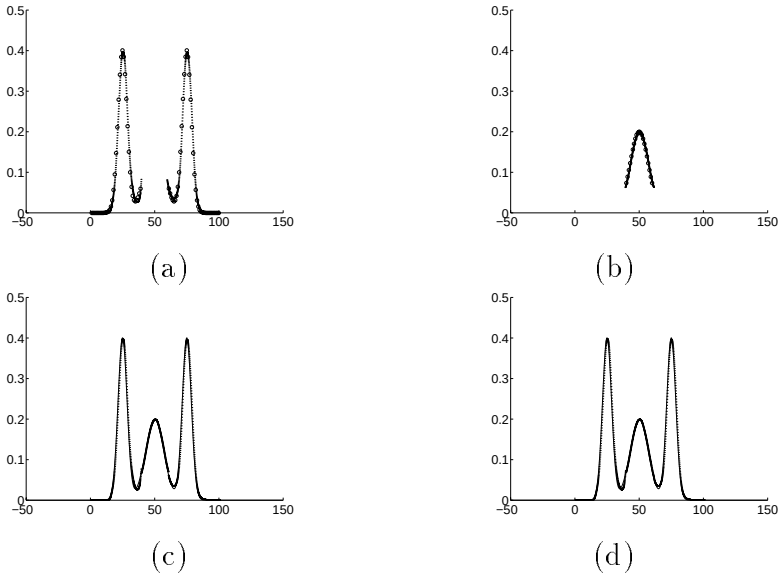


Figure 7.14: Results on 3-Peaks by using BVN: (a) Generalization of NP_1 (b) Generalization of NP_2 (c) Generalization of the aggregated networks NP_1 and NP_2 when aggregated by Max aggregation (d) Generalization of the aggregated networks NP_1 and NP_2 when aggregated by Average aggregation

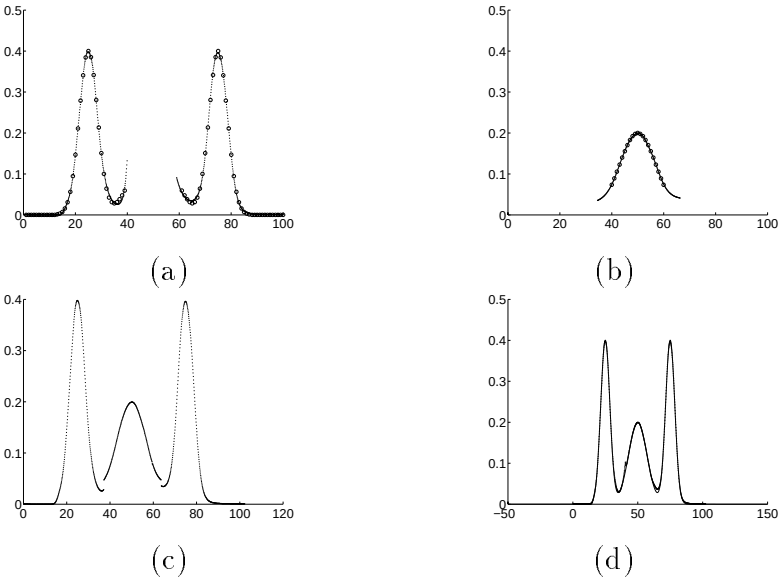


Figure 7.15: Results on 3-Peaks by using RVN : (a) Generalization of NP_1 (b) Generalization of NP_2 (c) Generalization of the aggregated networks NP_1 and NP_2 when aggregated by Max aggregation (d) Generalization of the aggregated networks NP_1 and NP_2 when aggregated by Average aggregation

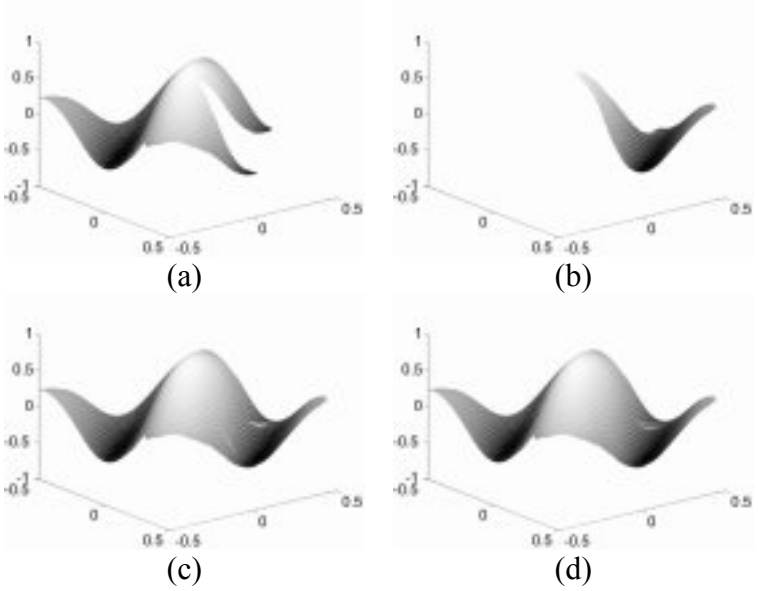


Figure 7.16: Results on Gabor by using BVN (a) Generalization of NG_1 (b) Generalization of NG_2 (c) Generalization of the aggregated networks NG_1 and NG_2 when aggregated by Max aggregation (d) Generalization of the aggregated networks NG_1 and NG_2 when aggregated by Average aggregation

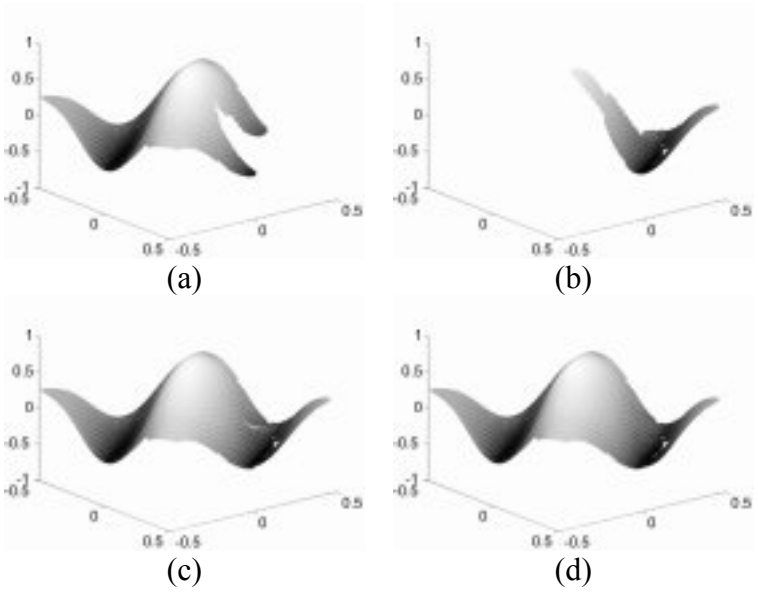


Figure 7.17: Results on Gabor by using RVN (a) Generalization of NG_1 (b) Generalization of NG_2 (c) Generalization of the aggregated networks NG_1 and NG_2 when aggregated by Max aggregation (d) Generalization of the aggregated networks NG_1 and NG_2 when aggregated by Average aggregation

7.6 Conclusions and Discussion

In this chapter we extended our methodology in Chapter 6 for function approximation problems. Our method is based on training two nets performing entirely different tasks. One network (the vigilance network) decides whether a test point lies within the boundary of the training sample and the other (the mapping network) learns the input-output mapping. Merging the output of these two networks we decide the output for a test point.

The crux of the method lies in the scheme to train the vigilance network. Training of the vigilance network has been considered by two different approaches. The first approach depends on an algorithm to generate points outside the boundary of the training set. This training scheme becomes computationally very expensive for high dimensional data sets. Hence, we have proposed another method to train the vigilance network, which does not need any additional points for training but it produces almost the same results. This new method of training the vigilance network can be used for classification problems also. For classification problems, we recommend use of RVN only for high dimensional data because our use of RVN for classification has certain limitations over the method discussed in Chapter 6. The network developed in Chapter 6 has capability to detect data points which lie in the area of overlap of classes and hence, gives us more information. But the mapping network and vigilance network pair discussed in this chapter cannot do so. Further investigations are needed to make it more useful for classification problems.

Chapter 8

Enhancing the Generalization Ability of Multilayer Perceptron Networks[1]

8.1 Introduction

In Chapters 6 and 7 we addressed one facet of generalization in MLPs. We showed through experiments that a conventional MLP may not produce desirable results for test points which lie far from the boundary of the training set. We also proposed some new techniques to train MLPs which make them perform *strict generalization*, i.e., these networks do not produce any response for points which are atypical to its training points. In this chapter we shall deal with the problem of overfitting in MLPs. The method that we discuss here has particular importance in situations where we have insufficient training data.

Given a training set $T = \{(\boldsymbol{x}_i, \boldsymbol{y}_i), i = 1, ..., N\}$, where $\boldsymbol{x}$ is the input and $\boldsymbol{y}$ is its output, there are plenty of procedures available in the literature which use T to form a predictor function $\phi(\boldsymbol{W}, \boldsymbol{x})$. $\boldsymbol{W}$ is a parameter vector which is decided using T. A very popular procedure of obtaining the predictor ϕ is by training a feed-forward neural network (like a multilayer perceptron (MLP)) with T. In this case the parameter vector $\boldsymbol{W}$ becomes the parameters (weights and biases) of the neural network, which are learned with the aid of the training set T. An optimization procedure selects a $\boldsymbol{W}$ minimizing the error on the training set. The operational performance measure for the trained network is the error on future data outside the training set, also known as the generalization error. Practice has shown that a direct minimization of the training error for a given

[1]Some materials of this chapter have been published in [30].

fixed training set by backpropagation or similar type of training algorithm does not necessarily imply a minimization of the generalization error. A common means to bypass this difficulty is to use a validation set to judge the generalization ability and decide on the architecture of the network or the stopping time for training. To improve generalization researchers have followed many different approaches including pruning [196], weight sharing [40], and complexity regularization [189, 190] (refer to the short survey in Section 2.2).

Generally, an MLP is trained using a finite training sample. The available training data are reused in every epoch and as a result, the neural network "concentrates" more and more on these points and this often results in a bad generalization. A probable solution to this problem would be to have an infinitely large training set which is seldom realizable in practice. Here we propose a method to expand the training set to any required size, for better generalization. A related concept, suggested in [82], adds noise to the training set. In [82] the authors considered adding white noise independently to the input and output vectors to generate new training samples. In [100], Karistinos and Pados gave an algorithmic procedure for random expansion of a given training set. They proposed a locally most entropic estimate of the true joint input-output probability density function of the training sample to generate new training samples. They argued that the method in [82] is an extreme special case of that in [100]. Here we present a scheme to generate new input vectors that are distributed in the same fashion as the training samples. We use a k-nearest neighbor heuristic to generate the outputs of the generated input vectors. Our experiments demonstrate that our method can produce nice generalizations. Our method is different from that in [82, 100] as it does not require an explicit density estimation, which is known to be unstable [226].

8.2 Expanding the Training Set: The Function Approximation Case

It is assumed that the training data are obtained from an unknown time invariant probability distribution. Thus, expansion of the training data can be best done if we attempt to learn the unknown probability distribution from which the data were generated and generate additional samples from the obtained distribution. Estimating probability distributions from data is an ill posed problem whose solutions are usually

unstable and tends to become inaccurate with the increase in dimensionality of the data [226, 237]. Here we do not attempt to learn the probability distribution of the data but we exploit its spatial distribution to generate additional data points which maintains the spatial distribution of the given training set.

First we consider FA type problems. Let us denote the input vector $\boldsymbol{x}_i$ augmented with its output y_i (which may be single valued or a vector, without loss of generality we consider a single valued output) by $\tilde{\boldsymbol{x}}_i$. We also assume that y is continuous. To each data point $(\boldsymbol{x}_i, y_i)$ in T, we assign a probability p_i which is a function of the density of input data points in the neighborhood of $\boldsymbol{x}_i$. We model p_i using the mountain potential which has been used successfully for numerous clustering applications [165, 241]. Thus

$$p_i = \frac{1}{Z} \sum_{j=1}^{N} exp(-||\boldsymbol{x}_i - \boldsymbol{x}_j||), \tag{8.1}$$

where Z is a normalizing constant chosen to make $\sum_{i=1}^{N} p_i = 1$. Our algorithm samples a point $\boldsymbol{x}_i$ from T with probability p_i. Then it finds the k nearest neighbors of $\tilde{\boldsymbol{x}}_i$, we call them $\tilde{\boldsymbol{x}}_i^1, \tilde{\boldsymbol{x}}_i^2, ..., \tilde{\boldsymbol{x}}_i^k$. While computing the nearest neighbors we consider the point $\tilde{\boldsymbol{x}}_i$ also, i.e., $\tilde{\boldsymbol{x}}_i^1 = \tilde{\boldsymbol{x}}_i$. A new point $\tilde{\boldsymbol{x}}_{new}$ is generated as a convex combination of the points $\tilde{\boldsymbol{x}}_i^1, \tilde{\boldsymbol{x}}_i^2, ..., \tilde{\boldsymbol{x}}_i^k$, i.e,

$$\tilde{\boldsymbol{x}}_{new} = \sum_{j=1}^{k} \lambda_j \tilde{\boldsymbol{x}}_i^j, \tag{8.2}$$

where

$$\sum_{j=1}^{k} \lambda_j = 1, \qquad 0 \leq \lambda_j \leq 1.$$

The λ_j's are randomly generated. Note that, $\tilde{\boldsymbol{x}}$ represents a new input-output pair but the probability distribution assumed for sampling, depends on the input vectors only. In each step, our algorithm samples a point according to the density, and generates a point in its neighborhood. So, more points will be generated in dense regions and less points in the sparse regions. This algorithm has a single user defined parameter k which denotes the number of nearest neighbors considered.

