

ABSTRACT

LIAO, YI. Neural Networks for Pattern Classification and Universal Approximation

(Under the direction of Dr. Shu-Cherng Fang and Dr. Henry L. W. Nuttle).

This dissertation studies neural networks for pattern classification and universal approximation. The objective is to develop a new neural network model for pattern classification, and relax the conditions for Radial-Basis Function networks to be universal approximators. First, the problem of pattern classification is introduced, which is followed by a brief introduction of three popular nonlinear classification techniques, that is, Multi-Layer Perceptrons (MLP), Radial Basis Function (RBF) networks, and Support Vector Machines (SVM). Then, based on the basic concepts of MLP, RBF and SVM, a new neural network model with bounded weights is proposed, and some experimental results are reported. Later, the problem of universal approximation by neural networks is introduced, and the researches on ridge activation functions and radial-basis activation functions are reviewed. Then, the relaxed conditions for RBF networks to be universal approximators are presented. We show that RBF networks can uniformly approximate any continuous function on a compact set provided that the radial basis activation function is continuous almost everywhere, locally essentially bounded, and not a polynomial. Some experimental results are reported to illustrate our findings. The dissertation ends with the conclusion and future research.

Neural Networks for Pattern Classification and Universal Approximation

by

YI LIAO

A dissertation submitted to the Graduate Faculty of

North Carolina State University

in partial fulfillment of the

requirements for the Degree of

Doctor of Philosophy

DEPARTMENT OF INDUSTRIAL ENGINEERING

Raleigh

2002

APPROVED BY:

Dr. Shu-Cherng Fang
Chair of Advisory Committee

Dr. Henry L. W. Nuttle
Co-Chair of Advisory Committee

Dr. Yuan-Shin Lee
Advisory Committee

Dr. Jesus Rodriguez
Advisory Committee

BIOGRAPHY

Yi Liao is a Ph.D. student of Department of Industrial Engineering at North Carolina State university. He received his B.S. in Department of Automatic Control from Hua Zhong University of Science and Technology, P. R. China, in 1994. Then he received his M.S. in the same department from the same university in 1997. In the Spring of 1998, he came to United States to start his Ph.D. study in Department of Industrial Engineering at North Carolina State University. While working on his Ph.D., he was employed as a research assistant in Department of Industrial Engineering from 1998-2000, a technical student at SAS Institute from 2000-2002. His research interests include neural networks, pattern recognition, data mining, artificial intelligence, fuzzy control, and mathematical optimization.

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to my advisors: Dr. Shu-Cherng Fang, for his thoughtful advice, encouragement, kindness and guidance through out my Ph.D. study; and Dr. Henry L. W. Nuttle, for his kindness, patience and valuable advice. I am also grateful to Dr. Yuan-Shin Lee and Dr. Rodriguez for their teaching, service as my committee members and thoughtful comments. I also would like to thank Dr. Edward Grant for being the Graduate Representative.

I would like to thank Dr. Radhika Kulkarni for her kindness, help and patience while working at SAS Institute. I also would also like to thank FANGroup members, especially, Shyh-Huei, Andrés, Ta-Wei, and Nan-Chieh, for their brotherly help and care. I also would like to thank other FANGroup members for their assists and support.

I am deeply indebted to my parents for their love and encouragement. Finally, I would like to thank my beloved girlfriend, Bingyan Wu, for her love and enduring support. Her encouragement and support made this study possible.

Contents

List of Figures	vii
------------------------	------------

List of Tables	viii
-----------------------	-------------

1 Introduction	1
1.1 Pattern Classification and Universal Approximation	1
1.2 Research Scope and Objective	3
1.3 Dissertation Organization	4
2 Pattern Classification	5
2.1 Problem Description	5
2.2 Basic Approaches	7
2.2.1 Parametric Methods	7
2.2.2 Density-Based Methods	9
2.2.3 Linear Discriminant Methods	12
2.2.4 Nonlinear Discriminant Methods	15
3 Neural Networks and Support Vector Machines	17
3.1 Multi-Layer Perceptrons	17
3.1.1 Introduction	17
3.1.2 Model Specification	18
3.1.3 Discussion	23
3.2 Radial Basis Function Networks	24
3.2.1 Introduction	24

3.2.2	Motivation	25
3.2.3	Model Specification	26
3.2.4	Discussion	29
3.3	Support Vector Machines	29
3.3.1	Introduction	29
3.3.2	SVM Model	30
3.3.3	Discussion	37
4	Neural Networks with Bounded Weights	38
4.1	Motivation	38
4.2	Proposed Model	39
4.2.1	Two-Class Case	39
4.2.2	Multi-Class Case	42
4.3	Experimental Results	45
4.3.1	Small Example	45
4.3.2	Real-World Examples	46
4.3.3	General Kernel Functions	50
4.3.4	Multi-Class Classification	52
5	Universal Approximation by Neural Networks	53
5.1	Introduction	53
5.2	Basic Definitions and Notation	55
5.3	Literature Review	58
5.3.1	Ridge Activation Functions	58
5.3.2	Radial-Basis Activation Functions	60
6	Relaxed Conditions	62
6.1	Main Result	62
6.2	Numerical Experiments	71
6.2.1	One-Dimensional Example	73
6.2.2	Multi-Dimensional Examples	73

7 Conclusion and Future Research	81
7.1 Conclusion	81
7.2 Future Research	82
Bibliography	84

List of Figures

2-1	Various Approaches in Supervised Classification	7
3-1	Multilayer Perceptron	19
3-2	SVM - Linearly Separable Data	31
3-3	SVM - Linearly Non-Separable Data	35
4-1	Bounded Neural Network: Two-Class Case	40
4-2	Bounded Neural Network: Multi-Class Case	43
4-3	The Support Vectors for the BNN Model	46
4-4	The Support Vectors for the SVM Model	47
6-1	The function to be approximated.	73
6-2	The result for the one dimension, non-Gaussian, and RS.	75
6-3	The result for the one dimension, Gaussian, and RS.	75
6-4	The result for the one dimension, non-Gaussian, and OLS.	76
6-5	The result for the one dimension, Gaussian, and OLS.	76
6-6	The result for the heart disease, non-Gaussian, and RS.	77
6-7	The result for the heart disease, Gaussian, and RS.	77
6-8	The result for the heart disease, non-Gaussian, and OLS.	78
6-9	The result for the heart disease, Gaussian, and OLS.	78
6-10	The result for the liver disorder, non-Gaussian, and RS.	79
6-11	The result for the liver disorder, Gaussian, and RS.	79
6-12	The result for the liver disorder, non-Gaussian, and OLS.	80
6-13	The result for the liver disorder, Gaussian, and OLS.	80

List of Tables

3.1	Comonly Used RBFs	26
3.2	Commonly Used Kernel Functions	36
4.1	The Experimental Results for the Gaussian Kernel Function	51
4.2	The Experimental Results for the Polynomial Kernel Function	51
4.3	The Experimental Results for the Sigmoid Kernel Function	51
4.4	The Experimental Results for General Kernel Functions	52
4.5	The Experimental Results for the Iris Flower Classification	52

Chapter 1

Introduction

1.1 Pattern Classification and Universal Approximation

Pattern classification is an important problem in a variety of engineering and scientific disciplines, such as biology, psychology, medicine, marketing, computer vision, artificial intelligence, and remote sensing. Recently, interest in the area of pattern classification has been increasing due to the emerging applications which are not only challenging but also computationally more demanding. These applications include data mining (identifying a “pattern”, e.g., correlation, or an outlier in millions of multi-dimensional data points), document classification (efficiently searching text documents), financial forecasting, organization and retrieval of multi-media databases, and biometrics (personal identification based on the various physical attributes such as face and fingerprints). The rapidly growing and available computing power, while enabling faster processing of huge data sets, has also facilitated the use of elaborate and diverse methods for pattern classification.

But what is pattern classification? In pattern classification, we are given a set of training samples with associated “class labels”. The problem is to design a classifier to predict the class label for any given sample. Pattern classification is one of the two categories of “Pattern Recognition”. The other category is “Clustering”, in which the given samples are not labelled and the problem is to classify the data into groups

with features that distinguish one group from another.

Among the various approaches for pattern classification (e.g., Bayes plug in rule, k -nearest neighbor, linear discrimination models.), multilayer feedforward neural networks have been shown to be quite effective in many different applications. Most papers report that multilayer feedforward neural networks perform at least as well as their traditional competitors. This success has led many researchers to undertake a rigorous analysis of the mathematical properties that enable multilayer feedforward neural networks to perform well in the field. The motivation for this line of research was eloquently described by Hornik, Stinchcombe and White [51] as follows: “The apparent ability of sufficiently elaborate feedforward networks to approximate quite well nearly any function encountered in applications leads one to wonder about the ultimate capabilities of such networks. Are the successes observed to date reflective of some deep and fundamental approximation capabilities, or are they merely fluke, resulting from selective reporting and fortuitous choice of problems?”

In literature, the attention has usually been focused on the approximation capability of three-layered feedforward neural networks with one output node. The class of functions realized by such networks has the following form:

$$\sum_{i=1}^N c_i g(x, \theta_i, b_i),$$

where N is the number of hidden nodes, $x \in R^n$ is a variable, $c_i \in R$, $\theta_i \in R^n$, $b_i \in R$ are parameters, and $g(x, \theta_i, b_i)$ is the activation function used in the hidden layer. For such networks, a fundamental question is: If we are free to choose θ_i , b_i and N , then what conditions should the activation function g satisfy, such that the neural network can approximate a given class of functions (such as continuous functions, or integrable functions), i.e., such that the network is a universal approximator for the class of functions?

1.2 Research Scope and Objective

This dissertation studies neural networks for pattern classification and neural networks for universal approximation. Our objective is to develop a new neural network model for pattern classification, and to relax the conditions for Radial-Basis Function networks to be universal approximators.

For pattern classification, Multi-Layer Perceptrons (MLP), Radial-Basis Function Networks (RBFN) and Support Vector Machines (SVM) are three powerful techniques. But they all have their own advantages and disadvantages. Specifically, MLP models are easy to be understood and applied, but the training requires nonlinear optimization, which can not guarantee global optimum. For RBFN models, although the training is equivalent to solving a linear equation, the performance of RBFN is usually not as good as that of MLP and SVM. For the SVM models, the training problem is a convex programming problem, but SVM are not easily applicable to multi-class classification. In this dissertation, we will develop a new neural network model to try to overcome the disadvantages of MLP, RBFN and SVM.

The studies on universal approximation by neural networks can be categorized into two classes. One is the neural networks with ridge activation functions. The other is the neural networks with radial-basis activation functions. The results on the ridge activation functions are extensive. It is proven that if the ridge function is continuous almost everywhere, locally essentially bounded and not a polynomial, then the neural networks with ridge activation functions are universal approximators. On the other hand, the results on the radial-basis activation function are less extensive. The most well known result states that if the radial-basis activation function is continuous almost everywhere, bounded and integrable, and the integral is not zero, then the neural networks with radial-basis activation functions are universal approximators. In this dissertation, we will relax these conditions for RBF networks to be like those for the ridge activation functions.

1.3 Dissertation Organization

This dissertation contains seven chapters. Chapter 1 introduces the problem of pattern classification and universal approximation by neural networks. This chapter also gives the research objective and the dissertation organization. Chapter 2 introduces the problem of pattern classification and the basic approaches for pattern classification. Chapter 3 introduces the three popular nonlinear classification methods: Multi-Layer Perceptrons (MLP), Radial Basis Function Networks (RBFN), and Support Vector Machines (SVM). Then in Chapter 4, a new neural network model is proposed based on the underlying ideas of MLP, RBFN and SVM. This chapter also presents some numerical results. Chapter 5 introduces the problem of universal approximation by neural networks, a literature review, and basic definitions and notation. Chapter 6 presents the relaxed conditions for RBF networks to be universal approximators. This chapter also provides some numerical results to demonstrate the validity of our theoretical findings. Chapter 7 draws conclusions and proposes future research.

Chapter 2

Pattern Classification

This chapter provides a brief introduction to pattern classification. The problem description is given in Section 2.1. The basic approaches for pattern classification are introduced in Section 2.2, which include parametric methods (Section 2.2.1), density-based methods (Section 2.2.2), linear discriminant methods (Section 2.2.3), and non-linear discriminant methods (Section 2.2.4).

2.1 Problem Description

Generally speaking, pattern recognition problems fall into two main categories: supervised classification (discriminant analysis) problems and unsupervised classification (clustering) problems. For the supervised classification, we are given a set of training samples with associated “class labels”. The problem is to design a classifier to predict the class label for any given sample. For the unsupervised classification, data are not labelled and we are to classify the data into groups with features that distinguish one group from another. Our study focuses on supervised classification.

In the supervised classification, each sample is assumed to come from one of the c possible classes Ω_j , $j = 1, 2, \dots, c$. We are given n training samples (\mathbf{x}_i, y_i) , $i = 1, \dots, n$, where $\mathbf{x}_i \in R^d$ represents the data and $y_i \in \{1, 2, \dots, c\}$ the corresponding class labels. The objective is to design a decision function $g(\mathbf{x})$ from these training samples such that $g(\mathbf{x})$ can accurately predict the class label for any given test sample \mathbf{x} .

A classical way to tackle this problem is to use *Bayes' decision rule* [4], which assigns a test sample \mathbf{x} to the class with the maximum posterior probability $p(\Omega_j|\mathbf{x})$, that is,

$$\mathbf{x} \in \Omega_k \text{ if } p(\Omega_k|\mathbf{x}) = \max_{j=1,\dots,c} p(\Omega_j|\mathbf{x})$$

where $p(\Omega_j|\mathbf{x})$ is defined as the conditional probability that the observed object belongs to class Ω_j given the associated sample data, \mathbf{x} . In practice these posterior probabilities are usually unknown. By applying Bayes' theorem, they may be calculated as

$$p(\Omega_j|\mathbf{x}) = \frac{p(\mathbf{x}|\Omega_j)p(\Omega_j)}{p(\mathbf{x})}$$

where $p(\mathbf{x})$ is the probability density that the sample \mathbf{x} occurs, $p(\Omega_j)$ is the prior probability that class Ω_j occurs and $p(\mathbf{x}|\Omega_j)$ is the class-conditional density of a sample \mathbf{x} occurring given that the sample is known to come from class Ω_j . Then the decision rule becomes

$$\mathbf{x} \in \Omega_k \text{ if } p(\mathbf{x}|\Omega_k)p(\Omega_k) \geq p(\mathbf{x}|\Omega_j)p(\Omega_j), \forall j = 1, \dots, c \quad (2.1)$$

Generally, the prior probability $p(\Omega_j)$ and the class-conditional densities $p(\mathbf{x}|\Omega_j)$ are unknown and need to be estimated. Compared with the estimation of the prior probability $p(\Omega_j)$, the estimation of the class-conditional densities $p(\mathbf{x}|\Omega_j)$ is much more difficult.

If we know the functional form of the class-conditional densities $p(\mathbf{x}|\Omega_j)$ (e.g., multivariate Gaussian), but do not know the parameters (e.g., mean and covariance), we face a parametric decision problem. In this case, a common strategy is to replace the unknown parameters in the density functions with some estimated values, resulting in the so-called *Bayes plug-in classifier*. If the form of the class-conditional densities is not known, then we operate in a non-parametric mode. In this case, we may either directly estimate the density for a given sample (e.g., *Parzen windows* and *k-Nearest Neighbor*) or construct a decision boundary based on the training samples. In the latter case, assuming that the decision boundary is linear leads to a linear discriminant

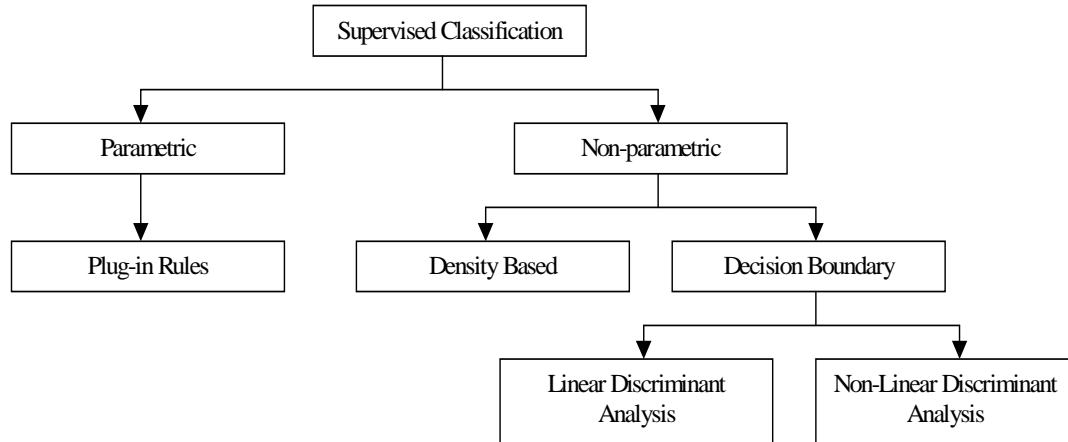


Figure 2-1: Various Approaches in Supervised Classification

method (e.g., *perceptrons* and *Fisher's rule*). Assuming that the decision boundary is not linear leads to a nonlinear discriminant method (e.g., *feedforward neural networks* and *support vector machines*). These basic approaches can be organized into the tree structure as shown in Figure 2-1.

In the next section, we give a brief review of these approaches.

2.2 Basic Approaches

2.2.1 Parametric Methods

If the functional form of the class-conditional densities $p(\mathbf{x}|\Omega_j)$ is known, but the parameters of the density functions are unknown, we may estimate the unknown parameters from the available samples. Without loss of generality, we may drop the class label and write $p(\mathbf{x}|\Omega_j)$ as $p(\mathbf{x}; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is a parameter vector to be estimated. The commonly used parametric models are the multivariate Gaussian distribution for continuous variables, the binomial distribution for binary variables, and the multinormal distribution for integer-valued (or categorical) variables.

Applying the most widely used estimator, namely the maximum likelihood estimator, we select as our estimate the particular value of $\boldsymbol{\theta}$ which gives the greatest

value of the “likelihood”, which is defined by

$$L(\mathbf{x}_1, \dots, \mathbf{x}_n; \boldsymbol{\theta}) = \prod_{i=1}^n p(\mathbf{x}_i; \boldsymbol{\theta}).$$

An important result of using this method is that, when $p(\mathbf{x}; \boldsymbol{\theta})$ is a multivariate Gaussian distribution, if the covariance matrices for different classes are assumed to be different, the optimal classifier provides a quadratic decision boundary. More interestingly, if the covariance matrices are assumed to be identical, then the optimal classifier has a linear decision boundary. The latter result provides a theoretical foundation for the linear discriminant methods.

In general, the parametric approach works well when the number of features (i.e. dimension of \mathbf{x}) is small. Unfortunately, the parametric approach suffers from a phenomenon known as “Curse of Dimensionality”, which states that the number of required samples is an exponential function of the feature dimension. To address this problem, various regularization techniques [72] were proposed to obtain a robust estimate for situations with small sample size and/or high-dimension. Among them, the most famous one is the *Regularized Discriminant Analysis* [37] proposed by Friedman.

In many real world problems, data may form subgroups in each class. In this case, we may approximate the density $p(\mathbf{x}; \boldsymbol{\theta})$ by a *finite mixture model* with

$$\hat{p}(\mathbf{x}; \boldsymbol{\theta}) = \sum_{j=1}^g \pi_j p(\mathbf{x}; \boldsymbol{\eta}_j) \quad (2.2)$$

where g is the number of mixture components, π_j is a mixing proportion ($\pi_j > 0$, $\sum_{j=1}^g \pi_j = 1$), and $p(\mathbf{x}; \boldsymbol{\eta}_j)$ is a component density function which depends on parameters $\boldsymbol{\eta}_j$. There are three sets of parameters to be estimated: π_j , $\boldsymbol{\eta}_j$ and g . Generally, the value of g is the most difficult to estimate. Wolfe [105] proposed a modified likelihood ratio test to estimate the number of components. This approach was later investigated by Everitt [29] and Anderson [2]. Other approaches include Bozdogan [8] and Celeux and Soromenho [13]. Their approaches are based on the information-theoretic criteria such as entropy. If the value of g is already known,

the values of π_j and $\boldsymbol{\eta}_j$ can be estimated by using the *Expectation Maximization* (EM) method [25], which is an extension of the maximum likelihood estimator for the uni-modal case (each class has only one group). Although the EM method is very easy to implement, its convergence rate can be poor, depending on the data distribution and the initial parameters. Jamshidian and Jennrich [55] proposed a generalized conjugate gradient method to speed up the EM method. Lindsay and Basak [64] described a method to initialize the EM model. In addition to EM, several authors also advocated the *Markov Chain Monte-Carlo* method [87], but this method is computationally demanding.

2.2.2 Density-Based Methods

The basic requirement of the parametric approach is the knowledge of the functional form of densities. But this is not generally available in practice. In this case, we must resort to some non-parametric methods. One thought is to directly estimate the class-conditional density $p(\mathbf{x}|\Omega_j)$ for the given data \mathbf{x} , then use the Bayes' decision rule (2.1) to decide the class label. *Parzen windows* and *k-Nearest Neighbor* (k -NN) are the best known methods.

Both *Parzen windows* [81] and k -NN [31, 32, 82] methods are based on one simple idea - the class-conditional density $p(\mathbf{x}|\Omega_j)$ may be approximated by the probability of training samples of class j falling into a small area around \mathbf{x} divided by the volume of that area (for example, if we define the small area as the d -dimensional hypercube whose edges are h units long, the volume of that area is $V = h^d$). Specifically, the estimated class-conditional density $\hat{p}(\mathbf{x}|\Omega_j)$ is

$$\hat{p}(\mathbf{x}|\Omega_j) = \frac{k_j/n_j}{V} \quad (2.3)$$

where V is the volume of the small area around \mathbf{x} , k_j is the number of training samples of class j falling in the small area, and n_j is the number of training samples of class j . It can be shown that if we have infinite number of training samples of class j , as the volume of the small area approaches zero, the density estimate $\hat{p}(\mathbf{x}|\Omega_j)$ in equation

(2.3) asymptotically converges to the true density $p(\mathbf{x}|\Omega_j)$.

