LIBRARY
Michigan State
University

PLACE IN RETURN BOX to remove this checkout from your record.
TO AVOID FINES return on or before date due.

DATE DUE	DATE DUE	DATE DUE
AUG 14 1993		

MSU Is An Affirmative Action/Equal Opportunity Institution
c:\circ\datedue.pm3-p.1

DESIGN AND ANALYSIS OF NEURAL NETWORKS FOR PATTERN RECOGNITION

By

Jianchang Mao

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Computer Science Department

1994

ABSTRACT

DESIGN AND ANALYSIS OF NEURAL NETWORKS FOR PATTERN RECOGNITION

By

Jianchang Mao

There are many common links between neural networks and classical statistical pattern recognition approaches for designing a recognition system. For example, many statistical pattern recognition approaches can be mapped onto a neural network architecture. On the other hand, some well-known neural network models are related to classical statistical methods. In spite of numerous successful applications of feedforward networks reported in the literature, their generalization behavior has not been well-understood. This dissertation further expands the common links between the two disciplines through design of neural networks for pattern recognition problems and analysis of the generalization ability of feedforward networks.

A number of neural networks and learning algorithms for feature extraction, data projection, classification and clustering are proposed. These networks include: linear discriminant analysis network, network for Sammon's projection, nonlinear projection based on Kohonen's self-organizing maps, nonlinear discriminant analysis network, network-based k-nearest neighbor classifier, and network for detecting hyperellipsoidal clusters. A comparative study of five networks for feature extraction and data projection is also conducted.

The generalization ability of a feedforward network refers to its performance on independent test data relative to the performance on the training data. Two different approaches, Vapnik's theory and Moody's approach, for studying the generalization ability of feedforward networks are compared. Moody's approach is extended to a more general setting which allows the sampling points of test data to be different from those in training data according to an additive noise model. Monte Carlo experiments are conducted to verify Moody's result and our extension of his model, and to demonstrate the role of the weight-decay regularization in reducing the effective number of parameters. We also present a taxonomy of regularization techniques in neural networks, and demonstrate that a variety of networks and learning algorithms can fit into this framework. One significant impact of the theoretical analysis of the generalization ability of feedforward networks is that it reveals how different factors involved in designing feedforward networks interact with each other. A practical node pruning method is proposed for designing parsimonious networks which generalize well, and for reducing the dimensionality of the feature vector.

ACKNOWLEDGMENTS

I am deeply indebted to my advisor, Professor Anil K. Jain, who has been an inexhaustive source of ideas, encouragement and support throughout my program at Michigan State University. Without him, this dissertation would not have been completed. I am very lucky to have such a distinguished and responsible professor as my advisor. I am grateful to professors John Weng, Michael Shanblatt, and Alvin Smucker, for serving on my committee, sharing ideas with me, and providing critical comments on my dissertation. I would like to express my special thanks to late Professor Richard Dubes, who had served on my dissertation committee till the last minute of his life. His invaluable comments on my dissertation proposal helped me to shape my dissertation research.

My sincere thank goes to my wife Yao for her love, patience, encouragement, support, and understanding. I share all my academic accomplishments with her. Thanks to my son, David, who was born in this period, for having brought a lot of joy to my family. I would also like to express my heartful thanks to my mother, Aiqing, for her lifelong love, sacrifice and support. Without her love, sacrifice, and hard work during my childhood and teenage years, I would not have received any education.

The last stage of this dissertation was completed during my employment at the IBM Almaden Research Center. I would like to express my deep gratitude to Dr. K. Mohiuddin, the manager of Integrated Data Management group, for his understanding, support, valuable discussions on my dissertation, and challenging me with

difficult real-world problems. Research reported in Chapter 7 was completed when I worked in his group as a student co-op in 1993. Thanks also to other members of his group, Dick Casey, Raymond Lorie, Andreas Kornai, and Sandeep Gopisetty for exposing me to their interesting research. A special acknowledgment goes to Dr. Eric Saund of XEROX Palo Alto Research Center, where I spent one summer as a research intern. A broad exposure to various research problems made my internship a valuable experience.

During my stay in the Pattern Recognition and Image Processing (PRIP) Laboratory at Michigan State University, I met many distinguished visitors and brilliant students. I would like to thank our two summer visitors, Martin Kraaijveld and Wouter Schmidt from Delft University of Technology, The Netherlands, for many motivated discussions and pleasant interaction. Thanks also to PRIPpies: Sushma Agrawal, Yao Chen, Shaoyun Chen, Jinlong Chen, Yuntao Cui, Chitra Dorai, Marrie-Piere Dubuisson, Sally Howden, Michiel Huizinga, Kalle Karu, Jan Linthorst, Sharathcha Pankanti, Nalini Ratha, Dan Swets, Marilyn Wulfekuhler, Yu Zhong, and former PRIPpies Hans Dulimart, Farshid Farrokhnia, Pat Flynn, Qian Huang, Greg Lee, Sateesh Nadabar, Tim Newman, Narayan Raja, and Deborah Trytten. Many of them have become good friends of mine. I enjoyed many open discussions and wonderful group activities which made my graduate study unforgettable.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Automatic (machine) recognition, description, classification, and grouping of patterns are important activities in a variety of engineering and scientific disciplines such as biology, psychology, medicine, marketing, computer vision, artificial intelligence, pattern recognition, and remote sensing. What is a pattern? Watanabe [172] defines a pattern as an entity, vaguely defined, that could be given a name. A pattern is represented by a vector of measured properties or features and their interrelationships. For example, the recognition problem can range from using fingerprints to identify a suspect in a criminal case to finding defects in a printed circuit board [79]. The corresponding features could be minutiae (i.e., location of minutiae and pattern type) in the fingerprint or the broken line segments or loops in the printed circuit board. A pattern recognition system is expected to (i) identify a given pattern as a member of already known or defined classes (supervised classification), or (ii) assign a pattern to a so far unknown class of patterns (unsupervised classification or clustering).

Design of a pattern recognition system essentially involves the following three steps: (i) data acquisition and preprocessing, (ii) representation/feature extraction, and (iii) decision making or clustering. The problem domain dictates the choice of sensor, preprocessing techniques, and decision making model. The domain-specific knowledge is implicit in the design and is not represented in a separate module (as

is commonly done in expert systems). Some of the early work in pattern recognition was biologically motivated; the objective was to develop a recognition system whose structure and strategy followed the one utilized by humans. The work on perceptrons [149] is a good example of such attempts which were not very successful. The current serious activity in the area of artificial neural networks and connectionist models is reminiscent of this early work.

The three most well-known models for designing a pattern recognition system are: (i) statistical, (ii) syntactic or structural, and (iii) neural networks. This thesis addresses the design and analysis of neural networks for pattern recognition. We will show that there is a considerable amount of overlap between the statistical pattern recognition (SPR) approaches and neural network approaches to design pattern recognition systems. The following two subsections present an overview of these two models.

1.1 Statistical Pattern Recognition

Statistical pattern recognition is a relatively mature discipline and a number of commercial recognition systems have been designed based on this approach. In statistical pattern recognition, a pattern is represented by a set of d numerical measurements or a d-dimensional feature vector. Thus, a pattern can be viewed as a point in this d-dimensional space. A SPR system is operated in two modes: training and recognition. In the training mode, the feature extractor is designed to find the appropriate features for representing the input patterns and the classifier is trained by a set of training data to partition the feature space so that the number of misclassified patterns is minimized. In the recognition mode, the trained classifier assigns the input test pattern to one of the categories/clusters based on the extracted feature vector.

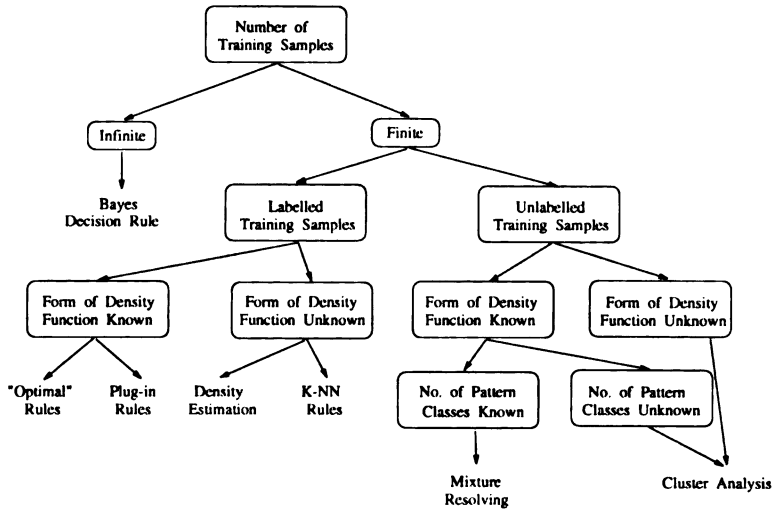


Figure 1.1. Dichotomies in the design of a SPR system.

The decision making process in SPR can be summarized as follows. Given a pattern represented by a d-dimensional feature vector, $\mathbf{x} = (x_1, x_2, \cdots, x_d)^T$, assign it to one of c categories, $\omega_1, \omega_2, \cdots, \omega_c$. Depending on what kind of information is available about the class-conditional densities, various strategies can be used to design a classifier. If all the class-conditional densities $p(\mathbf{x}|\omega_i)$, $i = 1, 2, \cdots, c$ are completely specified, then the optimal Bayes decision rule can be used to establish the decision boundaries between pattern classes. However, the class-conditional densities are never known in practice and must be learned from the training patterns. If the functional form of the class-conditional densities is known, but some of the parameters of the densities are unknown, then the problem is called a parametric decision making problem. Otherwise, either the density function must be estimated or some nonparametric decision rule must be used. The various dichotomies which appear in the design of a SPR system are shown in the tree diagram of Fig. 1.1. Various approaches in neural networks and statistical pattern recognition will be compared using this diagram.

1.2 Artificial Neural Networks

The field of neural networks is an interdisciplinary area of research. A thorough study of neural networks requires some understanding of neurophysiology, control theory, mathematics, statistics, decision making, and distributed computing. The diverse nature of topics studied under this heading makes it impossible for us to address all the related issues and topics. Neural networks have been viewed under various perspectives [109, 74, 147, 178, 9, 12].

Research in neural networks has experienced three consecutive cycles of enthusiasm and skepticism. The first peak, dating back to the 1940's, is due to McCullough and Pitt's pioneering work [117]. The second period of intense activity occurred in the 1960's which featured Rosenblatt's perceptron convergence theorem [149] and Minsky and Papert's work showing the limitations of simple perceptron [122]. Minsky and Papert's results dampened the enthusiasm of most researchers, especially those in the computer science community. As a result, there was a lull in the neural network research for almost 20 years. Since the early 1980's, neural networks have received considerable renewed interest. The major developments behind this resurgence include Hopfield's energy approach [68] in 1982, and the backpropagation learning algorithm for multilayer perceptrons (multilayer feedforward networks) which was first proposed by Werbos [176], reinvented several times, and popularized by Rumelhart et al. [152] in 1986. Anderson and Rosenfeld [3] provide a detailed historical account of developments in neural networks.

Different interconnection strategies lead to different types of neural networks (feedback versus feedforward) which require different learning algorithms. Another dichotomy is based on the type of learning algorithm used. There are three types of learning algorithms: supervised, unsupervised, and hybrid (combination of supervised

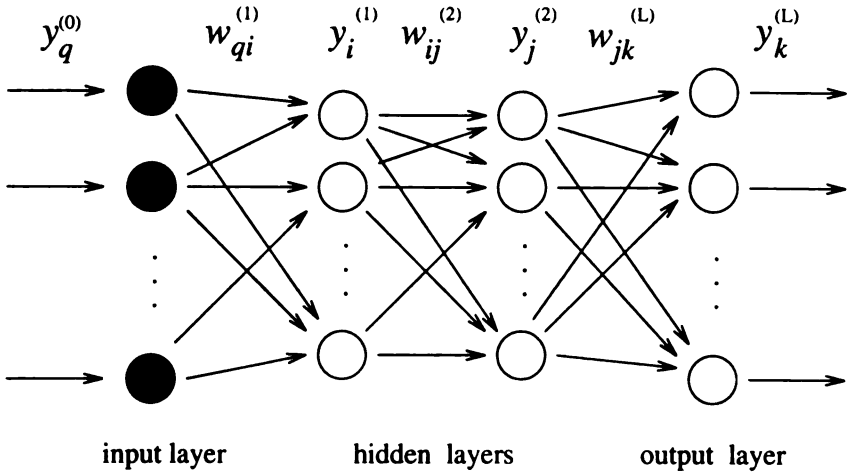


Figure 1.2. A typical three-layer feedforward network.

and unsupervised methods). Supervised learning includes a special case of reinforcement learning in which only qualitative target values (yes or no) are used.

Multilayer feedforward networks have emerged as a popular model for pattern classification tasks. A standard L-layer feedforward network* consists of one input stage, $L-1$ hidden layers, and one output layer of nodes which are successively connected in a feedforward fashion with no connections between nodes in the same layer and no feedback connections between layers. A typical three-layer feedforward network is shown in Figure 1.2. It has been shown by many researchers that a 2-layer network is capable of forming an arbitrary nonlinear decision boundary and approximating any continuous nonlinear function (see, [63, 188]), provided the number of nodes in the hidden layer can be arbitrarily large. Unfortunately, these theoretical studies have not addressed the practical problem of selecting the number of nodes in the hidden layer. In fact, the number of nodes in the hidden layer can not be arbitrarily large due to the phenomenon of "curse of dimensionality" [142]. Moreover, these theoretical studies have not introduced any new practical learning methods. As

*In this thesis, we adopt the convention that the input nodes are not counted as a layer.

a result, the backpropagation algorithm [152] has been the most commonly used method for training multilayer feedforward networks.

Neural networks were originally proposed as simplified models of biological neural networks, and have been studied using mathematical tools, computer software simulations, and hardware implementations. Although many existing neural network models are extremely simplified versions of biological neural networks, they are valuable for understanding some principles of biological computation. One of the ultimate goals of neural networks is to explore why biological systems can effortlessly perform some tasks, such as pattern recognition, which are so difficult for modern conventional computers (Von Neumann architecture), and to build some working systems to perform these tasks based on the state-of-the-art hardware (digital or analog) technology.

Neural networks offer some information processing advantages characterized by their robustness, adaptivity, self-organization, distributed and massive parallelism, and ease of implementation in hardware. In comparison to rule-based systems or knowledge-based systems, neural networks are claimed to be capable of automatically building the internal representation of knowledge from examples. Therefore, they can be used in situations where the domain knowledge is neither available nor complete, or the underlying problem is not fully understood (e.g., [155, 166]).

1.3 Common Links Between Neural Networks and Statistical Pattern Recognition

Machine pattern recognition techniques attempt to endow modern computers some of the recognition capabilities that humans possess. Some of the early work in pattern recognition, such as the work on perceptrons [149], was biologically motivated. Such approaches were not very successful in solving difficult recognition problems. As a result, statistical pattern recognition became the model of choice for designing recognition systems. The designers of pattern recognition systems are more concerned with the efficiency and performance issues rather than the biological plausibility of the underlying approaches. Interestingly, we have observed that many newly developed neural network models have also been influenced by this philosophy.

Neural networks and statistical pattern recognition are two closely related disciplines which share several common design problems. In the neural network community, pattern recognition is considered as one of the challenging problems. On the other hand, in the pattern recognition community, neural networks are viewed as a powerful complement to the well-known recognition approaches: statistical and structural. It is difficult to estimate the amount of overlap between neural networks and statistical pattern recognition, because it requires knowing the precise boundary between the disciplines. Werbos [177] estimated that about 80% of the work being done with neural networks deals with pattern recognition. Some links between the two disciplines have been established [156]. In the following sections, we will address the common links between neural networks and statistical pattern recognition according to different components of recognition systems and design issues.

1.3.1 Representation

Representation addresses the problem of how an input pattern is represented in a recognition system. How do we design a good representation scheme? A good pattern representation should satisfy at least the following requirements: (i) high data compression rate, (ii) good discriminatory power/information preservation, (iii) desired invariance, and (iv) good robustness to noise. In most statistical pattern recognition systems, representation schemes are often developed by the designers using their understanding of the underlying problem and the domain knowledge. These representation schemes are fixed once the systems have been built. Similarly, in many applications using multilayer feedforward networks, the extracted features are fed into a network. So, the network essentially performs the classification task. However, the feedforward networks have a desirable property of automatically building internal representation of input patterns (feature extraction), although it is sometimes difficult to interpret these representations. Due to this property, some researchers feed the raw data (e.g., pixels) into the network and let the network learn a representation from the training patterns (e.g., [105, 155, 83]). For example, Jain and Karu [83] used a feedforward network to learn texture discrimination masks directly from the input image array.

Gallinari et al. [53] have established a link between linear discriminant analysis and linear feedforward networks. Their theoretical analysis shows that in a linear two-layer feedforward network trained to minimize the square-error criterion, the role of the hidden layer with $rank(\Sigma_B)$ units is to realize a linear discriminant analysis that maximizes the criterion $|\Sigma_{B_h}|/|\Sigma_{T_h}|$, where Σ_B is the between-class covariance matrix in the input space, and Σ_{B_h} and Σ_{T_h} are the between-class covariance and total-class covariance matrices, respectively, in the space spanned by the hidden units.

Webb and Lowe [173] have investigated a class of multilayer feedforward networks

with nonlinear hidden units and linear output units, trained to minimize the squared error between the desired output and actual output of the network. They have found that the nonlinear transformation implemented by the subnetwork from the input layer to the final hidden layer of such networks maximizes the so-called network discriminant function, $\mathrm{Tr}\{S_B S_T^+\}$, where S_T^+ is the pseudo-inverse of the total scatter matrix of the patterns at the output of the final hidden layer, and S_B is the weighted between-class scatter matrix of the output of the final hidden layer (the weights are determined by the coding scheme of the desired outputs). The role of hidden layers is to implement a nonlinear transformation which projects input patterns from the original space to a space in which patterns are easily separated by the output layer.

A good representation can make the decision making process and network design much simpler, and lead to a good generalization. However, designing a good representation scheme requires a thorough understanding of the underlying problem and substantial domain knowledge. How to learn the representation scheme from given data remains an open research issue.

1.3.2 Feature Extraction and Data Projection

Feature extraction has a two-fold meaning. It is often referred to as the process of extracting some numerical measurements from the raw input patterns (initial representation). On the other hand, it is also defined as the process of forming a new (often smaller) set of features (refined representation) from the original feature set. However, both the processes can be considered as a mapping (or projection) from a d-dimensional input space to an m-dimensional output space ($m < d$).

A large number of artificial neural networks and learning algorithms have also been proposed for feature extraction and data projection [113]. These research efforts can be loosely classified into two groups. The first group contains new neural network designs, or existing neural network models for feature extraction and data projection.

Examples of this type of networks include Kohonen's self-organizing maps [93], the nonlinear projection methods (NP-SOM) [100], and nonlinear discriminant analysis (NDA) based on the functionality of hidden units in feedforward network classifiers [173, 111]. These networks exhibit some nice properties and can be used as new tools or supplementary approaches to classical feature extraction and data projection algorithms.

A second line of research has investigated the properties of neural networks and learning rules to establish links between classical approaches and neural networks. As a result, it has been found that some of these neural networks actually perform many of the well-known feature extraction and data projection algorithms [61, 132, 53]. Several classical feature extraction and data projection approaches have been implemented using neural network architectures, e.g., PCA networks (see [63, 69]), LDA networks [111, 102], and network for Sammon's projection (SAMANN) [85]. Although such efforts do not necessarily provide new approaches to feature extraction and data projection (from the view point of functionality performed by the networks), the resulting networks do possess following advantages over traditional approaches: (i) most learning algorithms and neural networks are adaptive in nature, thus they are well-suited for many real environments where adaptive systems are required, (ii) for real-time implementation, neural networks provide good, if not the best, architectures which are relatively easily implemented using current VLSI and optical technologies, and (iii) neural network implementations can overcome the drawbacks inherent in the classical algorithms. For example, Jain and Mao [85] proposed a neural network design for Sammon's nonlinear projection algorithm which offers the generalization ability of projecting new data, a property not present in the original algorithm.

1.3.3 Classification

In statistical pattern recognition, the goal is to assign a d-dimensional input pattern $\mathbf{x}$ to one of c classes, $\omega_1, \omega_2, \cdots, \omega_c$. If the class-conditional densities, $p(\mathbf{x}|\omega_i)$, and *a priori* class probabilities $p(\omega_i)$, $i = 1, 2, \cdots, c$, are completely known (the ideal case in Fig. 1.1 where we have infinite number of labeled training samples), the "optimal" strategy for making the assignment is the Bayes decision rule [34]: Assign pattern $\mathbf{x}$ into class ω_k if

$$p(\omega_k|\mathbf{x}) > p(\omega_i|\mathbf{x}), \ \forall i = 1, 2, \cdots, c; i \neq k, \tag{1.1}$$

where $p(\omega_k|\mathbf{x}) = p(\omega_k)p(\mathbf{x}|\omega_k)/p(\mathbf{x})$ is called the *a posteriori* probability of class ω_k, given the pattern vector $\mathbf{x}$. The rule minimizes the probability of error. The resulting "optimal" Bayes error also provides a lower bound for studying the asymptotic behavior of other classifiers.

In statistical pattern recognition, a decision rule can often be formulated as a set of c *discriminant functions*, $g_i(\mathbf{x})$, $i = 1, 2, \cdots, c$. We assign $\mathbf{x}$ to class ω_j if $g_j(\mathbf{x}) > g_i(\mathbf{x}), \forall i \neq j$. For the 2-class problem, only a single discriminant function is needed. Feedforward networks such as multilayer perceptrons and radial basis function networks also use this familiar notion. Each output node implements a special discriminant function, and the class membership of the input pattern is determined by a "winner-take-all" scheme, which is the "max" operation.

There are two methods to construct the discriminant functions in statistical pattern recognition. We can first estimate the class-conditional density functions and then use these densities to form the discriminant functions. The second method is to specify the form of discriminant functions and then learn the coefficients from training patterns. The neural networks take the second approach. In the following, we will discuss different classifiers in both statistical pattern recognition and neural networks (parametric versus nonparametric, on-shot versus tree classifiers).

Parametric Classifiers

If we have a finite set of labeled training patterns and the form of density functions is known (more often it is assumed), we can use either the "optimal" rule or the "plug-in" rule (see Fig. 1.1). In "optimal" decision rule, a Bayesian approach involving an *a priori* density on the unknown parameters is used. Since this approach is cumbersome, a simpler approach where parameter estimates are used to replace the unknown parameters is preferred. If the class-conditional densities are assumed to be multivariate Gaussian, then the "optimal" discriminant function is quadratic in $\mathbf{x}$. Further, if the covariance matrices of the pattern classes are equal, then the "optimal" discriminant function reduces to a linear form. The well-known perceptron can implement a linear decision boundary. Therefore, the perceptron classifier is optimal in the Bayes sense if the class-conditional densities of the pattern classes are Gaussian with equal covariance matrices. Several polynomial networks have also been proposed (e.g., [136, 164]).

We should be aware of the underlying assumptions which guarantee the optimality of a classifier. If the assumptions are not valid, parametric classifiers may perform very poorly. Yau and Manry [186] mapped the Gaussian classifier (quadratic under Gaussian assumption) into a Gaussian isomorphic network which can be trained by the backpropagation algorithm. They found that this mapping can improve the performance when the Gaussian assumption is violated.

Non-parametric Classifiers

In many classification problems, we do not even know the form of the class-conditional density functions. In such situations, the designer of a statistical pattern recognition system either estimates the class-conditional densities from the finite training set, or designs nonparametric classifiers such as k-nearest neighbor classifiers. The two most well-known methods for estimating the class-conditional densities are based on Parzen windows and k-nearest neighbors [34].

A number of neural networks have been proposed which are closely related to the Parzen window classifier. These networks essentially estimate class-conditional densities and perform classification based on them. Some examples of these networks are the probabilistic neural network (PNN) [164] and the Cerebellar Mode Arithmetic Computer (CMAC) [119]. The PNN network uses a Gaussian window, while the CMAC model uses a rectangular window. Another approach to the Parzen window classifier is the Radial Basis Function (RBF) network [123], which is a two-layer network. Each node in the hidden layer employs a radial basis function, such as a Gaussian kernel, as the activation function. Each output node implements a linear combination of these radial basis functions. Unlike in the Parzen window classifier, the number of kernels in the RBF network is usually less than the number of training patterns. Furthermore, not only the width of kernels but also the position of these kernels (windows) are learned from training patterns.

The Parzen window classifier and the related neural network models share another important design issue: window width selection. The choice of the window width, h, is very critical to Parzen density estimation [35, 164, 119, 123]. Too small a value of h would give a spiky or noisy estimate of $p(\mathbf{x}|\omega_i)$ whereas large values of h result in an oversmoothed estimate of $p(\mathbf{x}|\omega_i)$. In statistical pattern recognition literature, it is common to use a value of h based on the following relationship: $h \propto n_i^{-k/d}$, where k is a constant ($0 < k < 0.5$), and n_i is the number of training samples from class ω_i. This choice of h has been shown to be optimal by Silverman [160] and Fukunaga and Hostetler [47] for Gaussian class-conditional densities. Jain and Ramaswami [87] used the bootstrap technique in selecting the optimal width. These techniques would also be useful for the neural network models.

A number of stochastic network models have been proposed to learn the underlying probability distributions for the given set of training patterns. The well-known example is the Boltzmann machine [65]. The probability of the network states can

be trained to comply with the desired probability distribution (class-conditional or *a posteriori* densities). In a comparative study, Kohonen et al. [96] found that the Boltzmann machine achieved considerably better accuracy than a feedforward network trained using the backpropagation algorithm, and its performance was close to the optimal Bayes rule. However, the basic Boltzmann learning algorithm employs a Monte Carlo simulation and is extremely slow, which greatly limits the application of the Boltzmann machine to real-world problems. A number of variations of Boltzmann learning algorithm have been proposed [63] including the deterministic Boltzmann machine [137] and the Boltzmann Perceptron Network [185].

It has been shown by many researchers, both theoretically and empirically, that networks with the *1-of-C* coding of the desired outputs, such as the multilayer perceptron with sigmoidal nonlinearities, radial basis function networks, and high-order polynomial networks, trained using the squared-error, cross-entropy, and normalized-likelihood cost functions can produce outputs which estimate *a posteriori* probabilities (e.g., [146]). The estimation accuracy depends on the network complexity, the amount of training data, the degree to which training data reflect the true class-conditional densities and *a priori* probabilities. These results have shown that feedforward networks have the desirable property of providing both the discriminant functions (decision boundaries) and *a posteriori* probabilities which can be used as a confidence measure of the decision making process. This property also makes it easier to combine outputs of multiple networks to form a high-level decision, and to determine a meaningful rejection threshold.

Instead of explicitly estimating the class-conditional densities or *a posteriori* probabilities, the K-nearest neighbor (K-NN) classifier is a nonparametric classifier that directly constructs the decision rule from the training data. The K-NN rule classifies a pattern $\mathbf{x}$ into the class which is most frequently represented among its K nearest neighbors. The choice of K is data dependent. The rule of thumb is that K should

be a fraction (e.g., $\sqrt{n}$) of the number, n, of training samples. Several variants of the K-NN classifier are available (e.g., condensed and edited), which reduce its time complexity and memory requirements [30]. The K-NN classifier can be easily mapped into a neural network architecture [84].

One can argue whether the feedforward networks belong to the class of parametric or nonparametric classifiers. Since a feedforward network of fixed size explicitly specifies the form of the discriminant functions, it implicitly assumes the forms of class-conditional densities which lead to the specified discriminant functions. In this sense, feedforward networks are parametric classifiers. On the other hand, if the network size is determined from data or the network is powerful enough to approximate a large number of density and discriminant functions, we can say that feedforward networks are nonparametric classifiers.

Tree Classifiers

In a tree classifier, a terminal decision is reached through a sequence of intermediate decisions (nonterminal nodes) following a particular path from the root to a leaf node; the remaining nodes in the tree need not be visited. In some situations, e.g., when pairs of classes can be separated by only one or a few components of the feature vector, tree classifiers can be much more efficient than single-stage classifiers. Moreover, tree classifiers may have a better small training sample size performance than one-shot classifiers because only a small number of features is used for decision making at each node. Note that the decision rule used at each node of the tree can be either parametric or nonparametric.

Sethi [157] investigated the link between tree classifiers and multilayer feedforward networks, and found that a decision tree can be restructured as a three-layer feedforward network. This restructuring provided a different tool for network design and training. It also solved the "credit-assignment" problem in training feedforward networks thus making it possible to progressively train each layer separately. The

backpropagation algorithm can be used to refine the weights in the network after the initial restructuring.

There is not sufficient theoretical basis for making a conclusion on whether a multilayer feedforward network or a tree classifier performs better [4]; the performance is application-dependent. If the architecture of a feedforward network is carefully designed, then it can have a more compact structure than the tree classifier.

1.3.4 Clustering

Various clustering or unsupervised learning algorithms proposed in the literature [81] can be grouped into the following two categories: hierarchical and partitional. A hierarchical clustering is a nested sequence of partitions, whereas a partitional clustering is a single partition. A partition can be either hard or soft (e.g., fuzzy clustering [15]). Commonly used partitional clustering algorithms are the k-means algorithm and its variants (e.g., ISODATA [7]) which incorporate some heuristics for merging and splitting clusters and for handling outliers [81]. Competitive neural networks are often used to cluster input data. The well-known examples are Kohonen's Learning Vector Quantization (LVQ) and self-organizing map [93], and ART (Adaptive Resonance Theory) models [21, 22, 23]. Despite their attractive architectures and biological and neurophysiological plausibility, the updating procedures used in these neural network-based clustering are quite similar to some classical partitional clustering approaches. For example, the relationship between the sequential k-means algorithm and Kohonen's learning vector quantization (LVQ) has been addressed by Pal et al. [135]. The learning algorithm in ART models has been shown to be similar to the sequential leader clustering algorithm [58, 127].

Many partitional clustering algorithms [81] and competitive neural networks for unsupervised learning [63] are suitable primarily for detecting hyperspherical-shaped clusters, because the Euclidean distance metric is commonly used to compute the

distance between a pattern and the assigned cluster center. Other distance metrics, such as the Mahalanobis distance, can also be used. Mao and Jain [114] (see Chapter 4) have proposed a network for detecting hyperellipsoidal clusters (HEC), which uses the regularized squared Mahalanobis distance: In spite of the numerous research efforts, data clustering remains a difficult and essentially an unsolved problem [81].

1.3.5 "Curse of Dimensionality" and Generalization Issues

It is well-known that the error rate of the optimal Bayes classifier will not increase as the number of features is increased [34]. However, in a finite training sample size case, the additional discriminatory information provided by the new features can be outweighed by the increase in the inaccuracy of the parameter estimates needed in the classification rule. Thus, in practical classifiers, a "peaking" phenomenon is often observed: addition of new features decreases the classification error at first, then the error levels off, and finally it begins to increase [80, 36]. This phenomenon, often referred to as the "curse of dimensionality", is also observed in neural networks for both classification [142] and regression [55]. Many important results related to the curse of dimensionality have been published in the statistical pattern recognition literature. We summarize here some of these results.

Duin [36] derived a bound on the expected error of a classifier which uses the estimated class-conditional density functions, $\hat{p}_i(\mathbf{x})$, instead of the true class-conditional density functions, $p_i(\mathbf{x})$, $i = 1, 2$:

$$E\{\varepsilon\} \leq \varepsilon_{Bayes} + \frac{1}{2} E\left\{ \sum_{i=1}^{2} \int |p_i(\mathbf{x}) - \hat{p}_i(\mathbf{x})| d\mathbf{x} \right\}, \tag{1.2}$$

where ε_{Bayes} is the optimal Bayes error. The second term on the right-hand side is

basically the expected Kolmogorov distance between the true and estimated class-conditional densities. Simulations have shown that for Gaussian densities, the addition of new features may increase this expected Kolmogorov distance. However, evaluation of this bound is difficult because we never know the true densities in practice.

Jain and Waller [88] investigated a 2-class problem involving normal distributions with different class means but a common covariance matrix. They found that when the "plug-in" decision rule is used, the "peaking" phenomenon can be avoided if the addition of a new feature results in an increase in the Mahalanobis distance between the two classes by at least $1/(n-d-3)$ percent, where n is the number of training patterns and d is the number of features.

Raudys and Jain [143] showed that for the linear and quadratic discriminant functions which are obtained from the estimated mean vectors and covariance matrices, the expected classification error is given by $E\{\hat{\varepsilon}\} = \varepsilon + \frac{v}{n}$, where ε is the asymptotic error of the linear or quadratic classifier, and v is a constant which is proportional to d for the linear classifier and d^2 for the quadratic classifier. This relationship implies that the number of training patterns should increase linearly according to the number of free parameters in the classifier. This theoretical result is consistent with intensive computer simulations [143, 80]. A general guideline of having five to ten times as many training patterns as free parameters seems to be a good practice to follow [80].

The generalization ability of both statistical and network-based classifiers depends on the following three factors: (i) the number of training patterns, (ii) the underlying class-conditional distributions, and (iii) the discrimination ability (or complexity) of the classifier. The theoretical and empirical results on curse of dimensionality discussed above were obtained under the assumption of Gaussian class-conditional distributions. For the general case, the exact relationship between these three factors

becomes intractable. Vapnik's theory [170] provides a bound on the maximum deviation of the estimated error from the true error over a set of classifiers. The theory was developed based on the notion of Vapnik-Chervonenkis (VC) dimension (or capacity), which is defined as the maximum number (V) of data points in the space for which the classifier can implement all possible 2^V dichotomies. For example, the VC dimension of a linear classifier in R^d is $(d+1)$, and the VC dimension of a quadratic classifier is $\{2d+d(d-1)/2+1\}$. However, computing the VC dimension for a general classifier is very difficult. Baum and Haussler [14] have shown that the VC dimension of a one-hidden-layer feedforward network with threshold units and full connections between adjacent layers is bounded by $2\lfloor H/2 \rfloor d \leq V \leq 2W \log(eN)$, where e is the base of the natural logarithm, H, N and W are the number of hidden units, the total number of units and the total number of weights in the network, respectively. Note that the upper bound on the VC dimension is linear in W (assuming that N is fixed). The bounds on the VC dimension of feedforward networks with continuous nondecreasing activation function are more complicated [60], but are still roughly linear in W.

Once the VC dimension of a classifier is known, a lower bound can be established on the (training) sample size requirement [17]:

$$n \geq \max\left\{\frac{8}{\chi^2}\ln\frac{8}{\delta}, \frac{16V}{\chi^2}\ln\frac{16}{\chi^2}\right\}, \tag{1.3}$$

where χ and δ are two parameters which determine the degree of the maximum deviation of the estimated error from the true error, and the confidence on the maximum deviation, respectively. Kraaijveld [98] has derived a slightly better bound. Note that the sample size requirement is roughly linearly proportional to the VC dimension of the classifier. For a feedforward network classifier, the number of training patterns

should be approximately linearly proportional to the number of weights in the network. These results derived from Vapnik's theory have the following interesting and unique properties: (i) any classifier (or system), no matter how complicated, is characterized by a single number, i.e., the VC dimension; (ii) bounds on training sample size and on the maximum deviation of the true and estimated errors are distribution-free; (iii) bounds are independent of how the classifier is trained; and (iv) Vapnik's theory only captures the difference between the true and estimated errors, and provides no statement about the value of the true error itself.

In some sense, properties (ii) and (iii) are desirable, but these properties also make the bounds too conservative because now the worst case behavior must be considered. Therefore, the practical applicability of these bounds is questionable. For example, these bounds provide very little guidance even in the case of multivariate Gaussian distributions. Moreover, the capacity (or VC dimension) of a classifier may not be fully exploited by the learning algorithm, or may be restricted by regularization techniques. This means that the *effective capacity* of a classifier can be much smaller than its true capacity. Kraaijveld [98] studied the effect of backpropagation algorithm on the generalization properties of multilayer feedforward networks and found that the search capability of the backpropagation algorithm (or any other iterative algorithm) can not fully exploit the theoretical capacity of multilayer feedforward networks. Kraaijveld proposed a simple but computationally intensive procedure to estimate the effective capacity of a classifier. His simulation showed that the effective capacity can be orders of magnitude smaller than the bound for the true capacity of a multilayer feedforward network. We recommend that the number of training patterns should be at least five to ten times as large as the VC dimension of a classifier [80, 182].

In many practical situations, collecting a large number of training samples is expensive, time consuming, and sometimes impossible. Therefore, choosing a parsimonious

system (with a small number of parameters) is a very important issue. Various efforts have been made to improve the generalization ability of neural networks. This problem is similar to selecting the best classifier from a given set of classifiers in statistical pattern recognition [143].

The techniques which attempt to improve the generalization ability of a network can be grouped into the following four categories: (i) local connections and weight sharing, (ii) adaptive network pruning, (iii) adaptive network growing, and (iv) regularization. The main idea behind these techniques can be related to the principle of minimum description length [148] and the intuition of Occam's Razor. Methods in the first category try to reduce the number of free parameters (connection weights) by connecting a node to only a local region of its inputs or by sharing the same weight among many connections [106, 104]. Kraaijveld [98] has derived the upper bound on the VC dimension of weight-sharing networks.

1.4 Contributions of This Thesis

The contributions of this thesis are made to both the fields of pattern recognition and artificial neural networks. These contributions are summarized as follows.

1. Several links between statistical pattern recognition and neural networks have been established [86] (Chapter 1). These links can encourage communication between researchers in the two fields to inspire each other and to avoid repetitious work.

2. A number of neural networks and learning algorithms have been proposed. These include:

   - Linear Discriminant Analysis (LDA) network [111] (Chapter 2);
   - Nonlinear Discriminant Analysis (NDA) network [111] (Chapter 2);

- A network for Sammon's nonlinear projection (SAMANN) [85] (Chapter 2);
- A Nonlinear Projection algorithm based on Kohonen's Self-Organizing Maps (NP-SOM) [100] (Chapter 2);
- A network-based k-nearest neighbor classifier [84] (Chapter 3); and
- A self-organizing network for detecting HyperEllipsoidal Clusters (HEC) [114] (Chapter 4).

A comparative study of five typical networks for feature extraction and data projection has also been conducted [113] (Chapter 2).

3. The main theoretical contributions of this thesis are:

   - Vapnik's theory and Moody's approach for studying the generalization ability of feedforward networks have been compared. Moody's notion of the *effective* number of parameters has been extended to a more general noise model. We have shown that the addition of noise in both sampling points in test data and observations increases the deviation of the expected test set MSE from the expected training set MSE, and also increases the effective number of parameters. Monte Carlo experiments have been conducted to verify Moody's result and our extension, and to demonstrate the role of the weight-decay regularization in improving the generalization ability of feedforward networks. See Chapter 5.
   - We have provided a systematic study of regularization techniques in neural network design. We present a taxonomy for various forms of regularization and demonstrate that a variety of neural networks and learning algorithms do fit into this framework. Some results discussed and developed in Chapter 5 are used to explain a counter-intuitive phenomenon that a network with

a large number of free parameters and trained with a relative small number of patterns can still generalize well. See Chapter 6 and [112].

4. A practical method of node pruning has been proposed to design minimal networks which generalize well. This method can also be used for feature selection. See Chapter 7 and [115].

1.5 Organization of This Thesis

The rest of this thesis is organized as follows. The first part of this thesis (Chapters 2-4) focuses on designing neural networks for pattern recognition. Chapter 2 proposes a number of neural networks and learning algorithms for feature extraction and multivariate data projection. This chapter also conducts a comparative study of five typical networks for feature extraction and multivariate data projection. Chapter 3 implements the well-known k-nearest neighbor classifier and its variants using neural network architecture. Chapter 4 presents a self-organizing network for detecting hyperellipsoidal clusters.

The second part of this thesis (Chapters 5-7) is devoted to the theoretical analysis of generalization ability and practical techniques for improving the generalization ability of feedforward networks. Chapter 5 presents a theoretical study of the generalization ability of feedforward networks. It first introduces two types of frameworks for this study: Vapnik's theory and Moody's approach. The advantages and disadvantages of these two frameworks are analyzed. Then, we extend Moody's model to a more general setting. Monte Carlo experiments are conducted to verify both Moody's result and our extension, and to demonstrate the role of the weight-decay regularization in reducing the effective number of parameters in feedforward networks. We also summarize four different types of practical techniques for improving the generalization ability of feedforward networks. Chapters 6 and 7 address two of these four

types of techniques. Chapter 6 provides a survey of regularization techniques used in neural networks. Some theoretical results in Chapter 5 are used to explain the role of regularization techniques in the performance of neural networks. In Chapter 7, a node pruning method is proposed for designing minimal networks which generalize well, and for feature selection.

Finally, Chapter 8 summarizes this thesis and mentions topics for future research work.

CHAPTER 2

Neural Networks for Feature Extraction and Data Projection

Classical feature extraction and data projection methods have been well-studied in the statistical pattern recognition and exploratory data analysis literature. In this chapter, we propose a number of networks and learning algorithms which provide new or alternative tools for feature extraction and data projection. These networks include a network (SAMANN) for Sammon's nonlinear projection, a linear discriminant analysis (LDA) network, a nonlinear discriminant analysis (NDA) network, and a network for nonlinear projection (NP-SOM) based on Kohonen's self organizing map. A common attribute of these networks is that they all employ adaptive learning algorithms which makes them suitable in some environments where the distribution of patterns in feature space changes with respect to time. The availability of these networks also facilitates hardware implementation of well-known classical feature extraction and projection approaches. Moreover, the SAMANN network offers the generalization ability of projecting new data, which is not present in the original Sammon's projection algorithm; the NDA method and NP-SOM network provide new powerful approaches for visualizing high dimensional data. We evaluate five representative

neural networks for feature extraction and data projection based on a visual judgement of the 2-dimensional projection maps and three quantitative criteria on eight data sets with various properties. Our conclusions which are based on analysis and simulations can be used as a guideline for choosing a proper method for a specific application.

2.1 Feature Extraction and Data Projection

Feature extraction and data projection can be formulated as a mapping Ψ from a d-dimensional input space to an m-dimensional output space (map space), $\Psi : R^d \rightarrow R^m$, $m \leq d$, such that some criterion, $\mathcal{C}$, is optimized. This formulation is similar to a function approximation problem. However, unlike the function approximation problem where the mapping function is estimated from training patterns which are input-output pairs (desired outputs are known), in feature extraction and data projection the desired outputs are often not available even for supervised approaches (e.g., linear discriminant analysis). Feature selection, which is used for choosing an optimal subset of features, can be viewed as a special case of feature extraction, where Ψ is a linear $m \times d$ permutation matrix. For data visualization purpose, the value of m is usually set to 2 or 3.

A large number of approaches for feature extraction and data projection are available in the pattern recognition literature [16, 81]. These approaches differ from each other in the characteristics of the mapping function Ψ, how Ψ is learned, and what optimization criterion $\mathcal{C}$ is used. The mapping function can be either linear or nonlinear, and can be learned through either supervised or unsupervised methods. The combination of these two factors results in the following four categories of feature extraction and data projection: (i) unsupervised linear, (ii) supervised linear, (iii) unsupervised nonlinear, and (iv) supervised nonlinear. Within each category, the methods differ in

the criterion function C being used. This taxonomy is shown in Figure 2.1. In general, linear methods are attractive because they require less computation than nonlinear methods. Analytical solution is often available for linear methods. On the other hand, nonlinear methods are more powerful than linear methods. However, since the analytical solution is often not available, a numerical optimization method must be used to obtain the solution, which is usually computationally demanding. Furthermore, the optimization procedure often gets trapped into a local minimum. It is also generally true that supervised methods have better performance than unsupervised methods in situations where the category information is available [16].

Various neural networks including those proposed in this chapter for feature extraction and data projection can also be grouped into the above four categories. The boxes at the leaf nodes in Figure 2.1 list a number of representative networks in each category which will be described and evaluated in this chapter.

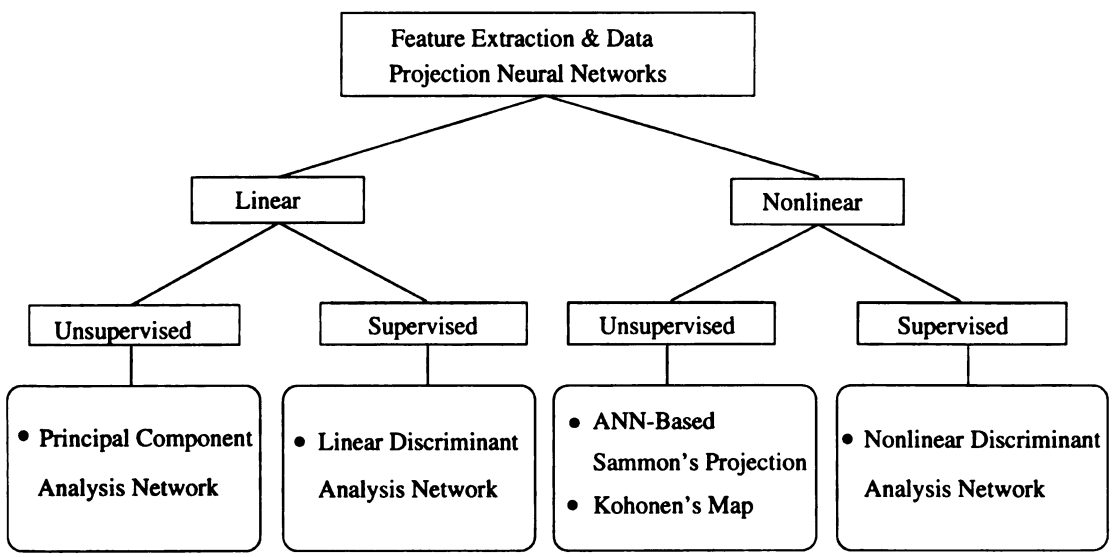


Figure 2.1. A taxonomy of neural networks for feature extraction and data projection.

Choice of a proper feature extraction and data projection method (both the traditional and neural network approaches) for a given problem is driven by (i) the available information (supervised versus unsupervised), (ii) *a priori* knowledge about the underlying distribution of the data (linear versus nonlinear, and C), and (iii) the task requirement (classification or visualization). Knowing the details and the characteristics of all the available approaches is crucial to making a good choice. In this chapter, we will investigate the characteristics of a number of neural networks through analysis and evaluation experiments.

2.1.1 Principal Component Analysis (PCA) Networks

Principal component analysis, also called Karhunen-Loeve transform, is a well-known statistical method for feature extraction, data compression and multivariate data projection, and has been widely used in communication, signal and image processing, pattern recognition and data analysis. It is a linear orthogonal transform from a d-dimensional input space to an m-dimensional output space, $m \leq d$, such that the coordinates of the data in the new m-dimensional space are uncorrelated and maximal amount of variance of the original data is preserved by only a small number of coordinates.

Projection pursuit is a more general approach in this category which includes principal component analysis as a special case. Projection pursuit was first introduced by Friedman and Tukey [44] as a technique for exploratory analysis of multivariate data sets. Huber [73] provided a unified framework of projection pursuit which included principal component analysis, multidimensional scaling, factor analysis, nonparametric regression (projection pursuit regression), density estimation (projection pursuit density estimation) and deconvolution. The standard 2-layer (one hidden layer) feedforward network is closely related to projection pursuit regression in approximating functions although the backpropagation algorithm used for training the feedforward

network is different from the algorithms used in projection pursuit regression. A number of researchers have explored using projection pursuit regression and density estimation in training or constructing feedforward networks [13, 75, 187].

Let $\boldsymbol{\xi_k} = (\xi_{k1}, \xi_{k2}, \cdots, \xi_{kd})^T$ denote the k^{th} d-dimensional input vector, $k = 1, 2, \cdots, n$. Assume zero-mean vectors without a loss of generality. Let Σ be the covariance matrix of the data, with eigenvalues, $\lambda_1, \lambda_2, \cdots, \lambda_d$, in the decreasing order. The m principal components of the data can be obtained by a linear transform $\mathbf{o_k} = \Phi^T \boldsymbol{\xi_k}$, where $\mathbf{o_k} = (o_{k1}, o_{k2}, \cdots, o_{km})^T$, and Φ is a $d \times m$ matrix whose columns are m eigenvectors corresponding to the m largest eigenvalues of the covariance matrix Σ. It is easy to show that the covariance matrix of the transformed data is a diagonal matrix, $diag\{\lambda_1, \lambda_2, \cdots, \lambda_m\}$, which means that the components of the transformed data are uncorrelated. The variance of the original data retained in the new m-dimensional space is $\sum_{i=1}^{m} \lambda_i$, which is the largest retained value among all linear orthogonal transforms of the same output dimensionality. Principal component analysis is optimal in the mean-square error sense among all linear orthogonal transforms.

A large number of neural networks and learning algorithms have been proposed for principal component analysis [1, 133, 151, 150, 154, 6, 40, 69, 101, 103]. We have extremely used the PCA network proposed by Rubner et al. [151, 150].

Rubner et al.'s PCA network architecture is shown in Figure 2.2. It consists of d input nodes and m output units. Assume $m = d$ without any loss of generality. Each input node i is connected to each output unit j with connection strength w_{ij}. We denote the weight vector associated with output unit j from the d inputs by $\mathbf{w}_j = (w_{1j}, w_{2j}, \cdots, w_{dj})^T$. All the output units are hierarchically organized in such a way that the output unit i is connected to the output unit j with connection strength u_{ij} if and only if $i < j$. Let $\{\boldsymbol{\xi_k} \mid k = 1, 2, \cdots, n\}$ be the set of n input vectors with zero-mean, and $\{\boldsymbol{o_k} \mid k = 1, 2, \cdots, n\}$ be their corresponding output vectors produced

by the network.

$$o_{kj} = \mathbf{w_j} \cdot \boldsymbol{\xi}_k + \sum_{l<j} u_{lj} o_{kl}. \tag{2.1}$$

The weights on connections between the input nodes and the output layer are adjusted upon presentation of an input vector $\boldsymbol{\xi}_k = (\xi_{k1}, \xi_{k2}, \cdots, \xi_{kd})$ according to the Hebbian rule, i.e.

$$\Delta w_{ij} = \eta \xi_{ki} o_{kj}, \quad i, j = 1, 2, \cdots, d, \tag{2.2}$$

where η is the learning rate. The lateral synaptic weights, on the other hand, are updated according to the anti-Hebbian rule:

$$\Delta u_{lj} = -\mu o_{kl} o_{kj}, \quad l < j, \tag{2.3}$$

where μ is another positive learning parameter.

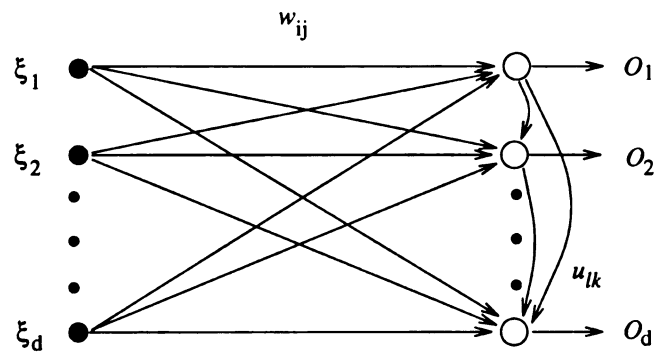


Figure 2.2. PCA network proposed by Rubner et al.