An MLP is trained with the original training set T in the first epoch and in the subsequent epochs it faces points generated by the above described algorithm. Thus, in each iteration, the MLP faces a new set of points.

8.3 Expanding the Training Set: The Classification Case

The method of generation of points described in the previous section can be applied for classification problems also with minor modifications. In case of classification, the output vectors are class labels. If we have at hand a t class problem, we encode the class labels by t dimensional hard label vectors in N_{ht}(refer eq. (4.3)). In other words, if a feature vector belongs to class i, then the i^{th} component of the output vector is 1 and the other $t-1$ components are all zero. Thus, an input-output pair can be represented as $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, $\boldsymbol{x}_i \in \Re^s$ and $\boldsymbol{y}_i \in N_{ht}$.

We follow the same strategy as in Section 8.2, and assign probabilities to each data point in the training set based on the their spatial positions (refer eq. (8.1)). Next we sample a point $\boldsymbol{x}_i$ with probability p_i. We find the k nearest neighbors of $\boldsymbol{x}_i$ and denote them by $\boldsymbol{x}_i^1, \boldsymbol{x}_i^2, \ldots, \boldsymbol{x}_i^k$. Note that, unlike the FA case, here we are not using the joint input-output vectors to find the nearest neighbors, but using the input vectors only. A new input vector is generated by a convex combination of $\boldsymbol{x}_i^1, \boldsymbol{x}_i^2, \ldots, \boldsymbol{x}_i^k$ and its class label as a convex combination of $\boldsymbol{y}_i^1, \boldsymbol{y}_i^2, \ldots, \boldsymbol{y}_i^k$. Where $\boldsymbol{y}_i^j$ is the class label corresponding to $\boldsymbol{x}_i^j$. Thus,

$$\boldsymbol{x}_{new} = \sum_{j=1}^{k} \lambda_j \boldsymbol{x}_i^j, \tag{8.3}$$

$$\boldsymbol{y}_{new} = \sum_{j=1}^{k} \lambda_j \boldsymbol{y}_i^j, \tag{8.4}$$

λ_js are randomly generated. The same set of λ_js are used in eq. (8.3) and eq. (8.4). And,

$$\sum_{j=1}^{k} \lambda_j = 1, \qquad 0 \leq \lambda_j \leq 1.$$

The labels generated for the new points are convex combination of hard labels, hence they are fuzzy labels.

The input vector of the new generated point is a combination of k original points. The contribution of a particular point in generating the new point is decided by the weight λ_j. If a point $\boldsymbol{x}_i^j$ has a greater λ_j associated with it then it contributes more to the new point. The labels of the individual points are also weighted by the same factor to generate the label for the new point. Thus, if a vector $\boldsymbol{x}_i^j$ has more share in the new point then the label vector of the new point will also be more close to the label

of $\boldsymbol{x}_i^j$. If all k nearest neighbors of a sampled point have the same label then the label generated for the new point will be the same as that of the k neighbors. Otherwise, the label vector will be a fuzzy label vector with appropriate weights.

8.4 Results

Here we present results on one function approximation and three classification problems. The data sets we use for simulation are Sine, Iris, Breast-Cancer and Bupa.

We compare our results with the methods in [82, 100]. The method in [100] assumes a mixture of Gaussians model to estimate the joint probability distribution of the input-output samples in the training set. To determine the number of modes in the distribution, they apply a clustering algorithm on the data. The number of clusters is chosen so that the Gaussian mixture exhibits the minimum differential entropy. For high dimensional data the method involves computation of multiple integrals using the Monte-Carlo method, which is computationally expensive. Also, their method cannot be applied directly to classification problems. So, we compare our method with that of [100] only for the function approximation problem.

The method in [82] finds a kernel density estimate for the input-output pairs in case the outputs are continuous and for classification problems they compute the class conditional kernel density estimates. The authors in [82] provided certain guidelines to choose the bandwidth of the Gaussian kernels including a method of cross validation which was first proposed by Duin [51]. For implementing the method in [82] we use the method of cross validation [51] to decide the bandwidth of the kernels. We compare our method with the method in [82] for both function approximation and classification problems.

In all the simulations, MLP networks with sigmoidal activation functions are used and the networks are trained using the Lavenberg Marquardt algorithm [69]. In the following subsections we present the results.

8.4.1 Results on Function Approximation

Sine data

We consider the problem of learning a noisy sine curve [100]

$$f(x) = 0.4 \sin\ x + 0.5, \quad x \in \Re. \tag{8.5}$$

As in [100], we generate 36 input-output pairs $\tilde{\boldsymbol{x}}_i = (x_i, y_i)$, $i = 1, .., 36$, with x_i generated uniformly on $[-\pi, \pi]$ and $y_i = f(x_i) + r_i$, where $r_i \sim \mathcal{N}(0, \sigma^2)$ accounts for the noise with variance $\sigma^2 = 0.01$.

With this data sets we trained conventional MLPs, where the same set of 36 points were repeated in each epoch. Also we trained MLPs using points generated according to the methods in [100], [82] and by our method. In each run we fix an initialization of an MLP and train networks with all four methods. We make 10 such runs and name them as (a)-(j). Hence, in total we train 40 networks. Each of these 40 networks has an architecture of 1:10:1. For implementing the method in [100] we consider the data as a mixture of 11 Gaussians (the same was considered in [100]). For the method in [82] we use a bandwidth of $h = 0.1509$ for the Gaussian kernels. This value of h is obtained by a method of cross validation [51]. For generating points using our method we used $k = 5$.

Table 8.1 gives the sum of square error (SSE) of the 40 networks. Each row of Table 8.1 represents performance of the four methods of training keeping the initialization fixed. The last two rows of Table 8.1 give the mean and standard deviation of the performance of the 10 networks trained by a particular method. The generalizations produced by the 40 networks whose performance is depicted in Table 8.1 are pictorially shown in Figs. 8.1-8.4.

Figure 8.1 reveals that for all the initializations, an MLP trained by the conventional method yields pretty bad generalization. The generalization is so bad because we have chosen 10 hidden nodes, which is pretty high, given the size of the data set. Hence, in almost all cases of Fig. 8.1 we see that the network has tried to memorize the training data points. The poor performance in all cases is also revealed by the high SSE values as depicted in column 2 of Table 8.1. Figure 8.2 shows a better scenario, as we see that in this case the generalizations produced are consistent and more smooth than

Table 8.1: Results on Sine Data: The sum of square error on test set for networks trained with different methods for 10 different initializations

run No.	Ord. MLP.	Method in [100]	Method in [82]	our method
(a)	162.31	37.40	23.23	20.39
(b)	77.78	62.46	103.29	41.51
(c)	116.70	23.31	86.18	20.39
(d)	88.58	32.28	115.00	27.12
(e)	148.62	31.52	124.68	21.69
(f)	106.55	38.12	65.12	16.07
(g)	154.59	25.83	115.94	13.64
(h)	56.59	39.76	62.36	20.77
(i)	178.52	44.79	97.19	42.87
(j)	69.44	25.75	16.90	17.86
Mean	**115.96**	**36.12**	**80.98**	**24.23**
Std. Dev.	**42.95**	**11.55**	**38.17**	**10.11**

those shown in Fig. 8.1. The generalizations produced by the method in [82] are shown in Fig. 8.3. Figure 8.3 reveals that the generalizations produced by the networks are better than those in Fig. 8.1 but are worse than those in Fig. 8.2. Figure 8.4 shows the generalizations produced by networks trained using our method of generating points. All the 10 networks in Fig. 8.4 produces consistent generalizations and they are better than those in Figs. 8.1-8.3.

From Table 8.1 we see that both the mean and standard deviation of the 10 networks trained by our method are much less compared to other methods. From Table 8.1 we can conclude that for Sine data our method is much better than a conventional MLP and the method in [82], and it is marginally better than the method in [100].

8.4.2 Results on Classification

We test the proposed method on three classification data sets, namely, Iris, Breast-Cancer and Bupa. Iris data set is discussed in Section 4.6.1 and the Breast-Cancer data set in Section 5.4. The Bupa data [17] contains 345 data points in six dimension. The data points are divided into two classes.

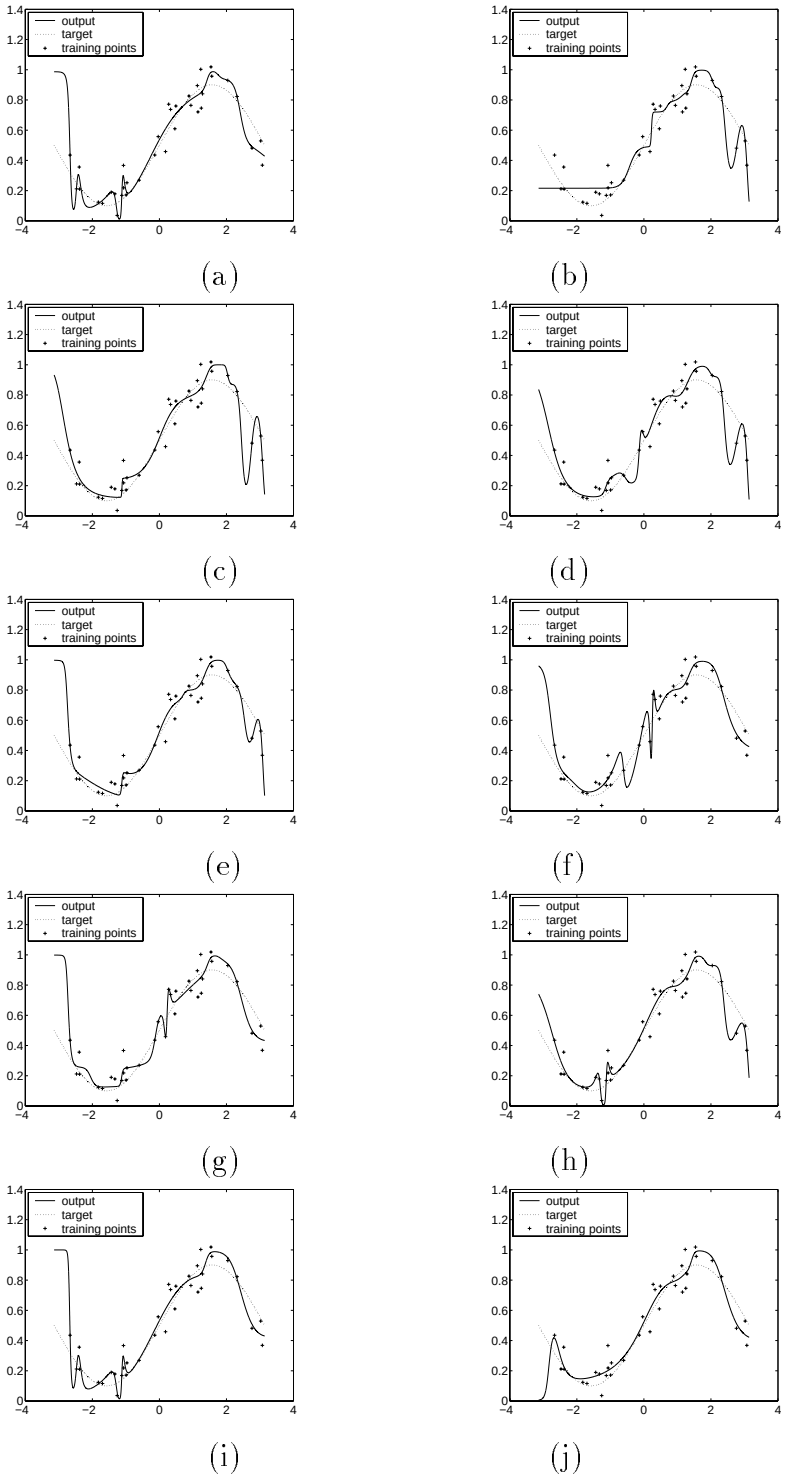


Figure 8.1: Generalizations on Sine data by MLP trained by conventional method

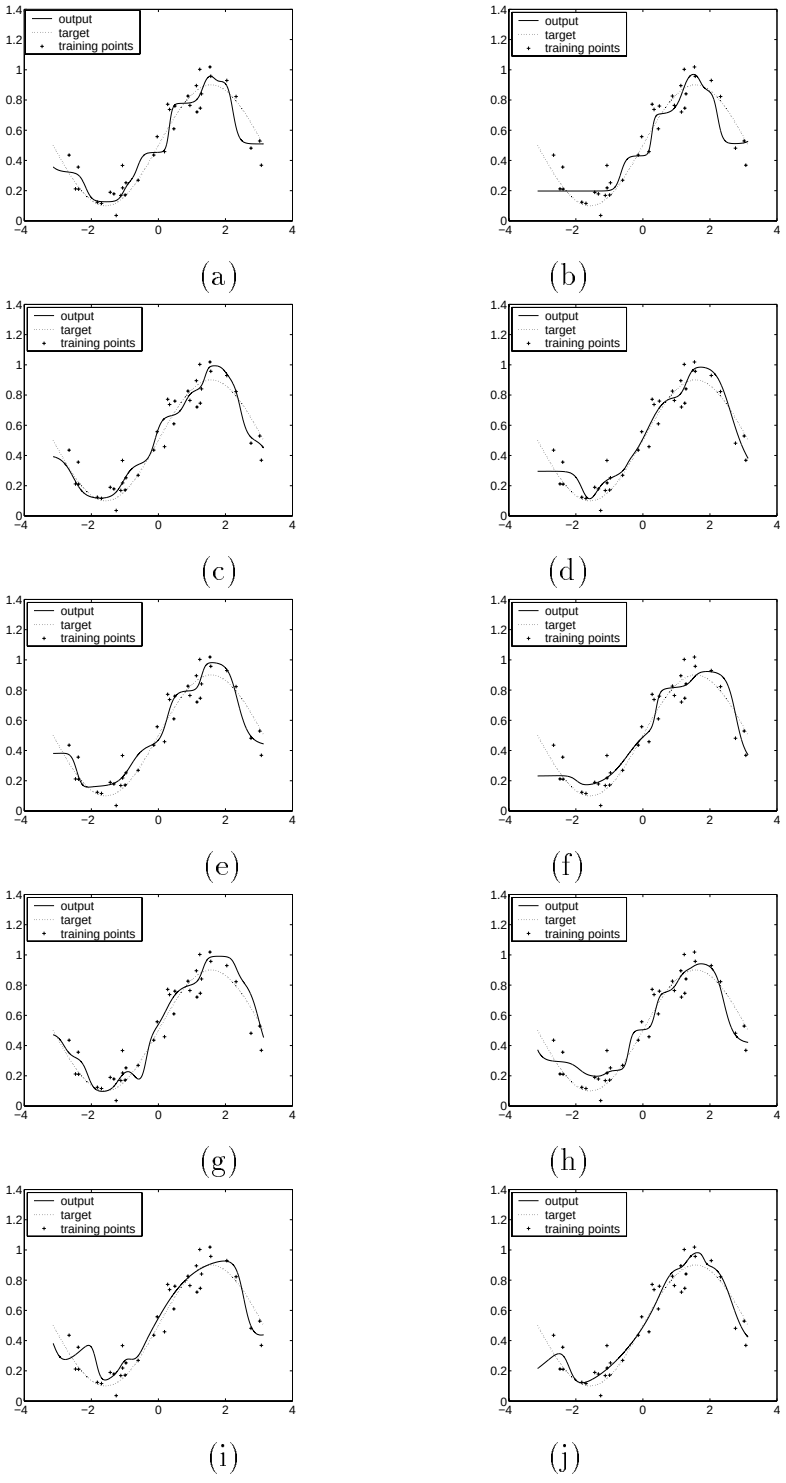


Figure 8.2: Generalizations on Sine data by MLP trained by points generated by the method in [100]

