國 立 交 通 大 學

電機與控制工程學系

博 士 論 文

支持向量模糊類神經網路及其在資料分類和函數近似之應用

Support-Vector based Fuzzy Neural Networks and its Applications to Pattern Classification and Function Approximation

研 究 生：葉 長 茂

指導教授：林 進 燈

中 華 民 國 九 十 六 年 一 月

支持向量模糊類神經網路及其在資料分類和函數近似之應用

Support-Vector based Fuzzy Neural Networks and its Applications to Pattern Classification and Function Approximation

研 究 生：葉長茂　　Student：Chang-Mao Yeh

指導教授：林進燈 博士　　Advisor：Dr. Chin-Teng Lin

國 立 交 通 大 學

電 機 與 控 制 工 程 學 系

博 士 論 文

A Dissertation
Submitted to Department of Electrical and Control Engineering
College of Electrical Engineering and Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy
in
Electrical and Control Engineering
September 2006
Hsinchu, Taiwan, Republic of China

中華民國九十六年一月

支持向量模糊類神經網路及其在資料分類和函數近似之應用

摘　　要

模糊類神經網路經常使用倒傳遞學習演算法或分群學習演算法學習調整模糊規則和歸屬函數的參數以解決資料分類和函數回歸等問題，但是此學習演算法經常不能將訓練誤差及預測誤差同時地最小化，這將造成在資料分類之預測階段無法達到最好的分類效能，且對含有雜訊的訓練資料進行回歸近似時，常有過度訓練而造成回歸效能大大降低的問題。

本論文結合支持向量學習機制與模糊類神經網路的優點，提出一個新的支持向量模糊類神經網路(SVFNNs)，此 SVFNNs 將高維度空間具有極優越分類能力的支持向量機(SVM)和極優越強健抗雜訊能力的支持向量回歸(SVR)與能夠有效處理不確定環境資訊的類似人類思考的模糊類神經網路之優點結合。首先我們提出一個適應模糊核心函數(adaptive fuzzy kernel)，進行模糊法則建構，此模糊核心函數滿足支持向量學習所須之默塞爾定理(Mercer's theorem)， SVFNNs 的學習演算法有參個學習階段，在第一個階段，藉由分群原理自動產生模糊規則和歸屬函數，在第二階段，利用具有適應模糊核心函數之 SVM 和 SVR 來計算模糊神經網路的參數，最後在第三階段，透過降低模糊規則的方法來移除不重要的模糊規則。我們將 SVFNNs 應用到 Iris、Vehicle、Dna、Satimage、Ijcnn1 五個資料集和兩個單變數及雙變數函數進行資料分類與函數近似應用，實驗結果顯示我們提出的 SVFNNs 能在使用較少的模糊規則下有很好的概化(generalization)之資料分類效能和強健抗雜訊的函數近似效能。

Support-Vector based Fuzzy Neural Networks and its Applications to Pattern Classification and Function Approximation

Student: Chang-Mao Yeh　　　　　　　　Advisor: Chin-Teng Lin

Department of Electrical and Control Engineering

National Chiao-Tung University

Abstract

Fuzzy neural networks (FNNs) have been proposed and successfully applied to solving these problems such as classification, identification, control, pattern recognition, and image processing, etc. Fuzzy neural networks usually use the backpropagation or C-cluster type learning algorithms to learn the parameters of the fuzzy rules and membership functions from the training data. However, such learning algorithm only aims at minimizing the training error, and it cannot guarantee the lowest testing error rate in the testing phase. In addition, the local solutions and slow convergence often impose practical constraints in the function approximation problems

In this dissertation, novel fuzzy neural networks combining with support vector learning mechanism called support-vector based fuzzy neural networks (SVFNNs) are proposed for pattern classification and function approximation. The SVFNNs combine the capability of minimizing the empirical risk (training error) and expected risk (testing error) of support vector learning in high dimensional data spaces and the efficient human-like reasoning of FNN in handling uncertainty information. First, we propose a novel adaptive fuzzy kernel, which has been proven to be a Mercer kernel, to construct initial fuzzy rules. A learning algorithm consisting of three learning phases is developed to construct the SVFNNs and train the parameters. In the first phase, the fuzzy rules and membership functions are automatically determined by the clustering principle. In the second phase, the parameters of FNN are calculated by the SVM and SVR with the proposed adaptive fuzzy kernel function for pattern classification and function approximation, respectively. In the third phase, the relevant fuzzy rules are selected by the proposed fuzzy rule reduction method. To investigate the effectiveness of the proposed SVFNNs, they are applied to the Iris、Vehicle、Dna、Satimage and Ijcnn1 datasets for classification, and one- and two- variable functions for approximation, respectively. Experimental results show that the proposed SVFNNs can achieve good pattern classification and function approximation performance with drastically reduced number of fuzzy kernel functions (fuzzy rules).

誌　　謝

首先感謝指導教授林進燈院長多年來的指導。在學術上，林教授以其深厚的學識涵養及孜孜不倦的教導，使我在博士研究期間，學到了許多寶貴的知識、研究的態度及解決問題的能力。除了學術上的收穫之外，也相當感謝林教授在日常生活上給予的關心與鼓勵，每當我在最徬徨的時侯，總能指點我正確的方向，在我最失意的時侯，讓我重拾信心，另外林教授的學識淵博、做事衝勁十足、熱心待人誠懇與風趣幽默等特質，都是我非常值得學習的地方。今日之所以能夠順利完成學業，都是老師悉心的教導，在此對林教授獻上最誠摯的敬意。此外亦非常感謝諸位口試委員在論文上所給于的寶貴建議與指教，使得本論文更加完備。

在家人方面，首先感謝我的父親葉哲元先生與母親林寶葉女士從小到大對我的教養與栽培，由於您們長年以來的支持與鼓勵，使得我無後顧之憂的專心於學業方面。及感謝我的哥哥長青與妹妹蕙蘭平日對我的關心。同時也感謝我的妻小：維珊和子鴻和芸芸；多年來因為學業的關係，很少陪伴你們，日後當會好好的補償你們。

在學校方面，感謝勝富和翊方學長在研究與生活的熱心幫忙，這段一起研究討論的日子，不但讓我學到研究方法，也讓我學到解決問題的能力。及感謝實驗室裡每位學長、學弟及同學在學業及日常的照顧，這段一起唸書、閒話家常的日子，不僅讓我總能抱著愉快的心情學習，更讓我在這求學過程多采多姿。

在工作方面，感謝學校內長官與同事多年來的支持，得以使我能夠順利完成本論文。

最後，謹以本論文獻給我的家人與關心我的師長與朋友們。

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

It is an important key issue in many scientific and engineering fields to classify the acquired data or estimate an unknown function from a set of input-output data pairs. As is widely known, fuzzy neural networks (FNNs) have been proposed and successfully applied to solving these problems such as classification, identification, control, pattern recognition, and image processing. most previous researches issue the method of automatically generating fuzzy rules from numerical data and use the backpropagation (BP) and/or C-cluster type learning algorithms to train parameters of fuzzy rules and membership functions from the training data. However, such learning algorithm only aims at minimizing the training error, and it cannot guarantee the lowest testing error rate in the testing phase. In addition, the local solutions and slow convergence often impose practical constraints in the function approximation problems. Therefore, it is desired to develop a novel FNNs, that achieve good pattern classification and function approximation performance with drastically reduced number of fuzzy kernel functions (fuzzy rules).

1.1 Fuzzy Neural Network

Both fuzzy logic and neural networks are aimed at exploiting human-like knowledge processing capability. The fuzzy logic system using linguistic information can model the qualitative aspects of human knowledge and reasoning processes without employing precise quantitative analyses [1]. However, the selection of fuzzy if-then rules often conventionally relies on a substantial amount of heuristic observation to express proper strategy's knowledge. Obviously, it is difficult for human experts to examine all the input-output data to find a number of proper rules for the fuzzy system. Artificial neural networks are efficient computing models which have shown their strengths in solving hard problems in artificial intelligence. The neural networks are a popular generation of information processing systems that demonstrate the ability to learn from training data [2]. However, one of the major criticisms is their being black boxes, since no satisfactory explanation of their behavior has been offered. This is a significant weakness, for without the ability to produce comprehensible decision, it is hard to trust the reliability of networks addressing real-world problems. Much research has been done on fuzzy neural networks (FNNs), which combine the capability of fuzzy reasoning in handling uncertain information and the capability of neural networks in learning from processes [3]-[5]. Fuzzy neural networks are very effective in solving actual problems described by numerical examples of an unknown process. They have been successfully applied to classification, identification, control, pattern recognition, and image processing, *etc*. In particular, many learning algorithms of fuzzy (neural) have been presented and applied in pattern classification and decision-making systems [6], [7]. Moreover, several researchers have investigated the fuzzy-rule-based methods for function approximation and pattern classification [8]-[15].

A fuzzy system consists of a bunch of fuzzy if-then rules. Conventionally, the selection of fuzzy if-then rules often relies on a substantial amount of heuristic observation to express proper strategy's knowledge. Obviously, it is difficult for human experts to examine all the input-output data to find a number of proper rules for the fuzzy system. Most pre-researches used the backpropagation (BP) and/or C-cluster type learning algorithms to train parameters of fuzzy rules and membership functions from the training data [16], [17]. However, such learning only aims at minimizing the classification error in the training phase, and it cannot guarantee the lowest error rate in the testing phase. Therefore we apply the support vector mechanism with the superior classification power into learning phase of FNN to tackle these problems.

1.2 Support Vector Machine

Support vector machines (SVM) has been revealed to be very effective for general-purpose pattern classification [18]. The SVM performs structural risk minimization and creates a classifier with minimized VC dimension. As the VC dimension is low, the expected probability of error is low to ensure a good generalization. The SVM keeps the training error fixed while minimizing the confidence interval. So, the SVM has good generalization ability and can simultaneously minimize the empirical risk and the expected risk for pattern classification problems. SVM construct a decision plane separating two classes with the largest margin, which is the maximum distance between the closest vector to the hyperplane. In other word, the main idea of a support vector machine is to construct a hyperplane as the decision surface in such a way that the margin of separation between positive and negative examples is maximized. More importantly, an SVM can work very well in a high dimensional feature space. The support vector method can also be applied in regression (functional approximation) problems. When SVM is employed to tackle the problems of function approximation and regression estimation, it is referred as the support vector regression (SVR). SVR can perform high accuracy and robustness for function approximation with noise.

However, the optimal solutions of SVM rely heavily on the property of selected kernel functions, whose parameters are always fixed and are chosen solely based on heuristics or trial-and-error nowadays. The regular SVM suffers from the difficulty of long computational time in using nonlinear kernels on large datasets which come from many real applications. Therefore, our dissertation proposes a systematical procedure to reduce the support vectors to deal with this problem.

1.3 Research Objectives and Organization of this dissertation

In this dissertation, novel fuzzy neural networks (FNNs) combining with support vector learning mechanism called support-vector-based fuzzy neural networks (SVFNNs) are proposed for pattern classification and function approximation. The SVFNNs combine the capability of minimizing the empirical risk (training error) and expected risk (testing error) of support vector learning in high dimensional data spaces and the efficient human-like reasoning of FNN in handling uncertainty information. There have been some researches on combining SVM with FNN [19]-[22]. In [19], a self-organizing map with fuzzy class memberships was used to reduce the training samples to speed up the SVM training. The objective of [20]-[22] was on improving the accuracy of SVM on multi-class pattern recognition problems. The overall objective of this dissertation is to develop a theoretical foundation for the FNN using the SVM method. We exploit the knowledge representation power and learning ability of the FNN to determine the kernel functions of the SVM adaptively, and propose a novel adaptive fuzzy kernel function, which has been proven to be a Mercer kernel. The SVFNNs can not only well maintain the classification accuracy, but also reduce the number of support vectors as compared with the regular SVM. Organization and objectives of the dissertation are as follows.

In chapter 2, a novel adaptive fuzzy kernel is proposed for combining FNN with SVM. We exploit the knowledge representation power and learning ability of the FNN to determine the kernel functions of the SVM adaptively and develop a novel adaptive fuzzy kernel function. The objective of this chapter is to prove that the adaptive fuzzy kernel conform to the Mercer theory.

In chapter 3, a support-vector based fuzzy neural network (SVFNN) is proposed. This network is developed for solving pattern recognition problem. Compared to conventional neural fuzzy network approaches, the objective of this chapter is to construct the learning algorithm of the proposed SVFNN with simultaneously minimizing the empirical risk and the expected risk for good generalization ability and characterize the proposed SVFNN with good classification performance.

In chapter 4, a support-vector based fuzzy neural network for function approximation is proposed. This network is developed for solving function approximation. The objective of this chapter is to integrate the statistical support vector learning method into FNN and characterize the proposed SVFNN with the capability of good robustness against noise.

The applications and simulated results of the SVFNNs are presented at the ends of Chapter 3 and 4, respectively. Finally, conclusions are made on Chapter 5.

CHAPTER 2
SUPPORT-VECTOR BASED FUZZY NEURAL NETWORK AND THE ADAPTIVE FUZZY KERNEL

In this chapter, adaptive fuzzy kernel is proposed for applying the SVM technique to obtain the optimal parameters of FNN. The adaptive fuzzy kernel provides the SVM with adaptive local representation power, and thus brings the advantages of FNN (such as adaptive learning and economic network structure) into the SVM directly. On the other hand, the SVM provides the advantage of global optimization to the FNN and also its ability to minimize the expected risk; while the FNN originally works on the principle of minimizing only the training error.

2.1 Structure of the FNN

A four-layered fuzzy neural network (FNN) is shown in Fig 2.1, which is comprised of the input, membership function, rule, and output layers. Layer 1 accepts input variables, whose nodes represent input linguistic variables. Layer 2 is to calculate the membership values, whose nodes represent the terms of the respective linguistic variables. Nodes at Layer 3 represent fuzzy rules. The links before Layer 3 represent the preconditions of fuzzy rules, and the link after Layer 3 represent the consequences of fuzzy rules. Layer 4 is the output layer. This four-layered network realizes the following form of fuzzy rules:

Rule R$_j$: If x_1 is A$_{1j}$ and …x_i is A$_{ij}$….. and x_M is A$_{Mj}$, Then y is d_j , j=1, 2, $\cdots$, N, (2.1)

where A$_{ij}$ are the fuzzy sets of the input variables x_i, i =1, 2, $\cdots$, M and d_j are the consequent parameter of y. For the ease of analysis, a fuzzy rule 0 is added as:

Rule 0 : If x_1 is A$_{10}$ and …….. and x_M is A$_{M0}$, Then y is d_0, (2.2)

where A$_{k0}$ is a universal fuzzy set, whose fuzzy degree is 1 for any input value x_i, i =1, 2, $\cdots$, M and d_0 is the consequent parameter of y in the fuzzy rule 0. Define $O^{(P)}$ and $a^{(P)}$ as the output and input variables of a node in layer P, respectively. The signal propagation and the basic functions in each layer are described as follows.