Rubner and Tavan [151] proved that if the learning parameters η and μ are properly chosen according to

$$\frac{\eta(\lambda_1 - \lambda_d)}{\lambda_1(1 + \eta\lambda_d)} < \mu < 2/\lambda_1, \tag{2.4}$$

then all the lateral weights will rapidly vanish and the network will converge to a state in which the d weight vectors associated with the d output units are the d eigenvectors of the covariance matrix of input patterns with eigenvalues, $\lambda_1 \geq \lambda_2 \cdots \geq \lambda_d$. Although, in practice, it is difficult to determine the values of η and μ according to the inequalities in Eq. (2.4) without first computing the eigenvalues, Eq. (2.4) does provide a range for the values of η and μ if λ_1 and λ_d can be somehow estimated. The asymptotically vanishing connection strengths also provide a stopping criterion for the learning procedure.

We have found that introducing the momentum term and letting the learning rate and momentum decay with time can speed up the convergence. In our experiments, we use

$$\Delta w_{ij}(t+1) = \eta(t)\xi_{ki}o_{kj} + \beta(t)\Delta w_{ij}(t), \tag{2.5}$$

and

$$\Delta u_{lj}(t+1) = -\mu(t)o_{kl}o_{kj} + \beta(t)\Delta u_{lj}(t), \tag{2.6}$$

where $\eta(t+1) = max(\alpha\eta(t), 0.001)$, $\mu(t+1) = max(\alpha\mu(t), 0.002)$, $\beta(t+1) = max(\alpha\beta(t), 0.001)$, t is the iteration index and α is the decay factor. The non-zero minimum values for η, μ and β enable the network to retain the plasticity so that it can adapt itself to the new input data.

We summarize the learning algorithm for principal component analysis [151, 150]

in Box 1.

Box 1

PCA Algorithm

1. Initialize all connection weights to small random values and choose the values of the learning parameters.
2. Randomly select a d-dimensional pattern and present it to the network.
3. Update the connection weights between the input nodes and the output layer according to the Hebbian rule, Eq. (2.5).
4. Normalize each weight vector to a unit vector.
5. Update the lateral weights according to the anti-Hebbian rule, Eq. (2.6).
6. Modify parameters, η, μ and β.
7. If all the lateral weights are sufficiently small for a given number of presentations (their absolute values are below some threshold), then stop; else go to step 2.

Our computer simulations of Runber's PCA network have shown that the precision of the computed eigenvalues and eigenvectors is very high (of the order of 10^{-5} compared to values obtained by the commercial Eispack software) [111]. However, we have experienced a very slow convergence of the PCA algorithm when the input dimensionality d is high and all the d eigenvectors need to be computed, especially when some eigenvalues are very small.

2.1.2 Linear Discriminant Analysis (LDA) Network

If the category information of patterns is known, then it is more appropriate to use supervised learning. Linear Discriminant Analysis incorporates the *a priori* category

information into the projection or transform by maximizing the between-class scatter while holding the within-class scatter constant.

Let $\boldsymbol{\xi}_i^{(l)} = (\xi_{i1}^{(l)}, \xi_{i2}^{(l)}, \cdots, \xi_{id}^{(l)})^T$ denote the i^{th} pattern in class l, $i = 1, 2, \cdots, n_l$, $l = 1, 2, \cdots, c$, where c is the number of categories. Let $n = \sum_{l=1}^{c} n_l$ denote the total number of patterns. Define the within-class covariance matrix, Σ_W, as

$$\Sigma_W = \frac{1}{n} \sum_{l=1}^{c} \sum_{i=1}^{n_l} (\boldsymbol{\xi}_i^{(l)} - \mathbf{m}^{(l)})(\boldsymbol{\xi}_i^{(l)} - \mathbf{m}^{(l)})^T, \tag{2.7}$$

where $\mathbf{m}^{(l)}$ is the mean vector of class l, $l = 1, 2, \cdots, c$. Similarly, define the between-class covariance matrix, Σ_B, as

$$\Sigma_B = \frac{1}{n} \sum_{l=1}^{c} n_l (\mathbf{m}^{(l)} - \mathbf{m})(\mathbf{m}^{(l)} - \mathbf{m})^T, \tag{2.8}$$

where $\mathbf{m}$ is the mean vector of the pooled data. The total scatter matrix is, therefore,

$$\Sigma_T = \frac{1}{n} \sum_{l=1}^{c} \sum_{i=1}^{n_l} (\boldsymbol{\xi}_i^{(l)} - \mathbf{m})(\boldsymbol{\xi}_i^{(l)} - \mathbf{m})^T = \Sigma_W + \Sigma_B. \tag{2.9}$$

The goal of linear discriminant analysis is, then, to find a $d \times m$ transform Φ such that $|\Phi^T \Sigma_B \Phi| / |\Phi^T \Sigma_W \Phi|$ is maximized, where $|\cdot|$ denotes the determinant. It can be proved that such a transform, Φ, is composed of m eigenvectors corresponding to the m largest non-zero eigenvalues of $\Sigma_W^{-1} \Sigma_B$. Due to the fact that the matrix Σ_B has a maximum rank of $c - 1$, the value of m must be less than c. Therefore, the dimensionality of the projected space is limited by the number of categories. Several variations of linear discriminant analysis have been reported in the literature. For example, Foley and Sammon [41] and Okada and Tomita [134] derived a set of discriminant vectors by selecting the projection axes one at a time under an orthogonality constraint. We use the total covariance matrix Σ_T instead of the between-class covariance matrix Σ_B in the criterion, so that the output dimensionality is not limited

by the number of categories.

We have proposed a neural network and a supervised self-organizing learning algorithm for multivariate linear discriminant analysis [111]. The linear discriminant

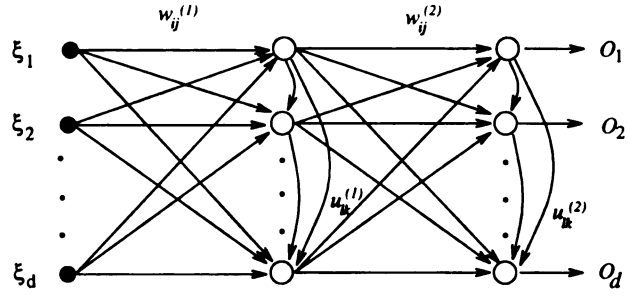


Figure 2.3. Architecture of the LDA network.

analysis (LDA) network is shown in Figure 2.3. It consists of two layers, each of which is identical to the PCA network proposed by Rubner et al. [151, 150] shown in Figure 2.2. Linear activation function is used for all the units. Let $w_{ij}^{(1)}$ ($w_{ij}^{(2)}$) be the weights on the interlayer connections from the i^{th} unit in the input (hidden) layer to the j^{th} unit in the hidden (output) layer. Let $u_{ij}^{(1)}$ ($u_{ij}^{(2)}$) denote the lateral weights from the i^{th} unit to the j^{th} unit within the hidden (output) layer. Moreover, let $\boldsymbol{\xi} = (\xi_1, \xi_2, \cdots, \xi_d)^T$ be a d-dimensional input pattern, and $\boldsymbol{\rho} = (\rho_1, \rho_2, \cdots, \rho_d)^T$ and $\boldsymbol{o} = (o_1, o_2, \cdots, o_d)^T$ be output vectors of the hidden layer and the output layer, respectively. We can write these outputs as

$$\begin{aligned} \rho_j &= \sum_{i=1}^{d} w_{ij}^{(1)} \xi_i + \sum_{l<j} u_{lj}^{(1)} \rho_l, \\ o_j &= \sum_{i=1}^{d} w_{ij}^{(2)} \rho_i + \sum_{l<j} u_{lj}^{(2)} o_l, \ j = 1, 2, \cdots, d. \end{aligned} \tag{2.10}$$

Let $T = \{\boldsymbol{\zeta}_k^{(l)} \mid k = 1, 2, \cdots, n_l, \ l = 1, 2, \cdots, c\}$ denote the set of input training patterns. Let $T_{T0} = \{\boldsymbol{\xi}_k^{(l)}, \ k = 1, 2, \cdots, n_l, \ l = 1, 2, \cdots, c\}$ denote the normalized

training data set obtained by subtracting each pattern in T by the global mean vector of the pooled training data set T, $\boldsymbol{\xi}_k^{(l)} = \boldsymbol{\zeta}_k^{(l)} - \mathbf{m}$. We form another training data set with class-conditional zero-mean vectors, $T_{W0} = \{\boldsymbol{\eta}_k^{(l)},\ k = 1, 2, \cdots, n_l,\ l = 1, 2, \cdots, c\}$ by subtracting each pattern in T by the mean vector of the corresponding class, $\boldsymbol{\eta}_k^{(l)} = \boldsymbol{\zeta}_k^{(l)} - \mathbf{m}^{(l)}$. The category information is only used to form the class-conditional zero-mean training set T_{W0}. Once this is done, the learning algorithm is fully self-organizing.

The proposed supervised self-organizing learning algorithm is given in Box 2.

Box 2

LDA Algorithm

1. Form training sets T_{W0} and T_{T0} from T.
2. Train the first layer using the PCA algorithm (Box 1) using T_{W0}.
3. Project all the patterns in T_{W0} using the first trained layer; compute the standard deviation for each output unit; scale all the weights on connections to each output unit by the standard deviation associated with this unit. $w_{ij}^{(1)\prime} = w_{ij}^{(1)}/\sigma_j,\ i, j = 1, 2, \cdots, d.$
4. Form data set T_H by collecting the output of the hidden layer when the training set T_{T0} is presented to the input layer.
5. Train the second layer using the PCA algorithm (Box 1) on data set T_H.

The learning parameters for the *PCA* and *LDA* algorithms in all the experiments in this chapter are specified as follows: $\eta(0) = 0.2$, $\mu(0) = 1.5$, $\beta(0) = 0.1$, $\alpha = 0.99999$. The learning process stops when the sum of weights on the lateral connections is below the threshold $t_u = 0.000001$, or the maximum number of iterations, 1,000,000, is reached* All the data sets are normalized by their maximum range, so

*In this chapter, one iteration means one presentation of a pattern.

these learning parameters satisfy the inequalities in Eq. (2.4).

The LDA algorithm actually performs a simultaneous diagonalization of two covariance matrices, Σ_W and Σ_T, of training data sets, T_{W0} and T_{T0}, respectively. In Step 2 of the LDA algorithm, the first layer tries to find the principal vectors of the data T_{W0}. Step 3 "whitens" the data set T_{W0} and, in the meantime, changes the distribution of the data set T_{T0} when it is transformed by the first layer in Step 4. Then, in Step 5 of the LDA algorithm, the second layer attempts to search for the principal vectors of T_H, the transformed version of T_{T0}.

Let $\Phi = [w_{ij}^{(1)}]_{d\times d}$ and Φ_s be the weight matrix when Steps 2 and 3 are executed, respectively. Let $\Psi = [w_{ij}^{(2)}]_{d\times d}$ be the weight matrix of the second layer after training. The overall mapping that the network performs is $\mathbf{o}_k^{(l)} = A^T \boldsymbol{\xi}_k^{(l)}$, where $A = \Phi_s \Psi$. We have shown that the columns of matrix A are composed of the eigenvectors of $\Sigma_W^{-1}\Sigma_T$ corresponding to the eigenvalues in the decreasing order [111]. When the learning process is finished, these eigenvalues can be obtained by computing the variances of the output units when the training set T_{T0} is presented to the input nodes. The precision of the neural computation is shown to be high enough for feature selection and projection purposes [111].

Independently of our work, Kuhnel and Tavan [102] have also investigated a network for linear discriminant analysis, but no simulation and comparison results were provided in [102].

Gallinari et al. [53] have established a link between linear discriminant analysis and linear feedforward networks. Their theoretical analysis shows that in a linear two-layer feedforward network with $rank(\Sigma_B)$ units in the hidden layer, trained to minimize the square-error criterion, the role of the hidden layer is to realize a linear discriminant analysis that maximizes the criterion $|\Sigma_{B_h}|/|\Sigma_{T_h}|$, where Σ_B is the between-class covariance matrix in the input space, and Σ_{B_h} and Σ_{T_h} are the between-class covariance and total-class covariance matrices, respectively, in the space spanned

by the hidden units. This result provides another method to perform a discriminant analysis of the input data by simply collecting the outputs of the hidden units. Comparing our proposed LDA network with this approach, the LDA network has the following two main advantages. First, the LDA algorithm operates in a "forward-propagation" fashion (first train the first layer, then the second layer, thus training of the two layers is independent), while the most commonly used method, backpropagation algorithm, for training the multilayer feedforward is in a "backpropagation" mode. Therefore, the convergence speed of the LDA algorithm is faster than the backpropagation algorithm. Secondly, the weight updating rules (strictly local) for the LDA network are much simpler than the one used in backpropagation algorithm where the updating a weight needs information to be back-propagated from all the units in the next layer. Therefore, the learning circuit of the LDA network is simpler than the one for the backpropagation network (feedforword network trained using backpropagation algorithm).

2.1.3 A Neural Network for Sammon's Projection (SAMANN)

Sammon [153] proposed a nonlinear projection technique that attempts to maximally preserve all the interpattern distances. Let $\boldsymbol{\xi}(\mu)$, $\mu = 1, 2, \cdots, n$, be the n d-dimensional patterns, and $\mathbf{y}(\mu)$, $\mu = 1, 2, \cdots, n$, be the n corresponding patterns in the m-dimensional projected space, $m < d$. The mapping error, also called Sammon's stress, is defined as

$$E = \frac{1}{\sum_{\mu=1}^{n-1} \sum_{\nu=\mu+1}^{n} d^*(\mu,\nu)} \sum_{\mu=1}^{n-1} \sum_{\nu=\mu+1}^{n} \frac{[d^*(\mu,\nu) - d(\mu,\nu)]^2}{d^*(\mu,\nu)}, \tag{2.11}$$

where $d^*(\mu,\nu)$ and $d(\mu,\nu)$ are the distances between pattern μ and pattern ν in the input space and in the projected space, respectively. Euclidean distance is commonly

used in this projection algorithm.

Sammon's stress E is a measure of how well the interpattern distances are preserved when the patterns are projected from a high dimensional space to a lower dimensional space. Sammon [153] used a gradient descent algorithm to find a configuration of n patterns in the m-dimensional space that minimizes E. This method views the positions of n patterns in the m-dimensional space as nm variables in an optimization problem. There are many local minima on the energy (or error) surface and it is unavoidable for the algorithm to get stuck in a local minimum. One usually runs the algorithm several times with different random initial configurations and chooses the configuration with the lowest stress. Sammon's algorithm involves a large amount of computation. Since for every step within an iteration, $n(n-1)/2$ distances have to be computed, the algorithm quickly becomes impractical for a large number of patterns.

Sammon's algorithm does not provide an explicit mapping function governing the relationship between the patterns in the original space and in the configuration (projected) space. Therefore, it is impossible to decide where to place new d-dimensional data in the final m-dimensional configuration created by Sammon's algorithm. In other words, Sammon's algorithm has no generalization capability. In order to project new data, one has to run the program again on pooled data (old data and new data).

We have proposed an *unsupervised* backpropagation learning algorithm to train a multilayer feedforward neural network to perform the well-known Sammon's nonlinear projection. The proposed learning algorithm, which needs no category information about patterns, is an extension of the backpropagation algorithm. Figure 1.2 shows the neural network architecture. The number of input nodes, d, is set to the input dimensionality of the feature space. The number of output units, m, is specified as the dimensionality of the projected space, $m < d$.

Let $\boldsymbol{\xi} = (\xi_1, \xi_2, \cdots, \xi_d)$ be a d-dimensional input pattern vector. We denote the output of j^{th} unit in layer l by $y_j^{(l)}$, $j = 1, 2, \cdots, n_l$, $l = 0, 1, 2, \cdots, L$, where n_l is the

number of units in layer l, L is the number of layers, and $y_j^{(0)} = \xi_j,\ j = 1, 2, \cdots, d$. The weight on connection between unit i in layer $l-1$ and unit j in layer l is represented by $w_{ij}^{(l)}$. We denote $w_{0j}^{(l)}$ as the bias in the j^{th} unit in the l^{th} layer, and $y_0^{(l)} = 1.0$. The sigmoid activation function $g(h)$ whose range is (0.0, 1.0) is used for each unit, where h is the weighted sum of all the inputs to the unit. Therefore, the output of the j^{th} unit in layer l can be written as

$$y_j^{(l)} = g\left(\sum_{i=0}^{n_l} w_{ij}^{(l)} y_i^{(l-1)}\right),\ l = 1, 2, \cdots, L. \tag{2.12}$$

Using the above notation, the $d(\mu, \nu)$ term in Eq. (2.11) can be expressed as

$$d(\mu, \nu) = \{\sum_{k=1}^{m} [y_k^{(L)}(\mu) - y_k^{(L)}(\nu)]^2\}^{1/2}, \tag{2.13}$$

where μ and ν are the two pattern indices.

Note that the range of each output is between 0.0 and 1.0. If the range of input values is large, it is impossible for the projection algorithm to preserve the interpattern distances no matter which learning rule is used. Therefore, we normalize all the input patterns to equalize the maximum interpattern distances in the original space and in the projected space. Let

$$\lambda = \frac{1}{\sum_{\mu=1}^{n-1} \sum_{\nu=\mu+1}^{n} d^*(\mu, \nu)}, \tag{2.14}$$

which is independent of the network and can be computed beforehand, and

$$E_{\mu\nu} = \lambda \frac{[d^*(\mu, \nu) - d(\mu, \nu)]^2}{d^*(\mu, \nu)}, \tag{2.15}$$

then

$$E = \sum_{\mu=1}^{n-1} \sum_{\nu=\mu+1}^{n} E_{\mu\nu}. \tag{2.16}$$

Note that $E_{\mu\nu}$ is proportional to the interpattern distance changes between patterns μ and ν, due to the projection from the d-dimensional feature space to the m-dimensional projected space. So, $E_{\mu\nu}$ is more appropriate for the pattern-by-pattern-based updating rules.

We have derived the updating rule for the multilayer feedforward network which minimizes Sammon's stress defined in Eq. (2.11) based on the gradient descent method. For the output layer ($l = L$),

$$\begin{aligned}\frac{\partial E_{\mu\nu}}{\partial w_{jk}^{(L)}} &= \left(\frac{\partial E_{\mu\nu}}{\partial d(\mu,\nu)}\right)\left(\frac{\partial d(\mu,\nu)}{\partial [y_k^{(L)}(\mu) - y_k^{(L)}(\nu)]}\right)\left(\frac{\partial [y_k^{(L)}(\mu) - y_k^{(L)}(\nu)]}{\partial w_{jk}^{(L)}}\right) \\ &= \left(-2\lambda \frac{d^*(\mu,\nu) - d(\mu,\nu)}{d^*(\mu,\nu)}\right)\left(\frac{y_k^{(L)}(\mu) - y_k^{(L)}(\nu)}{d(\mu,\nu)}\right) \\ &\quad \left(g'(h_{k,\mu}^{(L)}) y_j^{(L-1)}(\mu) - g'(h_{k,\nu}^{(L)}) y_j^{(L-1)}(\nu)\right) \qquad (2.17)\end{aligned}$$

where $g'(h_{k,\cdot}^{(L)})$ is the derivative of the sigmoid function of unit k in layer L (output layer) with respect to the net input, $h_{k,\cdot}$, of this unit,

$$g'(h_{k,\cdot}) = (1 - y_k^{(L)}(\cdot)) y_k^{(L)}(\cdot). \qquad (2.18)$$

Let

$$\delta_k^{(L)}(\mu,\nu) = -2\lambda \frac{d^*(\mu,\nu) - d(\mu,\nu)}{d^*(\mu,\nu) d(\mu,\nu)} [y_k^{(L)}(\mu) - y_k^{(L)}(\nu)], \qquad (2.19)$$

$$\Delta_k^{(L)}(\mu) = \delta_k^{(L)}(\mu,\nu)[1 - y_k^{(L)}(\mu)] y_k^{(L)}(\mu), \qquad (2.20)$$

$$\Delta_k^{(L)}(\nu) = \delta_k^{(L)}(\mu,\nu)[1 - y_k^{(L)}(\nu)] y_k^{(L)}(\nu), \qquad (2.21)$$

where $\delta_k^{(L)}(\mu,\nu)$ is the change in the output scaled by the normalized interpattern distance change when we present patterns μ and ν. As we will see later, $\Delta_k^{(L)}(\mu)$ and $\Delta_k^{(L)}(\nu)$ will be backpropagated to layer $L-1$. Substituting Eqs. (2.18)-(2.21) into

Eq. (2.17), we get

$$\frac{\partial E_{\mu\nu}}{\partial w_{jk}^{(L)}} = \Delta_k^{(L)}(\mu) y_j^{(L-1)}(\mu) - \Delta_k^{(L)}(\nu) y_j^{(L-1)}(\nu). \tag{2.22}$$

The updating rule for the output layer is

$$\begin{aligned} \Delta w_{jk}^{(L)} &= -\eta \frac{\partial E_{\mu\nu}}{\partial w_{jk}^{(L)}} \\ &= -\eta \left(\Delta_k^{(L)}(\mu) y_j^{L-1}(\mu) - \Delta_k^{(L)}(\nu) y_j^{(L-1)}(\nu) \right), \end{aligned} \tag{2.23}$$

where η is the learning rate.

Similarly, we can obtain the general updating rule for all the hidden layers, $l = 1, \cdots, L-1$.

$$\begin{aligned} \Delta w_{ij}^{(l)} &= -\eta \frac{\partial E_{\mu\nu}}{\partial w_{ij}^{(l)}} \\ &= -\eta \left(\Delta_j^{(l)}(\mu) y_i^{(l-1)}(\mu) - \Delta_j^{(l)}(\nu) y_i^{(l-1)}(\nu) \right), \end{aligned} \tag{2.24}$$

where

$$\Delta_j^{(l)}(\mu) = \delta_j^{(l)}(\mu)[1 - y_j^{(l)}(\mu)] y_j^{(l)}(\mu), \tag{2.25}$$

$$\Delta_j^{(l)}(\nu) = \delta_j^{(l)}(\nu)[1 - y_j^{(l)}(\nu)] y_j^{(l)}(\nu), \tag{2.26}$$

and

$$\delta_j^{(l)}(\mu) = \sum_{k=1}^{m} \Delta_k^{(l+1)}(\mu) w_{jk}^{(l+1)}, \tag{2.27}$$

$$\delta_j^{(l)}(\nu) = \sum_{k=1}^{m} \Delta_k^{(l+1)}(\nu) w_{jk}^{(l+1)}. \tag{2.28}$$

Analogous to the backpropagation learning algorithm, $\delta_j^{(l)}(\mu)$ and $\delta_j^{(l)}(\nu)$ are changes in layer l backpropagated from its successor layer, $l+1$, for pattern μ and pattern ν, respectively. As we can see from Eqs. (2.23) and (2.24), to update the

weights, we need to present a pair of patterns to the network instead of one pattern at a time in the backpropagation learning algorithm. This can be accomplished by either building two identical networks or just storing all the outputs of the first pattern before presenting the second pattern. The above learning rule for Sammon's projection makes use of the popular backpropagation algorithm. But, we do not need category information of the patterns. Therefore, we have extended the backpropagation algorithm to perform unsupervised learning.

The SAMANN unsupervised backpropagation algorithm is summarized in Box 3.

Box 3

SAMANN Unsupervised Backpropagation Algorithm

1. Initialize weights randomly in the SAMANN network.
2. Select a pair of patterns randomly, present them to the network one at a time, and evaluate the network in a feedforward fashion.
3. Update the weights using Eqs. (2.23) and (2.24) in the backpropagation fashion starting from the output layer.
4. Repeat steps 2–3 a number of times.
5. Present all the patterns and evaluate the outputs of the network; compute Sammon's stress; if the value of Sammon's stress is below a prespecified threshold or the number of iterations (from steps 2–5) exceeds the prespecified maximum number, then stop; otherwise, go to step 2.

Similar to the backpropagation algorithm, we can add a momentum term in updating the weights in order to speed up the convergence.

The selection of the number of hidden layers and the number of units in each hidden layer in a multilayer feedforward network is an important yet difficult problem. In order to achieve the representation power of Sammon's algorithm, we have to use a

network with at least mn free parameters, where mn is the number of variables in Sammon's algorithm. So, mn becomes a lower bound for the total number of free parameters. This lower bound becomes very large when the number of patterns is large. However, we have found that it is often not necessary to use such a large number of free parameters to obtain reasonable projection maps. On the other hand, the network should be as small as possible to obtain good generalization capability. A compromise between these two factors is necessary for selecting an appropriate network.

A significant advantage of the SAMANN network is that the network is able to project new patterns after training because of the effect of regularization [112]. Some preliminary simulation results were reported in [85], which demonstrated this good generalization capability.

The local minimum problem in both the methods can be improved by choosing good initial configuration in the original Sammon's algorithm and good initial weights in the SAMANN network. We have found that the values of Sammon's stress of principal component analysis and of Sammon's projection are close for many data sets. This is because when all the interpattern distances in a data set are maximally preserved, the variance of the data is also retained to a very high degree. Therefore, we propose to use the PCA projection map as an initial configuration in Sammon's projection. The PCA projection map can be obtained from the PCA network. Then, the SAMANN network is trained using the standard backpropagation algorithm to approximate principal component analysis. Finally, the proposed unsupervised backpropagation algorithm is used to take advantage of the nonlinearity of the SAMANN network to refine the projection map.

In all the experiments reported in this chapter, a two-layer (one hidden layer) network with 20 hidden units is used. The SAMANN network is trained using the standard backpropagation algorithm with a learning rate of 0.7 and momentum value

of 0.3 for 200,000 iterations. Then, the unsupervised backpropagation algorithm is used to train the SAMANN network for 60,000 iterations. The learning rate and momentum value in the unsupervised backpropagation algorithm are 0.02 and 0.01, respectively for the artificial data sets, and 0.05 and 0.02, respectively for the real data sets.

2.1.4 A Nonlinear Projection Based on Kohonen's Self-Organizing Map (NP-SOM)

Kohonen's self-organizing map (SOM) [92, 93, 94] belongs to the category of unsupervised nonlinear projection methods. Kohonen's Self-Organizing map has a very desirable property of topology preserving, which captures an important aspect of the feature maps in human brain. The network architecture is shown in Figure 2.4. It basically consists of a two-dimensional array of units, each of which is connected to all the d input nodes. Let $\mathbf{w}_{ij}$ denote the d-dimensional vector associated with the unit at location (i, j) of the 2-D array. Kohonen's self-organizing map algorithm is briefly summarized in Box 4.

A slightly different formulation of the learning rule, which is used in this chapter, is the kernel updating rule [94]:

$$\mathbf{w}_{ij}(t+1) = \mathbf{w}_{ij}(t) + h_{c_i c_j}(t)[\boldsymbol{\xi}(t) - \mathbf{w}_{ij}(t)], \tag{2.29}$$

where $h_{c_i c_j}(t)$ is a Gaussian weighting function:

$$h_{c_i c_j}(t) = h_0(t) exp\left(-\frac{[(i-c_i)^2 + (j-c_j)^2]}{2\sigma(t)^2}\right). \tag{2.30}$$

Here, $h_0(t)$ and $\sigma_0(t)$ are chosen as suitable decreasing functions of time. We use the following parameters in all our experiments: $h_0(0) = 0.05$, $\sigma(0) = 66.666$, $h_0(t+1) =$

$max\{0.9999 \times h_0(t), 0.0001\}$, $\sigma(t+1) = max\{0.9999 \times \sigma(t), 1.0\}$, and the maximum number of iterations is 100,000.

Box 4

Kohonen's Self-Organizing Map Algorithm

1. Initialize weights to small random numbers, set initial learning rate and neighborhood;

2. Present a pattern $\boldsymbol{\xi}$, and evaluate the network outputs;

3. Select the unit (c_i, c_j) with the minimum output:

$$||\boldsymbol{\xi} - \mathbf{w}_{c_i c_j}|| = \min_{ij} ||\boldsymbol{\xi} - \mathbf{w}_{ij}||$$

4. Update all the weights according to the kernel-based learning rule;

$$\mathbf{w}_{ij}(t+1) = \begin{cases} \mathbf{w}_{ij}(t) + \alpha(t)[\boldsymbol{\xi}(t) - \mathbf{w}_{ij}(t)], & \text{if } (i,j) \in N_{c_i c_j}(t), \\ \mathbf{w}_{ij}(t), & \text{otherwise,} \end{cases}$$

where $N_{c_i c_j}(t)$ is the neighborhood of unit (c_i, c_j) at time t, and $\alpha(t)$ is the learning rate.

5. Decrease the value of $\alpha(t)$ and shrink the neighborhood $N_{c_i c_j}(t)$;

6. Repeat steps 2 – 5 until the change in weight values is less than a pre-specified threshold, or the maximum number of iterations is reached.

Because of the topology-preserving property of the SOM map, some insight into the data can be gained by examining the activation pattern of the 2D array of units when all the input patterns are presented to the network after the training is done. If the category information of the data is available, we can label each unit in the array by the majority class label of the patterns which are projected onto the unit.

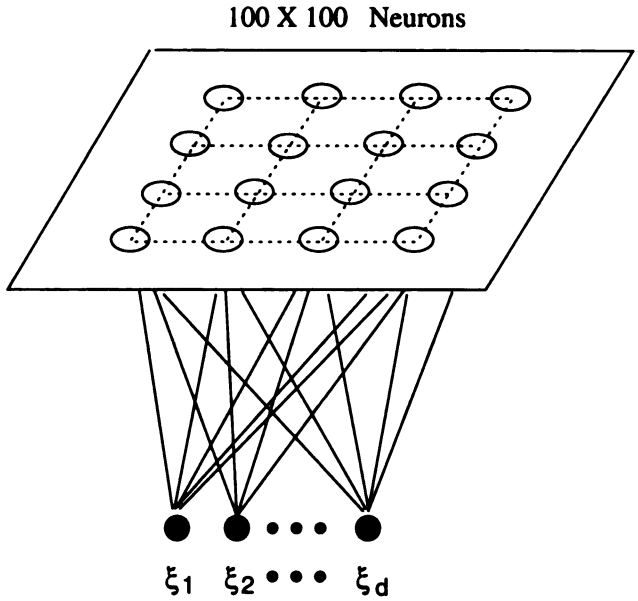


Figure 2.4. A sketch of the NP-SOM network.

If the classes in the data are well-separated, the 2D projection map will be divided into regions by classes. Kohonen [93] used this method to obtain the phoneme map which was used in the phonetic typewriter. Since each unit in the network has a tendency to encode (become a prototype of) a region in the input space according to the density distribution of the data, all the units have approximately the same chance to be activated. Therefore, patterns in the SOM map are quite uniformly distributed. This makes it very difficult to visualize the structure of the data when the category information is not available. Moreover, we are not able to display high dimensional weight vector associated with each unit. So, Kohonen's SOM maps are not suitable for visualization of high dimensional data with no category information.

We have developed a nonlinear projection method (NP-SOM) based on Kohonen's SOM to facilitate visualization of data, where a 100×100 array of units is used [100]. A sketch of the NP-SOM network is shown in Figure 2.4. First, we train the network using Kohonen's self-organizing learning algorithm on a data set. Then, we display the network as a two-dimensional distance image, in which each pixel represents

a unit in the 2-D network. The gray value of a pixel is the maximum distance in the weight vector space between the corresponding unit and its four neighbors. In the distance image, dark regions separated by bright curves represent different clusters. This display technique helps us in visualizing the clustering tendency and underlying structure of the data. Experiments in Section 2.2 will show that the clustering tendency of many data sets becomes apparent in the 2D distance image. The inter-unit distance is also defined which enables us to compute Sammon's stress for this projection [100].

2.1.5 Nonlinear Discriminant Analysis (NDA) Network

Webb and Lowe [173] have investigated a class of multilayer feedforward networks with nonlinear hidden units and linear output units, trained to minimize the squared error between the desired output and actual output of the network. They have found that the nonlinear transformation implemented by the subnetwork from the input layer to the final hidden layer of such networks maximizes the so-called network discriminant function, $\mathrm{Tr}\{S_B S_T^+\}$, where S_T^+ is the pseudo-inverse of the total scatter matrix of the patterns at the output of the final hidden layer, and S_B is the weighted between-class scatter matrix of the output of the final hidden layer (the weights are determined by the coding scheme of the desired outputs). Although this result was derived for the specific class of multilayer feedforward networks with nonlinear hidden units and linear output units, its interpretation can be extended to general multilayer feedforward networks. The role of hidden layers is to implement a nonlinear transformation which projects input patterns from the original space to a space in which patterns are easily separated by the output layer. The first part of the network (from the input layer to the output layer) actually realizes a nonlinear discriminant analysis.

We have used this theoretical result to perform a nonlinear discriminant analysis

(NDA) of input data [111]. The main objective of this method is to visualize high dimensional data to determine whether it is necessary to compute a nonlinear decision boundary in the output layer (e.g., by using another multilayer network, or k-nearest neighbor classifier). This projection method falls into the category of supervised nonlinear approaches. The network architecture is the same as the SAMANN (Figure 1.2). We specify the number of input nodes to be the number of features and the number of units in the output layer to be the number of pattern classes in the data set. Furthermore, we fix the number of units in the last hidden layer to m, the dimensionality of the projected space. Thus, from input layer to the last hidden layer, the network implements a nonlinear projection from d-dimensional space to m-dimensional space. If the entire network can correctly classify a linearly-nonseparable data set, then this projection actually transforms the linearly-nonseparable data to a linearly-separable data in some "other" space. Therefore, by visualizing the data in this m-dimensional space ($m = 2, 3$), we can gain some insight into the structure of the data. In all our simulations, we use $m = 2$. The backpropagation learning algorithm is used to train the feedforward network with two hidden layers. Sigmoid activation function is used in all the units. In order to avoid squashing the data for the projection map, we replace the sigmoid activation function by a linear function in all the units in the last hidden layer after the training is done.

2.1.6 Summary of Networks for Feature Extraction and Data Projection

Table 2.1 lists several important features of the five representative networks for feature extraction and data projection: PCA, LDA, SAMANN, NDA and NP-SOM.

Table 2.1. Some features of the five representative networks.

Network	PCA	LDA	SAMANN	NDA	NP-SOM
Class in taxonomy of Fig. 2.1	unsupervised linear	unsupervised linear	unsupervised nonlinear	supervised nonlinear	unsupervised nonlinear
Architecture	Fig. 2.2	Fig. 2.3	Fig. 1.2	Fig. 1.2	Fig. 2.4
Activation function	linear	linear	sigmoid	sigmoid	Euclidean
No. inputs	d	d	d	d	d
No. outputs	m	m	m	c	100×100
No. layers	1	2	> 2	> 2	1
Learning algorithm	PCA (Box 1)	LDA (Box 2)	unsupervised backpropagation (Box 3)	standard backpropagation	Kohonen's SOM (Box 4)

2.2 Comparative Study

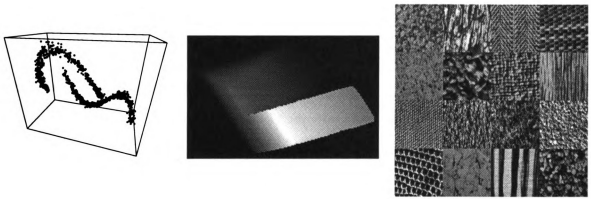
We will evaluate the performance of the five networks for feature extraction and data projection based on a visual judgment of the 2D projection maps and three numerical criteria over eight data sets with various properties. These networks are: (i) Rubner's PCA network, (ii) LDA network, (iii) NDA network, (iv) network for Sammon's projection (SAMANN), and (v) the nonlinear projection method based on Kohonen's self-organizing map (NP-SOM).

2.2.1 Methodology

We have generated or collected eight data sets for evaluating feature extraction and data projection approaches. These 8 data sets are briefly described below.

1. Two normally distributed clusters ($\Sigma_1 = \Sigma_2 = I$ and $\mu_1 = -\mu_2 = (1, 1, \cdots, 1)^T$) in a 10-dimensional space with 500 patterns per class. This is a well-separated data set with a Bayes error of 0.078%.

2. Two elongated clusters in a 3-dimensional space. A perspective view of this data set is shown in Figure 2.5(a). This example is not linearly separable.
3. Two uniformly distributed sets of points on two 3-dimensional spherical surfaces. One sphere containing 800 patterns is centered at (0, 0, 0) with radius 1, and the other containing 200 patterns is centered at (0, 0, 0.2) with radius 0.1. This data set has a relatively symmetric structure and its two classes are not linearly separable.
4. Uniformly distributed "noise" in a 10-dimensional unit hypercube. All the 1,000 points are randomly labeled into two classes with 500 points per class. This data set is used to test how feature extraction and data projection algorithms behave on data sets with no meaningful structure.
5. IRIS data set. The well-known data set consisting of 150 4-dimensional patterns from 3 classes. It contains 4 measurements on 50 flowers from each of the 3 species of the Iris flower.
6. 8OX data set. It consists of 45 8-dimensional patterns from 3 classes (the hand-printed characters "8", "O" and "X") with 15 patterns per class. This is a very sparse data set.
7. Data set containing 4 features extracted from a range image [66]. Figure 2.5(b) shows the range image of a polyhedral object. The gray level at each pixel in this range image encodes the distance (depth) from the sensor to the corresponding point on the object's surface. The 4 features are the three components of the surface normal vector and the depth value at each pixel. A total of 1,000 pixels are randomly chosen from the polyhedral object. These 1,000 patterns are labeled with 5 classes (4 visible surfaces plus a class of boundary and edge points).
8. Data set containing 22 Gabor filter features [82] extracted from an image with 16 different textures shown in Figure 2.5(c). A total of 1,000 pixels are randomly



(a) (b) (c)

Figure 2.5. Some data sets used in the comparative study. (a) A perspective view of data set 2. (b) Range image and (c) texture image from which data sets 7 and 8 are extracted, respectively.

chosen from the image (512×512).

These eight data sets differ from each other in one or more characteristics, such as the data source (artificial/real data), dimensionality of the pattern vector (d), linear dimensionality (d_l), number of classes/clusters (c), number of patterns (n), linear separability (λ_s), inherent structure of the data, and sparseness. The linear dimensionality (d_l) of a data set is measured by the number of significant eigenvalues (more than 97% of the total variance is retained by the first d_l principal components) of the covariance matrix of the data. The linear separability (λ_s) is defined as the largest eigenvalue of $\Sigma_T^{-1}\Sigma_B$ [53]. Note that λ_s is restricted to the [0.0,1.0] range. As λ_s increases from 0.0 to 1.0, the data set becomes more and more linearly separable. The inherent structure of the data is ranked as "random", "weak", "mediun" or "strong" based on our *a priori* knowledge about the data set. The sparseness of a data set is measured by the ratio of the dimensionality to the number of patterns in the data set; the larger this ratio, the sparser the data. This index does not take

the spread of data into consideration because the spread can be easily scaled. These characteristics of the 8 data sets are summarized in Table 2.2, from which we can see that they constitute a reasonable benchmark for the evaluation of feature extraction and data projection methods.

Table 2.2. Characteristics of the eight data sets.

Data set	Name	source	d	d_l	c	n	λ_s	structure	sparseness
1	Gauss2	artificial	10	10	2	1000	0.91	strong	0.01
2	Curve2	artificial	3	2	2	1000	0.70	strong	0.003
3	Sphere2	artificial	3	3	2	1000	0.38	strong	0.003
4	Random	artificial	10	10	1(2)	1000	0.48	random	0.01
5	IRIS	real	4	2	3	150	0.97	medium	0.027
6	8OX	real	8	7	3	45	0.87	weak	0.178
7	Range	real	4	3	5	1000	0.81	strong	0.004
8	Texture	real	22	15	16	1000	0.95	medium	0.022

The performance of the five neural networks on these 8 data sets are evaluated based on visual judgment of the 2D projection maps and three numerical criteria. The three criteria proposed in [100] are modified and utilized in this chapter for the numerical evaluation of neural networks for feature extraction and data projection. These are: (i) Sammon's stress (Eq. (2.11)), (ii) the nearest-neighbor classification error rate P_e^{NN} on projected data, and (iii) the minimum-distance classification error rate P_e^{MD} of projected data. Since the classification error of the projected data P_e(projection) depends on the classification error of the original data P_e(original), we propose to use a normalized ratio, $R_e = \frac{1+P_e(\text{projection})}{1+P_e(\text{original})}$. The value of R_e is within the range of (0.5,2.0). When $R_e = 1.0$, it means that the extracted features have the same discriminatory power as the original features for a given classifier. If $R_e < 1.0$, the extracted features have a better performance in terms of classification accuracy. This could happen, for example, if the effect of the "curse of dimensionality" is eliminated

by the feature extraction and data projection. Sammon's stress measures how well the projection preserves the interpattern distances. The nearest-neighbor classification error rate indicates how well the extracted features or projection preserves the local category structure of the data, while the minimum-distance classification error rate provides some information on the linear separability in the new feature space. The error rates are evaluated using the "leave-one-out" (cross-validation) technique.

2.2.2 Visualization of Projection Maps

Information about the structure, intrinsic dimensionality, clustering tendency and cluster shape of the data is very useful in choosing a proper classification or clustering method. However, this information is not directly accessible for high dimensional data. Data projection is a tool which is frequently used to explore various properties of the data. Here, we project all the 8 data sets onto two dimensions using the five networks. From these maps, we can observe not only some of the properties of the 8 data sets, but also different characteristics of the five networks. Figures 2.6-2.13 show the 2D projection maps for the 8 data sets (for the data sets containing 1,000 patterns, only 500 randomly-chosen patterns are displayed). In each of these figures, there are four 2D maps which are generated by the following four networks: PCA, LDA, SAMANN and NDA networks. The projection maps obtained by the NP-SOM network will be discussed later.

The PCA network performs a linear orthogonal projection. The 2D PCA map produced by the PCA network is spanned by two orthogonal vectors (in the original space) along which the data have the largest and the second largest variances. Therefore, PCA maps are the easiest one to interpret. For some of the data sets, for example, **Gauss2** (Figure 2.6(a)), IRIS (Figure 2.10(a)), and **Range** (Figure 2.12(a)), their PCA maps also exhibit much of the class-discriminatory information. But, this is not always true. For some other data sets, e.g., **Sphere2** (Figure 2.8(a)), **Curve2**

(Figure 2.7(a)), and **8OX** (Figure 2.11(a)), patterns from different classes either overlap or are not linearly separable in the PCA projection.

The LDA network also implements a linear, but not necessarily orthogonal, projection. It first attempts to find a plane in the high dimensional space and then skews the plane in order to increase the discriminatory ability of both the projection axes. Examples are the LDA maps of the data sets: **Curve2** (Figure 2.7(b)), **8OX** (Figure 2.11(b)) and **IRIS** (Figure 2.10(b)). Note that the elongated clusters in the PCA map of the **Range** data are squashed in the LDA map. However, the LDA network can not improve the linear separability of the data sets **Sphere2** (Figure 2.8(b)) and **Random** (Figure 2.9(b)), because it is a linear method.

The SAMANN maps are very similar to the PCA maps for all the 8 data sets. This is not only because the SAMANN network uses the PCA map as the initial pattern configuration which determines the initial weights of the SAMANN network, but also because the Sammon's projection attempts to maximally preserve the interpattern distances, which means that it also preserves the variance as much as possible. However, the SAMANN network can refine the PCA map using its inherent nonlinearity in order to obtain a map with less distortion in interpattern distances.

The NDA network tries to reorganize and group all the patterns by categories such that patterns in the projection map can be separated much more easily. This property is fully demonstrated on all the 8 data sets, especially on the linearly non-separable data sets such as, **Curve2** (Figure 2.7(d)) and **Sphere2** (Figure 2.8(d)). It is remarkable to see that the data points on the entire outer sphere are stripped off from the inner sphere (see the small black "blob" which is isolated from a cloud of patterns (on the outer sphere) in Figure 2.8(d))! This indicates that incorporating a nonlinear transform as well as category information leads to a very powerful method for projecting multivariate data.

Figure 2.14 shows the distance images of Kohonen's SOM superimposed on the

2D projection maps (white dots) for the 8 data sets using the NP-SOM network. We can see that these distance images are divided into several dark regions separated by bright wide curves. Pixels (units) in the same dark region usually have relatively small distances to each other in the original feature space. The clustering tendency of the 8 data sets becomes apparent from these distance images. Data sets, **Gauss2** (Figure 2.14(a)), **Curve2** (Figure 2.14(b)), **Sphere2** (Figure 2.14(c)) and **Range** (Figure 2.14(g)), have well-separated (linearly or nonlinearly) clusters. The **IRIS** data set (Figure 2.14(e)) has two well-isolated clusters, which is consistent with our *a priori* knowledge about the **IRIS** data set (two of the three categories are slightly overlapping in the feature space). The **Random** data set has no clustering tendency. The number of clusters in a data set can be roughly estimated by counting the number of dark regions in the corresponding distance image. This is easy to do with data sets containing well-isolated clusters, but difficult with others. The separation between two clusters is indicated by the brightness of the boundary between them. It is interesting to notice that each pattern in the **8OX** data set (Figure 2.14(f)) occupies a separate dark region. This is because **8OX** data set (containing only 45 patterns in an 8D space) is too sparse and the size of the map (100×100) is too large. Since the category information for all the data sets except for the **Random** data set is known, we can determine the class labels for these white dots on the projection maps. If more than one pattern is projected into the same unit (pixel) in the map, the class label of the pixel is determined by the majority pattern class (ties are broken arbitrarily).

We have manually drawn the boundaries (bright thin curves) between different classes on the projection maps, which are superimposed on the corresponding distance images. We can see from Figure 2.14 that the 2D projection maps fit the corresponding distance images very well. The patterns from the same class are grouped into a single region except for the **Random** data set. For the **IRIS** data set (Figure 2.14(e)), even though there is no clear bright curve between two of the three classes, the two

classes occupy different regions in the image. For the **Range** data set, most patterns from the class containing edge (jump and crease) points in the range image (Figure 2.5(b)) are scattered over a bright region in Figure 2.14(g), which implies that these patterns do not form a compact cluster in the feature space, and should be considered as outliers. This observation is consistent with the fact that the surface normals at these points on the object are not reliably estimated. The distance images are quite helpful in exploring the structure of the data, when the category information is not available. Since the patterns are relatively uniformly distributed in the 2D projection maps generated by the NP-SOM network compared to the PCA and SAMANN networks, it is very difficult to observe the clustering tendency based on the unlabeled 2D projection map for most data sets without the distance image.

Properties of the 8 data sets revealed by visualizing them in 2D maps can suggest what type of classification or clustering algorithms to use. For the **Gauss2** data set, a linear classifier or a squared-error clustering algorithm can be applied. In fact, most classifiers and clustering algorithms will work well on this data set. We also notice that one single feature (the first principal component) contains almost all the discriminatory information. On the other hand, from the projection maps of the **Curve2** and **Sphere2** data sets, linear classifiers are not powerful enough to distinguish the two classes in these data sets. It also becomes clear that the square-error clustering algorithm is not suitable for these two data sets. These projection maps suggest the use of nonlinear classifiers such as, the k-nearest neighbor classifier and multilayer feedforward networks, or some hierarchical clustering algorithms [81] such as, single-link and complete-link clustering algorithms. For the data sets, **IRIS**, **8OX** and **Texture**, both linear and nonlinear classifiers can be applied, but a nonlinear classifier may give a better performance. The square-error clustering algorithm may also be applied to these data sets. Moreover, from the projection maps of the **Texture** data (Figure 2.13), it is apparent that the two extracted features are not sufficient for

classification. This will be demonstrated further by the classification error rate using the 2D projection maps. An interesting property of the **Range** data set revealed by the projection maps in Figure 2.12 is that this data contains mainly four elongated clusters. These elongated clusters are roughly linearly separable. This property suggests that a square-error clustering algorithm with the Euclidean distance should not be used for this data set. The regularized Mahalanobis distance (see Chapter 4) would lead to a better clustering solution.

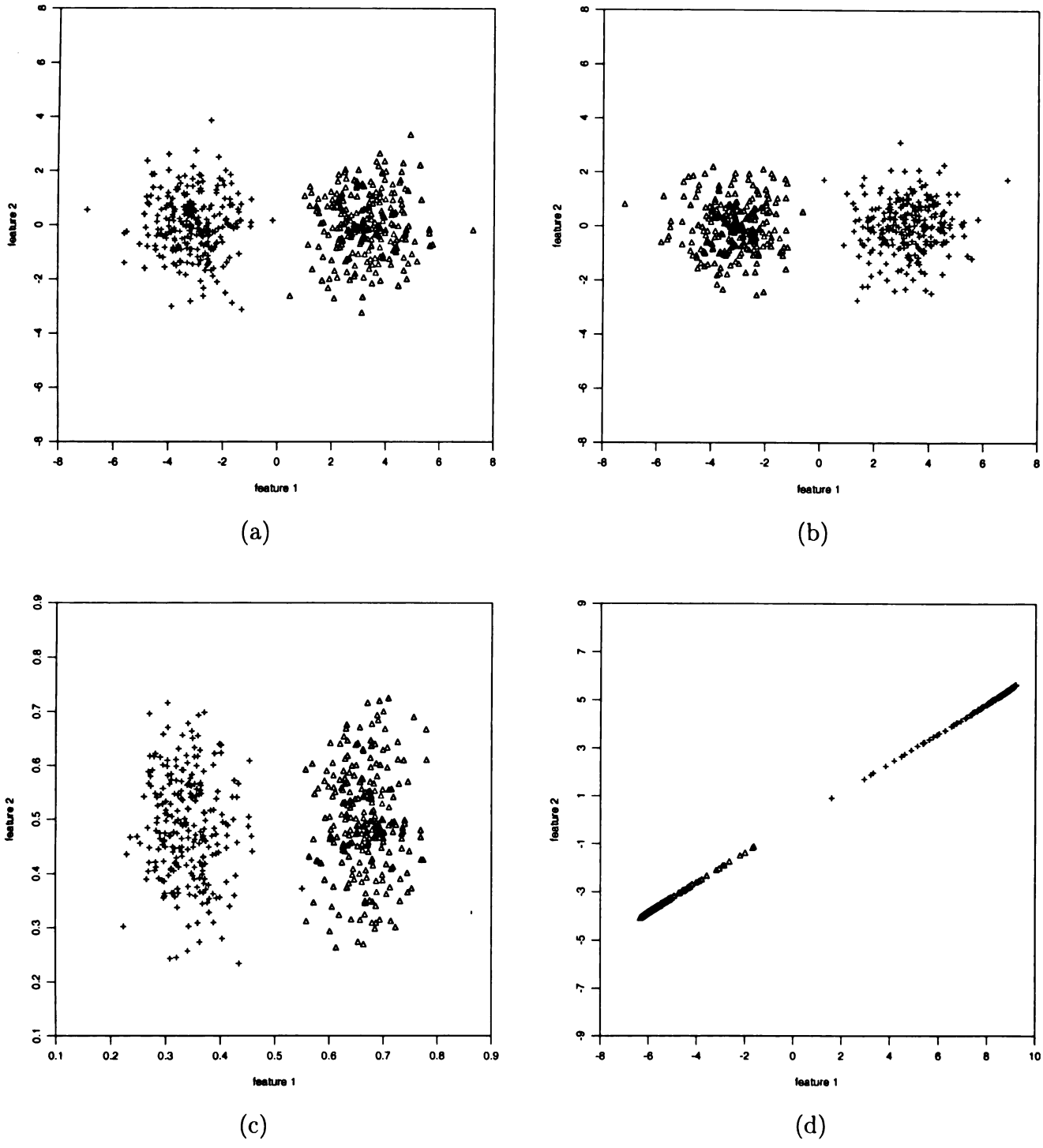


Figure 2.6. 2D projection maps of data set **Gauss2** using the four networks. (a) PCA network, (b) LDA network, (c) SAMANN network, and (d) NDA network.