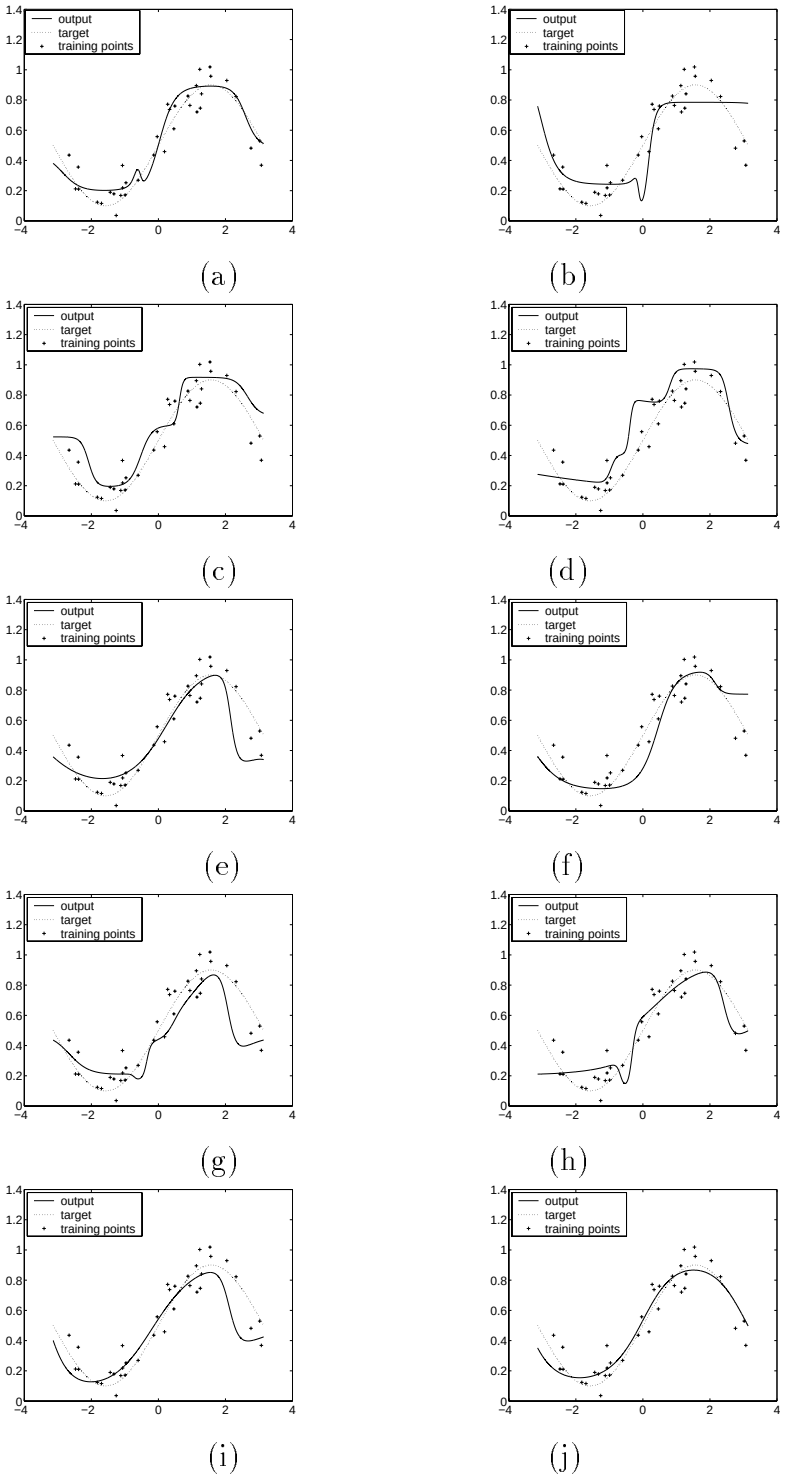


Figure 8.3: Generalizations on Sine data by MLP trained by points generated by the method in [82]

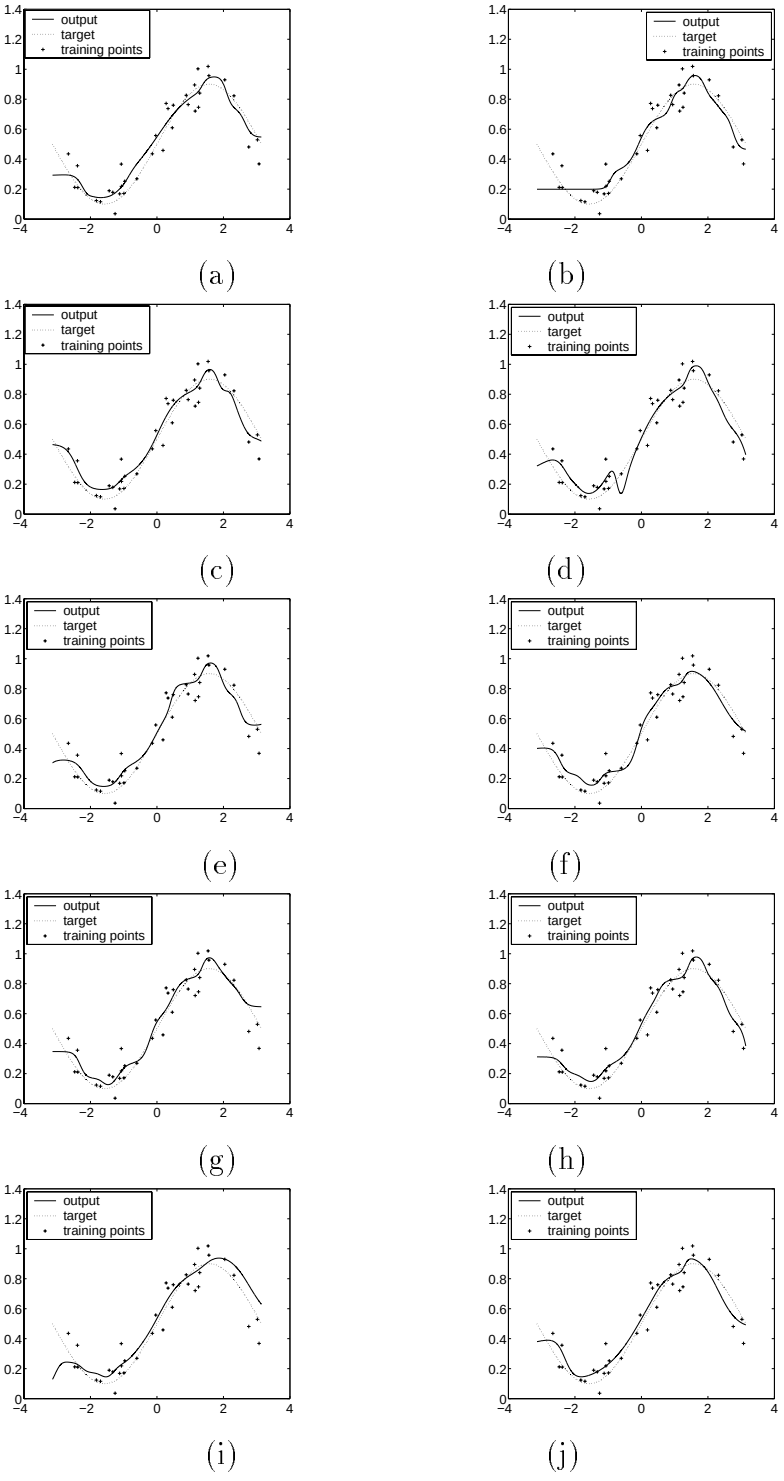


Figure 8.4: Generalizations on Sine data by MLP trained by points generated by our proposed method

Table 8.2: Bandwidths used for the classification data sets for implementing the method in [82]

Data Set	Class	Bandwidth
Iris	Class 1	0.220
	Class 2	0.170
	Class 3	0.160
Breast-Cancer	Class 1	0.510
	Class 2	1.590
Bupa	Class 1	0.069
	Class 2	0.062

We did a 10 fold cross validation for all data sets. In 10 fold cross validation the data set is randomly divided into 10 equal sized subsets. In each step one subset is left out and a network is trained with the data points in the remaining 9 subsets. The trained network is then tested with the data points in the left out subset. This can be carried out 10 times, and each time the network gets trained and tested with different data points.

In our simulations, we train MLPs using the conventional method, the method in [82] and our proposed method. Using our method we train MLPs with $k = 3, 5, 7$ and 10. For each of the methods, for each training-test partition we train 10 networks with different initializations. Hence by each method we train a total of 100 MLPs. For each of the MLPs we use 10 nodes in the hidden layer. The kernel bandwidths used for different data sets for different classes that we used for implementing the method in [82] are shown in Table 8.2. We obtained these bandwidths using the method of cross-validation [51].

Table 8.3 shows the mean misclassification of the 100 trained networks on the test data sets. The second and third rows of Table 8.3 show the mean misclassifications obtained by conventional MLP and by the method in [82] respectively. The remaining four rows show the mean misclassifications produced by our proposed method using various values of k. Table 8.3 shows that for all data sets and almost for all values of k our method produces lesser number of misclassifications compared to a conventional

Table 8.3: Mean misclassification on different data sets using different methods

	Iris	Breast-Cancer	Bupa
Conventional MLP	3.66%	5.29%	35.15%
Method in[82]	3.73%	4.51%	34.57%
Our Method, $k = 3$	3.53%	4.15%	32.92%
Our Method, $k = 5$	3.20%	4.09%	32.14%
Our Method, $k = 7$	4.13%	4.40%	31.88%
Our Method, $k = 10$	3.40%	4.21%	31.79%

Table 8.4: Standard deviation of misclassifications on different training-test partitions of Iris data

	I	**II**	**III**	**IV**	**V**	**VI**	**VII**	**VIII**	**IX**	**X**
Ord MLP	0.31	0.51	0.00	0.67	0.42	0.48	0.42	0.31	0.42	0.31
Method in[82]	0.00	0.51	0.00	0.52	0.67	0.73	0.67	0.51	0.42	0.00
Our Method, $k = 3$	**0.00**	**0.42**	0.31	**0.52**	**0.42**	**0.56**	**0.42**	**0.31**	**0.00**	**0.31**
Our Method, $k = 5$	**0.00**	**0.42**	**0.00**	**0.48**	**0.31**	**0.69**	0.73	**0.31**	**0.00**	**0.00**
Our Method, $k = 7$	**0.00**	0.69	0.31	**0.52**	**0.52**	**0.31**	0.73	0.52	**0.00**	**0.00**
Our Method, $k = 10$	**0.00**	**0.51**	**0.00**	**0.51**	**0.52**	**0.31**	0.70	**0.42**	**0.00**	**0.00**

MLP and the method in [82]. Only in case of Iris data, when $k = 7$ our method produces more misclassifications than the other two methods. In case of Iris and Breast-Cancer $k = 5$ gives the best performance, while for Bupa, the best performance is obtained when $k = 10$.

In Table 8.4 we show the standard deviation of the number of misclassifications produced by the 10 runs in each training-test partition of Iris by the three methods. Tables 8.5 and 8.6 include the same statistics for the other two data sets. The roman numbers in the title of each column of Tables 8.4,8.5 and 8.6 depict the training-test partition number. In these tables we write the standard deviation in bold face if it is equal to or lower than that produced by an ordinary MLP or by the method in [82].