Layer 1- Input layer: No computation is done in this layer. Each node in this layer, which corresponds to one input variable, only transmits input values to the next layer directly. That is

$$O^{(1)} = a_i^{(1)} = x_i , \tag{2.3}$$

where x_i , i=1, 2, $\cdots$, M are the input variables of the FNN.

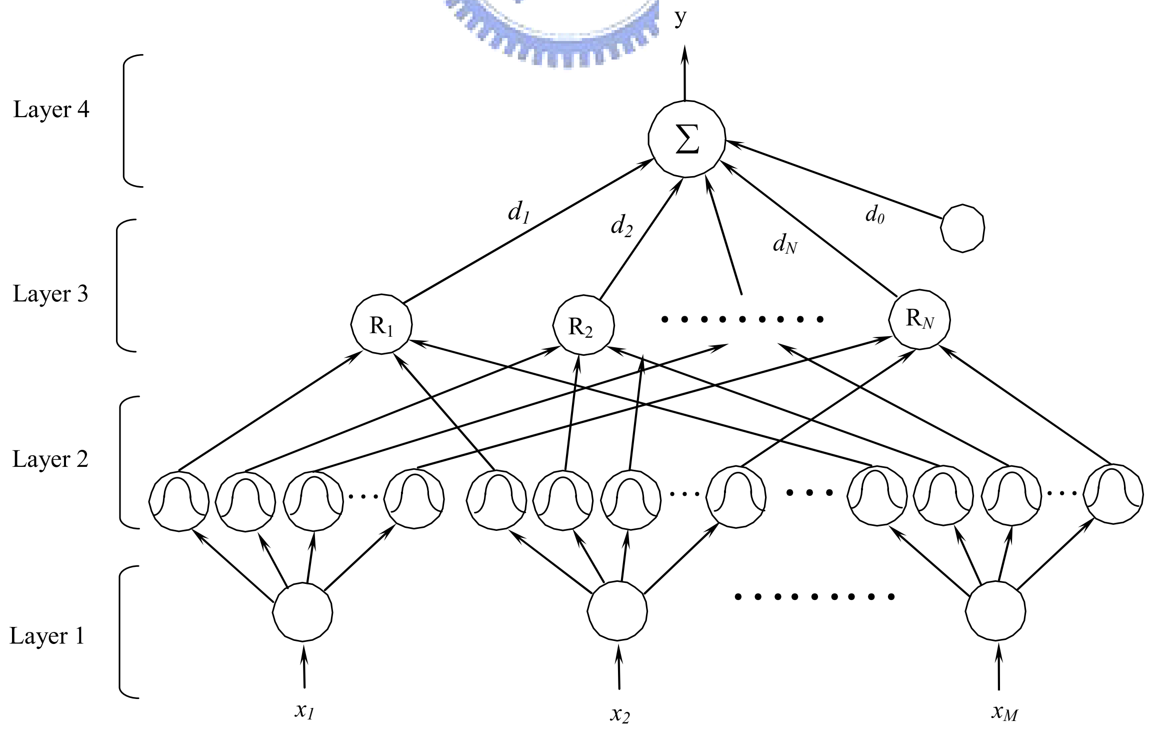


Fig. 2.1 The structure of the four-layered fuzzy neural network.

Layer 2 – Membership function layer: Each node in this layer is a membership function that corresponds one linguistic label (e.g., fast, slow, etc.) of one of the input variables in Layer 1. In other words, the membership value which specifies the degree to which an input value belongs to a fuzzy set is calculated in Layer 2:

$$O^{(2)} = u_i^{(j)}(a_i^{(2)}), \tag{2.4}$$

where $u_i^{(j)}(\cdot)$ is a membership function $u_i^{(j)}(\cdot): R \to [0,\ 1]$, $i = 1,\ 2,\ \cdots,\ M$, j=1, 2, $\cdots$, N. With the use of Gaussian membership function, the operation performed in this layer is

$$O^{(2)} = e^{-\frac{(a_i^{(2)} - m_{ij})^2}{\sigma_{ij}^2}}, \tag{2.5}$$

where m_{ij} and σ_{ij} are, respectively, the center (or mean) and the width (or variance) of the Gaussian membership function of the j-th term of the i-th input variable x_i.

Layer 3 – Rule layer: A node in this layer represents one fuzzy logic rule and performs precondition matching of a rule. Here we use the AND operation for each Layer 2 node

$$O^{(3)} = \prod_{i=1}^{M} a_i^{(3)} = e^{-[\mathbf{D}_j(\mathbf{x}\text{-}\mathbf{m}_j)]^T[\mathbf{D}_j(\mathbf{x}\text{-}\mathbf{m}_j)]}, \tag{2.6}$$

where $\mathbf{D}_j = diag\left(\frac{1}{\sigma_{1j}},\ \cdots,\ \frac{1}{\sigma_{Mj}}\right)$, $\mathbf{m}_j$=$[m_{1j}, m_{2j}, \ldots, m_{Mj}]^T$, $\mathbf{x}$=$[x_1, x_2, x_3, \cdots, x_M]^T$ is the FNN input vector. The output of a Layer-3 node represents the firing strength of the corresponding fuzzy rule.

Layer 4 – Output layer: The single node $O^{(4)}$ in this layer is labeled with Σ, which computes the overall output as the summation of all input signals:

$$O^{(4)} = \sum_{j=1}^{N} d_j \times a_j^{(4)} + d_0 , \tag{2.7}$$

where the connecting weight d_j is the output action strength of the Layer 4 output associated with the Layer 3 rule and the scalar d_0 is a bias. Thus the fuzzy neural network mapping can be rewritten in the following input-output form:

$$O^{(4)} = \sum_{j=1}^{N} d_j \times a_j^{(4)} + d_0 = \sum_{j=1}^{N} d_j \prod_{i=1}^{M} u_i^{(j)}(x_i) + d_0 . \tag{2.8}$$

2.2 Fuzzy Clustering and Input/Output Space Partitioning

For constructing the initial fuzzy rules of the FNN, the fuzzy clustering method is used to partition a set of data into a number of overlapping clusters based on the distance in a metric space between the data points and the cluster prototypes.

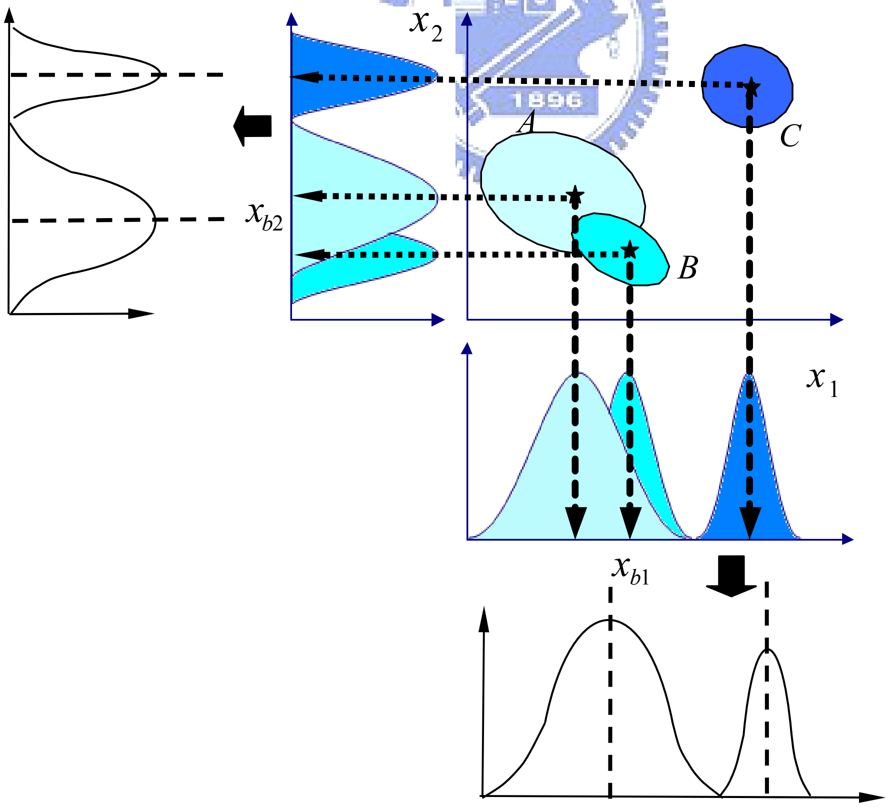


Fig. 2.2 The aligned clustering-based partition method giving both less number of clusters as well as less number of membership functions.

Each cluster in the product space of the input-output data represents a rule in the rule base. The goal is to establish the fuzzy preconditions in the rules. The membership functions in Layer 2 of FNN can be obtained by projections onto the various input variables x_i spanning the cluster space. In this work, we use an aligned clustering-based approach proposed in [23]. This method produces a partition result as shown in Fig. 2.2.

The input space partitioning is also the first step in constructing the fuzzy kernel function in the SVFNNs. The purpose of partitioning has a two-fold objective:

- It should give us a minimum yet sufficient number of clusters or fuzzy rules.
- It must be in spirit with the SVM-based classification scheme.

To satisfy the aforementioned conditions, we use a clustering method which takes care of both the input and output values of a data set. That is, the clustering is done based on the fact that the points lying in a cluster also belong to the same class or have an identical value of the output variable. The class information of input data is only used in the training stage to generate the clustering-based fuzzy rules; however, in testing stage, the input data excite the fuzzy rules directly without using class information. In addition, we also allow existence of overlapping clusters, with no bound on the extent of overlap, if two clusters contain points belonging to the same class. We may have a clustering like the one shown in Fig. 2.3. Thus a point may be geometrically closer to the center of a cluster, but it can belong only to the nearest cluster, which has the points belonging to the same class as that point.

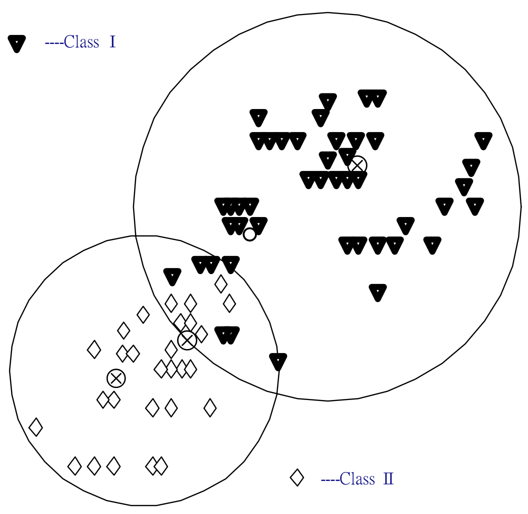


Fig. 2.3 The clustering arrangement allowing overlap and selecting the member points according to the labels (or classes) attached to them.

2.3 Fuzzy Rule Generation

A rule corresponds to a cluster in the input space, with $\mathbf{m}_j$ and $\mathbf{D}_j$ representing the center and variance of that cluster. For each incoming pattern $\mathbf{x}$, the strength a rule is fired can be interpreted as the degree the incoming pattern belongs to the corresponding cluster. It is generally represented by calculating degree of membership of the incoming pattern in the cluster [24]. For computational efficiency, we can use the firing strength derived in (2.6) directly as this degree measure

$$F^j(\mathbf{x}) = \prod_{i=1}^{M} a_i^{(3)} = e^{-[\mathbf{D}_j(\mathbf{x}-\mathbf{m}_j)]^T[\mathbf{D}_j(\mathbf{x}-\mathbf{m}_j)]} \in [0,1], \qquad (2.9)$$

where $F^j(\mathbf{x}) \in [0,\ 1]$. In the above equation the term $[\mathbf{D}_j(\mathbf{x}-\mathbf{m}_j)]^T[\mathbf{D}_j(\mathbf{x}-\mathbf{m}_j)]$ is the distance between $\mathbf{x}$ and the center of cluster j. Using this measure, we can obtain the following criterion for the generation of a new fuzzy rule. Let $\mathbf{x}$ be the newly incoming pattern. Find

$$J = \arg \max_{1 \le j \le c(t)} F^j(\mathbf{x}), \qquad (2.10)$$

where $c(t)$ is the number of existing rules at time t. If $F^J \leq \bar{F}(t)$, then a new rule is generated, where $\bar{F}(t) \in (0,\ 1)$ is a prespecified threshold that decays during the learning process. Once a new rule is generated, the next step is to assign initial centers and widths of the corresponding membership functions. Since our goal is to minimize an objective function and the centers and widths are all adjustable later in the following learning phases, it is of little sense to spend much time on the assignment of centers and widths for finding a perfect cluster. Hence we can simply set

$$\mathbf{m}_{[c(t)+1]} = \mathbf{x}, \tag{2.11}$$

$$\mathbf{D}_{[c(t)+1]} = \frac{-1}{\chi} \cdot diag\left[\frac{1}{\ln(F^J)} \quad \cdots \quad \frac{1}{\ln(F^J)}\right], \tag{2.12}$$

according to the first-nearest-neighbor heuristic [25], where $\chi \geq 0$ decides the overlap degree between two clusters. Similar methods are used in [26], [27] for the allocation of a new radial basis unit. However, in [26] the degree measure doesn't take the width $\mathbf{D}_j$ into consideration. In [27], the width of each unit is kept at a prespecified constant value, so the allocation result is, in fact, the same as that in [26]. In this dissertation, the width is taken into account in the degree measure, so for a cluster with larger width (meaning a larger region is covered), fewer rules will be generated in its vicinity than a cluster with smaller width. This is a more reasonable result. Another disadvantage of [26] is that another degree measure (the Euclidean distance) is required, which increases the computation load.