For the *Parzen windows* method, we let D be a d -dimensional hypercube, whose edges are h units long (h is called the smoothing parameter), and define a window function

$$\kappa(\mathbf{z}) = \begin{cases} 1 & z_i \leq 0.5, \ i = 1, \dots, d \\ 0 & \text{otherwise} \end{cases} \quad (2.4)$$

where $\mathbf{z} \in R^d$. Then, $\kappa\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)$ is 1, if \mathbf{x}_i falls in the hypercube D centered at \mathbf{x} , and 0 otherwise. Thus, the number of training samples of class j falling into the hypercube D is

$$k_j = \sum_{i=1}^{n_j} \kappa\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) \quad (2.5)$$

Substituting equation (2.5) into equation (2.3), we obtain the density estimate

$$\hat{p}(\mathbf{x}|\Omega_j) = \frac{1}{n_j V} \sum_{i=1}^{n_j} \kappa\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) \quad (2.6)$$

where $V = h^d$ is the volume of the hypercube. Notice that in contrast to the general density functions, because of the window function (2.4), this estimate is not continuous. To resolve this problem, continuous window functions are usually used. It can be easily proven that if $\kappa(\cdot)$ satisfies $\kappa(\mathbf{z}) \geq 0$ and $\int_{-\infty}^{+\infty} \kappa(\mathbf{z}) d\mathbf{z} = 1$, then $\hat{p}(\mathbf{x}|\Omega_j)$ also satisfies the necessary conditions for being a probability density function. Examples of the commonly used window functions include the Gaussian function $\kappa(z) = (2\pi)^{-d/2} \exp(-z^T z/2)$ and the Bartlett-Epanechnikov function $\kappa(z) = (1 - z^T z)(d+2)/(2c_d)$ for $|z| < 1$ (0 otherwise) where $c_d = \pi^{d/2}/\Gamma((d/2)+1)$, $\Gamma(\cdot)$ is the Gamma function.

One important issue with the *Parzen windows* method is the choice of the smoothing parameter h . If h is too big, $\hat{p}(\mathbf{x}|\Omega_j)$ will not reflect the curvature of the true density. If h is too small, $\hat{p}(\mathbf{x}|\Omega_j)$ will be zero except in the very small neighborhood around each sample. A general rule is that h should be small in high density areas and be large in low density areas. Surveys of various means by which to choose h can be found in the articles of Jones, Marron and Sheather [58] and Marron [67].

In the *Parzen windows* method, we use equation (2.3) to estimate the density with a fixed volume V . In the *k-Nearest Neighbor* method, we also use (2.3) to estimate the density, but we fix probability, k_j/n_j (or, equivalently, fix k_j for a given number of samples n_j), and would like to determine the volume V which contains k_j samples of class j centered at the point \mathbf{x} . Unfortunately, in contrast to the *Parzen windows* method, this density estimate is not, in fact, a density function. Nevertheless, this does not deter the wide use of the k -NN method for data classification because the k -NN method leads to a very simple and intuitive classification rule.

Suppose among the k nearest neighbors of \mathbf{x} , there are k_j samples in class Ω_j (so that $\sum_{j=1}^c k_j = k$). Let the total number of samples in class Ω_j be n_j (so that $\sum_{j=1}^c n_j = n$). We may estimate the class-conditional density $p(\mathbf{x}|\Omega_j)$, as $\hat{p}(\mathbf{x}|\Omega_j) = \frac{k_j}{n_j V}$, and the prior probability $p(\Omega_j)$, as $\hat{p}(\Omega_j) = \frac{n_j}{n}$. Then substituting them into the Bayes' decision rule (2.1), we obtain

$$\mathbf{x} \in \Omega_i \quad \text{if } k_i \geq k_j, \forall j = 1, \dots, c$$

Thus, the decision rule is to assign \mathbf{x} to the class which has the majority of the k nearest neighbors of \mathbf{x} .

Two big issues with the k -NN method are the choice of k and the choice of the distance metric. Often in practice, k is chosen empirically and the Euclidean distance is used. But as it turns out, other distance measures which incorporate prior knowledge can significantly enhance the classification performance [93].

One of the biggest disadvantages for both the *Parzen windows* and the k -NN methods is that all training samples need to be retained in order to classify any given data. For large data sets, this leads to huge memory requirement and slow computation speed. To resolve this problem, several data reduction techniques, such as editing [43] and condensing [44], have been proposed. A branch and bound method [39] was also proposed to speed up the k -NN method. For the *Parzen windows* method, Silverman [94] proposed to use a Fourier transform to speed up the computation of kernel functions for univariate density estimates. In the multivariate case, procedures for

approximating the kernel density using a reduced number of kernels were described by Fukunaga and Hayes [40] and Babich and Camps [3].

2.2.3 Linear Discriminant Methods

With the linear discriminant approach, one tries to find some linear boundary (hyperplanes) to separate different classes. Although simple in nature, this approach often does well in practice. Furthermore, it forms the basis for various nonlinear discriminant methods.

In linear discriminant analysis, for the two-class classification problem, we seek a weight vector $\mathbf{w} \in R^d$ and a bias $b \in R$ such that

$$\mathbf{w}^T \mathbf{x} + b \begin{cases} > 0 \\ < 0 \end{cases} \Rightarrow \mathbf{x} \in \begin{cases} \Omega_1 \\ \Omega_2 \end{cases}$$

In the c -class case ($c > 2$), a data point \mathbf{x} will be assigned to the class k that satisfies

$$g_k(\mathbf{x}) = \max_j g_j(\mathbf{x})$$

where $g_j(\mathbf{x}) = \mathbf{w}_j^T \mathbf{x} + b_j$, $\mathbf{w}_j \in R^d$ and $b_j \in R$, $j = 1, \dots, c$. Compared with the *Parzen windows* and k -NN methods, this approach has the advantages of small memory requirement and fast prediction speed, since only a few parameters (\mathbf{w}_j , b_j) need to be used in prediction.

A common feature of various existing linear discriminant methods is that the parameters (the weights and the biases) are found by optimizing some criterion. An obvious criterion is to minimize the number of classification errors which occur when using the training samples. Unfortunately, this criterion is difficult to handle because the number of errors is an integer. A simple extension is the *perceptron criterion* [88] which minimizes the total of the distances of the misclassified samples from the decision boundary. For the two-class classification problem, the *perceptron criterion*

is

$$\min J_p(\mathbf{w}, b) = \sum_{\mathbf{x} \in M} |\mathbf{w}^T \mathbf{x} + b|$$

where $\mathbf{w} \in R^d$ is a weight vector, $b \in R$ is a bias, M is the set of misclassified samples, and $|\cdot|$ is the absolute value. The *Perceptron* has a very simple error-correction algorithm [79] for training. This training procedure cycles through the training samples, modifying the weight vector whenever a sample is misclassified, that is

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \mathbf{x}_i$$

where \mathbf{x}_i is a training sample that has been misclassified by weights $\mathbf{w}(k)$, η is the step size, and k is the iteration number. It can be proven that as long as the training samples are linearly separable (any pair of classes can be separated by a hyperplane), the error-correction algorithm converges to a solution in a finite number of iterations [79]. Nevertheless, even though the resulting hyperplane is able to perfectly separate training data, it cannot guarantee the quality of prediction on (additional) testing samples. Therefore, a separating hyperplane with a certain “margin” [27] relating to the boundary points was proposed to ease this problem (This “margin” idea was also used in developing *support vector machines* [7]). If the training samples are not linearly separable, the error correction algorithm does not converge. Various methods [27] have been proposed to handle this problem, including “early stop”, relaxation, linear programming approaches. The Perceptron can be generalized to handle the c -class classification problem by using “Kesler’s Construction” [79].

The *least squared error* is another popular criterion. In this case, it is assumed that each sample \mathbf{x}_i has a target value t_i . For example, $t_i = 1$ if $\mathbf{x}_i \in \Omega_1$, $t_i = 0$ if $\mathbf{x}_i \in \Omega_2$. The *least squared error* approach aims to minimize the total of the squared distances between the target values and the decision function outputs. For the two-class classification problem, the *least squared error* criterion is

$$\min J_{lse}(\mathbf{w}, b) = \sum_{i=1}^n ||t_i - (\mathbf{w}^T \mathbf{x}_i + b)||^2$$

This criterion has a very nice property [27]. If we set the target value in such a way that $t_i = 1$ if $\mathbf{x}_i \in \Omega_1$, $t_i = 0$ if $\mathbf{x}_i \in \Omega_2$, as the number of training samples increases to infinity, the output value of the decision function, $g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$, asymptotically converges to the difference between the posterior probabilities, $p(\Omega_1|\mathbf{x}) - p(\Omega_2|\mathbf{x})$. Therefore, the approximate values of posterior probabilities can be computed from $p(\Omega_1|\mathbf{x}) \approx (1 + g(\mathbf{x})) / 2$ and $p(\Omega_2|\mathbf{x}) \approx (1 - g(\mathbf{x})) / 2$. Unlike the *perceptrons*, the *least squared error* rule does not necessarily produce a separable solution, even when the classes are linearly separable. Modifications of the *least squared error* rule have been proposed (e.g., the Ho-Kashyap procedure [50]) to remedy this problem, but the optimal approximation to the posterior probability is no longer achieved. Another interesting property of the *least squared error* rule is that with a proper choice of the target value t_i , the rule leads to the same result as *Fisher's rule* [30, 27], for which the basic idea is to find the direction along which two classes are best separated. Furthermore, the *least squared error* rule can be easily extended to the multi-class case. As we will see in Chapter 2, the *least squared error* rule is the most popular criterion used for feed-forward neural networks.

For some difficult problems, a nonlinear decision boundary is required in order to obtain a good separation of different classes. One important generalization of the linear discriminant analysis is the *generalized linear discriminant function*. The basic idea is to nonlinearly map the data from the original d -dimensional space to some \hat{d} -dimensional space, then apply linear discrimination in the mapped space. For the c -class classification problem, the discriminant functions become

$$g_j(\mathbf{x}) = \sum_{i=1}^{\hat{d}} w_{ij} \phi_i(\mathbf{x}) + b_j, \quad j = 1, \dots, c \quad (2.7)$$

where $w_{ij} \in R$ is a weight, $b_j \in R$ is a bias, and $\phi_i(\mathbf{x}) : R^d \rightarrow R^{\hat{d}}$ is a nonlinear mapping that often called the “basis function”. Hopefully, the transformed samples $\phi(\mathbf{x})$ are more linearly separable in the mapped space than the samples \mathbf{x} were in the original space. Specht [96] and Flaszinski and Lewicki [33] proposed the use of polynomial functions as the basis functions. But the problem with polynomial functions

is that the number of terms increases rapidly with the order of the polynomial.

2.2.4 Nonlinear Discriminant Methods

For the nonlinear discriminant methods, the discriminant model is further generalized by assuming that the mapping function ϕ itself has some parameters, and these parameters need to be determined from the training process. Specifically, we assume a discriminant function of the form

$$g_j(\mathbf{x}) = \sum_{i=1}^{\hat{d}} w_{ij} \phi_i(\mathbf{x}; \mathbf{u}_i) + b_j, \quad j = 1, \dots, c \quad (2.8)$$

where there are \hat{d} basis functions ϕ_i , the i th of which has a parameter vector \mathbf{u}_i . As we can see, equation (2.8) is of the same form as the generalized linear discriminant functions (2.7), but we allow more flexibility (parameters \mathbf{u}_i) in the nonlinear function ϕ_i .

Various nonlinear discriminant methods have been proposed. They can be divided into three categories: *feedforward neural networks*, *support vector machines*, and statistical methods. Since Chapter 2 is devoted to the *feedforward neural networks* and *support vector machines*, we (briefly) discuss only the statistical methods.

The *projection pursuit* method was originally proposed by Kruskal [61] and developed by Friedman and Tukey [35] as a technique for finding “interesting” projections of some given data. Its discriminant functions can be expressed as the sums of nonlinear univariate functions of projections. Interestingly, under some mild conditions, the *multi-layer perceptrons* are shown to be a special case of the *projection pursuit* method [54]. In *alternating conditional expectations* [9], discrimination is rephrased in a regression framework using scaling. It is an attractive method which can readily handle variables of mixed type (continuous and discrete). Another method which can handle mixed type variables is the *multivariate adaptive regression spline* [38], whose discriminant function is the sum of products of univariate linear spline functions. The *hinging hyperplanes* method which uses a linear combinations of “hinge” functions as

the discriminant functions was proposed [10]. Interestingly, although all these methods have been reported to have similar or even better performance than *feedforward neural networks*, there are far fewer real-world applications of these statistical methods than of *feedforward neural networks*. Part of the reason is that the *feedforward neural networks* are much easier to understand and to implement.

Chapter 3

Neural Networks and Support Vector Machines

Multi-layer perceptrons and *radial basis function networks* are two major types of feedforward neural networks. They are both powerful methods for nonlinear discriminant analysis. The *support vector machines* method is a relatively new method for nonlinear discriminant analysis. Practical experience has shown that it has similar or even better performance than the *multi-layer perceptrons* and *radial basis function networks*. In this chapter, the basic *multi-layer perceptron* model is introduced in Section 3.1; the *radial basis function networks* model is introduced in Section 3.2; and the *support vector machines* method is introduced in Section 3.3.

3.1 Multi-Layer Perceptrons

3.1.1 Introduction

Regular *perceptrons* cannot perform nonlinear classification [71]. *Multi-Layer Perceptrons* (MLP) were proposed by Rumelhart, Hinton and Williams [90] in 1986 as a potential solution. The basic MLP model uses a nonlinear transformation to map a pattern \mathbf{x} in the d -dimensional space R^d to a c -dimensional point $g(\mathbf{x}) =$

$(g_1(\mathbf{x}), \dots, g_c(\mathbf{x}))^T$ with

$$g_j(\mathbf{x}) = \sum_{k=1}^m w_{jk} \phi_k \left(\sum_{i=1}^d w_{ki} x_i + w_{k0} \right) + w_{j0}, \quad j = 1, \dots, c \quad (3.1)$$

where m is a constant set by the user, w_{jk} and w_{ki} are weights, w_{j0} and w_{k0} are biases. The weights and biases are to be determined by the training procedure. The nonlinear functions ϕ_k (also called activation functions) are usually taken to be identical for all k and of the form of the logistic function,

$$\phi_i(z) = \phi(z) = \frac{1}{1 + \exp(-az)} \quad (3.2)$$

where $z \in R$ and $a > 0$ is a constant called the “gain” parameter. Often in practice, the number of outputs, c , is taken as the number of classes, with each output corresponding to one class. A test sample is assigned to the class with the largest output value. Therefore, the MLP model (3.1) can be regarded as a generalized linear discriminant function (2.7).

The MLP model is often presented in a diagrammatic form as shown in Figure 3-1. The input nodes accept the data sample. There are weights associated with the links between the input nodes and the hidden nodes. The hidden nodes accept a weighted combination $z = \sum_{i=1}^d w_{ki} x_i + w_{k0}$, and perform the nonlinear transformation $\phi(z)$. The output nodes then take a linear combination of the outputs $\phi(z)$ of the hidden nodes and deliver outputs typically using a linear transformation function. In principle, there could be several hidden layers, each one performing a transformation. Also, instead of the linear transformations, nonlinear transformations may be performed at the output nodes.

3.1.2 Model Specification

Model Structure

To construct a MLP model, first we need to specify the network structure, which includes the number of hidden layers, the number of hidden nodes within each layer,

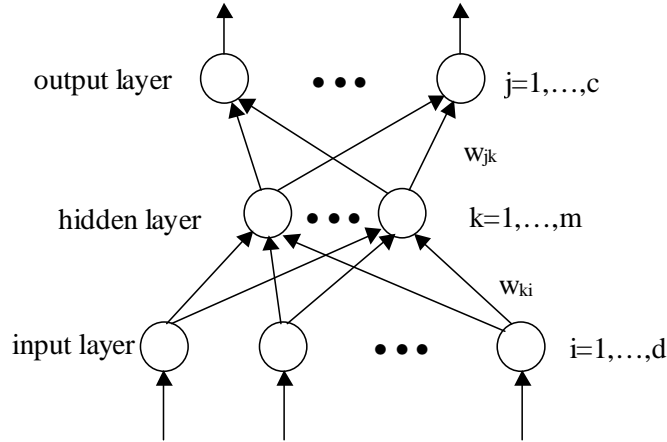


Figure 3-1: Multilayer Perceptron

and the form of the activation functions. It has been shown that the MLP model with a single hidden layer and nonlinear activation functions at hidden nodes is a universal approximator. It can approximate any (continuous bounded integrable) function with arbitrary accuracy [53]. Also, practical experience has shown that very good results can be achieved by using a single hidden layer network. On other hand, for pattern classification problems, output nodes with a linear mapping result in similar classification error rate to those with a nonlinear mapping. Therefore, single hidden layer MLPs with linear output nodes are often used for data discrimination. Most MLP models found in literature adopt the logistic function for the activation function on the hidden nodes. Every node is connected to all nodes in the previous layer (full connection) and there is no direct connection between nodes in nonadjacent layers. The choice of the number of hidden nodes will be discussed later.

Model Parameters

After specifying the network structure, we need to learn (i.e., determine) the optimal parameters (weights and biases) of the network from training samples. The basic approach in MLP learning is to start with an untrained network, present a training pattern to the input layer, pass the signals through the network and determine the output values at the output layer. The outputs are then compared to some target

values. Any difference corresponds to an error. The parameters of the network are adjusted to reduce the error.

In practice, the *least squared error* is often used as an optimization criterion. We define the least squared error by

$$J = \frac{1}{2} \sum_{i=1}^n ||\mathbf{t}_i - g(\mathbf{x}_i)||^2$$

where $g(\mathbf{x}_i)$ is the output vector for the input \mathbf{x}_i and $\mathbf{t}_i = (t_{i1}, \dots, t_{ic})^T$ is the corresponding target vector. For discrimination problems, the class labels are usually coded in the way that $t_{ij} = 1$, if \mathbf{x}_i is in class Ω_j , 0 otherwise.

In the MLP model, the weights are optimized by using the backpropagation algorithm. Suppose we have the weights at the k th iteration, $\mathbf{w}(k)$. Then at the $(k+1)$ st iteration, $\mathbf{w}(k)$ is updated to

$$\begin{aligned} \mathbf{w}(k+1) &= \mathbf{w}(k) + \Delta \mathbf{w}(k) \\ \Delta \mathbf{w} &= -\eta \frac{\partial J}{\partial \mathbf{w}} \end{aligned} \tag{3.3}$$

where $\eta > 0$ is a “learning rate”. Notice that the biases w_{j0} and w_{k0} may be regarded as the special weights with inputs taken to be 1, so the updating rule (3.3) applies to the bias parameters too. For ease of exposition, we denote the output of a node with “ y ” and denote the input of a node with “ z ”..

Consider first the hidden-to-output link weights, w_{jk} , where j is the index of the output node, k is the index of the hidden node. Because the error does not explicitly depend upon w_{jk} , we need to use the chain rule for differentiation, so that

$$\begin{aligned} \Delta w_{jk} &= -\eta \frac{\partial J}{\partial w_{jk}} \\ &= -\eta \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial z_j} \frac{\partial z_j}{\partial w_{jk}} \\ &= \eta (t_j - g_j(\mathbf{x})) y_k \end{aligned} \tag{3.4}$$

where $y_k = \phi(z_k)$ is the output of the hidden node k , and $z_j = \sum_{k=1}^m y_k w_{jk} + w_{j0}$ is

the input to the output node j .

Now consider the hidden-to-input link weights, w_{ki} , where k is the index of the hidden node, i is the index of the input node. We have

$$\Delta w_{ki} = -\eta \frac{\partial J}{\partial y_k} \frac{\partial y_k}{\partial z_k} \frac{\partial z_k}{\partial w_{ki}} \quad (3.5)$$

where $z_k = \sum_{i=1}^d x_i w_{ki} + w_{k0}$ is the input of the hidden node k , and $y_k = \phi(z_k)$ is the output of the hidden node k . The first term on the right-hand side of (3.5) involves all of the weights w_{jk} .

$$\begin{aligned} \frac{\partial J}{\partial y_k} &= \frac{\partial}{\partial y_k} \left[\frac{1}{2} \sum_{j=1}^c (t_j - y_j)^2 \right] \\ &= - \sum_{j=1}^c (t_j - y_j) \frac{\partial y_j}{\partial y_k} \\ &= \sum_{j=1}^c (t_j - y_j) w_{jk} \end{aligned}$$

where y_j is the output of the output node j . Plugging this term into (3.5), we get

$$\Delta w_{ki} = -\eta \left[\sum_{j=1}^c (t_j - y_j) w_{jk} \right] \phi'_k(z_k) \mathbf{x}_i \quad (3.6)$$

where $\phi'_k(z_k)$ is the derivative of activation function ϕ at node k with respect to its input, z_k . Note that for the logistic function (3.2), $\phi'(z) = a\phi(z)(1 - \phi(z))$; therefore, equation (3.6) can be easily evaluated.

Given (3.4) and (3.6), the basic backpropagation algorithm is as follows:

Step 1: initialize the weights w_{jk} , $j = 1, \dots, c$, $k = 1, \dots, m$ and w_{ki} , $k = 1, \dots, m$, $i = 1, \dots, d$, and biases w_{j0} and w_{k0} ;

Step 2: randomly choose a training sample and input it into the MLP model;

Step 3: update weights and biases using $w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$, for $j = 1, \dots, c$, $k = 1, \dots, m$, with Δw_{jk} given by 3.4, and using $w_{ki} \leftarrow w_{ki} + \Delta w_{ki}$, for

$k = 1, \dots, m, i = 1, \dots, d$, with Δw_{jk} given by (3.6);

Step 4: if stopping criteria are met, stop; otherwise, go to step 2.

Usually, the stopping criteria are a maximum number of iterations and/or some threshold value of weight changes. Lots of variants of the algorithm, such as batch backpropagation, updating with momentum and updating with the second order information, have been studied. More detailed information can be found in Bishop [5] and Wu [106].

Regularization and Pruning

Whereas the MLP model can provide an excellent nonlinear classification capability, overfitting may occur when the model has too many parameters (weights). On the other hand, if it has too few weights, the nonlinear behavior in the training samples may not be well learned. Therefore, the number of weights needs to be controlled.

A commonly used approach to help control the number of weights is to select the proper number of hidden nodes using empirical experiments: (i) a sequence of MLP models with an increasing number of hidden nodes is trained and tested, (ii) as each model is tested, the test error rate is plotted as a function of the number of hidden nodes, and (iii) the process stops with the number of hidden nodes when the test error rate starts to rise.

Another general approach to help control the number of weights is called regularization. The basic idea is to use an optimization criterion that depends not only on the classical training error (least squared error), but also on the classifier complexity. Formally, we can write the criterion as the sum of the empirical training error J_{emp} plus a regularization term J_{reg} , i.e.,

$$J = J_{emp} + \lambda J_{reg}$$

where parameter $\lambda \geq 0$ is adjusted to control the effect of the regularization term.

For the regularization approach, one commonly used method is the “weight decay” [49], which is based on the idea that the weights of large magnitude have more

potential to lead to overfitting than the weights of small magnitude. Therefore, the weights of large magnitude are penalized. There is no theoretical foundation for such a method, but it is found to work in most cases. The basic approach is to start with a network with many weights and “decay” all weights during training, i.e., after each update, new weights decay according to

$$\mathbf{w}^{new} = \mathbf{w}^{old} (1 - \varepsilon)$$

where $0 < \varepsilon < 1$. In this way, weights that do not contribute to the reduction of the criterion function become smaller and smaller, possibly so small that the associated arc can be eliminated altogether. It can be shown that the weight decay is effectively equivalent to gradient descent applied to a new criterion function

$$J_{eff} = J_{emp} + \frac{2\varepsilon}{\eta} \mathbf{w}^T \mathbf{w}$$

The second term on the right-hand side is a regularization term that preferentially penalizes the large weights.