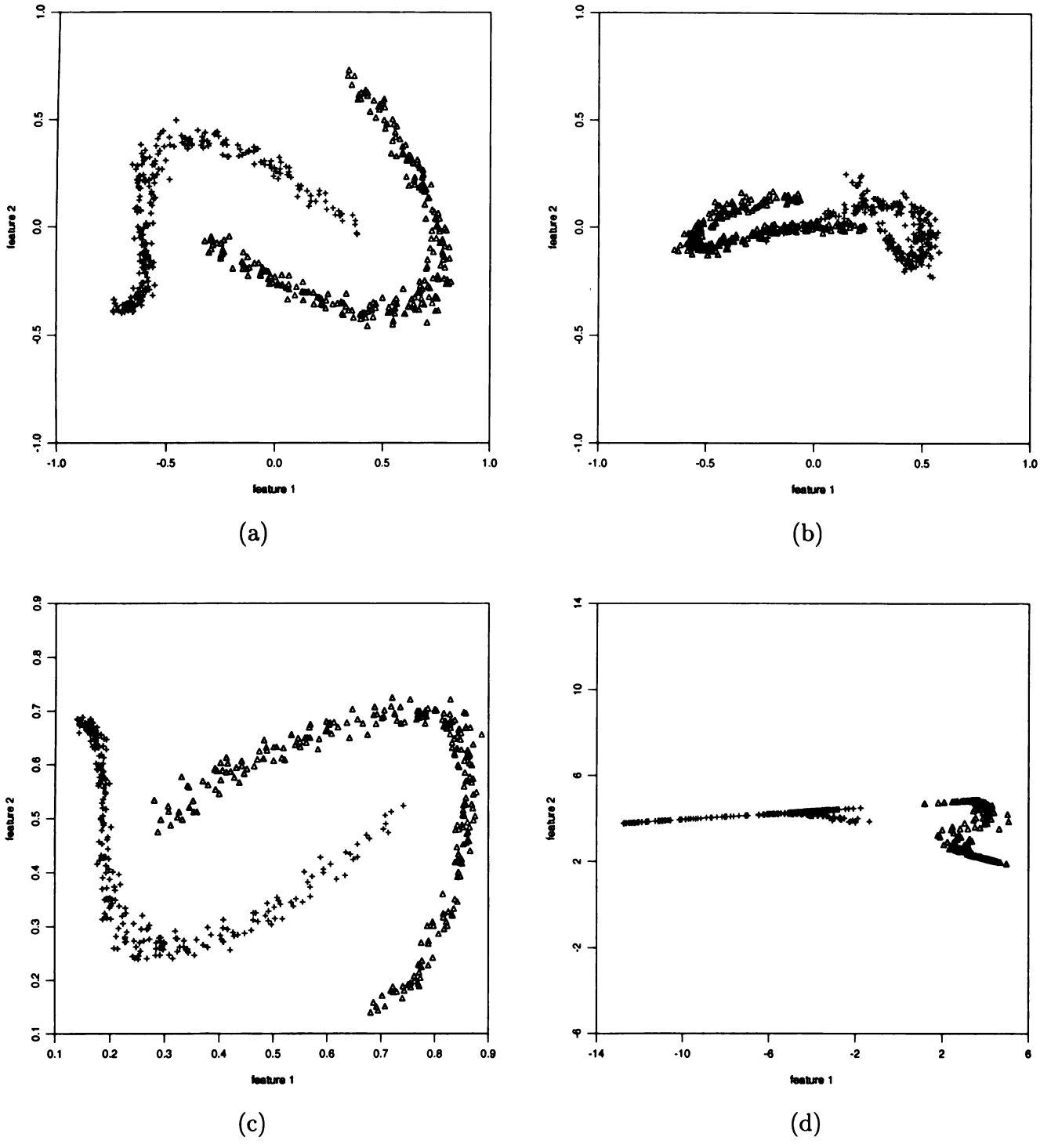


Figure 2.7. 2D projection maps of data set **Curve2** using the four networks. (a) PCA network, (b) LDA network, (c) SAMANN network, and (d) NDA network (2 classes become linearly well-separable).

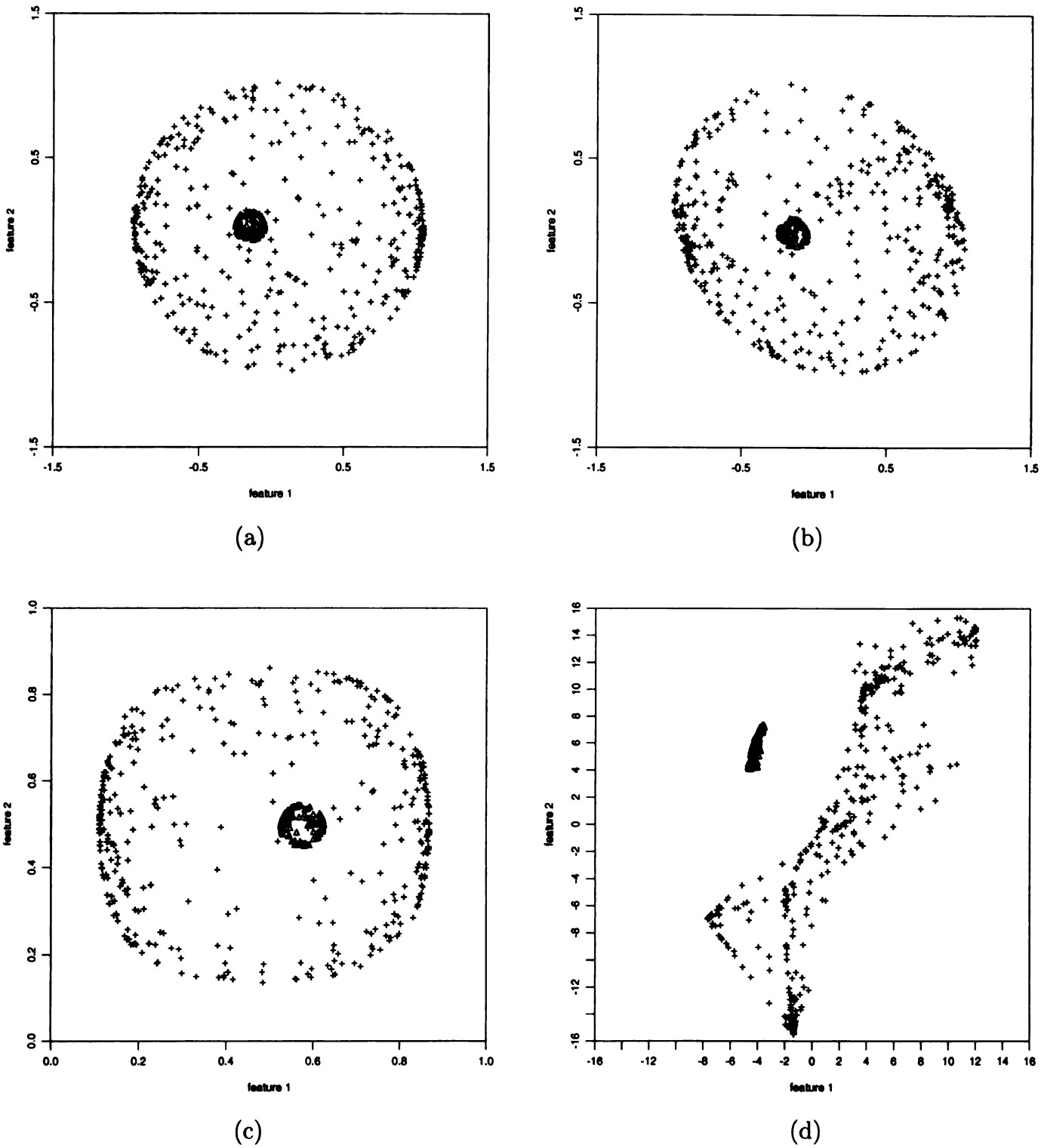


Figure 2.8. 2D projection maps of data set **Sphere2** using the four networks. (a) PCA network, (b) LDA network, (c) SAMANN network, and (d) NDA network (the black blob exclusively contains all the patterns on the surface of the inner sphere).

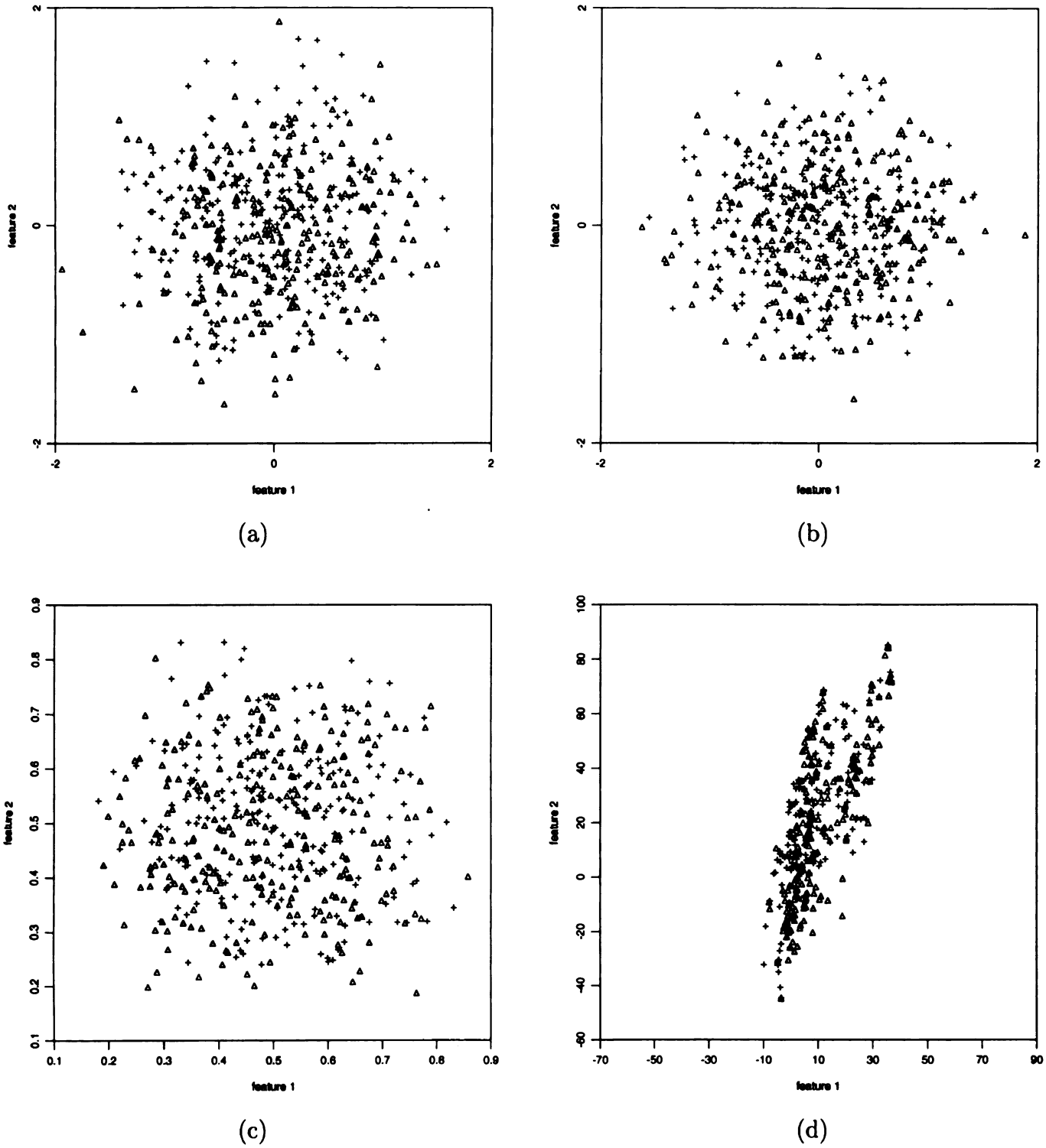


Figure 2.9. 2D projection maps of data set **Random** using the four networks. (a) PCA network, (b) LDA network, (c) SAMANN network, and (d) NDA network.

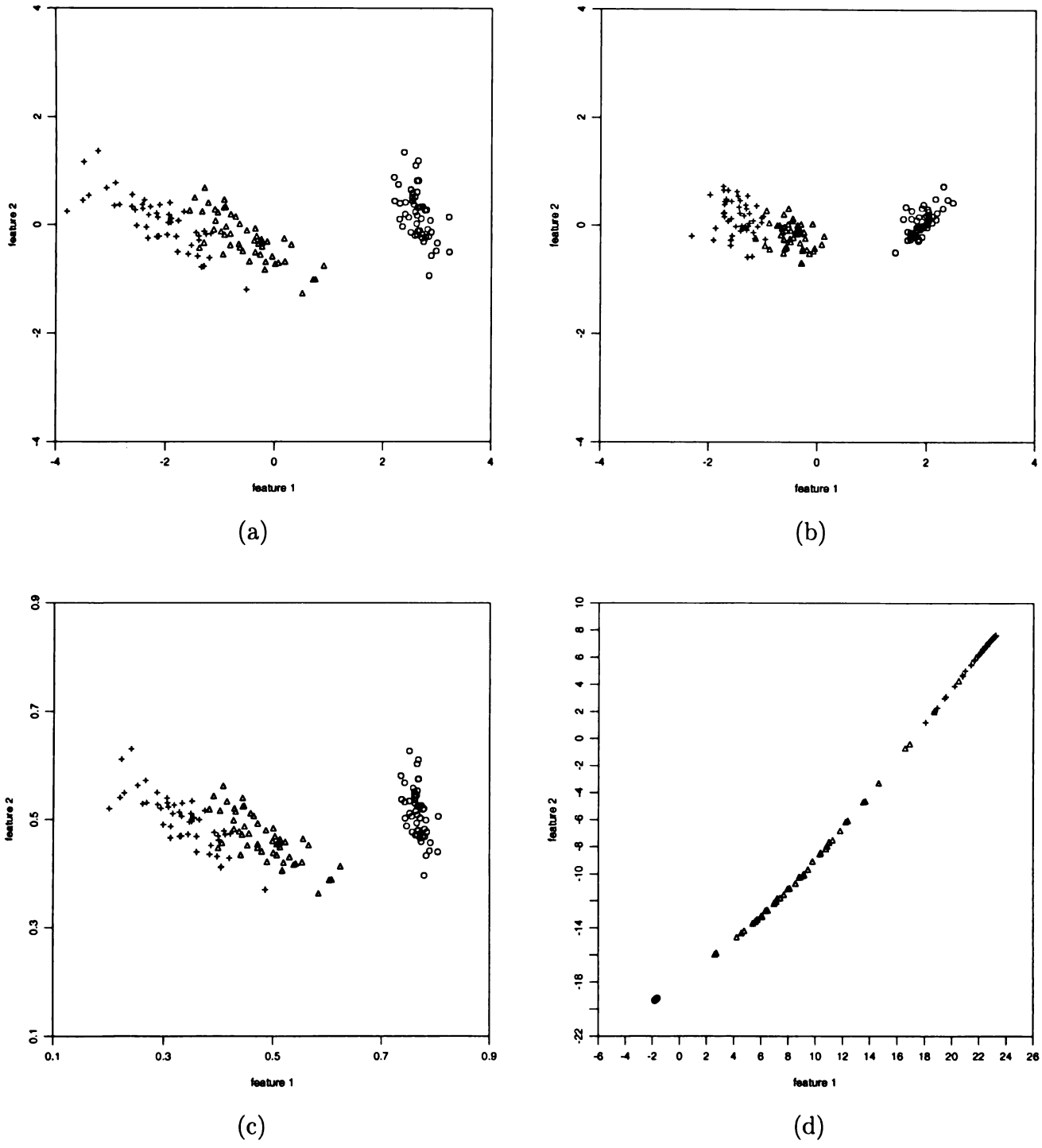


Figure 2.10. 2D projection maps of the IRIS data set using the four networks. (a) PCA network, (b) LDA network, (c) SAMANN network, and (d) NDA network (the black dot on the lower-left corner exclusively contains all the 50 patterns from class 1).

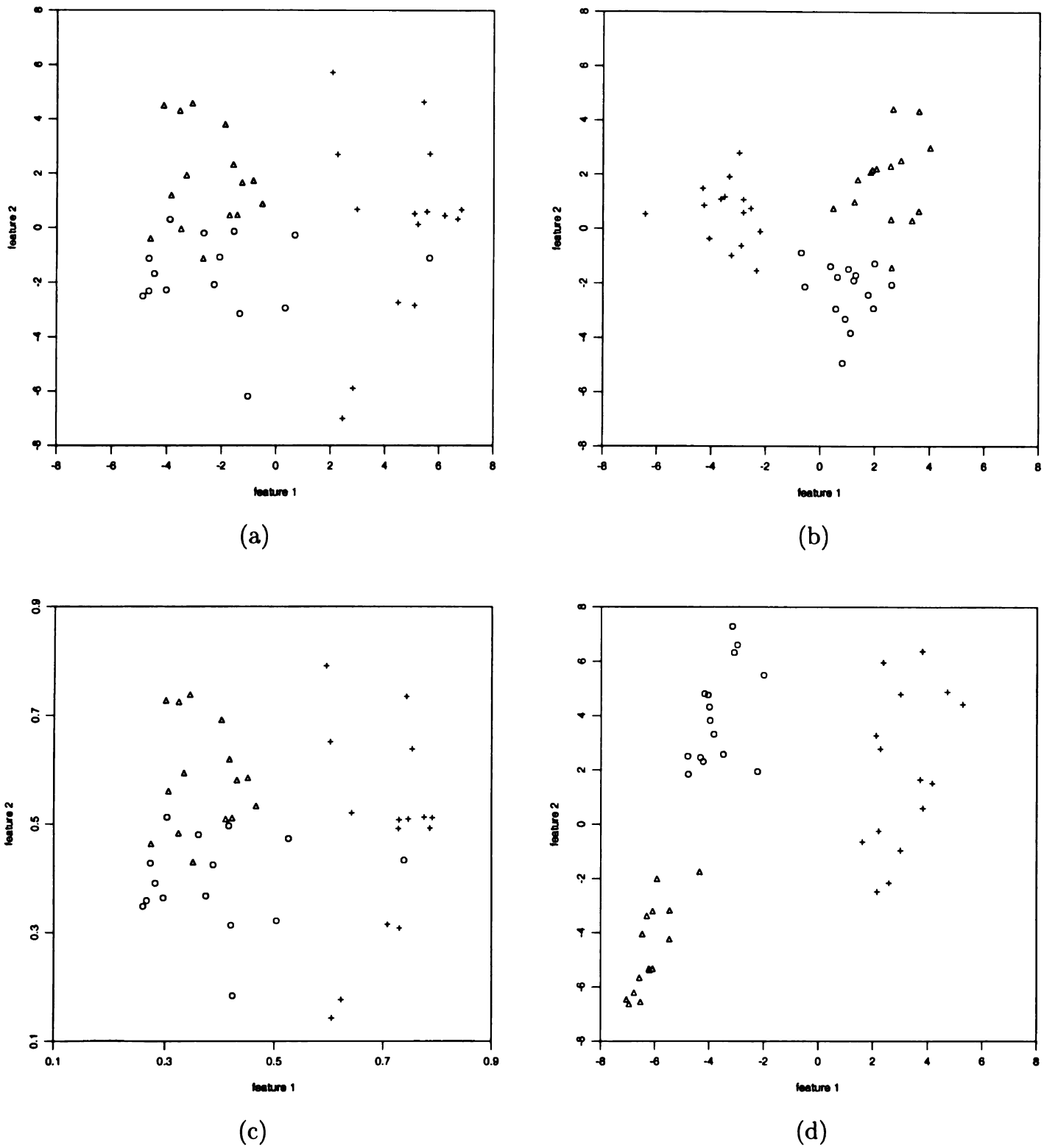


Figure 2.11. 2D projection maps of the 80X data set using the four networks. (a) PCA network, (b) LDA network, (c) SAMANN network, and (d) NDA network.

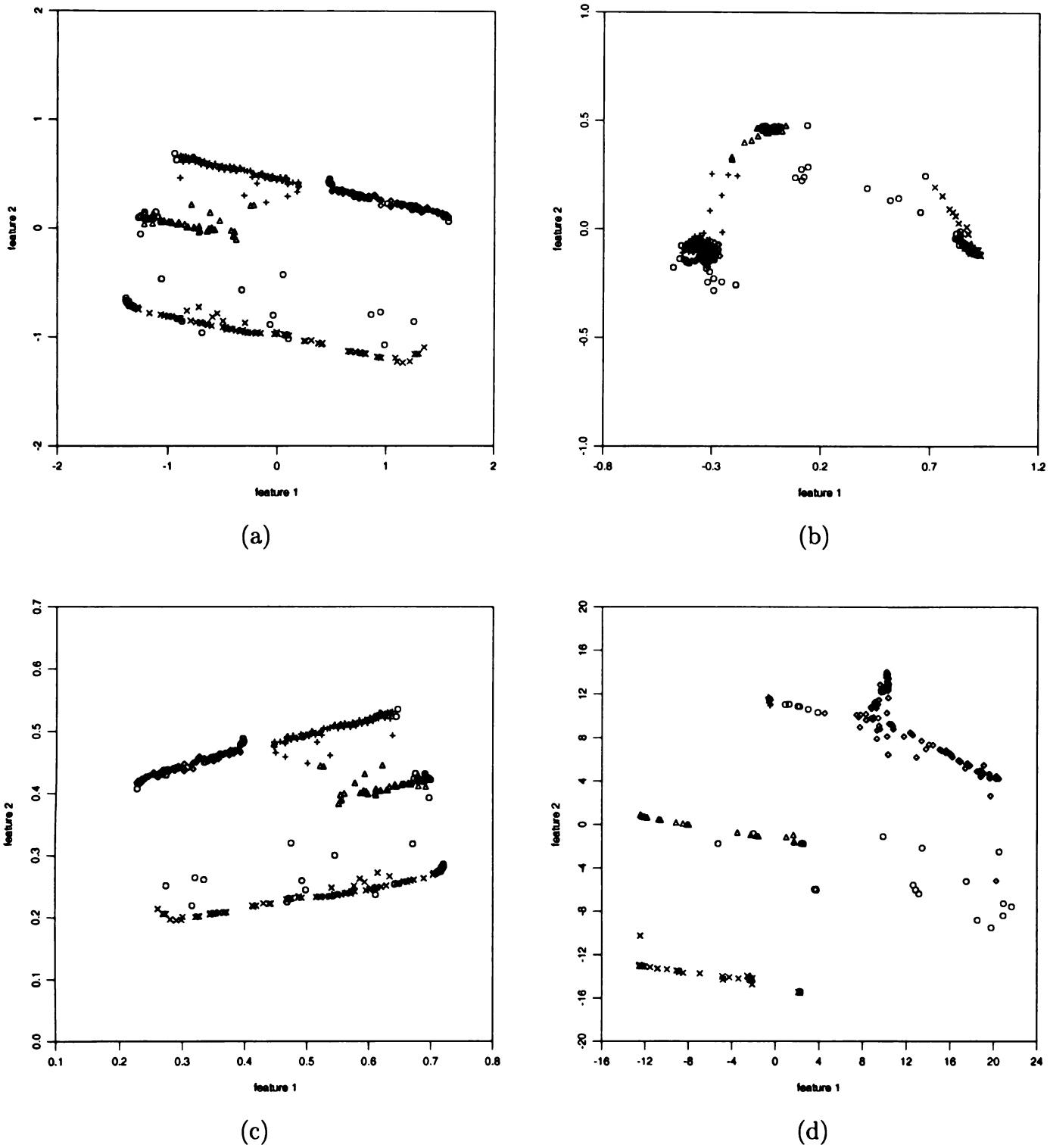


Figure 2.12. 2D projection maps of data set **Range** using the four networks. (a) PCA network, (b) LDA network, (c) SAMANN network, and (d) NDA network.

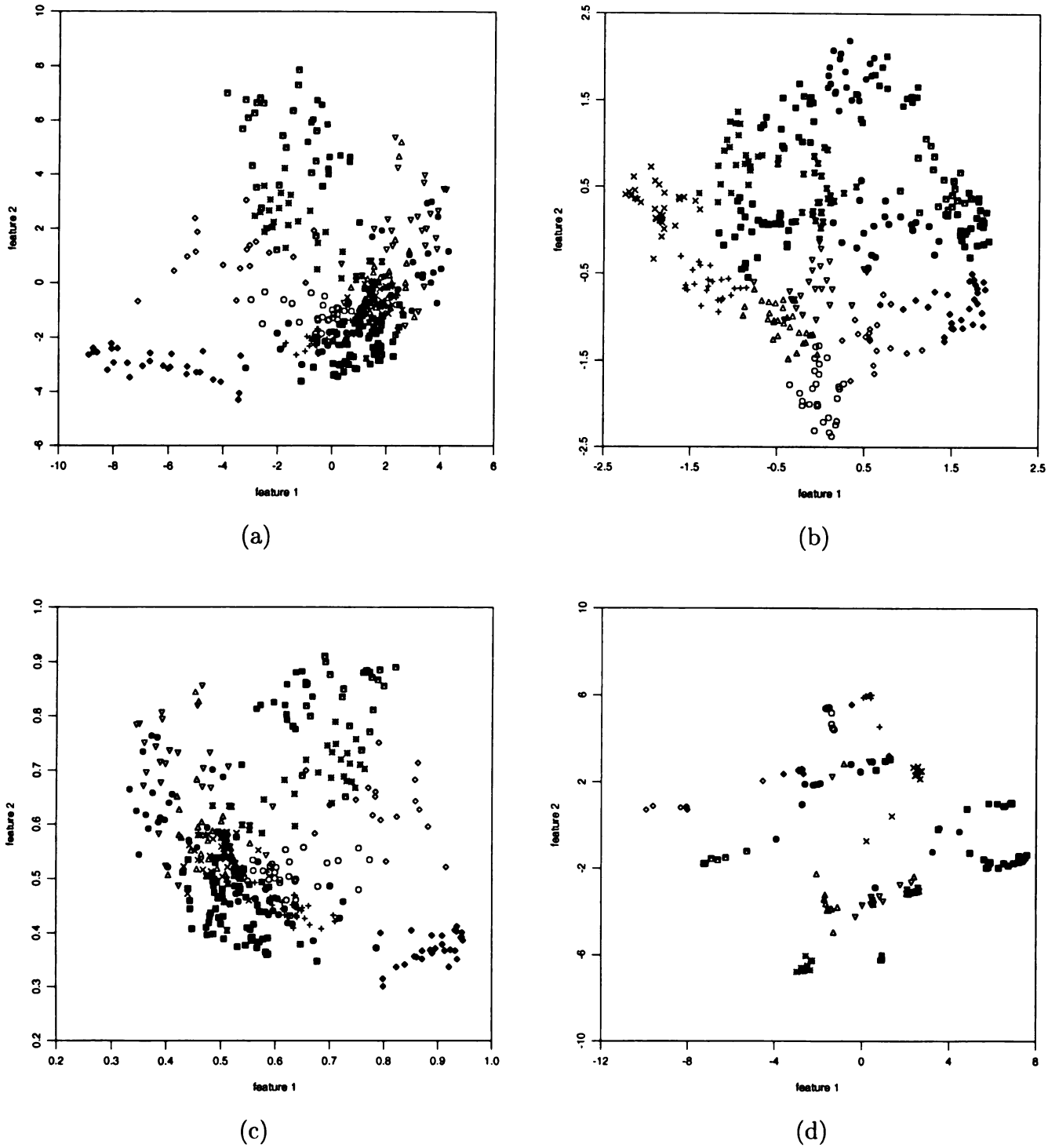


Figure 2.13. 2D projection maps of data set **Texture** using the four networks. (a) PCA network, (b) LDA network, (c) SAMANN network, and (d) NDA network.

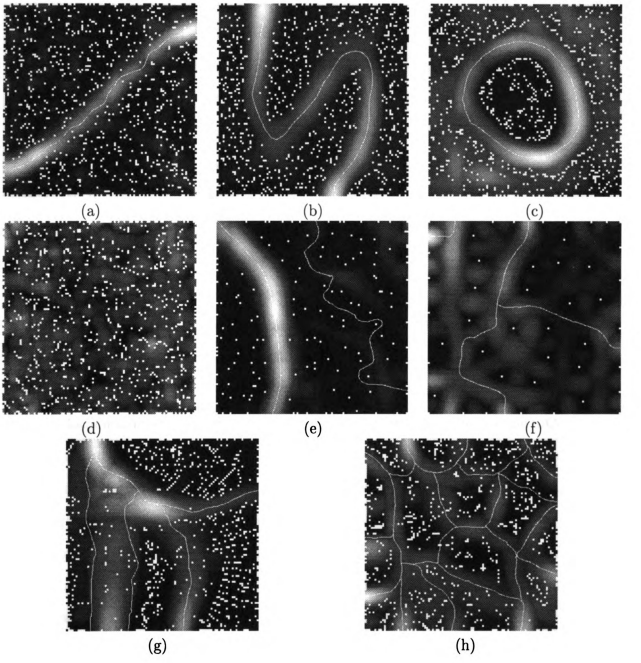


Figure 2.14. The distance images of Kohonen's SOM superimposed by the 2D projection maps and boundaries between classes for the 8 data sets using the NP-SOM network. (a) `Gauss2`, (b) `Curve2`, (c) `Sphere2`, (d) `Random`, (e) `IRIS`, (f) `8OX`, (g) `Range`, and (h) `Texture`.

2.2.3 Evaluation Based on Quantitative Criteria

We have seen various characteristics of the five networks for the purpose of visualizing high dimensional data. In this subsection, we provide a quantitative evaluation of these five networks based on three criteria using the 8 data sets. The dimensionality of the output space is 2 for all the five networks. Every simulation of each of the five networks on each of the 8 data sets was repeated 10 times with different initial random weights in the networks. The average performance over these ten trials is reported in the following tables.

Table 2.3 shows the average values of Sammon's stress of the projections performed by the five networks on the 8 data sets. Among the 5 networks, the SAMANN network performs the best for 7 out of the 8 data sets in terms of Sammon's stress. The exception is the `Curve2` data set on which the PCA network has the lowest Sammon's stress. This happens because the initial configuration generated by the backpropagation algorithm in the SAMANN network is not a perfect PCA map due to the local minimum problem. These results are not surprising because the SAMANN network is designed to minimize the Sammon's stress, while others are not. The PCA network performs the second best on all the 8 data sets (except the `Curve2` data on which it is the best). This is largely because the PCA network performs a linear orthogonal projection which maximally retains the variance of the data. The LDA network is comparable to the PCA network on the `Gauss2`, `Sphere2` and `Random` data sets because for these data sets, the discriminant vectors are similar to the principal vectors (see Figures 2.6(a), 2.6(b), 2.8(a), 2.8(b), 2.9(a), and 2.9(b)). For other data sets, the LDA network generates large values of Sammon's stress because the discriminant vectors are not orthogonal. The NDA network produces large values of Sammon's stress due to the fact that it reorganizes patterns and groups them by categories such that they can be easily separated, resulting in distortions in the

interpattern distances. Some values are extremely large due to the different spreads of the input features and output features. The average performance of the NP-SOM network is similar to that of the LDA network.

Table 2.3. The average values of Sammon's stress for the PCA, LDA, SAMANN, NDA and NP-SOM networks on eight data sets.

Data set		1	2	3	4	5	6	7	8
Sammon's stress, E	PCA	0.151	0.003	0.062	0.361	0.007	0.141	0.009	0.147
	LDA	0.161	0.169	0.077	0.386	0.134	0.289	0.408	0.519
	NDA	1.400	53.78	116.1	335.5	27.15	0.224	146.7	1.513
	SAMANN	0.115	0.005	0.052	0.231	0.007	0.084	0.008	0.084
	NP-SOM	0.346	0.005	0.299	0.954	0.034	0.493	0.069	0.420

Table 2.4 lists the average values of the nearest-neighbor classification error rate P_e^{NN} and relative ratio R_e^{NN} for the five networks on the 8 data sets. The NDA network achieves the best performance in terms of the nearest-neighbor classification error rate among all the networks over all the 8 data sets with the exception of **Range** and **Texture** on which the NP-SOM network performs slightly better. The values of P_e^{NN} on the projected data is comparable to those on the data in the original space (the values of R_e^{NN} are close to 1.0). These results are consistent with the known fact that feedforward network classifiers and nearest-neighbor classifiers have comparable performances [71]. The NP-SOM network also achieves comparable performance with the nearest-neighbor classifier on the original data. This is because in the NP-SOM network, the nearest-neighbor classification is actually done in the quantized feature space with the same dimensionality as the input space rather than in the projected space. The PCA and LDA networks perform worse than the NDA and NP-SOM networks on most of the data sets because they are driven by the global properties of the data, not by the nearest neighbor's category information. Although

the SAMANN network is designed to maximally preserve the interpattern distances, the nearest neighbor classification is not good for some projected data sets obtained by this method since it makes no use of the category information.

Table 2.4. The average values of the nearest neighbor classification error rate P_e^{NN} and relative ratio R_e^{NN} for the PCA, LDA, SAMANN, NDA and NP-SOM networks on eight data sets.

Data set		1	2	3	4	5	6	7	8
P_e^{NN}	Original	0.002	0.000	0.000	0.521	0.040	0.044	0.041	0.036
	PCA	0.005	0.000	0.007	0.479	0.040	0.216	0.083	0.405
	LDA	0.003	0.037	0.009	0.501	0.037	0.067	0.126	0.101
	NDA	0.000	0.000	0.000	0.355	0.035	0.002	0.078	0.080
	SAMANN	0.004	0.000	0.005	0.488	0.051	0.200	0.084	0.405
	NP-SOM	0.003	0.000	0.000	0.507	0.041	0.049	0.073	0.046
R_e^{NN}	PCA	1.003	1.000	1.007	0.972	1.000	1.165	1.040	1.356
	LDA	1.001	1.037	1.009	0.987	0.997	1.022	1.081	1.063
	NDA	0.998	1.000	1.000	0.891	0.995	0.960	1.036	1.042
	SAMANN	1.002	1.000	1.005	0.978	1.011	1.149	1.041	1.356
	NP-SOM	1.001	1.000	1.000	0.991	1.001	1.005	1.031	1.010

The average values of the minimum-distance classification error rate P_e^{MD} and relative ratio R_e^{MD} for the five networks on the 8 data sets are shown in Table 2.5. Comparing the PCA network and the LDA network (both are linear networks), the LDA network performs better than the PCA network over all the eight data sets. For the **IRIS**, **80X** and **Texture** data sets, which are all real data sets, the LDA network achieves significant improvements over the PCA network. Note that the nearest mean classifier is a linear classifier based on global structure of the data set. The LDA network can find the best two features which have the most linear discriminant power. The NDA algorithm is comparable to the LDA for the linearly-separable data sets (**Gauss2**, **IRIS**, **Range**, and **Texture**), but much better for linearly nonseparable data sets (**Curves**, **Sphere2**, and **80X**). In general, the NDA network performs the best

on most of the data sets. The performances of the PCA, SAMANN and NP-SOM networks are comparable.

Table 2.5. The average values of the minimum distance classification error rate P_e^{MD} and relative ratio R_e^{MD} for the PCA, LDA, SAMANN, NDA and NP-SOM networks on eight data sets.

Data set		1	2	3	4	5	6	7	8
P_e^{MD}	Original	0.001	0.133	0.390	0.483	0.080	0.044	0.152	0.065
	PCA	0.001	0.133	0.391	0.458	0.100	0.164	0.187	0.420
	LDA	0.001	0.106	0.387	0.455	0.020	0.022	0.166	0.112
	NDA	0.001	0.006	0.072	0.487	0.058	0.007	0.117	0.100
	SAMANN	0.001	0.136	0.409	0.458	0.093	0.178	0.186	0.413
	NP-SOM	0.031	0.135	0.474	0.506	0.066	0.129	0.158	0.088
R_e^{MD}	PCA	1.000	1.000	1.001	0.983	1.019	1.115	1.030	1.333
	LDA	1.000	0.976	0.998	0.981	0.944	0.979	1.012	1.044
	NDA	1.000	0.888	0.771	1.003	0.980	0.965	0.970	1.033
	SAMANN	1.000	1.003	1.014	0.983	1.012	1.128	1.030	1.328
	NP-SOM	1.030	1.002	1.060	1.016	0.987	1.081	1.005	1.022

Both the PCA and SAMANN networks perform very badly on the **Texture** data in terms of the nearest neighbor and minimum distance classification error rates. This is because the linear dimensionality of the **Texture** data is 15, which is much larger than the number of extracted features (dimensionality of the projected map). However, other networks still perform reasonably well on this data set due to their nonlinearity and the use of category information. We should mention an interesting phenomenon, namely that sometimes the nearest neighbor classifier and minimum distance classifier perform better on projected data than on the original data ($R_e < 1.0$), particularly for the **IRIS** and **BOX** data sets which have a relatively small number of patterns. This phenomenon may be explained by the fact that the effect of "curse of dimensionality" [80] can be reduced by feature extraction or data projection, which are obtained using the entire data set (not just the training data) in our experiments (but classification

errors are evaluated using the "leave-one-out" technique).

2.3 Summary

We have designed the following neural networks for projecting multivariate pattern vectors: linear discriminant analysis (LDA) network, SAMANN network for Sammon's projection, nonlinear projection based on Kohonen's SOM (NP-SOM), and nonlinear discriminant analysis network (NDA) using a feedforward network classifier. A common attribute of these networks is that they all employ adaptive learning algorithms which makes them suitable in some environments where the distribution of patterns in feature space changes with respect to time. The availability of these networks also facilitates hardware implementation of the well-known classical feature extraction and projection approaches. Moreover, the SAMANN network offers the generalization ability of projecting new data, which is not present in the original Sammon's projection algorithm; the NDA method and NP-SOM network provide new powerful approaches for visualizing high dimensional data.

The performances of the five networks (PCA, LDA, SAMANN, NDA and NP-SOM) for feature extraction and data projection have been evaluated based on visual inspection of the projection maps and on three numerical criteria on 8 data sets. Based on our experimental results and analysis, we draw the following general conclusions: (i) the NP-SOM network has a good performance for data visualization. It reveals the cluster tendency in the multivariate data very well due to the role of the distance image; (ii) the SAMANN and PCA networks preserve the data structure, cluster shape, and interpattern distances better than the LDA, NDA and NP-SOM networks; (iii) the NDA network is superior to all the other networks for classification purposes, especially when the data are not linearly-separable, but it severely distorts

the structure of the data and the interpattern distances; (iv) there is no general conclusion on whether the PCA or the LDA is better for preserving the nearest-neighbor category information; (v) the LDA network outperforms the PCA, SAMANN and NP-SOM networks in the sense of nearest mean classification error; (vi) feature extraction can help to reduce or eliminate the "curse of dimensionality"; (vii) it is often necessary to use more than one method (network) in order to reveal various properties of multivariate data; and (viii) knowledge of the structure of the data set obtained from the projection maps can guide in choosing a proper classification and clustering tools.

Our simulation results are consistent with the high-level analysis based on our knowledge of the traditional feature extraction and data projection approaches. This is one of the main advantage of studying the links between neural networks and traditional pattern recognition and data analysis approaches.

CHAPTER 3

Neural Network-Based K-Nearest Neighbor Classifier

We propose a neural network architecture to implement the well-known k-nearest neighbor (k-NN) classifier. This neural network architecture employs a k-Maximum network which has some advantages over the "winner-take-all" type of networks and other techniques used to select the maximum input. This k-Maximum network has fewer interconnections than other networks, and is able to select exactly k maximum inputs as long as its k^{th} and $(k+1)^{th}$ maximum inputs are distinct. The classification performance of the k-NN neural network classifier is exactly the same as the traditional k-NN classifier, hence the well-established characteristics of the traditional k-NN classifier can be directly applied to the k-NN neural network classifier. However, the parallelism of the network greatly reduces the computational requirement of the traditional k-NN classifier. Unlike the multilayer perceptrons which involve slowly-converging back-propagation algorithms, the basic k-NN neural network classifier only needs a one-pass training algorithm. Three modified k-NN rules, including the condensed nearest neighbor rule, reduced nearest neighbor rule and edited nearest neighbor rule, have also been incorporated in the learning algorithms of the k-NN neural network in order to reduce the number of nodes.

3.1 K-Nearest Neighbor Classifier

The k-nearest neighbor classifier [34, 45] is a well-known and commonly used classifier in statistical pattern recognition. It is a nonparametric classifier which makes no assumption about the underlying pattern distributions.

Let the n training pattern vectors be denoted as: $\mathbf{x}^{(i)}(l)$, $i = 1, 2, \cdots, n_l$, $l = 1, 2, \cdots, c$, where n_l is the number of training patterns from class ω_l, $\sum_{l=1}^{c} n_l = n$, and c is the total number of categories.

The k-nearest neighbor classifier examines the k nearest neighbors of a test pattern $\mathbf{x}$ and assigns it to the pattern class most heavily represented among the k neighbors. In mathematical terms, let $K_l(k, n)$ be the number of patterns from class ω_l among the k nearest neighbors of pattern $\mathbf{x}$. The nearest neighbors are computed from the n training patterns. The k-NN decision rule $\delta(\mathbf{x})$ is defined as

$$\delta(\mathbf{x}) = \omega_j \text{ if } K_j(k, n) \geq K_i(k, n) \text{ for all } i \neq j$$

The Euclidean distance metric is commonly used to calculate the k nearest neighbors. Other metrics, such as the optimal global nearest neighbor metric proposed by Fukunaga and Flick [46], can also be used. Let $D(\mathbf{x}, \mathbf{x}^{(i)})$ denote the Euclidean distance between two pattern vectors, $\mathbf{x}$ and $\mathbf{x}^{(i)}$, then

$$D(\mathbf{x}, \mathbf{x}^{(i)}) = \sum_{j=1}^{d} (x_j - x_j^i)^2 = -2M(\mathbf{x}, \mathbf{x}^{(i)}) + \sum_{j=1}^{d} x_j^2, \tag{3.1}$$

where

$$M(\mathbf{x}, \mathbf{x}^{(i)}) = \sum_{j=1}^{d} x_j x_j^i - 1/2 \sum_{j=1}^{d} (x_j^i)^2, \tag{3.2}$$

and d is the number of features. We define $M(\mathbf{x}, \mathbf{x}^{(i)})$ as the matching score between the test pattern $\mathbf{x}$ and the training pattern $\mathbf{x}^{(i)}$. If all patterns are normalized to

unit vectors, the second term in Eq. (3.1) will be a constant and can be ignored. So, finding the minimum Euclidean distance is equivalent to finding the maximum matching score. Computation of the matching scores can be easily done using a neural network architecture.

A severe drawback of the k-NN classifier is that it requires a large amount of computation time. In order to find the k nearest neighbors of an input pattern, we have to compute its distances to all the stored training patterns. Because of this computational burden, the k-NN classifier is not very popular where real-time requirements have to be met. Two approaches are commonly used to cut down the computational effort needed to compute the nearest neighbors: i) Several efficient algorithms for computing nearest neighbors, such as the branch and bound algorithm [48], have been proposed; and ii) many modifications to the k-NN classifier have been proposed in the literature to eliminate a large number of "redundant" training patterns, such as, the condensed nearest neighbor (CNN) rule [57], the reduced nearest neighbor (RNN) rule [54], and the edited nearest neighbor (ENN) rule [183, 29]. In Section 3.3, we will show that all these methods can be incorporated in the learning algorithms of the k-NN neural network classifier.

3.2 A Neural Network Architecture for the k-NN Classifier

The k-NN classifier can be directly mapped into a neural network architecture [84]. Figure 3.1 shows the schematic diagram of this neural network architecture. This architecture is similar to the feature map classifier and the kernel-based classifier [72, 110]. However, the k-NN neural network classifier only needs a one-pass training algorithm which requires only the initial setting of the connection weights.

As shown in Figure 3.1, the k-NN neural network consists of four basic building

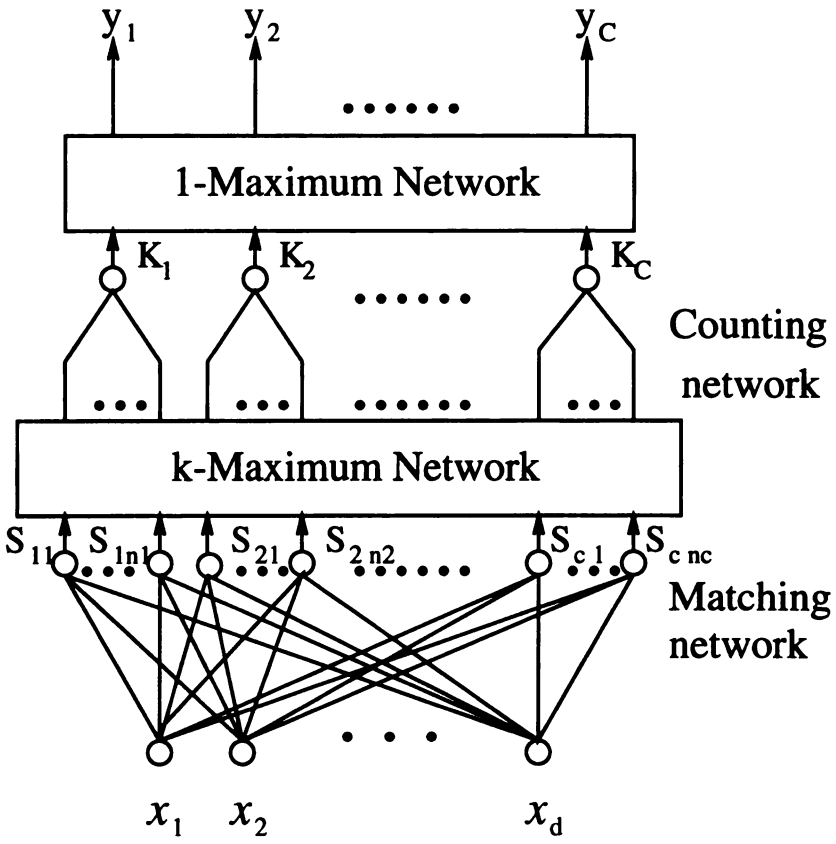


Figure 3.1. A neural network architecture of the k-NN classifier

blocks: Matching network, k-Maximum network, Counting network, and 1-Maximum network. The functions performed by these subnetworks are described below.

Matching network

As the name implies, the Matching network calculates the matching scores between the input pattern and each of the stored patterns (training patterns), $M(\mathbf{x}, \mathbf{x}^{(i)})$. The network has d input nodes corresponding to a d-dimensional pattern vector, and n output nodes generating n matching scores, one for each training pattern. The training patterns are stored in the interconnections of this network in the following way. We label the n output nodes of the Matching network in the same order as the training patterns, S_{i_l}, $i_l = 1, 2, \cdots, n_l$, $l = 1, 2, \cdots, c$. Let W_{j,i_l} denote the weight assigned to the connection between the input node j and the output node i_l,

$j = 1, 2, \cdots, d,\ i = 1, 2, \cdots, n_l,\ l = 1, 2, \cdots, c$. Then we set $W_{j,i_l} = x_j^{(i)}(l)$, and the bias, b_{i_l}, of node S_{i_l} to $b_{i_l} = -\sum_{j=1}^{d}(x_j^{(i)}(l))^2$. So, the output of node S_{i_l} is equal to the matching score $M(\mathbf{x}, \mathbf{x}^{(i)}(l))$.

The outputs of the Matching network provide a set of initial values for the k-Maximum network.

K-Maximum network

The function of the k-Maximum network is to select k maximum matching scores among the n outputs of the Matching network. Details of this network are given in Section 3.1.

Counting network

The Counting network provides a count of how many nodes in each pattern class are excited, i.e., it calculates $K_l(k, n)$. All the interconnection weights in this network are set to "1". The specific activation functions used in nodes in this network are not important as long as they are the same monotonically increasing function, because the next stage determines the maximum output of the Counting network.

1-Maximum network

The function of the 1-Maximum network is to select the maximum output of the Counting network, i.e., it finds the maximum of $K_l(k, n)$, $l = 1, \cdots, c$. There are c output nodes corresponding to c pattern classes. The 1-Maximum network is a special case of the k-Maximum network with $k = 1$.

3.2.1 Design of the k-Maximum Network

In statistical pattern recognition, we frequently need to compute a maximum or a minimum value (e.g., maximum *a posteriori* density function, minimum Mahalanobis distance). Many neural networks, such as ART models, Kohonen's self-organizing network, Hamming network, kernel-based classifiers, etc., often employ a subnetwork

to select the most "active" node. Several different types of neural networks can be used to perform this operation. In this section, we briefly compare three different types of networks and propose a simple, stable, and efficient k-Maximum network which is best suited for the k-NN neural network classifier.

"winner-take-all" type of networks

The "winner-take-all" networks, such as the MAXNET used in the Hamming network [109], are commonly used networks to select the maximum value. These networks mimic the lateral inhibition observed in the biological neural networks [90] and can be generalized to a k-Maximum network which selects the k maximum inputs [84]. When the k-Maximum network reaches an equilibrium state, hopefully, exactly k nodes corresponding to k maximum initial input values will be "on". Unfortunately, simulation experiments [84] showed that the actual number of "on" nodes at the equilibrium state is often not equal to the specified value of k, especially when the network involves a large number of nodes. Table 3.1 shows the simulation results on two different sized networks: $n = 10$ and $n = 100$, where n is the number of nodes in the k-Maximum network. The value k_n, $n = 10, 100$, is the average number of nodes which are "on" when the network reaches an equilibrium state over ten trials with different input values. In our experiments, the local capacities and local resistances, $C_i = R_i, i = 1, 2, \cdots, n$, were chosen to be "1". From Table 3.1, we see that when the number of nodes is small, the network is likely to reach the global minimum state. However, when the number of nodes is large, the network can reach a local minimum. Note that $k_{100} < k$ for all k. Another problem is that the k-Maximum network is

Table 3.1. Actual number of "on" nodes

k	1	2	3	4	5	6	7	8	9	10
k_{10}	1	2	3	4	5	6	7	8	9	10
k_{100}	0	0	1	2	3	4	4	5	7	8

nearly fully-connected (no self-feedback), so there are $n^2 - n$ connections. This makes the construction of a k-Maximum network with a large number of nodes infeasible.