After a rule is generated, the next step is to decompose the multidimensional membership function formed in (2.11) and (2.12) to the corresponding 1-D membership function for each input variable. To reduce the number of fuzzy sets of each input variable and to avoid the existence of highly similar ones, we should check the similarities between the newly projected membership function and the existing ones in each input dimension. Before going to the details on how this overall process works, let us consider the similarity measure first. Since Gaussian membership functions are used in the SVFNNs, we use the formula of the similarity measure of two fuzzy sets with Gaussian membership functions derived previously in [28]. Suppose the fuzzy sets to be measured are fuzzy sets A and B with membership function $\mu_A(x) = \exp\left\{-(x-c_1)^2/\sigma_1^2\right\}$ and $\mu_B(x) = \exp\left\{-(x-c_2)^2/\sigma_2^2\right\}$, respectively. The union of two fuzzy sets A and B is a fuzzy set $A \cup B$ such that $u_{A\cup B}(x) = \max[u_A(x), u_B(x)]$, for every $x \in U$. The intersection of two fuzzy sets A and B is a fuzzy set $A \cap B$ such that $u_{A\cap B}(x) = \min[u_A(x), u_B(x)]$, for every $x \in U$. The size or cardinality of fuzzy set A, $M(A)$, equals the sum of the support values of A: $M(A) = \sum_{x\in U} u_A(x)$. Since the area of the bell-shaped function, $\exp\{-(x-m)^2/\sigma^2\}$, is $\sigma\sqrt{\pi}$ [29] and its height is always 1, it can be approximated by an isosceles triangle with unity height and the length of bottom edge $2\sigma\sqrt{\pi}$. We can then compute the fuzzy similarity measure of two fuzzy sets with such kind of membership functions. Assume $c_1 \ge c_2$ as in [28], we can compute $M|A\cap B|$ by

$$M|A\cap B| = \sum_{x\in U}(\min[u_A(x), u_B(x)]) = \frac{1}{2}\frac{h^2\left[c_2 - c_1 + \sqrt{\pi}(\sigma_1+\sigma_2)\right]}{\sqrt{\pi}(\sigma_1+\sigma_2)} + \frac{1}{2}\frac{h^2\left[c_2 - c_1 + \sqrt{\pi}(\sigma_1-\sigma_2)\right]}{\sqrt{\pi}(\sigma_2-\sigma_1)} + \frac{1}{2}\frac{h^2\left[c_2 - c_1 - \sqrt{\pi}(\sigma_1+\sigma_2)\right]}{\sqrt{\pi}(\sigma_1-\sigma_2)}, \tag{2.13}$$

where $h(\cdot) = \max\{0, \cdot\}$. So the approximate similarity measure is

$$E(A,B) = \frac{M|A\cap B|}{M|A\cup B|} = \frac{M|A\cap B|}{\sigma_1\sqrt{\pi} + \sigma_2\sqrt{\pi} - M|A\cap B|}, \tag{2.14}$$

where we use the fact that $M(A) + M(B) = M(A\cap B) + M(A\cup B)$ [28]. By using this similarity measure, we can check if two projected membership functions are close enough to be merged into one single membership function $\mu_c(x) = \exp\left\{-(x-c_3)^2 / \sigma_3^2\right\}$. The mean and variance of the merged membership function can be calculated by

$$c_3 = \frac{c_1 + c_2}{2}, \tag{2.15}$$

$$\sigma_3 = \frac{\sigma_1 + \sigma_2}{2}. \tag{2.16}$$

2.4 Adaptive Fuzzy Kernel for the SVM/SVR

The proposed fuzzy kernel $K(\hat{\mathbf{x}}, \hat{\mathbf{z}})$ in this dissertation is defined as

$$K\left(\hat{\mathbf{x}},\ \hat{\mathbf{z}}\right) = \begin{cases} \prod_{i=1}^{M} u_j(x_i) \cdot u_j(z_i), & \text{if } \hat{\mathbf{x}} \text{ and } \hat{\mathbf{z}} \text{ are both in the } j\text{-th cluster} \\ 0, & \text{otherwise,} \end{cases} \tag{2.17}$$

where $\hat{\mathbf{x}}$=[x_1, x_2, x_3, $\cdots$, x_M] $\in R^M$ and $\hat{\mathbf{z}}$=[z_1, z_2, z_3, $\cdots$, z_M] $\in R^M$ are any two training samples, and $u_j(x_i)$ is the membership function of the j-th cluster. Let the training set be **S={(x**$_1$, y_1), (**x**$_2$, y_2), $\cdots$, (**x**$_v$, y_v)} with explanatory variables $\mathbf{x}_i$ and the corresponding class labels y_i, for all $i = 1,\ 2,\ \cdots,\ v$, where v is the total number of training samples. Assume the training samples are partitioned into l clusters through fuzzy clustering in Section II. We can perform the following permutation of training samples

$$\begin{aligned} &cluster\ \ 1 = \left\{\left(\mathbf{x}_1^1, y_1^1\right), \ldots, \left(\mathbf{x}_{k_1}^1, y_{k_1}^1\right)\right\} \\ &cluster\ \ 2 = \left\{\left(\mathbf{x}_1^2, y_1^2\right), \ldots, \left(\mathbf{x}_{k_2}^2, y_{k_2}^2\right)\right\} \\ &\qquad\vdots \\ &cluster\ \ l = \left\{\left(\mathbf{x}_1^l, y_1^l\right), \ldots, \left(\mathbf{x}_{k_l}^l, y_{k_l}^l\right)\right\}, \end{aligned} \tag{2.18}$$

where k_g, $g = 1,\ 2,\ \cdots, l$ is the number of points belonging to the g-th cluster, so that we have $\sum_{g=1}^{l} k_g = v$. Then the fuzzy kernel can be calculated by using the training set in (2.18), and the obtained kernel matrix **K** can be rewritten as the following form

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{K}_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{K}_l \end{bmatrix} \in R^{v \times v} \tag{2.19}$$

where $\mathbf{K}_g,\ g = 1,\ 2,\ \cdots, l$ is defined as

$$\mathbf{K}_g = \begin{bmatrix} K\left(x_1^g, x_1^g\right) & K\left(x_1^g, x_2^g\right) & \cdots & K\left(x_1^g, x_{k_g}^g\right) \\ K\left(x_2^g, x_1^g\right) & K\left(x_2^g, x_2^g\right) & \ddots & \vdots \\ \vdots & \ddots & \ddots & K\left(x_{k_g-1}^g, x_{k_g}^g\right) \\ K\left(x_{k_g}^g, x_1^g\right) & \cdots & K\left(x_{k_g}^g, x_{k_g-1}^g\right) & K\left(x_{k_g}^g, x_{k_g}^g\right) \end{bmatrix} \in R^{k_g \times k_g} \tag{2.20}$$

In order that the fuzzy kernel function defined by (2.17) is suitable for application in SVM, we must prove that the fuzzy kernel function is symmetric and positive-definite Gram Matrices [30]. To prove this, we first quote the following theorems.

Theorem 1 (**Mercer theorem** [30]) **:** Let X be a compact subset of R^n. Suppose K is a continuous symmetric function such that the integral operator $T_K : L_2(X) \rightarrow L_2(X)$

$$(T_K f)(\cdot) = \int_X K(\cdot, \mathbf{x}) f(\mathbf{x}) \ \ d\mathbf{x} \geq 0, \tag{2.21}$$

is positive; that is

$$\int_{X \times X} K(\mathbf{x},\ \ \mathbf{z}) f(\mathbf{x}) f(\mathbf{z}) d\mathbf{x} d\mathbf{z} \geq 0,\ \ \forall f \in L_2(X) \tag{2.22}$$

for all $f \in L_2(X)$. Then we can expand K**(x, z)** in a uniformly convergent series (on $X \times X$) in terms of T_K's eigen-functions $\phi_j \in L_2(X)$, normalized in such a way that $\|\phi_j\|_{L_2} = 1$, and positive associated eigenvalues $\lambda_j > 0$,

$$K(\mathbf{x},\ \mathbf{z}) = \sum_{j=1}^{\infty} \lambda_j \phi_j(\mathbf{x}) \phi_j(\mathbf{z}). \tag{2.23}$$

The kernel is referred to as Mercer's kernel as it satisfies the above Mercer theorem.

Proposition 1 [31] **:** A function $K(\mathbf{x},\ \mathbf{z})$ is a valid kernel iff for any finite set it produces symmetric and positive-definite Gram matrices.

Proposition 2 [32] **:** Let K_1 and K_2 be kernels over $X \times X$, $X \subseteq R^n$. Then the $K(\mathbf{x},\mathbf{z}) = K_1(\mathbf{x},\mathbf{z})K_2(\mathbf{x},\mathbf{z})$ function is also a kernel.

Definition 1 [33] **:** A function $f: R \to R$ is said to be a positive-definite function if the matrix $[f(x_i - x_j)] \in R^{n \times n}$ is positive semidefinite for all choices of points $\{x_1, \cdots, x_n\} \subset R$ and all $n = 1,\ 2,\ \cdots\cdots$.

Proposition 3 [33] **:** A block diagonal matrix with the positive-definite diagonal matrices is also a positive-definite matrix.

Theorem 2 **:** For the fuzzy kernel defined by (2.17), if the membership functions $u(x_i): R \to [0,\ 1],\ i = 1,\ 2,\ \cdots,\ n,$ are positive-definite functions, then the fuzzy kernel is a Mercer kernel.

Proof:

First, we prove that the formed kernel matrix $\mathbf{K} = \left(K\left(\mathbf{x}_i, \mathbf{x}_j\right)\right)_{i,j=1}^{n}$ is a symmetric matrix. According to the definition of fuzzy kernel in (2.17), if x_i and z_i are in the *j*-th cluster,

$$K(\mathbf{x},\ \mathbf{z}) = \prod_{\mathrm{k}=1}^{n} u_{\mathrm{j}}(x_k) \cdot u_{\mathrm{j}}(z_k) = \prod_{\mathrm{k}=1}^{n} u_{\mathrm{j}}(z_k) \cdot u_{\mathrm{j}}(x_k) = K(\mathbf{z},\ \mathbf{x}),$$

otherwise,

$$K(\mathbf{x},\mathbf{z}) = K(\mathbf{z},\ \mathbf{x}) = 0.$$

So the kernel matrix is indeed symmetric. By the elementary properties of *Proposition 2*, the product of two positive-defined functions is also a kernel function. And according to *Proposition 3*, a block diagonal matrix with the positive-definite diagonal matrices is also a positive-definite matrix. So the fuzzy kernel defined by (2.17) is a Mercer kernel. □

Since the proposed fuzzy kernel has been proven to be a Mercer kernel, we can apply the SVM technique to obtain the optimal parameters of SVFNNs. It is noted that the proposed SVFNNs is not a pure SVM, so it dose not minimize the empirical risk and expected risk exactly as SVMs do. However, it can achieve good classification performance with drastically reduced number of fuzzy kernel functions.

CHAPTER 3
SUPPORT-VECTOR BASED FUZZY NEURAL NETWORK FOR PATTERN CLASSIFICATION

In this chapter, we develop a support-vector-based fuzzy neural network (SVFNN) for pattern classification, which is the realization of a new idea for the adaptive kernel functions used in the SVM. The use of the proposed fuzzy kernels provides the SVM with adaptive local representation power, and thus brings the advantages of FNN (such as adaptive learning and economic network structure) into the SVM directly. On the other hand, the SVM provides the advantage of global optimization to the FNN and also its ability to minimize the expected risk; while the FNN originally works on the principle of minimizing only the training error. The proposed learning algorithm of SVFNN consists of three phases. In the first phase, the initial fuzzy rule (cluster) and membership of network structure are automatically established based on the fuzzy clustering method. The input space partitioning determines the initial fuzzy rules, which is used to determine the fuzzy kernels. In the second phase, the means of membership functions and the connecting weights between layer 3 and layer 4 of SVFNN (see Fig. 2.1) are optimized by using the result of the SVM learning with the fuzzy kernels. In the third phase, unnecessary fuzzy rules are recognized and eliminated and the relevant fuzzy rules are determined. Experimental results on five datasets (Iris, Vehicle, Dna, Satimage, Ijcnn1) from the UCI Repository, Statlog collection and IJCNN challenge 2001 show that the proposed SVFNN classification method can automatically generate the fuzzy rules, improve the accuracy of classification, reduce the number of required kernel functions, and increase the speed of classification.

3.1 Maximum Margin Algorithm

An SVM constructs a binary classifier from a set of labeled patterns called training examples. Let the training set be **S** = {($\mathbf{x}_1$, y_1), ($\mathbf{x}_2$, y_2), …, ($\mathbf{x}_v$, y_v)} with explanatory variables $\mathbf{x}_i \in \mathbf{R}^d$ and the corresponding binary class labels $y_i \in \{-1,+1\}$, for all $i = 1, \cdots, v$, where v denotes the number of data, and d denotes the dimension of the datasets. The SVM generates a maximal margin linear decision rule of the form

$$f(\mathbf{x}) = \operatorname{sign}(\mathbf{w} \cdot \mathbf{x} + b), \tag{3.1}$$

Where **w** is the weight vector and b is a bias. The margin M can be calculated by M=2/||**w**|| that show in Fig. 3.1. For obtaining the largest margin, the weight vector, $\mathbf{w}$, must be calculated by

$$\min \ \frac{1}{2}\|\mathbf{w}\|^2$$

$$\text{s.t.} \quad y_i(\mathbf{x}_i\mathbf{w} + b) - 1 \geq 0, \ \ \forall i = 1, \ \cdots, v. \tag{3.2}$$

The optimization problem be converted to a quadratic programming problem, which can be formulated as follows:

$$\text{Maximize} \quad L(\alpha) = \sum_{i=1}^{v} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{v} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{subject to} \quad \alpha_i \geq 0, \quad i = 1, 2, \ldots, v \quad \text{and} \quad \sum_{i=1}^{v} \alpha_i y_i = 0. \tag{3.3}$$

where α_i denotes Lagrange multiplier.

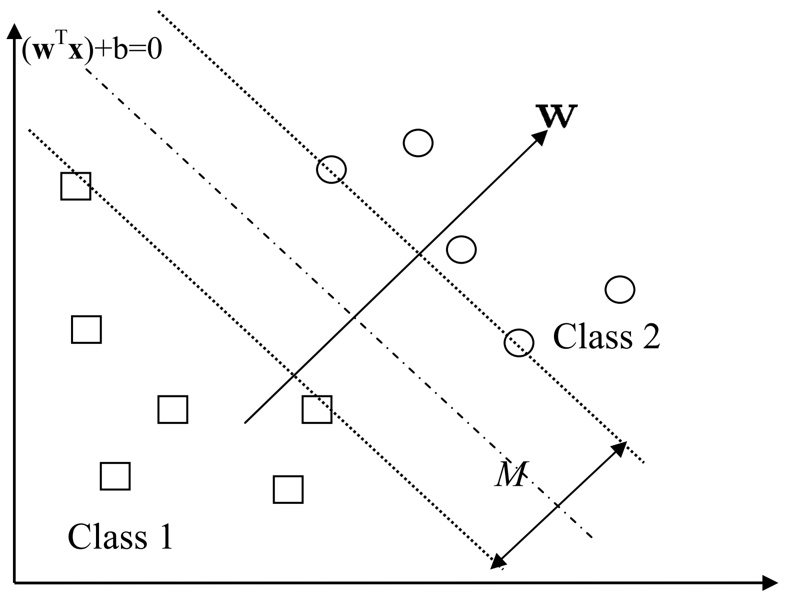


Fig 3.1 Optimal canonical separating hyperplane with the largest margin between the two classes.