Another popular approach to control the number of weights is to prune weights that are “least needed”. While it is natural to imagine that after training the weights with the smallest magnitude can be eliminated, sometimes weights with small magnitudes are important for learning. One way of pruning is to use the Optimal Brain Damage [23] algorithm or its descendent, Optimal Brain Surgeon [45, 46]. These algorithms use a second-order approximation to predict how the training error depends upon each weight, and eliminate the weight that leads to the smallest increase in the training error. Because they use the second-order information, these algorithms are computationally expensive.

3.1.3 Discussion

MLP models are very flexible. They perform well on a wide range of problems in pattern recognition. This approach is easy to understand and easy to implement.

But since MLP is a nonlinear model, nonlinear optimization strategies need to be adopted. Furthermore, the objective function of a MLP model is not convex with respect to its parameters, the backpropagation algorithms often get stuck at a local optimum. Often in practice, to find a good solution, the backpropagation training needs to be run several times each with different initial weights. This further increases the computational load and is often considered as the major drawback of the MLP model.

3.2 Radial Basis Function Networks

3.2.1 Introduction

Radial Basis Function (RBF) networks were originally proposed for the exact interpolation problem by Powell [83, 66] and used for discrimination by Broomhead and Lowe [11], and Moody and Darken [73]. Compared with the MLP model, the RBF network usually has much faster training speed, while maintaining the nonlinear classification capability. Therefore, it has gained popularity since it was originally proposed.

The basic RBF network provides a nonlinear transformation of a pattern $\mathbf{x} \in R^d$ to R^c according to

$$g_j(\mathbf{x}) = \sum_{i=1}^m w_{ji} \phi\left(\frac{|\mathbf{x} - \boldsymbol{\mu}_i|}{h}\right) + b_j, \quad j = 1, \dots, c \quad (3.7)$$

where $\phi(\cdot)$ is a radially symmetric basis function (also called the kernel function), m is the number of basis functions, w_{ji} is a weight, b_j is a bias, $\boldsymbol{\mu}_i \in R^d$ is called the center vector, and $h \in R$ is called the kernel width (or smoothing parameter). (3.7) almost has the same mathematical form as the MLP model (3.1). The major difference is that the logistic activation function is replaced by the radial basis function, whose value is usually the largest at its center, $\boldsymbol{\mu}_i$, and decreases as \mathbf{x} moves away from the center (One example is the Gaussian function $\phi(\mathbf{x}) = \exp\left(-\frac{|\mathbf{x} - \boldsymbol{\mu}|}{h}\right)$). Therefore, the RBF network may be simply viewed as putting a small “bump” at each center, and adding the bumps together to form the classification boundary.

3.2.2 Motivation

Radial basis functions have been in existence in one form or another for a very long period of time. They are very closely related to *Parzen windows* for density estimation and *finite mixture models* for parametric discrimination. If bias b_j is set to be zero and the number of centers m is taken to be equal to the number of samples in the class j , with $\boldsymbol{\mu}_i = \mathbf{x}_i$ (one center at each data sample), the RBF model (3.7) has the same form as the *Parzen window* density model (2.6). If the number of centers m is set to be the number of mixture components and bias b_j is set to be zero, the RBF model is equivalent to the *finite mixture model* (2.2).

The RBF model may be also motivated from several other perspectives. Webb [102] demonstrates its connection with the “noise on variables” model, in which an approximation $g(\cdot)$ to a known function $f(\cdot)$ is derived when the input variables are noisy. Green and Silberman [42] and Bishop [5] show the derivation from the regularization perspective, in which case the noise is assumed to be in the target values.

The RBF model was originally proposed for solving the exact interpolation problem, in which we are given n data points $\mathbf{x}_i \in R^d, i = 1, \dots, n$, with corresponding target values $t_i \in R, i = 1, \dots, n$, and we seek a mapping $g : R^d \rightarrow R$ which satisfies the condition $g(\mathbf{x}_i) = t_i, i = 1, \dots, n$. The RBF model solves this problem by positioning a center $\boldsymbol{\mu}_i$ at each data point \mathbf{x}_i , ignoring the bias b , and seeking a solution $\mathbf{w} \in R^n$ which satisfies

$$\mathbf{t} = \boldsymbol{\phi} \mathbf{w}$$

where $\boldsymbol{\phi}$ is the $n \times n$ matrix with $\phi(|\mathbf{x}_i - \mathbf{x}_j|/h)$ as its (i, j) th element. For a large class of functions Micchelli [69] has shown that the inverse of $\boldsymbol{\phi}$ exists and the solution for \mathbf{w} is given by

$$\mathbf{w} = \boldsymbol{\phi}^{-1} \mathbf{t}$$

In general, exact interpolation is not good for discrimination. The fitted function can be highly oscillatory. It leads to poor generalization performance in many pattern classification problems. A common practice is to limit the degrees of freedom of the

RBF model by taking $m < n$.

In the next section, we introduce the basic RBF model for discrimination.

3.2.3 Model Specification

When used as a discrimination tool, the basic RBF model is of the form (3.7), while the number of the basis functions, m , is often taken to be far less than the number of data points, n . Like the MLP model, the number of outputs, c , is usually taken as the number of classes.

There are three main stages in constructing an RBF model:

Stage 1. Specify the kernel function ϕ ;

Stage 2. Determine the number and positions of the centers;

Stage 3. Determine the weights and the biases.

In the following, we consider each of these stages.

Functional Forms

To a large extent, specifying the functional form of the kernel function is independent of the data and the problem, although the parameters of these functions are not. Even though a certain type of problem may end up matched inappropriately to a certain form of kernel function, the actual form of the kernel function is relatively unimportant compared to the number and the positions of the centers. The typical forms of kernel functions are listed in table 3.1.

Table 3.1: Commonly Used RBFs	
Name	Mathematical form of $\phi(z)$, $z = \mathbf{x} - \boldsymbol{\mu} /h$
Gaussian	$\exp(-z^2)$
Exponential	$\exp(-z)$
Quadratic	$z^2 + \alpha z + \beta$
Inverse quadratic	$1/(1 + z^2)$
Thin-plate spline	$z^\alpha \log(z)$

The two most popular forms are the Gaussian, $\phi(z) = \exp(-z^2)$, and the thin-plate spline, $\phi(z) = z^2 \log(z)$. The Gaussian is more suitable for discrimination and

density estimation, while the thin-plate spline is more suitable for curve fitting [66].

Centers and Width

The selection of the number of centers for the RBF model is in principle no different from the selection of the number of hidden nodes for the MLP model. Therefore, the various approaches applicable for the MLP model can be also applied to the RBF model. With a specified number of centers, the positions of centers may be found by one of the following approaches:

1. Randomly select from the data set: Random selection is an approach that is commonly used. An advantage is that it is fast. A disadvantage is that it may not provide the best solution for a mapping problem.
2. Clustering approach: The positions of the centers are obtained by clustering algorithms such as k -means [77, 59], agglomerative clustering [77], and SOM [60]. Clustering offers the advantage that the centers need not to be training samples, thus providing more flexibility for the RBF model.
3. Gaussian mixture model: If we use Gaussian as the kernel function, then it seems sensible to use centers resulting from a Gaussian mixture model for the underlying distribution. In this case, these parameters may be determined by using the Expectation Maximization algorithm [25] to maximize the likelihood.
4. Orthogonal least squares: Chen, Cowan and Grant [16, 17] construct a set of centers incrementally, considering the complete set of data samples to be candidates for centers. Suppose that we have a set of $k - 1$ centers, positioned over $k - 1$ different data points. At the k th stage, that point among the remaining $n - (k - 1)$ data samples that reduces the prediction error the most is added to the set of centers. This approach was further developed in [15, 80].

Weights and Biases

The last stage in determining the RBF model is the calculation of the weights and biases, which is relatively easy. Usually, it is carried out by optimizing some criterion, the most common of which is the least squared error, J . Specifically, J is minimized with respect to the weights $\mathbf{W} \in R^{c \times m}$ and bias $\mathbf{b} \in R^c$. J is defined as

$$\begin{aligned} J &= \sum_{i=1}^n \|\mathbf{t}_i - (\mathbf{W}\phi(\mathbf{x}_i) + \mathbf{b})\|^2 \\ &= \|\mathbf{T}^T - \mathbf{W}\phi^T - \mathbf{b}\mathbf{e}^T\|^2 \end{aligned}$$

where $\mathbf{T} = [\mathbf{t}_1, \dots, \mathbf{t}_n]^T$ is an $n \times c$ target matrix whose i th row is the target for input \mathbf{x}_i . $\phi = [\phi(\mathbf{x}_1), \dots, \phi(\mathbf{x}_n)]^T$ is an $n \times m$ matrix whose i th row is the set of kernel function values at \mathbf{x}_i , and \mathbf{e} is an $n \times 1$ vector of 1's.

The solution for \mathbf{W} is

$$\mathbf{W} = \hat{\mathbf{T}}^T (\hat{\phi}^T)^\dagger$$

where \dagger denotes the pseudo-inverse of a matrix. The matrices $\hat{\mathbf{T}}$ and $\hat{\phi}$ are defined as

$$\begin{aligned} \hat{\mathbf{T}} &= \mathbf{T} - \mathbf{e}\bar{\mathbf{t}}^T \\ \hat{\phi} &= \phi - \mathbf{e}\bar{\phi}^T \end{aligned}$$

where

$$\begin{aligned} \bar{\mathbf{t}} &= \frac{1}{n} \sum_{i=1}^n \mathbf{t}_i = \frac{1}{n} \mathbf{T}^T \mathbf{e} \\ \text{and } \bar{\phi} &= \frac{1}{n} \sum_{i=1}^n \phi(\mathbf{x}_i) = \frac{1}{n} \phi^T \mathbf{e} \end{aligned}$$

are the mean values of the targets and kernel function outputs, respectively. Therefore, we may get the weights \mathbf{W} by using a linear method such as the *singular value decomposition* [95]. In case that the matrix dimension of ϕ is too big (or there are too many training samples), we may resort to some iterative methods such as the *Widrow-Hoff* rule [104].

The solution for \mathbf{b} is

$$\mathbf{b} = \bar{\mathbf{t}} - \mathbf{W}\bar{\phi}$$

3.2.4 Discussion

Like the MLP network, the RBF network is also a universal approximator [84, 85, 19]. But in contrast to the MLP model, no nonlinear optimization scheme is required in the RBF training procedure. The RBF method utilizes some existing pattern recognition techniques (such as clustering and the Gaussian mixture models) to determine the centers, and then a set of linear equations is solved to find the weights and biases. Therefore, the required training time is shorter than that with the MLP model. In practice, the RBF model often requires more hidden nodes (the centers) than the MLP model. Therefore, it demands more memory and has a slower prediction speed than the MLP model. It has been claimed that the RBF model is more suitable for the midsize problem (not too many training samples), while the MLP model is more suitable for the large size problem [65].

3.3 Support Vector Machines

3.3.1 Introduction

Support vector machines (SVM) is a relatively new approach for pattern classification. The study of this approach started in the late seventies [99], but only in recent years has it received much attention. The SVM method is based on statistical learning theory [100, 101] and motivated by two well-known and simple ideas: classification with margins and mapping data to a higher dimension. The SVM method has been successfully applied to many practical pattern recognition problems, such as handwritten digit recognition [22, 12], object recognition [6], text categorization [56], etc.. In most of these cases, SVM's generalization performance (i.e., error rates on test sets) either matches or is significantly better than that of competing methods. The SVM method is also an important tool for regression analysis [75, 26] and density

estimation [103]. It has been extended to other pattern recognition tasks such as “kernel principle component analysis” [92] and “kernel Fisher’s rule” [70].

The basic idea of SVM is to classify data in a transformed feature space, which usually has much higher dimension than the original one. With an appropriate nonlinear mapping $\phi(\cdot)$ to a sufficiently high dimension, we may expect that data from two classes will be separated by a hyperplane. Although this mapping idea is well-known, rarely has it been put into practice because there are too many free parameters in the higher dimensional space. SVM resolves this difficulty by using some tricks, which will be introduced in Section 3.3.2. Furthermore, by incorporating the “large margin” concept, the SVM model is formulated as a convex quadratic programming problem. Therefore, in contrast to the MLP model, a global optimal solution is theoretically guaranteed. Interestingly, after solving the corresponding quadratic programming problem, the optimal discrimination function is expressed in terms of relatively few training samples, which are called “support vectors”.

In the next section, we give a brief introduction to the basic model of support vector machines.

3.3.2 SVM Model

Basically, the SVM model is a two-class classifier, although some extensions have been proposed for the multi-class case [98, 7]. For a two-class classification problem, we are given n training data $\mathbf{x}_i \in R^d$ together with corresponding labels $y_i \in \{-1, 1\}$, $i = 1, 2, \dots, n$. The goal is to find some decision function $g(\mathbf{x})$ which accurately predicts the label of any given data \mathbf{x} .

Linearly Separable Case

Let’s first consider the simplest case where the training samples are linearly separable, i.e., there exists some hyperplane $g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$, $\mathbf{w} \in R^d$, $b \in R$, which perfectly

separates the training samples for the two classes, that is, for $i = 1, \dots, n$,

$$\begin{cases} \mathbf{w}^T \mathbf{x}_i + b \geq 0 & \text{for } y_i = 1 \\ \mathbf{w}^T \mathbf{x}_i + b \leq 0 & \text{for } y_i = -1. \end{cases} \quad (3.8)$$

Note that the two set of inequalities in (3.8) can be combined into one, namely,

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 0, \quad i = 1, 2, \dots, n. \quad (3.9)$$

If more than one hyperplane satisfies the constraint (3.9), we wish to use the one which has the highest generalization capability (lowest test error rate). In this case, the constraint (3.9) is equivalent to saying that there exists some positive number $\varepsilon > 0$, such that

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq \varepsilon, \quad i = 1, 2, \dots, n. \quad (3.10)$$

Since constraint (3.10) can be multiplied by a positive constant without changing sign, it is equivalent to

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1, \quad i = 1, 2, \dots, n. \quad (3.11)$$

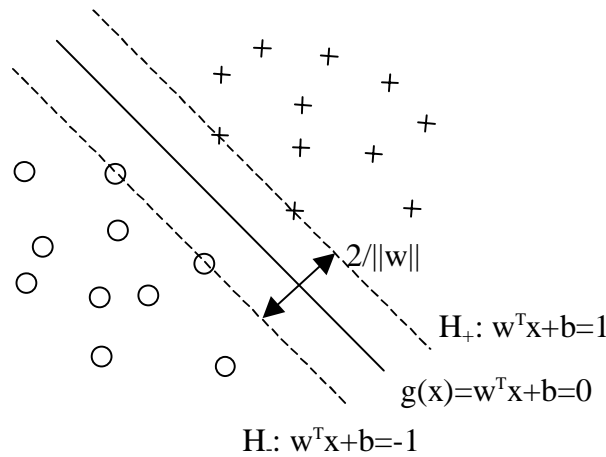


Figure 3-2: SVM - Linearly Separable Data

Let d_+ , d_- be the distance from the separating hyperplane to the closest positive and negative samples, respectively (positive samples are the samples with $y_i = 1$; negative samples are the samples with $y_i = -1$). Define the “margin” of a separating hyperplane to be $(d_+ + d_-)$. For the linearly separable case, the SVM model simply looks for the separating hyperplane with the largest margin.

Notice that the closest positive and negative samples to the separating hyperplane lie on the bounding hyperplanes $H_+ : \mathbf{w}^T \mathbf{x} + b = 1$ and $H_- : \mathbf{w}^T \mathbf{x} + b = -1$, respectively. Therefore, the distances from these samples to the separating hyperplane are $d_+ = d_- = \frac{1}{\|\mathbf{w}\|}$, thus the margin of the separating hyperplane is $\frac{2}{\|\mathbf{w}\|}$ (shown in Figure 3-2). Therefore, we can find the optimal separating hyperplane by minimizing $\|\mathbf{w}\|^2$, subject to the constraint (3.11). This leads to the following convex quadratic programming problem:

$$\begin{aligned} \min_{(\mathbf{w}, b)} \quad & \frac{1}{2} \mathbf{w}^T \mathbf{w} \\ \text{s.t.} \quad & y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \\ & i = 1, 2, \dots, n. \end{aligned} \tag{3.12}$$

In order to get the dual form of the problem (3.12), we introduce Lagrange multipliers $\alpha_i > 0$, $i = 1, \dots, n$, corresponding to each constraint. The Lagrange function of the problem (3.12) becomes

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^n \alpha_i [y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1].$$

Therefore, the KKT conditions for the problem (3.12) are:

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{w}} = 0 & \Rightarrow \mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \\ \frac{\partial L}{\partial b} = 0 & \Rightarrow \sum_{i=1}^n \alpha_i y_i = 0 \\ y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 & \geq 0 \\ \alpha_i [y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1] & = 0 \\ \alpha_i \geq 0, \quad i & = 1, \dots, n. \end{aligned}$$

After rearranging terms and variables, we get the following dual problem:

$$\begin{aligned}
\min_{\alpha} \quad & \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j - \sum_{i=1}^n \alpha_i \\
s.t. \quad & \sum_{i=1}^n y_i \alpha_i = 0 \\
& \alpha_i \geq 0, \quad i = 1, \dots, n.
\end{aligned} \tag{3.13}$$

This is also a convex quadratic programming problem, but with more manageable constraints. Notice that after solving this dual problem, we can immediately determine the primal variable \mathbf{w} by the formula $\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i$. The primal variable b can be easily found by using the KKT complementarity condition, $\alpha_i [y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1] = 0$, with any $\alpha_i > 0$.

Notice that there is a Lagrange multiplier α_i for each training sample \mathbf{x}_i . At the optimal solution, those samples for which $\alpha_i > 0$ are called “support vectors”. The support vectors lie on the bounding hyperplanes H_+ and H_- , that is, $y_i (\mathbf{w}^T \mathbf{x}_i + b) = 1$. All other training samples have $\alpha_i = 0$. Because of the constraint $y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 \geq 0$, they lie either on H_+ or H_- (that is, $y_i (\mathbf{w}^T \mathbf{x}_i + b) = 1$) or on the outside of H_+ or H_- (that is, $y_i (\mathbf{w}^T \mathbf{x}_i + b) > 1$). Since the discrimination function is

$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b,$$

the support vectors are the critical elements of the set of training samples. They lie the closest to the decision boundary. If all other training samples were removed and the training process were repeated, the same separating hyperplane would be found.

Linearly Non-Separable Case

The above algorithm is for linearly separable training samples. When it is applied to linearly non-separable data, the algorithm finds no feasible solution. The SVM model resolves this issue by relaxing the constraints (3.11), but only as necessary. This is done by introducing some slack variables $\xi_i \in R$, $i = 1, \dots, n$, into the constraints and penalizing them in the objective function. This leads to the following convex

programming problem:

$$\begin{aligned}
& \min_{(\mathbf{w}, b, \xi)} \quad \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^n \xi_i \\
& s.t. \quad y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i \\
& \quad \xi_i \geq 0, \quad i = 1, 2, \dots, n,
\end{aligned} \tag{3.14}$$

where $C > 0$ is a penalty parameter to be chosen by the user. A large C corresponds to assigning a higher penalty for errors. The Lagrange function of the problem (3.14) is

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^n \xi_i - \sum_{i=1}^n \alpha_i [y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 + \xi_i] - \sum_{i=1}^n \xi_i \beta_i,$$

where $\alpha_i \geq 0$, $\beta_i \geq 0$, $i = 1, \dots, n$, are Lagrange multipliers. Applying the KKT conditions,

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{w}} = 0 &\Rightarrow \mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \\
\frac{\partial L}{\partial b} = 0 &\Rightarrow \sum_{i=1}^n \alpha_i y_i = 0 \\
\frac{\partial L}{\partial \xi_i} = 0 &\Rightarrow \alpha_i + \beta_i = C \\
y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 + \xi_i &\geq 0 \\
\alpha_i [y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 + \xi_i] &= 0 \\
\xi_i \beta_i &= 0 \\
\xi_i \geq 0, \alpha_i \geq 0, \beta_i \geq 0, \quad i &= 1, \dots, n,
\end{aligned} \tag{3.15}$$

we get the following dual problem:

$$\begin{aligned}
& \min_{\boldsymbol{\alpha}} \quad \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j - \sum_{i=1}^n \alpha_i \\
& s.t. \quad \sum_{i=1}^n y_i \alpha_i = 0 \\
& \quad 0 \leq \alpha_i \leq C, \quad i = 1, \dots, n.
\end{aligned} \tag{3.16}$$

Interestingly, the only difference between (3.16) and the linearly separable dual (3.13) is that each α_i now has an upper bound C . From the KKT conditions (3.15), we get

the following necessary and sufficient conditions for optimality:

$$\begin{cases} \alpha_i = 0 & \Rightarrow y_i g(\mathbf{x}_i) \geq 1 \\ \alpha_i = C & \Rightarrow y_i g(\mathbf{x}_i) \leq -1 \\ 0 < \alpha_i < C & \Rightarrow y_i g(\mathbf{x}_i) = 1, \end{cases} \quad (3.17)$$

where

$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b. \quad (3.18)$$

Notice that the bounding hyperplanes can be written as $H_+ : g(\mathbf{x}) = 1$ and $H_- : g(\mathbf{x}) = -1$. Let's define the data points "outside" the bounding hyperplanes as the data points which satisfy $y_i g(\mathbf{x}_i) > 1$, and the data points "inside" the bounding hyperplanes as the data points which satisfy $y_i g(\mathbf{x}_i) < 1$. At the optimal solution, the data points outside H_+ and H_- have $\alpha_i = 0$, the data points inside H_+ and H_- have $\alpha_i = C$, and data points on H_+ and H_- have $0 \leq \alpha_i \leq C$. The SVM model for linearly nonseparable data is illustrated in Figure 3-3.