An additional problem with the "winner-take-all" network is that it only selects the nodes with maximum input values, but the maximum input values are neither kept nor passed through the network. Some modifications of the k-NN classifier such as the distance-weighted k-NN classifier [5] do need the distances of the test pattern to its k nearest neighbors.

Comparator-based technique

This technique [109] uses a comparator subnet as the building block, which uses threshold logic nodes to select and forward the maximum value. No feedback connections are needed in this kind of network and all the weights are fixed. The advantage is that the maximum value is passed through the network. However, there are two disadvantages. First, the network cannot select more than one maximum input. The other drawback is that the network involves a large number of nodes if it is used to select the maximum from n inputs, because in this case, comparator subnets must be layered into roughly $\log_2(n)$ layers.

Threshold-based techniques

Here the maximum input is selected by using an array of hard-limiting nodes with internal thresholds set to the desired threshold values [109]. Outputs of these nodes will be -1 unless the inputs exceed the threshold values. Alternatively, thresholds could be set adaptively using a common inhibitory input fed to all the nodes. This threshold could be ramped up or down until the output of only one node is positive. Note that the input values do not pass through the network. This method can be easily generalized to select k maximum inputs. The problem is how to adjust the thresholds adaptively.

***K*-Maximum network**

We have designed a k-Maximum network which uses a common feedback node (the

right most node in Figure 3.2) to adjust adaptively the input signals instead of the internal thresholds. Figure 3.2 shows the network architecture to implement the adaptive selection of k maxima. In Figure 3.2, all the nodes which connect to inputs,

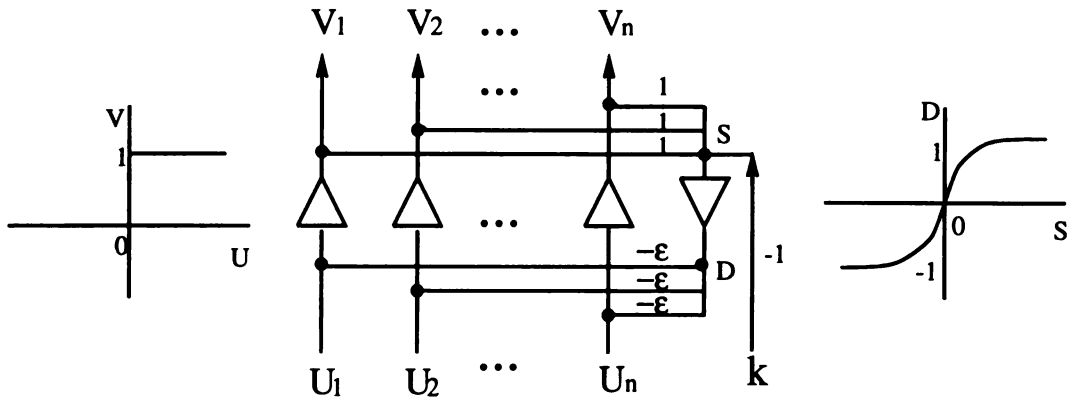


Figure 3.2. k-Maximum network

$U_1, U_2, \cdots, U_n$, use the same hard-limiting activation function with internal threshold set to zero. The feedback node uses a sigmoid function, $tanh(\cdot)$. All the connection weights between the feedback node and the outputs of all other nodes are set to 1, and the weight on connection to the input k is set to -1, so that the input of the feedback node is a summation of the outputs of all other nodes with bias $-k$. In other words, $S = \sum_{i=1}^{n} V_i - k$. The feedback node monitors the states of all other nodes, and its output serves as a common adaptive signal fed to all other nodes through an inverse weight $-\varepsilon$. When the summation of the outputs of all other nodes is greater than k, the output of the feedback node is positive, so it tries to inhibit more nodes. On the other hand, when the summation of the outputs of all other nodes is less than k, it serves the role of an excitatory signal which tries to turn more nodes "on". Only when the summation of the outputs of all other nodes is equal to k, which is the desired situation, will the network reach the equilibrium state. Therefore, we get exactly k nodes "on" when the network reaches the equilibrium state. The weight ε

controls the speed of convergence and also determines the stability of the network. Large values of ε can lead the network to quickly approach the equilibrium state at first, but may cause oscillations near the equilibrium state. Small values of ε will guarantee the stability of the k-Maximum network.

The k-Maximum network has many desirable properties. First, we avoid the local minimum problem which is always associated with the "winner-take-all" types of networks. Secondly, it greatly reduces the number of connections which is desirable for hardware implementation. Besides, the k-Maximum network proposed here has fewer nodes and connections than the comparator-based network. Unlike the "winner-take-all" type of network whose connection weights are dependent on the value of k, the k-Maximum network is more flexible because k is the external input.

3.2.2 Performance of the k-NN Neural Network Classifier

Because the k-Maximum network can select exactly k maximum inputs as long as its k^{th} and $(k+1)^{th}$ maximum inputs are distinct, the classification rate of the k-NN neural network classifier is the same as the traditional k-NN classifier. However, due to the parallelism of the network, the large amount of computation involved in the traditional k-NN classifier is drastically reduced.

Table 3.2 lists the classification error rates of the k-NN classifiers on three well-known data sets [81]: 80X, IRIS, IMOX. The 80X and IMOX data sets consist of 8-dimensional patterns extracted from the Munson's handprinted character database on the characters, 8, 0, and X and I, M, O, and X, respectively. The 80X data set contains 15 patterns from each of the 3 classes, and IMOX data set contains 48 patterns from each of the 4 classes. The IRIS data set consists of 150 4-dimensional patterns from 3 classes. It contains 4 measurements on 50 flowers from each of the 3 species of the Iris flower. The error rates are estimated using the "leave-one-out" method.

Table 3.2. Error rates of k-NN classifiers

k	1	3	5	7	9
P_e% (80X)	4.4	6.7	8.9	13.3	13.3
P_e% (IRIS)	4.0	4.0	3.3	3.3	3.3
P_e% (IMOX)	5.7	4.2	5.2	6.8	6.8

3.3 Modified k-NN algorithms

We note that the total number of nodes required in the design of a k-NN neural network is $N = 2(n + c + 1)$, where n is the number of training patterns and c is the number of categories. When n is large which is often the case in practice, the network will require a large number of nodes. Several approaches to reduce the number of stored training patterns have been proposed. These include, Condensed Nearest Neighbor rule [57], Reduced Nearest Neighbor rule [54], and Edited Nearest Neighbor rules [183, 29]. These methods can be used for "preprocessing" the training data. In this section, we will show that these methods can also be incorporated in the learning algorithms of the k-NN neural network. Note that the original k-NN neural network needs only a one-pass training procedure.

3.3.1 Condensed nearest neighbor algorithm

The condensed nearest neighbor algorithm [57] begins with the set T of all the training patterns and creates a consistent subset T_{CNN} that may or may not be minimal. Let $\{(\mathbf{x}_1, \theta_1), (\mathbf{x}_2, \theta_2), \cdots, (\mathbf{x}_n, \theta_n)\}$ be the n pairs of training pattern vectors ($\mathbf{x}$) and the corresponding class labels (θ), and $\delta(\mathbf{x}_j, T_{CNN})$ be the nearest neighbor decision given by the k-NN neural network based on the training pattern set T_{CNN}.

The k-NN neural network which incorporates the condensed nearest neighbor rule begins with one node in each subnetwork. The node in the Matching network stores

the first training pattern $\mathbf{x}_1$ (it can be any other pattern also). Then, the k-NN neural network classifies each pattern in the set $(T - T_{CNN})$ sequentially. Every time a pattern, say $\mathbf{x}_j$, is misclassified, this pattern is added to T_{CNN}, and the Matching network adds a new node to store this pattern. Correspondingly, the k-Maximum network also adds a new node to collect the matching score. A node corresponding to the label of the training pattern $\mathbf{x}_j$ in the Counting network and 1-Maximum network is added, provided no pattern from that class has been presented yet. Connections between newly added nodes and others are established according to the k-NN neural network architecture. This process is repeated until all the patterns in the set $(T - T_{CNN})$ are correctly classified or $(T - T_{CNN})$ is empty. The CNN algorithm is given below.

Table 3.3 lists the classification error rates of the 1-NN classifier on T and T_{CNN}, the number of original training patterns (n_T), the average number of patterns in T_{CNN} ($n_{T_{CNN}}$) obtained by the CNN rules in 10 trials with different random orderings of the input pattern, and the average percentage of training patterns used in the CNN rule. Again, we perform experiments on the three well-known data sets: 80X, IRIS, IMOX. The "leave-one-out" error estimation technique is used. We see from Table 3.3 that the classification error rate of the CNN rule is higher than that of the 1-NN classifier. The large difference between the error rates of the 1-NN and CNN rules for the 80X data is due to the small number of training patterns. Still, the CNN rule selects only a small portion of the original training set.

Table 3.3. Error rates and number of training patterns retained in the CNN rule

Data Set	n_T	$n_{T_{CNN}}$	$P_e\%(T)$	$P_e\%(T_{CNN})$	$n_{T_{CNN}}/n_T$
80X	44	12	4.4	10.2	27.3%
IRIS	149	18	4.0	5.5	12.1%
IMOX	191	35	5.7	7.1	18.3%

3.3.2 Reduced nearest neighbor rule

The subset T_{CNN} obtained from the condensed nearest neighbor rule is not necessarily a minimal consistent subset. The resulting subset depends on the order in which the patterns are presented. Gates [54] proposed an extension to the CNN called the reduced nearest neighbor rule (RNN) whose objective is to minimize the number of training patterns in the consistent subset and, thus, approach the ideal of a minimal consistent subset of training patterns.

Table 3.4 shows the classification error rate of the RNN rule, and the number of patterns in the T_{RNN} set. The values presented in Table 3.4 for the RNN rule are averaged over 10 trials. Note that the RNN rule also depends on the order in which the patterns are presented. However, the RNN rule can further reduce the number of patterns in the CNN rule without significantly sacrificing the classification accuracy of the CNN rule.

Table 3.4. Error rates and number of training patterns in the RNN rule

Data Set	n_T	$n_{T_{RNN}}$	$P_e\%(T)$	$P_e\%(T_{RNN})$	$n_{T_{RNN}}/n_T$
80X	44	10	4.4	12.0	22.7%
IRIS	149	16	4.0	6.3	10.7%
IMOX	191	30	5.7	7.8	15.7%

3.3.3 Edited nearest neighbor rule

The drawback of the condensed nearest neighbor rule and the reduced nearest neighbor rule is that they cannot eliminate or reduce the affects of the outliers. After the training patterns are condensed and reduced, one cannot use k-nearest neighbor

classifiers with a large value of k. Wilson [183] and Devijver and Kittler [29] proposed another modification of the k-NN rule called the edited nearest neighbor rule (ENN). The idea here is to eliminate patterns which are sparsely represented among pattern classes. This type of editing does not seek a consistent subset of the original training patterns, but tries to smooth the data by eliminating those patterns which, by chance, are in the "wrong" parts of the pattern space. It has been proven that the performance of the edited nearest neighbor rule is very close to the Bayes optimal rule, provided that the learning set is sufficiently large. Kraaijveld and Duin [99] investigated the behavior of the back-propagation learning algorithm on the edited data set and found that the editing algorithm can effectively lead to a near Bayes performance.

The basic editing procedure is relatively simple. For a suitable value of k, eliminate all the training patterns which are misclassified by a k-NN classifier. The remaining patterns can, therefore, be used in the 1-NN classifier. This procedure can be incorporated in the k-NN neural network by removing all the nodes corresponding to misclassified training patterns. The edited set of training patterns does not depend on the order in which the patterns are inspected.

Table 3.5 shows the classification error rates of the 1-NN classifier on the original training set and on the T_{ENN}, which was obtained by the ENN rule with $k = 1$, and the numbers of patterns in T and T_{ENN}. Only a small number of patterns are removed as outliers. The classification error rate remains essentially the same for the 80x and IMOX data sets, but decreases slightly for the IRIS data set.

It is easy to calculate how many nodes have been removed and what is the reduction rate in terms of the number of training patterns: $N - N_* = 2(n - n_{T_*})$, and $\frac{N-N_*}{N} = \frac{n-n_{T_*}}{n+c+1}$, where N (N_*) and n (n_{T_*}) are the total number of nodes in the network and the number of training patterns, respectively. The symbol "$*$" represents one of the three modified k-NN rules: CNN, RNN and ENN. Note that the reduction in number

Table 3.5. Error rates and number of training patterns in the ENN rule

Data Set	n_T	$n_{T_{ENN}}$	$P_e\%(T)$	$P_e\%(T_{ENN})$	$n_{T_{RNN}}/n_T$
80X	44	41	4.4	4.4	93.2%
IRIS	149	143	4.0	3.3	96.0%
IMOX	191	181	5.7	5.7	94.8%

of nodes and the reduction rate are data-dependent. Table 3.6 shows these values on three data sets. We can see that the CNN and RNN rules have high reduction rates, but the ENN rule has very low reduction rates.

Table 3.6. Reductions in number of nodes and reduction rates of the CNN, RNN and ENN rules on three data sets

Data Set	N	$N - N_{CNN}$	$\frac{N-N_{CNN}}{N}$	$N - N_{RNN}$	$\frac{N-N_{RNN}}{N}$	$N - N_{ENN}$	$\frac{N-N_{ENN}}{N}$
80X	96	64	66.7%	68	70.8%	6	6.3%
IRIS	306	262	85.6%	266	86.9%	12	3.9%
IMOX	392	312	79.6%	322	82.1%	20	5.1%

3.4 Summary

We have presented a neural network architecture to implement the well-known k-NN classifier. Although the k-NN neural network classifier and the traditional k-NN classifier have the same classification performance, the large amount of computation required in the traditional k-NN classifier is reduced due to the parallelism of the network. However, the well-established characteristics of the traditional k-NN classifier, such as the bound on its asymptotic error probability, can be directly applied to the

k-NN neural network classifier. Some modified k-NN rules have also been incorporated in the learning algorithms of the k-NN neural network classifier to reduce the total number of nodes without significantly sacrificing the classification accuracy.

The k-Maximum network, which is used in the proposed k-NN architecture, has many advantages over the "winner-take-all" type of networks and other techniques in selecting k maximum inputs. It has the properties of adaptivity, flexibility, and fewer connections, and it guarantees the selection of exactly k "on" nodes as long as its k^{th} and $(k+1)^{th}$ maximum inputs are distinct.

CHAPTER 4

A Network for Detecting Hyperellipsoidal Clusters (HEC)

We propose a self-organizing network (HEC) for detecting hyperellipsoidal clusters. The HEC network consists of two layers. The first layer employs a number of principal component analysis subnetworks which are used to estimate the hyperellipsoidal shapes of currently-formed clusters. The second layer then performs a competitive learning using the cluster shape information provided by the first layer. The HEC network performs a partitional clustering using the proposed regularized Mahalanobis distance. This regularized Mahalanobis distance is designed to deal with the problems in estimating the Mahalanobis distance when the number of patterns in a cluster is less than (ill-posed problem) or not considerablely larger than (poorly-posed problem) the dimensionality of the feature space. This regularized distance also achieves a tradeoff between hyperspherical and hyperellipsoidal cluster shapes so as to prevent the HEC network from producing unusually large or unusually small clusters. The significance level of the Kolmogrov-Smirnov test on the distribution of the Mahalanobis distances of patterns in a cluster to the cluster center under the Gaussian cluster assumption is used as a compactness measure of the cluster. The HEC network has been tested on a number of artificial data sets and real data sets. We also

apply the HEC network to texture segmentation problems. Experiments show that the HEC network can lead to a significant improvement in the clustering results over the K-means algorithm (and other partitional clustering algorithms and competitive learning networks) with the Euclidean distance measure. Our results on real data sets also indicate that hyperellipsoidal shaped clusters are often encountered in practice.

4.1 Hyperspherical Versus Hyperellipsoidal Clusters

Given a set of n patterns in a d-dimensional space, $\{\mathbf{x}_i \mid \mathbf{x}_i \in \mathcal{R}^d,\ i = 1, 2, \cdots, n\}$, the objective of a partitional clustering algorithm is to determine an assignment (or a partition) $\{M_{ij} | M_{ij} \in \{0, 1\}; i = 1, 2, \cdots, n;\ j = 1, 2, \cdots, K\}$, such that some cost function is minimized, where $M_{ij} = 1$ if pattern $\mathbf{x}_i$ is assigned to the j^{th} cluster, and $M_{ij} = 0$ otherwise; K is the number of clusters which is usually specified by the user*. For the unique assignment (non-overlapping partition), $\sum_{j=1}^{K} M_{ij} = 1,\ i = 1, 2, \cdots, n$. In the soft clustering approaches, such as fuzzy c-means [15], M_{ij} can be any continuous value in $[0, 1]$. Let $D(\mathbf{x}_i, \mathbf{x}_j)$ denote a distance measure between two pattern vectors $\mathbf{x}_i$ and $\mathbf{x}_j$. The cost function can be defined as

$$E_K(M) = \sum_{i=1}^{n} \sum_{j=1}^{K} M_{ij} D(\mathbf{x}_i, \mathbf{m}_j), \tag{4.1}$$

where $\mathbf{m}_j = \sum_{i=1}^{n} M_{ij}\mathbf{x}_i / \sum_{i=1}^{n} M_{ij}$. Vector $\mathbf{m}_j$ is called the center (or prototype) of the j^{th} cluster, $j = 1, 2, \cdots, K$. The most commonly used distance measure is the

*Automatically determining the number of clusters in a data set is an extremely difficult problem [81, 33] which has not been solved in a general setting. Although some clustering algorithms do not require the user to explicitly specify the number of clusters, the number of clusters produced by these algorithms is controlled by some other parameters in the algorithm.

Euclidean distance:

$$D_E(\mathbf{x}_i, \mathbf{m}_j) = (\mathbf{x}_i - \mathbf{m}_j)^T(\mathbf{x}_i - \mathbf{m}_j). \tag{4.2}$$

A clustering algorithm with the Euclidean distance favors hyperspherical-shaped clusters of equal size. This has the undesirable effect of splitting large as well as elongated clusters under some circumstances [34] (see also Figures 4.2, 4.7(a) and 4.8(a)). Simple feature normalization schemes can not solve this problem [34, 120]. First, the same feature normalization scheme must be applied to the whole data set, not individual clusters, because we do not know the clusters *a priori*. Therefore, feature values in individual clusters are often not normalized. Moreover, the global normalization may adversely affect the shape of individual clusters [34]. For example, normalizing the two components in Figure 4.1(a) so that they have the same range will make the three clusters even more elongated. Second, most feature normalization schemes do not consider the correlations between the features, which implies that these methods can not "whiten" the pattern distribution (features are uncorrelated and have the same spread) if the features are correlated. In Section 4.3, we will show that the commonly used z-score normalization scheme can only slightly improve the segmentation results.

It is easy to find examples of data sets in real applications which do not have spherically-shaped clusters of equal size. For example, Figure 4.1 shows the 2-dimensional projections of the well-known IRIS data (see Chapter 2 for its description) onto two planes spanned by the first two principal eigenvectors (Figure 4.1(a)) and the first and the fourth principal eigenvectors (Figure 4.1(b)). Since we know the true category information of the patterns in the IRIS data set, we label each pattern by its category in the projection maps. There are two obvious clusters, but neither of them has a spherical shape. From Figure 4.1, we can see that the larger cluster is a mixture of two classes (iris versicolor and iris virginica) which have nonspherical distributions.

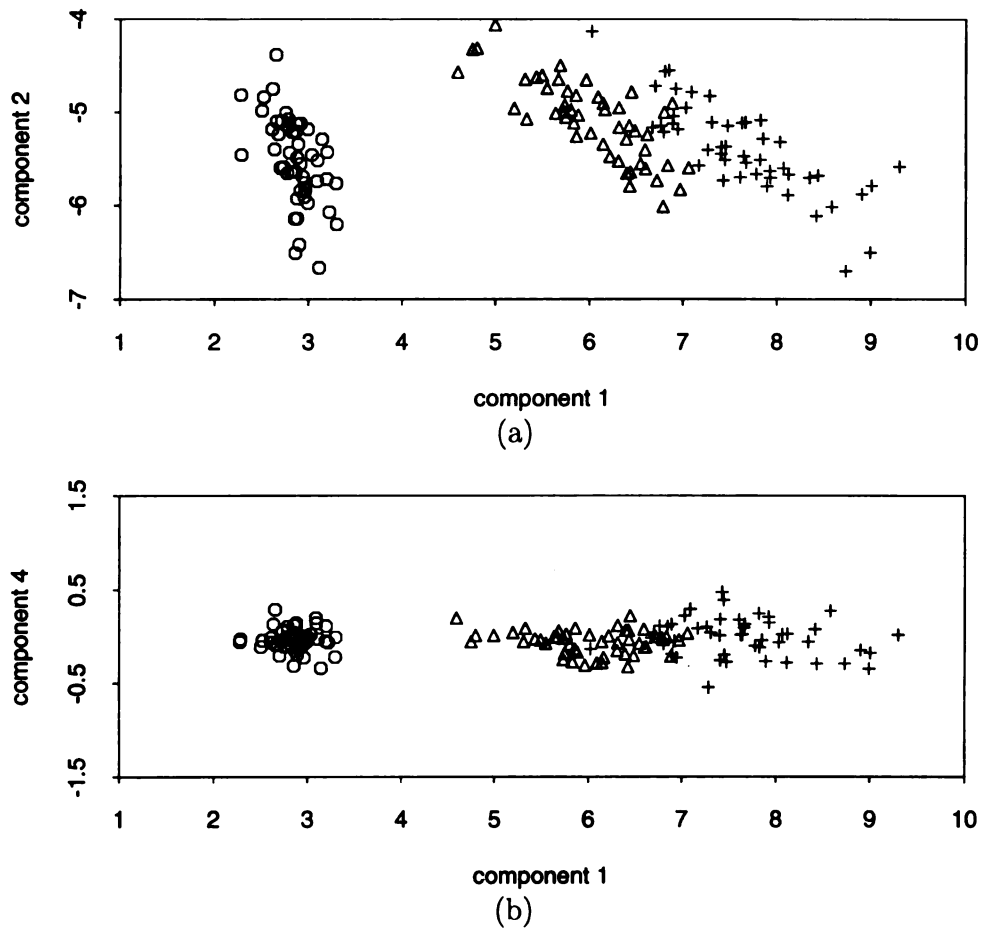


Figure 4.1. Principal component (eigenvector) projection of the IRIS data. (a) Projection on the first and the second components. (b) Projection on the first and fourth components. Small circles (o), triangles ($\triangle$) and "plus" sign (+) denote patterns from classes 1 (iris setosa), 2 (iris versicolor) and 3 (iris virginica), respectively.

The nonspherical nature of the clusters can be easily observed by examining their covariance matrices as shown in Table 4.1. We notice that all the off-diagonal elements of these covariance matrices are not zero, and the diagonal elements are not identical.

Other distance measures, such as the Mahalanobis distance, can also be used in the clustering criterion to take care of hyperellipsoidal-shaped clusters. The Mahalanobis

Table 4.1. The 4 × 4 covariance matrices of the three classes in the IRIS data set.

class 1				class 2				class 3			
0.124	0.099	0.016	0.010	0.266	0.085	0.183	0.056	0.404	0.094	0.303	0.049
0.099	0.144	0.012	0.009	0.085	0.098	0.083	0.041	0.094	0.104	0.071	0.048
0.016	0.012	0.030	0.006	0.183	0.083	0.221	0.073	0.303	0.071	0.304	0.049
0.010	0.009	0.006	0.011	0.056	0.041	0.073	0.039	0.049	0.048	0.049	0.075

distance between pattern vector $\mathbf{x}_i$ and the center of the j^{th} cluster $\mathbf{m}_j$ is defined as

$$D_M(\mathbf{x}_i, \mathbf{m}_j) = (\mathbf{x}_i - \mathbf{m}_j)^T \Sigma_j^{-1} (\mathbf{x}_i - \mathbf{m}_j), \tag{4.3}$$

where Σ_j^{-1} is the inverse of the $d \times d$ covariance matrix of the j^{th} cluster. However, there are a number of difficulties associated with incorporating the Mahalanobis distance in a clustering method: (i) The pattern category information is not available in unsupervised learning, thus the covariance matrices can not be computed *a priori*; the Mahalanobis distance requires computation of the inverse of the sample covariance matrix every time a pattern changes its cluster category (sequential mode), which is computationally expensive. (ii) If the number of patterns in a cluster is small compared to the input dimensionality d, then the $d \times d$ sample covariance matrix of the cluster may be singular. (iii) According to our experience, the K-means clustering algorithm with the Mahalanobis distance tends to produce unusually large or unusually small clusters. A number of graph-theoretic clustering methods, such as the minimal spanning tree (MST) approach, can handle nonspherical clusters. But, they all suffer from other difficulties such as definition of "inconsistent" edges [81]. As a result, the squared-error clustering method with Euclidean distance is the most commonly used partitional clustering method in practice.

Jolion et al. [89] proposed an iterative clustering algorithm based on the minimum volume ellipsoid (MVE) robust estimator. At each iteration in this algorithm, the

best fitting hyperellipsoidal-shaped cluster is detected by comparing the distribution of the patterns inside a minimum volume hyperellipsoid (containing a fraction of the data points) with a Gaussian density, and then this cluster is removed from the data. The process stops whenever the number of remaining data points is less than the prespecified minimum cluster size, or the number of clusters detected exceeds an upper bound. Theoretically, one has to search over the entire feature space to locate the best-fitting cluster, which is extremely computationally demanding. Jolion et al. [89] employed a subsampling technique to reduce the computational requirements. At each iteration, a number of sampling positions (25 in [89]) are randomly located in the d-dimensional space, and $(d+1)$ patterns are then randomly sampled inside a hypercube centered at each of these positions. The fitness is computed from only these $(d+1)$ patterns. Since clusters are sequentially detected, there is no competition between different clusters. If clusters in a data set do not have hyperellipsoidal shape, then this algorithm may produce many fragments which are not assigned to any cluster but scattered in the feature space.

We propose a neural network (HEC) for hyperellipsoidal clustering which can adaptively estimate the hyperellipsoidal shape of each cluster, and use the estimated shape information in competitive learning. The competitive learning will enforce an optimal (or suboptimal) partition which minimizes the cost function even when the hyperellipsoidal assumption is violated. By introducing the regularization technique, the proposed network can recover from the singularity of the sample covariance matrices of clusters.

The HEC network actually implements a partitional clustering algorithm using the following regularized Mahalanobis distance:

$$D_{RM}(\mathbf{x}_i, \mathbf{m}_j) = (\mathbf{x}_i - \mathbf{m}_j)^T[(1-\lambda)(\Sigma_j + \varepsilon I)^{-1} + \lambda I](\mathbf{x}_i - \mathbf{m}_j), \qquad (4.4)$$

where Σ_j is the covariance matrix of the j^{th} cluster, and I is the $d \times d$ identity matrix. Let

$$\Sigma_j^{-1*} = (1 - \lambda)(\Sigma_j + \varepsilon I)^{-1} + \lambda I. \tag{4.5}$$

We refer to Σ_j^{-1*} as the regularized inverse of the covariance matrix Σ_j. The proposed self-organizing network performs clustering based on the regularized Mahalanobis distance measure, which takes the shape of each cluster into consideration. When $\lambda = 0$ and $\varepsilon = 0$, D_{RM} becomes the Mahalanobis distance measure. When $\lambda = 1$, D_{RM} reduces to the Euclidean distance. When $0 < \lambda < 1$, D_{RM} is a linear combination of the Mahalanobis distance and the Euclidean distance. Therefore, λ can be used as a parameter to control the degree that the distance measure deviates from the commonly used Euclidean distance. In situations where Σ_k^{-1} can not be reliably estimated or learned, a larger value of λ should be used. When Σ_k is singular which occurs very often during the first few cycles of learning, the regularization parameters, λ and ε, play a very important role in stabilizing the learning process. The role of ε is to convert a singular matrix (ill-posed problem) to a nonsingular matrix by adding a diagonal matrix with small diagonal elements. In our method, diagonal covariance matrix is an assumption of regularity.

It is well known that the K-means clustering algorithm with the Euclidean distance measure has the undesirable property of splitting big and elongated clusters (see Figure 4.2). We will see an example of this property in Figures 4.7(a) and 4.8(a). On the other hand, use of the Mahalanobis distance sometimes causes a big cluster to absorb nearby small clusters, which leads to the creation of unusually large or unusually small clusters (An unusually small cluster often consists of a single outlier or few neighboring outliers). An illustrative scenario is shown in Figure 4.2 in which two clusters (solid ellipsoids) are merged into a bigger cluster (dashed ellipsoid), and an outlier forms a cluster of its own (this is one possible solution). Introducing the

regularized Mahalanobis distance results in a tradeoff between the above two complementary properties and prevents the clustering algorithm from producing unusually large or unusually small clusters.

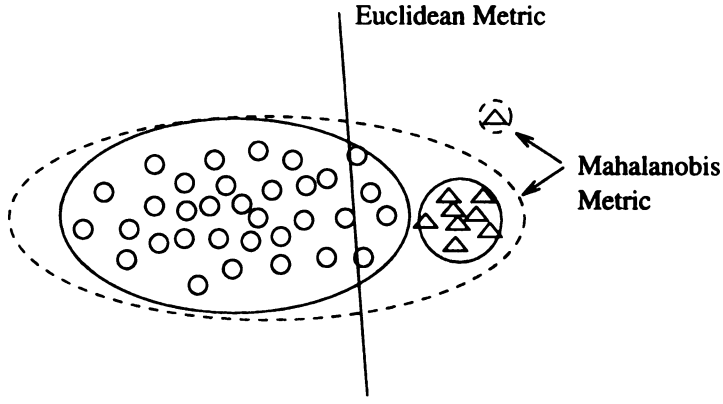


Figure 4.2. A scenario showing the possible 2-cluster solutions by the K-means clustering algorithm with the Euclidean and Mahalanobis distance measures.

Friedman [43] used other forms of regularized covariance matrices and demonstrated the resulting improvements in the performance of quadratic discriminant analysis when the number of patterns in a pattern class is less than (ill-posed) or not considerablely larger than (poorly-posed) the input dimensionality.

The validity of the resulting partition (cluster validity problem) is an important yet difficult problem [81]. In this chapter, the compactness of a cluster is measured by the significance level of the Kolmogorov-Smirnov test on the multivariate normality of clusters. However, directly testing the normality of data in a high dimensional space is a difficult task [163]. Fortunately, under the Gaussian cluster assumption, we can prove that the Mahalanobis distance D_M satisfies the χ^2 distribution with number of degrees of freedom equal to d, which is the input dimensionality. Therefore, the test reduces to the Kolmogorov-Smirnov test on the distribution of the Mahalanobis distances of patterns to their centroid with respect to the theoretical χ^2 distribution.

Let $P_n(x)$ be the empirical cumulative distribution function of a random variable x based on n data points, and $P(x)$ be the theoretical cumulative distribution function. The Kolmogorov-Smirnov statistic Δ is defined as the maximum absolute difference between the two cumulative distributions.

$$\Delta = \max_{-\infty<x<\infty} |P_n(x) - P(x)|. \tag{4.6}$$

The distribution of the Kolmogorov-Smirnov statistic under the null hypothesis is known. Given an observation of Kolmogorov-Smirnov statistic Δ_{obs}, the significance (probability) of Δ_{obs} can be approximated as [141]

$$C(\Delta_{obs}) \equiv P(\Delta > \Delta_{obs}) = Q_{KS}([\sqrt{N} + 0.12 + 0.11/\sqrt{N}]\Delta_{obs}), \tag{4.7}$$

where

$$Q_{KS}(z) = 2\sum_{j=1}^{\infty}(-1)^{j-1}exp\{-2j^2z^2\}. \tag{4.8}$$

Note that $0 \leq C(\Delta_{obs}) \leq 1$. The larger the value of $C(\Delta_{obs})$, the closer are the two distributions.

4.2 Network for Hyperellipsoidal Clusters (HEC)

In the previous section, we mentioned the three difficulties associated with integrating the Mahalanobis distance measure into the K-means clustering algorithm. The proposed regularized Mahalanobis distance measure can overcome the second and the third difficulties. However, the first difficulty dealing with the computational issue still remains unsolved. Real-time implementation of the K-means algorithm with the Mahalanobis distance measure may be pursued only through hardware implementation. What architecture to use is an important issue. In this section, we present a

self-organizing network (HEC) for hyperellipsoidal clustering.

4.2.1 The HEC Network and Learning Algorithm

The proposed network architecture for detecting hyperellipsoidal clusters is shown in Figure 4.3. The input layer has d input nodes, where d is the number of features. The hidden layer is composed of $K \times d$ linear nodes which are grouped into K subnetworks with d nodes each, where K is the number of clusters. We can view the hidden layer as a $K \times d$ 2-dimensional array. Each subnetwork is designed to perform a principal component analysis of a cluster consisting of those input patterns which activate the corresponding output node. We choose the PCA network (see Figure 2.2 proposed by Rubner et al. [151, 150] as a building block in our network for hyperellipsoidal clustering, but other PCA networks [1, 6, 40, 101, 103, 133, 151, 150, 154] can also be used here.

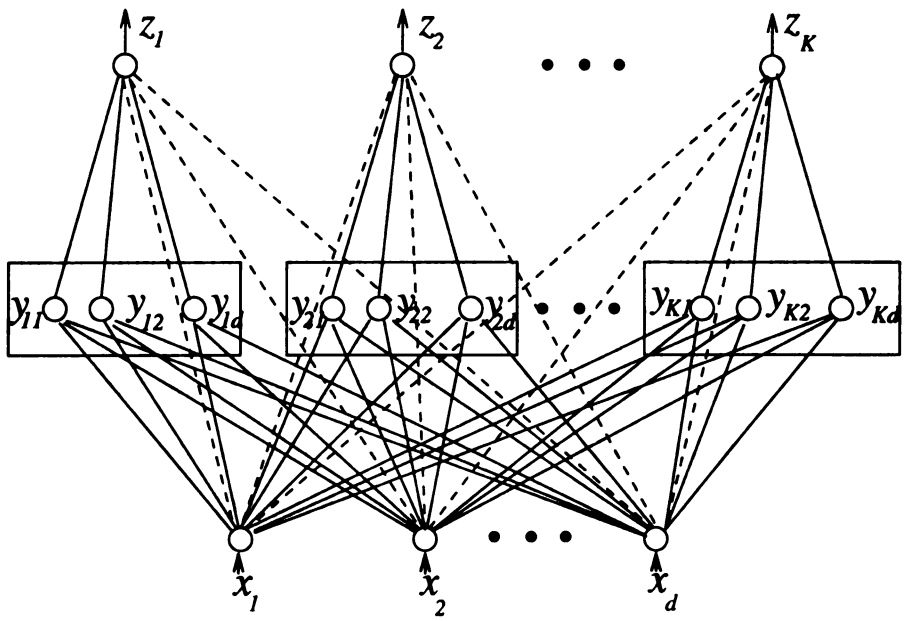


Figure 4.3. Architecture of the neural network (HEC) for hyperellipsoidal clustering.

Let $Var\{h_{kj}|z_k = 1\}$ be the variance of h_{kj} based on only those transformed input pattern vectors which are assigned to the k^{th} cluster by the output layer. Then, $\lambda_{kj} = Var\{h_{kj}|z_k = 1\}$ is the j^{th} eigenvalue of the covariance matrix of the k^{th} cluster. If we choose s_{kj} as

$$s_{kj} = 1/\sqrt{Var\{h_{kj}|z_k = 1\} + \varepsilon},\ k = 1, 2, \cdots, K,\ j = 1, 2, \cdots, d, \tag{4.9}$$

where ε is a small positive number to prevent a zero denominator (in our experiments, $\varepsilon = 0.000001$), then each subnetwork "whitens" its corresponding cluster [45, 111]; if a cluster in the input space has a hyperellipsoidal shape, then the output of its corresponding subnetwork will have a hyperspherical shape.

The output layer has K nodes corresponding to K clusters, and is basically a "winner-take-all" type of network; the lateral connections have not been shown. Competitive learning is performed in the output layer. The input layer is fully connected to the output layer (by dashed lines in Figure 4.3). However, all the output nodes in the k^{th} subnetwork are connected only to the k^{th} node in the output layer. Let m_{ik} denote the weight on the connection between the i^{th} input node and the k^{th} node in the output layer, and v_{jk} be the weight on the connection between the j^{th} output node of the k^{th} subnetwork and the k^{th} output node in the output layer. Each output node computes the weighted squared-Euclidean distance between its inputs (both the input pattern and outputs of the connected subnetwork) and the stored pattern on the connections. This node activation function is similar to the one used in Kohonen's self-organizing network [92, 93, 94].

Let D_k denote the total signal intensity (potential value) which the k^{th} output node receives from the input layer and hidden layer. Then,

$$D_k = (1 - \lambda)\sum_{j=1}^{d}(v_{jk} - y_{kj})^2 + \lambda\sum_{i=1}^{d}(m_{ik} - x_i)^2,\ k = 1, 2, \cdots, K, \tag{4.10}$$

where $0 \leq \lambda \leq 1$ is a regularization parameter. The output values, z_k, $k = 1, 2, \cdots, K$, are determined by competitive learning. The node with the *smallest* potential value D_k^* will be the winner.

The weight vector $\mathbf{m}_k = (m_{1k}, m_{2k}, \cdots, m_{dk})^T$ is designed to store the centroid (in the original input space) of the k^{th} cluster formed by the network, $k = 1, 2, \cdots, K$. Kohonen's LVQ algorithm [93], for example, can be used to learn these weight vectors. The weight vector, $\mathbf{m}_k$, is used both in learning the weight matrix Φ_k (to subtract it from the input pattern vector) and in computing the distance D_k for competitive learning as a stabilizer.

Similar to $\mathbf{m}_k$, the weight vector $\mathbf{v}_k = (v_{1k}, v_{2k}, \cdots, v_{dk})^T$ is designed to store the centroid in the rotated and whitened space by Φ_k and $\mathbf{s}_k$ of the k^{th} cluster currently formed by the network, $k = 1, 2, \cdots, K$. Again, Kohonen's LVQ algorithm can be used to learn $\mathbf{v}_k$, $k = 1, 2, \cdots, K$.

Rewriting Eq. (4.10) in the vector form, we obtain

$$D_k = (1 - \lambda)(\mathbf{y}_k - \mathbf{v}_k)^T(\mathbf{y}_k - \mathbf{v}_k) + \lambda(\mathbf{x} - \mathbf{m}_k)^T(\mathbf{x} - \mathbf{m}_k). \tag{4.11}$$

Since $s_{kj} = 1/\sqrt{\lambda_{kj} + \varepsilon}$, and

$$\mathbf{y}_k = \Lambda_k^{-1/2}\Phi_k\mathbf{x}, \tag{4.12}$$

where

$$\Lambda_k = \text{diag}\{\lambda_{k1}, \lambda_{k2}, \cdots, \lambda_{kd}\} + \varepsilon I, \tag{4.13}$$

and I is a $d \times d$ identity matrix, then

$$\mathbf{v}_k = \Lambda_k^{-1/2}\Phi_k\mathbf{m}_k. \tag{4.14}$$

Substituting $\mathbf{y}_k$ and $\mathbf{v}_k$ into Eq. (4.11), we obtain

$$\begin{aligned} D_k &= (\mathbf{x}-\mathbf{m}_k)^T[(1-\lambda)\Phi_k^T\Lambda_k^{-1}\Phi_k+\lambda I](\mathbf{x}-\mathbf{m}_k) \\ &= (\mathbf{x}-\mathbf{m}_k)^T[(1-\lambda)(\Sigma_k+\varepsilon I)^{-1}+\lambda I](\mathbf{x}-\mathbf{m}_k), \end{aligned} \tag{4.15}$$

where Σ_k is the covariance matrix of the k^{th} cluster, and I is the $d \times d$ identity matrix. Let

$$\Sigma_k^{-1*} = (1-\lambda)(\Sigma_k+\varepsilon I)^{-1}+\lambda I. \tag{4.16}$$

We refer to Σ_k^{-1*} as the regularized inverse of the covariance matrix Σ_k, and D_k as the regularized Mahalanobis distance. The proposed self-organizing network performs clustering based on the regularized Mahalanobis distance measure, which takes the shape of each cluster into consideration. In our implementation, the regularization parameter $\varepsilon = 0.000001$, and λ is a decreasing function of the number of cycles (number of times that the algorithm performs Steps 2-4 in the HEC learning algorithm):

$$\lambda_t = \max(\lambda_{min}, \lambda_{t-1} - \Delta\lambda), \tag{4.17}$$

where $\lambda_0 = 1.0$. The values of λ_{min} and $\Delta\lambda$ are specified by the user.

Note that in the HEC network, the following parameters need to be learned from the input data: (i) weight matrix $\Phi_k = [w_{ij}^{(k)}]_{d\times d}$ for each subnetwork, $k = 1, 2, \cdots, K$, and scaling factors $\mathbf{s}_k = \{s_{k1}, s_{k2}, \cdots, s_{kd}\}$, $k = 1, 2, \cdots, K$; (ii) weight vector $\mathbf{m}_k = \{m_{1k}, m_{2k}, \cdots, m_{dk}\}$ for each output node, $k = 1, 2, \cdots, K$; and (iii) weight vector $\mathbf{w}_k = \{w_{1k}, w_{2k}, \cdots, w_{dk}\}$ for each output node, $k = 1, 2, \cdots, K$.

We summarize the learning algorithm for determining these weights and scaling

factors in the HEC network as follows.

1. Initialization: set weights $m_{ik} = v_{ik}$, $i = 1, 2, \cdots, d$, $k = 1, 2, \cdots, K$, to small random values; set all the weights, $w_{ij}^{(k)} = 1$ if $i = j$, and 0 otherwise; set all the scaling factors $s_{kj} = 1$. Set $\varepsilon = 0.000001$ and $\lambda_0 = 1.0$. Therefore, the network starts with a preference for hyperspherical shaped clusters.

2. Learn all the weights associated with the output nodes, $\mathbf{m}_k$, $k = 1, 2, \cdots, K$, using the LVQ algorithm with the activation function defined in Eq. (4.10). (In our implemetation, a batch-mode algorithm is used).

3. Learn all the weights in the subnetworks using the modified Rubner's PCA algorithm (with a momentum term and scaling factors). Note that upon presentation of a pattern to the network, only the weights in the winner PCA subnetwork (whose corresponding output node is the winner) are updated.

4. Decrease the value of λ according to Eq. (4.17).

5. Repeat steps 2-4 until the cluster membership of input patterns does not change, or the maximum number of cycles is reached.

6. Perform the Kolmogrov-Smirnov test on all the clusters.

Step 6 can also be performed for every cycle so that we can monitor the compactness of the currently-formed clusters. We can see that in the HEC learning algorithm, all the weights associated with a subnetwork and its corresponding output node are not updated unless the output node becomes the winner. However, some variations of

this algorithm can be considered. For example, instead of updating the winner (winner subnetwork) only, the neighbors† (neighboring subnetworks) are also updated in a manner proposed by Pal at el. [135] in order for the HEC network to be less sensitive to the initial weights. Conversely, the rival node (rival subnetwork) of the winner [184] can be *penalized* so that the HEC network is able to recover a meaningful clustering solution from an inappropriately-specified number of clusters. In this thesis, we are not pursuing these variations.

The LVQ algorithm and Rubner's PCA algorithm have been proved to converge under certain conditions on the learning parameters [95, 151, 69]. However, we have experienced slow convergence of Rubner's PCA algorithm when the input feature dimensionality is high and some of the eigenvalues are small. After numerous trials with different schemes for choosing the learning parameters, we have found the following scheme to work relatively well on our data sets. The Hebbian rule (Eq. (4.18)) and anti-Hebbian rule (Eq. (4.19)) are modified as follows.

$$\Delta w_{ij}^{(k)}(t+1) = \eta(t) x_i h_{kj} + \beta(t) \Delta w_{ij}^{(k)}(t), \tag{4.18}$$

and

$$\Delta u_{lj}^{(k)}(t+1) = -\mu(t) h_{kl} h_{kj} + \beta(t) \Delta u_{lj}^{(k)}(t), \tag{4.19}$$

where $\eta(t+1) = 0.99999\eta(t)$, $\mu(t+1) = 0.99999\mu(t)$, $\beta(t+1) = 0.99999\beta(t)$, t is the iteration index in the PCA learning (one iteration means one presentation of a pattern), with initial values $\eta(0) = 0.2$, $\mu(0) = 1.5$, and $\beta(0) = 0.1$. We scale all the data sets to make the maximum range of feature values to be 1.0 without changing the shape of clusters. Thus, the same set of parameter values is applied to all the data sets.

†Neighbors can be either defined in the topological sense as in Kohonen's self-organizing mapping [93] or defined in the input feature space [135].

4.2.2 Simulations

In this subsection, we demonstrate the performance of the HEC network on two artificial data sets and the well-known IRIS data set. In all the experiments reported in this subsection, $\lambda_{min} = 0.0$, $\Delta\lambda = 0.2$, and the maximum number of cycles (Steps 2-4) is 10. The batch-mode K-means algorithm (similar to Forgy's K-means algorithm [42]) is used for the comparison purpose in all our experiments. Since the clustering solutions provided by the K-means algorithm and the HEC network depend on initial centroids or initial weights, we repeated all the experiments ten times with different initializations and the best solutions obtained by both the methods are reported. We should mention that only a few distinct solutions were generated by both the algorithms and the probability of producing the best solution is much higher than producing other solutions. This sensitivity to the initialization can be reduced by incorporating a technique proposed by Pal at el. [135].

The two artificial data sets, which contain a mixture of spherical and ellipsoidal clusters (Gaussian distributions with different mean vectors and covariance matrices), are very illustrative of typical cases where the K-means clustering algorithm (and other partitional algorithms and competitive networks) with the Euclidean distance measure do not work well, but the HEC performs well. Figure 4.4 shows the results of applying the K-means algorithm and the HEC network to the two artificial data sets. We notice that several patterns which are "misclassified" by the K-means algorithm are correctly assigned by the HEC network. Table 4.2 lists the significance levels of the Kolmogrov-Smirnov test on all the clusters shown in Figures 4.4(a)-4.4(d). We should point out that in all our experiments, the Kolmogrov-Smirnov test is applied to the Mahalanobis distances (not the Euclidean distances) of the patterns in the individual clusters. We can see from Table 4.2 that the compactness of all the clusters in terms of the significance level of the Kolmogrov-Smirnov test is improved (by a substantial

amount for some clusters) by using the HEC network. Of course since these artificial data sets were generated from Gaussian distribution, the significance levels of the Kolmogrov-Smirnov test is unusually high. Such large values are seldom observed for real data sets.

Table 4.2. The significance levels of the Kolmogrov-Smirnov test for all the clusters in Figures 3(a)-3(d).

clustering method	K-means		HEC network	
	cluster 1 (△)	cluster 2 (○)	cluster 1 (△)	cluster 2 (○)
data set 1	0.318	0.954	0.857	0.989
data set 2	0.831	0.354	0.935	0.996

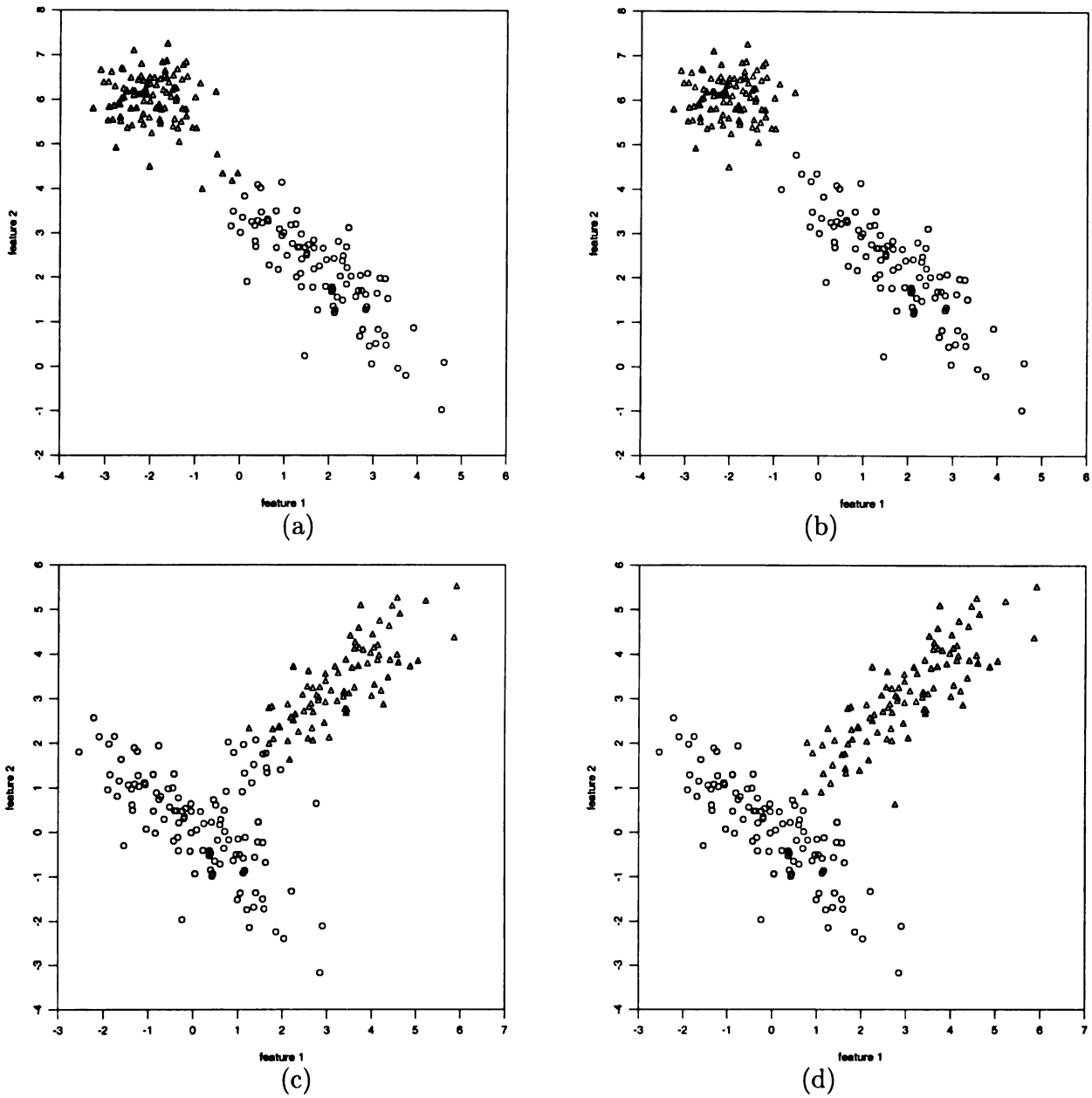


Figure 4.4. The K-means clustering with the Euclidean distance measure versus the HEC clustering on two artificial data sets, each containing two clusters. (a)-(b): data set 1; (c)-(d): data set 2; (a) and (c): results of the K-means clustering algorithm with the Euclidean distance measure; (b) and (d): results of the HEC network.