From Table 8.4 we see that except for $k = 7$ (row 6) in each row the number of bold faced entries in nine. For $k = 7$ the number of bold faced entries is six. Thus, for Iris data, in most of the cases, the networks trained by our method are more or equally

Table 8.5: Standard deviation of misclassifications on different training-test partitions of Breast-Cancer data

	I	**II**	**III**	**IV**	**V**	**VI**	**VII**	**VIII**	**IX**	**X**
Ord MLP	1.79	1.94	1.50	1.50	1.64	0.73	0.70	1.05	1.50	0.96
Method in[82]	1.83	2.23	1.59	1.22	1.13	0.56	0.99	0.67	0.99	0.67
Our Method, $k = 3$	2.37	**1.33**	**0.82**	**1.10**	**1.03**	0.97	**0.99**	**0.63**	**1.19**	**0.87**
Our Method, $k = 5$	3.39	**0.78**	**0.51**	**1.03**	**0.63**	0.94	**0.87**	**0.94**	**0.99**	**0.51**
Our Method, $k = 7$	2.66	**1.24**	**0.73**	**1.42**	**1.50**	**0.66**	**0.69**	**0.84**	**0.48**	**0.67**
Our Method, $k = 10$	**1.44**	**1.76**	**1.15**	**0.94**	**1.31**	1.17	**0.82**	**0.67**	**0.69**	**0.81**

Table 8.6: Standard deviation of misclassifications on different training-test partitions of Bupa data

	I	**II**	**III**	**IV**	**V**	**VI**	**VII**	**VIII**	**IX**	**X**
Ord MLP	2.75	2.67	2.85	1.95	1.87	3.53	1.26	1.89	2.22	2.58
Method in[82]	3.48	4.97	1.37	0.42	0.69	2.53	1.61	2.60	1.77	1.77
Our Method, $k = 3$	**3.25**	**1.70**	**1.95**	**1.59**	**1.25**	**1.54**	**1.05**	**0.87**	**1.79**	**1.06**
Our Method, $k = 5$	**2.20**	**2.55**	**1.25**	**1.41**	**1.33**	**1.89**	**1.39**	**1.28**	2.35	**1.91**
Our Method, $k = 7$	**3.44**	**2.27**	**1.10**	**0.99**	**0.94**	**1.05**	**0.94**	**1.41**	**1.77**	**1.88**
Our Method, $k = 10$	**2.99**	**1.88**	**1.63**	**1.19**	**1.10**	**1.75**	2.04	**0.91**	**1.59**	**1.39**

consistent than the networks trained by the methods in [82] or an ordinary MLP. As revealed by Tables 8.5 and 8.6, the same is true for Breast-Cancer and Bupa data sets. In fact for Bupa data our network gives an excellent performance as only in two (out of forty) cases our network produces a standard deviation greater than both conventional MLP and MLP trained using the method in [82].

Thus, Tables 8.3-8.6 show that for all the three data sets considered our method produces lesser misclassifications and more or equally consistent generalizations as compared to MLPs trained by the conventional method and MLPs trained using the method in [82].

8.5 Conclusions and Discussion

We presented a simple but effective method to improve the generalization ability of an MLP for function approximation problems. The proposed method involves expanding the available training set and then using new data points in each epoch to avoid overfitting. Our method does not depend on any explicit density estimation technique to generate additional points. It uses a simple k nearest neighbor heuristic which generates additional points maintaining the spatial distribution present in the original data set. The method depends only on a single user defined constant, i.e., the number of nearest neighbors considered. We have also modified our method to deal with classification problems.

The simulation results show that the performance of our system is always better than an MLP trained by the actual data set and also for most cases it is better than other techniques, which use additional points generated through explicit density estimation to train MLPs. Thus, from our limited experiments we may conclude that the points generated by our methods follow the same distribution as that of the original training data set and can be suitably used to avoid overfitting in MLP. However, a formal verification of this fact is needed.

Chapter 9

Conclusions and Future Work

9.1 Conclusions

This thesis dealt with theories and methodologies for designing *efficient* systems from data using neural and neuro-fuzzy frameworks. Efficiency of a system built from data depends not only on the prediction capability of the system but also on many other factors. We identified the following characteristics that can make a system efficient: readability, low complexity, ability to do feature selection, generalization ability, ability to say "don't know", and incremental learnability. We have dealt with the problem of feature selection and have developed systems which have an inherent property of selecting suitable features when they get trained. We have also designed systems capable of online sensor selection. It is worth mentioning here that reducing the number of features also reduces the design cost and complexity as well as the running cost of the system. The use of a smaller feature set also helps to enhance readability. Some of the systems developed in this thesis have the capability to say "don't know" when appropriate and also have incremental learnability. In the preceding six chapters we have developed several methodologies to design systems from input-output data which are *efficient* in some respect. In each chapter we have drawn conclusions regarding the methods proposed and the results presented therein. In this chapter we summarize those again highlighting the main contributions in this thesis.

Chapters 3-5 dealt with a new paradigm for feature selection which we termed as *online* feature selection. The *online* feature selection scheme is unique in the sense that it is integrated with the main learning task. Thus, it equips a system to select the relevant features as it gets trained. This produces better systems, as they use less number of

features and consequently have low complexity and better generalization abilities.

In Chapter 3 we proposed an online feature selection scheme for designing function approximation type systems in a neuro-fuzzy framework. The system is realized using a five layered network. Each layer of the network performs specific task of a fuzzy rule-based system. The system is different from other neural-fuzzy systems for its inherent ability of feature selection. Feature selection is realized using feature gates modeled through special types of functions which we call modulator functions. The feature modulators are so designed that the training process restricts the entry of irrelevant/bad features into the network. The network starts with all possible rules and in subsequent phases it gets pruned to a smaller size. After completion of training we get a small but adequate network which represents a reduced rule base. The rules that are represented by the various parameters of the network can be easily extracted from the network. The system is tested on synthetic and real data sets and found to perform well for both function approximation and feature selection tasks.

The methodology developed in Chapter 3 was modified in Chapter 4 to cope with the classification task. The structure of the classification rules are different from the rules governing a function approximation system. Hence, the structure of the classification network is different from that of the function approximation network. For the classification network we have considered a few issues that are also applicable to the function approximation network. For example, we considered the pruning of *less used* rules, that are never fired by any data point or are fired only by a few data points. We also considered tuning of the input membership functions, and have pointed out exactly when the input membership functions should be tuned to avoid conflict between the update of membership parameters and the feature modulators. The classification network is tested on a set of benchmark problems and the results obtained are compared with other methods available in literature. We found that with almost all data sets our results are better than the other reported results, though our system uses less number of features.

Chapter 5 dealt with the feature selection problem in a different setting. There, we have assumed that the feature set can be partitioned into groups according to the sensory origin of individual feature or by some other criteria. And, our method aims to find the bad group of features, not a bad individual feature. This has immediate applications in designing many real life systems. For most real life systems, data

are collected through sensors, and generally from a single sensory information several features are computed. The designer often wants to reduce the number of sensors. Thus, removing a sensor means removing the entire set of features computed from that sensory information. This leads to a new problem of sensor selection or group feature selection. Selecting individual feature is a special case of group feature selection. Given any grouping of features, if we can find the groups that are not required for a particular task, then we can discard that group or the sensor associated with that group of features. This can give rise to savings in terms of hardware cost and computation cost. This problem of sensor selection or group feature selection has been addressed using two feed-forward architectures called GFSRBF and GFSMLP which are modified versions of the conventional RBF and MLP networks. We have shown that these modified versions of the RBF and MLP networks retain their universal approximation properties. The experimental results show that these networks perform reasonably well both for classification and function approximation problems. We have also studied experimentally the utility of various features selected by GFSRBF and GFSMLP. Our experiments show that for a given data set, the GFSMLP and GFSRBF select different subsets of features. The features selected by GFSRBF are best suited for RBF networks and those selected by GFSMLP are best suited for MLP networks. This reconfirms the fact that the suitability of the features not only depends on the task but also on the tool that is used to do the task.

In Chapters 6-8 we have addressed the problem of generalization in MLP networks. Multilayer perceptron networks are widely used to realize nonlinear mappings between a set of inputs and outputs. Also, it is believed that MLPs can generalize on unknown data with reasonable accuracy. In Chapters 6 and 7 we demonstrated that the generalization capability of MLPs is often over estimated and they can generalize well only on test points which are in the vicinity of the training data. The outputs of an MLP for points which lie far away from the boundary of its training sample are never reliable. This fact, though known, is seldom considered while training or using neural networks. An user who gets a trained neural network may (usually will) not have the training data with him (her). Thus, it is not possible for the user to know about the domain in which the network can perform meaningful generalization. The experiments reported in Chapters 6 and 7 clearly demonstrate that a trained MLP can produce very high response (for classification tasks) on a test point which is far away from the boundary

of the training data. And in most cases such responses are *useless*, i.e., they do not provide acceptable generalization. Ideally, a trained network must not respond to test points which lie far away from its training sample. If a network is equipped with this feature we call it to have a *strict generalization* capability.

In Chapter 6 we proposed a scheme which realizes strict generalization for classification problems. Our method utilizes a scheme to generate additional training examples outside the boundary of each class, and trains a subnet for each class. The subnets are then merged to solve the multiclass problem. This method of training enables an MLP to perform strict generalization. Additionally, it equips an MLP to deal with overlapped classes in a better manner. An MLP trained by our method can signal whether a test point lies in an area of overlap between two or more classes. Thus, the output produced by our method is more interpretable and useful. Another important advantage of this method is that it equips an MLP with incremental learning ability. New knowledge contained in a new training set can be easily incorporated into an MLP trained by our method without affecting its previous memories.

In Chapter 7 we addressed the problem of strict generalization for function approximation tasks. The method developed in Chapter 6 cannot be directly applied to function approximation problems. For function approximation task, we train two networks together. One of the networks detects whether a test point lies within the boundary of the training set, we call this the vigilance network. The other network is an usual MLP which learns the input-output mapping present in the data. The final output is produced by a suitable combination of the outputs of the vigilance and the mapping networks. We have proposed two schemes to train the vigilance network. One scheme requires additional training examples generated outside the boundary of the training sample (as in Chapter 6). The other method does not require any additional training examples but it builds receptive fields around clusters of data points in the training set. The responses of the receptive fields for a test point are then used to decide whether the test point is within the boundary of the training sample or not. We have also demonstrated that a mapping network, vigilance network pair can be used for classification problems. This method of training MLPs for FA problems also equips it for incremental learning.

In Chapter 8 we dealt with the problem of training with insufficient data. This is also related to overfitting in MLP networks. To deal with this problem, we proposed an

easy way to expand any given training set using the density information and use these generated points to train the network. The suggested training scheme uses a set of new data points in every epoch. This reduces the problem associated with a finite training set. Our experimental results show that expanding the training set enhances the generalization ability of an MLP to a large extent.

9.2 Scope of Further Improvement and Research

In this section we summarize some of the weaknesses of the methods that we have described in the preceding six chapters and point out the directions along which the methods can be improved. We also discuss possible extensions of the methods.

In the online feature selection schemes using neuro-fuzzy paradigm discussed in Chapters 3 and 4, we have not considered the important problem of deciding the initial fuzzy sets. Also we did not provide any guideline to decide on the number of fuzzy sets. We used a few fuzzy sets to span the entire domain of each feature. Each fuzzy set has considerable overlap with its adjacent fuzzy sets. Then we have used the pruning phases to discard the rules which are not required by the system. Such blind initialization of a network can lead to a huge initial architecture in case of high dimensional data sets. If there are s input features and we define l fuzzy sets on each feature then we have to start with an initial network which represents l^s rules. For reasonably high values of s the number of rules would be very large and it increases exponentially with the number of features. An effective way to handle this problem would be to apply a clustering algorithm on the input data to decide on the antecedents. Cluster validity indices [12] may be used to decide on the number of clusters and hence the number of fuzzy sets. There have been many previous attempts to select initial rules using clustering algorithms [38, 105, 159, 167]. Use of clustering algorithm to decide on the initial rules (i.e., the initial architecture of the network) does not effect the other functioning of the network. The feature selection strategy will work equally well with a network whose architecture is defined using results of a clustering algorithm. In Chapter 4, we have used exploratory data analysis for identification of the initial structure of the network for RS-Data. But the method described for RS-Data needs further modification to make it general in nature.

Another problem of interest would be to measure the sensitivity of the outputs of a neuro-fuzzy system (or a fuzzy rule based system) with respect to its internal parameters. For both neuro-fuzzy systems described in this thesis, the output depends on the membership function parameters and the certainty factors of the rules. A quantification of sensitivity of output with respect to these parameters would help to gain more insight into the method, and can lead to further improvement of these systems. This is important as we do not like to use a system that is optimal with respect to the design criteria (here training error) but too sensitive to small changes in the system parameters. Such a system may indicate that a minor change in the training data would result in a quite different system.

The group feature selection methodology of Chapter 5 is developed for Radial Basis Function and Multilayer Perceptron type networks. The method can be extended for neuro-fuzzy framework also. Further investigation in this regard is necessary to test the feasibility of the method in case of neuro-fuzzy systems.

The strict generalization for classification discussed in Chapter 6 is a powerful method for low dimensional data sets. For high dimensional data, the generation of points outside the boundary becomes computationally expensive. We have provided a solution to this by using a receptive field vigilance network (Chapter 7). But when a RVN is used for classification problems it cannot provide information about overlapped classes. Our method in Chapter 6 has enhanced ability to deal with overlapped classes and can signal that a test point lies in a region of overlap of two or more classes. Use of RVN for classification task does not equip the network with such capabilities. Further investigation is required for using the RVNs for classification problems. The performance of the system depends on the number of receptive fields chosen, and the value of σ for the receptive fields. Investigations are required to provide guidelines to decide on these parameters. The crux of the method in Chapter 6 is to detect the boundary of the training sample. This problem is different from computing the convex hull of a multidimensional data, as in case the classes are not convex and if the points in the same class form several clusters then the convex hull fails to serve as the boundary of a class. Our definition of boundary based on neighborhood provides a reasonable solution. The algorithm GENERATE (Table 6.4) produces nice results if a proper α is chosen. But we believe that further investigation is required to define class boundaries. The methodologies for strict generalization have been tested with MLP

networks only, but the schemes are quite general in nature and can be applied in case of other learning systems like radial basis function networks, fuzzy rule based systems.