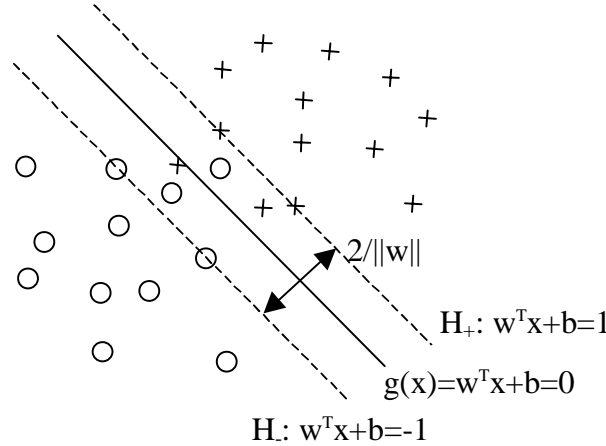


Figure 3-3: SVM - Linearly Non-Separable Data

Nonlinear SVM

The above linear SVM model (3.16) can be easily extended to handle nonlinear classification problems. Suppose we first map the samples to some high dimensional

(possibly infinite) space F , using a mapping $\phi : R^n \rightarrow F$, and apply the linear support vector machine (3.16) to the mapped samples $\phi(\mathbf{x}_i)$. We get the following problem:

$$\begin{aligned} \min \quad & \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) - \sum_{i=1}^n \alpha_i \\ \text{s.t.} \quad & \sum_{i=1}^n y_i \alpha_i = 0 \\ & 0 \leq \alpha_i \leq C, \quad i = 1, \dots, n. \end{aligned}$$

If there is some “kernel function” $\kappa(\mathbf{x}_i, \mathbf{x}_j)$ such that $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$, we can replace $\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ with it (even without explicitly knowing the functional form of the mapping function $\phi(\cdot)$). Therefore, the problem becomes

$$\begin{aligned} \min \quad & \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) - \sum_{i=1}^n \alpha_i \\ \text{s.t.} \quad & \sum_{i=1}^n y_i \alpha_i = 0 \\ & 0 \leq \alpha_i \leq C, \quad i = 1, \dots, n. \end{aligned} \tag{3.19}$$

But what kind of kernel function can be formulated in the form of $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$? The answer is given by Mercer’s condition [68, 1, 7]: there exists a mapping ϕ and an expansion $\kappa(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) \phi(\mathbf{y})$ if and only if, for any $f(\mathbf{x})$ such that $\int f(\mathbf{x})^2 d\mathbf{x}$ is finite, then $\int \int \kappa(\mathbf{x}, \mathbf{y}) f(\mathbf{x}) f(\mathbf{y}) d\mathbf{x} d\mathbf{y} \geq 0$. Some commonly used kernel functions which satisfy Mercer’s condition are listed in Table 3.2.

Table 3.2: Commonly Used Kernel Functions

Name	Mathematical Form
Polynomial Kernel	$\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i^T \mathbf{x}_j + \delta)^q$
Gaussian Kernel	$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\ \mathbf{x}_i - \mathbf{x}_j\ ^2}{2h}\right)$
Sigmoid Kernel	$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma \mathbf{x}_i^T \mathbf{x}_j + \delta)$

An equivalent way to characterize the Mercer kernels is that they give rise to positive semi-definite matrices $\kappa_{ij} := \kappa(\mathbf{x}_i, \mathbf{x}_j)$ for all $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ [91]. Therefore, problem (3.19) is a convex quadratic programming problem and the global optimum is guaranteed.

The optimal decision function resulting from problem (3.19) is

$$\begin{aligned}
g(\mathbf{x}) &= \sum_{i=1}^n \alpha_i y_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x}) - b \\
&= \sum_{i=1}^n \alpha_i y_i \kappa(\mathbf{x}_i, \mathbf{x}) - b.
\end{aligned} \tag{3.20}$$

Therefore, the decision function can also be expressed in the form of kernel functions, rather than the mapping function $\phi(\cdot)$. At the same time, the necessary and sufficient optimality conditions (3.17) still hold; the only difference is that the decision function $g(\mathbf{x})$ is in the form of (3.20), rather than (3.18).

3.3.3 Discussion

The SVM model has a close connection with the MLP and RBF models. When it adopts the logistic (sigmoid) kernel function, the SVM model produces a MLP discrimination function (3.1). Since the SVM model is formulated as a convex programming problem, the “local optimum” difficulty in the MLP model is avoided. On the other hand, when it uses Gaussian kernel function, the SVM model leads to an RBF discrimination function (3.7). But unlike the RBF model, the “centers” and the “weights” are found at the same time. Another advantage of the SVM model is that it has relatively few parameters to adjust. The only parameters are the penalty parameter, C , and the parameters of the kernel functions, such as the smoothing parameter h in the Gaussian kernel function.

One of the major disadvantages of the SVM method is that a convex quadratic programming solver is required for the SVM training. Furthermore, since problem (3.19) has a dense Hessian matrix $\kappa(\mathbf{x}_i, \mathbf{x}_j)$, if there are millions of training samples (which is quite possible in practice), problem (3.19) becomes large and dense, and is difficult to solve even if a commercial software is used. Several training procedures have been proposed for this case. See SMO [86] and SVM-Light [57].

Chapter 4

Neural Networks with Bounded Weights

Based on the basic ideas of the MLP, RBF, and SVM models, a new neural network model is proposed in this chapter. This neural network model adopts the least squared error as the optimization criterion, but limits the magnitudes of the weights to a certain range. Similar to the SVM model, the proposed model is also formulated as a convex quadratic programming problem. However, it does not require the kernel functions to satisfy Mercer's condition, and it can be readily extended for multi-class classification. In this chapter, the motivation for the proposed model is discussed in Section 4.1; the proposed model is introduced in Section 4.2; and experimental results are reported in Section 4.3.

4.1 Motivation

From neural networks' perspective, support vector machines can be regarded as a special type of feedforward neural network. Like other feedforward neural networks, the SVM model also has a layered structure, in which a data vector is fed into the input nodes and nonlinearly transformed at the hidden nodes. Then at the output nodes, the outputs of the hidden nodes are linearly combined to produce the final output. The special characteristics of the SVM model is that it treats each training sample as a potential hidden node; a subset of hidden nodes are selected by solving a convex quadratic programming problem.

Examining the problem formulation (3.19) of the SVM model, we notice that the

bound constraint, $0 \leq \alpha_i \leq C$, $i = 1, \dots, n$, plays an important role in the hidden nodes selection. Since each α_i is bounded below by 0, some α_i become 0 at the optimal solution. Since α_i is the weight of the link connecting the output node and the hidden node i , when α_i is 0, the hidden node i actually makes no contribution to the network output. In other words, the hidden nodes with $\alpha_i = 0$ are pruned after the training; only the hidden nodes with $\alpha_i > 0$ are retained as actual hidden nodes.

In the bound constraint $0 \leq \alpha_i \leq C$, the parameter C limits the magnitudes of the weights that controls the learning capability of the SVM model. A large C allows the SVM model to have more freedom to learn any nonlinearities embedded in the training samples, but with greater chance of overfitting. On the other hand, a small C causes the SVM model to have less learning capability, but with less chance of overfitting. The idea of “using small weights to avoid overfitting” is similar to that used in the “weight decay” method for the MLP model.

Based on the above analysis, we speculate that the concept of “bounded weights” may be applied to neural network models which adopt objective functions different from the one in the SVM model. In this part, we propose a neural network model in which the *least squared error* is adopted as the optimization criterion for weight specification, and the magnitudes of the weights on the links between the hidden nodes and output nodes are limited to a range $[0, C]$.

4.2 Proposed Model

4.2.1 Two-Class Case

Let us first consider the two-class classification problem, in which we are given n training samples consisting of input data vectors $\mathbf{x}_i \in R^d$ together with corresponding labels $y_i \in \{-1, 1\}$, $i = 1, 2, \dots, n$. The goal is to find some decision function $g(\mathbf{x})$ which accurately predicts the label of any given data \mathbf{x} .

For this problem, we propose a three-layer neural network model as shown in Figure 4-1. The network has d input nodes, n potential hidden nodes (each training

sample corresponds to one hidden node), and 1 output node. The data vector $\mathbf{x}_i \in R^d$ is entered at each input node and passed to each of the hidden nodes. At hidden node k the data is fed into the transformation function $z_k = \kappa(\mathbf{x}_k, \mathbf{x})$, where $z_k \in R$ is the output of the hidden node k , \mathbf{x}_k is the k th training sample, and $\kappa(\cdot, \cdot)$ is a nonlinear function (we call it a kernel function). The output node delivers the output as $\sum_{k=1}^n \alpha_k y_k z_k + b$, where $\alpha_k \in R$ is the weight of the link connecting the hidden node k and the output node, y_k is the class label of training sample \mathbf{x}_k , and $b \in R$ is the bias of the output node. As we can see from Figure 4-1, the “actual” weight of the link connecting the hidden node k and the output node is $\alpha_k y_k$, rather than α_k . But since y_k is either -1 or 1 and is known, we simply refer to α_k as the “weight”.

Similar to the basic MLP and RBF models, we adopt the *least squared error* as the optimization criterion for weight specification for the proposed model. Also similar to the SVM model, we limit the magnitudes of the weights α_k , $k = 1, 2, \dots, n$, to the range $[0, C]$, where $C > 0$ is a constant chosen by the user. Since the weights are bounded, we call the proposed model a “Bounded Neural Network” (BNN).

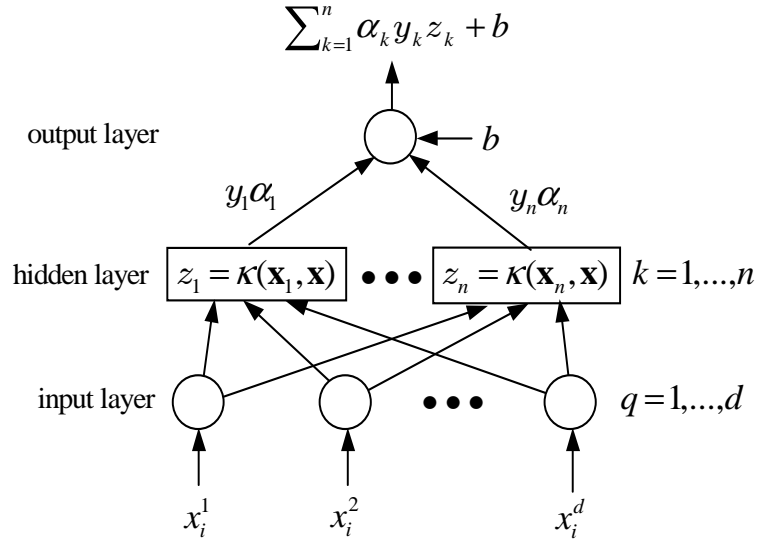


Figure 4-1: Bounded Neural Network: Two-Class Case

Specifically, in the training stage, we try to find the optimal parameters (weights

α_k , $k = 1, \dots, n$, and bias b) by solving the the following problem:

$$\begin{aligned} \min_{(\alpha, b)} \quad & \frac{1}{2} \sum_{i=1}^n \|y_i - (\sum_{k=1}^n \alpha_k y_k \kappa(\mathbf{x}_k, \mathbf{x}_i) + b)\|^2 \\ \text{s.t.} \quad & 0 \leq \alpha_k \leq C, \quad k = 1, 2, \dots, n. \end{aligned} \quad (4.1)$$

For prediction, the decision rule is

$$\text{assign } \mathbf{x} \text{ to } \begin{cases} \text{class 1} \\ \text{class 2} \end{cases} \quad \text{if } \begin{cases} g(\mathbf{x}) > 0 \\ g(\mathbf{x}) < 0, \end{cases}$$

where the discrimination function, $g(\mathbf{x})$, is

$$g(\mathbf{x}) = \sum_{k=1}^n \alpha_k y_k \kappa(\mathbf{x}_k, \mathbf{x}) + b. \quad (4.2)$$

Notice that before training (solving problem (4.1)), the network has n potential hidden nodes, with one hidden node corresponding to each training sample. Since the weight $\alpha_k \in [0, C]$, some weights may become 0 after training. Since a hidden node with 0 weight plays no role in the discrimination function (4.2), such a hidden node is actually pruned after the training. Therefore, the actual number of hidden nodes may be less than n .

While in the SVM model, the kernel functions are required to satisfy Mercer's condition, the BNN model does not have this requirement for the kernel functions. To see this, let us first write the BNN model in the matrix form:

$$\begin{aligned} \min_{(\alpha, b)} \quad & \frac{1}{2} \|\mathbf{y} - (\kappa(\mathbf{X}, \mathbf{X}) \mathbf{Y} \boldsymbol{\alpha} + b \mathbf{e})\|^2 \\ \text{s.t.} \quad & 0 \leq \boldsymbol{\alpha} \leq C \mathbf{e}, \end{aligned} \quad (4.3)$$

where $\mathbf{y} = (y_1, \dots, y_n)^T$ and $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_n)^T$ are n -dimensional vectors, $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)^T$ is an $n \times d$ matrix with the training sample \mathbf{x}_i^T being the i th row, $\kappa(\mathbf{X}, \mathbf{X})$ is an $n \times n$ matrix with $\kappa(\mathbf{x}_i, \mathbf{x}_j)$ being its (i, j) th element, $\mathbf{Y} = \text{diag}(y_1, \dots, y_n)$ is an $n \times n$ diagonal matrix with y_i as its (i, i) element, and \mathbf{e} is an n -dimensional column vector of 1's. For ease of representation, let us denote $\kappa(\mathbf{X}, \mathbf{X})$ as simply \mathbf{K} .

The objective function of the problem (4.3) becomes

$$\begin{aligned}
J &= \frac{1}{2} \|\mathbf{y} - (\mathbf{KY}\boldsymbol{\alpha} + b\mathbf{e})\|^2 \\
&= \frac{1}{2} \boldsymbol{\alpha}^T \mathbf{Y}^T \mathbf{K}^T \mathbf{KY} \boldsymbol{\alpha} + b\mathbf{e}^T \mathbf{KY} \boldsymbol{\alpha} + \frac{1}{2} b\mathbf{e}^T \mathbf{e} b - \mathbf{y}^T (\mathbf{KY} \boldsymbol{\alpha} + b\mathbf{e}) + \frac{1}{2} \mathbf{y}^T \mathbf{y} \\
&= \frac{1}{2} \hat{\boldsymbol{\alpha}}^T \hat{\mathbf{K}}^T \hat{\mathbf{K}} \hat{\boldsymbol{\alpha}} - \mathbf{q}^T \hat{\boldsymbol{\alpha}} + \frac{1}{2} \mathbf{y}^T \mathbf{y},
\end{aligned}$$

where $\hat{\boldsymbol{\alpha}}^T = \begin{pmatrix} \boldsymbol{\alpha}^T, & b \end{pmatrix}$ is an $(n+1)$ -dimensional vector, $\hat{\mathbf{K}} = (\mathbf{KY} \mid \mathbf{e})$ is an $n \times (n+1)$ matrix, $\mathbf{q}^T = \begin{pmatrix} \mathbf{y}^T \mathbf{KY}, & \mathbf{y}^T \mathbf{e} \end{pmatrix}$ is an $(n+1)$ -dimensional vector. Notice that $\frac{1}{2} \mathbf{y}^T \mathbf{y}$ is a constant, thus problem (4.3) is equivalent to

$$\begin{aligned}
&\min_{(\boldsymbol{\alpha}, b)} \quad \frac{1}{2} \hat{\boldsymbol{\alpha}}^T \hat{\mathbf{K}}^T \hat{\mathbf{K}} \hat{\boldsymbol{\alpha}} - \mathbf{q}^T \hat{\boldsymbol{\alpha}} \\
&s.t. \quad 0 \leq \boldsymbol{\alpha} \leq C\mathbf{e}.
\end{aligned} \tag{4.4}$$

Since the $\hat{\mathbf{K}}^T \hat{\mathbf{K}}$ is positive semi-definite, problem (4.4) is a convex quadratic programming problem, no matter what kernel function $\kappa(\cdot, \cdot)$ we choose. Thus, Mercer's condition is not required for the kernel functions. Thus, compared with the SVM model, the BNN model provides more flexibility in choosing the form of the kernel functions, which in turn provides more potential for learning.

Notice that the weight specification problems in both the BNN model and the SVM model are formulated as a convex quadratic programming problem. The BNN problem (4.4) has only the bound constraint, $0 \leq \boldsymbol{\alpha} \leq C\mathbf{e}$. For the convex quadratic programming problem with "bound constraints", a variety of algorithms [74, 78, 63] can be readily applied. On the other hand, the SVM problem (3.19) has the equality constraint $\mathbf{y}^T \boldsymbol{\alpha} = 0$ in addition to the bound constraint $0 \leq \boldsymbol{\alpha} \leq C\mathbf{e}$. Thus special care needs to be taken in designing an algorithm to solve the SVM problem.

4.2.2 Multi-Class Case

The model for the two-class classification problems can be easily extended for multi-class classification. In the multi-class classification, each sample is assumed to come from one of the c ($c \geq 2$) possible classes Ω_j , $j = 1, 2, \dots, c$. We are given n training

sample inputs $\mathbf{x}_i \in R^d$ together with corresponding labels $\mathbf{y}_i = (y_{i1}, \dots, y_{ic})^T$, where $y_{ij} = 1$ if \mathbf{x}_i is in class Ω_j , and $y_{ij} = -1$ if \mathbf{x}_i is not in class Ω_j . The goal is to find some decision function $g(\mathbf{x})$ which accurately predicts the class label of any given data \mathbf{x} .

For the multi-class problem, the BNN model for the two-class case needs to be modified to have c output nodes, as shown in Figure 4-2. Similar to the two-class case, the input data \mathbf{x} is nonlinearly transformed at the hidden node k as $z_k = \kappa(\mathbf{x}_k, \mathbf{x})$, where $z_k \in R$ is the output of the hidden node k , and $\kappa(\cdot, \cdot)$ is a nonlinear kernel function. At each output node j , the output is delivered as $\sum_{k=1}^n \alpha_{kj} y_{kj} z_k + b_j$, where $\alpha_{kj} \in R$ is the weight of the link connecting the hidden node k to the output node j , $b_j \in R$ is the bias of the output node j . Similar to the two-class case, the actual weight of the link connecting the hidden node k to the output node j is $\alpha_{kj} y_{kj}$, rather than α_{kj} . But for ease of exposition, we call α_{kj} the “weight”.

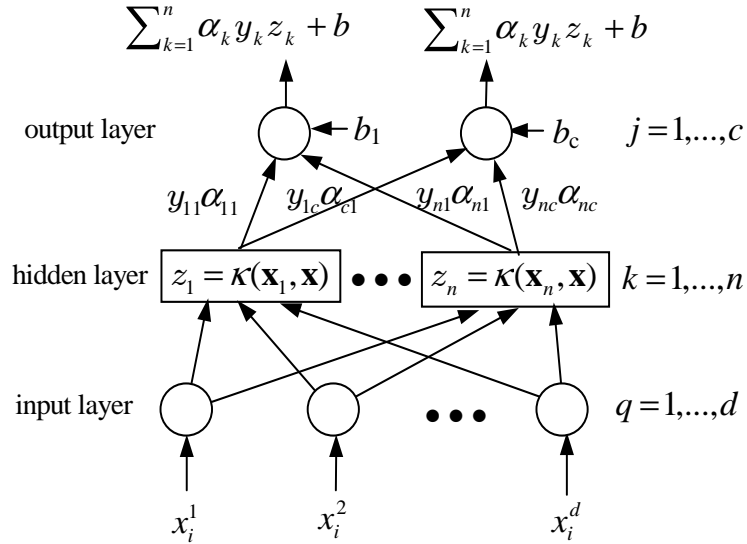


Figure 4-2: Bounded Neural Network: Multi-Class Case

Specifically, in the training stage, we try to find the optimal parameters (weights

α_{kj} , $k = 1, \dots, n$, $j = 1, \dots, c$, and biases b_j , $j = 1, \dots, c$) for the following problem:

$$\begin{aligned} \min_{(\alpha, b)} \quad & \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^c \|y_{ij} - (\sum_{k=1}^n \alpha_{kj} y_{kj} \kappa(\mathbf{x}_i, \mathbf{x}_k) + b_j)\|^2 \\ \text{s.t.} \quad & 0 \leq \alpha_{kj} \leq C, \quad k = 1, 2, \dots, n, \quad j = 1, 2, \dots, c. \end{aligned} \quad (4.5)$$

For prediction, the decision rule is

$$\text{assign } \mathbf{x} \text{ to class } k \text{ if } g_k(\mathbf{x}) \geq g_j(\mathbf{x}), \quad \forall j = 1, \dots, c,$$

where the discrimination function, $g_j(\mathbf{x})$, is

$$g_j(\mathbf{x}) = \sum_{k=1}^n \alpha_{kj} y_{kj} \kappa(\mathbf{x}, \mathbf{x}_k) + b_j, \quad j = 1, \dots, c.$$

Notice that in problem (4.5), there are $n \times c$ weight variables α_{kj} , $k = 1, \dots, n$, $j = 1, \dots, c$. The problem in (4.5) can be recast in the following form:

$$\begin{aligned} \min_{(\tilde{\alpha}, \tilde{\mathbf{b}})} \quad & \frac{1}{2} \left\| \tilde{\mathbf{y}} - (\tilde{\mathbf{K}} \tilde{\mathbf{Y}} \tilde{\alpha} + \tilde{\mathbf{E}} \tilde{\mathbf{b}}) \right\|^2 \\ \text{s.t.} \quad & 0 \leq \tilde{\alpha} \leq C \tilde{\mathbf{e}}, \end{aligned} \quad (4.6)$$

where $\tilde{\alpha} = (\alpha_{11} \cdots \alpha_{n1}, \dots, \alpha_{1c} \cdots \alpha_{nc})^T$ and $\tilde{\mathbf{y}} = (y_{11} \cdots y_{n1}, \dots, y_{1c} \cdots y_{nc})^T$ are nc -dimensional vectors, $\tilde{\mathbf{b}} = (b_1, \dots, b_c)^T$ is a c -dimensional vector, $\tilde{\mathbf{e}}$ is an nc -dimensional vector of 1's. $\tilde{\mathbf{Y}} = \text{diag}(\tilde{\mathbf{y}})$ is an $nc \times nc$ diagonal matrix with \tilde{y}_i being its i th diagonal

element. $\tilde{\mathbf{K}} = \begin{pmatrix} \kappa(\mathbf{X}, \mathbf{X}) & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \kappa(\mathbf{X}, \mathbf{X}) \end{pmatrix}$ is an $nc \times nc$ block diagonal matrix

which has c blocks of the matrix $\kappa(\mathbf{X}, \mathbf{X})$ as its diagonal elements. $\kappa(\mathbf{X}, \mathbf{X})$ is an $n \times n$ matrix with $\kappa(\mathbf{x}_i, \mathbf{x}_j)$ being its (i, j) element, and $\tilde{\mathbf{E}}$ is an $nc \times c$ matrix whose (i, j) element is 1 if $(j-1) \times n + 1 \leq i \leq j \times n$, and 0 otherwise.

The problem as stated in (4.6) can be further simplified as

$$\begin{aligned} \min_{(\tilde{\alpha}, \tilde{b})} \quad & \frac{1}{2} \tilde{\alpha}^T \hat{\mathbf{K}}^T \hat{\mathbf{K}} \tilde{\alpha} - \mathbf{q}^T \tilde{\alpha} \\ \text{s.t.} \quad & 0 \leq \tilde{\alpha} \leq C \tilde{\mathbf{e}} \end{aligned} \tag{4.7}$$

where $\hat{\alpha}^T = \left(\tilde{\alpha}^T, \tilde{\mathbf{b}} \right)$ is an $(nc + c)$ -dimensional vector, $\hat{\mathbf{K}} = \left(\tilde{\mathbf{K}}\tilde{\mathbf{Y}} \mid \tilde{\mathbf{E}} \right)$ is an $nc \times (nc + c)$ matrix, and $\mathbf{q}^T = \left(\tilde{\mathbf{y}}^T \tilde{\mathbf{K}}\tilde{\mathbf{Y}}, \tilde{\mathbf{y}}^T \tilde{\mathbf{E}} \right)$ is an $(nc + c)$ -dimensional vector. As with the 2-class case, (4.7) is a convex quadratic programming problem, no matter what kernel function we take.