We also compare the HEC network with the K-means algorithm on the well-known IRIS data set. Since the IRIS data are in a four-dimensional space, for the visualization purpose, we display only its eigenvector projections onto the 2-dimensional planes spanned by the first two principal eigenvectors and by the first and the fourth principal eigenvectors (We use two projection maps so that we can easily visualize the shape of clusters in the data). The position of each pattern in the map is determined by its eigenvector projection, while its label is obtained by using either the K-means algorithm or the HEC network operating in the original 4-dimensional space. For the 3-cluster solutions (the number of clusters is specified as three) as shown in Figures 4.5 and 4.6, both the K-means and the HEC network correctly group patterns from the first class (setosa) into one cluster, because the patterns in the first class (setosa) are well separated from the rest of the data. However, since the second class (versicolor) and the third class (virginica) are slightly overlapped and both are not spherically shaped (see Figures 4.5(a) and 4.6(a)), the K-means algorithm produces a partition which does not correctly group patterns from these two classes; the compactness of the corresponding two clusters is also poor. On the other hand, the HEC network can recover the clusters which reflect the true classes very well. This can be observed by comparing Figure 4.5(c) with Figure 4.5(a) and Figure 4.6(c) with Figure 4.6(a).

The numerical values of misclassification rate and compactness measures of the clusters generated by the K-means algorithm and the HEC network are shown in Table 4.3. Note that the K-means algorithm misclassifies 16 patterns (compared to the true categories), while the HEC network has misclassified only five patterns. The compactness of the second cluster is also significantly improved by using the HEC network at the cost of slightly degrading the compactness of the third cluster. Figures 4.7 and 4.8 show the eigenvector projections of the 2-cluster solutions by the K-means algorithm and the HEC network. We notice that although the patterns in the first class (setosa) are well-separated from the rest of the data, due to the

substantially different spreads of the two natural clusters in the feature space, there are still three patterns from the bigger cluster (mixture of classes 2 (versicolor) and 3 (virginica)) which are assigned to the smaller cluster by the K-means algorithm with the Euclidean distance measure. These three misclassified patterns significantly degrade the compactness of the two resulting clusters. On the other hand, the HEC network correctly produces the two natural clusters.

Table 4.3. The significance levels of the Kolmogrov-Smirnov test and numbers of misclassified patterns by the K-means algorithm and the HEC network for the IRIS data. The eigenvector projections of the 3-cluster solutions are shown in Figures 4(b)-4(c) and 5(b)-5(c), and those of the 2-cluster solutions are shown in Figures 6 and 7.

method		K-means			HEC network		
3-cluster solution	#misclassified/n	16/150			5/150		
	compactness of the 3 clusters	0.559	0.676	0.857	0.559	0.897	0.814
2-cluster solution	#misclassified/n	3/150			0/150		
	compactness of the 2 clusters	0.271	0.603		0.559	0.720	

We should point out that although the comparison is performed between the K-means algorithm with the Euclidean distance measure and the HEC network, these conclusions can also be applied to other partitional clustering algorithms and competitive networks such as, Kohonen's LVQ and ART models, which use the Euclidean distance measure.

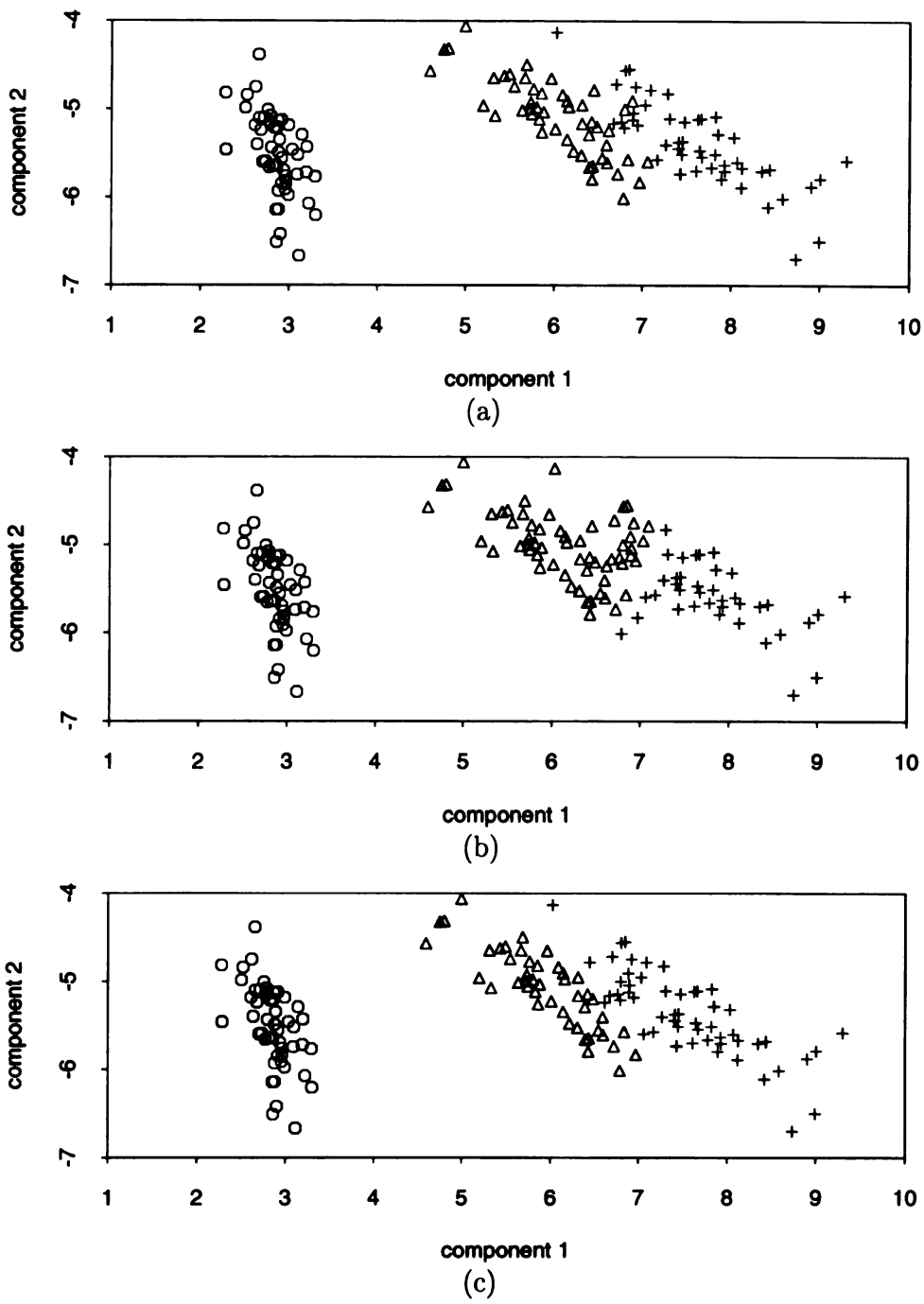


Figure 4.5. Two-dimensional projection maps (the first two principal components) of 3-cluster solutions by the K-means clustering with the Euclidean distance measure and the HEC clustering on the IRIS data. (a) The three true classes. (b) Result of the K-means clustering algorithm with the Euclidean distance measure. (c) Result of the HEC network.

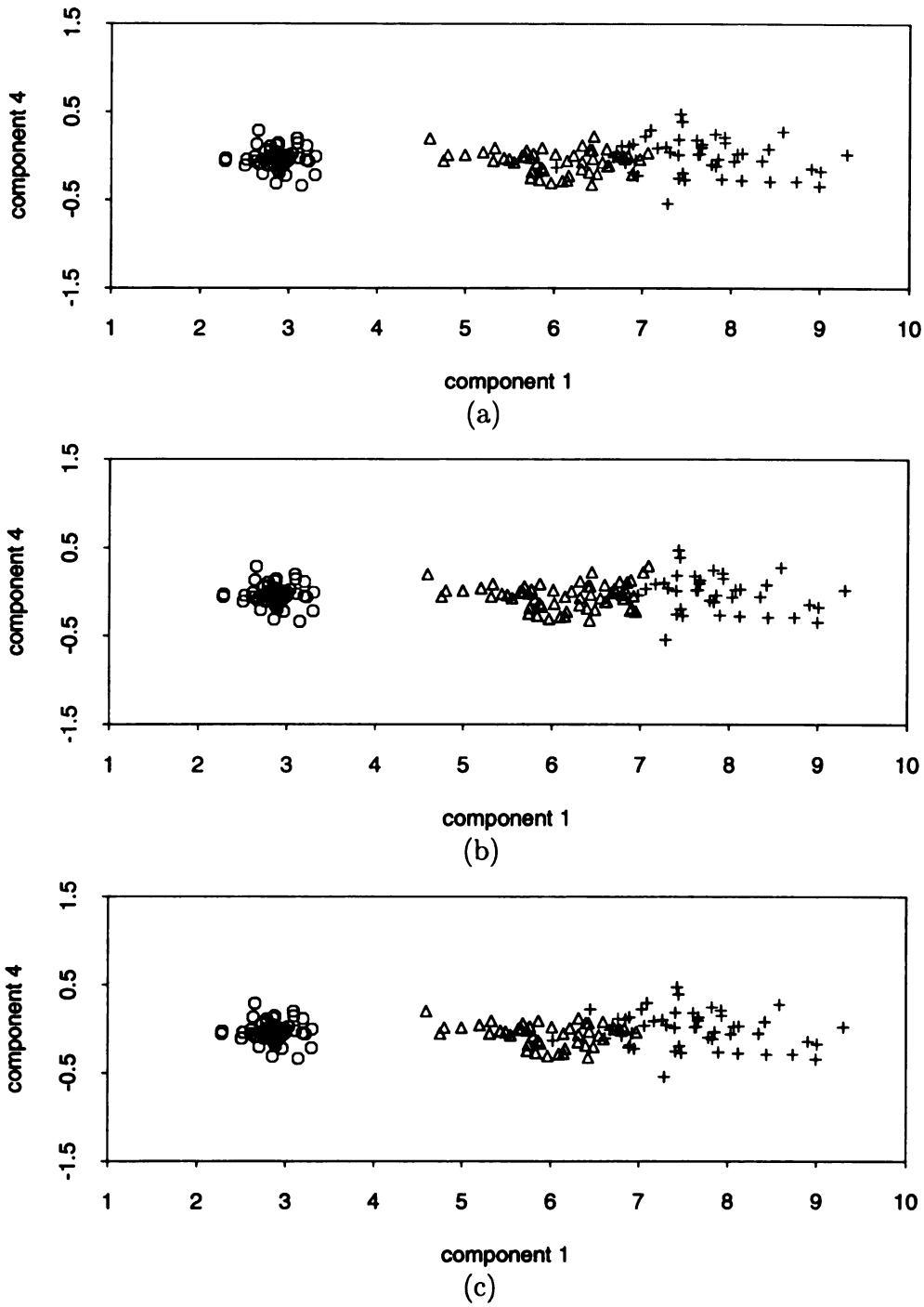


Figure 4.6. Two-dimensional projection maps (the first and fourth principal components) of 3-cluster solutions by the K-means clustering with the Euclidean distance measure and the HEC clustering on the IRIS data. (a) The three true classes. (b) Result of the K-means clustering algorithm with the Euclidean distance measure. (c) Result of the HEC network.

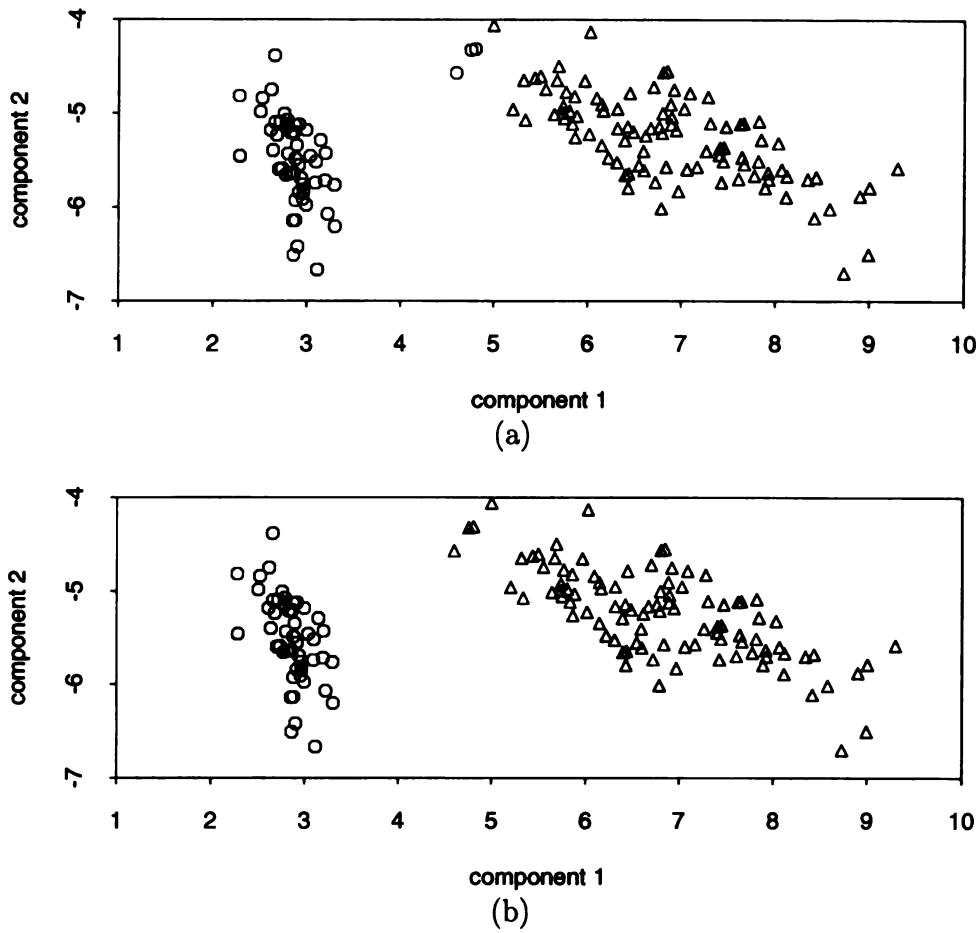


Figure 4.7. Two-dimensional projection maps (the first two principal components) of 2-cluster solutions by the K-means clustering with the Euclidean distance measure and the HEC clustering on the IRIS data. (a) Two clusters produced by the K-means clustering algorithm with the Euclidean distance measure. Note the three "misclassified" patterns. (b) Two clusters produced by the HEC network.

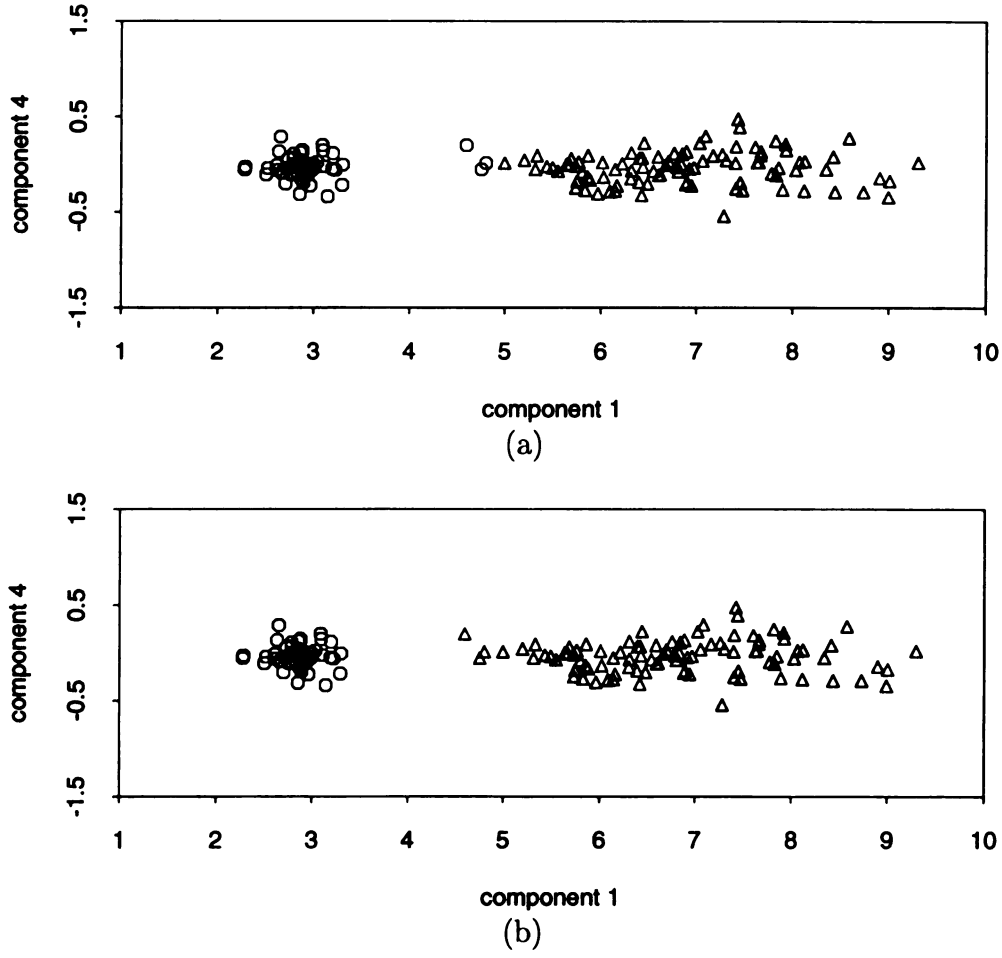


Figure 4.8. Two-dimensional projection maps (the first and fourth principal components) of 2-cluster solutions by the K-means clustering with the Euclidean distance measure and the HEC clustering on the IRIS data. (a) Two clusters produced by the K-means clustering algorithm with the Euclidean distance measure. Note the three "misclassified" patterns. (b) Two clusters produced by the HEC network.

4.3 Image Segmentation Using the HEC Network

Clustering-based image segmentation consists of two basic components: representation and grouping. The representation component can be considered as a transformation from the input image to a set of feature images. What representation scheme to use is largely driven by image properties and the requirements of the segmentation task. The next subsection will discuss the representation scheme that we will use for texture segmentation problems. After the representation is obtained, the HEC network is applied to group a set of 1,000 randomly sampled pixels from a typically large image. This scheme was used [84, 39] in order to reduce the amount of computation. The trained HEC network (or K-means algorithm for comparison) is then used to classify every pixel in the image.

4.3.1 Representation

Texture can be characterized by the local spatial frequency and orientation properties present in the image. Daugman [28] has shown that Gabor filters exhibit optimal localization properties in both spatial domain and the frequency domain. So, Gabor filters are ideally suited for texture segmentation problems [39, 84]. In spatial domain, a two-dimensional even-symmetric Gabor filter is represented as

$$h(x,y) = exp\left\{-\frac{1}{2}\left[\frac{x^2}{{\sigma_x}^2} + \frac{y^2}{{\sigma_y}^2}\right]\right\} cos(2\pi u_0 x),$$

where σ_x and σ_y are the standard deviations of the Gaussian envelope along the x and y directions, respectively, and u_0 is the frequency of the sinusoidal plane wave along the x-direction (0° orientation) [84]. A rotation of the $x-y$ plane by angle θ will result in Gabor filters at orientation θ. For an image with a width of N_c pixels (assume that N_c is a power of 2), the following values of radial frequency u_0 (cycles/image-width)

can be used: $1\sqrt{2}, 2\sqrt{2}, \cdots, N_c/4\sqrt{2}$. The value of θ is quantized into four orientations ($\theta = 0°, 45°, 90°, 135°$). The standard deviations of the Gaussian envelope (σ_x and σ_y) are specified in terms of u_0 [39].

A texture representation scheme using Gabor filters, which was used in [84], is adopted here. It consists of two main steps: (i) Filter the input image through a bank of n even-symmetric Gabor filters, to obtain n *filtered images.* (ii) Compute the *feature images* which are the "local energy" estimates over Gaussian windows of appropriate size centered at every pixel in each of the filtered images. Thus, each pixel in a textured image is represented as a point in an n-dimensional feature space. Regions of homogeneous texture can then be segmented by a clustering algorithm.

Choice of a set of appropriate filters to a class of images is often crucial to achieve a good segmentation [39, 84]. This filter selection problem is beyond the scope of this thesis. We will use the four highest-frequency filters for segmenting textured images. Since there are four different orientations for each frequency, a total of 16 feature images for a single texture image are generated. The spatial information (x-y coordinates) is not used in our segmentation scheme.

4.3.2 Segmentation Results

Figures 4.9(a) and 4.9(b) show the two 256 × 256 texture images, containing 4 (D68, D55, D84, D77) and 5 (D77, D55, D24, D84, D17) different textures, respectively, from the Brodatz album [18]. Sixteen Gabor filters (4 highest radial frequencies and 4 different orientations per frequency) are applied to these two texture images. Therefore, 16 256 × 256 feature images are obtained for each texture image. A total number of 1,000 pixels are randomly chosen to form a set of "training" (without category labels) patterns. The K-means algorithm and the HEC network are applied only to this set of 1,000 patterns instead of all the 64K pixels in order to reduce the amount of computation. For both the K-means and the HEC network, we specify

the number of clusters to be 4 for the texture image in 4.9(a) and 5 for 4.9(b). The minimum Euclidean distance classifier with the centroids obtained from the K-means algorithm is used to classify every pixel in the image.

We apply the K-means algorithm to both the 16 Gabor filter features and the 16 z-score normalized Gabor filter features to investigate the effect of normalization on the segmentation results. Figures 4.9(c) and 4.9(f) show the segmentation results using the K-means algorithm without feature normalization, and Figures 4.9(d) and 4.9(g) show the results with the z-score normalization. We can see that the z-score normalization does not help a lot in improving the segmentation results. The trained HEC network ($\lambda_{min} = 0.5$, $\Delta\lambda = 0.1$, and the maximum number of cycles is 10) is also used to classify every pixel in the image, and the segmentation results are shown in Figures 4.9(e) and 4.9(h). Note that the HEC network does not need any normalization scheme. Comparing Figure 4.9(e) with Figures 4.9(c)-4.9(d) and Figure 4.9(h) with Figures 4.9(f)-4.9(g), we find that a significant improvement has been achieved by using the regularized Mahalanobis distance measure. Many small noisy patches in Figures 4.9(c)-4.9(d) disappear as shown in Figure 4.9(e). The texture boundaries in Figures 4.9(f)-4.9(g) are improved as shown in Figure 4.9(h). Table 4.4 lists the "misclassification" rates of the segmentation results in Figures 10(c)-10(h) with respect to the ground-truth images of textures in Figures 10(a) and 10(b). The "misclassification" rate is defined as the ratio of the number of "misclassified" pixels to the total number of pixels in the image. We should point out that this measurement is not a very reliable indicator of segmentation quality, because the "misclassification" rate can be small but the visual quality of the resulting segmentation may still be poor. We can see from Table 4.4 that the K-means algorithm with the z-score normalization slightly reduces the "misclassification" rate. However, these slight reductions in the "misclassification" rate do not noticeably improve the visual quality of the segmentation results (see Figures 10(d) and 10(g)). On the other hand, the

HEC network achieves consistent improvements in both the visual quality and the "misclassification" rate.

Table 4.4. The "misclassification" rates of the texture segmentation results in Figures 10(c)-10(h).

texture image	K-means without normalization	K-means with z-score normalization	HEC network
Figure 10(a)	7.1%	6.5%	5.8%
Figure 10(b)	5.4%	4.8%	2.9%

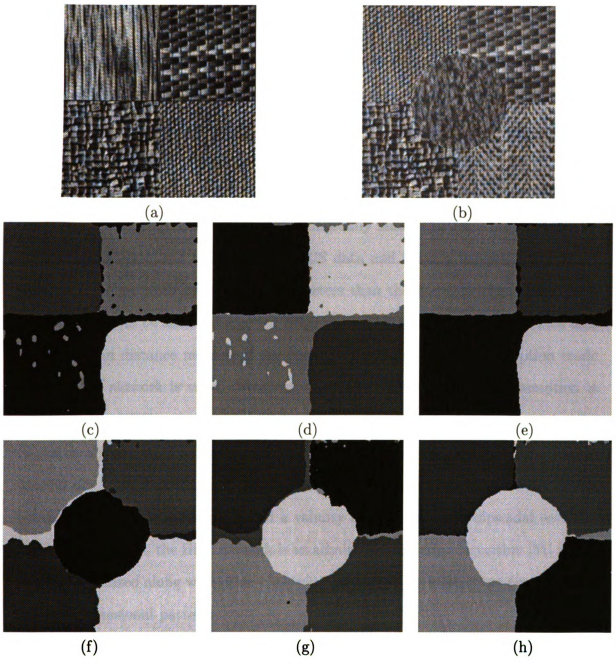


Figure 4.9. The K-means clustering with the Euclidean distance measure versus the HEC clustering for texture segmentation. (a) and (b): Original composite textured images. (c) and (d): 4-class segmentations of the image in (a) using the K-means algorithm on the original features and z-score normalized features, respectively. (e): 4-class segmentation using the HEC network. (f) and (g): 5-class segmentations of the image in (b) using the K-means algorithm on the original features and z-score normalized features, respectively. (h): 5-class segmentation using the HEC network.

4.4 Summary

We have proposed a self-organizing network for hyperellipsoidal clustering. The network performs a partitional clustering using the proposed regularized Mahalanobis distance measure. This measure can deal with the ill-posed and poorly-posed problems in estimating the Mahalanobis distance and achieves a tradeoff between using the Euclidean distance and the Mahalanobis distance so as to prevent the HEC network from producing unusually large or unusually small clusters. Experiments on both artificial data and real data (the IRIS data and texture images) have shown that the HEC network provides better clusters than the K-means algorithm. Same improvement will be achieved over any other partitional clustering method which uses the Euclidean distance measure if the underlying hyperellipsoidal assumption made in the HEC network is valid. However, even if this hyperellipsoidal assumption is not true, the HEC network can still be used to tune the results produced by the K-means algorithm. The Komogrov-Smirnov test on the distribution of the Mahalanobis distances between patterns in a cluster and the corresponding cluster center provides a compactness measure and a validity test of the hyperellipsoidal assumption. In conclusion, the HEC network is an admissible clustering algorithm [81] which should be utilized along with other clustering algorithms to understand the structure of multidimensional patterns.

Future work in this area will be on improving the convergence of the PCA subnetworks, used in the HEC network, especially when the input dimensionality is high and many eigenvalues of the covariance matrices of individual clusters are small. Removing outliers in estimating the shapes of clusters can also improve the clustering results. Other schemes for introducing penalty on illformed clusters (e.g., clusters with poor compactness, unusually large or unusually small) in competitive learning should be further investigated. The current implementation of the HEC network

requires a prespecified number of clusters. The rival penalized competitive learning proposed by Xu at el. [184] may be incorporated into the HEC network to recover a meaningful clustering solution if an inappropriate number of clusters is specified.

CHAPTER 5

Generalization Ability of Feedforward Networks

Supervised learning from examples can be formulated as a problem of function regression or approximation. We are given a set of n training patterns, $\mathcal{T}_n = \{\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_n\}$, where $\mathbf{z}_i = (\mathbf{x}_i, y_i)$ is an input-output pair, $\mathbf{x}_i \in R^d$, $i = 1, 2, \cdots, n$. Without a loss of generality, assume that $y_i \in R$. We assume that all the patterns in $\mathcal{T}_n$ are independently drawn from an unknown joint distribution, $\Xi(\mathbf{x}, y)$. The goal of supervised learning is to determine a function $f : R^d \rightarrow R$ from the training samples such that $y = f(\mathbf{x})$, is an estimator of the unknown function, $\mu(\mathbf{x})$, which governs the input-output relationship. A mathematical criterion is needed to determine how good $f(\mathbf{x})$ approximates and generalizes the unknown function $\mu(\mathbf{x})$. In the neural network literature, the squared error criterion is most commonly used to measure the error between the observed value y and estimated value $f(\mathbf{x})$, $d(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$. Other error criteria can also be used. In this thesis, the approximation error is measured by the empirical squared error on the training set,

$$E(\mathcal{T}_n) = \frac{1}{n} \sum_{i=1}^{n} d(y_i, f(\mathbf{x}_i)). \tag{5.1}$$

The generalization error is defined as

$$E = \langle d(y, f(\mathbf{x})) \rangle, \tag{5.2}$$

where $\langle \cdot \rangle$ denote the mathematical expectation. In order to evaluate E, one needs to know the joint distribution Ξ which is usually not available. In this case, the generalization error is replaced by the *empirical prediction error* on a set of n' test patterns, $\mathcal{T}'_{n'} = \{\mathbf{z}'_1, \mathbf{z}'_2, \cdots, \mathbf{z}'_{n'}\}$, where $\mathbf{z}'_i = (\mathbf{x}'_i, y'_i)$, $i = 1, 2, \cdots, n'$.

$$E(\mathcal{T}'_{n'}) = \frac{1}{n'} \sum_{i=1}^{n'} d(y'_i, f(\mathbf{x}'_i)). \tag{5.3}$$

For classification problems, the two corresponding classification error probabilities (on training set and test set) are used. The objective of supervised learning is to find a function $f^* \in \mathcal{A}$ such that both $E(\mathcal{T}_n)$ and E or $E(\mathcal{T}'_{n'})$ are minimized over $\mathcal{A}$, where $\mathcal{A}$ is the set of all measurable functions in R^d. Therefore, there are two tasks involved in supervised learning. First, we need to determine what form the function f should take, i.e., to determine a subset $\mathcal{F}$ of $\mathcal{A}$. For example, $\mathcal{F}$ can be $\mathcal{C}$, a set of all continuous functions, or a set of all polynomial functions of order up to some value, or a set of all functions that a feedforward network with a given topology can approximate. Second, once the form of the function f is determined, we need to estimate the parameters involved in the function from the training samples. The generalization ability of feedforward networks can be measured by the deviation of the empirical prediction error (on test set) from the empirical error on the training set. The smaller this deviation is, the better generalization ability the network has.

There are three fundamental and practical issues associated with learning from samples: i) adequacy of $\mathcal{F}$, ii) sample complexity, and iii) time complexity. Any learning theory must address these three issues.

The first issue concerns whether the set $\mathcal{F}$ contains the true solution $\mu(\mathbf{x})$. If not, we can never hope to obtain the optimal solution. This remains a difficult and open problem. Choosing a right solution space $\mathcal{F}$ often requires both a considerable insight into the particular problem domain (the nature of the true solution $\mu(\mathbf{x})$) and the approximation capabilities of available networks (what functions can be implemented). The approximation capabilities of feedforward neural networks have recently been investigated by many researchers [27, 49, 51, 62, 70, 77, 165, 179, 52, 20, 78]. A fundamental result of these studies has shown that 3-layer, or even 2-layer, feedforward networks with an arbitrarily large number of nonlinear hidden units are capable of implementing any continuous mapping and its derivatives with a prespecified accuracy under some mild conditions. Unfortunately, most of these theoretical studies ignore the learnability problem that is concerned with whether there exist methods to learn the network weights from empirical observations of the mappings. Furthermore, these theoretical analyses have not introduced any new practical learning methods.

The second issue, sample complexity, determines the number of training patterns needed to train the network in order to guarantee a valid generalization. Too few patterns may cause the "overfitting" problem where the network performs well on the training data set, but poorly on the independent test patterns drawn from the same joint distribution Ξ as the training patterns.

The third issue is the computational complexity of the learning algorithm used to estimate a solution from the training patterns. Many existing learning algorithms have high computational complexity. For example, the popular backpropagation learning algorithm for feedforward networks is computationally demanding because of its slow convergence. Designing efficient algorithms for neural network learning is a very active research topic. Various learning algorithms have been proposed for different types of neural networks. However, in some situations where learning can be done off-line, this issue is less critical.

This chapter will focus on the generalization and sample size requirement issues in feedforward networks. We first introduce two different frameworks for studying the generalization issue: Vapnik's theory and Moody's approach. We point out the advantages and disadvantages of the two frameworks. We then extend Moody's model to a more general setting. Monte Carlo experiments are conducted to verify both Moody's result and our extension, and to demonstrate the role of the weight-decay regularization in reducing the effective number of parameters in feedforward networks. At the end of this chapter, we summarize four different practical techniques for improving the generalization ability of feedforward networks.

5.1 Vapnik's Theory

Vapnik's theory [170] provides a mathematical framework for studying the generalization issue. The theory is based on the notion of Vapnik-Chervonenkis (VC) dimension.

5.1.1 Vapnik-Chervonenkis Dimension

The VC dimension [170] (or *capacity* [26], *index* [31]) is defined as the maximum number of points in the input space that can be given an arbitrary Boolean label by the functions in $\mathcal{F}$. It measures the power and complexity of the given class of functions. We can define another quantity $\Delta_{\mathcal{F}}(\mathcal{T}_n)$ as the number of distinct dichotomies of $\mathcal{T}_n$ induced by functions $f \in \mathcal{F}$. Let $\Delta_{\mathcal{F}}(n)$ denote the maximum of $\Delta_{\mathcal{F}}(\mathcal{T}_n)$ over all $\mathcal{T}_n \in R^d$ of cardinality n. It is obvious that the maximum value of $\Delta_{\mathcal{F}}(n)$ is 2^n. Therefore, if V is the VC dimension of $\mathcal{F}$, then $\Delta_{\mathcal{F}}(V) = 2^V$, and V is the largest n such that $\Delta_{\mathcal{F}}(n) = 2^n$.

The definition of the VC dimension is not difficult to understand, but computing it for classes of functions is not trivial. A substantial amount of research has

been done on computing the VC dimensions of different types of classifiers in the pattern recognition literature. For example, the VC dimension of a linear classifier in R^d is $(d+1)$ [26, 175, 31]; the VC dimension of a quadratic classifier in R^d is $\left[2d + \frac{d(d-1)}{2} + 1\right]$, which can be derived by converting the quadratic classifier into a generalized linear classifier [34]; Baum and Haussler [14] have proved that the VC dimension of a multilayer feedforward network with hard-limiting units is bounded by

$$V < 2W \log_2(eN), \tag{5.4}$$

where W and N are the number of weights and the number of nodes in the network, respectively, and e is the base of the natural logarithm. It is also shown in [14] that $\Delta_F(n) \leq (Nen/W)^W$ for all $n \geq W$. The values of W and N are determined by the topology of the network. Suppose the number of nodes, N, is fixed. Then, the bound for the VC dimension of a feedforward network with hard-limiting units is linear in W, the number of free parameters in the network. Therefore, a weight pruning method can efficiently reduce the bound for the VC dimension. Computing the VC dimension, or even a bound on the VC dimension of a feedforward network with continuous units is very complicated. Haussler [60] has shown that the bound for the VC dimension of a feedforward network with continuous units is still linear in the number of weights in the network when the number of nodes is fixed.

5.1.2 Bound on Deviation of True Expected Error From Empirical Error

Vapnik [170] has established a relationship between the true expected error probability and the empirical error probability measured on the training data for a set of binary-valued functions. Theorem 1 provides this relationship.

Theorem 1 [170]: *Let $F(\mathbf{x}, \alpha)$ be the class of classification functions of bounded VC*

dimension V, where α is the set of parameters involved in the class of functions, and let the frequency of errors computed on the training patterns of size $n > V$ equal $\varepsilon_{\alpha_{emp}}$. Then, with probability $1 - \delta$, one can assert that the following bound is valid for all the functions in the class $F(\mathbf{x}, \alpha)$.

$$\varepsilon_\alpha \leq \varepsilon_{\alpha_{emp}} + 2\frac{V(\ln(2n/V) + 1) - \ln(\delta/12)}{n}\left(1 + \sqrt{1 + \frac{n\varepsilon_{\alpha_{emp}}}{V(\ln(2n/V) + 1) - \ln(\delta/12)}}\right). \tag{5.5}$$

In Theorem 1, $(1 - \delta)$ is the confidence level. The proof of Theorem 1 is very involved. However, the qualitative explanation is rather intuitive. As the number of training patterns n approaches infinity, the empirical error probability for all the functions in the class $F(\mathbf{x}, \alpha)$ will simultaneously converge to the true expected error probability, because the second term on the right-hand side of inequality in Eq. (5.5) will approach zero. Furthermore, the rate of convergence of the empirical error probability to the true expected error probability is determined by the ratio, $\frac{n}{V}$; the larger the ratio, the faster the convergence. It is very important to note that Theorem 1 is independent of the distributions of the underlying classes. This implies that the theorem is valid for any classification problem. While this is a very powerful aspect of the theorem, the bound that the theorem provides is very conservative because it covers the worst-case distribution in order to guarantee the uniform convergence of the empirical error probability to the true expected error probability. Theorem 1 does not address the question of whether $F(\mathbf{x}, \alpha)$ contains the true optimal solution f^*. Increasing the size of the set $F(\mathbf{x}, \alpha)$ will increase the probability of containing f^*. However, doing this may increase the value of the VC dimension, thus sacrificing the rate of convergence.

5.1.3 Bounds on Sample Size Requirements

Theorem 1 can be stated in many different ways to reveal some hidden properties, such as the number of training patterns required for a valid generalization. For example, Blumer [17] has shown the following bound for the number of training patterns.

Theorem 2 [17]: *Let $F(\mathbf{x}, \alpha)$ be the class of classification functions of a bounded VC dimension V, and for each rule $f(\mathbf{x}, \alpha)$ let the frequency of errors computed on the training set of size n equal to $\hat{\varepsilon}_\alpha$. Then, for any n satisfying*

$$n \geq \max\left\{\frac{8}{\gamma^2\varepsilon} ln\frac{8}{\delta},\ \frac{16V}{\gamma^2\varepsilon} ln\frac{16}{\gamma^2\varepsilon}\right\}, \tag{5.6}$$

the probability that there exists a classification function in $F(\mathbf{x}, \alpha)$ with $\varepsilon_\alpha \geq \varepsilon$, such that $\hat{\varepsilon}_\alpha \leq (1-\gamma)\varepsilon_\alpha$ is at most δ.

From Theorem 2, we can see that the number of training patterns required for a valid generalization increases linearly with the VC dimension of the class of classifiers. Since the upper bound for the VC dimension of feedforward networks is roughly linear in the total number of weights (if the number of units N is fixed), the sample size requirement is also linear in the total number of weights. This provides a theoretical justification of the efficiency of weight pruning methods.

These theoretical results are of little practical value, because the bound for the number of training patterns required for a valid generalization is very loose. For example, for a linear classifier, $V = d+1$. If we choose $\varepsilon_\alpha = \varepsilon = \gamma = 0.1$, the number of training samples obtained from Theorem 2 should be at least $155,000 \times (d+1)$, which is much more than approximately $10 \times (d+1)$, a recommended rule of thumb in pattern recognition practice [80]. Furthermore, this bound is a sufficient condition for a valid generalization. This means that the network trained with a fewer number of training patterns may also lead to a valid generalization.

5.2 Moody's Approach

Vapnik's theory provides a bound on the maximum deviation of the expected true probability of error from the empirical probability of error* To derive the exact relationship between the expected true error and empirical error is extremely difficult, if not impossible. However, under some rather restrictive assumptions, a data-dependent relationship can be established.

It is well known in the statistical literature [2, 8, 37] that for the linear model, $y = \mu(\mathbf{x}) + \varepsilon$, where the noise term ε is drawn from a noise distribution (not necessarily Gaussian) with zero-mean and variance σ^2, the relationship between the expected mean squared errors (MSEs) on training set and test set is governed by

$$\langle E_{test} \rangle_{\mathcal{T}_n \mathcal{T}'_n} = \langle E_{training} \rangle_{\mathcal{T}_n} + 2\sigma^2 \frac{p}{n}, \tag{5.7}$$

where p is the number of free parameters in the linear model, $p = (d + 1)$, and d is the input dimensionality. $\mathcal{T}_n$ and $\mathcal{T}'_n$ are the sets of training and test patterns, respectively.

This result can also be applied to an L-layer feedforward network with a linear activation function. Let n_l be the number of nodes in layer l, $l = 0, 1, \cdots, L$, $n_0 = d$ ($l = 0$ denotes the input layer). The total number of free parameters is $N_{free} = \sum_{l=1}^{L}(n_{l-1} + 1)n_l$. Each layer implements a linear mapping, $\mathbf{o}_l = A_l \mathbf{o}_{l-1}$, $l = 1, 2, \cdots, L$, where A_l is an $n_l \times (n_{l-1} + 1)$ weight matrix for layer l, and $\mathbf{o}_0 = \mathbf{x}$ and $\mathbf{o}_L = \mathbf{y}$. Due to the fact that the composition of linear functions remains a linear function, the whole network computes a linear mapping $\mathbf{y} = A\mathbf{x}$, where A is a $n_L \times (d + 1)$ matrix, $A = A_L A_{L-1} \cdots A_1$. If no other constraints on f are specified, then the effective number of parameters (which will be defined shortly) is $n_L \times (d+1)$.

*Vapnik's theory [170] also provides bound on the maximum deviation of the expected true squared-error from the empirical square error.

When $n_L = 1$ ($\mathbf{y}$ is a scalar), $N_{eff} = d + 1$, which is the number of free parameters in a linear classifier (two-class case).

5.2.1 "Effective" Number of Parameters

Motivated by the above linear model, Moody [126] introduced the notion of the "effective" number of parameters for nonlinear models. It is defined as

$$p_{eff} = \frac{n}{2\sigma^2}[\langle E_{test}\rangle_{\mathcal{T}_n\mathcal{T}'_n} - \langle E_{training}\rangle_{\mathcal{T}_n}]. \tag{5.8}$$

It is the effective number of parameters, not the number of free parameters, that determines the complexity of networks (functions), and has a direct effect on the sample size requirement and generalization ability. Analogous to the VC dimension, it is a complexity measure of a nonlinear system. However, unlike the VC dimension, the effective number of parameters is dependent on the data and the training algorithm.

5.2.2 Direct Relationship between Expected MSEs on Training and Test Sets

Moody [126] extended the results for linear models to the following nonlinear model. Consider a set of n real-valued training pattern pairs $\mathcal{T}_n = \{(\mathbf{x}_i, y_i)|i = 1, 2, \cdots, n\}$ drawn from a stationary distribution $\Xi(\mathbf{x}, y)$, where $\mathbf{x}_i \in R^d$, $i = 1, 2, \cdots, n$. Without a loss of generality, assume that $y_i \in R$. These patterns can be viewed as being generated according to the additive model:

$$y = \mu(\mathbf{x}) + \varepsilon, \tag{5.9}$$

where ε is the *i.i.d.* noise term with zero mean and variance σ^2 which is sampled with distribution $\Phi(\varepsilon)$ (not necessarily Gaussian), and $\mu(\mathbf{x})$ is the conditional mean,

an unknown nonlinear function of $\mathbf{x}$.

Suppose that a feedforward network is used to estimate this unknown function. The weights in the network are determined by a training algorithm to minimize the following regularized cost function (in Chapter 6, we will survey various forms of the regularized cost function, $E_\lambda(\mathbf{w}, \mathcal{T}_n)$, and the stabilizer P).

$$E_\lambda(\mathbf{w}, \mathcal{T}_n) = nE(\mathbf{w}, \mathcal{T}_n) + \lambda \|Pf\|^2, \tag{5.10}$$

where

$$E(\mathbf{w}, \mathcal{T}_n) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(\mathbf{w}, \mathbf{x}_i))^2. \tag{5.11}$$

Moody established the following relationship between the expected MSEs on training set $\mathcal{T}_n$ and test set $\mathcal{T}_n' = \{(\mathbf{x}_i, y_i')|i = 1, 2, \cdots, n\}$.

$$\langle E_{test}\rangle_{\mathcal{T}_n\mathcal{T}_n'} \approx \langle E_{training}\rangle_{\mathcal{T}_n} + 2\sigma^2\frac{p_{eff_y}(\lambda)}{n}, \tag{5.12}$$

where $p_{eff_y}(\lambda)$ is the effective number of parameters which is a function of the regularization parameter λ. The value of $p_{eff_y}(\lambda)$ can be computed using

$$p_{eff_y}(\lambda) = \frac{1}{2}\sum_{ijk} T_{ij}U_{jk}^{-1}T_{ik}, \tag{5.13}$$

where

$$T_{ij} = \frac{\partial}{\partial y_i}\frac{\partial}{\partial w_j}(nE(\mathbf{w}, \mathcal{T}_n)), \tag{5.14}$$

$$U_{jk} = \frac{\partial}{\partial w_j}\frac{\partial}{\partial w_k}E_\lambda(\mathbf{w}, \mathcal{T}_n). \tag{5.15}$$

Note that the value of $p_{eff_y}(\lambda)$ is evaluated at the equilibrium point of the regularized cost function, $E_\lambda(\mathbf{w}, \mathcal{T}_n)$, minimized using a training algorithm, such as the backpropagation algorithm. We can see that the effective number of parameters $p_{eff_y}(\lambda)$ differs

from the true number of free parameters p and depends on the nonlinearity of the model, the stabilizer and the regularization parameter. However, it is not easy to observe the relationship between $p_{eff_y}(\lambda)$, p and λ.

In practice, we never know the variance of noise, σ^2. So, it must be estimated from the training data. Moody [125] proposed to use

$$\hat{\sigma}^2 = \frac{n}{n - \hat{p}_{eff_y}(\lambda)} E_{train}(\mathbf{w}, \mathcal{T}_n) \tag{5.16}$$

as an estimate of σ^2.

5.2.3 Sample Size Requirement

Once we obtain the effective number of parameters, the samples size requirement becomes obvious. If we follow the rule of thumb that the number of training patterns should be at least 5 to 10 times larger than the effective number of parameters, i.e., $n > cp_{eff_y}(\lambda)$, $5 \leq c \leq 10$, then we expect that the deviation of the expected MSE on the test set and on training set will not exceed $0.4\sigma^2$. We should point out here that the value of p_{eff_y} must be estimated using a training data set, and is also a function of n. The sensitivity of p_{eff_y} to the value of n has not been studied.

5.3 Vapnik's Theorem Versus Moody's Results

Vapnik's framework and Moody's approach have many interesting and rather contrasting properties:

1. Vapnik's theory provides a universal framework. It is distribution-free (no data model is assumed), and it does not depend on the training algorithm. In some sense, these properties are desirable, but these properties also make the bounds

too conservative because now the worst-case behavior must be considered. Therefore, the practical applicability of these bounds is questionable. For example, these bounds provide very little guidance even in the case of multivariate Gaussian distributions. In contrast, Moody's approach provides an approximate direct relationship. However, Moody's approach makes rather restrictive assumptions on data model, training set and test set (see Section 5.4). It also depends on the training algorithm.

2. Vapnik's theory offers only the bound for the maximum deviation of the true error from empirical error, and the bound for the sample size requirement. However, Moody's approach can establish an approximate direct relationship between the expected MSEs on the training and test data sets.

3. Vapnik's theory [170] can deal with both classification error and MSE, whereas Moody's approach only applies to the MSE.

4. Moody's approach takes regularization into consideration, while Vapnik's theory does not.

5. In Vapnik's framework, any classifier (or system), no matter how complicated, is characterized by a single number, i.e., the VC dimension. The VC dimension is independent of data and learning algorithm. Therefore, the VC dimension is an inherent property of the system. On the other hand, Moody's notion of the effective number of parameters in a nonlinear system (e.g., feedforward networks) depends not only on the system itself, but also on the data and training algorithm. Therefore, it is not an inherent property of the system.

   The capacity (VC dimension) of feedforward networks is not likely to be fully exploited by a learning algorithm, such as the backpropagation algorithm. Kraaijveld [98] studied the effect of backpropagation algorithm on the generalization

properties of multilayer feedforward networks and found that the search capability of the backpropagation algorithm (or any other iterative algorithm) can not fully exploit the theoretical capacity of multilayer feedforward networks. Based on this observation, the notion of *effective* capacity (or empirical VC dimension) which depends on both the data and training algorithm has been introduced [171, 98]. Kraaijveld proposed a simple but computationally intensive procedure to estimate the effective capacity of a classifier. His simulation showed that the effective capacity can be orders of magnitude smaller than the bound for the true capacity of a multilayer feedforward network. Moody's notion of effective number of parameters can be viewed as a sort of effective capacity of feedforward networks for a specific data model and a specific training algorithm.

6. Vapnik's theory only captures the difference between the true and empirical errors, and provides no statement about the value of the true error itself. In contrast, Moody's approach can predict the true expected MSE if the restrictive assumptions on the data model, training and test sets are valid. Moody [125] proposed a generalized prediction error (GPE) for this purpose.

$$GPE(\lambda) = E_{train}(\mathbf{w}, \mathcal{T}_n) + 2\hat{\sigma}^2 \frac{\hat{p}_{eff_y}(\lambda)}{n}. \tag{5.17}$$

Although their practical applicability is questionable, both Vapnik's framework and Moody's approach provide considerable insights into how various quantities (e.g., the true error, empirical error, number of training patterns, expected test and training set MSEs, nonlinearity, and regularization) interact with each other. These results help us to understand the inner workings of the "black-box" (feedforward networks).

5.4 Extension of Moody's Model

The detailed derivation of Moody's result (Eq. (5.12)) has not yet been published. We have derived, under Moody's assumptions, a more general formula for the number of effective parameters (See Appendix A).

$$p'_{eff_y}(\lambda) = \frac{1}{2}\text{trace}[TU^{-1}V^T], \tag{5.18}$$

where

$$T = \frac{\partial^2}{\partial \mathbf{y} \partial w} nE(\mathbf{w}, \mathcal{T}_n), \tag{5.19}$$

$$V = \frac{\partial^2}{\partial \mathbf{y} \partial w} E_\lambda(\mathbf{w}, \mathcal{T}_n), \tag{5.20}$$

and

$$U = \frac{\partial^2}{\partial \mathbf{w}^2} E_\lambda(\mathbf{w}, \mathcal{T}_n). \tag{5.21}$$

Note that $\mathbf{y}$ is a vector, $\mathbf{y} = [y_1, y_2, \cdots, y_n]^T$, where n is the number of training patterns. When there are no variables $y_i, i = 1, 2, \cdots, n$ involved in the stabilizer P, i.e., the partial derivative of the regularized term in $E_\lambda(\mathbf{w}, \mathcal{T}_n)$ with respect to y_i is zero for all $i = 1, 2, \cdots, n$, then $p'_{eff_y}(\lambda) = p_{eff_y}(\lambda)$. From now on, we assume that the regularizer does not contain variables $y_i, i = 1, 2, \cdots, n$.

Moody's derivation (Eq. 5.12) is based on the following rather restrictive assumption, which largely limits its applicability. The test set differs from the training set only in the observed value y which is subject to the additive noise ε_y, but the input component $\mathbf{x}$ in the corresponding training pair and test pair must be the same. This assumption may be true for some regression situations where observations are obtained from a set of fixed measurement points. Unfortunately, it is generally violated in most pattern recognition problems. However, it would still be valuable to gain some insight on the effective number of parameters for pattern recognition problems.