The method to avoid overfitting in MLPs in Chapter 8 depends on a heuristic which generates additional points following the same spatial density as that of the training set. MLPs trained with these generated points give better generalization than MLPs trained with a fixed training set. Thus, it seems that the points generated by our method follow the same probability distribution as that of the training data, but this fact needs to be verified. There are many problems for which data collection is very expensive, and sometimes even risky. Consequently, to design systems for such applications one has to rely on limited ("small") training data. One such example from mining is development of a prediction system for ground vibration from characteristics of rocks and blast materials. For this problem, usually, the available data are very limited because data are site specific and their generation is not only costly but also risky. Our method of generating additional points for training could be useful in this regard. We plan to investigate this in near future. The method for avoiding overfitting in MLP can be used in other paradigms also where an iterative learning procedure is used. We plan to investigate its use for other kinds of networks in future.

Bibliography

[1] H. Akaike, "A new look at statistical model identification," *IEEE Trans. Automatic Control*, vol. 19, pp. 716-723, 1974.

[2] M. Anthony and P.L. Bartlett, *Neural Network Learning: Theoretical Foundations*, Cambridge University Press, Cambridge, UK, 1999.

[3] S. Amari, N. Murata, K.-R. Muller, M. Finke and H.H. Yang, "Asymtotic statistical theory of overtraining and cross-validation," *IEEE Transactions on Neural Networks*, vol. 8, no. 5, pp. 985-996, 1997.

[4] H.R. Barenji and P. Khedkar, "Learning and tuning fuzzy controllers through reinforcements," *IEEE Trans. on Neural Networks*, vol. 3, no. 5, pp. 724-740, 1992.

[5] A. Barron, J. Rissanen, and B. Yu, "The minimum description length principle in coding and modeling," *IEEE Trans. Information Theory*, vol. 44, no. 6, pp. 2743-2760, 1998.

[6] R. Battiti, "Using mutual information for selecting features in supervised neural net learning," *IEEE Transactions on Neural Networks*, vol. 5, no. 4, pp. 537-550, 1994.

[7] L.M. Beleue and K.W. Bauer, " Determining input features for multilayered perceptron," *Neurocomputing*, vol. 7, no. 2, pp. 111-121, 1995.

[8] M. Ben-Bassat, "Use of distance measures, information measures and error bounds in feature evaluation," In: *Handbook of Statistics*, vol. 2, (P. R. Krishnaiah and L. N. Kanal, eds.), North Holland, pp. 773-791, 1982.

[9] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*, Plenum, NY, USA, 1981.

[10] J. C. Bezdek, "A review of probabilistic, fuzzy and neural models for pattern recognition," *Journal of Intelligent and Fuzzy Systems*, vol. 1, no. 1, pp. 1-23, 1993.

[11] J. C. Bezdek, "What is computational Intelligence?," In: J. Zurada, R. Marks, C. Robinson (eds.),*Computational Intelligence: Imitating Life*, IEEE Press, Piscataway, pp. 1-12, 1994.

[12] J. C. Bezdek, J. Keller, R. Krishnapuram and N. R. Pal, *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*, Kluwer, Massachusetts, 1999.

[13] J. C. Bezdek and S. K. Pal, *Fuzzy Models for Pattern Recognition*, IEEE Press, Piscataway, NJ, USA, 1992.

[14] C. Bishop, "Training with noise is equivalent to Tikohonv regularization," *Neural Computation*, vol. 7, pp. 108-116, 1995.

[15] C. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, 1996.

[16] C. Bhattacharyya, L. R. Grate, A. Rizki, D. Radisky, F. J. Molina, M. I. Jordan, M. J. Bissell and I. S. Mian, "Simultaneous classification and relevant feature identification in high-dimensional spaces: application to molecular profiling data," *Signal Processing*, vol. 83, pp. 729-743, 2003.

[17] C. L. Blake and C. J. Merz, UCI Repository of machine learning databases [http://www.ics.uci.edu/ mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science, 1998.

[18] A. L. Blum and P. Langley, "Selection of relevant features and examples in machine learning," *Artificial Intelligence*, vol. 97, no. 1-2, pp. 245-271, 1997.

[19] L. Breiman, "Bagging Predictors," *Machine Learning*, vol 24, no. 2, pp. 123-140, 1996.

[20] L. Breiman, J. H. Friedman, R.A. Olshen and C.J. Stone, Classification and Regression Trees, Belmont, CA: Wadsworth.

[21] J. J. Buckley and Y. Hayashi, "Fuzzy Neural Networks: a survey," *Fuzzy Sets and Systems*, vol. 66, pp. 1-13, 1994.

[22] F. Z. Brill, D. E. Brown and W. N. Martin, "Fast genetic selection of features for neural network classifiers," *IEEE Trans. Neural Networks*, vol. 3, no. 2, pp. 324-328, 1992.

[23] G. A. Carpenter and S. Grossberg, "A massively parallel architecture for a self-organizaing neural pattern recognition machine," *Computer Vision, Graphics and Image Processing*, vol. 37, pp. 54-115, 1987.

[24] D. Chakraborty and N. R. Pal, "Integrated feature analysis and fuzzy rule-based system identification in a neuro-fuzzy paradigm," *IEEE Trans. on Systems Man Cybernetics B*, vol. 31, no. 3, pp. 391-400, 2001.

[25] D. Chakraborty and N. R. Pal, "Designing rule-based classifiers with on-line feature selection: a neuro-fuzzy approach," *Advances in Soft Computing, LNAI 2275*, Springer, pp. 252-260, 2002.

[26] D. Chakraborty and N. R. Pal, "Making a multilayered perceptron say Dont know' when it should, *ICONIP02, Proceedings of the 9-th International Conference on Neural Information Processing*, Singapore, pp. 45-49, 2002.

[27] D. Chakraborty and N. R. Pal, "A Neuro-Fuzzy Scheme for Simultaneous Feature Selection and Fuzzy Rule-Based Classification," *IEEE Trans. on Neural Networks*, vol. 15, no. 1, pp. 110-123, 2004.

[28] D. Chakraborty and N. R. Pal, "Two Connectionist Schemes for Selecting Groups of Features (Sensors)," *Proceedings of FUZZ-IEEE 2003*, pp. 161-166 , 2003.

[29] D. Chakraborty and N. R. Pal, "A Novel Training Scheme for Multilayered Perceptrons to Realize Proper Generalization and Incremental Learning," *IEEE Trans. on Neural Networks*, vol. 14, no. 1, pp. 1-14, 2003.

[30] D. Chakraborty and N. R. Pal, "Expanding the training set for better generalization in MLP," Proceedings of *International Conference on Communication, Devices and Intelligent Systems*, CODIS-2004, pp. 454-457, 2004.

[31] D. Chakraborty and N. R. Pal, "Selecting useful groups of features (sensors) in a connectionist framework," *IEEE Trans. Systems Man Cybernetics B*, (communicated).

[32] D. Chakraborty and N. R. Pal, "Training multilayered perceptrons to realize proper generalization and incremental learning for function approximation", *IEEE Trans. Neural Networks*, (communicated).

[33] S. Chakraborty and N. R. Pal, "Selection of structure preserving features with neural networks," Proceedings of 12^{th} *IEEE Conference on Fuzzy Systems*, FUZZ-IEEE 2003, vol. 2, pp. 822-827, 2003.

[34] Z. Chen and S. Hayken, "On different facets of regularization theory," *Neural Computation*, vol. 14, no. 12, pp. 2791-2846, 2002.

[35] K. J. Cherkauer, "Human expert level performance on a scientific image analysis task by a system using combined artificial neural networks," in: P. Chan, S. Stolfo and D. Wolpert (Eds.), *Proceedings of AAAI-96 Workshop on Integrating Multiple Learned Models for Improving and Scaling Machine Learning Algorithms*, Portland, OR, AAAI Press, Menlo Park, CA, pp. 15-21, 1996.

[36] S. L. Chiu, "Extracting fuzzy rules for pattern classification by cluster estimation", Proc. *sixth Int. Fuz. Systs. Assoc, World Congress (IFSA '95)*, Sao Paulo, Brazil, pp. 1- 4, July 1995.

[37] S. L. Chiu, "Fuzzy model identification based on cluster estimation", J. Int. and Fuzzy Sys., vol. 2, pp. 267 - 278, 1994.

[38] S. L. Chiu, "Extracting fuzzy rules from data for function approximation and pattern classification," *Fuzzy Information Engineering*, eds. D. Dubois, H. Prade and R.R. Yagar, Wiley and Sons, NY, pp. 149-162.

[39] K. J. Cios and W. Pedrycz, "Neuro-fuzzy systems", In: *Handbook of Neural Computation*, IOP Press and Oxford University Press, 1997.

[40] Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541-551, 1989.

[41] Y. Le Cun, J. S. Denker and S. A. Solla, "Optimal Brain Damage," *Advances in Neural Information Processing Systems*, vol. 2, pp. 598-605, Morgan Kaufmann, San Mateo, CA, 1990.

[42] M. Dash and H. Liu, "Feature selection for classification," *Intelligent Data Analysis*, vol. 1, no. 3, pp. 131-156, 1997.

[43] R. De, N. R. Pal and S. K. Pal, " Feature analysis: neural network and fuzzy set theoretic approaches," *Pattern Recognition*, vol. 30, no. 10, pp. 1579-1590, 1997.

[44] P. Devijver and J. Kittler, *Pattern Recognition: A Statistical Approach*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1982.

[45] K. I. Diamantaras and S. Y. Kung, *Principal Component Neural Networks: Theory and Applications*, Wiley, New York, USA, 1996.

[46] D. Driankov, H. Hellendoorn, M. Reinfrank, *An Introduction to Fuzzy Control*, Springer-Verlag, Berlin, 1993.

[47] W. Duch, R. Setiono and J. M. Zurada, "Computational intelligence methods for rule-based data understanding," *Proceedings of the IEEE*, vol. 92, no. 5, pp. 771-805, 2004.

[48] R. O. Duda, P. E. Hart, D. G. Stork, *Pattern Classification*, John Wiley, New York, 2000.

[49] F.-B. Duh and C. T. Lin, "Tracking a Maneuvering Target Using Neural Fuzzy Network," *IEEE Trans. Systems Man and Cybernetics-B*, vol. 34, no. 1, pp. 16-33, 2004.

[50] F.-B. Duh, C.-F. Juang and C. T. Lin, "A neural fuzzy network approach to radar pulse compression," *IEEE Geoscience and Remote Sensing Letters*, vol. 1, no. 1, pp. 15-20, 2004.

[51] R. P. W. Duin, "On the choice of smoothing parameters for parzen estimators of probability density functions," *IEEE Trans. on Computers*, vol. 25, no. 11, pp. 1175-1179, 1976.

[52] P. Dylan and L. F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*, MIT Press, 2001.

[53] B. Efron and R. Tibshirani, *An Introduction to the Bootstrap*, Chapman and Hall, New York, 1993.

[54] A. P. Engelbrecht, "A new pruning heuristic based on variance analysis of sensitivity information," *IEEE Trans. Neural Networks*, vol. 12, no. 6, pp. 1386-1389, 2001.

[55] I. Foroutan and J. Sklansky, "Feature selection for automatic classification of non-Gaussian data," *IEEE Trans. Systems, Man and Cybernetics*, vol. 17, pp. 187-198, 1987.

[56] M. Figureiredo, F. Gomide, "Design of fuzzy systems using neuro-fuzzy networks," *IEEE Transactions on Neural Networks*, vol. 10, no. 4, pp. 815-827, 1999.

[57] B. Firtzke,"Fast learning with incremental RBF networks," *Neural Processing Letters*, vol. 1, no. 1, pp. 2-5, 1994.

[58] D. Fogel, "Review of 'Computational Intelligence: Imitating Life," *IEEE Trans. Neural Networks*, vol. 6, pp. 1562-1565, 1995.

[59] Y. Freund, "Boosting a weak algorithm by majority," *Information and Computation*, vol. 121, no. 2, pp. 256-285, 1995.

[60] L. Fu, H. Hsu, and J. C. Principe, "Incremental backpropagation learning networks," *IEEE Transactions on Neural Networks*, vol. 7, no. 3, pp. 757-761, 1996.

[61] K. Fukunaga, *Statistical Pattern Recognition*, Academic Press, San Diego, CA, USA, 1991.

[62] A. E. Gaweda, J. M. Zurada and R. Setiono, "Input selection in data driven fuzzy modeling," *Proceedings of 2001 IEEE International Conference on Fuzzy Systems*, Melbourne, Australia, 2001.

[63] F. Girosi, M. Jones and T. Poggio, "Regularization theory and neural network architecture," *Neural Computation*, vol. 7, pp. 219-269, 1995.

[64] G. H. Golub and C. F. Van Loan, *Matrix Computations*, John Hopkins University Press, Baltimore, 1989.

[65] A. González and R. Pérez, "Selection of relevant features in a fuzzy genetic learning algorithm," *IEEE Trans. on Systems Man and Cybernetics B*, vol. 31, no. 3, pp. 417-425, 2001.

[66] Y. Grandvalet, "Anisotropic noise injection for input variables relevance determination," *IEEE Transactions on Neural Networks*, vol. 11, no. 6, pp. 1201-1212, 2000.

[67] I. Guyon, A. Elisseeff (Eds.), *Journal of Machine Learning Research*, vol. 3, March 2003.

[68] I. Guyon, A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157-1182, 2003.

[69] M. T. Hagan and M. B. Menhaj, "Training Feedforward Networks with the Marquardt Algorithm," *IEEE Trans. on Neural Networks*, vol. 5, no. 6, pp. 989-993, 1994.

[70] S. Halgamuge and M. Glesner, "Neural networks in designing fuzzy systems for real world applications," *Fuzzy Sets and Syst.*, vol. 65, pp.1-12, 1994.

[71] Y. Hamamoto, S. Uchimura, Y. Matsunra, K. Kanaoka and S. Tomita, "Evaluation of the branch and bound algorithm for feature selection," *Pattern Recognition Letters*, vol. 11, pp. 453-457, 1990.