In practical applications for non-ideal data, the data contain some noise and overlap. The slack variables ξ, which allow training patterns to be misclassified in the case of linearly non-separable problems, and the regularization parameter *C*, which sets the penalty applied to margin-errors controlling the trade-off between the width of the margin and training set error, are added to SVM. The equation is altered as follows:

$$\min \ \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{2}\sum_i \xi_i^2$$

$$\text{s.t.} \quad y_i(\mathbf{x}_i\mathbf{w}+b) \geq 1-\xi_i, \ \ \forall i = 1, \ \cdots, v. \qquad (3.4)$$

To construct a non-linear decision rule, the kernel method mappin an input vector $\mathbf{x}\in\mathbf{R}^d$ into a vector of a higher-dimensional feature space F ($\phi(\mathbf{x})$, where ϕ represents a mapping $\mathbf{R}^d \rightarrow \mathbf{R}^q$) is discovered. Therefore, the maximal margin linear classifier can solve the linear and non-linear classification problem in this feature space. Fig. 3.2 show the training data map into a higher-dimensional feature space. However, the computation cannot easily map the input vector to the feature space. If the dimension of transformed training vectors is very large, then the computation of the dot products is prohibitively expensive. The transformed function $\phi(\mathbf{x}_i)$ is not known a priori. The Mercer's theorem provides a solution for those problems. The equation $\phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j)$ can be calculated directly by a positive definite symmetric kernel function $K(\mathbf{x}_i,\mathbf{x}_j)=\phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j)$ which complies with Mercer's theorem. Popular choices for Kernel function include

$$\text{Gaussian kernel}: \quad K(\mathbf{x},\overline{\mathbf{x}}) = \exp\left(-\frac{\|\mathbf{x}-\overline{\mathbf{x}}\|^2}{2\sigma^2}\right) \tag{3.5a}$$

$$\text{and Polynomial kernel}: \quad K(\mathbf{x},\overline{\mathbf{x}}) = \left(1+\frac{\mathbf{x}\cdot\overline{\mathbf{x}}}{\sigma^2}\right)^2. \tag{3.5b}$$

To obtain an optimal hyperplane for any linear or nonlinear space, Eq. (3.4) can be rewritten to the following dual quadratic optimization

$$\max_{\alpha} L(\alpha)=\sum_{i=1}^{v}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{v}y_i y_j\alpha_i\alpha_j K(\mathbf{x_i},\mathbf{x_j})$$

$$\text{subject to} \quad 0\le\alpha_i\le C,\ i=1,2,\ldots,v \ \text{ and } \sum_{i=1}^{v}y_i a_i=0. \tag{3.6}$$

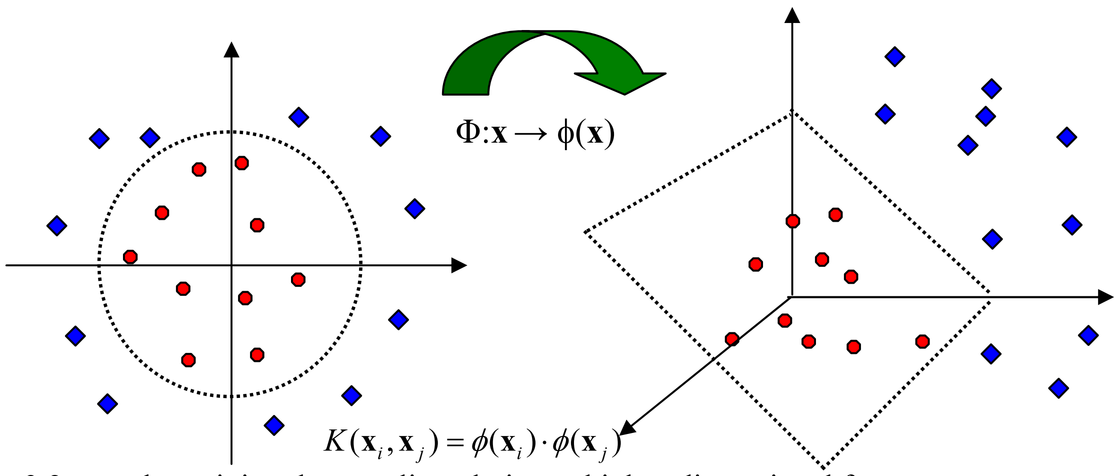

$\Phi:\mathbf{x}\rightarrow\phi(\mathbf{x})$
$K(\mathbf{x}_i,\mathbf{x}_j)=\phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j)$

Fig.3.2 map the training data nonlinearly into a higher-dimensional feature space

The dual Lagrangian $L(\alpha)$ must be maximized with respect to $\alpha_i \geq 0$. The training patterns with nonzero Lagrange multipliers are called support vectors. The separating function is given as follows

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{N_{sv}} y_i \alpha_i^0 K(\mathbf{x}_i \mathbf{x}) + b_0 \right). \tag{3.7}$$

where N_{sv} denotes the number of support vectors; $\mathbf{x}_i$ denotes a support vectors; α_i^0 denotes a corresponding Lagrange coefficient, and b_0 denotes the constant given by

$$b_0 = -\frac{1}{2}\left[\left(\mathbf{w}_0 \mathbf{x}^*(1) \right) + \left(\mathbf{w}_0 \mathbf{x}^*(-1) \right) \right], \tag{3.8}$$

where $\mathbf{x}^*(1)$ denote some support vector belonging to the first class and $0 \leq \alpha_i \leq C$. $\mathbf{x}^*(-1)$ denote some support vector belonging to the second class, where $0 \leq \alpha_i \leq C$. In next section, we proposed the learning algorithm of SVFNN that combine the capability of minimizing the empirical risk (training error) and expected risk (testing error) of support vector learning in high dimensional data spaces and the efficient human-like reasoning of FNN in handling uncertainty information.

3.2 Learning Algorithm of SVFNN

The learning algorithm of the SVFNN consists of three phases. The details are given below:

Learning Phase 1 – Establishing initial fuzzy rules

The first phase establishes the initial fuzzy rules, which were usually derived from human experts as linguistic knowledge. Because it is not always easy to derive fuzzy rules from human experts, the method of automatically generating fuzzy rules from numerical data is issued. The input space partitioning determines the number of fuzzy rules extracted from the training set and also the number of fuzzy sets. We use the centers and widths of the clusters to represent the rules. To determine the cluster to which a point belongs, we consider the value of the firing strength for the given cluster. The highest value of the firing strength determines the cluster to which the point belongs. The whole algorithm for the generation of new fuzzy rules as well as fuzzy sets in each input variable is as follows. Suppose no rules are existent initially.

```
IF $\mathbf{x}$ is the first incoming input pattern THEN do
  PART 1. { Generate a new rule with center $\mathbf{m}_l = \mathbf{x}$ and width
            $\mathbf{D}_1 = diag\left(\frac{1}{\sigma_{init}}, \cdots, \frac{1}{\sigma_{init}}\right)$,
            IF the output pattern $\mathbf{y}$ belongs to class 1 (namely, $\mathbf{y} = [1 \quad 0]$),
              { $\boldsymbol{w}_{Con-1} = [1 \quad 0]$ for indicating output node 1 been excited, }
            ELSE { $\boldsymbol{w}_{Con-1} = [0 \quad 1]$ for indicating output node 2 been excited.}
          }
ELSE for each newly incoming input $\mathbf{x}$, do
  PART 2. {Find $J = \arg \max_{1 \le j \le c(\mathrm{t})} F^j(\mathbf{x})$, as defined in (2.9).
      IF $\boldsymbol{w}_{Con-J} \ne \mathbf{y}$,
        { set $c(t+1) = c(t) + 1$ and generate a new fuzzy rule, with
$\mathbf{m}_{c(t+1)} = \mathbf{x}$ , $\mathbf{D}_{c(t+1)} = \frac{-1}{\chi} diag\left(\frac{1}{\ln\left(F^J\right)}, \cdots, \frac{1}{\ln\left(F^J\right)}\right)$ and
$\boldsymbol{w}_{Con-c(t+1)} = \mathbf{y}$, where $\chi$ decides the overlap degree between two
clusters. In addition, after decomposition, we have $m_{new-i} = x_i$,
$\sigma_{new-i} = -\chi \times \ln(F^J)$ , $i = 1, \cdots, M$ . Do the following fuzzy
measure for each input variable $i$:
{ $Degree(i, t) \equiv \max_{1 \le j \le k_i} E\left[\mu(m_{new-i}, \sigma_{new-i}), \mu(m_{ij}, \sigma_{ij})\right]$
    , where $E(\cdot)$ is defined in (2.14).
    IF $Degree(i, t) \le \rho(t)$
    THEN adopt this new membership function, and set
            $k_i = k_i + 1$, where $k_i$ is the number of partitions of
            the $i$th input variable.
      ELSE merge the new membership function with closest one
            $m_{new-i} = m_{closest} = \frac{m_{new-i} + m_{closest}}{2}$,
            $\sigma_{new-i} = \sigma_{closest} = \frac{\sigma_{new-i} + \sigma_{closest}}{2}$.
  }   } ELSE
{If $F^J \le \bar{F}_{in}(t)$
      {generate a new fuzzy rule with $\mathbf{m}_{c(t+1)} = \mathbf{x}$,
      $\mathbf{D}_{c(t+1)} = \frac{-1}{\chi} diag\left(\frac{1}{\ln\left(F^J\right)}, \cdots, \frac{1}{\ln\left(F^J\right)}\right)$, and the respective consequent
      weight $\boldsymbol{w}_{Con-a(t+1)} = \mathbf{y}$.   In addition, we also need to do the
      fuzzy measure for each input variable $i$.  }   } }
```

In the above algorithm, σ_{init} is a prespecified constant, $c(t)$ is the rule number at time *t*, χ decides the overlap degree between two clusters, and the threshold $\overline{F}_{in}$ determines the number of rules generated. For a higher value of $\overline{F}_{in}$, more rules are generated and, in general, a higher accuracy is achieved. The value $\rho(t)$ is a scalar similarity criterion, which is monotonically decreasing such that higher similarity between two fuzzy sets is allowed in the initial stage of learning. The pre-specified values are given heuristically. In general, $\overline{F}(t) = 0.35$, $\beta = 0.05$, $\sigma_{init} = 0.5$, χ=2. In addition, after we determine the precondition part of fuzzy rule, we also need to properly assign the consequence part of fuzzy rule. Here we define two output nodes for doing two-cluster recognition. If output node 1 obtains higher exciting value, we know this input-output pattern belongs to class 1. Hence, initially, we should assign the proper weight $\mathbf{w}_{Con-1}$ for the consequence part of fuzzy rule. The above procedure gives us means ($\mathbf{m}_{ij}$) and variances (σ_{ij}^2) in (2.9). Another parameter in (2.7) that needs concern is the weight d_j associated with each $a_j^{(4)}$. We shall see later in **Learning Phase 2** how we can use the results from the SVM method to determine these weights.

Learning Phase 2 - Calculating the parameters of SVFNN

Through learning phase (1), the initial structure of SVFNN is established and we can then use SVM [34], [35] to find the optimal parameters of SVFNN based on the proposed fuzzy kernels. The dual quadratic optimization of SVM [36] is solved in order to obtain an optimal hyperplane for any linear or nonlinear space:

$$\text{maximize} \quad L(\vec{\alpha}) = \sum_{i=1}^{v} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{v} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{subject to} \quad 0 \le \alpha_i \le C,\ i = 1,\ 2,\ \cdots,\ v,\ \text{and} \ \sum_{i=1}^{v} y_i \alpha_i = 0, \qquad (3.9)$$

where $K(\mathbf{x}_i, \mathbf{x}_j)$ is the fuzzy kernel in (2.17) and C is a user-specified positive parameter to control the tradeoff between complexity of the SVM and the number of nonseparable points. This quadratic optimization problem can be solved and a solution $\vec{\alpha}_0 = (\alpha_1^0,\ \alpha_2^0,\ .....,\ \alpha_{nsv}^0)$ can be obtained, where α_i^0 are Lagrange coefficients, and *nsv* is the number of support vectors. The corresponding support vectors $\mathbf{sv} = [\mathbf{sx}_1,\ \mathbf{sx}_2,\ \cdots,\ \mathbf{sx}_i,\ \cdots,\ \mathbf{sx}_{nsv}]$ can be obtained, and the constant (threshold) d_0 in (2.7) is

$$d_0 = \frac{1}{2}\left[\left(w_0 \cdot x^*(1)\right) + \left(w_0 \cdot x^*(-1)\right)\right] \quad \text{with} \quad w_0 = \sum_{i=1}^{nsv} \alpha_i y_i x_i, \qquad (3.10)$$

where *nsv* is the number of fuzzy rules (support vectors); the support vector $x^*(1)$ belongs to the first class and support vector $x^*(-1)$ belongs to the second class. Hence, the fuzzy rules of SVFNN are reconstructed by using the result of the SVM learning with fuzzy kernels. The means and variances of the membership functions can be calculated by the values of support vector $\mathbf{m}_j = \mathbf{sx}_j$, j=1, 2, $\cdots$, *nsv*, in (2.5) and (2.6) and the variances of the multidimensional membership function of the cluster that the support vector belongs to, respectively. The coefficients d_j in (2.7) corresponding to $\mathbf{m}_j = \mathbf{sx}_j$ can be calculated by $d_j = y_j \alpha_j$. In this phase, the number of fuzzy rules can be increased or decreased. The adaptive fuzzy kernel is advantageous to both the SVM and the FNN. The use of variable-width fuzzy kernels makes the SVM more efficient in terms of the number of required support vectors, which are corresponding to the fuzzy rules in SVFNN.