Now, we try to relax this restrictive assumption in Moody's model. Consider that the sampling points $\mathbf{x}_i'$ are also subject to an additive noise model.

$$\mathbf{x}' = \mathbf{x} + \varepsilon_x, \tag{5.22}$$

where ε_x is the *i.i.d.* noise vector with mean zero and $d \times d$ covariance matrix $\Lambda_x = \sigma_x^2 I$ (diagonal matrix).

We obtain the following relationship which governs the expected MSEs on training set $\mathcal{T}_n$ and test set $\mathcal{T}_n' = \{(\mathbf{x}_i', y_i') | i = 1, 2, \cdots, n\}$ (see Appendix A).

$$\langle E_{test} \rangle_{\mathcal{T}_n \mathcal{T}_n'} \approx \langle E_{training} \rangle_{\mathcal{T}_n} + 2\sigma_{eff}^2 \frac{p_{eff}(\lambda)}{n}, \tag{5.23}$$

where

$$p_{eff}(\lambda) = \frac{\sigma_y^2}{\sigma_{eff}^2} p_{eff_y}(\lambda) + \frac{\sigma_x^2}{\sigma_{eff}^2} p_{eff_x}(\lambda), \tag{5.24}$$

$$\sigma_{eff}^2 = \sigma_y^2 + \sigma_x^2, \tag{5.25}$$

$$p_{eff_y}(\lambda) = \frac{1}{2} \text{trace}[TU^{-1}T^T], \tag{5.26}$$

$$p_{eff_x}(\lambda) = \frac{1}{4} \text{trace}[G], \tag{5.27}$$

and

$$G = \sum_{i=1}^{n} \frac{\partial^2}{\partial_{\mathbf{x}_i}^2} nE(\mathbf{w}, \mathcal{T}_n). \tag{5.28}$$

Property 1: *$p_{eff_y}(\lambda)$ is non-negative at minimal points of the regularized cost function $E_\lambda(\mathbf{w}, \mathcal{T}_n)$ with respect to the weights in the feedforward network.*

Moody [126] presumed that $p_{eff_y}(\lambda)$ is non-negative without providing a proof. We now prove this property.

Proof: At a minimal (either local or global) point $w = w^*$ of the regularized cost function $E_\lambda(\mathbf{w}, \mathcal{T}_n)$, the Hessian matrix $U_{p \times p}$ defined in Eq. (5.21) is positive definite.

Therefore, U^{-1} is also positive-definite, that is, for any p-dimensional vector $\mathbf{a} \neq 0$, $\mathbf{a}^T U^{-1}\mathbf{a} > 0$. Thus,

$$p_{eff}(\lambda) = \frac{1}{2}\text{trace}[TU^{-1}T^T] = \frac{1}{2}\sum_{i=1}^{p}(\mathbf{a}_i^T U^{-1}\mathbf{a}_i) \geq 0, \tag{5.29}$$

where $\mathbf{a}_i$ is the i^{th} row vector of the matrix T.

Property 2: *$p_{eff_y}(\lambda)$ is non-negative at pattern-wise minimal points† of the cost function (without regularization) $E(\mathbf{w}, \mathcal{T}_n)$ with respect to the weights in the feedforward network.*

Proof: Let us first introduce an intermediate variable h_j^k which is called the net input to the j^{th} node in the first hidden layer, $h_j^k = \sum_{i=1}^{d} w_{ij}x_i^k$, when the k^{th} pattern $\mathbf{x}^k = [x_1^k, x_2^k, \cdots, x_d^k]$ is presented. We denote $E = \frac{1}{n}\sum_{k=1}^{n} E_k$. Now compute

$$\frac{\partial^2 E}{\partial {x_i^k}^2} = \sum_j \frac{\partial^2 E}{\partial {h_j^k}^2}\left(\frac{\partial h_j^k}{\partial x_i^k}\right)^2 + \sum_j \frac{\partial E}{\partial h_j^k}\frac{\partial^2 h_j^k}{\partial {x_i^k}^2}. \tag{5.30}$$

Since $\frac{\partial^2 h_j^k}{\partial {x_i^k}^2} = 0$ and $\frac{\partial h_j^k}{\partial x_i^k} = w_{ij}$, we obtain

$$\frac{\partial^2 E}{\partial {x_i^k}^2} = \sum_j w_{ij}^2 \frac{\partial^2 E_k}{\partial {h_j^k}^2}. \tag{5.31}$$

Similarly, we have

$$\frac{\partial^2 E_k}{\partial w_{ij}^2} = \frac{\partial^2 E_k}{\partial {h_j^k}^2}\left(\frac{\partial h_j^k}{\partial w_{ij}}\right)^2 + \frac{\partial E_k}{\partial h_j^k}\frac{\partial^2 h_j^k}{\partial w_{ij}^2} = {x_i^k}^2 \frac{\partial^2 E_k}{\partial {h_j^k}^2}. \tag{5.32}$$

At a pattern-wise minimal point of the cost function $E(\mathbf{w}, \mathcal{T}_n)$, $\frac{\partial^2 E_k}{\partial w_{ij}^2} > 0$. Thus, $\frac{\partial^2 E}{\partial {h_j^k}^2} > 0$, for $x_i^k \neq 0$. In the case of $x_i^k = 0$, by the continuity of $\frac{\partial^2 E}{\partial {h_j^k}^2}$, $\frac{\partial^2 E}{\partial {h_j^k}^2}$ can not be negative at $x_i^k = 0$, because $\frac{\partial^2 E}{\partial {h_j^k}^2} > 0$, for any $x_i^k = \varepsilon \neq 0$, where ε can be arbitrarily

†Suppose that a pattern-wise iterative algorithm is used. At each iteration, the algorithm tries to minimize the square error for the pattern which is currently presented.

small. From Eq. (5.31), we prove that $\frac{\partial^2 E}{\partial x_i^{k^2}} \geq 0$, which leads to $p_{eff_x}(\lambda) \geq 0$.

Note that we have not proved that the property $p_{eff_x}(\lambda) \geq 0$ still holds at a minimal point of the regularized cost. We should also point out that a pattern-wise minimal point is also a minimal point of E, but not vice versa. The positiveness of p_{eff_x} at non-minimal points, batch-mode minimal points, and the minimal points of the regularized cost function, needs to be investigated further. However, we find that in our simulation, p_{eff_x} is always positive in the neighborhood of the minimal points of the regularized cost function. The significance of these properties is that i) the noise in the observation will always (with or without regularization) increase the effective number of parameters, and ii) the noise in the sampling points of test data will always increase the effective number of parameters without using regularization.

5.5 Simulation Results

In this section, we use the Monte Carlo method to verify the relationship (in Eq. (5.12)) established by Moody and our extension (in Eq. (5.23)). We also demonstrate the role of the weight-decay regularization [64] in reducing the effective number of parameters in feedforward networks.

The data used in our Monte Carlo experiments are generated from the following nonlinear system.

$$y_i = \sin(x_{i1} + x_{i2}) - \sin(2x_{i1}) - \sin(2x_{i2}) + \varepsilon_i, \ i = 1, 2, \cdots, n, \tag{5.33}$$

where x_{i1} and x_{i2} are uniformly distributed in $[-1.0, +1.0]$, and ε_i is drawn from a uniform distribution in $[-0.1, +0.1]$. In Moody's model, the x-component of patterns in the test set must be the same as in the training set. We choose the 8×8 equally-spaced grid points in the $[-1.0, +1.0] \times [-1.0, +1.0]$ square area as 64 sampling points

($n = 64$). In the extended model, the sampling process of these grid points is subject to a noise source which is again uniformly distributed in the $[-0.1, +0.1] \times [-0.1, +0.1]$ square area. The weight-decay regularization term is used in the cost function.

$$E_\lambda(\mathbf{w}, \mathcal{T}_n) = \sum_{i=1}^{n} (y_i - f(\mathbf{w}, \mathbf{x}_i))^2 + \lambda \|\mathbf{w}\|^2, \tag{5.34}$$

A total of 40 training data sets are generated from the model. For each training data set, 10 independent test sets for both Moody's model and our extended model are drawn. A 2-layer network with two input nodes, 10 hidden nodes, and one output node, is trained on each training data set 10 times with different initial random weights. The empirical effective number of parameters, p_{eff-em}, is estimated using Eq. (5.8), in which the expected training set MSE and the expected test set MSE are estimated from these 400 and 4000 Monte Carlo trials, respectively. The value of p_{eff-em} is compared with the averaged value of p_{eff} obtained either from Eq. (5.13) or from Eq. (5.24). Note that the total number of free parameters in the network is 41.

It is required (by the derivation in Appendix A) that all the Hessian matrices (U, T, and G) should be evaluated at a minimal point. However, in simulation as well as in practice, the training algorithm never reaches an exact minimal point (all the derivatives of the cost function with respect to weights are zero). The training procedure is usually terminated when the network reaches a neighborhood of a minimal point. Therefore, it is desirable to investigate the sensitivity of the effective number of parameters to a small perturbation to the minimal point in the weight space. As a case study, we first train the network using 4000 cycles (one pattern per cycle) with the values of the learning rate η and momentum factor γ in the backpropagation algorithm set to 0.05 and 0.1, respectively. Then we train the network using another 2000 cycles with $\eta = 0.01$ and $\gamma = 0.01$ which are very small because the square

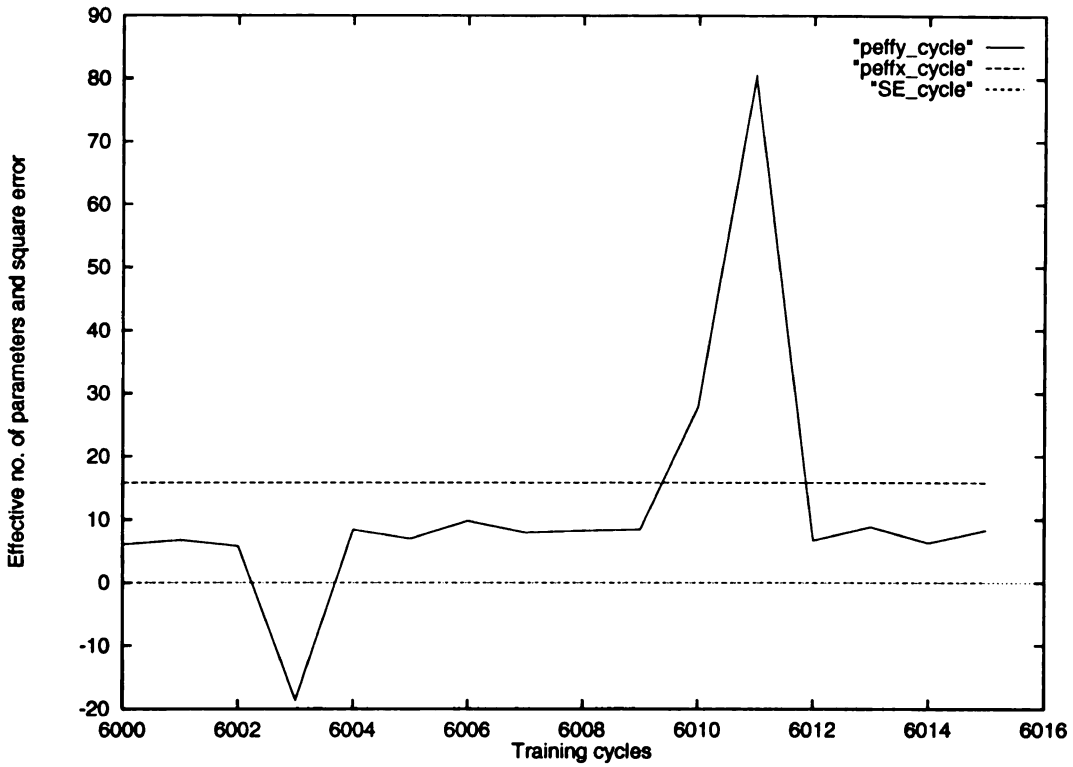


Figure 5.1. Values of p_{eff_x} and p_{eff_y} at the 15 consecutive cycles starting with 6000. The values of the square error (`SE-cycle` curve) on the training data set are also shown.

error on the training data set has stabilized (see Figure 5.1). The square error cost function with no regularization term is used in this case study. The values of p_{eff_x} and p_{eff_y} at the next 15 consecutive cycles are shown in Figure 5.1. We can see that p_{eff_x} (`peffx_cycle` curve) and the square error (`SE_cycle` curve) on the training set are almost constant. However, the value of p_{eff_y} (`peffy_cycle` curve) oscillates with a large amplitude as the number of cycles increases. It even takes negative values (the positiveness of p_{eff_y} is guaranteed only at the minimal point). This observation indicates that p_{eff_y} is very sensitive to a small perturbation in the minimal points in the weight space. The standard deviations of the estimates of p_{eff_x}, p_{eff_y} and p_{eff} in our Monte Carlo experiments are shown in Figure 5.3. This undesirable behavior makes the estimation of p_{eff_y} very difficult. Fortunately, we notice that in spite of

these outliers (e.g., negative values and extremely large values), most of the values are rather close. Therefore, a robust estimator of p_{eff_y} is desirable which can detect and remove these outliers. In our Monte Carlo trials (each trial uses the same training procedure as in the case study), we simply use the following heuristics to discard those Monte Carlo trials which satisfy i) $p_{eff_y} < 0$ or $p_{eff_y} > 1.5p_{free}$ ($p_{free} = 41$); or ii) the square error on the training set is two times larger than the average square error on the training set. In the second case, it is more likely that the network does not reach a minimal point. A more reliable method is to check if all the derivatives of the cost function with respect to weights are smaller than some threshold.

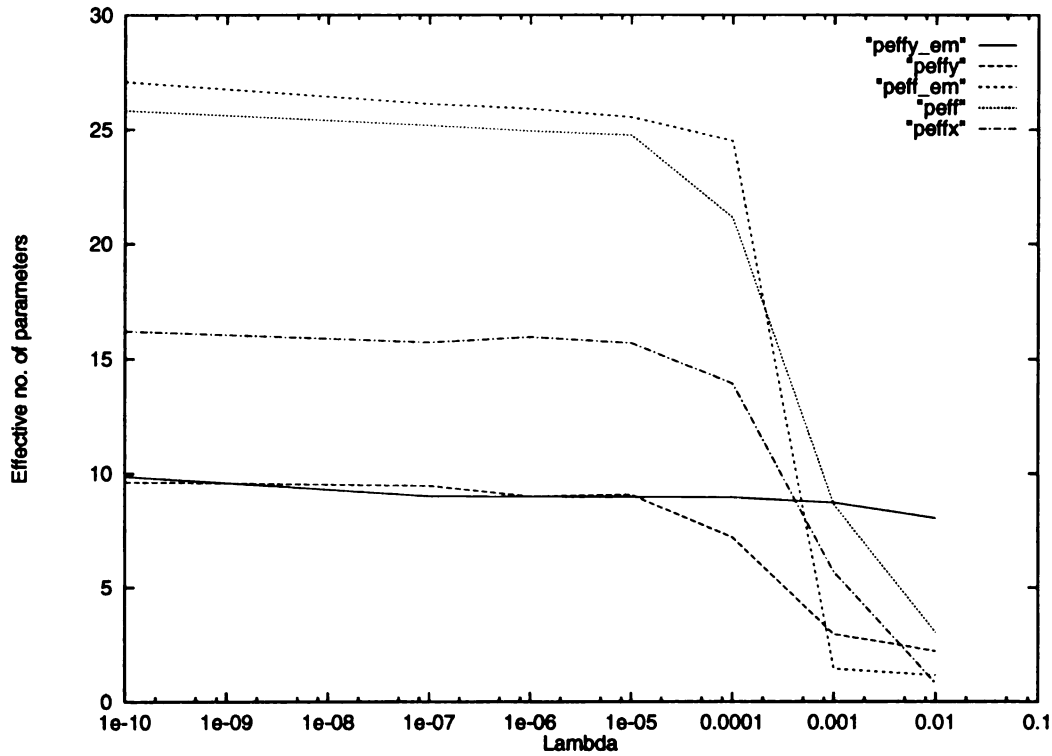


Figure 5.2. Empirical values of $p_{eff_{y-em}}$ and p_{eff-em}, and the average values of p_{eff_y}, p_{eff_x} and p_{eff} versus the regularization parameter λ.

Figures 5.2 and 5.3 show the average values and standard deviations, respectively,

of the effective number of parameters in the network (with 41 free parameters) versus the regularization parameter λ. These values are estimated from Eq. (5.8) (**peffy_em** and **peff_em** curves) and from Eqs. (5.24) (**peff** curve), (5.27) (**peffx** curve) and (5.26) (**peffy** curve). Note that the leftmost point on each curve corresponds to $\lambda = 0$. We obtain only a single value (for a fixed value of λ and a fixed model) for the empirical estimate of the effective number of parameters from our Monte Carlo experiments, because the expectation in Eq. (5.8) is evaluated by taking an average over all the valid trials. Therefore, the standard deviation information is not available for p_{eff-em} and p_{eff_y-em} in Figure 5.3. But, the standard deviation of the empirical estimate should be a decreasing function of the number of trials used to evaluate the mathematical expectation. We can make the following observations from Figures 5.2 and 5.3.

1. The effective number of parameters (p_{eff_y} for Moody's model and p_{eff} for the extended model) is smaller than the number of free parameters in the network ($p_{free} = 41$) even without an explicit regularization term ($\lambda = 0$). In Moody's noise-free-input model, the effective number of parameters (p_{eff_y}) is only about one-fourth of p_{free}. However, in the extended model, the difference between p_{eff} and p_{free} becomes smaller. This indicates that the notion of the effective number of parameters is very data-dependent.

2. The relationship (Eq. (5.12)) established by Moody causes a large discrepancy if the input components in the test data set and training data set are not the same.

3. The effective number of parameters generally decreases as the value of λ increases. There is a sudden drop in the effective number of parameters when λ changes from 0.0001 to 0.001. This demonstrates the role of weight-decay regularization technique in improving the generalization ability of feedforward

networks.

4. The theoretical estimates of p_{eff} and p_{eff_y} agree quite well with the empirical estimates (Eq. (5.8)) of p_{eff-em} and p_{eff_y-em}, respectively, when λ is small. However, this is not true if the outliers are not removed. Note that one advantage of using theoretical estimate as opposed to the empirical estimate is that only the training data set is required to evaluate p_{eff}, p_{eff_x} and p_{eff_y}.

5. The estimation of the effective number of parameters (p_{eff_y}) using Eq. (5.26) is not reliable; the estimator has a large variance (see Figure 5.3). This causes a large variance in estimating p_{eff} using Eq. (5.24) even though the estimation of p_{eff_x} using Eq. (5.27) is very reliable.

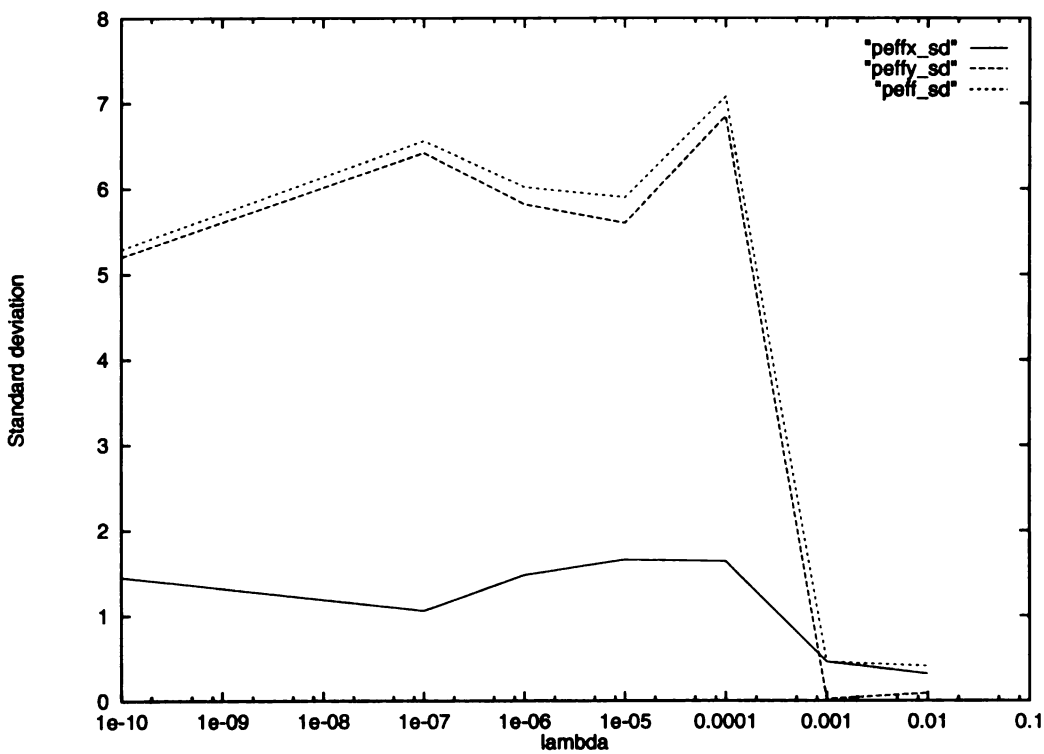


Figure 5.3. Standard deviations of the estimates of p_{eff_x}, p_{eff_y} and p_{eff} with respect to the regularization parameter λ.

5.6 Improving Generalization Abilities of Feedforward Networks

A common property shared by most artificial neural networks is that they involve a large number of free parameters. For example, in an L-layer feedforward network, the number of free parameters (connection weights and biases) is $N_{free} = \sum_{l=1}^{L}(n_{l-1}+1)n_l$, where n_l is the number of nodes in layer l, $l = 0, 1, \cdots, L$. These free parameters have to be estimated from training patterns. In Sections 5.1 and 5.2, we have shown that the number of training patterns should be sufficiently large in order to guarantee a reliable estimate of the free parameters thus leading to a valid generalization. Unfortunately, collecting a large number of training samples is expensive, time consuming, and sometimes impossible in many practical situations. Therefore, choosing a parsimonious system (with a small number of parameters) is a very important issue in network design. In addition to the good generalization ability, smaller networks (with fewer parameters) also offer several computational and hardware implementational advantages. Finding an optimal architecture or a network with a good generalization ability principally requires an exhaustive search over all possible architectures, which is not feasible. Various techniques have been proposed to avoid such an exhaustive search. These techniques can be grouped into the following four categories:

1. local connections and weight sharing,
2. adaptive network pruning,
3. adaptive network growing, and
4. regularization.

The main idea behind these techniques can be related to the principle of minimum description length [148, 32].

Methods in the first category try to reduce the number of free parameters (connection weights) through connecting a node to only a local region of its inputs or sharing the same weight among many connections [106, 104].

Adaptive network growing techniques start by training a small-sized network. The size of the network grows if it can not approximate the training set well enough. Examples of such techniques are the cascade-correlation algorithm [38], the tiling algorithm [118], neural trees [161], projection pursuit [187, 75, 76], the constructive algorithm [116], and synthesis of polynomial networks [12, 9].

In contrast to network growing techniques which start with a small network, adaptive network pruning techniques start with a large (than necessary) network, and prune its weights or nodes which are not important for the network to perform function approximation and classification. A number of approaches have been proposed for pruning weights and nodes in feedforward networks (e.g., [19, 107, 59, 159, 129, 91, 180]). Some of them, such as, the optimal brain damage [107], optimal brain surgeon [59], and skeletonization [129], are based on certain sensitivity measures of the cost function with respect to weights and nodes, while some others are guided by a set of heuristics [19, 159]. Reed [144] provides a survey of such pruning algorithms.

Regularization techniques are frequently employed to design neural networks. Typical examples are weight decay [64], non-proportional weight decay [24, 56], weight elimination [174], soft weight sharing [131], and complexity regularization [10]. Mao and Jain [112] (see Chapter 6) provide a survey of regularization techniques in designing neural networks. The role of regularization techniques is to reduce the number of *effective* parameters in a neural network although the network size is often not reduced [126, 124]. The reduction in the number of effective parameters helps neural networks to generalize well. Some of these regularization techniques may be considered as a sort of soft pruning methods [144], because the training algorithm drives

some weights and/or the outputs of nodes to zero. Regularization techniques are often combined with the adaptive pruning algorithms.

5.7 Summary

The main contributions of this chapter are: i) The advantages and disadvantages of Vapnik's framework versus Moody's approach are analyzed; ii) Moody's notion of the *effective* number of parameters has been extended to a more general noise model. We have also proved that the effective number of parameters p_{eff_y} (due to the noise in observation values) is non-negative at the minimal points of the regularized cost function, and that p_{eff_z} (due to the noise in sampling points) is non-negative at the pattern-wise minimal points of the square-error cost function; iii) The relationships established by Moody and our extension have been verified through Monte Carlo experiments. We have observed that Moody's result has a large discrepancy when the sampling points of the test data are subject to noise. Moreover, the estimation of p_{eff_y} is very unreliable. However, if the outliers in the empirical estimation can be removed using some heuristics, then the theoretical estimate agrees quite well with the empirical method; and iv) We have demonstrated the role of the weight-decay regularization in reducing the effective number of parameters.

CHAPTER 6

Regularization Techniques in Neural Networks

In Chapter 5, we analyzed the generalization ability of feedforward neural networks, and pointed out four types of techniques for improving the generalization ability. This chapter is devoted to one of these four types of techniques: the regularization techniques. In Section 5.4, we established a direct relationship between expected MSE on training set and test sets for a generic regularized cost function. This chapter will present various forms of regularized cost functions.

Design of neural networks generally involves estimating a large number of free parameters (connection weights) from a finite number of training patterns using some learning algorithm. Even though the number of training patterns is often less than the number of system parameters, many neural networks perform very well in various pattern recognition and optimization applications. Learning the free parameters in neural networks is often an *ill-posed* problem. In this chapter, we provide a systematic study of the role that regularization theory plays in neural networks and present a taxonomy of regularization techniques in neural networks: Type I (network-inherent), Type II (algorithm-inherent), and Type III (explicitly-specified) regularization. A particular neural network may employ more than one type of regularization technique.

Our systematic study leads to a general framework for neural network design and learning. We demonstrate that a variety of neural networks and learning algorithms do fit into this framework. We study the role of regularization techniques in improving robustness and generalization abilities of neural networks.

6.1 Introduction to Regularization Techniques

Many engineering and scientific problems, particularly in computer vision, are *ill-posed*. The definition of the ill-posedness was first introduced by Hadamard in the field of partial differential equations. For a long time, mathematicians felt that ill-posed systems cannot describe real phenomena, because their solution is not unique, not stable, or does not exist. It was not clear in what sense ill-posed problems could have solutions that would be meaningful in applications. Tikhonov [168] gave a precise mathematical definition of "approximate solutions" for general classes of ill-posed problems, and provided a procedure for constructing "optimal" solutions.

Many application problems (e.g., computer vision problems) can be formulated as finding the solution $z \in Z$ from observed data $u \in U$ of the following equation

$$Az = u, \tag{6.1}$$

where A is an operator (not necessarily linear), and U, Z are two metric spaces with metrics $\rho_U(u_1, u_2)$ for $u_1, u_2 \in U$ and $\rho_Z(z_1, z_2)$ for $z_1, z_2 \in Z$, respectively.

Tikhonov [168] gave the following precise definition of *well-posedness*. Eq. (6.1) is said to be well-posed on the pair of metric spaces (Z, U) if the following three conditions are satisfied:

(i) $\forall\ u \in U$, there exists a solution $z \in Z$ (Solvability);

(ii) the solution is unique (Uniqueness);

(iii) the problem is stable on the spaces (Z, U) (Stability).

Conditions (i) and (ii) guarantee the mathematical determinacy of the problem, while condition (iii) is concerned with the physical determinacy and the possibility of applying numerical methods to solve Eq. (6.1) based on approximate data. Condition (iii) assures that small changes in observed data will not move the estimated solution far away from the true solution. This condition requires the inverse operator A^{-1} to be continuous on Z. A problem that does not satisfy one or more of these three conditions is said to be ill-posed.

The main approach for solving the ill-posed problem is to reduce the solution space by using an *a priori* knowledge of the problem, thus converting the ill-posed problem to a well-posed problem. Regularization theory is a popular approach based on this idea. The *a priori* knowledge appears in the form of quantitative and qualitative constraints. In most cases, we only have some general natural constraints, such as smoothness. These constraints are usually specified using a nonnegative functional, Ωz (also called the stabilizing functional). In most computer vision problems [140], Ωz takes the form of $\|Pz\|^2$, where $\|\cdot\|^2$ is the Euclidean norm and P is called a stabilizer or a stabilizing operator.

The regularization method of solving the ill-posed problem in Eq. (6.1) can be formulated as finding z that minimizes

$$\rho_U(Az, u) + \lambda \Omega z, \tag{6.2}$$

where λ is a regularization parameter which controls the tradeoff between the degree of regularization and the faithfulness of the solution to the data. Therefore, the choice of λ is an important issue. The standard regularization theory provides techniques to choose the best value of λ [168, 167, 50]. However, this applies only when A is a linear operator, and metric ρ_U and stabilizing functional Ωz are quadratic. For general

cases, the choice of λ remains a difficult task, and is often determined empirically.

Another method to formulate the regularization method is as follows. Given an error tolerance ε, find z that satisfies $\rho_U(Az, u) < \varepsilon$, such that Ωz is minimized. This method tries to find the most "regular" solution which sufficiently "fits" the data. This method can be integrated with the standard classical optimization problem. Some classical optimization techniques, such as sequential quadratic programming, can be used. We will show an example later.

Regularization techniques are very popular in computer vision, because most early vision problems are ill-posed. Poggio et al. [140] provide an excellent review of regularization techniques in computer vision, which contains a unified theoretical framework for many of the early vision processes.

6.2 A Taxonomy of Regularization Techniques in Neural Networks

It is well-known that neural networks usually involve a large number of parameters. Estimation of these parameters is an underdetermined problem, so the second necessary condition of well-posed problems is violated, resulting in ill-posed problems. Therefore, regularization has been frequently employed in neural network paradigms. Regularization techniques play an important role in neural network performance. Unfortunately, unlike in computer vision, regularization theory has not received significant attention from the neural network community.

In this section, we will provide a systematic study of regularization techniques in neural networks. We classify various regularization techniques into three types: Type I: Architecture-Inherent, Type II: Algorithm-Inherent, and Type III: Explicitly-Specified. The following three subsections will give a detailed description of these three types of regularization techniques.

6.2.1 Type I: Architecture-Inherent Regularization

Type I regularization involves specifying a network architecture for a given problem. This regularization scheme is obvious, but is often ignored. For example, in supervised learning of feedforward networks, we often specify the network size and topology. This actually limits the solution to $\mathcal{F}$, a set of functions which can be implemented by the specified network architecture. Consequently, specifying the network architecture introduces some degree of regularization.

Poggio and Girosi [138] have shown the equivalence between regularization and a class of 2-layer feedforward networks called regularization networks, which are not only equivalent to generalized spline regressions, but also closely related to the Radial Basis Function networks.

Poggio and Girosi [138] have shown that the solution of the following regularized minimization problem

$$E(f) = \sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))^2 + \lambda \|Pf\|^2, \tag{6.3}$$

is given by

$$f(\mathbf{x}) = \frac{1}{\lambda}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))G(\mathbf{x}, \mathbf{x}_i) + f_0(\mathbf{x}), \tag{6.4}$$

where P is a stabilizer, $\|\cdot\|^2$ is the L^2 norm, $G(\mathbf{x}, \mathbf{x}_i)$ is Green's function centered at the point $\mathbf{x}_i$, and f_0 is in the null space of the stabilizer P. Any function in the null space will disappear when it is operated on by the stabilizer P. If P is a differential operator of order larger than m, then the null space of this P is a set of all the polynomial functions of order less than m. The form of f_0 is determined by both the stabilizer P and the boundary conditions. The following analysis ignores f_0.

Let $c_i = (y_i - f(\mathbf{x}_i))/\lambda$, then

$$f(\mathbf{x}) = \sum_{i=1}^{n} c_i G(\mathbf{x}, \mathbf{x}_i). \tag{6.5}$$

The constants c_i, $i = 1, 2, \cdots, n$ can be solved from n linear equations of Eq. (6.5) by substituting n input-output training pairs. If the stabilizer P is rotational and translational invariant, then $G(\mathbf{x}, \mathbf{x}_i)$ will be a radial function: $G(\mathbf{x}, \mathbf{x}_i) = G(\|\mathbf{x} - \mathbf{x}_i\|^2)$. In this case, the regularized solution is

$$f(\mathbf{x}) = \sum_{i=1}^{n} c_i G(\|\mathbf{x} - \mathbf{x}_i\|^2). \tag{6.6}$$

Eq. (6.6) has the same form as the Radial Basis Function network. Therefore, when we apply a Radial Basis Function network to a problem, we unintentionally introduce a rotational and translational invariant stabilizer to the function space. Regularization is inherent whenever we choose a network even though we do not specify an explicit regularization form.

Following are some of the examples of stabilizers.

(1) m^{th}-order stabilizer

$$\|Pf\|^2 = \|O^m f\|^2 = \sum_{i_1, i_2, \cdots, i_m}^{d} \int_{R^d} d\mathbf{x} (\partial_{i_1, i_2, \cdots, i_m} f(\mathbf{x}))^2, \tag{6.7}$$

where $m \geq 1$. It is rotational and translational invariant.

In this case, the Green's function in Eq. (6.4) takes the following form.

$$G(\mathbf{x}) = \begin{cases} \|\mathbf{x}\|^{2m-d} ln \|\mathbf{x}\|, & \text{if } 2m > d \text{ and } d \text{ is even,} \\ \|\mathbf{x}\|^{2m-d}, & \text{otherwise.} \end{cases} \tag{6.8}$$

In the special case of $m = d = 2$, the stabilizer takes the form

$$\|Pf\|^2 = \|O^2 f\|^2 = \int_{R^2} dxdy \left[\left(\frac{\partial^2 f}{\partial x^2} \right)^2 + \left(2 \frac{\partial^2 f}{\partial x \partial y} \right)^2 + \left(\frac{\partial^2 f}{\partial y^2} \right)^2 \right], \tag{6.9}$$

and the Green's function is the well-known "thin plate spline",

$$G(r) = r^2 \ln r, \tag{6.10}$$

where $r = (x^2 + y^2)^{1/2}$.

(2) Infinite order stabilizer

$$\|Pf\|^2 = \sum_{i=0}^{\infty} a_i \|O^i f\|^2. \tag{6.11}$$

The corresponding Green's function is

$$G(\mathbf{x}, \mathbf{x}_i) = C \exp^{-\|\mathbf{x}-\mathbf{x}_i\|^2/2\sigma^2}, \tag{6.12}$$

where C is a normalization constant. It is a Gaussian function which is frequently used in Radial Basis Function Networks.

Based on these analyses, Poggio et al. [139, 138] have proposed a regularization network. It is difficult to show what type of stabilizer leads to the solution of a regularized minimization problem to be a feedforward network with sigmoid nonlinearity in each node. However, it is clear that by choosing the standard feedforword network of a fixed size, the solution space, denoted by $\mathcal{F}$, is greatly reduced compared to the function space $\mathcal{A}$ consisting of all functions with the same number of free parameters as in the network. Furthermore, because the sigmoid function is a very smooth function (all its derivatives of any order are continuous), the solution is very smooth. Therefore, $\mathcal{F} \subset \mathcal{C}^\infty \subset \mathcal{A}$, where $\mathcal{C}^\infty$ is a set of all continuous functions with continuous

derivatives up to any order.

The neural network-based Sammon's projection (SAMANN) proposed in Chapter 2 is a good example of using Type I regularization for improving the generalization ability of the original Sammon's projection algorithm. As we discussed in Chapter 2, the original Sammon's algorithm, the positions of n patterns in the m-dimensional output space are treated as mn free parameters in an optimization problem. It does not provide an explicit function governing the relationship between patterns in the original space and in the configuration (projected) space. Therefore, it is impossible to decide where to place new d-dimensional data in the final m-dimensional configuration created by Sammon's algorithm. In other words, Sammon's algorithm has no generalization ability. However, SAMANN employs a feedforward network to govern the projection mapping from the input space to the output space. Therefore, the solution space is reduced to a set of functions which are realizable by a fixed-size feedforward network. After training is done, the network can be used to project new data. Figures 6.1 (a) and (b) show the projection maps of the **IRIS** data using a 2-layer neural network with 100 hidden nodes and the original Sammon's procedure. The network is trained on all the 150 patterns for 5,000,000 cycles (a pair of patterns per cycle). Both the maps have very similar configurations and similar values of stress. One advantage of the neural network-based projection method is that it is able to project new data after training. To demonstrate this generalization ability, we used 75 patterns, randomly chosen from the original 150 patterns, as the training set. Figure 6.2 (a) shows the projection of these 75 training patterns after training the network for 500,000 cycles. The corresponding stress is 0.0049. Figure 6.2 (b) shows the projection of all the 150 patterns using this trained network. The corresponding stress is 0.0057. Comparing Figure 6.2 (b) and Figure 6.1 (b), we find that the two projections are very similar. The relative positions of the patterns in the projected space are also roughly the same in the two configurations which indicates that the

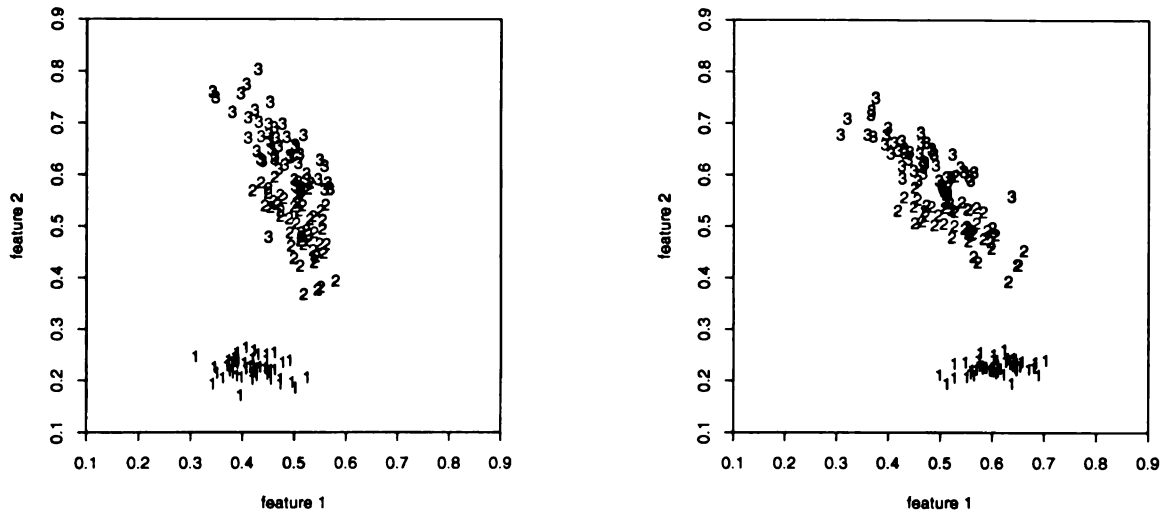


Figure 6.1. Projection maps of the IRIS data. (a) Original Sammon's algorithm, E=0.0068; (b) 2-layer neural network with 100 hidden nodes, E=0.0061.

network generalizes the projection well.

6.2.2 Type II: Algorithm-Inherent Regularization

Type II regularization is obtained by specifying a learning algorithm to train the network. This regularization scheme is inherent in the learning algorithm, hence it is not easily recognized.

In training the feedforward network using the backpropagation learning algorithm, a number of stopping criteria can be used. Typical criteria are: (i) a prespecified maximum number of iterations is reached; (ii) the squared error between the actual outputs and desired outputs is below some threshold; and (iii) change in connection weights between two successive iterations is less than some threshold. In practice, non-zero thresholds and a maximum number of iterations are specified. Therefore, when the training is terminated, a global minimum, or even a local minimum is seldom reached. Surprisingly, this helps the network to generalize well sometimes.

Sjoberg and Ljung [162] have shown that terminating the learning process without

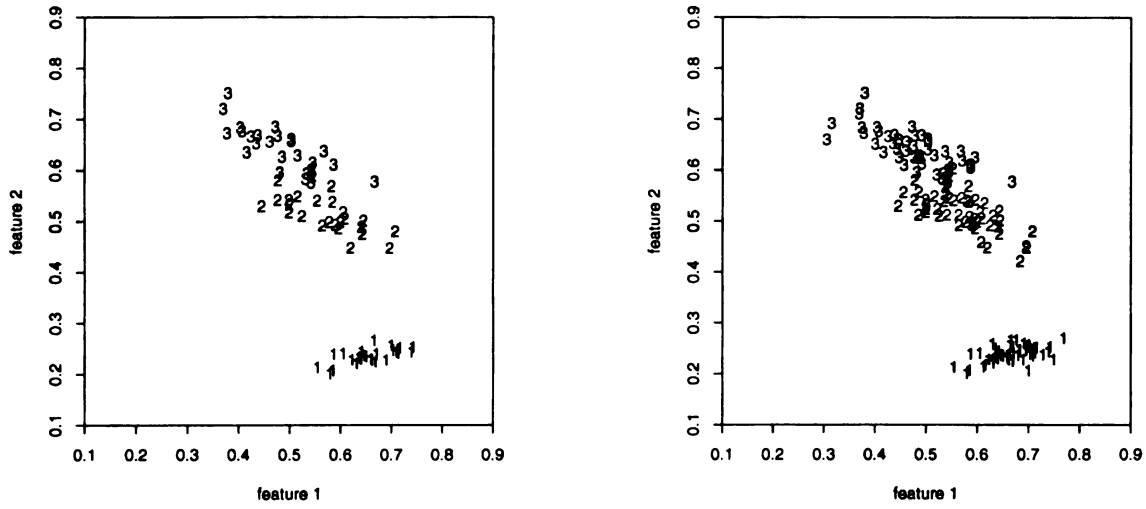


Figure 6.2. Generalization ability of the network. (a) Projection of 75 randomly chosen training patterns from the IRIS data using a 2-layer network with 100 hidden nodes, E=0.0049; (b) Projection of all the 150 patterns using the trained network in (a), E=0.0057.

reaching a local/global minimum introduces a sort of regularization, which we refer to as Algorithm-Inherent regularization in our taxonomy. Let $\mathcal{T}_n$ denote the set of n input-output training pairs, $\mathcal{T}_n = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_n, y_n)\}$, and $\mathbf{w}$ be a vector of all the weights in the network. Sjoberg and Ljung [162] have shown that the effect of an "unfinished" search for minimum is very similar to minimizing the regularized error function

$$E(\mathbf{w}, \mathcal{T}_n) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - f(\mathbf{w}, \mathbf{x}_i))^2 + \lambda \|\mathbf{w} - \mathbf{w}^0\|^2, \tag{6.13}$$

assuming that the learning rate is small, where $\mathbf{w}^0$ is the initial weight vector and λ is the regularization parameter. The regularization term constrains the amount of change in values of the connection weights. As a result, the final weight values should not deviate too far from the initial weight values. This property also indicates that the choice of initial weight values is very important in training the network.

Adding noise to the training patterns has been shown to be helpful in obtaining a good generalization [121, 67, 97, 108, 121, 128, 159]. This procedure artificially increases the number of training patterns and is frequently utilized when the training sample size is small. Reed et al. [145] have shown that adding i.i.d. Gaussian noise with zero-mean and covariance matrix $\sigma^2 I$ to the training samples is approximately (in the first-order) equivalent to minimizing the following cost function

$$E(\mathbf{w}, \mathcal{T}_n) = \sum_{i=1}^{n} (y_i - f(\mathbf{w}, \mathbf{x}_i))^2 + \sigma^2 \sum_{i=1}^{n} \|\partial_{\mathbf{x}} f(\mathbf{w}, \mathbf{x}_i)\|^2. \tag{6.14}$$

Note that the regularization term imposes a smoothness constraint on the first-order derivative of the function with respect to the input variables. Reed et al. [145] also showed that if the second-order approximation is used, then the equivalent cost function becomes

$$E(\mathbf{w}, \mathcal{T}_n) = \sum_{i=1}^{n} \left(y_i - f(\mathbf{w}, \mathbf{x}_i) - \frac{\sigma^2}{2} Tr(H_{\mathbf{x}_i}) \right)^2 + \sigma^2 \sum_{i=1}^{n} \|\partial_{\mathbf{x}} f(\mathbf{w}, \mathbf{x}_i)\|^2 + \sum_{i=1}^{n} \frac{\sigma^4}{2} Tr(H_{\mathbf{x}_i}^2), \tag{6.15}$$

where $Tr(H_{\mathbf{x}_i})$ is the trace of matrix $H_{\mathbf{x}_i}$, the Hessian matrix of functional f with respect to $\mathbf{x}$ evaluated at $\mathbf{x}_i$. The third term in Eq. (6.15) imposes a smoothness constraint on the second-order derivative (curvature) of the function to be estimated with respect to the input variables. However, this approximation also introduces a bias of $\frac{\sigma^2}{2} Tr(H_{\mathbf{x}_i})$ to the squared error.

Adding random noise to weights during training has also been proposed to increase the robustness of the network. We now show that this method also regularizes the solution by following the technique of Reed et al. [145]. When the second-order approximation is used, adding the i.i.d. Gaussian noise with zero-mean and covariance matrix $\sigma^2 I$ to the weights is equivalent to minimizing the following regularized cost

function (see Appendix B).

$$E(\mathbf{w}, \mathcal{T}_n) = \sum_{i=1}^{n} \left(y_i - f(\mathbf{w}, \mathbf{x}_i) - \frac{\sigma^2}{2} Tr(H_{w_t}) \right)^2 + \sum_{i=1}^{n} \frac{\sigma^4}{2} Tr(H_{\mathbf{w}_t}^2), \tag{6.16}$$

where $H_{\mathbf{w}_t}$ is the Hessian matrix of function f with respect to the weight vector $\mathbf{w}_t$ evaluated at the t^{th} iteration. Similarly, we impose a smoothness constraint on the second-order derivative (curvature) with respect to the weights of the function to be estimated. The smoothness of function f with respect to the weight vector $\mathbf{w}$ leads to the robustness of the network to small changes in weight values or damage to some connections.

What happens if we add noise to both the training patterns and the weights simultaneously? Let us assume a noise model $N(0, \sigma_1^2 I)$ for the training patterns and $N(0, \sigma_2^2 I)$ for the weights, with the two noise sources being independent. Then it can be shown (see Appendix B) that the net effect is to minimize the following regularized cost function for the first-order approximation.

$$E(\mathbf{w}, \mathcal{T}_n) = \sum_{i=1}^{n} (y_i - f(\mathbf{w}, \mathbf{x}_i))^2 + \sigma_1^2 \sum_{i=1}^{n} \|\partial_{\mathbf{x}} f(\mathbf{w}, \mathbf{x}_i)\|^2. \tag{6.17}$$

For the second-order approximation, the regularized cost function becomes

$$\begin{aligned} E(\mathbf{w}, \mathcal{T}_n) = & \sum_{i=1}^{n} \left(y_i - f(\mathbf{x}_i, \mathbf{w}) - \frac{\sigma_1^2}{2} Tr(H_{\mathbf{w}}) - \frac{\sigma_2^2}{2} Tr(H_{\mathbf{x}_i}) \right)^2 \\ & + \sum_{i=1}^{n} \left(\sigma_2^2 \|\partial_x f(\mathbf{w}, \mathbf{x}_i)\|^2 \right) \\ & + \sum_{i=1}^{n} \left(\frac{\sigma_1^4}{2} Tr(H_{\mathbf{w}}^2) + \frac{\sigma_2^4}{2} Tr(H_{\mathbf{x}_i}^2) + \sigma_1^2 \sigma_2^2 Tr(H_{\mathbf{wx}}^T H_{\mathbf{wx}}) \right). \end{aligned} \tag{6.18}$$

6.2.3 Type III: Explicitly-Specified Regularization

The most obvious way to reduce the solution space is to explicitly specify the regularization stabilizer Ωz based on our knowledge of the underlying function. Smoothness is a natural constraint which is frequently used. In this case, the stabilizer is a differential operator. We call this type of regularization as Type III (Explicitly-Specified) regularization.

We now briefly review the Type III regularization techniques in neural networks.

The simplest regularized cost function is the weight-decay regularization [64].

$$E(\mathbf{w}, \mathcal{T}_n) = \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i, \mathbf{w}))^2 + \lambda \sum_{i=1}^{p} w_i^2, \tag{6.19}$$

where the complexity of the network is defined as the sum of squared weights.

Rumelhart et al. [152], Weigend et al. [174], and Hanson and Pratt [56] used the following cost function:

$$E(\mathbf{w}, \mathcal{T}_n) = \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i, \mathbf{w}))^2 + \lambda \sum_{i=1}^{p} \frac{w_i^2/u^2}{1 + w_i^2/u^2}, \tag{6.20}$$

where u is a prespecified constant. The second term on the right hand side in Eq. (6.20) penalizes very large and very small weights ($|w_i| \gg u$). This regularization term may be interpreted as the "complexity" of the network, which is approximately proportional to the number of bits needed to encode the weights. The goal of the learning algorithm is to find a network that has the lowest complexity and at the same time fits the data adequately. Weigend et al. [174] developed a training algorithm to adaptively choose the value of λ.

Furthermore, Chauvin [25] studied the generalization behavior of the following

regularized cost function.

$$E(\mathbf{w}, \mathcal{T}_n) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i, \mathbf{w}))^2 + \lambda_1 \sum_{i=1}^{h} \frac{h_i^2}{1+h_i^2} + \lambda_2 \sum_{i=1}^{p} \frac{w_i^2}{1+w_i^2}, \tag{6.21}$$

where h_i is the output of the i^{th} hidden unit, $i = 1, 2, \cdots, h$, and λ_1, λ_2 are regularization parameters. The first regularization term penalizes large outputs of hidden nodes. The second term is the same as in Rumelhart's regularized cost function. These two regularized terms help to obtain a network with a small number of bits to encode the network weights and outputs of the hidden nodes. This regularization also leads to a network with fewer crucial weights and units, thus making the network more robust. Chauvin [25] showed that for the speech labeling task, the generalization performance was stable at about the 95% level, independent of the network size.

Sjoberg [162] used the following regularized cost function to model a hydraulic robot arm:

$$E(\mathbf{w}, \mathcal{T}_n) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i, \mathbf{w}))^2 + \lambda \|\mathbf{w} - \mathbf{w}^0\|^2, \tag{6.22}$$

where $\mathbf{w}^0$ is the initial weight vector. This regularization term prevents all the weights from deviating too much from the initial values. Using this method, a good initial guess of weight values is crucial. This regularized cost function is suitable in situations where good initial weight values can either be obtained through other methods, or from a previously trained network. So, the regularizer is used to refine the network.