4.3 Experimental Results

To demonstrate the competitiveness of the proposed model, we conducted several numerical experiments. In our experiments, the quadratic programming solver of Matlab 5.3 was used to solve the weight specification problem for the BNN model, while the LIBSVM2.2 [14] was used for the SVM model.

4.3.1 Small Example

For both the BNN and SVM models, we define the support vectors as the training samples with $\alpha_i > 0$. The first experiment was designed to show the different location of the support vectors for the two models. In this experiment, we randomly generated two classes of data, both of which are normally distributed but with different mean vectors, one at $(0, 0)^T$ and the other at $(2.5, 2.5)^T$, and the same covariance matrix $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$.

For each class, we randomly generated 50 data points. They are plotted in Figure 4-3 and Figure 4-4. Then we used the BNN model (4.1) and the SVM model (3.19) to train the data for the support vectors. For comparison purposes, we set the parameter C to be 1 in both models. We also adopted the same kernel function, namely the simple inner product function, $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$.

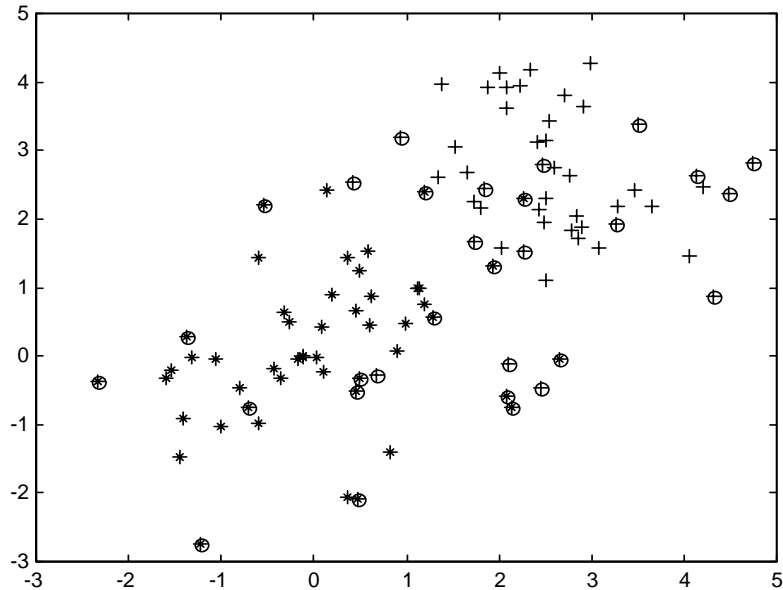


Figure 4-3: The Support Vectors for the BNN Model

The support vectors for the BNN and SVM models are shown in Figure 4-3 and Figure 4-4, respectively, where the support vectors are plotted as circles. As we can see from the figures, the support vectors for the SVM model lie close to the boundary between the two classes, while the support vectors for the BNN model are scattered throughout the input space. Since in classification problems, the key issue is to obtain the classification boundary, intuitively, it would seem better for the support vectors to “lie close to boundary” rather than “scattered around”. However, this is not always the case, as we show in an example in the next section.

4.3.2 Real-World Examples

To demonstrate the competitiveness of the proposed model, we tested the BNN model on five benchmark real-world data sets which are publicly available from a University of California at Irvine (UCI) data repository [76]. The results were compared with those reported for the SVM model. All of the five data sets are for two-class classification. They are listed as follows:

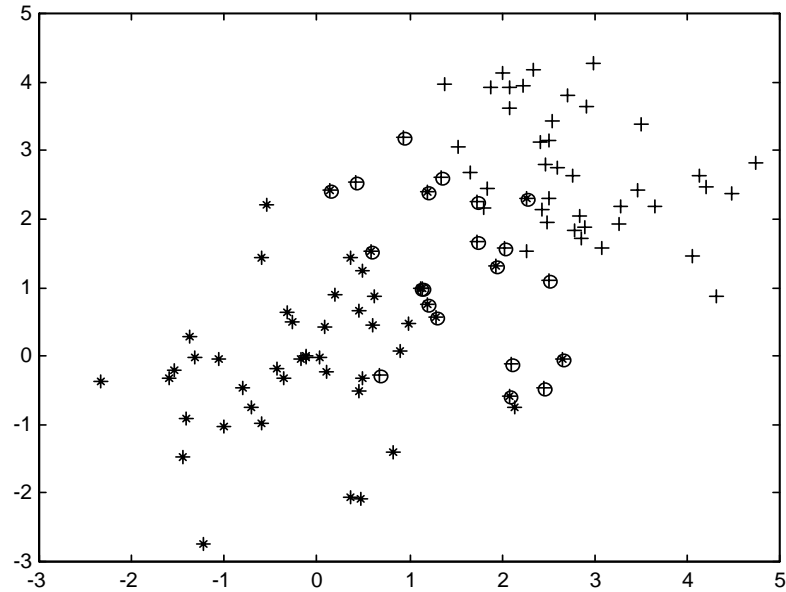


Figure 4-4: The Support Vectors for the SVM Model

- Wisconsin Breast Cancer. This breast cancer data set was from the University of Wisconsin Hospital, Madison. The objective is to detect whether a breast tumor is “benign” or “malignant”. The data set contains 699 points, each point consisting of 9 features. All features are continuous variables. The “benign” class contains 458 points; the “malignant” class contains 241 points.
- Cleveland Heart Disease. This data set was collected by the Cleveland Clinic Foundation. It was used to diagnose heart disease. The data set contains 303 points, each point consisting of 13 features. All features are continuous variables. The “positive” class (heart disease) contains 164 points; the negative class (no heart disease) contains 139 points.
- Liver Disorders. This data set was collected by the BUPA Medical Research Ltd.. It was used to detect the liver disorders caused by excessive alcohol consumption. It contains 345 points, each point consisting of 6 features. The first 5 features are the results of blood test which are thought

to be sensitive to liver disorders. The sixth feature is the number of half-pint equivalents of alcoholic beverages drunk per day. All features are continuous variables. The “positive” class (liver disorders) contains 145 points; the “negative” class (no liver disorders) contains 200 points.

- **Ionosphere.** This data set was collected by a radar system in Goose Bay, Labrador. The targets were free electrons in the ionosphere. The “good” radar signals are those showing evidence of a particular type of structure in the ionosphere. The “bad” signals are those that do not. The data set contains 351 points, each point consisting of 34 features. All features are continuous variables. The “good” signals class contains 225 points; the “bad” signals class contains 126 points.
- **Votes.** This data set was from the 1984 United States Congressional Voting Records Database. It contains the voting records for each of the U.S. House of Representatives Congressmen on the 16 key votes identified by the Congressional Quarterly Almanac. The objective is to identify the voting person is a “democrat” or a “republican”. The data set contains 435 points, each consisting of 16 features. All features are binary variables. The “democrats” class contains 267 points; the “republicans” class contains 168 points.

For the above data sets, the feature values are frequently not in the same range. In our experiments, we normalized all feature values to the range $[0, 1]$.

Our experiments were conducted using the ten-fold cross-validation technique, which is a commonly used technique to evaluate the performance of a pattern classification algorithm [28]. We conducted the ten-fold cross-validation in the following way: we randomly partitioned each data set into ten groups, each group having approximately the same number of points. Then we ran the BNN (or SVM) model ten times. Each time, one different group of data was held out for use as the test set; the other nine groups were used as the training set. The reported results are average values for these ten runs.

In our experiments, we tried a set of the values of the parameter, C . The reported results are the best results obtained.

For each data set, we tested the BNN and SVM models with three different types of kernel functions: the Gaussian kernel function, the polynomial kernel function and the sigmoid kernel function. For the Gaussian kernel function,

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|}{2h}\right),$$

we set the kernel width $h = d/2$, where d is the number of features. For the polynomial kernel function,

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i^T \mathbf{x}_j + \delta)^q,$$

we set $q = 3$, $\gamma = 1$, and $\delta = 1$. For the sigmoid kernel function,

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma \mathbf{x}_i^T \mathbf{x}_j + \delta),$$

we set $\gamma = 1$ and $\delta = 1$. Table 4.1, Table 4.2, and Table 4.3 list the experimental results for the Gaussian kernel function, polynomial kernel function, and sigmoid kernel function, respectively. In these tables, the “training correctness” (or “test correctness”) is defined as the ratio of the number of correctly classified training (or test) samples divided by the total number of training (or test) samples. “SV” denotes “Support Vector”, while “BSV” denotes “Bounded Support Vector”, which is defined as the support vector with $\alpha_i = C$.

As we can see from the tables, the BNN model and the SVM model have comparable training and test correctness. For these two models, the number of support vectors were also approximately in the same general range. For the “Liver Disorders” test, the BNN model outperformed the SVM model in both training and testing correctness, and the number of support vectors was much smaller than that of the SVM model. For the other tests, the performance of the BNN model is also close to that of the SVM model.

From these results, we may conclude that the performance of these two models are comparable. Interestingly, it seems there was not much difference in performance with the three different kinds of kernel functions.

4.3.3 General Kernel Functions

The next experiment was conducted to demonstrate that the kernel functions in the BNN model are not required to satisfy Mercer’s condition. In this experiment, we tested the BNN model with an “invalid” Mercer kernel function (i.e., a kernel function which does not satisfy Mercer’s condition). The result was compared with the one obtained using a valid Mercer kernel function.

The valid Mercer kernel function we tested is

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + 1)^3, \quad (4.8)$$

while the invalid Mercer kernel function we tested is,

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = -(\mathbf{x}_i^T \mathbf{x}_j + 1)^3. \quad (4.9)$$

The kernel matrix $\boldsymbol{\kappa}(\mathbf{X}, \mathbf{X})$ is defined as the $n \times n$ matrix with $\kappa(\mathbf{x}_i, \mathbf{x}_j)$ being its (i, j) th element. Function (4.8) is a valid Mercer kernel function since the kernel matrix $\boldsymbol{\kappa}(\mathbf{X}, \mathbf{X})$ is positive semi-definite. The function (4.9) is not a valid Mercer kernel function because the kernel matrix $\boldsymbol{\kappa}(\mathbf{X}, \mathbf{X})$ is negative semi-definite.

We tested the BNN model on the Cleveland Heart Disease data set using each of these two kernel functions. As we expected, the BNN model was able to find the solution to problem (4.4) with both kernel functions. The test results are listed in Table 4.4. As we can see from the table, the two kernel functions lead to identical testing correctness, while the training correctness for the valid Mercer kernel is slightly higher than that for the invalid Mercer kernel. Interestingly, the number of support vectors for the invalid Mercer kernel is much smaller.

From this test, we may conclude that the kernel functions, indeed, are not required

Table 4.1: The Experimental Results for the Gaussian Kernel Function

data set $n \times d$	Algorithm	Training correctness	Test correctness	Number of SVs	Number of BSVs
Breast Cancer 699×9	BNN	98.15%	96.43%	36	7
	SVM	97.97%	96.37%	33	19
Cleveland Heart 297×13	BNN	89.15%	84.87%	98	64
	SVM	86.87%	85.11%	114	91
Liver Disorders 345×6	BNN	76.12%	72.67%	136	128
	SVM	73.48%	71.35%	236	226
Ionosphere 351×34	BNN	95.37%	92.33%	123	85
	SVM	98.37%	93.12%	89	50
Votes 435×16	BNN	96.56%	95.15%	76	72
	SVM	96.79%	96.11%	60	58

Table 4.2: The Experimental Results for the Polynomial Kernel Function

data set $n \times d$	Algorithm	Training correctness	Test correctness	Number of SVs	Number of BSVs
Breast Cancer 699×9	BNN	98.21%	96.33%	36	9
	SVM	97.85%	96.37%	37	20
Cleveland Heart 297×13	BNN	88.18%	84.67%	98	65
	SVM	88.04%	84.67%	113	83
Liver Disorders 345×6	BNN	75.36%	73.42%	138	127
	SVM	74.88%	72.85%	235	226
Ionosphere 351×34	BNN	95.37%	91.73%	123	82
	SVM	97.23%	92.15%	93	58
Votes 435×16	BNN	96.02%	94.58%	74	71
	SVM	96.23%	95.46%	62	60

Table 4.3: The Experimental Results for the Sigmoid Kernel Function

data set $n \times d$	Algorithm	Training correctness	Test correctness	Number of SVs	Number of BSVs
Breast Cancer 699×9	BNN	97.85%	96.52%	36	11
	SVM	97.93%	96.23%	34	19
Cleveland Heart 297×13	BNN	87.32%	85.21%	94	59
	SVM	86.67%	85.17%	114	89
Liver Disorders 345×6	BNN	76.23%	73.35%	139	129
	SVM	75.84%	73.17%	238	228
Ionosphere 351×34	BNN	96.13%	93.56%	121	89
	SVM	97.75%	94.37%	91	55
Votes 435×16	BNN	96.13%	94.56%	73	70
	SVM	96.34%	95.63%	60	59

to satisfy Mercer’s condition in the BNN model. Thus the BNN model provides more flexibility to choose the kernel functions than the SVM model.

Table 4.4: The Experimental Results for General Kernel Functions

data set $n \times d$	Kernel	Training correctness	Test correctness	Number of SVs	Number of BSVs
Cleveland Heart 297×13	Valid	88.18%	84.67%	98	65
	Invalid	86.98%	84.67%	51	26

4.3.4 Multi-Class Classification

The final experiment was designed to demonstrate the capability of the BNN model to perform multi-class classification. In this experiment, we tested the BNN model on the IRIS flower classification problem. In this problem, we are given three classes of data, which represent three different types of iris flowers. Each class contains 50 points, with each point consisting of 4 features. The objective is to predict the class label based on the feature information.

We applied the BNN model to this problem. Again, we conducted the experiment by using the ten-fold cross-validation technique. The kernel function adopted was the Gaussian kernel function, $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|}{2h}\right)$, with the kernel width $h = d/2$, where d is the number of features, i.e., in this case $h = 2$. For the purpose of comparison, we applied the 1-Nearest Neighbor algorithm [41] to the Iris flower problem. The results are given in Table 4.5. The “Total” column contains the average value of the (training or test) correctness for class 1, class 2 and class 3.

As we can see from the table, the performance of the BNN model is not as good as that of the 1-Nearest Neighbor algorithm. While further investigation is needed, these results show that the BNN model can, indeed, perform multi-class classification.

Table 4.5: The Experimental Results for the Iris Flower Classification

Algorithm	Training Correctness				Test Correctness			
	Class 1	Class 2	Class 3	Total	Class 1	Class 2	Class 3	Total
BNN	100%	98.33%	87.56%	95.30%	100%	97.23%	82.89%	93.37%
1-NN	NA	NA	NA	NA	100%	94.00%	92.00%	95.33%

Chapter 5

Universal Approximation by Neural Networks

This chapter provides a brief introduction on universal approximation by neural networks. The basic definitions and notation are presented in Section 5.2. The research on universal approximation by neural networks is reviewed in Section 5.3, which includes the research on the neural networks with ridge activation functions (Section 5.3.1), and the research on the neural networks with radial-basis activation functions (Section 5.3.2).

5.1 Introduction

Feedforward Neural networks have been successfully applied to many areas, especially in pattern classification and nonlinear regression. In practice, feedforward neural networks perform quite well compared with their traditional competitors, such as linear classifier and Bayes classifier. This success has led many researchers to perform a rigorous analysis of the mathematical properties of feedforward neural networks. It was soon discovered that feedforward neural networks are capable of universal approximation, that is, feedforward neural networks can approximate a large class of functions.

Universal approximation by feedforward neural networks has been studied by many authors [24, 52, 53, 84, 85, 20]. Under very mild assumptions on the activation

functions in the hidden layer, it has been shown that a three-layered feedforward neural network is capable of approximating a large class of functions, including the continuous functions and integrable functions.

The known results in literature are mainly built upon the three-layered neural networks with one linear output node. We adopt this standard setting. The class of functions realized by a three-layered feedforward neural network has the following form:

$$\sum_{i=1}^N c_i g(x, \theta_i, b_i),$$

where N is the number of hidden nodes, $x \in R^n$ is a variable, $c_i \in R$, $\theta_i \in R^n$, $b_i \in R$ are parameters, and $g(x, \theta_i, b_i)$ is the activation function used in the hidden layer. Notice that most activation functions used in the hidden layer can be categorized into two classes - the ridge functions and radial-basis functions. A ridge function has the following form

$$g(x, \theta, b) = \sigma(\theta^T x + b)$$

where σ is a mapping from R into R , $x \in R^n$ is a variable, $\theta \in R^n$ is a “direction vector”, and $b \in R$ is a “threshold”. The commonly used sigmoid function $g(x) = 1/(1 + \exp(-(\theta^T x + b)))$ is an example. A radial-basis function has the form

$$g(x, \theta, b) = \phi\left(\frac{x - \theta}{b}\right)$$

where ϕ maps from R^n into R , $x \in R^n$ is a variable, $\theta \in R^n$ is a “center vector”, and $b \in R$ is a “spread parameter”. The Gaussian function $g(x) = \exp\left(-\frac{\|x - \theta\|^2}{b}\right)$ is a typical example.

The research on neural networks with ridge activation functions is extensive [24, 52, 53, 62]. Leshno et. al. [62] provide one of the most general results. They showed that if the ridge activation function used in the hidden layer is continuous almost everywhere, locally essentially bounded, and not a polynomial, then a three-layered neural network can approximate any continuous function with respect to the uniform norm.

The research on neural networks with radial-basis activation functions is less extensive [84, 85, 20]. The most well-known result is due to Park and Sandberg [84, 85]. They showed that if the radial-basis activation function used in the hidden layer is continuous almost everywhere, bounded and integrable on R^n , and the integration is not zero, then a three-layered neural network can approximate any function in $L^p(R^n)$ with respect to the L^p norm with $1 \leq p < \infty$.

In the following, we relax Park and Sandberg's result by showing that the integrability assumption is not necessary for the radial-basis function networks to be universal approximators. Instead, the relaxed conditions are more like the results of Leshno. More specifically, we show that, if the radial-basis activation function used in the hidden layer is continuous almost everywhere, locally essentially bounded, and not a polynomial, then the three-layered radial-basis function network can approximate any continuous function with respect to the uniform norm. Moreover, RBF networks can approximate any function in $L^p(\mu)$, where $1 \leq p < \infty$ and μ is any finite measure, if the radial-basis activation function used in the hidden layer is essentially bounded and not a polynomial.

5.2 Basic Definitions and Notation

The following notation will be used in the reminder of the dissertation.

R^n the n -dimensional Euclidean space.

K some compact set in R^n .

$C(K)$ the set of all continuous functions defined on K , with the uniform norm $\|f\|_{C(K)} = \max_{x \in K} |f(x)|$.

$C^\infty(R^n)$ the set of all infinitely differentiable functions defined on R^n .

$C_c^\infty(R^n)$ the set of all infinitely differentiable functions with compact support in R^n .

$S(R^n)$ all Schwartz functions in distribution theory [107], i.e., all the infinitely differentiable functions, which are rapidly decreasing at infinity.

$S'(R^n)$ all the distributions defined on $S(R^n)$, i.e., all the linear continuous functionals defined on $S(R^n)$.

We review the following definitions.

Definition 1 For a measurable function f in R^n , define

$$\|f\|_{L^p(\mu)} = \left(\int_{R^n} |f(x)|^p d\mu(x) \right)^{1/p},$$

where $1 \leq p < \infty$. Let $L^p(\mu)$ consist of all f for which

$$\|f\|_{L^p(\mu)} < \infty.$$

Definition 2 The essential supremum of a given function $f(x)$ is defined as

$$\operatorname{ess\,sup}_{x \in R^n} f(x) = \inf \{ \lambda | \mu \{ x \in R^n : |f(x)| \geq \lambda \} = 0 \},$$

where μ is Lebesgue measure. Define

$$\|f\|_{L^\infty(R^n)} = \operatorname{ess\,sup}_{x \in R^n} f(x),$$

and let $L^\infty(R^n)$ consist of all f for which

$$\|f\|_{L^\infty(R^n)} < \infty.$$

We call the functions in $L^\infty(R^n)$ essentially bounded. Similarly, for a compact set $K \subset R^n$, we have the $\|\cdot\|_{L^\infty(K)}$ norm with $L^\infty(K)$ space. Let $L_{loc}^\infty(R^n)$ consist of all f for which

$$\|f\|_{L^\infty(K)} < \infty$$

for every compact set $K \subset R^n$. We call the functions in $L_{loc}^\infty(R^n)$ locally essentially bounded.

Definition 3 A set of functions, S , is said to be dense in $C(K)$ (or $L^p(\mu)$), if for any $\varepsilon > 0$ and $f \in C(K)$ (or $f \in L^p(\mu)$), there is a function $g \in S$, such that $\|g - f\|_{L^\infty(K)} < \varepsilon$ (or $\|g - f\|_{L^p(\mu)} < \varepsilon$).

Definition 4 A function $\sigma : R^1 \rightarrow R^1$ is called a (generalized) sigmoidal function, if it satisfies

$$\begin{cases} \lim_{x \rightarrow -\infty} \sigma(x) = 0 \\ \lim_{x \rightarrow +\infty} \sigma(x) = 1 \end{cases}$$

Definition 5 A function is said to be continuous almost everywhere (with respect to a measure), if the measure of the set of all discontinuous points of the function is zero.

The convolution of two functions is denoted by $f * g$ and defined as $f * g(x) = \int f(x-t)g(t)dt$. The Fourier transform of a Fourier transformable function f is denoted as \hat{f} . The support of function f is denoted by $\text{supp } f$. An n -tuple $\alpha = (\alpha_1, \dots, \alpha_n)$ of nonnegative integers is called a multi-index. We define $|\alpha| = \alpha_1 + \dots + \alpha_n$ and $\alpha! = \alpha_1! \dots \alpha_n!$. The differential operator D^α is defined as

$$D^\alpha = \left(\frac{\partial}{\partial x_1}\right)^{\alpha_1} \dots \left(\frac{\partial}{\partial x_n}\right)^{\alpha_n}.$$

For a function $\phi(ax + \theta)$, where $x \in R^n$, $a \in R$ and $\theta \in R^n$, $\text{span}\{\phi(ax + \theta) : a \in R, \theta \in R^n\}$ denotes the set of all functions on R^n of the form

$$x \rightarrow \sum_{i=1}^N \beta_i \phi(a_i x + \theta_i),$$

where $\beta_i \in R^n$ and N is a given positive integer. Related terminologies and properties of functional analysis can be found in Rudin [89].