Learning Phase 3 – Removing irrelevant fuzzy rules

In this phase, we propose a method for reducing the number of fuzzy rules learning in Phases 1 and 2 by removing some irrelevant fuzzy rules and retuning the consequent parameters of the remaining fuzzy rules under the condition that the classification accuracy of SVFNN is kept almost the same. Several methods including orthogonal least squares (OLS) method and singular value decomposition QR (SVD-QR) had been proposed to select important fuzzy rules from a given rule base [37]-[39]. In [37] the SVD-QR algorithm select a set of independent fuzzy basis function that minimize the residual error in a least squares sense. In [38], an orthogonal least-squares method tries to minimize the fitting error according to the error reduction ratio rather than simplify the model structure [39]. The proposed method reduces the number of fuzzy rules by minimizing the distance measure between original fuzzy rules and reduced fuzzy rules without losing the generalization performance. To achieve this goal, we rewrite (2.8) as

$$O^{(4)} = \sum_{j=1}^{N} d_j \times a_j^{(4)} + d_0 = \sum_{j=1}^{N} d_j \prod_{i=1}^{M} e^{-\frac{(x_i - m_{ij})^2}{\sigma_{ij}^2}} + d_0 \ , \tag{3.11}$$

where N is the number of fuzzy rules after Learning phases 1 and 2. Now we try to approximate it by the expansion of a reduced set :

$$O^{\mathrm{Re}(4)} = \sum_{q=1}^{R_z} \beta_q \times a_q^{\mathrm{Re}(4)} + d_0 = \sum_{q=1}^{R_z} \beta_q \prod_{i=1}^{M} e^{-\frac{(x_i - m_{iq}^{\mathrm{Re}})^2}{\sigma_{iq}^{\mathrm{Re}^2}}} + d_0 \text{ and } a_q^{\mathrm{Re}(4)}(\mathbf{x}) = \prod_{i=1}^{M} e^{-\frac{(x_i - m_{iq}^{\mathrm{Re}})^2}{\sigma_{iq}^{\mathrm{Re}^2}}} \tag{3.12}$$

where R_z is the number of reducing fuzzy rules with $N > R_z$, β_q is the consequent parameters of the remaining fuzzy rules, and m_{iq}^{Re} and $\sigma_{iq}^{\mathrm{Re}}$ are the mean and variance of reducing fuzzy rules. To this end, one can minimize [40]

$$\left\| O^{(4)} - O^{\mathrm{Re}(4)} \right\|^2 = \sum_{j,q=1}^{N} d_j \times d_q \times a_j^{(4)}(\mathbf{m}_q) + \sum_{j,q=1}^{R_z} \beta_j \times \beta_q \times a_j^{\mathrm{Re}(4)}(\mathbf{m}_q^{\mathrm{Re}}) - 2 \times \sum_{j=1}^{N} \sum_{q=1}^{R_z} d_j \times \beta_q \times a_j^{(4)}(\mathbf{m}_q^{\mathrm{Re}}), \quad (3.13)$$

where $\mathbf{m}_q^{\mathrm{Re}} = [m_{1q}^{\mathrm{Re}},\ m_{2q}^{\mathrm{Re}},\ \cdots, m_{Mq}^{\mathrm{Re}}]^T$. Evidently, the problem of finding reduced fuzzy rules consists of two parts: one is to determine the reduced fuzzy rules and the other is to compute the expansion coefficients β_i. This problem can be solved by choosing the more important R_z fuzzy rules from the old N fuzzy rules. By adopting the sequential optimization approach in the reduced support vector method in [41], the approximation in (3.4) can be achieved by computing a whole sequence of reduced set approximations

$$O_r^{\mathrm{Re}(4)} = \sum_{q=1}^{r} \beta_q \times a_q^{\mathrm{Re}(4)}, \quad (3.14)$$

for r=1, 2, $\cdots$, R_Z. Then, the mean and variance parameters, $\mathbf{m}_q^{\mathrm{Re}}$ and σ_q^{Re}, in the expansion of the reduced fuzzy-rule set in (3.4) can be obtained by the following iterative optimization rule [41] :

$$\mathbf{m}_{q+1}^{\mathrm{Re}} = \frac{\sum_{j=1}^{N} d_j \times a_j^{(4)}(\mathbf{m}_q^{\mathrm{Re}}) \times \mathbf{m}_j}{\sum_{j=1}^{N} d_j \times a_j^{(4)}(\mathbf{m}_q^{\mathrm{Re}})}. \quad (3.15)$$

According to (3.7), we can find the parameters, $\mathbf{m}_q^{\mathrm{Re}}$ and σ_q^{Re}, corresponding to the first most important fuzzy rule and then remove this rule from the original fuzzy rule set represented by $\mathbf{m}_j$, j=1, 2, $\cdots$, N and put (add) this rule into the reduced fuzzy rule set. Then the procedure for obtaining the reduced rules is repeated. The optimal coefficients β_q, $q = 1,\ 2,\ \cdots,\ R_z$, are then computed to approximate $O^{(4)} = \sum_{j=1}^{N} d_j \times a_j$ by $O^{\mathrm{Re}(4)} = \sum_{q=1}^{R_z} \beta_q \times a_q^{\mathrm{Re}}$ [41], and can be obtained as

$$\beta = [\beta_1, \beta_2 ...., \beta_{R_z}] = \mathbf{K}_{R_z \times R_z}{}^{-1} \times \mathbf{K}_{R_z \times N} \times \Theta, \tag{3.16}$$

where

$$\mathbf{K}_{R_z \times R_z} = \begin{pmatrix} a_1^{\mathrm{Re}(4)}(\mathbf{m}_1^{\mathrm{Re}}) & a_1^{\mathrm{Re}(4)}(\mathbf{m}_2^{\mathrm{Re}}) & \cdots & a_1^{\mathrm{Re}(4)}(\mathbf{m}_{R_z}^{\mathrm{Re}}) \\ a_2^{\mathrm{Re}(4)}(\mathbf{m}_1^{\mathrm{Re}}) & a_2^{\mathrm{Re}(4)}(\mathbf{m}_2^{\mathrm{Re}}) & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_{R_z-1}^{\mathrm{Re}(4)}(\mathbf{m}_{R_z}^{\mathrm{Re}}) \\ a_{R_z}^{\mathrm{Re}(4)}(\mathbf{m}_1^{\mathrm{Re}}) & \cdots & a_{R_z}^{\mathrm{Re}(4)}(\mathbf{m}_{R_z-1}^{\mathrm{Re}}) & a_{R_z}^{\mathrm{Re}(4)}(\mathbf{m}_{R_z}^{\mathrm{Re}}) \end{pmatrix} \tag{3.17}$$

and

$$\mathbf{K}_{R_z \times N} = \begin{pmatrix} a_1^{\mathrm{Re}(4)}(\mathbf{m}_1) & a_1^{\mathrm{Re}(4)}(\mathbf{m}_2) & \cdots & a_1^{\mathrm{Re}(4)}(\mathbf{m}_{R_x}) \\ a_2^{\mathrm{Re}(4)}(\mathbf{m}_1) & a_2^{\mathrm{Re}(4)}(\mathbf{m}_2) & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_{R_z-1}^{\mathrm{Re}(4)}(\mathbf{m}_N) \\ a_{R_z}^{\mathrm{Re}(4)}(\mathbf{m}_1) & \cdots & a_{R_z}^{\mathrm{Re}(4)}(\mathbf{m}_{N-1}) & a_{R_z}^{\mathrm{Re}(4)}(\mathbf{m}_N) \end{pmatrix} \tag{3.18}$$

and

$$\Theta = [d_1, d_2, \cdots, d_N]. \tag{3.19}$$

The whole learning scheme is iterated until the new rules are sufficiently sparse.

3.3 Experimental Results

The classification performance of the proposed SVFNN is evaluated on five well-known benchmark datasets. These five datasets can be obtained from the UCI repository of machine learning databases [42] and the Statlog collection [43] and IJCNN challenge 2001 [44], [45], respectively.

A. Data and Implementation

From the UCI Repository, we choose one dataset: Iris dataset. From Statlog collection we choose three datasets: Vehicle, Dna and Satimage datasets. The problem Ijcnn1 is from the first problem of IJCNN challenge 2001. These five datasets will be used to verify the effectiveness of the proposed SVFNN classifier. The first dataset (Iris dataset) is originally a collection of 150 samples equally distributed among three classes of the Iris plant namely Setosa, Verginica, and Versicolor. Each sample is represented by four features (septal length, septal width, petal length, and petal width) and the corresponding class label. The second dataset (Vehicle dataset) consists of 846 samples belonging to 4 classes. Each sample is represented by 18 input features. The third dataset (Dna dataset) consists of 3186 feature vectors in which 2000 samples are used for training and 1186 samples are used for testing. Each sample consists of 180 input attributes. The data are classified into three physical classes. All Dna examples are taken from Genbank 64.1. The four dataset (Satimage dataset) is generated from Landsat Multispectral Scanner image data. In this dataset, 4435 samples are used for training and 2000 samples are used for testing. The data are classified into six physical classes. Each sample consists of 36 input attributes. The five dataset (Ijcnn1 dataset) consists of 22 feature vectors in which 49990 samples are used for training and 45495 samples are used for testing. Each sample consists of 22 input attributes. The data are classified into two physical classes. The computational experiments were done on a Pentium III-1000 with 1024MB RAM using the Linux operation system.

For each problem, we estimate the generalized accuracy using different cost parameters C=[2^{12}, 2^{11}, 2^{10}, …, 2^{-2}] in (3.1). We apply 2-fold cross-validation for 100 times on the whole training data in Dna, Satimage and Ijcnn1, and then average all the results. We choose the cost parameter C that results in the best average cross-validation rate for SVM training to predict the test set. Because Iris and Vehicle datasets don't contain testing data explicitly, we divide the whole data in Iris and

Vehicle datasets into two halves, for training and testing datasets, respectively. Similarly, we use the above method to experiment. Notice that we scale all training and testing data to be in [-1, 1].

B. Experimental Results

Tables 3.1 to 3.5 present the classification accuracy rates and the number of used fuzzy rules (i.e., support vectors) in the SVFNN on Iris, Vehicle, Dna, Satimage and Ijcnn1 datasets, respectively. The criterion of determining the number of reduced fuzzy rules is the difference of the accuracy values before and after reducing one fuzzy rule. If the difference is larger than 0.5%, meaning that some important support vector has been removed, then we stop the rule reduction. In Table 3.1, the SVFNN is verified by using Iris dataset, where the constant *n* in the symbol SVFNN-*n* means the number of the learned fuzzy rules. The SVFNN uses fourteen fuzzy rules and achieves an error rate of 2.6% on the training data and an error rate of 4% on the testing data. When the number of fuzzy rules is reduced to seven, its error rate increased to 5.3%. When the number of fuzzy rules is reduced to four, its error rate is increased to 13.3%. Continuously decreasing the number of fuzzy rules will keep the error rate increasing. From Table 3.2 to 3.5, we have the similar experimental results as those in Table 3.1.

These experimental results show that the proposed SVFNN is good at reducing the number of fuzzy rules and maintaining the good generalization ability. Moreover, we also refer to some recent other classification performance include support vector machine and reduced support vectors methods [46]-[48]. The performance comparisons among the existing fuzzy neural network classifiers [49], [50], the RBF-kernel-based SVM (without support vector reduction) [46], reduced support vector machine (RSVM) [48] and the proposed SVFNN are made in Table 3.6.

TABLE 3.1 Experimental results of SVFNN classification on the Iris dataset.

SVFNN-*n* (SVFNN with *n* fuzzy rules)	Training process		Testing process	
	Error rate	C	Number of misclassification	Error rate
SVFNN-14	2.6%	2^{12}	3	4%
SVFNN -11	2.6%	2^{12}	3	4%
SVFNN -9	2.6%	2^{12}	3	4%
SVFNN -7	4%	2^{12}	4	5.3%
SVFNN -4	17.3%	2^{12}	10	13.3%
1. Input dimension is 4. 2. The number of training data is 75. 3. The number of testing data is 75.				

TABLE 3.2 Experimental results of SVFNN classification on the Vehicle dataset.

SVFNN-*n* (SVFNN with *n* fuzzy rules)	Training process		Testing porcess	
	Error rate	C	Number of misclassification	Error rate
SVFNN-321	13.1%	2^{11}	60	14.2%
SVFNN-221	13.1%	2^{11}	60	14.2%
SVFNN-171	13.1%	2^{11}	60	14.2%
SVFNN-125	14.9%	2^{11}	61	14.5%
SVFNN-115	29.6%	2^{11}	113	26.7%
1. Input dimension is 18. 2. The number of training data is 423. 3. The number of testing data is 423.				

TABLE 3.3 Experimental results of SVFNN classification on the Dna dataset.

SVFNN-*n* (SVFNN with *n* fuzzy rules)	Training process		Testing process	
	Error Rate	C	Number of misclassification	Error rate
SVFNN-904	6.2%	2^4	64	5.4%
SVFNN-704	6.2%	2^4	64	5.4%
SVFNN-504	6.2%	2^4	64	5.4%
SVFNN-334	6.4%	2^4	69	5.8%
SVFNN-300	9.8%	2^4	139	11.7%
1. Input dimension is 180. 2. The number of training data is 2000. 3. The number of testing data is 1186.				

TABLE 3.4 Experimental results of SVFNN classification on the Satimage dataset.

SVFNN-*n* (SVFNN with *n* fuzzy rules)	Training process		Testing process	
	Error Rate	C	Number of misclassification	Error Rate
SVFNN-1886	13.1%	2^6	176	8.8%
SVFNN-1586	13.1%	2^6	176	8.8%
SVFNN-1286	13.1%	2^6	176	8.8%
SVFNN-986	13.1%	2^6	176	8.8%
SVFNN-899	13.7%	2^6	184	9.2%
SVFNN-786	19.8%	2^6	316	15.8%
1. Input dimension is 36. 2. The number of training data is 4435. 3. The number of testing data is 2000.				

TABLE 3.5 Experimental results of SVFNN classification on the Ijnn1 dataset.

SVFNN-*n* (SVFNN with *n* fuzzy rules)	Training process		Testing porcess	
	Error rate	C	Number of misclassification	Error rate
SVFNN-1945	4.2%	2^{12}	1955	4.3%
SVFNN-1545	4.2%	2^{12}	1955	4.3%
SVFNN-1245	4.2%	2^{12}	1955	4.3%
SVFNN-1021	4.3%	2^{12}	2047	4.5%
SVFNN-977	14.5%	2^{12}	7416	16.3%
1. Input dimension is 22. 2. The number of training data is 49990. 3. The number of testing data is 45495.				

TABLE 3.6 Classification error rate comparisons among FNN, RBF-kernel-based SVM, RSVM and SVFNN classifiers, where NA means "not available".

Datasets	FNN [49, 50]		RBF-kernel-based SVM [46]		RSVM [48]		SVFNN	
	Number of fuzzy rules	Error rate	Number of support vectors	Error rate	Number of support vectors	Error rate	Number of Fuzzy rules	Error rate
Iris	NA	4.3%	16	3.3%	NA	NA	7	5.3%
Vehicle	NA	29.9%	343	13.4%	NA	NA	125	14.5%
Dna	NA	16.4%	1152	4.2%	372	7.7%	334	5.8%
Satimage	NA	8.9%	2170	8.3%	1826	10.1%	889	9.2%
Ijcnn1	NA	NA	4555	1.2%	200	8.4%	1021	4.5%

3.4 Discussions

These experimental results show that the proposed SVFNN is good at reducing the number of fuzzy rules and maintaining the good generalization ability. These results indicate that the SVFNN classifier produces lower testing error rates as compared to FNN classifiers [49], [50], and uses less support vectors as compared to the regular SVM using fixed-width RBF kernels [46]. As compared to RSVM [48], the proposed SVFNN can not only achieve high classification accuracy, but also reduce the number of support vectors quit well. It is noticed that although the SVFNN uses more support vectors in the Ijcnn1 dataset than the RSVM, it maintains much higher classification accuracy than the RSVM. In summary, the proposed SVFNN classifier exhibits better generalization ability on the testing data and use much smaller number of fuzzy rules.

CHAPTER 4

SUPPORT-VECTOR BASED FUZZY NEURAL NETWORK FOR FUNCTION APPROXIMATION

In this chapter, a novel support-vector based fuzzy neural network (SVFNN) which integrates the statistical support vector learning method into FNN and exploits the knowledge representation power and learning ability of the FNN to determine the kernel functions of the SVR adaptively is proposed. The SVFNN combine the capability of good robustness against noise and the efficient human-like reasoning of FNN in handling uncertainty information. The use of the proposed fuzzy kernels provides the SVR with adaptive local representation power such that the number of support vectors can be further reduced. The proposed learning algorithm consists of three learning phases to construct and train the SVFNN. In the first phase, the fuzzy rules and membership functions are automatically determined based on the fuzzy clustering method. In the second phase, the parameters of FNN are calculated by the SVR with the proposed adaptive fuzzy kernel function for function approximation. In the third phase, the relevant fuzzy rules are selected by the proposed fuzzy rule reduction method. The proposed SVFNN method can automatically generate the fuzzy rules and achieve good approximation performance with drastically reduced number of fuzzy rule and robustness.