We can also impose the smoothness constraints to the function implemented by the network. Following are a few examples of the smoothness constraint.

$$E(\mathbf{w}, \mathcal{T}_n) = \sum_{i=1}^{n}(y_i - f(\mathbf{x}_i, \mathbf{w}))^2 + \lambda \sum_{i=1}^{n} \|\partial_{\mathbf{x}} f(\mathbf{x}_i, \mathbf{w})\|^2, \tag{6.23}$$

$$E(\mathbf{w}, \mathcal{T}_n) = \sum_{i=1}^{n}(y_i - f(\mathbf{x}_i, \mathbf{w}))^2 + \lambda \sum_{i=1}^{n} \|\partial_{\mathbf{xx}} f(\mathbf{x}_i, \mathbf{w})\|^2, \tag{6.24}$$

$$E(\mathbf{w}, \mathcal{T}_n) = \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i, \mathbf{w}))^2 + \lambda \sum_{i=1}^{n} \|\partial_{\mathbf{ww}} f(\mathbf{x}_i, \mathbf{w})\|^2. \tag{6.25}$$

As we have shown in the previous section, adding the i.i.d. Gaussian noise with zero-mean and covariance matrix $\sigma^2 I$ to the training samples and weights has the similar effects as these explicitly specified regularizers.

Barron [11] proposed a complexity regularization criterion for model selection, and applied it to neural networks. The idea is to define the complexity of a function (or a network) in terms of the number of parameters in the function and the number of training patterns, and use this complexity measure to penalize selection of complicated functions. The motivation of the complexity regularization approach comes from the well-known theoretical results by Vapnik [170], Devroye [31] and Haussler [60], which show that the discrepancy between the empirical error and theoretical error is uniformly bounded by $O(\sqrt{C_n/n})$ in probability for families of functions whose complexity is bounded by C_n. As a result, minimizing the complexity-regularized cost function results in performance which is essentially as good as the one achievable by the theoretical analog of the criterion.

Consider a two-layer feedforward network with H nodes in the hidden layer. The complexity of a function $f_H(\cdot, \mathbf{w})$ implemented by this network is defined as

$$C_n(f_H(\cdot, \mathbf{w})) = \frac{p}{2} log_2 n + c_{\mathbf{w}} + c_H, \tag{6.26}$$

where n is the number of training patterns, and p is the total number of parameters in the network. The value of $\frac{p}{2} log_2 n + c_{\mathbf{w}}$ may be interpreted as a codelength for the parameter $\mathbf{w}$, and 2^{-c_w} as a prior density function for the parameters. Similarly, c_H can be interpreted as a codelength for the network, or 2^{-c_H} as a prior probability.

Given a collection $\mathcal{F}_n$ of functions (a set of networks), the method of complexity

regularization selects the function, $f \in \mathcal{F}_n$, to minimize

$$\frac{1}{n}\sum_{i=1}^{n} d(y_i, f(\mathbf{x}_i, \mathbf{w})) + \lambda \left(\frac{1}{n} C_n(f)\right)^{1/2}, \tag{6.27}$$

or

$$\frac{1}{n}\sum_{i=1}^{n} d(y_i, f(\mathbf{x}_i, \mathbf{w})) + \lambda \frac{1}{n} C_n(f), \tag{6.28}$$

where $d(\cdot, \cdot)$ is a distance metric, and λ is the regularization parameter. The first criterion (Eq. (6.27)) is used for the 0-1 distance metric and absolute distance metric for which the ideal rate of convergence of the risk would be close to $1/\sqrt{n}$. The second criterion (Eq. (6.28)) is used for the squared error and log-likelihood based distances for which the ideal rate would be close to $1/n$.

Neti et al. [130] have developed a learning algorithm for feedforward networks which can achieve uniform fault tolerance. The algorithm learns the weights in the network to perform the given computational task and has an additional property that whenever any single hidden unit is deleted from the network, the resulting reduced network continues to perform the computation satisfactorily. The uniformity of fault-tolerance is a measure of the extent to which the computation performed by the network is distributed throughout the hidden layer. Compared to other methods which also constrain the solution space by choosing a parsimonious network, such as constraints on the size of the hidden layer [24, 129], weight pruning [19], node pruning [158, 159], and optimal brain damage [107], the uniformly distributed computation is biologically more plausible.

Let $f^h(\mathbf{x}, \mathbf{w}^h)$ be a function implemented by the network whose hidden node, h, and all the connections incident to this node are removed, where $\mathbf{w}^h$ is the new weight

vector of the resulting network. Let

$$E_h(\mathbf{w}^h, \mathcal{T}_n) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f^h(\mathbf{x}_i, \mathbf{w}^h))^2, \tag{6.29}$$

and

$$g_h(\mathbf{w}, \mathcal{T}_n) = \frac{1}{n}\sum_{i=1}^{n}(f(\mathbf{x}_i, \mathbf{w}) - f^h(\mathbf{x}_i, \mathbf{w}^h))^2. \tag{6.30}$$

The network is then called *ε-fault-tolerant* with respect to the training set $\mathcal{T}_n$ if

$$g_h(\mathbf{w}, \mathcal{T}_n) \leq \varepsilon, \quad \text{for all } h \in S_h, \tag{6.31}$$

where S_h is the set of all hidden nodes. When $y_i = f(\mathbf{x}_i, \mathbf{w})$ for all $i = 1, 2, \cdots, n$, $g_h(\mathbf{w}, \mathcal{T}_n) = E_h(\mathbf{w}, \mathcal{T}_n)$.

Given an ε, the objective of creating an *ε-fault-tolerant* network is to determine a weight vector $\mathbf{w}^*$, such that

$$E(\mathbf{w}^*, \mathcal{T}_n) = \min_{\mathbf{w}} E(\mathbf{w}, \mathcal{T}_n), \tag{6.32}$$

subject to

$$E_h(\mathbf{w}^h, \mathcal{T}_n) - E(\mathbf{w}, \mathcal{T}_n) \leq \varepsilon, \quad \text{for all } h \in S_h. \tag{6.33}$$

Note that this is the second formulation of the regularization problem in Section 6.1.

A maximally fault-tolerant network can then be obtained by solving this regularization problem for a range of ε values, and choosing the result with the smallest ε whose corresponding $E(\mathbf{w}^*, \mathcal{T}_n)$ is near zero. Neti et al. [130] used a successive quadratic programming algorithm to solve this nonlinear optimization problem.

Experiments have shown that networks whose weights are learned using this algorithm achieve a harmony between generalization and fault tolerance. The computation is uniformly distributed among nodes, which leads to good fault tolerance.

Furthermore, due to a reduction of the solution space, good generalization is also achieved.

6.3 Role of Regularization in Neural Networks

In Chapter 5, we demonstrated that the weight-decay regularization can reduce the effective number of parameters in feedforward networks, thereby reducing the deviation of the expected MSE on the test set from the expected MSE on the training set. It is reasonable to expect that other regularization techniques will have a similar effect as the weight-decay regularization. We can give a general answer to the question of why some large networks with a relatively small number of training patterns can generalize well. In the absence of any constraints, the solution space is the entire space spanned by the free parameters. Introducing regularization in neural networks greatly reduces the solution space (the capacity or VC dimension of feedforward networks). These constraints imposed by the user or inherent in the network architecture and learning algorithms are usually designed to match the regularities of the underlying problem. It is this match that leads to good generalization and robustness. According to our classification of the regularization techniques in neural networks, application of neural networks to real-world problems such as pattern classification, prediction, and control, involves at least two types of regularization (Type I and Type II). When necessary, we can impose further constraints by using the Type III regularization. This analysis provides a qualitative explanation of why a large system trained with a relatively small number of patterns can generalize well.

6.4 Summary

The main contribution of this chapter is a systematic study of regularization techniques in neural network design. We have presented a taxonomy of regularization techniques in neural networks, and demonstrated that a variety of networks and learning algorithms can fit into this framework. Regularization plays a very important role in the performance of neural networks. It helps neural networks to achieve good generalization and robustness properties by introducing the *a priori* knowledge and constraints of the underlying problem and desirable properties of the solution into the optimization procedure. The reduction in the size of the solution space and in the effective number of free parameters resulting from the three types of regularization techniques is an important explanation of why many neural networks with a large number of free parameters trained with a relatively small number of training patterns perform well in several real applications.

CHAPTER 7

Improving Generalization Ability Through Node Pruning

We propose a node saliency measure which measures the importance of nodes in a feedforward network. A node is insalient if the removal of this node from the network causes the least increase in the value of the cost function. We provide a backpropagation type of algorithm to compute the node saliencies. A node-pruning procedure is then presented to remove insalient nodes in the network to create a small-sized network which can not only approximate faithfully the training set but also generalize well on the test patterns. Feature selection is performed simultaneously. The optimal/suboptimal subset of features are automatically selected by the network. The performance of the proposed approach for feature selection purpose is compared with the well-known Whitney's feature selection method. The small sample size effect on the computation of node/feature saliency is studied empirically. We conclude that the node-pruning procedure can improve the generalization ability of neural network classifiers even though the computation of node/feature saliency can not avoid the small sample size effect ("curse of dimensionality"). Application of the node-pruning procedure to feature selection in an OCR (optical character recognition) system is also discussed. A large number of redundant features can be removed in our OCR

systems using the proposed method.

The closest techniques to our work on node pruning are Le Cun's optimal brain damage (OBD) [107] and Mozer and Smolensky's skeletonization algorithm [129]. In the optimal brain damage approach, a saliency measure is defined for individual weights, and weights with low saliencies are removed from the network during the training. As a result, a sparse network is created. However, the number of nodes may remain approximately the same as the original network that the algorithm starts with because a node can be removed only if all of its incident connections or all of its outgoing connections have been removed. In contrast, the approach which we propose here can produce a network with the small number of nodes and can perform feature selection silmutaneously. In the skeletonization algorithm proposed by Mozer and Smolensky, a node relevancy measure is defined as the difference in errors when the node is removed and when it is left in place. The values of node relevancy are evaluated through the first-order approximation in the Taylor expansion with respect to a gate variable (for each node) which takes two values 0 (the node is removed) and 1 (the node is functioning). We have found that such a gate variable is not necessary and the approximation is poor when the high-order terms are not negligible. In our approach, the second-order information of the cost function is used. Similar to Le Cun's optimal brain damage approach, we approximate the Hessian matrix of the cost function by a diagonal matrix. Our approach can also work together with LeCun's OBD approach to prune both the nodes and the weights simultaneously or sequentially.

7.1 Node Saliency

Consider the feedforward network shown in Fig. 1.2. The number of input nodes, d, is set to the input feature vector dimensionality. The number of output nodes,

c, is specified as the dimensionality of the output space. For the classification purpose, c is often set to the number of categories. Let $\mathbf{x} = (x_1, x_2, \cdots, x_d)^T$ be a d-dimensional input vector. We denote the output of the j^{th} node in layer l by $y_j^{(l)}$, $j = 1, 2, \cdots, n_l$, $l = 0, 1, 2, \cdots, L$, where n_l is the number of nodes in layer l, L is the total number of layers, and $y_j^{(0)} = x_j$, $j = 1, 2, \cdots, d$. The weight on connection between node i in layer $l-1$ and node j in layer l is represented by $w_{ij}^{(l)}$. The sigmoid activation function $g(h)$ is used for each node, $g(h) = \frac{1}{1+\exp(-h)}$, where h is the net input to the node, which is the weighted sum of all inputs to the node. Therefore, the output of the j^{th} node in layer l can be written as

$$y_j^{(l)} = g\left(\sum_{i=1}^{n_l} w_{ij}^{(l)} y_i^{(l-1)}\right), \; l = 1, 2, \cdots, L. \tag{7.1}$$

The squared error function is most commonly used as a cost function in training the feedforward network.

$$E = \frac{1}{2}\sum_{k=1}^{N}\sum_{j=1}^{c}(y_j^{(L)}(k) - d_j(k))^2, \tag{7.2}$$

where $\mathbf{d}(k) = (d_1(k), d_2(k), \cdots, d_c(k))^T$ is the desired output for the k^{th} input pattern. The backpropagation learning algorithm [152] is commonly used to minimize this cost function. In the pattern-based learning mode, only the squared error for the current pattern is used to update the weights in the network.

$$E_k = \frac{1}{2}\sum_{j=1}^{c}(y_j^{(L)}(k) - d_j(k))^2, \; k = 1, 2, \cdots, N. \tag{7.3}$$

It is well-known that nodes and weights in the network trained using the backpropagation algorithm are not equally important to the cost function, especially when a network with a size larger than necessary is used.

We define the saliency of a node in the network as the amount of increase in the cost if this node is removed from the network.

We consider the cost E as a function of the outputs (which are collected in a vector $\mathbf{y}$) of all the nodes in the network. From the Taylor series expansion of the cost function with respect to the output of all the nodes, we have

$$\begin{aligned}\Delta E_k &= \Delta\mathbf{y}^T\frac{\partial E_k}{\partial \mathbf{y}} + \frac{1}{2}\Delta\mathbf{y}^T H \Delta\mathbf{y} + H.O.T. \\ &= \sum_{i=1}^{n}\frac{\partial E_k}{\partial y_i}\Delta y_i + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} h_{ij}\Delta y_i \Delta y_j + H.O.T., \end{aligned} \tag{7.4}$$

where $H = [h_{ij}]$ is the Hessian matrix of the cost E with respect to $\mathbf{y}$, and $h_{ij} = \frac{\partial^2 E_k}{\partial y_i \partial y_j}$.

If we neglect the high-order term in the Taylor expansion, and only the i^{th} component of the vector $\mathbf{y}$ changes at a time (we remove one node at a time), then Eq. (7.4) can be rewritten as

$$\Delta E_k = \frac{\partial E_k}{\partial y_i}\Delta y_i + \frac{1}{2}h_{ii}(\Delta y_i)^2. \tag{7.5}$$

Note that at this point we have not assumed that the Hessian matrix is a diagonal matrix. We can now interpret the value of ΔE_k as the amount of increase in error due to setting the output of the i^{th} node from y_i to zero. The saliency of the i^{th} node is then defined as

$$S_i = \sum_{k=1}^{N}\Delta E_k. \tag{7.6}$$

We still need to compute the first- and the second-order derivatives of the cost function with respect to the output of individual nodes. We have found that these derivatives can be computed in a backpropagation fashion.

For the output nodes,

$$\frac{\partial E_k}{\partial y_i^{(L)}} = y_i^{(L)} - d_i, \tag{7.7}$$

and

$$\frac{\partial^2 E_k}{\partial y_i^{(L)^2}} = 1.0. \tag{7.8}$$

For the nodes in the l^{th} layer,

$$\frac{\partial E_k}{\partial y_i^{(l)}} = \sum_{j=1}^{n_{l+1}} \frac{\partial E_k}{\partial y_j^{(l+1)}} g'(h_j) w_{ij}, \tag{7.9}$$

and

$$\frac{\partial^2 E_k}{\partial y_i^{(l)^2}} = \sum_{j=1}^{n_{l+1}} \frac{\partial}{\partial y_i^{(l)}} \left(\frac{\partial E_k}{\partial y_j^{(l+1)}} \right) g'(h_j) w_{ij} + \sum_{j=1}^{n_{l+1}} \frac{\partial E_k}{\partial y_j^{(l+1)}} g''(h_j) w_{ij}^2, \tag{7.10}$$

where $g'(h_j) = y_j^{(l+1)}(1 - y_j^{(l+1)})$, $g''(h_j) = y_j^{(l+1)}(1 - y_j^{(l+1)})(1 - 2y_j^{(l+1)})$, and

$$\frac{\partial}{\partial y_i^{(l)}} \left(\frac{\partial E_k}{\partial y_j^{(l+1)}} \right) = \sum_{q=1}^{n_{l+1}} \frac{\partial}{\partial y_q^{(l+1)}} \left(\frac{\partial E_k}{\partial y_j^{(l+1)}} \right) g'(h_q) w_{iq}. \tag{7.11}$$

If we assume that $\frac{\partial^2 E_k}{\partial y_q^{(l+1)} \partial y_j^{(l+1)}} = 0$ if $j \neq q$, then

$$\frac{\partial^2 E_k}{\partial y_i^{(l)^2}} = \sum_{j=1}^{n_{l+1}} \frac{\partial^2 E_k}{\partial y_j^{(l+1)^2}} (g'(h_j) w_{ij})^2 + \sum_{j=1}^{n_{l+1}} \frac{\partial E_k}{\partial y_j^{(l+1)}} g''(h_j) w_{ij}^2. \tag{7.12}$$

From the above equations, we can see that these first-order and second-order derivatives can be evaluated from the output layer back to the input layer for each pattern presented to the network.

7.2 Node-Pruning Procedure

Having defined the saliency measure for each node, we now propose the following node-pruning procedure to create a small network.

1. Choose an initial network architecture, which is often larger than necessary.

2. Train the network a number of epochs on the training data set to obtain a solution.

3. For each pattern in the training data set, evaluate the first- and the second-order derivatives in the backpropagation fashion.

4. Compute the saliency values for each input node and hidden node.

5. Remove the input node or hidden node with the lowest saliency value, together with all the incident and outgoing connections.

6. Retrain the new network on a small number of epochs.

7. Compute the squared errors and classification errors (if applicable) on both the training data and the test data.

8. Repeat steps 3-8 until the stopping criterion is satisfied.

We now define an optimal stopping criterion.

In the small sample size situations where the number of training patterns is not sufficiently larger than the number of weights in the network (an empirical guideline is that the number of training patterns should be at least 5-10 times larger than the number of weights in the network), the removal of a low saliency node will decrease the test set error rate (both the squared error and classification error) at the beginning, then level off, and eventually increase. The optimal network can be chosen at the point where the test set error rate starts to increase. This phenomenon is often referred to as the "curse of dimensionality", which provides us an optimal stopping criterion.

In the large sample size situations (the number of training patterns is sufficiently larger than the number of weights in the initial network architecture), both the training set error rate and the test set error rate may consistently increase as more and more nodes are removed. However, the errors increase very slowly at the beginning, then may increase suddenly if a critical node has been removed. The stopping point can be chosen just before a significant jump in errors occurs. If no significant jump

appears, then the stopping criterion may be chosen to achieve a good tradeoff between the test set error rate and the complexity of the network which determines the recognition time.

It would be better if we divide the available data into three subsets: training, test, and validation sets. We can use the validation set to determine the stopping point and use the test set to evaluate the performance of the resulting network. However, doing this will make the training set even smaller. In our experiments, we choose to use only the training set and test set.

7.3 Experimental Results

In this section, we demonstrate the behavior of our node/feature saliency measure and the node-pruning procedure using the following three data sets: (i) modified Trunk's data set, (ii) 80x data set, and (iii) NIST handprint character data set.

The modified Trunk's data set* is generated from two multivariate Gaussian distributions, $N(\boldsymbol{\mu}_1,\boldsymbol{I})$ and $N(\boldsymbol{\mu}_2,\boldsymbol{I})$, where $\boldsymbol{\mu}_1 = -\boldsymbol{\mu}_2 = (\mu_1, \mu_2, \cdots, \mu_d)^T$, and $\mu_i = 1/i$, $i = 1, 2, \cdots, d$. In this data set, the rank-order of the importance of individual features is the same as the feature index, provided the number of training samples is sufficiently large. Therefore, it can be used to verify the consistency of the saliency-based feature selection with the predetermined rank order. In our experiments, a total of twenty 50-dimensional training data sets are generated from the modified Trunk's model. Ten of these 20 data sets contain 50 patterns per data set, and the other ten sets have 500 patterns per data set. A test set containing 1000 patterns is also generated from the model. The same test data set is used for estimating classification

*Trunk [169] used $\mu_i = 1/\sqrt{i}$ is his example to show the existence of the "curse of dimensionality" in maximum likelihood classifiers. In his example, the Euclidean distance between the two class means is infinite as the number of features approaches infinity. Our modification makes the "peaking" phenomenon occur earlier than in the original Trunk's data as the dimensionality increases.

accuracies.

8OX data set consists of 45 8-dimensional patterns from 3 classes (the handprinted characters "8", "O" and "X") with 15 patterns per class [81]. This is a very sparse data set.

The NIST handprint digit data set contains a total of 140,374 digit characters. We partition the data into a training data set with 81,728 digits and a test set with 58,646 digits.

We used the same set of learning parameters in all our experiments. The learning rate and momentum value in the backpropagation algorithm is set to 0.35 for the initial training containing 1000 epochs, and to 0.1 for retraining (500 epochs) after each pruning.

7.3.1 Rank-Order of Features

In this subsection, we verify the consistency of our feature saliency with other measures, such as the distance between class means, and k-nearest neighbor classification accuracies of individual features. We also investigate the small sample size effect on the computation of node/feature saliencies.

An interesting property of the modified Trunk's data is that the separation of the two classes with respect to individual features decreases as the feature index increases. This is true for the model and data with a large number of patterns. In the finite (especially small) sample size situations, the feature rank-order may not be consistent with the feature index. To show this, we plot the value of the distance between the two class means for each of the 50 features in Figure 7.1. The distance values are computed from the model which generates data, and from the training data sets of different size (50 and 500 patterns), respectively. For each size, the data are generated ten times and the values of the distance between the two class means are averaged. We can see that the distance obtained from training sets is not a monotonically decreasing

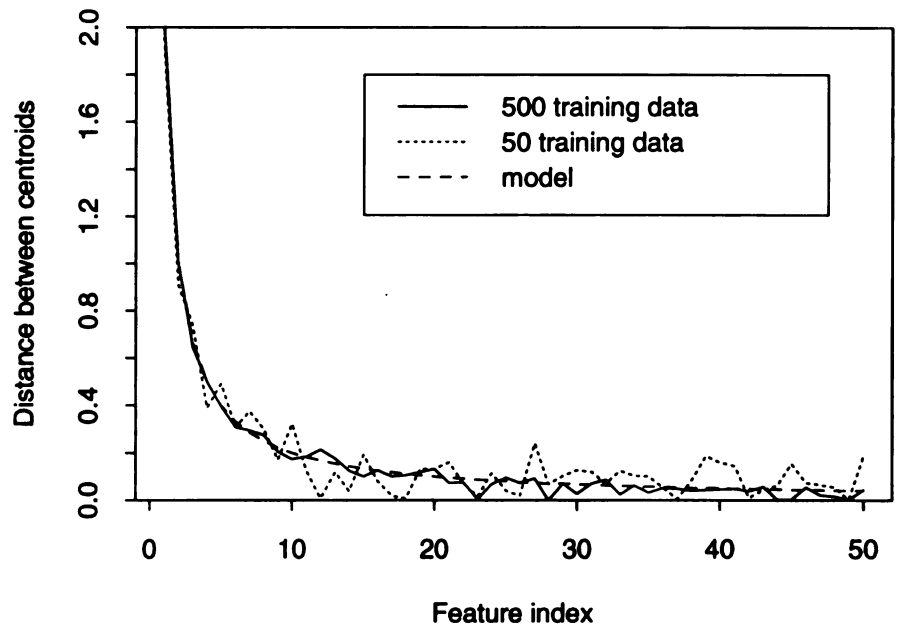


Figure 7.1. The distances between the two class means for individual features. The solid, dotted, and dashed curves plot the distance values obtained from the 500 training patterns, 50 training patterns, and the model which generates the data, respectively. The solid and dotted curves are smoothed by averaging over ten data sets of the corresponding sizes (500 and 50 patterns).

function of the feature index. The deviation from the model curve (dashed curve) decreases as the sample size increases. Moreover, the small sample size may create spurious correlation between individual features which can "fool" the neural network classifier.

Figure 7.2 shows the classification accuracies of the nearest neighbor classifier using individual features. Again, the values plotted in Figure 7.2 are averaged over 10 training data sets of specified size. Note that the overall nearest neighbor classification accuracies using 500 training patterns are consistently better than using 50 training patterns. Due to the finite sample size effect, the feature rank-order exhibits a certain fluctuation except for the first four features.

The average values of saliency for the 50 features are shown in Figure 7.3. The

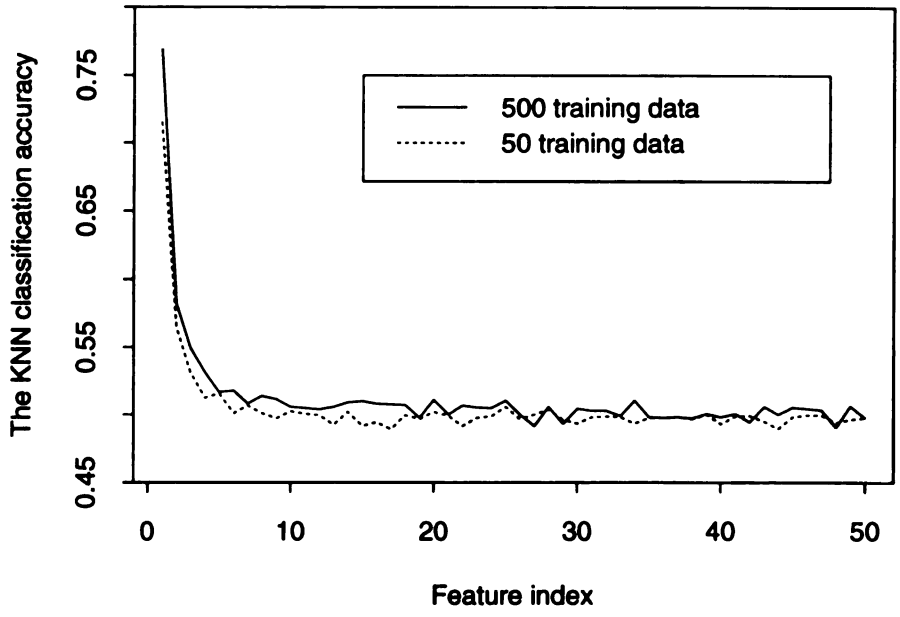


Figure 7.2. The nearest neighbor classification accuracies of individual features. The solid curve: using 500 training patterns. The dotted curve: using 50 training patterns. The values of accuracy are averaged over 10 training sets of the corresponding size. A test set containing 1000 patterns is used for estimating the classification accuracies.

averaging is taken over 100 trials (10 trials with different initial weights for each of the 10 training sets of a given size). The saliency values are normalized to a range of [0,1] by dividing them by the maximum saliency value among all the 50 features. We can see that the rank-order improves as the number of training patterns increases from 50 to 500. We should point out that although the saliency values are computed for individual features (or nodes), these features and nodes are competing/cooperating with each other to contribute to the target. Therefore, unlike the k-nearest neighbor classification accuracies of individual features, the saliencies of individual features from the network are not evaluated independently. The correlation between features and nodes will affect the saliencies of individual features. However, our approach can not detect the correlation between features or nodes. As a result, a high correlation between two features may reduce both the individual saliencies, but none (or both)

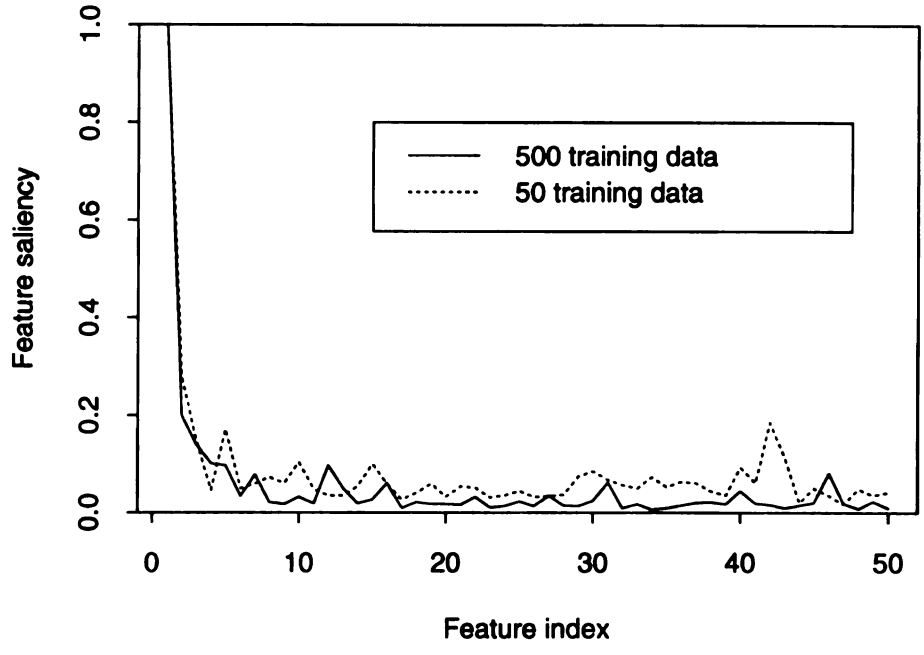


Figure 7.3. The averaged saliency measures for the 50 features. The averaging is taken over 100 trials (10 trials with different initial weights for each of the 10 training sets). Solid curve: data sets with 500 training patterns. Dotted curve: data sets with 50 training patterns.

of them would be removed by the node-pruning algorithm.

7.3.2 Feature Selection

Our node-pruning procedure is able to perform feature selection. We apply the node-pruning procedure to prune the trained networks (50 input nodes, 1 hidden node, and 2 output nodes) in Section 7.3.1. The same training data set as used in the initial training of the network is used for retraining after each pruning operation. Figure 7.4 shows the probabilities of individual features being included in the subset of 5 selected features over 100 trials using the node-pruning procedure. Both the curves oscillate. However, if we carefully examine the oscillating patterns of the dotted curve in Figure 7.1 and the dotted curve in Figure 7.4, we notice that they fit each other quite well. The oscillation is significantly reduced by increasing the number of training patterns.

For comparison, Figure 7.5 shows the probability (frequency) of individual features being included in the subset of 5 selected features using Whitney's method [181] over the 10 training data sets of each size. Whitney's method is a forward feature selection procedure which uses the k-nearest neighbor classifier (in our experiments, $k = 1$). The roughness of these curves can be improved by averaging over more training data sets. The first four features have a higher chance of being selected.

The average classification accuracies using the five features selected by the node-pruning procedure (classification using the pruned network) and Whitney's feature selection procedure (classification using the nearest neighbor classifier) are shown in Table 7.1. The classification accuracies are evaluated on the test data set with 1000 patterns. The pruned network achieves better classification accuracies than Whitney's approach. We should point out that this performance difference may be caused by the fact that two different classifiers are used in the two methods and that the nearest neighbor classifier is not an appropriate classifier for classifying Trunk's data and the modified Trunk's data.

Table 7.1. Classification accuracies using 5 features selected by the node-pruning procedure and Whitney's feature selection procedure.

	Node-Pruning	Whitney
50 training patterns	79.1%	76.6%
500 training patterns	86.3%	81.8%

Figure 7.6 shows the classification error rate versus the number of features being removed. Again, each curve is averaged over 100 trials. As we can see, the classification error rate slightly decreases as insalient features are removed, reaches a minimal point when a total of 39 features (when using 50 training patterns) and 46 features (when using 500 training patterns) have been removed, and then starts to increase.

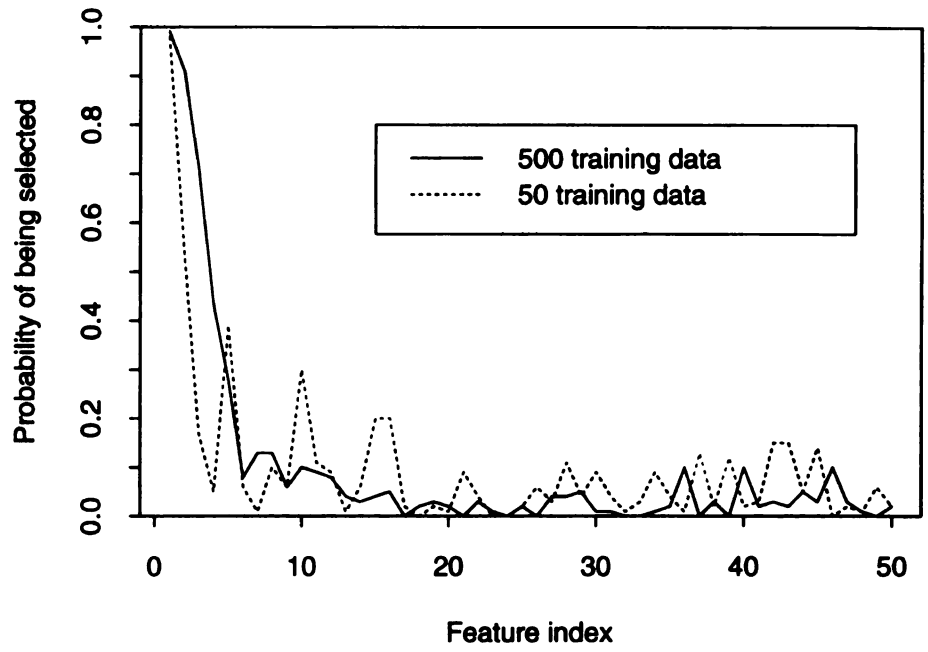


Figure 7.4. The probability of individual features belonging to the subset of 5 selected features using the node-pruning procedure on the modified Trunk's data. Solid curve: data sets with 500 training patterns. Dotted curve: data sets with 50 training patterns.

We also notice that a consistent lower classification error rate is achieved by using a larger number of training patterns. These results indicate that a large number of features can be deleted without a loss in the classification performance for the modified Trunk's data. Note that the smoothness of these curses can be improved by repeating more trials.

7.3.3 Creating a Small Network Through Node-Pruning

In many real classification problems, we often do not know what is the optimal size of the network to use. With the help of the proposed node-pruning procedure, we can start with a network which is larger than necessary, and then prune the network until an appropriate sized network is found.

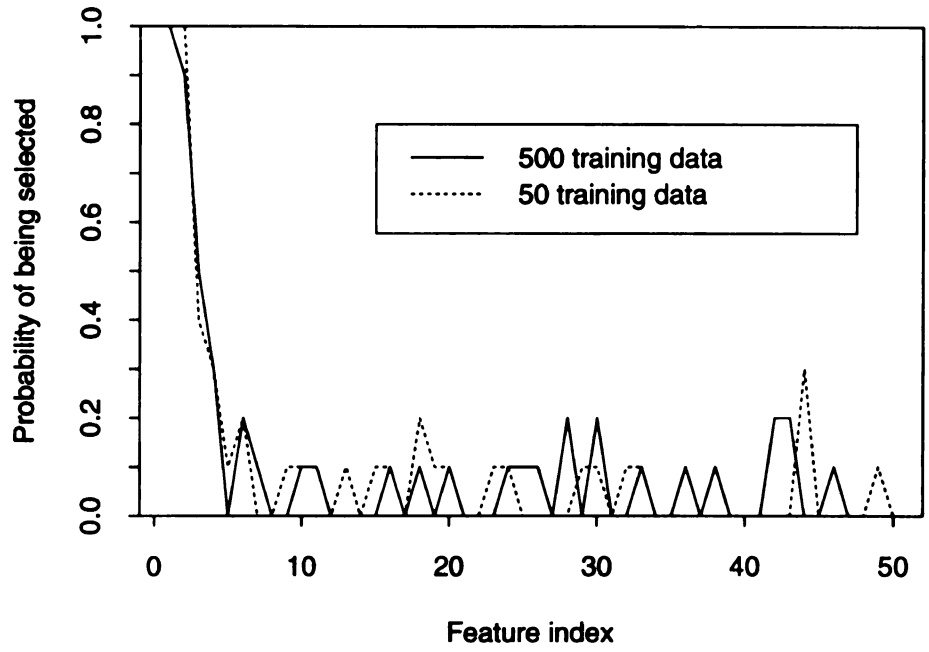


Figure 7.5. The probability of individual features belonging to the subset of 5 selected features using Whitney's feature selection procedure on the modified Trunk's data. Solid curve: data sets with 500 training patterns. Dotted curve: data sets with 50 training patterns. The curves are averaged over 10 training data sets of each size.

In this experiment, we want to design an small network for classifying the well-known 8OX data set. We start with a 2-layer network with 8 input nodes corresponding to 8 input features, 10 hidden nodes and 3 output nodes (corresponding to the three categories). We randomly partitioned the available data into two halves for training (23 patterns) and testing (22 patterns). We repeat this random partition ten times. For each partition, ten networks of the same size are trained with different initial weights. Figure 7.7 shows the classification error rates versus the number of nodes being removed. Again, the curve is averaged over 100 trials. The "optimal" network is found with 8 input nodes and 4 hidden nodes. Note that this "optimality" is in the statistical sense. For some trials, we were able to generate a network with lower test set error rate, but with 6 input nodes and 2 hidden nodes.

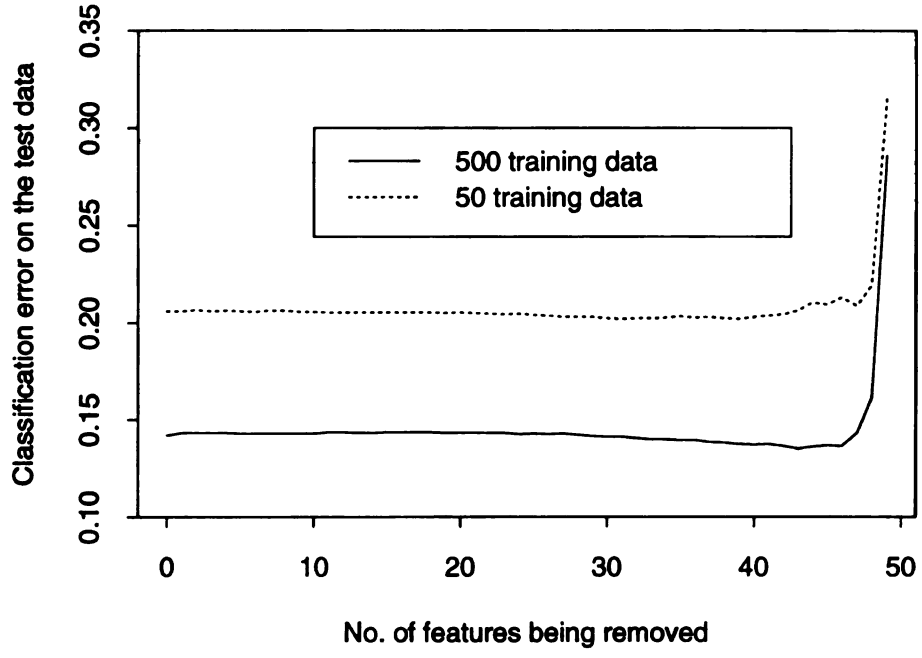


Figure 7.6. Classification error rates for the modified Trunk's data set versus the number of features being removed. The initial network contains a total of 50 input nodes (features). Solid curve: data sets with 500 training patterns. Dotted curve: data sets with 50 training patterns.

7.3.4 Application to an OCR system

A feedforward network with 184 input nodes, 40 hidden nodes, and 10 output nodes is designed to recognize handprinted digits. The network is fed with 184 features (88 local contour direction features and 96 features based on bending points) which are extracted from the 24×16 bitmap for each character. The training data contains 81,728 handprinted digits from the NIST digit database. The number of training patterns is substantially larger than the 7,810 total number of free parameters in the network. The trained network is tested on the test set (also from the NIST digit database) with 58,646 handprinted digits. A 96.4% recognition accuracy is achieved. Using our node-pruning procedure, 96 out of the 184 features can be removed from the original network, with only a slight loss of recognition accuracy (0.5%). The reduced network can not only save the recognition time, but also the feature extraction time

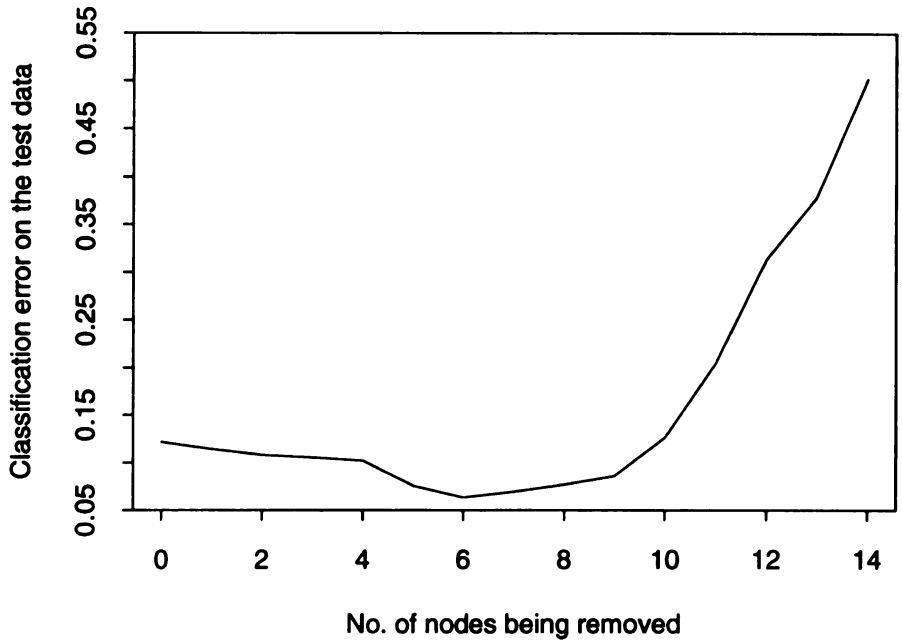


Figure 7.7. Classification error rates for the **80x** data set versus the number of nodes being removed. The initial network contains a total of 21 nodes.

because we do not need to extract those feature which were not selected by the node pruning procedure.

7.4 Summary

We have presented a node/feature saliency measure for feedforward networks. The node/feature saliency values can be evaluated using a backpropagation type of algorithm. A node-pruning procedure is also provided to remove insalient nodes in the network to create a small-sized network which can not only approximate faithfully the training set but also generalize well on the test patterns. The finite sample size effect on the computation of node/feature saliency is studied empirically. We have observed that the node-pruning procedure can still improve the generalization ability of

neural network classifiers if the sample size is sufficiently large even though the computation of node/feature saliency can not avoid the finite sample size effect ("curse of dimensionality"). The advantage of the node-pruning procedure over classical feature selection methods is that the node-pruning procedure can simultaneously "optimize" both the feature set and the classifier, while classical feature selection methods select the "best" subset of features with respect to a fixed classifier.

CHAPTER 8

Summary, Conclusions and Future Work

This dissertation can be logically divided into two parts. The first part (Chapters 2-4) focuses on designing neural networks for pattern recognition. The second part of this thesis (Chapters 5-7) is devoted to the theoretical analysis of generalization ability and practical techniques for improving the generalization ability of feedforward networks.

In Chapter 2, a number of neural networks and training algorithms for feature extraction and multivariate data projection have been proposed. These networks include: linear discriminant analysis (LDA) network, SAMANN network for Sammon's projection, nonlinear projection based on Kohonen's SOM (NP-SOM), and nonlinear discriminant analysis network (NDA) using a feedforward network classifier. This chapter also conducts a comparative study of five typical networks for feature extraction and multivariate data projection. Chapter 3 implements the well-known k-nearest neighbor classifier and its variants using neural network architecture. The k-Maximum network used in this k-nearest neighbor network architecture is novel, and has many advantages over the "winner-take-all" type of networks and other techniques for selecting k maximum input values. It has properties of adaptivity,

flexibility, and fewer connections, and it guarantees the selection of exactly k "on" nodes as long as its k^{th} and $(k+1)^{th}$ maximum inputs are distinct. Chapter 4 presents a self-organizing network (HEC) for detecting hyperellipsoidal clusters. The network performs a partitional clustering using the proposed regularized Mahalanobis distance. This distance measure can deal with the ill-posed and poorly-posed problems in estimating the Mahalanobis distance and achieves a tradeoff between using the Euclidean distance and the Mahalanobis distance so as to prevent the HEC network from producing unusually large or unusually small clusters.

The significance of the research work in the first part of this dissertation is as follows. i) A common attribute of these networks is that most of them employ adaptive training algorithms which make them suitable in environments where the distribution of patterns in the feature space changes with respect to time. ii) The availability of these networks also facilitates hardware implementation of well-known classical pattern classification and clustering approaches. iii) Some network implementations of classical pattern recognition approaches have additional advantages over the corresponding classical approaches. For example, the SAMANN network offers the generalization ability of projecting new data, which is not present in the original Sammon's projection algorithm; the HEC network employs the regularized Mahalanobis distance which achieves a tradeoff between using the Euclidean distance (which is likely to split large and elongated clusters) and the Mahalanobis distance (where large clusters tend to absorb small nearby clusters). iv) Some networks offer new approaches for solving pattern recognition problems. For instance, the NDA method and NP-SOM network provide a different perspective for visualizing high dimensional data.

Chapter 5 presents a theoretical study of the generalization ability of feedforward networks. We introduce two types of frameworks for this study: Vapnik's theory and Moody's approach, and point out the advantages and disadvantages of these two

frameworks. Vapnik's theory provides the distribution-free and training algorithm-free bounds for the maximum deviation of the true error from the empirical error, and for the training sample size requirement. Unfortunately, these bounds are too conservative to be of any practical use. On the other hand, Moody's approach establishes an approximate direct relationship (as oppose to the bounds in Vapnik's theory) between the expected MSEs on training and test data sets, and introduces the notion of the effective number of parameters. This direct relationship and the effective number of parameters are data-dependent and training algorithm-dependent. However, the rather restrictive assumptions on data model, training set and test set limit the practical applicability of Moody's approach.

We extend Moody's model to a more general setting by relaxing his assumptions on training and test data sets. This extension allows the sampling points of the test data to be different from those in the training data according to an additive noise model. We have shown that the noise in both sampling points in test data and observation increases the deviation of the expected test set MSE from the expected training set MSE, and also increases the effective number of parameters. Our Monte Carlo experiments have been conducted to verify Moody's result and our extension, and to demonstrate the role of the weight-decay regularization in improving the generalization ability of feedforward networks. Both Moody's result and our extension are consistent with the empirical results if the model assumptions are valid. However, we observed that Moody's result has a large discrepancy if the sampling points in the test data are subject to noise. Our Monte Carlo experiments have also shown that the estimation of the effective number of parameters is not reliable. Therefore, a robust estimator is desirable. One significant impact of these theoretical results on studying the generalization ability of feedforward networks is that they reveal how different quantities (e.g., the true error, empirical error, number of training patterns, expected

test and training set MSEs, nonlinearity, and regularization) which characterize feedforward networks interact with each other. Therefore, they shed some light on the behavior of the "black-box" as feedforward networks are often referred.

While the utility of these theoretical results in practice is still questionable, there are four different techniques in practice for improving the generalization ability of feedforward networks. Chapters 6 and 7 address two of these four techniques. Chapter 6 has provided a survey of regularization techniques used in neural networks. We have presented a taxonomy of regularization techniques in neural networks, and demonstrated that a variety of networks and learning algorithms can fit into this framework. Regularization plays a very important role in the performance of neural networks. It helps neural networks to achieve generalization and robustness properties by introducing the *a priori* knowledge and constraints of the underlying problem and desirable properties of the solution into the optimization procedure. The reduction in the size of the solution space and in the effective number of parameters resulting from the three types of regularization techniques is an important explanation of why many neural networks with a large number of free parameters trained with a relatively small number of training patterns perform well in several real applications.

We have proposed a node pruning method in Chapter 7 for designing parsimonious networks which generalize well, and for feature selection. We have presented a node/feature saliency measure for feedforward networks, which can be evaluated using a backpropagation type of algorithm. The insalient nodes are removed one at a time to create a small sized network which can not only approximate faithfully the training set but also generalize well on the test patterns. The finite sample size effect on the computation of node/feature saliency is studied empirically. We have found that the node-pruning procedure can still improve the generalization ability of neural network classifiers if the sample size is sufficiently large even though the computation of node/feature saliency suffers from the finite sample size effect ("curse

of dimensionality").

As we have pointed out, a large number of classical pattern recognition approaches have been mapped onto neural network architectures. Can the hardware implementation of these approaches derive any benefit from the mapping? Our preliminary and unpublished study (with Professor M. Shanblatt) on the VLSI implementation of the LDA network has shown that a chip with an area of $1cm \times 1cm$ can contain a LDA network with 20 inputs and 20 outputs, and the convergence speed for learning one pattern is of the order of a nano second. Since our experiments have shown that the number of iterations needed for convergence is of the order of 10^3, the LDA network can learn and perform discriminant analysis in about a microsecond. Although hardware implementation of neural networks is beyond the scope of this dissertation, it should be an important future research direction.

The practical value of the results on the generalization issue of feedforward networks needs to be verified through applications. It is generally believed that feature extraction and feature selection can help to avoid the "curse of dimensionality" and improve the generalization ability of a classifier. However, in finite sample size situations, since the transform from the original feature space to the new feature space with a lower dimensionality is often estimated from the finite number of training patterns, the feature extraction and selection itself suffers from the finite sample-size effect. Therefore, it would be of significant value if the theoretical results on generalization could be used to analyze the finite sample-size effect on various feature extraction and feature selection methods, such as principal component analysis and the node-pruning method proposed in Chapter 7.

In Chapter 5, we have only studied the role of the weight-decay regularization in reducing the effective number of parameters. A comparative study on the effect of different regularization cost functions on the effective number of parameters should also be conducted. This comparative study would be helpful in choosing a proper

regularizer.

APPENDICES

APPENDIX A

Derivation of the Effective Number of Parameters

Consider a set of n real-valued training pattern pairs $\mathcal{T}_n = \{(\mathbf{x}_i, y_i) | i = 1, 2, \cdots, n\}$ drawn from a stationary distribution $\Xi(\mathbf{x}, y)$, where $\mathbf{x}_i \in R^d$, $i = 1, 2, \cdots, n$. Without a loss of generality, assume that $y_i \in R$. These patterns can be viewed as being generated according to the additive model:

$$y = \mu(\mathbf{x}) + \varepsilon_y, \tag{A.1}$$

where ε_y is the *i.i.d.* noise with zero mean and variance σ_y^2 which is sampled with distribution $\Phi(\varepsilon)$ (not necessarily Gaussian), and $\mu(\mathbf{x})$ is the conditional mean, an unknown function. Let $\mathcal{T}_n' = \{(\mathbf{x}_i', y_i') | i = 1, 2, \cdots, n\}$ be a test set drawn from the same distribution as $\mathcal{T}_n$. We further assume that $\mathbf{x}_i'$ is subject to another additive noise model $\mathbf{x}_i' = \mathbf{x}_i + \varepsilon_{x_i}$. Note that in Moody's original model, $\mathbf{x}_i'$ in the test set should be exactly the same as $\mathbf{x}_i$ in the training set. Although Moody's detailed derivation has not yet been published, it should be a special case of our derivation.

In the following derivation, variables y, w and x with no subscripts, and ε_y are pooled vectors which collect all the corresponding scaler variables appearing in the

cost function. We do not make an explicit notational distinction between scalars, vectors, and 2-d, 3-d matrices. However, readers should be able to easily determine if the variables y, w and x are treated as vectors.