[72] J. Hampshire and A. Waibel, " A novel objective function for improved phoneme recognition using time delay neural networks," *IEEE Trans. Neural Networks*, vol. 1, no 2, pp. 216-228, 1990.

[73] S. J. Hanson and L. Y. Pratt, "Comparing biases for minimal network construction with backpropagation," In *Advances in Neural Information Processing*, vol. 1, pp. 177-185, Morgan Kaufman, San Mateo, CA, 1989.

[74] R. Haralick, K. Shanmugam and I. Dinstein, "Texture features for image classification," *IEEE Trans. Syst. Man Cybern*, vol. 3, no. 6, 610-621, 1973.

[75] Y. Hayashi, J. Buckley and E. Czogala , "Fuzzy neural network with fuzzy signals and weights," *Int. Jour. Intell. Systems*, vol. 8, pp. 527-537, 1993.

[76] T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning*, Springer, NY, 2001.

[77] S. Haykin, *Neural Networks - A Comprehensive Foundation*, Prentice Hall, New York, 1994.

[78] D.O. Hebb, *The Organization of Behaviour*, John Wiley and Sons, NY, USA, 1949.

[79] J. Hertz, A. Krogh, R. G. Palmer, *Introduction to the theory of neural computation*, Addison-Wesley Publishing Company, CA, USA, 1995.

[80] E. Hewitt, *Real and Abstract Analysis*, Springer Verlag, Berlin, 1965.

[81] G. E. Hinton, "Connectionist learning procedure," *Artificial Intelligence*, vol. 40, pp. 185-234, 1989.

[82] L. Holmstrom and P. Koistinen, "Using additive noise in backpropagation training," *IEEE Trans. Neural Networks*, vol. 3, pp. 24-38, 1992.

[83] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. National Academy of Sciences*, vol. 79, pp. 2554-2558, 1982.

[84] K. Hornik, M. Stinchcombe and H. White, "Multilayered feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 259-366, 1989.

[85] C.-D. Huang, C. T. Lin and N. R. Pal, "Hierarchical learning architecture with automatic feature selection for multiclass protein fold classification," *IEEE Transactions on Nanobioscience* vol. 2, no. 4, pp. 221- 232, 2003.

[86] G. F. Hughes, "On the mean accuracy of statistical pattern recognizers," *IEEE Trans. Information Theory*, vol. 14,pp. 55-63, 1968.

[87] H. Ishibuchi, K. Nozaki, N. Yamamoto, H. Tanaka, "Selecting fuzzy if-then rules for classification problems using genetic algorithms," *IEEE Transactions on Fuzzy Systems*, vol. 3, no. 3, pp. 260-270, 1995.

[88] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Networks*, vol. 1, pp. 297-307, 1988.

[89] A. K. Jain, B. Chandrasekaran, "Dimensionality and sample size considerations in pattern recognition practice," in P.R. Krishnaiah, N.L. Kanal (eds.), *Handbook of Statistics*, vol 2, North-Holland, Amsterdam, pp. 835-855, 1982.

[90] A. K. Jain, R. P. W. Duin, J. Mao,"Statistical pattern recognition: a review," *IEEE Transactions in Pattern Analysis and Machine Learning*, vol. 22, pp. 4-37, 2000.

[91] A. K. Jain and D. Zongker, "Feature selection: evaluation, application and small sample performance," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 19, no. 2, pp. 153-148, 1997

[92] J. -S. R. Jang, "ANFIS: Adaptive-network-based fuzzy inference system," *IEEE Trans. Syst. Man. Cybern.*, vol. 23, pp. 665-685, 1993.

[93] I. T. Jollife, *Principal Component Analysis*, Springer Verlag, New York, 1986.

[94] C. -F. Juang and C.-T. Lin," An online self-constructing neural fuzzy inference network and its applications," *IEEE Trans. Fuzzy Systems*, vol. 6, no. 1, pp. 12-32, 1998

[95] C. -F. Juang and C. -T. Lin, "A recurrent self-organizing neural fuzzy inference network," *IEEE Trans. Neural Networks*, vol. 10, no. 4, pp. 828-845, 1999.

[96] L. Kanal, "Patterns in pattern recognition," *IEEE Transactions in Information Theory*, vol. 20, pp. 697-722, 1974.

[97] A. Kandel and S. C. Lee, *Fuzzy Switching and Automata: Theory and Applications*, Crane Russak, NY, USA, 1979.

[98] P. P. Kanjilal and D. N. Banerjee, " On application of orthogonal transformations for design and analysis of feedforward networks," *IEEE Transactions on Neural Networks*, vol. 6, no. 5, pp. 1061-1070, 1995.

[99] J. Karhunen and J. Joutsensalo, "Representation and separation of signals using nonlinear PCA type learning," *Neural Networks*, vol. 7, pp. 113-127, 1994.

[100] G. N. Karystinos and D. A. Pados, "On overfitting, generalization, and randomly expanded training sets," *IEEE Trans Neural Networks* vol 11, no. 5, pp. 1050-1057, 2000.

[101] N. Kasabov, "Learning fuzzy rules and approximate reasoning in fuzzy neural networks and hybrid systems," *Fuzzy sets and Syst.*, vol. 82, pp. 135-149, 1996.

[102] N. Kasabov, "Evolving fuzzy neural networks for supervised/unsupervised on-line, knowledge-based learning," *IEEE Transactions on Systems, Man and Cybernetics B*, vol. 31, no. 6, pp. 902-918, 2001.

[103] N. Kasabov, R. I. Kilgour and S. J. Sinclair, "From hybrid adjustable neuro-fuzzy systems to adaptive connectionist based systems for phoneme and word recognition," *Fuzzy Sets and Systems*, vol. 103, pp. 349-367, 1999.

[104] N. Kasabov and R. Kozma eds., *Neuro-Fuzzy Techniques for Intelligent Information Systems*, Physica-Verlag, Heidelberg, 1999.

[105] N. Kasabov and Q. Song, "DENFIS: Dynamic evolving neural-fuzzy inference system and its application for time series prediction," *IEEE Transactions on Fuzzy Systems*, vol. 10, no. 2, pp. 144-154, 2002.

[106] N. Kasabov and B. Woodford, "Rule insertion and rule extraction from evolving fuzzy neural networks: algorithms and applications for building adaptive, intelligent expert systems," in Proc. IEEE Int. Conf. Fuzzy Syst. FUZIEEE 99, vol. 3, Seoul, Korea, pp. 1406-1411, Aug. 1999.

[107] M. J. Kearns and U. V. Vazirani,*An Introduction to Computational Learning Theory*, MIT Press, 1994.

[108] J. Keller, Z. Chen, "Learning in fuzzy neural networks utilizing additive hybrid operators," *Proc. Int. Conf. on Fuzzy Logic and Neural Networks, Iizuka, Japan*, pp. 85-87, 1992.

[109] J. Keller and R. Krishnapuram,"Fuzzy decision models in computer vision," In:*Fuzzy Sets, Neural Networks and Soft Computing*, eds. R. Yagar and L. Zadeh, Van Nostrand, pp. 213-232, 1994.

[110] J. Keller, R. Krishnapuram, Z. Chen, O. Nasraoui, "Fuzzy additive hybrid operators for network based decision making," *International Journal of Intelligent Systems*, vol. 9, no. 11, pp. 1001-1024, 1994.

[111] J. M. Keller and H. Qiu, "Fuzzy sets method in pattern recognition," in *Pattern Recognition*, LNCS 301, eds. J. Kittler, Springer-Verlag, pp. 173-182, 1988.

[112] J. Keller and H. Tahani, "Implementation of conjunctive and disjunctive fuzzy logic rules in neural networks," *Int. Jour. Approx. Reasoning*, vol. 6, pp. 221-240, 1992.

[113] J. Keller, R. Yager and H. Tahani, " Neural network implementation of fuzzy logic," *Fuzzy Sets and Systems*, vol. 45, pp. 1-12, 1992.

[114] J. D. Kelly and L. Davis, "Hybridizing the genetic algorithm and the k-nearest neighbors classification algorithm," in *Proc. 4th Intl. Conf. Genetic Algorithms Appl.*, pp. 377-383, 1991.

[115] J. Kim and N. Kasabov, " HyFIS: adaptive neuro-fuzzy inference systems and their application to nonlinear dynamical systems," *Neural Networks*, vol. 12, pp. 1301-1319, 1999.

[116] J. Kittler, "Feature set search algorithms," *Pattern Recognition and Signal Processing*, C.H. Chen ed., Sijthoff and Noordhoff, Alphen aan den Rijn, The Netherlands, pp. 41-60, 1974.

[117] G. J. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic -Theory and Applications*, Prentice Hall, NJ, 1995.

[118] F. Kohavi and G. John, "Wrappers for feature subset selection," *Artificial intelligence*, vol. 97, no. 1, pp. 273-342, 1997.

[119] T. Kohonen, "Self-organization and associative memory," *Springer*, Berlin, 1998.

[120] S. Konishi, T. Ando and S. Imoto, "Bayesian information criteria and smoothing parameter selection in radial basis function networks," *Biometrika*, vol. 91, no. 1, pp. 27-43, 2004.

[121] A. S. Kumar, S. Chowdhury and K. L. Mazumder, "Combination of neural and statistical approaches for classifying space-borne multispectral data," *Proc. of ICAPRDT99, Calcutta, India*, pp. 87-91, 1999.

[122] L. Kuncheva, *Fuzzy Classifier Design*, Physica-Verlag, 2000.

[123] R. Krishnapuram and J. Lee, " Propagation of uncertainty in neural networks," *Proc. SPIE Conf. on Robotics and Computer Vision, 1002, SPIE, Bellingham, WA*, pp. 377-383, 1988.

[124] R. Krishnapuram and J. Lee, "Determining the structure of uncertainty management networks," *Proc. SPIE Conf. on Robotics and Computer Vision, 1192, SPIE, Bellingham, WA*, pp. 592-597, 1989.

[125] R. Krishnapuram and J. Lee, "Fuzzy connective based hierarchical aggregation networks for decision making," *Fuzzy Sets and Systems*, vol. 46, no. 1, pp. 11-27, 1992.

[126] R. Krishnapuram and J. Lee, "Fuzzy-sets-based hierarchical aggregation networks for information fusion in computer vision," *Neural Networks*, vol. 5, pp. 335-350, 1992.

[127] N. Kwak and C. -H. Choi, "Input feature selection for classification problems," *IEEE Transactions on Neural Networks*, vol. 13, no. 1, pp. 143-159, 2002.

[128] A. Laha and N. R. Pal, "Some novel classifiers designed using prototypes extracted by a new scheme based on self organizing feature map," *IEEE trans. Systems Man and Cybernertics B*, vol. 31, no. 8, pp. 881-890, 2001.

[129] C. C. Lee, "Fuzzy logic in control systems: fuzzy logic controler-I," *IEEE Trans. Systems man Cybernetics*, vol. 20, no. 2, pp. 404-418.

[130] C. C. Lee, "Fuzzy logic in control systems: fuzzy logic controler II," *IEEE Trans. Systems man Cybernetics*, vol. 20, no. 2, pp. 419-435.

[131] H. -M. Lee, C. -M. Chen, J.-M. Chen and Y.-L. Jou, "An efficient fuzzy classifier with feature selection based on fuzzy entropy," *IEEE Trans. Systems, Man and Cybernetics*, vol. 31, no. 3, pp. 426-432, 2001.

[132] K. Lee, D. Kwang and H. L. Wang,"A Fuzzy neural network model for fuzzy inference and rule tuning," *International Journal of Uncertainty, Fuzziness, Knowledge-Based Systems*, vol. 2, no. 3, pp. 265-277, 1994.

[133] S. C. Lee and E. T. Lee, " Fuzzy neural networks," *Math. Bio Sc.* vol. 23, pp. 151-177, 1975.

[134] C. C. Li and C. J. Wu, " Generating fuzzy rules for a neural fuzzy classifier," in: *Proc. 3rd IEEE International Conference on Fuzzy Systems FUZZ IEEE'94*, Orlando, pp. 1719-1724, 1994.

[135] C. -T. Lin, I. F. Chung, "A reinforcement neuro-fuzzy combiner for multiobjective control," *IEEE Transactions on System Man and Cybernetics B*, vol. 29, no. 6, pp. 726-744, 1999.

[136] C. -T. Lin and C. S. G. Lee," Neural network based fuzzy logic control and decision system," *IEEE Transactions on Computers*, vol. 40 no. 12, pp. 1320-1335, 1993.

[137] C. -T. Lin and C. S. G. Lee, *Neural Fuzzy Systems*, Prentice Hall, NJ, 1996.

[138] C. -T. Lin, Y. C. Lu, " A neural fuzzy system with linguistic teaching signals," *IEEE Transactions on Fuzzy Systems*, vol. 3, no. 2, pp. 169-189, 1995.

[139] C. -T. Lin, R. -C. Wu and G. -D. Wu, "Noisy speech segmentation/enhancement with multiband analysis and neural fuzzy networks," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 16, no. 7, pp. 927-955, 2002.

[140] Y. Lin and G. A. Cunningham III, "A new approach to fuzzy neural system modeling," *IEEE Transactions on Fuzzy Systems*, vol. 3, no. 2, pp. 190-198, 1995.

[141] Y. Lin, G. A. Cunningham III and S.V. Coggeshall, "Input variable identification - fuzzy curves and fuzzy surfaces," *Fuzzy sets and Systems*, vol. 82, pp. 65-71, 1996.

[142] D. A. Linkens, M. Y. Chen, "Input selection and partition validation for fuzzy modeling using neural networks," *Fuzzy Sets and Systems*, vol. 107, pp. 299-308, 1999.

[143] H. Liu and R. Setiono, "Incremental feature selection," *Applied Intelligence*, vol. 9, no. 3, pp. 217-230, 1998.

[144] C. G. Looney, *Pattern Recognition Using Neural Networks: Theory and Algorithms for Engineers and Scientists*, Oxford University Press, 1997.