4.1 Support Vector Regression Algorithm

In ε-SV regression, the goal is to find a function $f(\mathbf{x})$ that has at most ε deviation from the actually obtained targets y_i for all the training data, and at the same time is as flat as possible. In other words, we do not care about errors as long as they are less than ε, but will not accept any deviation larger than this.

For this reasons, the linear regression function is considered first as follows:

$$f(\mathbf{x})=\mathbf{w}^{\mathrm{T}}\mathbf{x}+b \tag{4.1}$$

Where **w** is the weight vector and b is a bias. The error of approximation is used instead of the margin between an optimal separating hyperplane and support vectors. Vapnik introduced a general type of loss function, the linear loss function with ε-insensitivity zone:

$$|y-f(\mathbf{x})|_{\varepsilon}=\begin{cases} 0 & \text{if } |y-f(\mathbf{x})|\le\varepsilon, \\ |y-f(\mathbf{x})|-\varepsilon & \text{otherwise.} \end{cases} \tag{4.2}$$

The loss is equal to zero if the difference between the predicted $f(\mathbf{x})$ and the measured value is less than ε. The ε-insensitivity loss function defines an ε tube. If the predicted value is within the tube, the loss is zero. For all other predicted points outside the tube, the loss is equal to the magnitude of the difference between the predicted value and the radius ε of the tube. Figure 4.1 shows the soft margin loss setting for a regression problem.

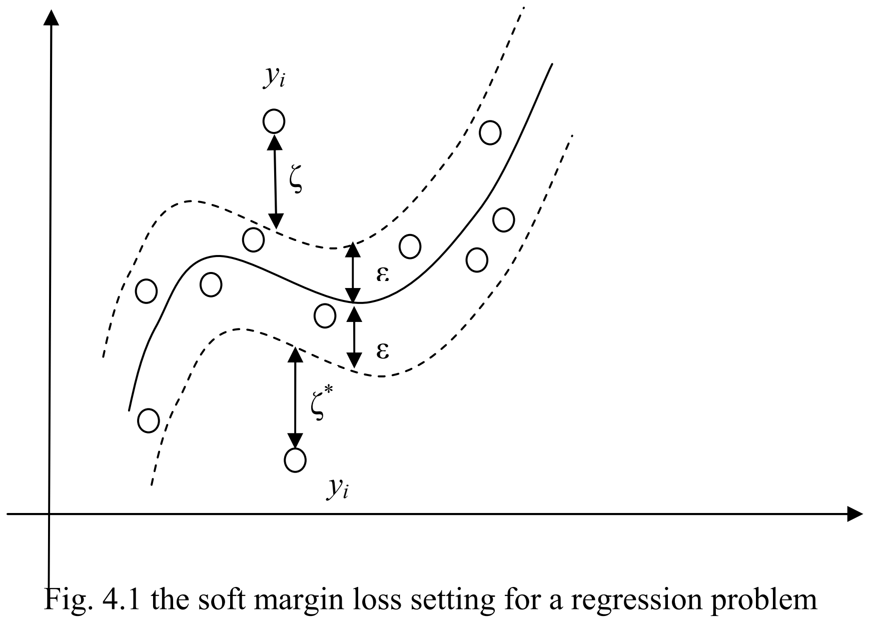


Fig. 4.1 the soft margin loss setting for a regression problem

From Fig. 4.1, the slack variables ξ_i, ξ_i^* cope with the large outliers in the regression problem. In formulating support vector algorithm for regression, the objective is to minimize the empirical risk and $\|\mathbf{w}\|^2$ simultaneously. The primal problem can therefore be defined as follows:

$$\text{minimize} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*)$$
$$\text{subject to} \quad \begin{cases} y_i - f(\mathbf{x}) \le \varepsilon + \xi_i \\ f(\mathbf{x}) - y_i \le \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \ge 0 \end{cases} \tag{4.3}$$

The constant C>0 determines the trade-off between the flatness of $f(\mathbf{x})$ and the amount up to which deviations larger than ε are tolerated. The optimization problem can be converted to the dual optimization problem, which can be formulated as follows:

maximize

$$L(\alpha,\alpha^*) = -\varepsilon\sum_{i=1}^{v}(\alpha_i^* + \alpha_i) + \sum_{i=1}^{v}(\alpha_i^* - \alpha_i)y_i - \frac{1}{2}\sum_{i,j=1}^{v}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)\mathbf{x}_i^T\mathbf{x}_j$$

$$\text{subject to} \quad \sum_{i=1}^{v}\alpha_i^* = \sum_{i=1}^{v}\alpha_i \ ,0 \le \alpha_i^* \le C,\ \ 0 \le \alpha_i \le C,\ i = 1,\ 2,\ \cdots,\ v \tag{4.4}$$

The kernel method can be added to above optimization to solve the nonlinear problem, too. The parameter ε in the ε-insensitive function and the regular constant C are powerful means for regularization and adaptation to the noise in training data. Both parameters control the network complexity and the generalization capability of SVR. In next section, we proposed the learning algorithm of SVFNN that combine the capability of good robustness against noise and the efficient human-like reasoning of FNN in handling uncertainty information. The SVFNN use the fuzzy kernels to provide the SVR with adaptive local representation power such that the number of support vectors can be further reduced.

4.2 Learning Algorithm of SVFNN

The proposed learning algorithm of SVFNN consists of three phases. In the first phase, the initial fuzzy rule (cluster) and membership of network structure are automatically established based on the fuzzy clustering method. The input space partitioning determines the initial fuzzy rules, which is used to determine the fuzzy kernels. In the second phase, the means of membership functions and the connecting weights between layer 3 and layer 4 of SVFNN (see Fig. 2.1) are optimized by using the result of the support vector learning method with the fuzzy kernels function approximation. In the third phase, unnecessary fuzzy rules are recognized and eliminated and the relevant fuzzy rules are determined.

Learning Phase 1 – Establishing initial fuzzy rules

The first phase establishes the initial fuzzy rules. The input space partitioning determines the number of fuzzy rules extracted from the training set and also the number of fuzzy sets. We use the centers and widths of the clusters to represent the rules. To determine the cluster to which a point belongs, we consider the value of the firing strength for the given cluster. The highest value of the firing strength determines the cluster to which the point belongs. The input vector $\mathbf{x}_i$ will combine the corresponding output value y_i in the training set $\mathbf{S}=\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_v, y_v)\}$ to input the learning phase 1. For generating a compact structure, the Cartesian product-space of the input and output is applied to the clustering algorithm [60]. The training samples are partitioned into characteristic regions where the system behaviors are approximated. The input data set is formed by combining the input vector $\mathbf{x}=[x_1, x_2, x_3, \ldots, x_M]^T$ and the corresponding output value y_i. Based on the clustering-based approach to construct initial fuzzy rules of FNN, first the input data is partitioned. For each incoming pattern **b**,

$$\mathbf{b}=[\mathbf{x};y]^T. \tag{4.5}$$

The whole algorithm of SVFNN for the generation of new fuzzy rules as well as fuzzy sets in each input variable is as follows. Suppose no rules are existent initially.

```
IF $\mathbf{b}=[\mathbf{x};y]_{(n+1)\times 1}$ is the first incoming input pattern THEN do
*PART 1.* { Generate a new rule with center $\mathbf{m}_1 = \mathbf{b}$ and width
        $\mathbf{D}_1 = diag\left(\frac{1}{\sigma_{init}},\cdots,\frac{1}{\sigma_{init}}\right)$. After decomposition, we have $n$
        one-dimensional membership functions, with $m_{1i}=b_i$ and $\sigma_{1i}=\sigma_{init}$, $i=1, \cdots, n+1$.
    }
ELSE for each newly incoming input $\mathbf{b}=[\mathbf{x};y]$, do
*PART 2.* {Find $J = \arg \max_{1\le j\le c(t)} F^j(z)$, as defined in (2.10).
    IF $F^J \ge \overline{F}_{in}(t)$
        do nothing
    ELSE
        { set $c(t+1) = c(t)+1$ and generate a new fuzzy rule, with $\mathbf{m}_{c(t+1)} = \mathbf{b}$,
        $\mathbf{D}_{c(t+1)} = \frac{-1}{\chi} diag\left(\frac{1}{\ln\left(F^J\right)},\cdots,\frac{1}{\ln\left(F^J\right)}\right)$, where $\chi$ decides the overlap
        degree between two clusters. In addition, after decomposition, we have
        $m_{new-i} = \mathbf{b}_i$, $\sigma_{new-i} = -\chi\times\ln(F^J)$, $i = 1,\cdots,M$. Do the following fuzzy
        measure for each input variable $i$:
            { $Degree(i,t) \equiv \max_{1\le j\le k_i} E\left[\mu(m_{new-i},\sigma_{new-i}), \mu(m_{ij},\sigma_{ij})\right]$
        , where $E(\cdot)$ is defined in (2.14).
        IF $Degree(i, t) \le \rho(t)$
         THEN adopt this new membership function, and set
              $k_i = k_i + 1$, where $k_i$ is the number of partitions of
              the $i$th training pattern.
        ELSE merge the new membership function with closest one
              $m_{new-i} = m_{closest} = \frac{m_{new-i} + m_{closest}}{2}$,
              $\sigma_{new-i} = \sigma_{closest} = \frac{\sigma_{new-i} + \sigma_{closest}}{2}$.
} } }
```

In the above algorithm, σ_{init} is a prespecified constant, $c(t)$ is the rule number at time t, χ decides the overlap degree between two clusters, and the threshold $\overline{F}_{in}$ determines the number of the generated rules. For a higher value of $\overline{F}_{in}$, more rules are generated and, in general, a higher accuracy is achieved. The value $\rho(t)$ is a scalar similarity criterion, which is monotonically decreasing such that higher similarity between two fuzzy sets is allowed in the initial stage of learning. The pre-specified values are given heuristically. In addition, after we determine the precondition part of fuzzy rule, we also need to properly assign the consequence part of fuzzy rule. Hence, initially, we should assign the proper weight $\boldsymbol{w}_{Con-1}$ for the consequence part of fuzzy rule. The above procedure gives us means ($\mathbf{m}_{ij}$) and variances (σ_{ij}^2) in (2.12). Another parameter in (2.7) that needs concern is the weight

d_j associated with each $a_j^{(4)}$. It is presented in **Learning Phase 2** to show how we can use the results from the SVR method to determine these weights.

Learning Phase 2 - Calculating the parameters of SVFNN

Through above method, the optimal parameters of SVFNN are trained by using the ε-insensitivity loss function SVR [35] based on the fuzzy kernels [61]. The dual quadratic optimization of SVR [36], [62] is solved in order to obtain an optimal hyperplane for any linear or nonlinear space:

maximize $L(\alpha,\alpha^*) = -\varepsilon\sum_{i=1}^{v}(\alpha_i^* + \alpha_i) + \sum_{i=1}^{v}(\alpha_i^* - \alpha_i)y_i - \frac{1}{2}\sum_{i,j=1}^{v}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)K(\mathbf{x}_i,\mathbf{x}_j)$

subject to constraints $\sum_{i=1}^{v}\alpha_i^* = \sum_{i=1}^{v}\alpha_i$, $0 \le \alpha_i^* \le C$, $0 \le \alpha_i \le C$, $i = 1, 2, \cdots, v$. (4.6)

where $K(\mathbf{x}_i,\mathbf{x}_j)$ is the fuzzy kernel that is defined as (2.17), ε is a previously chosen nonnegative number for ε-insensitive loss function and C is a user-specified positive parameter to control the tradeoff between complexity of the SVR and the number of nonseparable points. This quadratic optimization problem can be solved and a solution $\vec{\alpha} = (\alpha_1, \alpha_2, \ldots.., \alpha_{nsv})$ and $\vec{\alpha}^* = (\alpha_1^*, \alpha_2^*, \ldots.., \alpha_{nsv}^*)$ can be obtained, where α_i and α_i^* are Lagrange coefficients, and *nsv* is the number of support vectors. The corresponding support vectors $\mathbf{sv} = [\mathbf{sx}_1, \mathbf{sx}_2, \cdots, \mathbf{sx}_i, \cdots, \mathbf{sx}_{nsv}]$ can be obtained, and the constant (threshold) d_0 in (2.7) is

$$d_0 = \frac{1}{v}\left(\sum_{i=1}^{v}(y_i - \mathbf{x}_i^T\mathbf{w}_0)\right) \quad \text{with} \quad \mathbf{w}_0 = \sum_{i=1}^{nsv}(\alpha_i^* - \alpha_i)\mathbf{x}_i, \qquad (4.7)$$

where *nsv* is the number of fuzzy rules (support vectors). Hence, the fuzzy rules of SVFNN are reconstructed by using the result of the SVR learning with fuzzy kernels.

The means and variances of the membership functions can be calculated by the values of support vector $\mathbf{m}_j = \mathbf{sx}_j$, j=1, 2, $\cdots$, *nsv*, in (2.6) and (2.7) and the variances of the multidimensional membership function of the cluster that the support vector belongs to, respectively. The coefficients d_j in (2.8) corresponding to $\mathbf{m}_j = \mathbf{sx}_j$ can be calculated by $d_j = y_j(\alpha_j^* - \alpha_j)$. In this phase, the number of fuzzy rules can be increased or decreased. The adaptive fuzzy kernel is advantageous to both the SVR and the FNN. The use of variable-width fuzzy kernels makes the SVR more efficient in terms of the number of required support vectors, which are corresponding to the fuzzy rules in SVFNN.

Learning Phase 3 – Removing irrelevant fuzzy rules

In this phase, the number of fuzzy rules learning in Phases 1 and 2 are reduced by removing some irrelevant fuzzy rules. The method of reducing fuzzy rules attempts to reduce the number of fuzzy rules by minimizing the distance measure between original fuzzy rules and reduced fuzzy rules without losing the generalization performance. The reducing method is the same as in Section 2 of Chapter 3

4.3 Experimental Results

In this section we present some experimental results to demonstrate the performance and capabilities of the proposed SVFNN. First, we apply the SVFNN to four function approximation problems to examine its rule-reduction performance. Then the robustness of SVFNN is evaluated by these functions with noise.