5.3 Literature Review

5.3.1 Ridge Activation Functions

The research on neural networks with ridge activation functions is extensive [47, 24, 21, 52, 53, 62]. One of the earliest works is in Hecht-Nielson [47]. The author used an improved version of Kolmogorov's theorem due to Sprecher [97] which states that every continuous function $f : [0, 1]^n \rightarrow R$ can be written as

$$f(x) = \sum_{h=1}^{2d+1} \phi_h \left(\sum_{k=1}^d \lambda^h \psi(x_k + \epsilon h) + h \right),$$

where the real λ and the continuous monotonically increasing function ψ are independent of f , the constant ϵ is a positive number and the continuous function ϕ_h , $1 \leq h \leq 2d+1$, depend on f . This equation can be interpreted as a three-layered neural network where the h th hidden node computes the function $z_h = \sum_{k=1}^d \lambda^h \psi(x_k + \epsilon h) + h$, and the output nodes compute $\sum_{h=1}^{2d+1} \phi_h(z_h)$. However, this is not the network architecture commonly used in practice.

One of the most elegant approaches to prove universal approximation is proposed by Cybenko [24]. By using the Hahn-Banach Theorem and the Riesz Representation Theorem, he showed that if the ridge activation function, σ , is a continuous sigmoid, then the set of $\sum_{i=1}^N c_i \sigma(\theta_i^T x + b_i)$ is dense in $C(K)$ with respect to uniform norm. Later, his approach was adopted by many authors to prove their results.

Chui and Li [21] adopted another approach to prove universal approximation. They showed that if the ridge activation function σ , is a continuous sigmoid and the direction vector θ satisfies some interpolation conditions, then the set of $\sum_{i=1}^N c_i \sigma(\theta_i^T x + b_i)$ is dense in $C(K)$ with respect to uniform norm. They constructed their proof by

showing that it is possible to realize polynomials as a sum of ridge activation functions. Since polynomials are dense in $C(R^n)$, it follows that the three-layered neural networks are dense in $C(K)$ with respect to uniform norm.

Another approach is to use the inverse Radon transform [48]. The Radon transform \tilde{f} represents a function $f \in C^\infty(K)$ by the sets of all integrals over hyperplanes in R^n . Furthermore, by the inverse Radon transform, f can be written as

$$f(x) = \int_{\|v\|=1} h(v^T x, v) d\mu(v),$$

where h is called the filtered back-projection function and is obtained by differentiation and Hilbert transforms of the Radon transform \tilde{f} . The basic idea of this approach is to transform the integral into a sum of a finite number of terms and to approximate f by

$$\bar{f}(x) = \sum_{i=1}^N h(v_i^T x, v_i) \mu_i,$$

where v_1, \dots, v_N are a set of vectors and $\mu_i = d\mu(v_i)$. Since each term $h(v_i^T x, v_i)$ can be regarded as a ridge function, $f(x)$ can be approximated by a three-layered feedforward neural network.

In Chen et. al. [18], by using the above argument, the authors showed that the continuity assumption usually imposed on the sigmoid functions is unnecessary. Instead, they proved that if the ridge activation function σ , is a bounded sigmoid, then the set of $\sum_{i=1}^N c_i \sigma(\theta_i^T x + b_i)$ is dense in $C(K)$ with respect to uniform norm. They also pointed out that in order to prove the neural network in the n -dimensional case, all one needs to do is to prove the case for one dimension.

In Hornik [52], the author adopted Cybenko's approach to prove universal approximation. He showed the sigmoid assumption usually imposed on the ridge function is unnecessary. Instead, he proved that if the ridge activation function σ , is continuous, bounded and nonconstant, then the set of $\sum_{i=1}^N c_i \sigma(\theta_i^T x + b_i)$ is dense in $C(K)$ with respect to uniform norm. At the same time, he proved that if the ridge activation function σ , is bounded and nonconstant, then the set of $\sum_{i=1}^N c_i \sigma(\theta_i^T x + b_i)$ is dense

in $L^p(\mu)$ with respect to L^p norm for $1 \leq p < \infty$ and a finite measure μ .

The work of Leshno et. al. [62] provides one of the most general results. They showed that if the ridge activation function σ , is continuous almost everywhere, locally essentially bounded, and not a polynomial, then the set of $\sum_{i=1}^N c_i \sigma(\theta_i^T x + b_i)$ is dense in $C(K)$ with respect to uniform norm.

The basic idea of their proof is as follows. First, they prove that if $g \in C^\infty(R^n)$ and g is not a polynomial, then $\sum_{i=1}^N c_i g(\theta_i^T x + b_i)$ is dense in $C(R^n)$. Then they prove that the function $h = \sigma * \varphi$ (where σ is the ridge activation function and $\varphi \in C_c^\infty(R^n)$), satisfies $h \in C^\infty(R^n)$, h is not a polynomial, and h can be uniformly approximated by $\sum_{i=1}^N r_i \sigma(\eta_i^T x + q_i)$. The result follows.

5.3.2 Radial-Basis Activation Functions

The research on neural networks with radial-basis activation functions is less extensive [84, 85, 20].

The most well-known result is due to Park and Sandberg [84, 85]. In [84], they proved that if the radial-basis activation function, ϕ , is continuous almost everywhere, bounded and integrable on R^n , and the integration is not zero, then the set of $\sum_{i=1}^N c_i \phi\left(\frac{x-\theta_i}{b_i}\right)$ is dense in $L^p(R^n)$ with respect to the L^p norm with $1 \leq p < \infty$.

The basic idea of their proof is as follows. Assume that the radial-basis activation function, $\phi\left(\frac{x-\theta_i}{b_i}\right)$, is integrable on R^n and $\int_{R^n} \phi(x) dx = 1$. Then the convolution of ϕ_ε and $f \in L^p(R^n)$, $\phi_\varepsilon * f$, satisfies $\|f - \phi_\varepsilon * f\|_{L^p(R^n)} \rightarrow 0$ as $\varepsilon \rightarrow 0$, where $\phi_\varepsilon(x) = \varepsilon^{-n} \phi\left(\frac{x}{\varepsilon}\right)$, $\varepsilon > 0$. The next step is to approximate $\phi_\varepsilon * f$ by a finite sum in the form of

$$\tilde{f}(x) = \sum_{i=1}^N f(\theta_i) \phi_\varepsilon(\theta_i - x),$$

where θ_i , $i = 1, \dots, N$, are selected according to a grid over R^n . Since $\|\tilde{f} - \phi_\varepsilon * f\|_{L^p(R^n)} \rightarrow 0$ as $N \rightarrow \infty$, the result follows.

Following the same approach, Park and Sandberg presented extended results in [85]. They proved that if radial-basis activation function ϕ , is integrable, then the set of $\sum_{i=1}^N c_i \phi\left(\frac{x-\theta_i}{b_i}\right)$ is dense in $L^1(R^n)$ if and only if $\int_{R^n} \phi(x) dx \neq 0$. They also gave

other conditions for the other norms.

Chen and Chen [20] studied the approximation capability of the set of functions

$$\sum_{i=1}^N c_i g(a_i \|x - \theta_i\|),$$

where $x \in R^n$, $\theta_i \in R^n$, and $c_i, a_i \in R$, $i = 1, \dots, N$. They proved that if the radial-basis activation function $g \in C(R) \cap S'(R)$ (i.e., all those continuous functions such that $\int_R g(x) s(x) dx$ makes sense for all $s \in S(R)$), then the family $\sum_{i=1}^N c_i g(a_i \|x - \theta_i\|)$ is dense in $C(K)$ if and only if g is not an even polynomial. Compared with Park and Sandberg's results, this result does not require radial-basis activation function to be integrable. However, it does require the activation function to be a continuous distribution function, which is a strong requirement. Furthermore, a norm was imposed on $(x - \theta_i)$, therefore, the network structure is not general enough.

Chapter 6

Relaxed Conditions

This chapter presents the relaxed conditions for RBF networks to be universal approximators. We show that if the radial-basis activation function used in the hidden layer is continuous almost everywhere, locally essentially bounded, and not a polynomial, then the three-layered radial-basis function networks can approximate any continuous function with respect to the uniform norm. Moreover, RBF networks can approximate any function in $L^p(\mu)$, where $1 \leq p < \infty$ and μ is any finite measure, if the radial-basis activation function used in the hidden layer is essentially bounded and not a polynomial. The theoretical results are presented in Section 6.1. Numerical results are reported in Section 6.2 to demonstrate the validity of our theoretical findings.

6.1 Main Result

The original radial-basis function is of the form $\phi\left(\frac{x-\theta}{b}\right)$, where $x \in R^n$, $\theta \in R^n$ and $b \in R$. In the following sections, for convenience, we write it as $\phi(ax + \theta)$, where $x \in R^n$, $a \in R$ and $\theta \in R^n$.

First, we have the following lemma due to Cybenko [24].

Lemma 1 *Let ϕ be a mapping from R^n to R and $\phi(x) \in C(K)$. If $\Phi = \text{span}\{\phi(ax + \theta) : a \in R, \theta \in R^n\}$ is not dense in $C(K)$, then there exists a non-zero signed finite measure λ , such that $\int_X \phi(ax + \theta) d\lambda(x) = 0$ for all $a \in R$ and $\theta \in R^n$.*

Proof. Since $\phi(x) \in C(K)$, Φ is a linear subspace of space $C(K)$. Since Φ is not dense in $C(K)$, then closure of Φ , say $\overline{\Phi}$, is a closed proper subspace of $C(K)$. By the Hahn-Banach Theorem [89], there is a bounded linear functional on $C(K)$, call it Λ , with the property that $\Lambda \neq 0$ but $\Lambda(\overline{\Phi}) = \Lambda(\Phi) = 0$.

By the Riesz Representation Theorem [89], this bounded linear functional, Λ , is of the form

$$\Lambda(h) = \int_X h(x) d\lambda(x)$$

for some finite, signed regular Borel measure λ on K , for all $h \in C(K)$. In particular, since $\phi(ax + \theta)$ is in Φ for all $a \in R$ and $\theta \in R^n$, we must have that

$$\int_X \phi(ax + \theta) d\lambda(x) = 0$$

for all $a \in R$ and $\theta \in R^n$. Hence, the proof is complete. ■

Then, we have the following result.

Theorem 1 *Let ϕ be a mapping from R^n to R . If $\phi \in C^\infty(R^n)$ and is not a polynomial, then for any compact set $K \subset R^n$, $\Phi = \text{span}\{\phi(ax + \theta) : a \in R, \theta \in R^n\}$ is dense in $C(K)$ with respect to the uniform norm, i.e., given any $f \in C(K)$ and any $\varepsilon > 0$, there exists a $g \in \Phi$ such that $|f(x) - g(x)| \leq \varepsilon$ for all $x \in K$.*

Proof. Assume that Φ is not dense in $C(K)$. Since $\phi \in C^\infty(R^n)$, by Lemma 1, there exists a non-zero signed finite measure λ on K , such that

$$\int_K \phi(ax + \theta) d\lambda(x) = 0$$

for all $a \in R$ and $\theta \in R^n$.

Since $\phi \in C^\infty(R^n)$, using the multivariate Taylor expansion, we have

$$\begin{aligned}\phi(ax + \theta) &= \sum_{|\alpha|=0}^{\infty} \frac{1}{\alpha!} (D^\alpha \phi)(\theta) (ax)^\alpha \\ &= \sum_{|\alpha|=0}^{\infty} \frac{a^{|\alpha|}}{\alpha!} (D^\alpha \phi)(\theta) x^\alpha \\ &= \phi(\theta) + a \sum_{|\alpha|=1} \frac{1}{\alpha!} (D^\alpha \phi)(\theta) x^\alpha + a^2 \sum_{|\alpha|=2} \frac{1}{\alpha!} (D^\alpha \phi)(\theta) x^\alpha + \dots\end{aligned}$$

Let $H(a) = \int_K \phi(ax + \theta) d\lambda(x)$. Since $H(a) = 0$ for every $a \in R$ and $\theta \in R^n$, the k -th derivative of H with respect to a becomes

$$\begin{aligned}\frac{d^k H}{da^k} &= \int_K \left[\sum_{|\alpha|=k} \frac{k!}{\alpha!} (D^\alpha \phi)(\theta) x^\alpha + \right. \\ &\quad \left. a \sum_{|\alpha|=k+1} \frac{(k+1)!}{\alpha!} (D^\alpha \phi)(\theta) x^\alpha + \dots \right] d\lambda(x) \\ &= 0\end{aligned}$$

for all $\theta \in R^n$. If we set $a = 0$, then

$$\begin{aligned}\left. \frac{d^k H}{da^k} \right|_{a=0} &= \int_K \left[\sum_{|\alpha|=k} \frac{k!}{\alpha!} (D^\alpha \phi)(\theta) x^\alpha \right] d\lambda(x) \\ &= \sum_{|\alpha|=k} \left[\frac{k!}{\alpha!} (D^\alpha \phi)(\theta) \int_{R^n} x^\alpha d\lambda(x) \right] \\ &= 0\end{aligned}$$

for all $\theta \in R^n$. Equivalently, we have

$$\sum_{i=1}^{r(k)} c_i(\theta) t_i = 0$$

for all $\theta \in R^n$, where $c_i(\theta) = k! (D^\alpha \phi)(\theta)$, $t_i = \frac{1}{\alpha!} \int_K x^\alpha d\lambda(x)$, and $r(k)$ is the number of α 's such that $|\alpha| = k$.

Since $\phi \in C^\infty(R^n)$ and is not a polynomial, $c_i(\theta)$ is continuous and not a constant. Therefore, $c_i(\theta)$ can have infinitely many values for different θ 's. Hence, there exist at least $(r(k) + 1)$ θ 's, such that the above linear system is overdetermined. Therefore, the only solution for the above linear system is $t_i = 0$ for all i . That is, $\int_K x^\alpha d\lambda(x) = 0$ for all multiindex $\alpha \geq 0$. This means that the Fourier transform (of λ) $\hat{\lambda}(t) = \int_K e^{-it^T x} d\lambda(x) = 0$ for all $t \in R^n$. By Theorem 1.3.7.b in Rudin [89], we have $\lambda = 0$. But this is impossible and, hence, the proof is complete. ■

The above theorem says that if the activation function used in the hidden layer is infinitely differentiable and not a polynomial, then the three-layered radial-basis function network is a universal approximator. The requirement of infinite differentiability is very strong in theory. But since neural networks are often trained with back-propagation algorithms, which usually assume the activation function used in the hidden layer to be differentiable, this requirement does not cause too much problem in practice. Fortunately, we can relax this requirement in the following derivation.

Lemma 2 *Let σ be a mapping from R^n to R . If $\sigma \in L_{loc}^\infty(R^n)$ and σ is not a polynomial, then there exists at least one $\omega \in C_c^\infty(R^n)$, such that $\sigma * \omega(x) = \int_{R^n} \sigma(x - t) \omega(t) dt$ is not a polynomial.*

Proof. Since $\sigma \in L_{loc}^\infty(R^n)$, $\sigma * \omega$ is well-defined and $\sigma * \omega \in C^\infty(R^n)$. Suppose that $\sigma * \omega(x)$ is a polynomial for every $\omega \in C_c^\infty(R^n)$. Then for any multi-index α such that $|\alpha| = \infty$, we have

$$D^\alpha \sigma * \omega(x) = 0$$

for all $\omega \in C_c^\infty(R^n)$ and $x \in R^n$. But according to Friedman [34], σ is a polynomial of degree $< |\alpha| = \infty$, which causes a contradiction, and the proof is complete. ■

Lemma 3 *Let σ be a mapping from R^n to R . If $\sigma \in L_{loc}^\infty(R^n)$ and σ is continuous almost everywhere, then for each $\omega \in C_c^\infty(R^n)$, $\sigma * \omega(x)$ can be uniformly approximated by $\Sigma = \text{span}\{\sigma(ax + \theta) : a \in R, \theta \in R^n\}$.*

Proof. Recall that

$$\sigma * \omega(x) = \int_{R^n} \sigma(x-t) \omega(t) dt$$

is well-defined.

Suppose that the $\text{supp } \omega \subseteq [-T, T]^n$. We show that $\sigma * \omega(x)$ can be uniformly approximated on $[-T, T]^n$ by

$$\sum_{i=1}^{m^n} \sigma(x-t_i) \omega(t_i) \left(\frac{2T}{m}\right)^n,$$

where $\{t_i \in R^n : i = 1, \dots, m^n\}$ is a set consisting of all points in $[-T, T]^n$ of the form $[-T + \frac{2i_1T}{m}, \dots, -T + \frac{2i_nT}{m}]^T$, $i_1, \dots, i_n = 1, 2, \dots, m$.

Set $\Delta_i = [t_{i-1}, t_i]$. By the triangular inequality, we have

$$\begin{aligned} & \left| \int_{R^n} \sigma(x-t) \omega(t) dt - \sum_{i=1}^{m^n} \sigma(x-t_i) \omega(t_i) \left(\frac{2T}{m}\right)^n \right| \\ &= \left| \int_{R^n} \sigma(x-t) \omega(t) dt - \sum_{i=1}^{m^n} \int_{\Delta_i} \sigma(x-t_i) \omega(t) dt + \right. \\ & \quad \left. \sum_{i=1}^{m^n} \int_{\Delta_i} \sigma(x-t_i) \omega(t) dt - \sum_{i=1}^{m^n} \sigma(x-t_i) \omega(t_i) \left(\frac{2T}{m}\right)^n \right| \\ &\leq \left| \int_{R^n} \sigma(x-t) \omega(t) dt - \sum_{i=1}^{m^n} \int_{\Delta_i} \sigma(x-t_i) \omega(t) dt \right| + \\ & \quad \left| \sum_{i=1}^{m^n} \int_{\Delta_i} \sigma(x-t_i) \omega(t) dt - \sum_{i=1}^{m^n} \sigma(x-t_i) \omega(t_i) \left(\frac{2T}{m}\right)^n \right|. \end{aligned} \tag{6.1}$$

Notice that in the second term of (6.1), $\int_{\Delta_i} dt = \left(\frac{2T}{m}\right)^n$. Therefore, we have

$$\begin{aligned} & \left| \sum_{i=1}^{m^n} \int_{\Delta_i} \sigma(x - t_i) \omega(t) dt - \sum_{i=1}^{m^n} \sigma(x - t_i) \omega(t_i) \left(\frac{2T}{m}\right)^n \right| \\ &= \left| \sum_{i=1}^{m^n} \int_{\Delta_i} \sigma(x - t_i) [\omega(t) - \omega(t_i)] dt \right| \\ &\leq \sum_{i=1}^{m^n} \int_{\Delta_i} |\sigma(x - t_i)| |\omega(t) - \omega(t_i)| dt. \end{aligned}$$

Since ω is continuous, it is uniformly continuous on $[-T, T]^n$. Therefore, we may choose m to be sufficiently large such that

$$\sum_{i=1}^{m^n} \int_{\Delta_i} |\sigma(x - t_i)| |\omega(t) - \omega(t_i)| dt \leq \varepsilon.$$

For the first term of (6.1), we have

$$\begin{aligned} & \left| \int_{R^n} \sigma(x - t) \omega(t) dt - \sum_{i=1}^{m^n} \int_{\Delta_i} \sigma(x - t_i) \omega(t) dt \right| \\ &= \left| \sum_{i=1}^{m^n} \int_{\Delta_i} [\sigma(x - t) - \sigma(x - t_i)] \omega(t) dt \right| \\ &\leq \sum_{i=1}^{m^n} \int_{\Delta_i} |\sigma(x - t) - \sigma(x - t_i)| |\omega(t)| dt. \end{aligned}$$

Since σ is continuous almost everywhere, the measure of the set of its discontinuous points is zero. Therefore, given any small number $\delta > 0$, we can find a countable number of intervals, whose union is of measure δ , such that σ is uniformly continuous on $[-T, T]^n \setminus t$. For any Δ_i , since $\Delta_i = (\Delta_i \setminus t) \cup (\Delta_i \cap t)$, the above equation can be

written as

$$\begin{aligned}
& \sum_{i=1}^{m^n} \int_{\Delta_i} |\sigma(x-t) - \sigma(x-t_i)| |\omega(t)| dt \\
&= \sum_{i=1}^{m^n} \int_{\Delta_i/t} |\sigma(x-t) - \sigma(x-t_i)| |\omega(t)| dt + \\
& \sum_{i=1}^{m^n} \int_{\Delta_i \cap t} |\sigma(x-t) - \sigma(x-t_i)| |\omega(t)| dt. \tag{6.2}
\end{aligned}$$

Now, for the first term of (6.2), since σ is uniformly continuous on $[-T, T]^n \setminus t$, we may choose m to be sufficiently large so that

$$\sum_{i=1}^{m^n} \int_{\Delta_i/t} |\sigma(x-t) - \sigma(x-t_i)| |\omega(t)| dt \leq \varepsilon.$$

For the second term of (6.2), since $\int_t dt = \delta$, we can choose δ to be sufficiently small such that

$$\begin{aligned}
& \sum_{i=1}^{m^n} \int_{\Delta_i \cap t} |\sigma(x-t) - \sigma(x-t_i)| |\omega(t)| dt \\
& \leq 2 \|\sigma\|_{L^\infty[-T, T]} \|\omega\|_{L^\infty} \delta \\
& \leq \varepsilon.
\end{aligned}$$

Therefore,

$$\sum_{i=1}^{m^n} \int_{\Delta_i} |\sigma(x-t) - \sigma(x-t_i)| |\omega(t)| dt \leq 2\varepsilon.$$

Consequently,

$$\left| \int_{R^n} \sigma(x-t) \omega(t) dt - \sum_{i=1}^{m^n} \sigma(x-t_i) \omega(t_i) \left(\frac{2T}{m} \right)^n \right| \leq 3\varepsilon$$

for all $x \in [-T, T]$. The proof is complete. ■

By combining the above lemmas and theorem, we have the following main result:

Theorem 2 *Let σ be a mapping from R^n to R . If σ is continuous almost everywhere,*

locally essentially bounded, and not a polynomial, then for any compact set $K \subset R^n$, $\Sigma = \text{span}\{\sigma(ax + \theta) : a \in R, \theta \in R^n\}$ is dense in $C(K)$ with respect to the uniform norm, i.e., given any $f \in C(K)$ and any $\varepsilon > 0$, there exists a $g \in \Sigma$, such that $\|f(x) - g(x)\|_{L^\infty(K)} \leq \varepsilon$ for all $x \in K$.

Proof. From Lemma 2, we know that there exists some $\omega \in C_c^\infty(R^n)$, such that $\sigma * \omega(x)$ is not a polynomial. Since $\sigma * \omega(x) \in C^\infty(R^n)$, by Theorem 1, we know $\text{span}\{\sigma * \omega(ax + \theta)\}$ is dense in $C(K)$. From Lemma 3, $\sigma * \omega$ can be uniformly approximated by Σ . It follows that $\text{span}\{\sigma * \omega(ax + \theta)\}$ can be uniformly approximated by Σ . Thus Σ is dense in $C(K)$. ■

The above result says that if the activation function used in the hidden layer is continuous almost everywhere, locally essentially bounded, and not a polynomial, then the three-layered radial-basis function network is a universal approximator. This significantly extends Park and Sandberg's results. In particular, the activation function is no longer required to be integrable. Notice that if the activation function used in the hidden layer is a polynomial, the neural network can only produce a polynomial of a certain degree. Therefore, the requirement of being "not a polynomial" is also a necessary condition. We do not know if the requirement of being "continuous almost everywhere and locally essentially bounded" is also a necessary condition.