[145] D. J. C. MacKay, "Bayesian interpolation," *Neural Computation*, vol. 4, no. 3, 1992

[146] D. J. C. MacKay, "A practical Bayesian framework for backprop networks," *Neural Computation*, vol. 4, no. 3, 1992.

[147] D. J. C. MacKay, "The evidence framework applied to classification networks," *Neural Computation*, vol. 4, no. 5, pp. 698-714, 1992

[148] K. Z. Mao, "Fast orthogonal forward selection algorithm for feature subset selection," *IEEE Trans. on Neural Networks*, vol. 13, no. 5, pp. 1218-1224, 2002.

[149] J. Mao and A. K. Jain, "Artificial neural networks for feature extraction and multivariate data projection," *IEEE Trans. on Neural Networks*, vol. 6, no. 2, pp. 296-317, 1995.

[150] W. S. McCulloch and W.H. Pitts, " A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys*, vol. 5, pp. 115-133, 1943.

[151] S. Mitra and S. K. Pal, "Fuzzy multilayered perceptron, inferencing and rule generation," *IEEE Trans. on Neural Networks*, vol. 6, pp. 51-63, 1995.

[152] P. M. Narendra and K. Fukunaga, "A branch and bound algorithm for feature subset selection," *IEEE Transactions Computers*, vol. 26, no. 9, pp. 917-922, 1977.

[153] D. Nauck and R. Kruse, "A neuro-fuzzy method to learn fuzzy classification rules from data," *Fuzzy Sets and Syst.*, vol 89, pp. 277-288, 1997.

[154] R. M. Neal, *Bayesian Learning for Neural Networks*, Lecture notes in statistics 118, Springer, 1996.

[155] S. J. Nowlan and G. E. Hinton, "Simplifying neural networks by soft weight sharing," *Neural Computation*, vol. 4, pp. 473-493, 1992.

[156] K. Nozaki, H. Ishibuchi, H. Tanaka, "Adaptive fuzzy rule-based classification systems," *IEEE Transactions on Fuzzy Systems*, vol. 4, no. 3, pp. 238-250, 1996.

[157] H. T. Nguyen and M. Sugeno, *Fuzzy Systems: Modeling and Control*, Kluwer Academic Publishers, MA, USA, 1998.

[158] E. Oja, "Principal components, minor components and neural networks," *Neural Networks*, vol. 5, pp. 927-936, 1992.

[159] K. Pal, R. Mudi and N. R. Pal, "A new scheme for fuzzy rule based system identification and its application to self-tuning fuzzy controllers," *IEEE Trans. Systems Man and Cybernetics B* , Aug 2002, to appear.

[160] K. Pal, N. R. Pal and J. M. Keller, "Some neural net realizations of fuzzy reasoning," *Int. Journal of Intelligent Systems*, vol. 13, pp. 859-886, 1998.

[161] N. R. Pal," Soft computing for feature analysis," *Fuzzy Sets and Systems*, vol. 103, pp. 201-221, 1999.

[162] N. R. Pal and C. Bose, "Context sensitive inferencing and "reinforcement-type" tuning algorithms for fuzzy logic systems," *International Journal of Knowledge-Based Intelligent Engineering Systems*, vol. 3, no. 4, 1999.

[163] N. R. Pal, T. C. Cahoon, J. C. Bezdek, K. Pal," A new approach to target recognition for LADAR data", *IEEE Trans. on Fuzzy Systems*, vol 9, no. 1, pp. 44-52.

[164] N. R. Pal and D. Chakraborty, "Simultaneous feature analysis and system identification in a neuro-fuzzy framework," in *Neuro-Fuzzy Pattern Recognition*, (eds.) Bunke and Kandel, pp. 3-22, World Scientific, 2000.

[165] N. R. Pal and D. Chakraborty," Mountain and Subtractive Clustering Method: Improvements and Generalizations," International Journal of Intelligent Systems, vol. 15, pp. 329-341, 2000.

[166] N. R. Pal and K.K. Chintalapudi, "A connectionist system for feature selection," *Neural, Parallel & Scientific Computations*, vol. 5, no. 3, pp. 359-381, 1997.

[167] N. R. Pal, K. Pal, J.C. Bezdek and T.A. Runkler, "Some issues in system identification using clustering," *Int. Joint Conf. on Neural Networks*, IJCNN 1997, IEEE Press, Piscataway, NJ, 2524-2529, 1997.

[168] N. R. Pal and T. Pal," On rule pruning using fuzzy neural networks," *Fuzzy sets and Systems*, vol. 106, pp. 335-347, 1999.

[169] N. R. Pal and E. Vijay Kumar, "Two efficient schemes for structure preserving dimensionality reduction," *IEEE Trans. Neural Networks*, vol. 9, no. 6, pp. 1142 -1153, 1998.

[170] N. R. Pal, E. Vijay Kumar and G.K. Mandal, " Fuzzy logic approaches to structure preserving dimensionality reduction," *IEEE Trans. Fuzzy Systems*, vol. 10, no. 3, pp. 277-286, 2002.

[171] S. K. Pal and B. Chakraborty, "Fuzzy set theoretic measure for automatic feature evaluation," *IEEE Trans. System Man Cybernetics*, vol. 16, no. 5, pp. 754-760, 1986.

[172] S. K. Pal, R. K. De and J. Basak, "Unsupervised Feature Evaluation : A Neuro-fuzzy Approach," IEEE Trans. Neural Networks, vol. 11, no. 2, pp. 366-376, 2000.

[173] S. K. Pal and D. K. Dutta-Majumder, *Fuzzy Mathematical Approach to Pattern Recognition*, Wiley, NY, USA, 1986.

[174] S. K. Pal and S. Mitra, *Neuro-Fuzzy Pattern Recognition*, Wiley, New York, 1999.

[175] S. K. Pal and S. Mitra, "Multilayered perceptrons, fuzzy sets and classification," *IEEE Trans. Neural Networks*, vol. 3, no. 5,pp. 683-697, 1992.

[176] S. K. Pal and N. R. Pal," Soft Computing: Goals, Tools and Feasibility," J. IETE 42(4-5), pp. 195-204, 1996.

[177] D. C. Park, M. A. El-Sharkawi and R. J. Marks II, "An adaptively trained neural network," *IEEE Trans. Neural Networks*, vol. 2, pp. 334-345, 1991.

[178] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Prientice-Hall, Engelwood Cliffs, NJ, 1980.

[179] S. Paul and S. Kumar, "Subsethood-product fuzzy neural inference system," *IEEE Trans. Neural Networks*, vol. 13, no. 3, 578-599, 2002

[180] W. Pedrycz, *Fuzzy Sets Engineering*, C.R.C. Press.

[181] W. Pedrycz, "Fuzzy sets in pattern recognition: methodology and methods," *Pattern Recognition*, vol. 23, pp. 121-146, 1990.

[182] W. Pedrycz, "Fuzzy neural networks and neurocomputations," *Fuzzy Sets and Systems*, vol. 56, pp. 1-28, 1993.

[183] W. Pedrycz, A. Kandel and Y. -Q. Zhang, "Neurofuzzy systems," in *Fuzzy Systems: Modeling and Control*, eds. H.T. Nyugen and M. Sugeno, Kluwer Academic Publishers, MA, USA, 1998.

[184] W. Pedrycz, C. H. Poskar and P. Czezowski, "A reconfigurable fuzzy neural network with in-situ learning," *IEEE Micro Magazine*, pp. 19-30, August 1995.

[185] W. Pedrycz and H. Reformat, "Evolutionary fuzzy modeling," *IEEE Trans. on Fuzzy Systems*, vol. 11, no. 5, pp. 652-665, 2003.

[186] W. Pedrycz and A. Rocha, "A fuzzy set based model of neurons and knowledge based neurons," *IEEE Trans. on Fuzzy Systems*, vol. 1, no. 4, pp. 254-266, 1993.

[187] S. Perkins, K. Lacker and J. Theiler, "Grafting: Fast, Incremental Feature Selection by Gradient Descent in Function Space," *Journal of Machine Learning Research*, vol. 3, pp. 1333-1356, 2003.

[188] S. Perkins and J. Theiler, "Online feature selection using grafting," *Prceedings of the 20-th International Conference on Machine Learning*, Washington D.C., pp. 592-599, 2003.

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

[190] T. Poggio and F. Girosi, "Regularization algorithms for learning that are equivalent to multilayered networks," *Science*, vol. 247, pp. 978-982, 1990.

[191] P. Pudil, J. Novovicova and J. Kittler, "Floating search methods in feature selection," *Pattern Recognition Letters*, vol. 15, pp. 1119-1125, 1994.

[192] W. F. Punch, E. D. Goodman, M. Pei, L. Chia-Shun, P. Hovland and R. Enbody, "Further research on feature selection and classification using genetic algorithms," in *Proc. Int. Conf. on Genetic Algorithms*, 1993, pp. 557-564.

[193] J. R. Quinlan, "Learning efficient classification procedures and their applications to chess end games," In R.S. Michalski, J.G. Carbonell and T.M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*, San Fransisco, CA: Morgan Kaufmann.

[194] J. R. Quinlan, C4.5: Programs for machine learning, San Fransisco: Morgan Kaufmann.

[195] M. L. Raymer, W. F. Punch, E. D. Goodman, L. A. Kuhn and A. K. Jain, "Dimensionality reduction using genetic algorithms," *IEEE Trans. on Evolutionary Computing*, vol. 4, no. 2, pp. 164-171, 2000.

[196] R. Reed, "Pruning algorithms - A survey," *IEEE Trans. on Neural Networks*, vol. 4, no. 5, pp. 740-747, 1993.

[197] M. R. Rezaee, B. Goedhart, B. P. F. Lelieveldt and J.H.C. Reiber, "Fuzzy feature selection," *Pattern Recognition*, vol. 32, pp. 2011-2019, 1999.

[198] B. D. Ripley and J. P. Rasson, "Finding the edge of a poisson forest," *Journal of Applied Probability*, vol. 14, pp. 483-491, 1977.

[199] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14 , pp. 465-471, 1978.

[200] F. Rosenblatt, "The Perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, pp. 386-488, 1958.

[201] J. Rubner and K. Schulten, "Development of feature detectors by self organization," *Biological Cybernetics*, vol. 62, pp. 192-199, 1990.

[202] J. Rubner, P. Tavan, "A self organizing network for principal component analysis," *Europhysics Letters*, vol. 10, pp. 693-698, 1989.

[203] D. W. Ruck, S. K. Rogers, M. Kabrisky, "Feature selection using a multilayered perceptron," *Journal of Neural Network Computing*, vol. 20, pp. 40-48, 1990.

[204] D. E. Rumelhart, G. E. Hinton, R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533-536, 1986.

[205] M. Russo, "Genetic fuzzy learning," *IEEE Trans. Evolutionary Computation*, vol. 4, no. 3, pp. 259-273, 2000.

[206] M. Russo, " Comments on "A New Approach to Fuzzy-Neural Systems Modeling"," *IEEE Transactions on Fuzzy Systems"*, vol. 4, no. 2, pp. 209-210, 1996.

[207] M. Russo, "FuGeNeSys - A fuzzy genetic neural system for fuzzy modeling," *IEEE Trans Fuzzy Systems*, vol. 6, no. 3, pp. 373-387.

[208] M. Sugeno, T. Yasukawa, "A fuzzy-logic based approach to qualitative modeling," *IEEE Transactions on Fuzzy Systems*, vol. 1, pp. 7-31, 1993.

[209] J. W. Sammon Jr., "A nonlinear mapping for data structure analysis," *IEEE Trans. Computers*, C-18, pp. 401-409, 1969.

[210] S. Schaal, C. G. Atkeson, " Constructive incremental learning from only local information," *Neural Computation*, vol. 10, pp. 2047-2084, 1998.

[211] B. Schölkopf, A. J. Smola and K. -R. Müller, "Nonlinear component analysis as a kernel eigen value problem," *Neural Computation*, vol. 10, no. 5, pp. 1299-1319, 1998.

[212] B. Schölkopf, J. Platt, J. Shawe-Taylor, A. J. Smola and R.C. Williamson, " Estimating the support of a high-dimensional distribution," *Neural Computation*, vol. 13, no. 7, pp. 1443-1471, 2001.

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

[214] R. Setiono, "Extracting rules from neural networks by pruning and hidden unit splitting," *Neural Computation*, vol. 9, no. 1, pp. 205-225, 1997.

[215] R. Setiono and H. Liu, "Neural network feature selector," *IEEE Trans. Neural Networks*, vol. 8, pp.654-662, 1997.

[216] R. Setiono, W. K. Leow and J. M. Zurada, "Extraction of rules from artificial neural networks for nonlinear regression," *IEEE Trans. Neural Networks*, vol. 13, no. 3, pp. 564-587, 2002.

[217] J. J. Shann, and H. C. Fu," A fuzzy neural network for rule acquiring on fuzzy control systems," *Fuzzy Sets and Systems*, vol. 71, pp. 345-357, 1995.

[218] W. Siedlecki and J. Sklansky, "A note on genetic algorithms for large scale feature selection," *Pattern Recognition Letters*, vol. 10, pp. 335-347, 1989.

[219] S. P. Smith, A. K Jain, "Testing of uniformity in multidimensional data," *IEEE Trans. on Pattern Analysis and Machine Learning*, vol. 6, no. 1, pp. 73-81, 1984.

[220] F. F. Soulie, P. Gallinary (eds.), *Industrial Applications of Neural Networks*, World Scientific, 1998.

[221] J. M. Steppe, K. W. Bauer, S. K. Rogers, "Integrated feature and architecture selection," *IEEE Trans. Neural Networks*, vol. 7, pp. 1007-1014, 1996.

[222] M. Sugeno and T. Yasukawa, " A Fuzzy-Logic based approach to qualitative modeling," *IEEE Transactions Fuzzy Systems*, vol. 1, no. 1, pp. 7-31, 1993.