First, we define the training set MSE, expected training set MSE, test set MSE, and expected test set MSE, respectively, as follows.

$$E_{train}(\mathcal{T}_n) = E(y, f(w(\mathcal{T}_n), \mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(w(\mathcal{T}_n), \mathbf{x}_i))^2. \tag{A.2}$$

$$\begin{aligned} \langle E_{train}\rangle_{\mathcal{T}_n} &= \langle \frac{1}{n}\sum_{i=1}^{n}(y_i - f(w(\mathcal{T}_n), \mathbf{x}_i))^2\rangle_{\mathcal{T}_n} \\ &= \int \frac{1}{n}\sum_{i=1}^{n}(y_i - f(w(\mathcal{T}_n), \mathbf{x}_i))^2 \left\{\prod_{i=1}^{n}\Phi(y_i|\mathbf{x}_i)dy_i\right\}. \end{aligned} \tag{A.3}$$

$$E_{test}(\mathcal{T}_n') = E(y', f(w(\mathcal{T}_n), \mathbf{x}') = \frac{1}{n}\sum_{i=1}^{n}(y_i' - f(w(\mathcal{T}_n), \mathbf{x}_i')). \tag{A.4}$$

$$\begin{aligned} \langle E_{test}\rangle_{\mathcal{T}_n\mathcal{T}_n'} &= \langle \frac{1}{n}\sum_{i=1}^{n}(y_i' - f(w(\mathcal{T}_n), \mathbf{x}_i'))^2\rangle_{\mathcal{T}_n\mathcal{T}_n'} \\ &= \int \frac{1}{n}\sum_{i=1}^{n}(y_i' - f(w(\mathcal{T}_n), \mathbf{x}_i'))^2 \left\{\prod_{i=1}^{n}\Phi(y_i|\mathbf{x}_i)\Phi(y_i'|\mathbf{x}_i')dy_i dy_i'\right\}. \end{aligned} \tag{A.5}$$

We treat the noise terms ε_i in both the training and test data sets as small perturbations to an ideal model which fits the corresponding noise-free data. The mathematical expectation is taken over these noise terms. The perturbed cost function can be expanded upto the second-order in the noise terms. Note that the test set MSE has three perturbation terms associated with y', y and x', respectively. First let us

expand the test set MSE with respect to x':*

$$[E(y', f(w(\mathcal{T}_n), x')]_{y'yx'} \approx [E(y', f(w(\mathcal{T}_n), x)]_{y'y} + \varepsilon_x^T \left[\frac{\partial E}{\partial x}\right]_{y'y} + \frac{1}{2}\varepsilon_x^T \left[\frac{\partial^2 E}{\partial^2 x}\right]_{y'y} \varepsilon_x. \tag{A.6}$$

Since the second term on the right-hand side of Eq. (A.6) is already of the second-order in ε_x, we can approximate $\left[\frac{\partial E}{\partial x}\right]_{y'y}$ by its zeroth order term, i.e.,

$$\left[\frac{\partial E}{\partial x}\right]_{y'y} \approx \frac{\partial E}{\partial x}. \tag{A.7}$$

Now, we take a mathematical expectation over the noise associated with x', and apply the property of zero-mean noise, which leads to

$$\langle [E(y', f(w(\mathcal{T}_n), x')]_{y'yx'} \rangle_{x'} \approx [E(y', f(w(\mathcal{T}_n), x)]_{y'y} + \frac{1}{2}\text{trace}\left[\frac{\partial^2 E}{\partial x^2}\Delta_x\right], \tag{A.8}$$

where Δ_x is a diagonal covariance matrix of the long noise vector $\varepsilon_x = [\varepsilon_{x_1}^T, \varepsilon_{x_2}^T, \cdots, \varepsilon_{x_n}^T]^T$. The second term on the right-hand side of Eq. (A.8) can be rewritten into

$$\text{trace}\left[\frac{\partial^2 E}{\partial x^2}\Delta_x\right] = \sum_{i=1}^{n} \text{trace}\left[\frac{\partial^2 E}{\partial x_i^2}\Lambda_x\right] = \sigma_x^2 \text{trace}\left[\sum_{i=1}^{n} \frac{\partial^2 E}{\partial x_i^2}\right], \tag{A.9}$$

where $\Lambda_x = \sigma_x^2 I$ is the diagonal covariance matrix of ε_{x_i}.

Similarly, we can expand the first term on the right-hand side of Eq. (A.8).

$$[E(y', f(w(\mathcal{T}_n), x)]_{y'y} \approx [E(y, f(w(\mathcal{T}_n), x)]_y + (\varepsilon_y' - \varepsilon_y)^T \left[\frac{\partial E}{\partial y}\right]_y + \frac{1}{2}(\varepsilon_y' - \varepsilon_y)^T \left[\frac{\partial^2 E}{\partial^2 y}\right]_y (\varepsilon_y' - \varepsilon_y). \tag{A.10}$$

*The subscripts of [ ] indicate those variables which are involved in the enclosed term and are subject to noise perturbation.

In Eq. (A.10), we can approximate $\left[\frac{\partial^2 E}{\partial^2 y}\right]_y$ by the zeroth order term,

$$\left[\frac{\partial^2 E}{\partial^2 y}\right]_y \approx \frac{\partial^2 E}{\partial^2 y}, \tag{A.11}$$

because the third term on the right-hand side of Eq. (A.10) is already of the second-order in ε_y.

$$\left[\frac{\partial E}{\partial y}\right]_y \approx \frac{\partial E}{\partial y} + \frac{\partial^2 E}{\partial w \partial y}\frac{\partial w}{\partial y}\varepsilon_y + \frac{\partial^2 E}{\partial^2 y}\varepsilon_y + \frac{1}{2}\varepsilon_y^T \frac{\partial^3 E}{\partial^3 y}\varepsilon_y. \tag{A.12}$$

For the squared-error cost function, $\frac{\partial^3 E}{\partial^3 y} = 0$ (note that $\frac{\partial^3 E}{\partial^3 y}$ and 0 are three-dimensional matrices). At the equilibrium point, we have

$$\frac{\partial E_\lambda}{\partial w} = 0. \tag{A.13}$$

Remember that E_λ is a function of w which is again a function of y. Taking partial derivatives on both the sides of Eq. (A.13) with respect to y, we have

$$\frac{\partial^2 E_\lambda}{\partial y \partial w} + \frac{\partial^2 E_\lambda}{\partial^2 w}\frac{\partial w}{\partial y} = 0, \tag{A.14}$$

from which we obtain

$$\frac{\partial w}{\partial y} = -\left(\frac{\partial^2 E_\lambda}{\partial^2 w}\right)^{-1} \frac{\partial^2 E_\lambda}{\partial y \partial w}. \tag{A.15}$$

Note that $\frac{\partial w}{\partial y}$ is a two-dimensional matrix. Substituting Eq. (A.15) into Eq. (A.12), we obtain

$$\left[\frac{\partial E}{\partial y}\right]_y \approx \frac{\partial E}{\partial y} + \frac{\partial^2 E}{\partial w \partial y}\left(\frac{\partial^2 E_\lambda}{\partial^2 w}\right)^{-1} \frac{\partial^2 E_\lambda}{\partial y \partial w}\varepsilon_y + \frac{\partial^2 E}{\partial^2 y}\varepsilon_y. \tag{A.16}$$

Substituting Eqs. (A.11) and (A.16) into Eq. (A.10), we have

$$
\begin{aligned}
[E(y', f(w(\mathcal{T}_n), \mathbf{x})]_{y'y} &\approx [E(y, f(w(\mathcal{T}_n), \mathbf{x})]_y + (\varepsilon'_y - \varepsilon_y)^T \frac{\partial E}{\partial y} \\
&- (\varepsilon'_y - \varepsilon_y)^T \frac{\partial^2 E}{\partial w \partial y} \left(\frac{\partial^2 E_\lambda}{\partial^2 w} \right)^{-1} \frac{\partial^2 E_\lambda}{\partial y \partial w} \varepsilon_y \\
&+ (\varepsilon'_y - \varepsilon_y)^T \frac{\partial^2 E}{\partial^2 y} \varepsilon_y + \frac{1}{2} (\varepsilon'_y - \varepsilon_y)^T \frac{\partial^2 E}{\partial^2 y} (\varepsilon'_y - \varepsilon_y). \quad \text{(A.17)}
\end{aligned}
$$

Now taking expectation on both the sides of Eq. (A.17) over the noise variables, and applying the Lemma in Appendix B, we obtain

$$
\langle E(y', f(w(\mathcal{T}_n), \mathbf{x}) \rangle_{y'y} \approx \langle E(y, f(w(\mathcal{T}_n), \mathbf{x}) \rangle_y + \sigma_y^2 \text{trace} \left[\frac{\partial^2 E}{\partial w \partial y} \left(\frac{\partial^2 E_\lambda}{\partial^2 w} \right)^{-1} \frac{\partial^2 E_\lambda}{\partial y \partial w} \right]. \quad \text{(A.18)}
$$

Taking mathematical expectation on both the sides of Eq. (A.8) over y' and y, and substituting Eq. (A.18) into it, we get

$$
\begin{aligned}
\langle E(y', f(w(\mathcal{T}_n), \mathbf{x}) \rangle_{x'y'y} &\approx \langle E(y, f(w(\mathcal{T}_n), \mathbf{x}) \rangle_y \\
&+ \sigma_y^2 \text{trace} \left[\frac{\partial^2 E}{\partial w \partial y} \left(\frac{\partial^2 E_\lambda}{\partial^2 w} \right)^{-1} \frac{\partial^2 E_\lambda}{\partial y \partial w} \right] \quad \text{(A.19)} \\
&+ \frac{1}{2} \sigma_x^2 \text{trace} \left[\sum_{i=1}^{n} \frac{\partial^2 E}{\partial x_i^2} \right].
\end{aligned}
$$

Let

$$
T = \frac{\partial^2}{\partial y \partial w} (nE(w, \mathcal{T}_n)), \quad \text{(A.20)}
$$

$$
U = \frac{\partial^2}{\partial^2 w} E_\lambda(w, \mathcal{T}_n), \quad \text{(A.21)}
$$

$$
V = \frac{\partial^2}{\partial y \partial w} E_\lambda(w, \mathcal{T}_n), \quad \text{(A.22)}
$$

$$
G = \sum_{i=1}^{n} \frac{\partial^2 nE}{\partial x_i^2}, \quad \text{(A.23)}
$$

$$
p'_{eff_y}(\lambda) = \frac{1}{2} \text{trace}[TU^{-1}V^T], \quad \text{(A.24)}
$$

and

$$p'_{eff_x}(\lambda) = \frac{1}{4}\text{trace}[G]. \tag{A.25}$$

We finally obtain

$$\langle E_{test}\rangle_{\mathcal{T}_n\mathcal{T}'_n} \approx \langle E_{training}\rangle_{\mathcal{T}_n} + 2\sigma_y^2\frac{p_{eff_y}(\lambda)}{n} + 2\sigma_x^2\frac{p_{eff_x}(\lambda)}{n}. \tag{A.26}$$

APPENDIX B

Regularization via Adding Noise to Weights and Training Patterns

The following lemma will be used in establishing the fact that adding noise to weights and training patterns results in what we called Type II regularization of network solution in Chapter 6. We state the lemma without proof.

Lemma: *Let* $\mathbf{r}, \mathbf{r}_1, \mathbf{r}_2$ *be* p-, p_1-, p_2*-dimensional random vectors drawn from iid Gaussian noise distributions with zero mean vector and covariance matrices* $\sigma^2 I$, $\sigma_1^2 I_1$, *and* $\sigma_2^2 I_2$, *respectively. Let* H *be a* $p \times p$ *symmetric matrix and* H_1 *be a* $p_1 \times p_2$ *matrix. Then the following properties hold:*

(i) $\langle H\mathbf{r} \rangle_{\mathbf{r}} = \mathbf{0}$;

(ii) $\langle \mathbf{r}^T H \mathbf{r} \rangle_{\mathbf{r}} = \sigma^2 Tr(H)$;

(iii) $\langle H_1 \mathbf{r}\mathbf{r}^T H_2 \mathbf{r} \rangle_{\mathbf{r}} = \mathbf{0}$;

(iv) $\langle \mathbf{r}^T H^T \mathbf{r}\mathbf{r}^T H \mathbf{r} \rangle_{\mathbf{r}} = \sigma^4 Tr^2(H) + 2\sigma^4 Tr(H^2)$;

(v). $\langle \mathbf{r}_1^T H_1 \mathbf{r}_2 \mathbf{r}_2^T H_1^T \mathbf{r}_1 \rangle_{\mathbf{r}_1, \mathbf{r}_2} = \sigma_1^2 \sigma_2^2 Tr(H_1 H_1^T)$,

where $\langle \cdot \rangle_{\mathbf{r}}$ *denotes the expectation with respect to* $\mathbf{r}$, *and* $Tr(H)$ *is the trace of matrix* H.

B.1 Adding Noise to Weights Only

Let $f(\mathbf{w}, \mathbf{x}) : R^p \times R^d \rightarrow R$ be a function implemented by a feedforward network with a p-dimensional weight vector $\mathbf{w}$ and d input nodes. Adding noise to weights is equivalent to minimizing the following cost function.

$$E = \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} [y_i - f(\mathbf{w} + \mathbf{r}_{ij}, \mathbf{x}_i)]^2. \tag{B.1}$$

Suppose m is large enough, then the above equation can be approximated by

$$E = \frac{1}{n} \sum_{i=1}^{n} \langle [y_i - f(\mathbf{w} + \mathbf{r}, \mathbf{x}_i)]^2 \rangle_{\mathbf{r}}. \tag{B.2}$$

To simplify the notation, let

$$E_{\mathbf{x}_i} = \langle [y_i - f(\mathbf{w} + \mathbf{r}, \mathbf{x}_i)]^2 \rangle_{\mathbf{r}}. \tag{B.3}$$

For small values of σ, using the second-order approximation of Taylor series expansion, we obtain

$$f(\mathbf{w} + \mathbf{r}, \mathbf{x}) \approx f(\mathbf{w}, \mathbf{x}) + (\partial_{\mathbf{w}} f)^T \mathbf{r} + \frac{1}{2} \mathbf{r}^T H_{\mathbf{w}} \mathbf{r}, \tag{B.4}$$

where $H_{\mathbf{w}}$ is the Hessian matrix of function f with respect to the weight vector $\mathbf{w}$. Substituting Eq. (B.4) into Eq. (B.3) and applying the lemma to simplify the right-hand side, we get

$$\begin{aligned}
E_{\mathbf{x}_i} &= \langle [y_i - f(\mathbf{w}, \mathbf{x}_i) - (\partial_{\mathbf{w}} f)^T \mathbf{r} - \frac{1}{2} \mathbf{r}^T H_{\mathbf{w}} \mathbf{r}]^2 \rangle_{\mathbf{r}} \\
&= [y_i - f(\mathbf{w}, \mathbf{x}_i)]^2 + \sigma^2 \|\partial_{\mathbf{w}} f\|^2 - \langle [y_i - f(\mathbf{w}, \mathbf{x}_i) - (\partial_{\mathbf{w}} f)^T \mathbf{r}] \mathbf{r}^T H_{\mathbf{w}} \mathbf{r} \rangle_{\mathbf{r}} \\
&\quad + \langle \frac{1}{4} \mathbf{r}^T H_{\mathbf{w}}^T \mathbf{r} \mathbf{r}^T H_{\mathbf{w}} \mathbf{r} \rangle_{\mathbf{r}} \\
&= [y_i - f(\mathbf{w}, \mathbf{x}_i)]^2 + \sigma^2 \|\partial_{\mathbf{w}} f\|^2 - [y_i - f(\mathbf{w}, \mathbf{x}_i)] \langle \mathbf{r}^T H_{\mathbf{w}} \mathbf{r} \rangle_{\mathbf{r}}
\end{aligned}$$

$$
\begin{aligned}
&+\langle(\partial_{\mathbf{w}} f)^T \mathbf{r}\mathbf{r}^T H_{\mathbf{w}} \mathbf{r}\rangle_{\mathbf{r}} + \langle \frac{1}{4}\mathbf{r}^T H_{\mathbf{w}}^T \mathbf{r}\mathbf{r}^T H_{\mathbf{w}} \mathbf{r}\rangle_{\mathbf{r}} \\
= \;& [y_i - f(\mathbf{w}, \mathbf{x}_i)]^2 - [y_i - f(\mathbf{w}, \mathbf{x}_i)]\sigma^2 Tr(H_{\mathbf{w}}) + \frac{\sigma^4}{4} Tr^2(H_{\mathbf{w}}) \\
&+\frac{\sigma^4}{2} Tr(H_{\mathbf{w}}^2) \\
= \;& [y_i - f(\mathbf{w}, \mathbf{x}_i) - \frac{\sigma^2}{2} Tr(H_{\mathbf{w}})]^2 + \frac{\sigma^4}{2} Tr(H_{\mathbf{w}}^2). \qquad \text{(B.5)}
\end{aligned}
$$

Note that at the minimal point, $\partial_{\mathbf{w}} f = 0$.

The equivalence of adding noise to training patterns and regularization can be shown in a similar way by exchanging $\mathbf{w}$ and $\mathbf{x}$. See also [145].

B.2 Adding Noise to Weights and Training Patterns Simultaneously

Adding a noise vector $\mathbf{r}_1$ to weights and $\mathbf{r}_2$ to training patterns is equivalent to minimizing the following cost function.

$$E = \frac{1}{n}\sum_{i=1}^{n} \langle [y_i - f(\mathbf{w} + \mathbf{r}_1, \mathbf{x}_i + \mathbf{r}_2)]^2 \rangle_{\mathbf{r}_1 \mathbf{r}_2}. \qquad \text{(B.6)}$$

To simplify the notation, let

$$E_{\mathbf{x}_i} = \langle [y_i - f(\mathbf{w} + \mathbf{r}_1, \mathbf{x}_i + \mathbf{r}_2)]^2 \rangle_{\mathbf{r}_1 \mathbf{r}_2}. \qquad \text{(B.7)}$$

Assume that σ_1 and σ_2 are small. Then using the first-order approximation of Taylor series expansion, we have

$$f(\mathbf{w} + \mathbf{r}_1, \mathbf{x} + \mathbf{r}_2) \approx f(\mathbf{w}, \mathbf{x}) + (\partial_{\mathbf{x}} f)^T \mathbf{r}_2. \qquad \text{(B.8)}$$

Substituting Eq. (B.8) into Eq. (B.7), expanding the square term, and then using the lemma, we obtain

$$\begin{aligned} E_{\mathbf{x}_i} &\approx \langle [y_i - f(\mathbf{w}, \mathbf{x}_i) - (\partial_{\mathbf{x}} f)^T \mathbf{r}_2]^2 \rangle_{\mathbf{r}_1 \mathbf{r}_2} \\ &= [y_i - f(\mathbf{w}, \mathbf{x}_i)]^2 + \sigma_2^2 \|\partial_{\mathbf{x}} f\|^2. \end{aligned} \tag{B.9}$$

For higher precision, we can use the second-order approximation of Taylor series expansion,

$$f(\mathbf{w} + \mathbf{r}_1, \mathbf{x} + \mathbf{r}_2) \approx f(\mathbf{w}, \mathbf{x}) + (\partial_{\mathbf{x}} f)^T \mathbf{r}_2 + \frac{1}{2}[\mathbf{r}_1^T H_{\mathbf{w}} \mathbf{r}_1 + \mathbf{r}_2^T H_{\mathbf{x}} \mathbf{r}_2 + 2\mathbf{r}_1^T H_{\mathbf{wx}} \mathbf{r}_2], \tag{B.10}$$

where $H_{\mathbf{wx}}$ is the Hessian matrix of function f with respect to weight vector $\mathbf{w}$ and input vector $\mathbf{x}$. Therefore, it is a $p \times d$ matrix. Substituting Eq. (B.10) into Eq. (B.7), expanding the square term, and then applying the lemma, we obtain

$$\begin{aligned} E_{\mathbf{x}_i} &\approx \langle [y_i - f(\mathbf{w}, \mathbf{x}_i) - (\partial_{\mathbf{x}} f)^T \mathbf{r}_2 - \frac{1}{2}\mathbf{r}_1^T H_{\mathbf{w}} \mathbf{r}_1 - \frac{1}{2}\mathbf{r}_2^T H_{\mathbf{x}} \mathbf{r}_2 - \mathbf{r}_1^T H_{\mathbf{wx}} \mathbf{r}_2]^2 \rangle_{\mathbf{r}_1 \mathbf{r}_2} \\ &= [y_i - f(\mathbf{w}, \mathbf{x}_i)]^2 - [y_i - f(\mathbf{w}, \mathbf{x}_i)][\sigma_1^2 Tr(H_w) + \sigma_2^2 Tr(H_x)] \\ &\quad + \sigma_1^2 \|\partial_{\mathbf{w}} f\|^2 + \sigma_2^2 \|\partial_{\mathbf{x}} f\|^2 + \frac{\sigma_1^4}{4} Tr^2(H_{\mathbf{w}}) + \frac{\sigma_1^4}{2} Tr(H_{\mathbf{w}}^2) + \frac{\sigma_2^4}{4} Tr^2(H_{\mathbf{x}}) + \frac{\sigma_2^4}{2} Tr(H_{\mathbf{x}}^2) \\ &\quad + \frac{\sigma_1^2 \sigma_2^2}{4} Tr(H_{\mathbf{w}}) Tr(H_{\mathbf{x}}) + \sigma_1^2 \sigma_2^2 Tr(H_{\mathbf{wx}}^2) \\ &= [y_i - f(\mathbf{w}, \mathbf{x}_i) - \frac{\sigma_1^2}{2} Tr(H_{\mathbf{w}}) - \frac{\sigma_2^2}{2} Tr(H_{\mathbf{x}})]^2 + \sigma_2^2 \|\partial_{\mathbf{x}} f\|^2 \\ &\quad + \frac{\sigma_1^4}{2} Tr(H_{\mathbf{w}}^2) + \frac{\sigma_2^4}{2} Tr(H_{\mathbf{x}}^2) + \sigma_1^2 \sigma_2^2 Tr(H_{\mathbf{wx}}^T H_{\mathbf{wx}}). \end{aligned} \tag{B.11}$$

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] H. M. Abbas and M. M. Fahmy. A neural model for adaptive Karhunen-Loeve Transform (KLT). In *Proc. of IEEE Intl. Joint Conf. on Neural Networks*, volume 2, pages 975–980, Baltimore, Maryland, June 1992.

[2] H. Akaike. Statistical predictor identification. *Ann. Inst. Stat. Math.*, 22:203, 1970.

[3] James A. Anderson and Edward Rosenfeld. *Neurocomputing: Foundations of Research.* MIT Press, Cambridge, Massachusetts, 1988.

[4] L. Atlas, R. Cole, Y. Muthusamy, A. Lippman, J. Connor, D. Park, M. Elsharkawi, and R. J. Marks II. A performance comparison of trained multilayer perceptrons and trained classification trees. *Proc. IEEE*, 78(10):1614–1619, Oct. 1990.

[5] T. Bailey and A. K. Jain. A note on distance-weighted k-nearest-neighbor rules. *IEEE Trans. Syst., Man, and Cybern.*, SMC-8:311–313, 1978.

[6] P. Baldi and K. Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2:53–58, 1989.

[7] G. H. Ball and D. J. Hall. Some fundamental concepts and synthesis procedures for pattern recognition preprocessors. In *Proc. Intl. Conf. on Microwaves, Circuit Theory, and Information Theory*, pages 281–297, Tokyo, September 1964.

[8] A. R. Barron. Predicted squared error: A criterion for automatic model selection. In S. Farlow, editor, *Self-Organizing Methods in Modeling.* Marcel Dekker, New York, 1984.

[9] A. R. Barron. Statistical properties of artificial neural networks. In *Proc. of the 28th Conference on Decision and Control*, pages 280–285, Tampa, Florida, Dec. 1989.

[10] A. R. Barron. Complexity regularization with application to artificial neural networks. In G. Roussas, editor, *Nonparametric Function Estimation and Related Topics*, pages 561–576. Kluwer Academic Publishers, The Netherlands, 1991.

[11] A. R. Barron. Complexity regularization with application to artificial neural networks. In G. Roussas, editor, *Nonparametric Function Estimation and Related Topics*, pages 561–576. Kluwer Academic Publishers, The Netherlands, 1991.

[12] A. R. Barron and R. L. Barron. Statistical learning networks: A unifying view. In E. G. Wegman, editor, *Proc. of the* 10^{th} *Symposium on the Interface*, pages 192–203, Washington D.C., 1988. American Statistical Association.

[13] A. R. Barron and R. L. Barron. Statistical learning networks: A unifying view. In E. G. Wegman, editor, *Proc. of the* 10^{th} *Symposium on the Interface*, pages 192–203, Washington D.C., 1988. American Statistical Association.

[14] E. B. Baum and D. Haussler. What size net gives valid generalization? *Neural Computation 1*, pages 151–160, 1989.

[15] J. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms.* Plenum, New York, 1981.

[16] G. Biswas, A. K. Jain, and R. C. Dubes. Evaluation of projection algorithms. *IEEE Trans. Pattern Anal. Machine Intell.*, PAMI-3(No.6):701–708, Nov. 1981.

[17] A. Blumer, A. Ehrenfeucht, D. Haussler, and K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, 1989.

[18] P. Brodatz. *Textures - A Photographic Album for Artists and Designers.* Dover, New York, 1966.

[19] D. J. Burr. Experiments on neural net recognition of spoken and written text. *IEEE Trans. Acoust., Speech, and Signal Process.*, 36:1162–1168, July 1988.

[20] P. Cardaliaguet and G. Euvrard. Approximation of a function and its derivative with a neural network. *Neural Networks*, 5:207–220, 1992.

[21] G. Carpenter and S. Grossberg. Adaptive resonance theory: Stable self-organization of neural recognition codes in response to arbitrary lists of input patterns. In *Eighth Annual Conference of the Cognitive Science Society*, pages 45–62. Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

[22] G. Carpenter and S. Grossberg. ART2: self-organization of stable category recognition codes for analog input patterns. *Applied Optics*, 26(23):4919–4930, Dec. 1987.

[23] G. Carpenter and S. Grossberg. ART3: Hierarchical search using chemical transmitters in self-organizing pattern recognition architectures. *Neural Networks*, 3:129–152, 1990.

[24] Y. Chauvin. A backpropagation algorithm with optimal use of hidden units. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 519–526. Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[25] Y. Chauvin. Dynamic behavior of constrained back-propagation networks. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 642–649. Morgan Kaufmann Publishers, San Mateo, CA, 1990.

[26] T. M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Trans. on Electronic Computers*, EC-14:326–334, 1965.

[27] G. Cybenko. Approximation by superpositions of sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.

[28] J. G. Daugman. Uncertainty relation for resolution in space, spatial-frequency, and orientation optimized by two-dimensional visual cortical filters. *Journal of Opt. Soc. Am.*, 2:1160–1169, 1985.

[29] P. A. Devijver and J. Kittler. On the edited nearest neighbor rule. In *Proc. Fifth Int. Conf. on Pattern Recognition*, pages 72–80, Miami Beach, FL, 1980.

[30] P. A. Devijver and J. Kittler. *Pattern Recognition: A Statistical Approach.* Prentice Hall, Englewood Cliffs, New Jersey, 1982.

[31] L. Devroye. Automatic pattern recognition: A study of the probability of error. *IEEE Trans. Pattern Anal. Machine Intell.*, 10(4):530–543, July 1988.

[32] B. Dom, W. Niblack, and J. Sheinvald. Feature selection with stochastic complexity. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 241–248, San Diego, June 1989.

[33] R. C. Dubes. How many clusters are there? – an experiment. *Pattern Recognition*, 20(6):645–663, 1987.

[34] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* Wiley, New York, 1973.

[35] R. P. W. Duin. On the choice of smoothing parameters for parzen estimators of probability density functions. *IEEE Trans. on Computers*, 25:1175–1179, 1976.

[36] R. P. W. Duin. *On the Accuracy of Statistical Pattern Recognizers.* PhD thesis, Dutch Efficiency Bureau, Pijnacker, The Netherlands, 1978.

[37] R. Eubank. *Spline Smoothing and Nonparametric Regression.* Marcel Dekker, 1988.

[38] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 524–532. Morgan Kaufmann Publishers, San Mateo, CA, 1990.

[39] F. Farrokhnia. *Multi-channel Filtering Techniques For Texture Segmentation and Surface Quality Inspection.* PhD thesis, Dept. of Electrical Engg., Michigan State University, 1990.

[40] P. Foldiak. Adaptive network for optimal linear feature extraction. In *Proc. of IEEE Intl. Joint Conf. on Neural Networks*, volume 1, pages 401–405, Washington D.C., 1989.

[41] D. H. Foley and J. W. Sammon. An optimal set of discriminant vectors. *IEEE Trans. on Computers*, C-24(No.3):281–289, March 1975.

[42] E. Forgy. Cluster analysis of multivariate data: efficiency versus interpretability of classification. *Biometrics*, 21:768, 1965.

[43] J. H. Friedman. Regularized discriminant analysis. *Journal of the American Statistical Association*, 84(405):165–175, March 1989.

[44] J. H. Friedman and J. W. Tukey. A projection pursuit algorithm for exploratory data analysis. *IEEE Trans. Comput.*, C-23:881–890, Sept. 1974.

[45] K. Fukunaga. *Introduction to Statistical Pattern Recognition.* Second Edition. New York: Academic Press, 1990.

[46] K. Fukunaga and T. E. Flick. An optimal global nearest neighbor metric. *IEEE Trans. Pattern Anal. Machine Intell.*, PAMI-6:314–318, 1984.

[47] K. Fukunaga and L. D. Hostetler. Optimization of k-nearest neighbor density estimates. *IEEE Trans. Inform. Theory*, 19:320–326, 1973.

[48] K. Fukunaga and P. M. Narendra. A branch and bound algorithm for computing k-nearest neighbors. *IEEE Trans. Comput.*, C-24:750–753, 1975.

[49] K. Funahashi. On the approximation realization of continuous mappings by the neural networks. *Neural Networks*, 2:183–192, 1989.

[50] N. P. Galatsanos and A. K. Katsaggelos. Methods for choosing the regularization parameter and estimating the noise variance in image restoration and their relation. *IEEE Trans. on Image Processing*, 1(3):322–336, July 1992.

[51] A. R. Gallant and H. White. There exists a neural network that does not make avoidable mistakes. In *Proc. IEEE Second Intl. Conf. on Neural Networks*, pages I657–664, San Diego, California, 1988.

[52] A. R. Gallant and H. White. On learning the derivatives of an unknown mapping with multilayer feedforward networks. *Neural Networks*, 5:129–138, 1992.

[53] P. Gallinari, S. Thiria, F. Badran, and F. Fogelman-Soulie. On the relations between discriminant analysis and multilayer perceptrons. *Neural Networks*, 4:349–360, 1991.

[54] G. W. Gates. The reduced nearest neighbor rule. *IEEE Trans. Info. Theory*, IT-18:431–433, May 1972.

[55] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58, 1992.

[56] S. J. Hanson and L. Y. Pratt. Comparing biases for minimal network construction with back-propagation. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 177–185. Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[57] P. E. Hart. The condensed nearest neighbor rule. *IEEE Trans. Info. Theory*, IT-14:515–516, May 1968.

[58] J. Hartigan. *Clustering Algorithms*. Wiley, New York, 1975.

[59] B. Hassibi and D. G. Stork. Optimal brain surgeon and general network pruning. In *Proc. of IEEE Intl. Conf. on Neural Networks*, volume 1, pages 293–299, San Francisco, California, 1993.

[60] D. Haussler. Decision theoretic generalization of the PAC model for neural net and other learning applications. Technical report, UCSC-CRL-91-02, Baskin Center for Computer Engineering and Information Sciences, University of California, Santa Cruz, 1991.

[61] D. O. Hebb. *The Organization of Behavior*. Wiley, New York, 1949.

[62] R. Hecht-Nielsen. Theory of back propagation neural network. In *Proc. Intl. Joint Conf. on Neural Networks*, pages I593–606, San Diego, California, 1989.

[63] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City, 1991.

[64] G. E. Hinton. Learning translation invariant recognition in a massively parallel network. In *Proc. Conf. Parallel Architectures and Languages Europe*, pages 1–13, Eindhoven, The Netherlands, 1987.

[65] G. E. Hinton and T. J. Sejnowski. Learning and relearning in Boltzman machines. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Exploration in the microstructure of cognition*, volume 1, pages 282–317. MIT Press, Cambridge, MA, 1986.

[66] R. L. Hoffman and A. K. Jain. Segmentation and Classification of Range Images. *IEEE Trans. Pattern Anal. Machine Intell.*, PAMI-9(5):608–620, 1987.

[67] L. Holmstrom and P. Koistinen. Using adaptive noise in back-propagation training. *IEEE Trans. Neural Networks.*, 3(1):24–38, Jan. 1992.

[68] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. In *Proc. Natl. Acad. Sci. USA 79*, pages 2554–2558, 1982.

[69] K. Hornik and C.-M. Kuan. Convergence analysis of local feature extraction algorithm. *Neural Networks*, 5:229–240, 1992.

[70] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[71] W. Y. Huang and R. P. Lippmann. Comparisons between neural net and traditional classifiers. In *IEEE 1st Intl. Conf. on Neural Networks*, pages IV485–IV493, San Diego, CA, June 1987.

[72] W. Y. Huang and R. P. Lippmann. Neural net and traditional classifiers. In D. Anderson, editor, *Neural Info. Processing Syst.*, pages 387–396. American Institute of Physics, New York, 1988.

[73] P. J. Huber. Projection pursuit. *Ann. Statist.*, 13:435–475, 1985.

[74] D. R. Hush and B. G. Horne. Progress in supervised neural networks. *IEEE Signal Processing Magazine*, pages 8–39, January 1993.

[75] J.-N. Hwang, S.-R. Lay, and A. Lippman. Unsupervised learning for multivariate probability density estimation: radial basis function and exploratory projection pursuit. In *1993 IEEE Intl. Conf. on Neural Networks*, volume III, pages 1486–1491, San Francisco, California, March 28 - April 1 1993.

[76] J.-N. Hwang, D. Li, M. Maechler, D. Martin, and J. Schimert. Regression modeling in back-propagation and projection pursuit learning. *To appear in IEEE Trans. on Neural Networks*, 1994.

[77] B. Irie and S. Miyake. Capabilities of three layer perceptrons. In *Proc. IEEE Second Intl. Conf. on Neural Networks*, pages I641–648, San Diego, California, 1988.

[78] Y. Ito. Approximation of continuous functions on r^d by linear combinations of shifted rotations of a sigmoid function with and without scaling. *Neural Networks*, 5:105–115, 1992.

[79] A. K. Jain. Pattern recognition. In *International Encyclopedia of Robotics Applications and Automation*, pages 1052–1063. Wiley, 1988.

[80] A. K. Jain and B. Chandrasekaran. Dimensionality and sample size considerations in pattern recognition practice. In P. R. Krishnaiah and L. N. Kanal, editors, *Handbook of Statistics, Vol. 2*, pages 835–855. North Holland Publishing Company, 1982.

[81] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data.* Prentice Hall, Englewood Cliffs, NJ, 1988.

[82] A. K. Jain and F. Farrokhnia. Unsupervised texture segmentation using Gabor filters. *Pattern Recognition*, 24(12):1167–1186, 1991.

[83] A. K. Jain and K. Karu. Learning filter masks for discriminating textures. In *IEEE Intl. Conf. on Neural Networks*, pages 4374–4379, Orlando, Florida, June 1994.

[84] A. K. Jain and J. Mao. A k-nearest neighbor artificial neural network classifier. In *Proc. 1991 IEEE Intl. Joint Conf. on Neural Networks*, volume 2, pages 515–520, Seattle, Washington, July 1991.

[85] A. K. Jain and J. Mao. Artificial neural network for nonlinear projection of multivariate data. In *Proc. of IEEE Intl. Joint Conf. on Neural Networks*, volume 3, pages 335–340, Baltimore, Maryland, June 1992.

[86] A. K. Jain and J. Mao. Neural networks and pattern recognition. In J. M. Zurada, R. J. Marks II, and C. J. Robinson, editors, *Computational Intelligence: Imitating Life*, pages 194–212. IEEE Press, New York, 1994.

[87] A. K. Jain and M. D. Ramaswami. Classifier design with Parzen windows. In E. S. Gelsma and L. N. Kanal, editors, *Pattern Recognition and Artificial Intelligence*, pages 211–228. Elsevier Science Publishers B. V. (North-Holland), Amsterdan, 1988.

[88] A. K. Jain and W. Waller. On the optimum number of features in the classification of multivariate Gaussian data. *Pattern Recognition*, 10:365–374, 1978.

[89] J.-M. Jolion, P. Meer, and S. Bataouche. Robust clustering with applications in computer vision. *IEEE Trans. Pattern Anal. Machine Intell.*, 13(8):791–802, 1991.

[90] E. R. Kandel and J. H. Schwartz. *Principles of Neural Science.* New York: Elsevier, 1985.

[91] E. D. Karnin. A simple procedure for pruning back-propagation trained neural networks. *IEEE Trans. Neural Networks.*, 1(2):239–242, 1990.

[92] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biol. Cybern.*, 43:59–69, 1982.

[93] T. Kohonen. *Self Organization and Associative Memory.* Third edition, Springer-Verlag, 1989.

[94] T. Kohonen. Self-organized network. *Proc. IEEE*, 43:59–69, 1990.

[95] T. Kohonen. Self-organizing maps: optimization approach. In T. Kohonen, K. Makisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, pages 981–990. Elsevier Science, 1991.

[96] T. Kohonen, G. Barna, and R. Chrisley. Statistical pattern recognition with neural networks: Benchmarking studies. In *Proc. 1988 IEEE Intl. Joint Conf. on Neural Networks*, pages I61–I68, San Diego, California, July 1988.

[97] P. Koistinen and L. Holmstrom. Kernel regression and backpropagation training with noise. In *Proc. of Intl. Joint Conf. on Neural Networks*, pages 367–372, Singapore, 1991.

[98] M. A. Kraaijveld. *Small Sample Behavior of Multi-Layer Feedforward Network Classifiers: Theoretical and Practical Aspects.* PhD thesis, Delft University, The Netherlands, 1993.

[99] M. A. Kraaijveld and R. P. W. Duin. On backpropagation learning of edited data sets. In *Proc. 1990 Intl. Neural Network Conf.*, pages 741–744, Paris, July 1990.

[100] M. A. Kraaijveld, J. Mao, and A. K. Jain. A non-linear projection method based on Kohonen's topology preserving maps. In *Proc. of Intl. Conf. on Pattern Recognition (To appear in IEEE Trans. on Neural Networks)*, volume 2, pages 41–45, The Hague, The Netherlands, 1992.

[101] A. Krogh and J. A. Hertz. Hebbian learning of principal components. In R. Eckmiller, G. Hartmann, and G. Hauske, editors, *Parallel Processing in Neural Systems and Computers*, pages 183–186. Elsevier Science Publishers B. V. (North-Holland), 1990.

[102] H. Kuhnel and P. Tavan. A network for discriminant analysis. In T. Kohonen, K. Makisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, pages 1053–1056. Elsevier Science, 1991.

[103] S. Kung and K. Diamantaras. A neural network learning algorithm for adaptive principal component extraction (APEX). In *IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, volume 1, pages 861–864, 1990.

[104] K. J. Lang, A. H. Waibel, and G. E. Hinton. A time-delay neural network architecture for isolated word recognition. *Neural Networks*, 3:23–43, 1990.

[105] Y. leCun. Generalization and network design strategies. Technical report, CRG-TR-89-4, University of Toronto, 1989.

[106] Y. leCun, 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*, 1(4):541–551, 1989.

[107] Y. leCun, J. Denker, and S. A. Solla. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. Morgan Kaufmann Publishers, San Mateo, CA, 1990.

[108] A. Lindon and J. Kindermann. Inversion of multilayer nets. In *Proc. of Intl. Joint Conf. on Neural Networks*, volume II, pages 425–430, 1989.

[109] R. P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, 4 (2):4–22, Apr. 1987.

[110] R. P. Lippmann. Pattern classification using neural networks. *IEEE Communications Magazine*, pages 47–64, Nov. 1989.

[111] J. Mao and A. K. Jain. Discriminant analysis neural networks. In *Proc. of IEEE Intl. Conf. on Neural Networks*, volume 1, pages 300–305, San Francisco, CA, March 1993.

[112] J. Mao and A. K. Jain. Regularization techniques in artificial neural networks. In *Proc. of World Congress on Neural Networks*, pages IV75–IV79, Portland, Oregon, July 1993.

[113] J. Mao and A. K. Jain. Artificial neural networks for feature extraction and multivariate data projection. *Accepted to appear in IEEE Trans. Neural Networks*, 1994.

[114] J. Mao and A. K. Jain. A self-organizing network for hyperellipsoidal clustering (HEC). In *Proc. IEEE Intl. Conf. on Neural Networks*, pages 2967–2972, Orlando, Florida, June 1994.

[115] J. Mao, K. Mohiuddin, and A. K. Jain. Minimal network design and feature selection through node-pruning. *To appear in 12th Intl. Conf. on Pattern Recognition, Jerusalem, Israel*, Oct. 9-13 1994.

[116] M. Marchand, M. Golea, and P. Rujan. A convergence theorem for sequential learning in two-layer perceptrons. *Europhysics Letters*, 11:487–492, 1990.

[117] W. S. McCulloch and W. Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[118] M. Mezard and J.-P. Nadal. Learning in feedforward layered networks: The tiling algorithm. *Journal of Physics A*, 22:2191–2204, 1989.

[119] W. T. Miller, F. H. Glanz, and L. Gordon Kraft. CMAC: An associative neural network alternative to backpropagation. *Proc. of IEEE*, 78(10):1561–1567, October 1990.

[120] G. W. Milligan and M. C. Cooper. A study of standardization of variables in cluster analysis. *Journal of Classification*, 5:181–204, 1988.

[121] J. I. Minnix. Fault tolerance of the backpropagation neural network trained on noisy inputs. In *Proc. of IEEE Intl. Joint Conf. on Neural Networks*, volume 1, pages 847–852, Baltimore, Maryland, June 1992.

[122] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, 1969.

[123] J. Moody and Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation 1*, pages 281–294, 1989.

[124] J. Moody and J. Utans. Principled architecture selection for neural networks: Application to corporate bond rating prediction. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*. Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[125] J. E. Moody. Note on generalization, regularization, and architecture selection in nonlinear learning systems. In B. H. Juang, S. Y. Kung, and C. A. Kamm, editors, *Neural Networks for Signal Processing - Proc. of the 1991 IEEE Workshop*, pages 1–10. 1991.

[126] J. E. Moody. The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems*. Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[127] B. K. Moor. ART 1 and pattern clustering. In *Proc. the 1988 Connectionist Summer School*, pages 174–185. Morgan-Kaufman, 1988.

[128] M. M. Moya and L. D. Hostetler. One-class generalization in second-order backpropagation networks for image classification. In *Proc. of Intl. Joint Conf. on Neural Networks*, volume II, pages 221–224, Washington D.C., 1990.

[129] M. C. Mozer and P. Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 107–115. Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[130] C. Neti, M. H. Schneider, and E. D. Young. Maximally fault tolerant neural networks. *IEEE Trans. Neural Networks*, 3(1):14–23, January 1992.

[131] S. J. Nowlan and G. E. Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4:473–493, 1992.

[132] E. Oja. A simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15:267–273, 1982.

[133] E. Oja. Neural networks, principal components, and subspaces. *Intl. Journal of Neural Systems*, 1:61–68, 1989.

[134] T. Okada and S. Tomita. An optimal orthonormal system for discriminant analysis. *Pattern Recognition*, 18:139–144, 1985.

[135] N. R. Pal, J. C. Bezdek, and E. C.-K. Tsao. Generalized clustering networks and Kohonen's self-organizing scheme. *IEEE Trans. Neural Networks*, 4(4):549–557, 1993.

[136] Y. H. Pao. *Adaptive Pattern Recognition and Neural Networks*. Addison-Wesley, Reading, MA, 1989.

[137] C. Peterson and J. R. Anderson. A mean field theory learning algorithm for neural networks. *Complex Systems*, 1:995–1019, 1987.

[138] T. Poggio and F. Girosi. Networks for approximation and learning. *Proc. IEEE*, 78(9):1481–1497, September 1990.

[139] T. Poggio and F. Girosi. Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, 247:978–982, February 1990.

[140] T. Poggio, V. Torre, and C. Koch. Computational vision and regularization theory. *Nature*, 317:638–643, 1985.

[141] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing.* Second Edition, Cambridge University Press, Cambridge, England, 1988.

[142] S. Raudys and A. K. Jain. Small sample size problems in designing artificial neural networks. In I. Sethi and A. K. Jain, editors, *Artificial Neural Networks and Pattern Recognition: Old and New Connections*, pages 33–50. Elsevier, Amsterdan, 1991.

[143] S. J. Raudys and A. K. Jain. Small sample size effects in statistical pattern recognition: Recommendations for practitioners. *IEEE Trans. Pattern Anal. Machine Intell.*, 13(3):252–264, March 1991.

[144] R. Reed. Pruning algorithms – a survey. *IEEE Trans. Neural Networks.*, 4(5):740–747, Sept. 1993.

[145] R. Reed, S. Oh, and R. Marks III. Regularization using jittered training data. In *Proc. of IEEE Intl. Joint Conf. on Neural Networks*, volume 3, pages 147–152, Baltimore, Maryland, June 1992.

[146] M. D. Richard and R. P. Lippmann. Neural network classifiers estimate Bayesian *a posteriori* probabilities. *Neural Compuatation*, 3:461–483, 1991.

[147] B. D. Ripley. Neural networks and related methods for classification. *J. R. Statist. Soc. B*, 56(3), 1994.

[148] J. Rissanen. *Stochastic Complexity in Statistical Inquiry.* World Scientific, Singapore, 1989.

[149] R. Rosenblatt. *Principles of Neurodynamics.* Spartan Books, New York, 1962.

[150] J. Rubner and K. Schulten. Development of feature detectors by self-organization. *Biol. Cybern.*, 62:193–199, 1990.

[151] J. Rubner and P. Tavan. A self-organizing network for principal component analysis. *Europhysics Letters*, 10:693–698, 1989.

[152] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Exploration in the microstructure of cognition*, volume 1, pages 318–362. MIT Press, Cambridge, MA, 1986.

[153] J. W. Sammon Jr. A non-linear mapping for data structure analysis. *IEEE Trans. Comput.*, C-18:401–409, 1969.

[154] T. D. Sanger. Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks*, 2:459–473, 1989.

[155] T. J. Sejnowski and C. R. Rosenberg. Parallel networks that learn to pronounce English text. *Complex Systems*, 1:145–168, 1987.

[156] I. Sethi and A. K. Jain (eds.). *Artificial Neural Networks and Pattern Recognition: Old and New Connections.* Elsevier, Amsterdan, 1991.

[157] I. K. Sethi. Entropy nets: From decision trees to neural networks. *Proc. IEEE*, 78(10):1605–1613, Oct. 1990.

[158] J. Sietsma and R. J. F. Dow. Neural net pruning – why and how. In *Proc. of IEEE Intl. Conf. on Neural Networks*, volume I, pages 325–333, San Diego, California, 1988.

[159] J. Sietsma and R. J. F. Dow. Creating artificial neural networks that generalize. *Neural Networks*, 4:67–79, Jan. 1991.

[160] B. W. Silverman. *Density Estimation for Statistics and Data Analysis.* Chapman and Hall, London, 1985.

[161] J.-A. Sirat and J.-P. Nadal. Neural trees: A new tool for classification. Technical report, Laboratories d'Electronique Philips, France, 1990.

[162] J. Sjoberg and L. Ljung. Overtraining, regularization, and searching for minimum in neural networks. Technical report, Department of Electrical Engineering, Linkoping University, S-581 83 Linkoping, Sweden, Feb. 1992.

[163] S. P. Smith and A. K. Jain. Testing for uniformity in multidimensional data. *IEEE Trans. Pattern Anal. Machine Intell.*, 6(1):73–81, 1984.

[164] D. F. Specht. Probabilistic neural networks and the polynomial adaline as complementary techniques for classification. *IEEE Trans. Neural Networks.*, 1(1):111–121, March 1990.

[165] M. Stinchcombe and H. White. Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions. In *Proc. Intl. Joint Conf. on Neural Networks*, pages I613–618, San Diego, California, 1989.

[166] G. Tesauro. Neurogammon wins computer Olympiad. *Neural Computation*, 1:321–323, 1989.

[167] A. M. Thompson, J. C. Brown, J. W. Kay, and D. M. Titterington. A study of methods of choosing the smoothing parameter in image restoration by regularization. *IEEE Trans. Pattern Anal. Machine Intell.*, 13(4):326–339, April 1991.

[168] A. N. Tikhonov and V. Y. Arsenin. *Solutions of Ill-Posed Problems.* Winston & Sons, Washington D.C., 1977.

[169] G. V. Trunk. A problem of dimensionality: a simple example. *IEEE Trans. Pattern Anal. Machine Intell.*, PAMI-1(3):306–307, 1979.

[170] V. N. Vapnik. *Estimation of Dependences Based on Empirical Data.* Springer Verlag, New York, 1982.

[171] V. N. Vapnik, E. Levin, and Y. LeCun. Measuring the VC dimension of a learning machine. *To appear in Neural Computation*, 1994.

[172] S. Watanabe. *Pattern Recognition: Human and Mechanical.* Wiley, New York, 1985.

[173] A. R. Webb and D. Lowe. The optimised internal representation of multilayer classifier networks performs nonlinear discriminant analysis. *Neural Networks*, 3:367–375, 1990.

[174] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman. Generalization by weight-elimination with application to forecasting. In R. P. Lippmann, J. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 875–882. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[175] R. S. Wenocur and R. M. Dudley. Some special Vapnik-Chervonenkis classes. *Discrete Mathematics*, 33:313–318, 1981.

[176] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Applied Mathematics, Harvard University, November 1974.

[177] P. J. Werbos. Links between artificial neural networks (ann) and statistical pattern recognition. In I. Sethi and A. K. Jain, editors, *Artificial Neural Networks and Pattern Recognition: Old and New Connections*, pages 11–31. Elsevier, Amsterdan, 1991.

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

[179] H. White. Connectionist nonparametric regression: Multilayer feedforward networks can learn arbitrary mappings. *Neural Networks*, 3:535–549, 1990.

[180] D. Whitley and C. Bogart. The evolution of connectivity: Pruning neural networks using genetic algorithms. In *Proc. of Intl. Joint Conf. on Neural Networks*, volume I, page 134, Washington, D.C., 1990.

[181] A. Whitney. A direct method of nonparametric measurement selection. *IEEE Trans. on Computers*, C-20:1100–1103, 1971.

[182] B. Widrow. ADALINE and MADALINE – 1963. In *IEEE 1st Intl. Conf. on Neural Networks*, pages I143–I157, San Diego, CA, June 1987.

[183] D. L. Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Trans. Syst., Man, and Cybern.*, SMC-2:408–420, 1972.

[184] L. Xu, A. Krzyzak, and E. Oja. Rival penalized competitive learning for clustering analysis, RBF net, and curve detection. *IEEE Trans. Neural Networks*, 4(4):636–649, 1993.

[185] E. Yair and A. Gersho. The Boltzmann perceptron network: A soft classifier. *Neural Networks*, 3:203–221, 1990.

[186] Hung-Chun Yau and M. T. Manry. Iterative improvement of a Gaussian classifier. *Neural Networks*, 3:437–443, 1990.

[187] Y. Zhao and C. G. Atkeson. Projection pursuit learning. In *Proc. 1991 IEEE Intl. Joint Conf. on Neural Networks*, volume 1, pages 869–874, Seattle, Washington, July 1991.

[188] J. M. Zurada. *Introduction to Artificial Neural Systems.* West Publishing Co., St. Paul, Minnesota, 1992.