A. Setup

1) Functions for approximation:

The function approximation problems include one- and two- variable functions which have been widely used in the literature [63]-[65]:

The fist function is a one-variable *sinc* function defined as

$$f^{(1)}(x) = \frac{\sin(x)}{x} \quad \text{with} \quad x \in [-10, \ 10]. \tag{4.8}$$

The second function is one-variable function defined as

$$f^{(2)}(x) = x^{2/3} \quad \text{with} \quad x \in [-2, \ 2]. \tag{4.9}$$

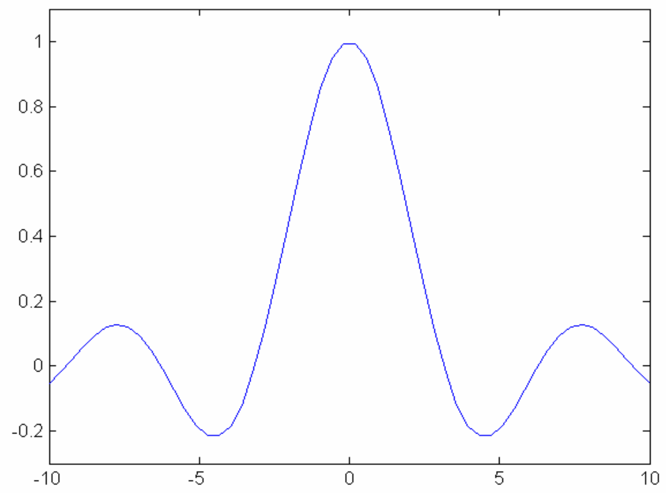
The third function is a two-variable Gaussian function defined as

$$f^{(3)}(x,y) = \exp\{-2(x^2 + y^2)\} \quad \text{with} \quad x \in [-1, \ 1], \quad y \in [-1, \ 1]. \tag{4.10}$$

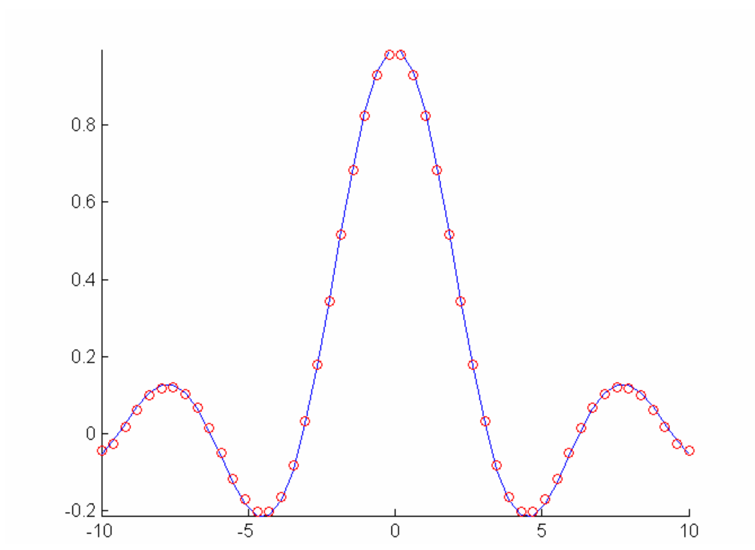
The fourth function, which exhibits a more complex structure, is defined as

$$f^{(4)}(x,y) = \frac{\sin(10\sqrt{x^2+y^2})}{10\sqrt{x^2+y^2}} \quad \text{with} \quad x \in [-1, \ 1], \quad y \in [-1, \ 1]. \tag{4.11}$$

Plots of these four functions are shown in subplots (a) of Figs. 4.2-4.5.

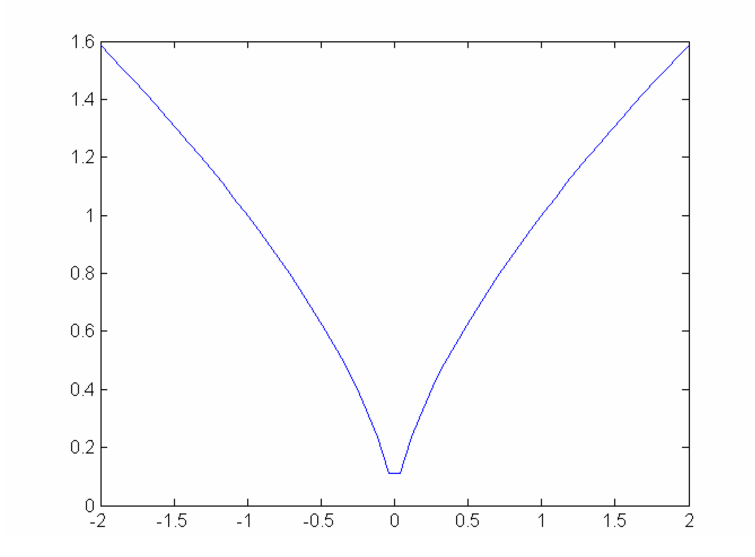


(a)

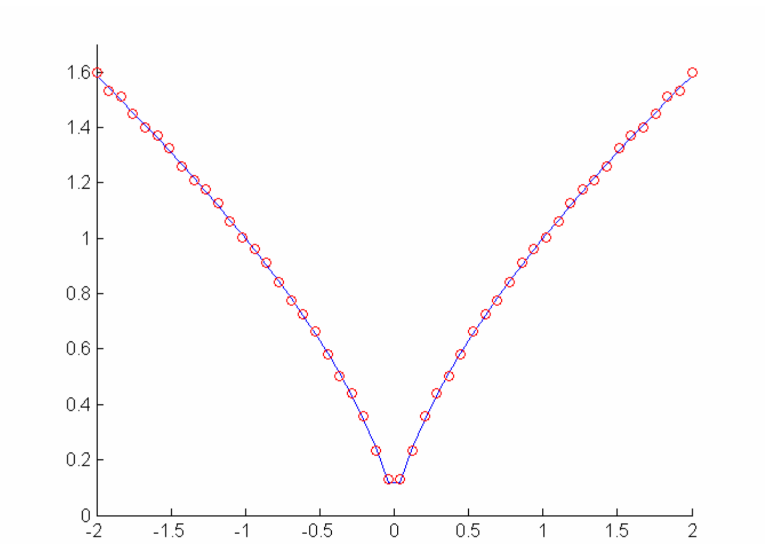


(b)

Fig.4.2 (a) The desired output of the function show in (4.8). (b) The resulting approximation by SVFNN.

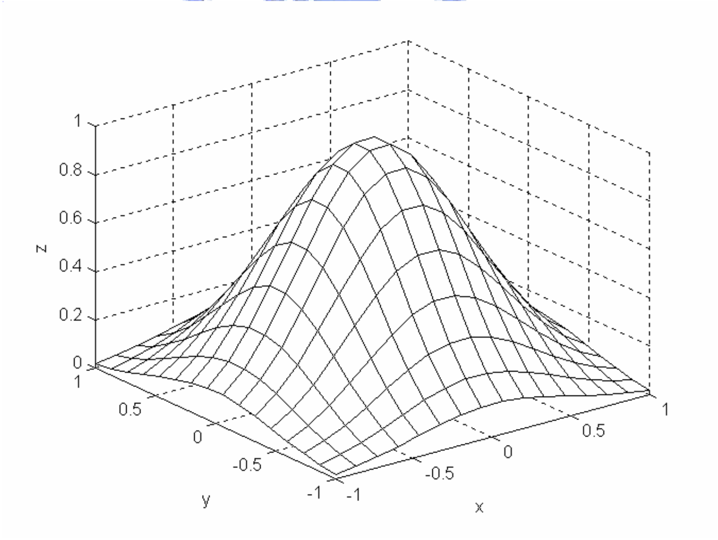


(a)

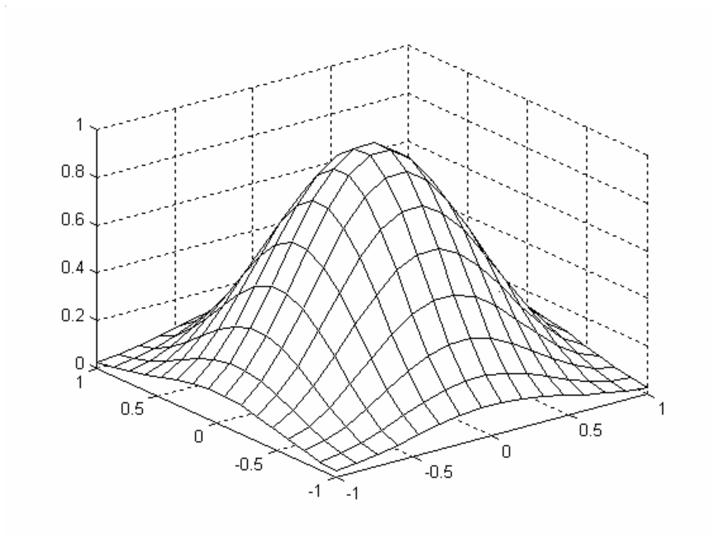


(b)

Fig 4.3 (a) The desired output of the function show in (4.9) (b) The resulting approximation by SVFNN.

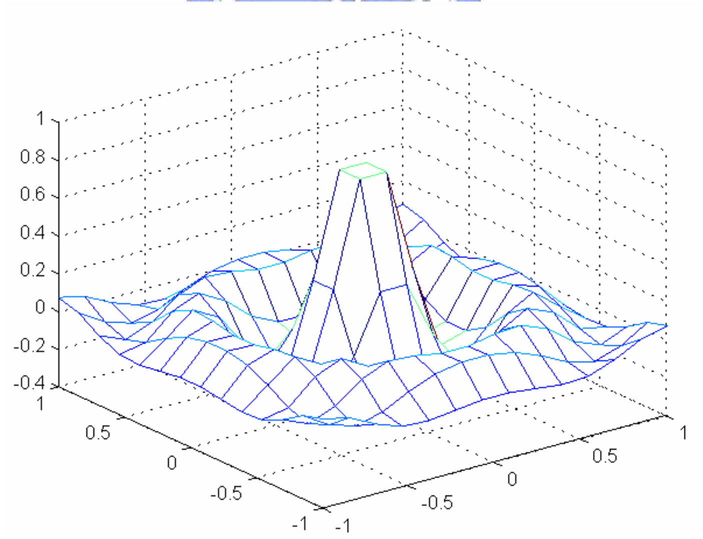


(a)

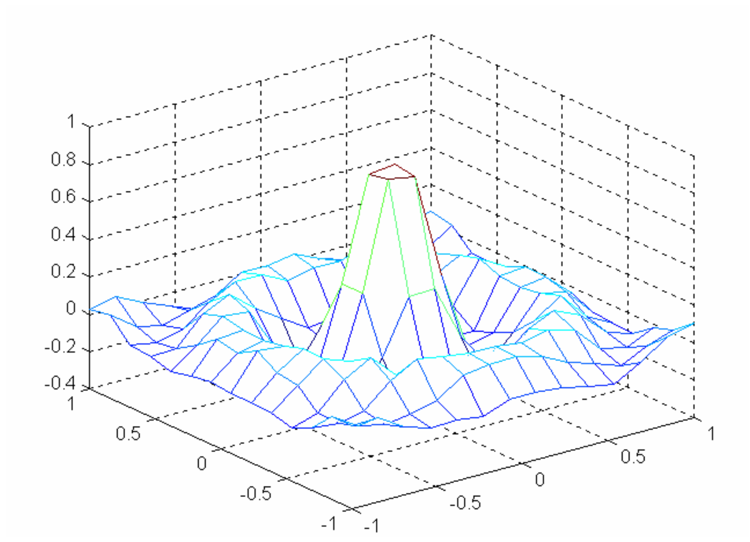


(b)

Fig 4.4 (a) The desired output of the function show in (4.10). (b) The resulting approximation by SVFNN.



(a)



(b)

Fig 4.5 (a) The desired output of the function show in (4.11). (b) The resulting approximation by SVFNN.

2) Training and Testing data:

There are two sets of training data for each function, one is noiseless and the other is noisy. In the first function, the noiseless training set has 50 points that are generated by randomly selecting, where $x \in [-10, \ 10]$. The testing set has 200 points that are randomly generated by the same function in the same range. The training and testing sets of the second function are generated by the same way, where $x \in [-2, \ 2]$. In the third function, the 150 training examples are generated by randomly selecting, where $x \in [-1, \ 1]$, $y \in [-1, \ 1]$. The testing set has 600 points that are randomly generated by the same function in the same range. In the fourth function, The 150 training examples are generated by randomly selecting, where $x \in [-1, \ 1]$, $y \in [-1, \ 1]$. The testing set has 600 points that is randomly generated by the same function in the same range. The noisy training sets are generated by adding independent and identically distributed (i.i.d.) Gaussian noise, with zero mean and 0.25 standard deviation, to the original training sets.

$$f_{noise}^{(j)}(x,y) = f^{(j)}(x,y) + 0.25\varepsilon_l, \quad j = 1, \ \cdots, \ 4. \tag{4.12}$$

Here $\varepsilon_l \sim N(0, \ 1)$, the zero mean unit variance Gaussian noise. It is noted that the signal to noise ratio (SNR) is roughly equal to 4 (1/0.25=4).

3) experimental particular

The computational experiments were done on a Pentium III-1000 with 1024MB RAM using the Microsoft window operation system. The simulations were conducted in the Matlab environment. The root-mean-square-error (RMSE) is used to quantify the performance of methods and it is defined as

$$\text{RMSE} = \sqrt{\sum_{i=1}^{v} (y_i - \hat{y}_i)^2 / v} \tag{4.13}$$

where y_i is the desired output, $\hat{y}_i$ is the system output, and v is the number of the used training or testing data. The ε-insensitivity parameter and cost parameter C in (4.6) are selected from the range of ε=[0.1, 0.01, 0.001, 0.0001] and C=[10^{-1}, 10^{0}, 10^{1}, …, 10^{5}], respectively. For the SVFNN training, we choose the ε-insensitivity parameter and cost parameter C that results in the best RMSE average to calculate the testing RMSE. Similarly, the parameters of SVR for comparison are also selected by using the same method, too.

B. Experimental Results

Tables 4.1 to 4.5 show the training and testing RMSEs and the number of used fuzzy rules (i.e., support vectors) in the SVFNN on the approximation of the four functions ((4.8) to (4.11)), respectively. The training and testing RMSEs can reach a nice level by selecting a proper parameter set for { ε , C }. The criterion of determining the number of reduced fuzzy rules is the difference of the accuracy values before and after reducing one fuzzy rule. If the difference is larger than 0.2%, meaning that some important support vector has been removed, then we stop the rule reduction. In Table 4.1 (a), the SVFNN is verified by the one-variable *sinc* function defined as (4.8), where the constant *n* in the symbol SVFNN-*n* means the number of the learned fuzzy rules. It uses sixteen fuzzy rules and achieves a root mean square error (RMSE) value of 0.0007 on the training data and an RMSE value of 0.0026 on the testing data. When the number of fuzzy rules is reduced to twelve, its testing error rate increased to 0.0029. When the number of fuzzy rules is reduced to eleven, its testing error rate is increased to 0.01. Continuously decreasing the number of fuzzy rules will keep the error rate increasing. Therefore, twelve fuzzy rules are used in this case. From Tables 4.2 (a) to 4.4 (a), we have the similar experimental results as those in Table 4.1 (a). Plots of these experimental results are shown in subplots (b) of Figs. 4.2-4.5. In Table 4.1 (b), the independent and identically distributed (i.i.d.) Gaussian noise, with zero mean and 0.25 standard deviation, is added to the function for approximation. It uses sixteen fuzzy rules and achieves a root mean square error (RMSE) value of 0.0085 on the training data and an RMSE value of 0.042 on the testing data. When the number of fuzzy rules is reduced to twelve, its testing error rate is increased to 0.045. When the number of fuzzy rules is reduced to eleven, its testing error rate is increased to 0.091. Therefore, twelve fuzzy rules are also used in this case. From Table 4.2 (b) to 4.4 (b), we have the similar experimental results as those in Table 4.1 (b) These experimental results show that the proposed SVFNN can properly reduce the number of required fuzzy rules and maintain the robustness against noise.