Besides the approximation on $C(K)$, we have the following result for universal approximation in the $L^p(\mu)$ space, where $1 \leq p < \infty$ and μ is a finite measure.

First, we have the following lemma due to Hornik [52].

Lemma 4 *Let σ be a mapping from R^n to R and $\sigma \in L^\infty(\mu)$ for any finite measure μ . If $\Sigma = \text{span}\{\sigma(ax + \theta) : a \in R, \theta \in R^n\}$ is not dense in $L^p(\mu)$, then there exists a non-zero signed finite measure λ , such that $\int_{R^n} \sigma(ax + \theta) d\lambda(x) = 0$ for all $a \in R$ and $\theta \in R^n$.*

Proof. Since $\sigma \in L^\infty(\mu)$, Σ is a linear subspace of $L^p(\mu)$. Since Σ is not dense in $L^p(\mu)$, the closure of Σ , say $\overline{\Sigma}$, is a closed proper subspace of $L^p(\mu)$. By the Hahn-Banach Theorem [89], there is a bounded linear functional on $L^p(\mu)$, call it Λ , with the property that $\Lambda \neq 0$ but $\Lambda(\overline{\Sigma}) = \Lambda(\Sigma) = 0$.

By Theorem 4.14.1 and Theorem 4.14.6 in Friedman [36], this bounded linear functional, Λ , is of the form

$$\Lambda(h) = \int_{R^n} h(x) g(x) d\mu(x)$$

for all $h \in L^p(\mu)$ with some $g(x) \in L^q(\mu)$, where q is the conjugate exponent of p ($\frac{1}{q} + \frac{1}{p} = 1$).

If we write $\lambda(B) = \int_B g(x) d\mu(x)$, by Hölder's inequality, we have that for all B ,

$$\begin{aligned} |\lambda(B)| &= \left| \int_{R^n} 1_B g(x) d\mu(x) \right| \\ &\leq \|1_B\|_{p,\mu} \|g\|_{q,\mu} \\ &\leq (\mu(R^n))^{1/p} \|g\|_{q,\mu} \\ &< \infty. \end{aligned}$$

Therefore, λ is a nonzero finite signed measure on R^n such that $\Lambda(h) = \int_{R^n} h(x) g(x) d\mu(x) = \int_{R^n} h(x) d\lambda(x)$.

Since $\sigma(ax + \theta)$ is in Σ for all $a \in R$ and $\theta \in R^n$, we must have that

$$\int_{R^n} \sigma(ax + \theta) d\lambda(x) = 0$$

for all $a \in R$ and $\theta \in R^n$. Hence, the proof is complete. ■

From Lemma 4, we have the following result.

Theorem 3 *Let σ be a mapping from R^n to R . For any finite measure μ , if $\sigma \in L^\infty(\mu)$ and is not a polynomial, then $\Sigma = \text{span}\{\sigma(ax + \theta) : a \in R, \theta \in R^n\}$ is dense in $L^p(\mu)$, for $1 \leq p < \infty$.*

Proof. Assume that Σ is not dense in $L^p(\mu)$. Since $\sigma \in L^\infty(\mu)$, by Lemma 4, there exists a non-zero signed finite measure λ such that

$$\int_{R^n} \sigma(ax + \theta) d\lambda(x) = 0$$

for all $a \in R$ and $\theta \in R^n$. From Lemma 2, we can choose a $\omega \in C_c^\infty(R^n)$ such that $\sigma * \omega$ is not a polynomial.

Now consider the integral

$$\int_{R^n} \sigma * \omega(ax + \theta) d\lambda(x) = \int_{R^n} \int_{R^n} \sigma(ax + \theta - t) \omega(t) dt d\lambda(x).$$

Since

$$\begin{aligned} & \int_{R^n} \int_{R^n} |\sigma(ax + \theta - t) \omega(t)| dt d|\lambda(x)| \\ & \leq \|\sigma(ax + \theta - t)\|_{L^\infty} \|\omega\|_{L^1} \lambda(R^n) \\ & < \infty, \end{aligned}$$

by Fubini's Theorem [89], we have

$$\begin{aligned} & \int_{R^n} \sigma * \omega(ax + \theta) d\lambda(x) \\ & = \int_{R^n} \int_{R^n} \sigma(ax + \theta - t) \omega(t) dt d\lambda(x) \\ & = \int_{R^n} \left[\int_{R^n} \sigma(ax + \theta - t) d\lambda(x) \right] \omega(t) dt \\ & = 0. \end{aligned}$$

Since $\sigma * \omega \in C^\infty(R^n)$ and it is not a polynomial, the rest of the proof is identical to that of Theorem 1. ■

6.2 Numerical Experiments

In this section, we present some numerical experiments to demonstrate the validity of the obtained results. We show that even when the activation function used in the hidden layer is not integrable, a radial-basis function network may still be a universal approximator as long as the activation function meets our (much relaxed) conditions.

In our experiments, we use two different activation functions in the hidden layer.

One is the traditional Gaussian function $g(x) = \exp\left(-\frac{\|x-\theta\|}{b}\right)$, where $x \in R^n$, $\theta \in R^n$ is a center vector, and $b \in R$ is a spread parameter. The Gaussian function is integrable and meets Park and Sandberg's requirements. The other activation function is $g(x) = \exp\left(\frac{\|x-\theta\|}{b}\right)$, where $x \in R^n$, $\theta \in R^n$ is a center vector, and $b \in R$ is a spread parameter. This function is not integrable, but it meets the relaxed conditions proposed in this dissertation.

Corresponding radial-basis function networks are constructed to approximate a set of N input-output pairs (x_i, y_i) , $i = 1, \dots, N$. We show that both radial-basis function networks are capable of performing universal approximation.

In our experiments, the Mean Squared Error between the targets and actual outputs is used as the performance measure. The radial-basis function networks are trained according to the following steps:

- Step 1: Set the number of hidden nodes to be 1;
- Step 2: Choose the spread parameter and the center vectors;
- Step 3: Optimize the weights on the links between the hidden layer and the output layer. Feed the inputs into the network and get the outputs;
- Step 4: Increase the number of hidden nodes by 1. If the number of hidden nodes is less than or equal to the number of samples, go to Step 2; otherwise, stop.

The optimization of weights in Step 3 is straightforward, since it only involves solving a linear system. We use two approaches to select the center vectors, θ . One is the Random Selection method, which randomly generates a set of centers. The other one is the Orthogonal Least Squares method [16], which considers the complete set of training samples to be candidates for centers, and selects the one that reduces the mean squared error the most. In the following, we denote the Random Selection method as RS, and the Orthogonal Least Squares method as OLS.

6.2.1 One-Dimensional Example

The first experiment is a one-dimensional example, which is used as an illustrative example for function approximation in Matlab 5.3. We are given 21 input-output pairs (x_i, y_i) , $i = 1, \dots, 21$, which are depicted in Figure 6-1.

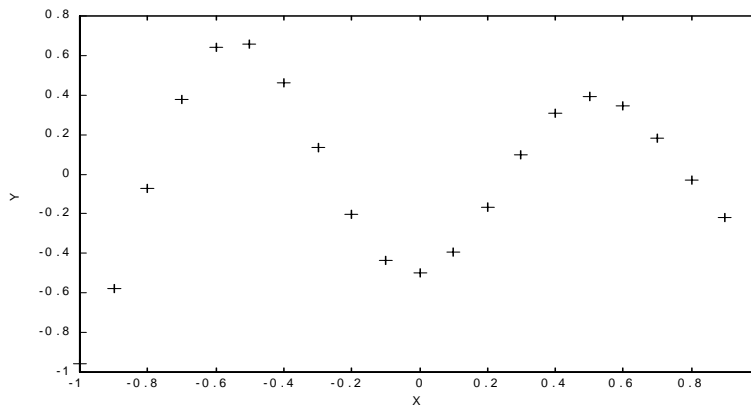


Figure 6-1: The function to be approximated.

The approximation results for different activation functions and center selection methods are depicted in Figures 6-2, 6-3, 6-4, and 6-5, respectively. As we can see from the figures, in all cases, the Mean Squared Errors between the targets and the actual outputs decreases to zero as the number of hidden nodes increases. This demonstrates that the activation functions used in the hidden layer need not be integrable in order to achieve universal approximation.

6.2.2 Multi-Dimensional Examples

In this example, we used the well-known Cleveland heart disease data [76] and the BUPA medical liver disorder data as the data samples [76]. The Cleveland heart disease data set was originally used for pattern classification to diagnose heart disease. It contains 303 points, each point consisting of 13 features. All features are continuous variables. The “positive” class (heart disease) contains 164 points and the negative class (no heart disease) contains 139 points. In the data set, the “positive” class is

denoted as 1 while the “negative” class is denoted as 0. Therefore, the input space is of dimension 13, and the output is either 1 or 0.

The BUPA medical liver disorder data set was also originally used for pattern classification to detect the liver disorders caused by excessive alcohol consumption. It contains 345 points, each point consisting of 6 features. All features are continuous variables. The “positive” class (liver disorders) contains 145 points; the “negative” class (no liver disorders) contains 200 points. In the data set, the “positive” class is denoted as 1, the “negative” class is denoted as 0. Therefore, the input space is of dimension 6, and the output is either 1 or 0.

In both data sets, since the outputs are discrete, these data sets are more difficult to approximate.

For the heart disease example, the approximation results for different activation functions and center selection methods are depicted in Figures 6-6, 6-7, 6-8, and 6-9, respectively. For the liver disorder example, the approximation results are depicted in Figures 6-10, 6-11, 6-12, and 6-13, respectively. The results are basically the same as before, i.e., the Mean Squared Errors decreases to zero as the number of hidden nodes increases. And as we can see from the figures, the speeds of convergence of these two different RBF networks are also about the same, even though the one RBF is not integrable. This further demonstrates the validity of our results.

Similar results have been obtained for other examples, including the data sets for Ionosphere [76] and Wisconsin Breast Cancer [76]. Due to the space limit, we do not present all of them in here.

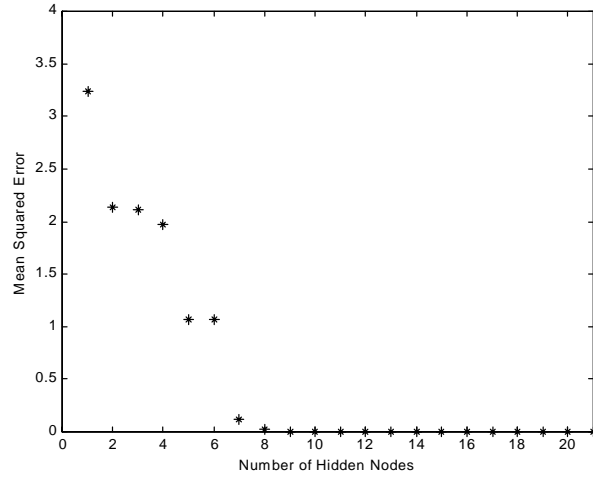


Figure 6-2: The approximation result for the one-dimensional example with activation function $g(x) = \exp\left(\frac{\|x-\theta\|}{b}\right)$, trained with RS.

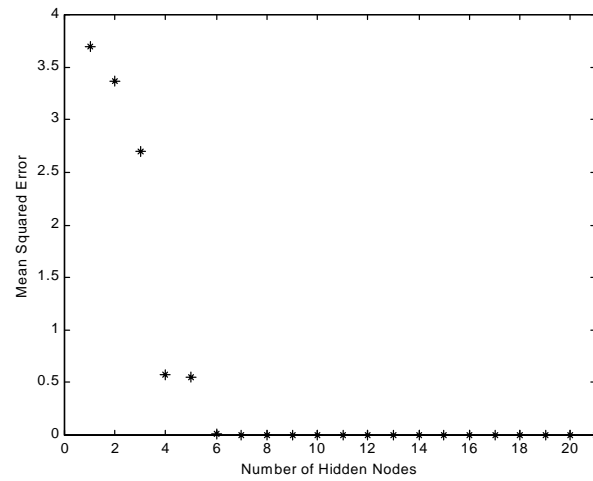


Figure 6-3: The approximation result for the one-dimensional example with activation function $g(x) = \exp\left(-\frac{\|x-\theta\|}{b}\right)$, trained with RS.

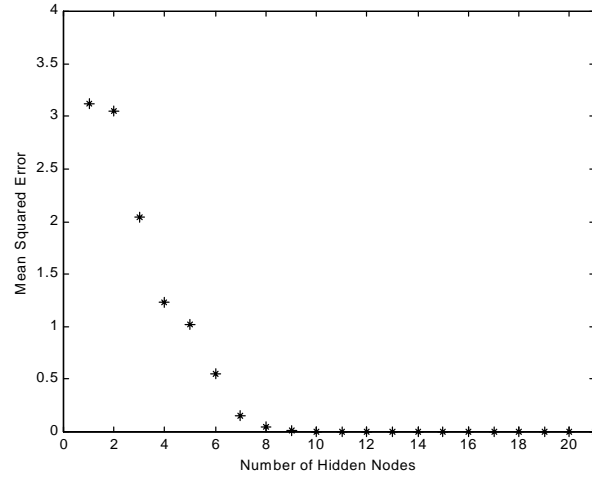


Figure 6-4: The approximation result for the one-dimensional example with activation function $g(x) = \exp\left(\frac{\|x-\theta\|}{b}\right)$, trained with OLS.

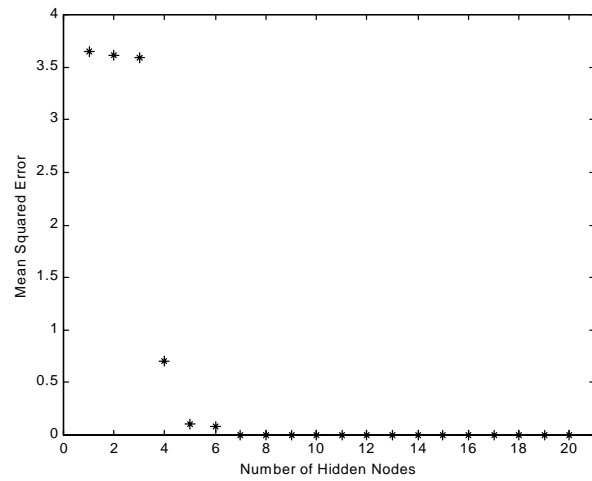


Figure 6-5: The approximation result for the one-dimensional example with activation function $g(x) = \exp\left(-\frac{\|x-\theta\|}{b}\right)$, trained with OLS.

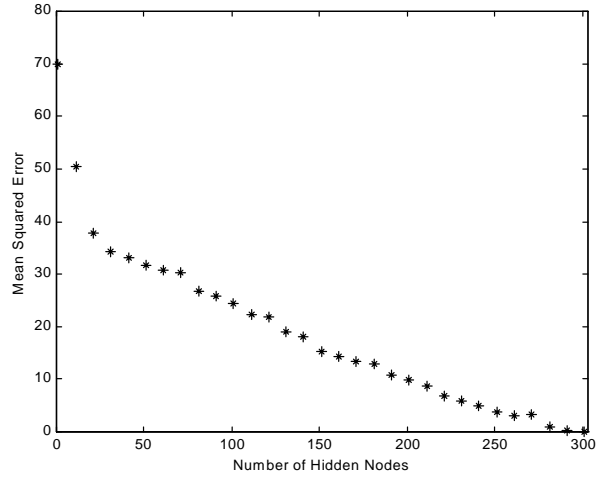


Figure 6-6: The approximation result for the heart disease example with activation function $g(x) = \exp\left(\frac{\|x-\theta\|}{b}\right)$, trained with RS.

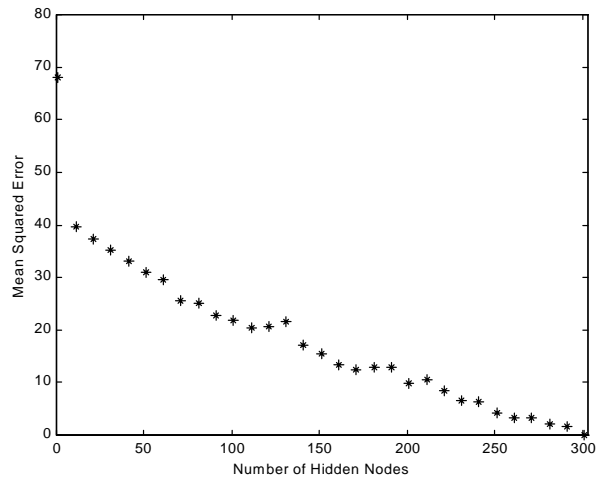


Figure 6-7: The approximation result for the heart disease example with activation function $g(x) = \exp\left(-\frac{\|x-\theta\|}{b}\right)$, trained with RS.

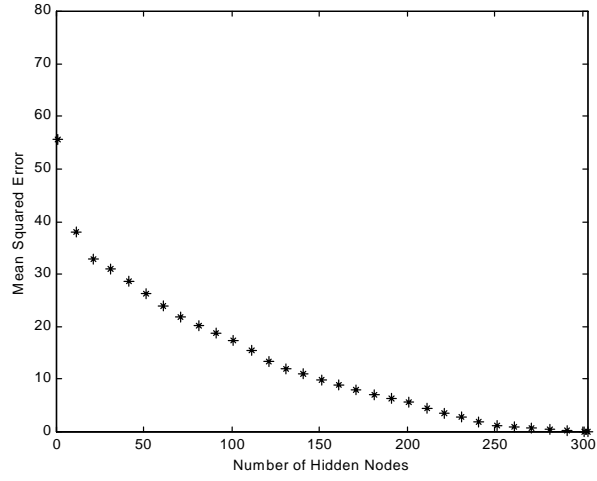


Figure 6-8: The approximation result for the heart disease example with activation function $g(x) = \exp\left(\frac{\|x-\theta\|}{b}\right)$, trained with OLS.

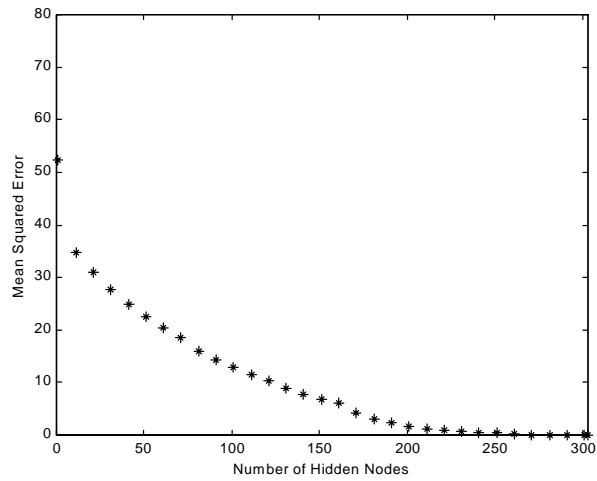


Figure 6-9: The approximation result for the heart disease example with activation function $g(x) = \exp\left(-\frac{\|x-\theta\|}{b}\right)$, trained with OLS.

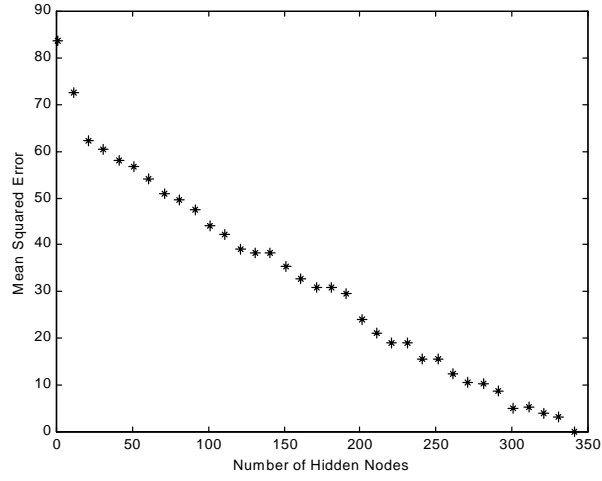


Figure 6-10: The approximation result for the liver disorder example with activation function $g(x) = \exp\left(\frac{\|x-\theta\|}{b}\right)$, trained with RS.

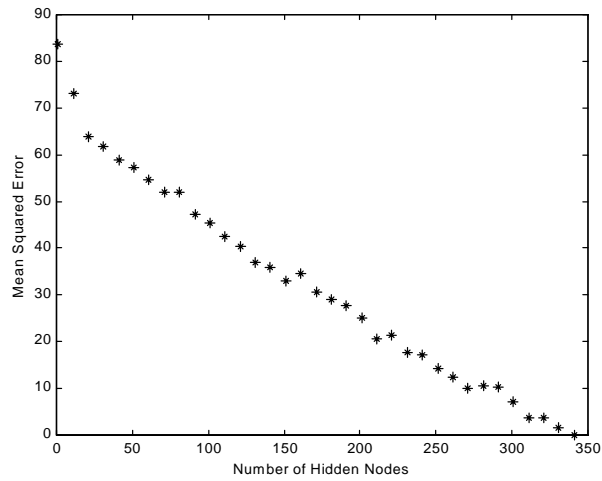


Figure 6-11: The approximation result for the liver disorder example with activation function $g(x) = \exp\left(-\frac{\|x-\theta\|}{b}\right)$, trained with RS.

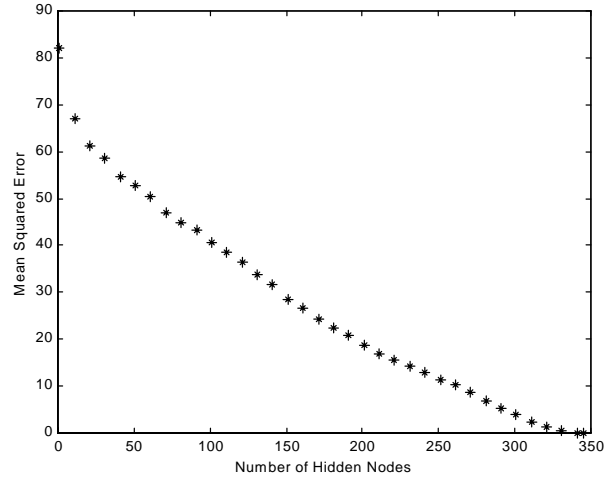


Figure 6-12: The approximation result for the liver disorder example with activation function $g(x) = \exp\left(\frac{\|x-\theta\|}{b}\right)$, trained with OLS.

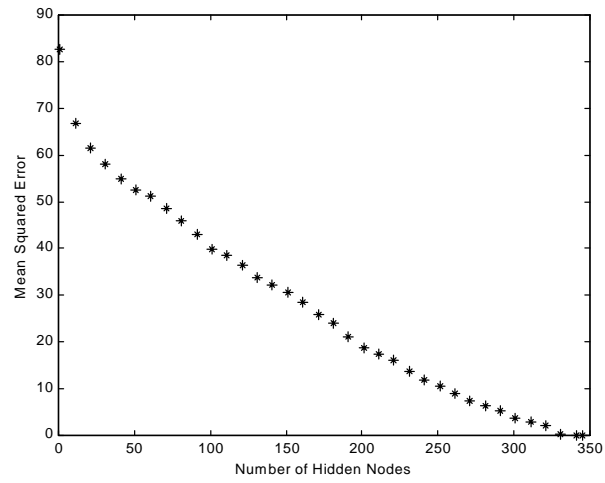


Figure 6-13: The approximation result for the liver disorder example with activation function $g(x) = \exp\left(-\frac{\|x-\theta\|}{b}\right)$, trained with OLS.