[223] G. Schwarz, "Estimating the dimension of a model," *The Annals of Statistics*, vol. 6, no. 2, pp. 461-464, 1978.

[224] I. A. Taha and J. Ghosh, "Symbolic interpretation of artificial neural networks," *IEEE Trans. Knowledge and Data Engg.*, vol. 11, pp. 448-462, 1998.

[225] A. B. Tickle, R. Andrews, M. Golea and J. Diederich,"The truth will come to light: directions and challenges in extracting the knowledge embedded within trained artificial neural networks," *IEEE Trans. Neural Networks*, vol. 9, pp. 1057-1068, 1998.

[226] A. N. Tikhonov and V. Y. Arsenin, *Solution of Ill Posed Problems*, Washington D.C.: W.H. Winston, 1977.

[227] G. G. Towell and J. W. Shavlik, "Extracting refined rules from knowledge-based neural networks," *Machine Learning*, vol. 13, no. 1, pp. 71-101, 1993.

[228] T. Trappenberg, *Fundamentals of Computational Neuroscience*, Oxford University Press, 2002.

[229] E. C. C. Tsang, D. S. Yeung and X. Z. Wang, " OFSS: optimal fuzzy-valued feature subset selection," *IEEE Trans. Fuzzy Systems*, vol. 11, no. 2, pp. 202-213, 2003.

[230] N. Ueda, "Optimal linear combination of neural networks for improving classification performance," *IEEE Transaction Pattern Analysis and Machine Intelligence*, vol. 22, no. 2, pp. 207-215, 2000.

[231] L. G. Valiant,"A theory of the learnable," *Communications of the ACM*, vol. 27, no. 11, pp. 1134-1142, 1984.

[232] V. N. Vapnik, *Statistical Learning Theory*, Wiley, New York, 1998.

[233] V. N. Vapnik, *Estimation of Dependences Based on Empirical Data*, Springer-Verlag, New York, 1982.

[234] M. Vidyasagar,*Theory of Learning and Generalization, with Application to Neural Networks and Control Systems*, Springer, New York, 1997.

[235] L. X. Wang, J. M. Mendel, "Generating fuzzy rules by learning from examples," *IEEE Transactions on Systems, Man, Cybernetics*, vol. 22, pp. 1414-1427, 1992.

[236] A. S. Weigend, "On overfitting and effective number of hidden units," In Proc. of *Connectionist Models's Summer School*, pp. 335-342, Lawrence Erlbaum, Hillsdale, N.J., 1993.

[237] J. Weston, A. Gammerman, M. Stitson, V. Vapnik, V. Vovk and C. Watkins, "Support vector density estimation", *Advances in Kernel Methods - Support Vector Learning*, MIT Press, Cambridge, MA, pp. 293-306, 1999.

[238] D. H. Wolpert, "Stacked Generalization," *Neural Networks*, vol. 5, no. 2, pp. 241-259, 1992.

[239] G. -D. Wu and C. -T. Lin, "A Recurrent Neural Fuzzy Network for Word Boundary Detection in Noisy Environment," *International Journal of Fuzzy Systems*, vol 2, no 1, pp. 31-38, 2000.

[240] S. Wu and M. J. Er,"Dynamic fuzzy neural networks - a novel approach to function approximation," *IEEE Transactions on System Man and Cybernetics B*, vol. 30, no. 2, pp.358-363, 2000.

[241] R. R. Yagar and D. P. Filev, "Approximate clustering via the mountain method," *IEEE Trans. Systems Man Cybernetics*, vol. 24, no. 8, pp. 1279-1284, 1994.

[242] T. Yamakawa, "Stabilization of an Inverted Pendulum by a High-speed Fuzzy Logic Controller Hardware System," *Fuzzy Sets and Systems*, vol. 32, pp. 161-180, 1989.

[243] T. Yamakawa, "A fuzzy inference engine in nonlinear analog mode and its application to a fuzzy logic control," *IEEE Trans. on Neural Networks*, vol. 4, no. 3, pp. 496-522, 1993.

[244] T. Yamakawa, "Silicon implementation of a fuzzy neuron," *IEEE Trans. on Fuzzy Systems*, vol. 4, no. 4, pp. 488-501, 1996.

[245] K. Yamauchi, N. Yamaguchi, N. Ishii, "Incremental learning methods with retrieving of interfed patterns," *IEEE Trans. Neural Networks*, vol. 10, no. 6, pp. 1351-1365, 1999.

[246] X. Yao and Y. Liu, "Making use of population information in evolutionary artificial neural networks," *IEEE Transactions on Systems Man and Cybernetics B*, vol. 28, no. 3, pp. 417-425, 1998.

[247] S. Yao, C. Wei, Z. He, "Evolving fuzzy neural networks for extracting rules," in: *Proc. 5th IEEE International Conference on Fuzzy Systems FUZZ IEEE'96*, New Orleans, pp. 361-367, 1996.

[248] D. Yeung and X. Sun, "Using function approximation for sensitivity analysis of MLP," *IEEE Trans. Neural Networks*, vol 13, pp. 34-44, 2002.

[249] L. Zadeh, "Fuzzy logic and soft computing: issues, contentions and perspectives," *Proc.* 3^{rd} *International Conference on Fuzzy logic, Neural Networks and Soft Computing, IZUKA*, Japan, pp. 1-2, 1994.

[250] L. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, pp. 338-353, 1965.

[251] X. Zeng, D. S. Yeung, "Sensitivity analysis of multilayer perceptron to input and weight perturbation," *IEEE Trans. Neural Networks*, vol. 12, no. 6, pp. 1358-1366, 2001.

[252] Z. -H. Zhou, J. Wu and W. Tang, "Ensembling neural networks: many could be better than all," *Artificial Intelligence*, vol. 137, no. 1, pp. 239-263.

[253] J. M. Zurada, *Introduction to Artificial Neural Systems*, West Publishing Company, MN, 1992.

[254] J. M. Zurada, A. Malinowski, S. Usui, "Perturbation method for deleting redundant inputs of perceptron networks," *Neurocomputing*, vol. 14, pp. 177-193, 1997

[255] J. M. Zurada and A. Lozowski, Generating linguistic rules from data using neuro-fuzzy framework, in *4th Int. Conf. on Soft Computing, IIZUKA96*, Iizuka, Japan, vol. 2, pp. 618621, 1996.

[256] *ftp://ftp.dice.ucl.ac.be/pub/neural-nets/ELENA*

[257] *http://www.dice.ucl.ac.be/neural-nets/Research/Projects/ELENA/elena.htm*

[258] *http://www.bangor.ac.uk/ mas00a/Z.txt* and *http://www.bangor.ac.uk/ mas00a/Zte.txt*

[259] B.E. Boser, I.M. Guyon and V.N. Vapnik, "A training algorithm for optimal margin classifiers," in 5^{th} *Annual ACM Workshop on Computational Learning Theory*, pp. 144-152, ACM Press, 1992.

[260] C.J.C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol 2, no. 2, pp. 1-47, 1998

[261] C. Cortes and V. Vapnik, "Support Vector Networks," *Machine Learning*, vol 20, n0. 3, pp. 273-297, 1995.

[262] N. Cristianini and J. Shaw-Taylor, *An Introduction to Support Vector Machines*, Cambridge University Press, 1999.

[263] T. Joachims, "Making large scale support vector learning practical," *Advances in Kernel Methods: Support Vector Learning*, pp. 169-184, MIT Press, 1999.

[264] T. Joachims, "Text categorization with support vector machines," in *European Conference on Machine Learning*, 1998.

[265] C.-J. Lin, "Formulations of support vector machines: a note from an optimization point of view," *Neural Computation*, vol. 17, pp. 307-317, 2001.

[266] J.C. Platt, "Fast training of support vector machines using sequential minimal optimization", *Advances in Kernel Methods: Support Vector Learning*, pp. 185-208, MIT Press, 1999.

[267] C.W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE Trans. Neural Networks*, vol. 13, no. 2, pp. 415-425, 2002.

[268] J.C. Platt, N. Cristianini, and J. Shawe-Taylor, " Large margin DAGs for multiclass classification," *Advances in Neural Information Processing Systems*, vol. 12, pp. 547-553, MIT Press, 2000.

[269] B. Schölkopf, A. J. Smola, *Learning with kernels*, MIT Press, 2002.

[270] A.J. Smola and B. Schölkopf, "A tutorial on support vector regression," *NeuroCOLT2 Technical Report*, NC2-TR-1998-030, 1998.

[271] C. Ding and I. Dubchak, "Multi-class protein fold recognition using support vector machines and neural networks," *Bioinformatics*, vol. 17, pp. 349-358, 2001

[272] A. Zien, G. Ratsch, S. Mika, B. Schölkopf, T. Lengauer, and K.-R. Muller, "Engineering support vector machine kernels that recognize translation initiation sites," *BioInformatics*, vol. 16, no. 9, pp. 799-807, 2000

[273] T. S. Furey, N. Duffy, N. Cristianini, D. Bednarski, M. Schummer, and D. Haussler, "Support Vector Machine Classification and Validation of Cancer Tissue Samples Using Microarray Expression Data," *Bioinformatics*, vol. 16, no. 10, pp. 906-914, 2000.

[274] I. Guyon, J. Weston, S. Barnhill, V. Vapnik, "Gene selection for cancer classification using support vector machines," *Machine Learning*, vol. 46, pp. 389-422, 2000.

[275] H. Drucker, D. Wu and V. Vapnik, "Support vector machines for spam categorization," *IEEE Trans. on Neural Networks* , vol 10, no. 5, pp. 1048-1054. 1999.

[276] I.-F. Chung, C.-D. Huang, Y.-H. Shen and C.-T. Lin, "Recognition of structure classification of protein folding by NN and SVM hierarchical learning architecture", *Proceedings of ICANN/ICONIP 2003*, pp. 1159-1167, Springer-Verlag, Berlin, 2003.

[277] M. Pontil and A. Verri, "Support vector machine for 3-D object recognition", *IEEE Trans. Pattern Analysis and Machine Learning*, vol. 20, pp. 637-646, 1998.

[278] E. Osuna, R. Fruend and F. Girosi, "Training support vector machines: an application to face detection", in *Proc. Computer Vision and Pattern Recognition*, pp. 181-201, 2001.

[279] Q. Zhao and J. Principe, "Automatic Target Recognition with Support Vector Machines," *NIPS-98 Workshop on Large Margin Classifiers*, 1998.

[280] S. Romdhani, P. Torr and B. Schölkopf and A. Blake, "Computationally efficient face detection", in *Proc. International Conference on Computer Vision*, vol. 2, pp. 695-700, 2001.

[281] Y. Grandvalet and S. Canu,"Adaptive scaling for feature selection in SVMs," *Neural Information Processing Systems*, vol. 15, 2002.

[282] K. Duan, S.S. Keerthi and A.N. Poo, "Evaluation of simple performance measures for tuning SVM hyperparameters," *Neurocomputing*, Vol. 51, pp. 41-59, 2003.

[283] J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio and V. Vapnik, "Feature selection for SVMs", in *Advances in Neural Information Processing Systems*, vol 13, pp. 668-674, 2000.

[284] J. Bi, K.P. Bennett, M. Embrechts, C.M. Breneman, M. Song, "Dimensionality reduction via sparse support vector machines", *Journal of Machine Learning Research*, vol. 1, pp. 1-16, 2002.

[285] A. Raktomamonjy, "Variable selection using SVM criteria", *Journal of Machine Learning Research*, vol. 3, pp. 1357-1370, 2003.

Publications of the Author Related to the Thesis

A1. D. Chakraborty and N.R. Pal, "Integrated feature analysis and fuzzy rule-based system identification in a neuro-fuzzy paradigm," *IEEE Trans. on Systems Man Cybernetics B*, vol. 31, no. 3, pp. 391-400, 2001.

A2. D. Chakraborty and N.R. Pal, "A Novel Training Scheme for Multilayered Perceptrons to Realize Proper Generalization and Incremental Learning," *IEEE Trans. on Neural Networks*, vol. 14, no. 1, pp. 1-14, 2003.

A3. D. Chakraborty and N.R. Pal, "A Neuro-Fuzzy Scheme for Simultaneous Feature Selection and Fuzzy Rule-Based Classification," *IEEE Trans. on Neural Networks*, vol. 15, no. 1, pp. 110-123, 2004.

A4. D. Chakraborty and N.R. Pal, "Selection useful groups of features (sensors) in a connectionist framework," *IEEE Trans. Systems Man Cybernetics B*, (communicated).

A5. D. Chakraborty and N.R. Pal, "Training multilayered perceptrons to realize proper generalization and incremental learning for function approximation", *IEEE Trans. Neural Networks*, (communicated).

A6. N.R. Pal and D. Chakraborty, "Simultaneous feature analysis and system identification in a neuro-fuzzy framework," in *Neuro-Fuzzy Pattern Recognition*, (eds.) Bunke and Kandel, pp. 3-22, World Scientific, 2000.

A7. D. Chakraborty and N.R. Pal, "Designing rule-based classifiers with on-line feature selection: a neuro-fuzzy approach," *Advances in Soft Computing, LNAI 2275*, Springer, pp. 252-260, 2002.

A8. D. Chakraborty and N.R. Pal, "Two Connectionist Schemes for Selecting Groups of Features (Sensors)," *Proceedings of FUZZ-IEEE 2003*, pp. 161-166 , 2003.

A9. D. Chakraborty and N.R. Pal, "Making a multilayered perceptron say "Dont know" when it should," ICONIP 02, *Proceedings of the 9-th International Conference on Neural Information Processing*, Singapore, pp. 45-49, 2002.

A10. D. Chakraborty and N.R. Pal, "Expanding the training set for better generalization in MLP," *Proceedings of International Conference on Communication, Devices and Intelligent Systems, CODIS-2004*, pp. 454-457, 2004.