Chapter 7

Conclusion and Future Research

7.1 Conclusion

In this dissertation, we have proposed a new neural network model which has bounded weights for the links between hidden nodes and output nodes, and which is trained using the least square error as the optimization criterion. As with the SVM model, the training procedure for the proposed *Bounded Neural Network* (BNN) guarantees a globally optimal set of weights since the training procedure is formulated as a convex quadratic programming problem. This is in contrast to the Multi-Layer Perceptrons (MLP) model for which training usually produces a locally optimal set of weights.

The weights specification problem for the BNN model has a similar formulation to that of the SVM model. They are both formulated as a convex quadratic programming problem with some bound constraints. The major difference lies in the objective functions. For the SVM model, the objective function is a weighted combination of the “large margin” and the “perceptron criterion”. For the BNN model, the objective function is the *least squared error*, which is often adopted in the MLP model. As reported in Chapter 4, the BNN model has comparable performance with the SVM model. Furthermore, the BNN model has the following advantages over the SVM model:

1. The kernel functions need not satisfy Mercer’s condition, which is one of the key requirements for the SVM model. Therefore, the BNN model provides greater

flexibility in choice of the kernel functions, which might provide more potential learning capability.

2. The BNN model can handle multi-class classification problems. In contrast, the SVM model is basically a two-class classifier. This is considered as one of the major drawbacks of the SVM model.
3. The formulation of the weight specification problem of the BNN model does not have the equality constraint ($\mathbf{y}^T \boldsymbol{\alpha} = 0$) as in the SVM model. Therefore, various algorithms designed for solving the bound-constrained convex programming problems are readily applicable.

Also in this dissertation, we have studied the universal approximation property of three-layered radial-basis function networks. We have shown that the integrability property usually imposed on the activation functions is not necessary. Moreover, we have shown that a radial basis function network can be a universal approximator in the continuous function space, if the activation function used in the hidden layer is continuous almost everywhere, locally essentially bounded, and not a polynomial. Furthermore, for the universal approximation in $L^p(\mu)$ space, with $1 \leq p < \infty$ and μ being a finite measure, we only need the activation function used in the hidden layer to be essentially bounded and not a polynomial. The experimental results support our theoretical findings.

7.2 Future Research

We propose the following future research directions.

1. Further relaxed conditions. Can we further relax the conditions on the ridge or the radial-basis activation functions, such that the neural networks are still universal approximators?
2. Necessary conditions. For both ridge and radial-basis activation functions,

“non-polynomial” is clearly a necessary condition for the three-layered feedforward neural networks to be universal approximators. But are there any other necessary conditions? And what are they?

3. Bounded weights. The “real” neural networks can successfully fulfill many tasks. It seems that they are universal approximators. But in the real neural networks, the weights for the links between neurons are always limited to a certain range. Therefore, it seems that the neural networks with bounded weights can be universal approximators. Is this true for the “artificial” neural networks? If it is true, how to prove it?
4. Constructive proof. Till now, almost all methods used to prove neural networks are universal approximators are of “existence” type. That is, they only proved that neural networks are capable of universal approximation, but they did not mention how to construct a neural network to do universal approximation. It will be much helpful if there is a constructive proof, such that from the proof we can derive some algorithms to construct a neural network to do universal approximation. Also from the proof, we may learn how many hidden nodes are needed to achieve a given approximation accuracy.

Bibliography

- [1] M. Aizerman, E. Braverman, and L. Rozonoer, Theoretical foundations of the potential function method in pattern recognition learning, *Automation and Remote Control*, Vol. 25, pp. 821-837, 1964.
- [2] J. J. Anderson, Normal mixtures and the number of clusters problem, *Computat. Statist. Quarterly*, Vol. 2, pp. 3-14, 1985.
- [3] G. A. Babich and O. I. Camps, Weighted Parzen windows for pattern classification, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 18, No. 5, pp. 567-570, 1996.
- [4] Thomas Bayes, An essay towards solving a problem in the doctrine of chances, *Philosophical Transactions of the Royal Society* (London), Vol. 53, pp. 370-418, 1763.
- [5] C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford, 1995.
- [6] V. Blanz, B. Scholkopf, H. Bülthoff, C. Burges, V. Vapnik, and T. Vetter, Comparison of view-based object recognition algorithms using realistic 3d models, In C. von der Malsburg, W. von Seelen, J. C. Vorbruggen, and B. Sendhoff, editors, *Artificial Neural Networks - ICANN'96*, pp. 251-256, Berlin, 1996, Springer Lecture Notes in Computer Science, Vol. 1112.
- [7] B. E. Boser, I. Guyon, and V. Vapnik, A training algorithm for optimal margin

- classifiers, In David Haussler, editor, *Proceedings of the 4th Workshop on Computational Learning Theory*, pp. 144-152, ACM Press, San Mateo, CA, 1992.
- [8] H. Bozdogan, Choosing the number of component clusters in the mixture-model using a new informational complexity criterion of the inverse-Fisher information matrix. In *Proc. Third Conference of IFCS*, Paris, 1992, IFCS.
 - [9] L. Breiman and J. H. Friedman, Estimating optimal transformations for multiple regression and correlation, *Journal of the American Statistical Association*, Vol. 80, pp.580-619, 1985.
 - [10] L. Breiman, Hinging hyperplanes for regression, classification and function approximation, *IEEE Transactions on Information Theory*, Vol. 39, No. 3, pp. 999-1013, 1993.
 - [11] D. S. Broomhead and D. Lowe, Multi-variable functional interpolation and adaptive networks, *Complex Systems*, Vol. 2, No. 3, pp. 269-303, 1998.
 - [12] C. J. C. Burges and B. Scholkopf, Improving the accuracy and speed of support vector learning machines, In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, Vol. 9, pp. 375-381, MIT Press, Cambridge, MA, 1997.
 - [13] G. Celex and G. Soromenho, An entropy criterion for assessing the number of clusters in a mixture model, *Journal of Classification*, Vol. 13, pp. 195-212, 1996.
 - [14] C.-C. Chang and C.-J. Lin, LIBSVM : a library for support vector machines, URL: <http://www.csie.ntu.edu.tw/~cjlin/libsvm>, 2001.
 - [15] S. Chen, E. S. Chng, and K. Alkadhim, Regularized orthogonal least squares algorithm for constructing radial basis function networks, *International Journal of Control*, Vol. 64, No. 5, pp. 829-837, 1996.

- [16] S. Chen, C. F. N. Cowan, and P. M. Grant, Orthogonal least squares learning algorithm for radial basis function networks, *IEEE Transactions on Neural Networks*, Vol. 2, No. 2, pp. 302-309, 1991.
- [17] S. Chen, P. M. Grant, and C. F. N. Cowan, Orthogonal least-squares algorithm for training multioutput radial basis function networks, *IEE Proceedings, Part F*, Vol. 139, No. 6, pp. 378-384, 1992.
- [18] T. Chen, H. Clien, and R. Liu, Approximation capability in $C(\overline{R}_n)$ by multi-layer feedforward networks and related problems, *IEEE Transactions on Neural Networks*, Vol. 6, No. 1, pp. 25-30.
- [19] T. Chen and H. Chen, Approximation capability to functions of several variables, nonlinear functionals, and operators by radial basis function neural networks, *IEEE Transactions on Neural Networks*, Vol. 6, No. 4, pp. 904-910, 1995.
- [20] T. Chen and H. Chen, Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, Vol. 6, No. 4, pp. 911-917, 1995.
- [21] C. Chui and X. Li, Approximation by ridge functions and neural networks with one hidden layer, *Journal of Approximation Theory*, Vol. 70, pp. 131-141.
- [22] C. Cortes and V. Vapnik, Support vector networks, *Machine Learning*, Vol. 20, pp. 273-297, 1995.
- [23] Y. L. Cun, J. S. Denker, and S. A. Solla, Optimal Brain Damage, In David S. Touretzky, editor, *Advances in Neural Information Processing Systems*, Vol. 2, pp. 589-605, Morgan Kaufmann, San Mateo, CA, 1990.
- [24] G. Cybenko, Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, Vol. 3, pp. 303-314, 1989.

- [25] A. P. Dempster, N. M. Laird, and D. B. Rubin, Maximum-likelihood from incomplete data via the EM algorithm (with discussion), *Journal of the Royal Statistical Society, Series B*, Vol. 39, pp. 1-38, 1977.
- [26] H. Drucker, C. J. C. Burges, L. Kaufman, A. Smola, and V. Vapnik, Support vector regression machines, *Advances in Neural Information Processing Systems*, Vol. 9, pp. 155-161, 1997.
- [27] R. Duda and P. Hart, *Pattern Classification and Scene Analysis*, Wiley, New York, 1973.
- [28] R. Duda, P. Hart, and D. Stork, *Pattern Classification (Second Edition)*, Wiley, New York, 2000.
- [29] B. S. Everitt, A Monte Carlo investigation of the likelihood ratio test for the number of components in a mixture of normal distributions, *Multivariate Behavioral Research*, Vol. 16, pp. 171-180, 1981.
- [30] R. A. Fisher, The use of multiple measurements in taxonomic problems, *Annals of Eugenics*, Vol.7, Part II, pp. 179-188, 1936.
- [31] E. Fix and J. L. Hodges, Jr., Discriminatory analysis: nonparametric discrimination: consistency properties, *USAF School of Aviation Medicine*, Vol. 4, pp. 261-279, 1951.
- [32] E. Fix and J. L. Hodges, Jr., Discriminatory analysis: nonparametric discrimination: small sample performance. *USAF School of Aviation Medicine*, Vol. 11, pp. 280-322, 1952.
- [33] M. Flasiński and G. Lewicki, The convergent method of constructing polynomial discriminant functions for pattern recognition, *Pattern Recognition*, Vol. 24, No. 10, pp. 1009-1015, 1991.
- [34] A. Friedman, *Generalized Functions and Partial Differential Equations*. Englewood Cliffs, N. J., Prentice-Hall, 1963.

- [35] J. H. Friedman, J. W. Tukey, A projection pursuit algorithm for exploratory data analysis, *IEEE Transactions on Computers*, Vol. 23, No. 9, pp. 881-889, 1974.
- [36] A. Friedman, *Fondations of Modern Analysis*, New York: Dover Publications, 1982.
- [37] J. H. Friedman. Regularized discriminant analysis. *Journal of the American Statistical Association*, Vol. 84, pp. 165-175, 1989.
- [38] J. H. Friedman, Multivariate adaptive regression splines, *Annals of Statistics*, Vol. 19, No. 1, pp. 1-141, 1991.
- [39] K. Fukunaga and P. M. Narendra, A branch and bound algorithm for computing k-nearest neighbors, *IEEE Transactions on Computers*, Vol. 24, pp. 750-753, 1975.
- [40] K. Fukunaga and R. R. Hayes, The reduced Parzen classifier, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11, No. 4, pp. 423-425, 1989.
- [41] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, Inc., London, second edition, 1990.
- [42] P. J. Green and B. W. Silverman, *Nonlinear Regression and Generalized Linear Models: A Roughness Penalty Approach*, Chapman and Hall, London, 1994.
- [43] D. J. Hand and B. G. Batchelor, An edited condensed nearest neighbor rule, *Information Sciences*, Vol. 14, pp. 171-180, 1978.
- [44] P. E. Hart, The condensed nearest neighbor rule, *IEEE Transactions on Information Theory*, Vol. 14, pp. 515-516, 1968.
- [45] B. Hassibi and D. G. Stork, Second-order derivatives for network pruning: Optimal Brain Surgeon, In Stephen J. Hanson, Jack D. Cowan, and C. Lee Giles,

- editors, *Advances in Neural Information Processing Systems*, Vol. 5, pp. 164-171, Morgan Kaufmann, San Mateo, CA, 1993.
- [46] B. Hassibi, D. G. Stork, and Greg Wolff, Optimal Brain Surgeon and general network pruning, In *Proceedings of the International Conference on Neural Networks*, Vol. 1, pp. 293-299, IEEE, San Francisco, CA, 1993.
 - [47] R., Hecht-Nielsen, Kolmogorov's mapping neural network existence theorem, In *International Joint Conference on Neural Networks*, Vol. 3, Washington, DC: IEEE, pp. 11-14.
 - [48] S. Helgason, *The Radon Transform*, Basel, Switzerland: Birkhauser, 1980.
 - [49] G. E. Hinton, Learning distributed representations of concepts, In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pp. 1-12, Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.
 - [50] Y.-C. Ho and R. L. Kashyap, An algorithm for linear inequalities and its applications, *IEEE Transactions on Electronic Computers*, Vol. 14, pp. 683-688.
 - [51] K. Hornik, M. Stinchcombe, and H. White, Approximation capabilities of multilayer feedforward networks, *Neural Networks*, Vol. 4, pp. 251-257, 1989.
 - [52] K. Hornik, Approximation capabilities of multilayer feedforward neural networks. *Neural Networks*, Vol. 4, pp. 251-257, 1991.
 - [53] K. Hornik, Some new results on neural network approximation, *Neural Networks*, Vol. 6, pp. 1069-1072, 1993.
 - [54] J.-N. Hwang, S.-R. Lay, M. Maechler, D. Martin, and J. Schimert, Regression modeling in back-propagation and projection pursuit learning, *IEEE Transactions on Neural Networks*, Vol. 5, No. 3, pp. 342-353, 1994.
 - [55] M. Jamshidian and R. I. Jennrich, Conjugate gradient acceleration of the EM algorithm, *Journal of the American Statistical Association*, Vol. 88, pp. 221-228, 1993.

- [56] T. Joachims, Text categorization with support vector machines, Technical report, LS VIII Number 23, University of Dortmund, 1997, <ftp://ftp-ai.informatik.uni-dortmund.de/pub/Reports/report23.ps.Z>.
- [57] T. Joachims, Making large-scale SVM learning practical, In B. Scholkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, pp. 169-184, MIT Press, 1999.
- [58] M. C. Jones, I. J. Marron, and S. J. Sheather, A brief survey of bandwidth selection for density estimation, *Journal of the American Statistical Association*, Vol. 91, pp. 401-407, 1996.
- [59] N. B. Karayiannis and G. M. Mi, Growing radial basis neural networks, merging supervised and unsupervised learning with network growth techniques, *IEEE Transactions on Neural Networks*, Vol. 8, No. 6, pp. 1492-1506, 1997.
- [60] T. Kohonen, *Self-Organizing Maps*, BerlinL Springer, 1995.
- [61] J. B. Kruskal, Linear transformation of multivariate data to reveal clustering, In R. N. Shepard, A. K. Romney, and S. B. Nerlove, editors, *Multidimensional Scaling: Theory and Applications in the Behavioral Sciences*, Vol. 1, pp. 179-191, Seminar Press, London, 1972.
- [62] M. Leshno, V. Lin, A. Pinkus, and S. Shochen, Multilayer feedforward networks with a polynomial activation function can approximate any function. *Neural Networks*, Vol. 6, pp. 861-867, 1993.
- [63] C.-J. Lin and J. J. More, Newton's method for large bound-constrained optimization problems, *SIAM Journal of Optimization*, Vol. 9, No. 4, pp. 1100-1127, 1999.
- [64] B. G. Lindsay and P. Basak, Multivariate normal mixtures: a fast consistent method of moments, *Journal of the American Statistical Association*, Vol. 88, pp. 468-476, 1993.

- [65] C. G. Looney, *Pattern Recognition Using Neural Networks: Theory and Algorithms for Engineers and Scientists*, Oxford University Press, 1997.
- [66] D. Lowe, Radial basis function networks, In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pp. 779-782, MIT Press, Cambridge, MA, 1995.
- [67] J. S. Marron, Automatic smoothing parameter selection: a survey, *Empirical Economics*, Vol. 13, pp. 187-208, 1988.
- [68] J. Mercer, Functions of positive and negative type and their connection with the theory of integral equations, *Philos. Trans. Roy. Soc. London*, series A, Vol. 209, pp. 415-446, 1909.
- [69] C. A. Micchelli, Interpolation of scattered data: distance matrices and conditionally positive definite matrices, *Constr. Approx.*, Vol. 2, pp. 11-22, 1986.
- [70] S. Mika, G. Rätsch, J. Weston, B. Schölkopf, and K.-R. Müller, Fisher discriminant analysis with kernels. In Y.-H. Hu, J. Larsen, E. Wilson, and S. Douglas, editors, *Neural Networks for Signal Processing IX*, pp. 41-48, IEEE, 1999.
- [71] M. L. Minsky and S. A. Papert, *Perceptron: An Introduction to Computational Geometry*, MIT Press, Cambridge, MA, 1968.
- [72] A. Mkhadri, G. Celeux, and A. Nasroallah. Regularization in discriminant analysis: an overview, *Computational Statistics and Data Analysis*, Vol. 23, pp. 403-423, 1997.
- [73] J. Moody and C. J. Darken, Fast learning in networks of locally-tuned processing units, *Neural Computation*, Vol. 1, pp. 281-294.
- [74] J. J. More and G. Toraldo, On the solution of large quadratic programming problems with bound constraints, *SIAM Journal of Optimization*, Vol. 1, No. 1, pp. 93-113, 1991.

- [75] K.-R. Muller, A. Smola, G. Ratsch, B. Scholkopf, J. Kohlmorgen, and V. Vapnik, Predicting time series with support vector machines, In *Proceedings, International Conference on Artificial Neural Networks*, pp. 999, Springer Lecture Notes in Computer Science, 1997.
- [76] P. M. Murphy and D.W. Aha, UCI repository of machine learning databases, 1992, URL: <http://www.ics.uci.edu/~mlearn/MLRepository.html>.
- [77] M. T. Musavi, W. Ahmed, K. H. Chan, K. B. Faris, and D. M. Hummels, On the training of radial basis function classifiers, *Neural Networks*, Vol. 5, pp. 595-603, 1992.
- [78] Q. Ni and Y. Yuan, A subspace limited memory quasi-Newton algorithm for large-scale nonlinear bound constrained optimization, *Mathematics of Computation*, Vol. 66, No. 220, pp. 1509-1520, 1997.
- [79] N. J. Nilsson, *Learning Machines, Foundations of Trainable Pattern Classifying Systems*, McGraw-Hill, New York, 1965.
- [80] M. J. L. Orr, Regularization in the selection of radial basis function centers. *Neural Computation*, Vol. 7, pp. 606-623, 1995.
- [81] E. Parzen, On estimation of a probability density function and mode, *Annals of Mathematical Statistics*, Vol. 33, No. 2, pp. 1065-1076, 1962.
- [82] E. A. Patrick and F. P. Fischer, III, A generalized k-nearest neighbor rule, *Information and Control*, Vol. 16, No. 2, pp. 128-152, 1970.
- [83] M. J. D. Powell, Radial basis functions for multivariable interpolation: a review, In J. C. Mason and M. G. Cox, editors, *Algorithms for Approximation*, pp. 143-167, Clarendon Press, Oxford, 1987.
- [84] J. Park and I. W. Sandberg, Universal approximation using radial-basis-function networks, *Neural Computation*, Vol. 3, pp. 246-257, 1991.

- [85] J. Park and I. W. Sandberg, Approximation and radial-basis-function networks, *Neural Computation*, Vol. 5, pp. 305-316, 1993.
- [86] J. Platt, Fast training of support vector machines using sequential minimal optimization, In B. Scholkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, pp. 185-208, MIT Press, 1999.
- [87] S. Richardson and P. Green, On Bayesian analysis of mixtures with unknown number of components, *Journal of Royal Statistical Society, Series B*, Vol. 59, pp. 731-792, 1997.
- [88] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, Spartan Books, Washington, D.C., 1962.
- [89] W. Rudin, *Real and Complex Analysis* (3rd edition). New York : McGraw-Hill, 1987.
- [90] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Learning internal representation by error propagation, In D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1, pp. 318-362, MIT Press, Cambridge, MA, 1986.
- [91] S. Saitoh, *Theory of Reproducing Kernels and its Applications*, Longman Scientific & Technical, Harlow, England, 1988.
- [92] B. Scholkopf, A. Smola, and K.-R. Muller, Nonlinear component analysis as a kernel eigenvalue problem, *Neural Computation*, 1998.
- [93] P. Simard, Y. L. Cun, and J. Denker, Efficient pattern recognition using a new transformation distance, In Stephen J. Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems*, Vol. 5, pp. 50-58, Morgan Kaufmann, San Mateo, CA, 1993.

- [94] B. W. Silverman, Kernel density estimation using the fast Fourier transform, *Applied Statistics*, Vol. 31, pp. 93-99, 1982.
- [95] S. R. Searle, *Matrix Algebra Useful for Statistics*, John Wiley & Sons, 1982.
- [96] D. F. Specht, Generation of polynomial discriminant functions for pattern recognition, *IEEE Transactions on Electron. Comput.*, Vol. 16, 308-319, 1967.
- [97] D. A. Sprecher, On the structure of continuous functions of several variables, *Transactions of the American Math Society*, Vol. 115, pp. 340-355.
- [98] J. A. K. Suykens and J. Vandewalle, Multiclass least squares support vector machines, In *IJCNN'99 International Joint Conference on Neural Networks*, Washington, DC, 1999.
- [99] V. Vapnik, *Estimation of Dependence Based on Empirical Data [in Russian]*, Nauka, Moscow, 1979. (English translation: Springer-Verlag, New York, 1982).
- [100] V. Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag, New York, 1995.
- [101] V. Vapnik, *Statistical Learning Theory*, John Wiley and Sons, Inc., New York, 1998.
- [102] A. R. Webb, Functional approximation in feed-forward networks: A least-squares approach to generalization, *IEEE Transactions on Neural Networks*, Vol. 5, No. 3, pp. 363-371.
- [103] J. Weston, A. Gammerman, M. O. Stitson, V. Vapnik, V. Vovk, and C. Watkins, Density estimation using support vector machines, Technical Report, Royal Holloway College, Report number CSD-TR-97-23, 1997.
- [104] B. Widrow and M. E. Hoff, Adaptive switching circuits, *IRE WESCON Convention Record*, Vol. 4, pp. 94-104, 1960, Reprinted in Anderson & Rosenfeld (1988).

- [105] J. H. Wolfe, A Monte Carlo study of the sampling distribution of the likelihood ratio for mixtures of multinormal distributions, *Technical Bulletin STB 72-7*, Navel Personnel and Training Research Laboratory, San Diego, CA, 92152, 1971.
- [106] P. Wu, *Neural Networks and Fuzzy Contol with Applications to Textile Manufacturing and Management*, Ph. D. Dissertation, Operations Research Program, North Carolina State University, 1997.
- [107] A. H. Zemanian, *Distribution Theory and Transform Analysis; An Introduction to Generalized Functions, with Applications*, New York, McGraw-Hill, 1965.