The performance comparisons among the Adaptive-network-based fuzzy inference system (ANFIS) [66], the robust neural network [67], the RBF-kernel-based SVR (without support vector reduction) [68], and the proposed SVFNN are made in Tables 4.5 and 4.6.

TABLE 4.1 (a) Experimental results of SVFNN on the first function using the training data without noise. (b) Experimental results of SVFNN on the first function using the training data with noise.

(a)

SVFNN-n (SVFNN with n fuzzy rules)	Training process		Testing process
	C	RMSE	RMSE
SVFNN – 16	100	0.0007	0.0026
SVFNN – 14	100	0.0007	0.0026
SVFNN – 12	100	0.0007	0.0029
SVFNN – 11	100	0.001	0.01
1. The first function is $f^{(1)}(x)=\frac{\sin(x)}{x}$ with $x\in[-10,\ 10]$. 2. The number of training data is 50. 3. The number of testing data is 200.			

(b)

SVFNN-n (SVFNN with n fuzzy rules)	Training process		Testing process
	C	RMSE	RMSE
SVFNN – 16	100	0.0085	0.042
SVFNN – 14	100	0.0085	0.042
SVFNN – 12	100	0.0085	0.045
SVFNN – 11	100	0.031	0.091
1. The first function is $f^{(1)}(x)=\frac{\sin(x)}{x}$ with $x\in[-10,\ 10]$. 2. The number of training data is 50. 3. The number of testing data is 200.			

TABLE 4.2 (a) Experimental results of SVFNN on the second function using the training data without noise. (b) Experimental results of SVFNN on the second function using the training data with noise.

(a)

SVFNN-*n* (SVFNN with *n* fuzzy rules)	Training process		Testing porcess
	C	RMSE	RMSE
SVFNN – 19	100	0.0009	0.0056
SVFNN – 16	100	0.0009	0.0056
SVFNN – 12	100	0.0009	0.0060
SVFNN - 11	100	0.0015	0.0092
1. The second function is $f^{(2)}(x) = x^{2/3}$ with $x \in [-2,\ 2]$. 2. The number of training data is 50. 3. The number of testing data is 200.			

(b)

SVFNN-*n* (SVFNN with *n* fuzzy rules)	Training process		Testing porcess
	C	RMSE	RMSE
SVFNN – 25	100	0.001	0.078
SVFNN – 20	100	0.001	0.078
SVFNN – 15	100	0.001	0.081
SVFNN - 14	100	0.0057	0.139
1. The second function is $f^{(2)}(x) = x^{2/3}$ with $x \in [-2,\ 2]$. 2. The number of training data is 50. 3. The number of testing data is 200.			

TABLE 4.3 (a) Experimental results of SVFNN on the third function using the training data without noise. (b) Experimental results of SVFNN on the third function using the training data with noise.

(a)

SVFNN-*n* (SVFNN with *n* fuzzy rules)	Training process		Testing process
	C	RMSE	RMSE
SVFNN- 33	1000	0.0018	0.0037
SVFNN- 24	1000	0.0018	0.0037
SVFNN- 17	1000	0.0018	0.0040
SVFNN- 16	1000	0.002	0.0089
1. The third function is $f^{(3)}(x,y)=\exp\{-2(x^2+y^2)\}$ with $x\in[-1,\ 1]$, $y\in[-1,\ 1]$. 2. The number of training data is 150. 3. The number of testing data is 600.			

(b)

SVFNN-*n* (SVFNN with *n* fuzzy rules)	Training process		Testing process
	C	RMSE	RMSE
SVFNN- 32	1000	0.018	0.051
SVFNN- 22	1000	0.018	0.051
SVFNN- 17	1000	0.018	0.054
SVFNN- 16	1000	0.045	0.121
1. The third function is $f^{(3)}(x,y)=\exp\{-2(x^2+y^2)\}$ with $x\in[-1,\ 1]$, $y\in[-1,\ 1]$. 2. The number of training data is 150. 3. The number of testing data is 600.			

TABLE 4.4 (a) Experimental results of SVFNN on the fourth function using the training data without noise. (b) Experimental results of SVFNN on the fourth function using the training data with noise.

(a)

SVFNN-*n* (SVFNN with *n* fuzzy rules)	Training process		Testing process
	C	RMES	RMES
SVFNN – 40	100	0.0059	0.0098
SVFNN – 30	100	0.0059	0.0098
SVFNN – 21	100	0.0063	0.01
SVFNN – 20	100	0.0099	0.032

1. The fourth function is $f^{(4)}(x,y)=\frac{\sin(10\sqrt{x^2+y^2})}{10\sqrt{x^2+y^2}}$ with $x\in[-1,\ 1]$, $y\in[-1,\ 1]$
2. The number of training data is 150.
3. The number of testing data is 600.

(b)

SVFNN-*n* (SVFNN with *n* fuzzy rules)	Training process		Testing process
	C	RMES	RMES
SVFNN – 45	100	0.01	0.071
SVFNN – 34	100	0.01	0.071
SVFNN – 22	100	0.01	0.073
SVFNN – 20	100	0.058	0.152

1. The fourth function is $f^{(4)}(x,y)=\frac{\sin(10\sqrt{x^2+y^2})}{10\sqrt{x^2+y^2}}$ with $x\in[-1,\ 1]$, $y\in[-1,\ 1]$
2. The number of training data is 150.
3. The number of testing data is 600.

TABLE 4.5 Comparisons RMSE using the training data without noise.

FUNCTION	ANFIS [66]		Robust NN [67]		RBF-kernel-based SVR [68]		SVFNN	
	Number of fuzzy rules	RMSE	Number of neurons	RMSE	Number of support vectors	RMSE	Number of Fuzzy rules	RMSE
$f^{(1)}(x)$	11	0.0071	12	0.0011	28	0.0018	12	0.0029
$f^{(2)}(x)$	11	0.0067	12	0.0047	50	0.0054	12	0.006
$f^{(3)}(x,y)$	9	0.0039	22	0.0035	122	0.0018	17	0.004
$f^{(4)}(x,y)$	16	0.015	35	0.0057	145	0.0092	21	0.01

TABLE 4.6 Comparisons RMSE using the training data with noise.

FUNCTION	ANFIS [66]		Robust NN [67]		RBF-kernel-based SVR [68]		SVFNN	
	Number of fuzzy rules	RMSE	Number of neurons	RMSE	Number of support vectors	RMSE	Number of Fuzzy rules	RMSE
$f_{noise}^{(1)}(x)$	15	0.726	12	0.053	49	0.035	12	0.045
$f_{noise}^{(2)}(x)$	12	0.5	12	0.07	49	0.07	15	0.081
$f_{noise}^{(3)}(x,y)$	9	0.305	22	0.056	139	0.04	17	0.054
$f_{noise}^{(4)}(x,y)$	16	1.76	30	0.09	150	0.062	22	0.073

4.4 Discussions

These results indicate that the SVFNN maintains the function approximation accuracy and uses less support vectors as compared to the regular SVR using fixed-width RBF kernels. The computational cost of the proposed SVFNN is also less than the regular SVR in the testing stage. In addition, according to Table 4.6 the testing results of SVFNN trained by the noisy data are close to results of SVFNN trained by the data without noise. It demonstrates that the proposed SVFNN have better robustness compared to ANFIS and the robust neural network, although the SVFNN uses little more rules compared with the ANFIS. In summary, the proposed SVFNN exhibits better generalization ability, maintains more robustness and uses less fuzzy rules.

CHAPTER 5
CONCLUSIONS

In this dissertation we proposed a support-vector-based fuzzy neural networks (SVFNNs) for solving more complex classification and function approximation problems. SVFNNs combines the superior classification power of support vector machine (SVM) in high dimensional data spaces and the efficient human-like reasoning of FNN in handling uncertainty information. The SVFNNs is the realization of a new idea for the adaptive kernel functions used in the SVM. The use of the proposed fuzzy kernels provides the SVM with adaptive local representation power, and thus brings the advantages of FNN (such as adaptive learning and economic network structure) into the SVM directly. SVFNNs combine the capability of good robustness against noise and global generalization of support vector learning and the efficient human-like reasoning of FNN in handling uncertainty information. A novel adaptive fuzzy kernel function is also proposed to bring the advantages of FNNs to the SVR directly and the use of the proposed fuzzy kernels provides the SVR with adaptive local representation power. The major advantages of the proposed SVFNNs are as follows:

(1) The proposed SVFNNs can automatically generate fuzzy rules, and improve the accuracy and learning speed of classification.

(2) It combined the optimal classification ability of SVM and the human-like reasoning of fuzzy systems. It improved the classification ability by giving SVM with adaptive fuzzy kernels and increased the speed of classification by reduced

fuzzy rules.

(3) The fuzzy kernels using the variable-width fuzzy membership functions can make the SVM more efficient in terms of the number of required support vectors, and also make the learned FNN more understandable to human.

(4) The ability of the structural risk minimization induction principle, which forms the basis for the SVM method to minimize the expected risk, gives better generalization ability to the FNN classification.

(5) The proposed SVFNN can automatically generate fuzzy rules and improve the accuracy of function approximation.

(6) The combination of the robust regression ability of SVR and the human-like reasoning of fuzzy systems improves the robust regression ability of FNN by using SVR training and increases the speed of execution by reduced fuzzy rules.

In the future work, we will try to develop a mechanism to automatically select the appropriate initial values of the parameters used in the first phase training and the penalty parameter in the second phase training. We will also apply the proposed method to deal with complex and huge classification problem and more complex and noisy functions.

REFERENCES

[1] K. Tanaka and H. O. Wang, *Fuzzy Control Systems Design and Analysis*, New York: Wiley, 2001.

[2] B. Kosko, *Neural Networks and Fuzzy Systems*, Englewood Cliffs, NJ: Prentice-Hall, 1992.

[3] M. Y. Chen and D. A. Linkens, “Rule-base self-generation and simplification for data-driven fuzzy models,” *Fuzzy Set and Syst.*, Vol. 142, pp 243-265, March 2004.

[4] J. S. Jang, “ANFIS: Adaptive-network-based fuzzy inference system,” *IEEE Trans. Syst. Man. Cybern.*, Vol. 23, pp. 665-685, May 1993.

[5] K. Tanaka, M. Sano, and H. Wantanabe, “Modeling and control of carbon monoxide concentration using a neuro-fuzzy technique,” *IEEE Trans. Fuzzy Syst.,* Vol. 3, pp. 271-279, Aug. 1995.

[6] L. Y. Cai and H. K. Kwan, “Fuzzy classifications using fuzzy inference networks,” *IEEE Trans. Syst., Man, Cybern. Pt B*, Vol. 28, pp. 334-347, June. 1998.

[7] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*, New York: Plenum, 1981.

[8] J. C. Bezdek, S. K. Chuah, and D. Leep, “Generalized K-nearest neighbor rules,” *Fuzzy Sets Syst.*, Vol. 18, pp. 237-256, Apr. 1986.

[9] J. S. Wang and C. S. G. Lee, “Self-Adaptive Neuro-Fuzzy Inference Systems for Classification Applications,” *IEEE Trans. Fuzzy Syst.,* Vol. 10, pp. 790-802, Dec. 2002.

[10] L. I. Kuncheva, “How good are fuzzy if-then classifiers?,” *IEEE Trans. Syst.,*

Man, Cybern. Pt B, Vol. 30, pp. 501-509, Aug. 2000.

[11] H. Ishibuchi and T. Nakashima "Effect of rule weights in fuzzy rule-based classification systems," *IEEE Trans. Fuzzy Syst.,* Vol. 9, pp. 506-515, Aug. 2001.

[12] W. Y. Wang, T. T. Lee, C. L. Liu, and C. H. Wang, "Function approximation using fuzzy neural networks with robust learning algorithm," *IEEE Trans. Syst., Man, Cybern. Pt B*, Vol. 27, pp. 740-747, Aug. 1997.

[13] C. C. Chuang, S. F. Su, and S. S. Chen, "Robust TSK fuzzy modeling for function approximation with outliers," *IEEE Trans. Fuzzy Syst.,* Vol. 9, pp. 810-821, Dec. 2001.

[14] H. Pomares, I. Rojas, J. Ortega, J. Gonzalez, and A. Prieto, "Systematic approach to a self-generating fuzzy rule-table for function approximation," *IEEE Trans. Syst., Man, Cybern. Pt B*, Vol. 30, pp. 431-447, June 2000.

[15] S. Wu, M. J. Er, and Y. Gao, "A fast approach for automatic generation of fuzzy rules by generalized dynamic fuzzy neural networks," *IEEE Trans. Fuzzy Syst.,* Vol. 9, pp. 578-594, Aug. 2001.

[16] B. Gabrys and A. Bargiela "General fuzzy min-max neural network for clustering and classification," *IEEE Trans. Neural Networks,* Vol. 11, pp. 769-783, May 2000.

[17] K. Nozaki, H. Ishibuchi, and H. Tanaka, "Adaptive fuzzy rule-based classification system" *IEEE Trans. Fuzzy Syst.,* Vol. 4, pp. 238-250, Aug. 1996.

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

[19] S. Sohn and C. H. Dagli, "Advantages of using fuzzy class memberships in self-organizing map and support vector machines," *Proc. International Joint Conference on Neural Networks* (IJCNN'01), Vol. 3, pp. 1886-1890, July 2001.

[20] C. F. Lin and S. D. Wang, "Fuzzy support vector machines," *IEEE Trans. Neural Networks*, Vol. 13, pp. 464-471, March 2002.

[21] T. Inoue and S. Abe, "Fuzzy support vector machines for pattern classification," *Proc. International Joint Conference on Neural Networks* (IJCNN'01), Vol. 2, pp. 15-19, July 2001.

[22] J. T. Jeng and T. T. Lee, "Support vector machines for the fuzzy neural networks," *IEEE International Conference on Systems, Man, and Cybernetics* (SMC'99), Vol. 6, pp. 12-15, Oct. 1999.

[23] C. F. Juang and C. T. Lin, "An on-line self-constructing neural fuzzy inference network and its applications," *IEEE Trans. Fuzzy Syst.*, Vol. 6, pp. 12-32, Feb. 1998.

[24] F. Hoppner, F. Klawonn, R. Kruse, and T. Runkler, *Fuzzy cluster analysis: methods for classification, data analysis and image recognition*, New York: Wiley, 1999.

[25] C. T. Lin and C. S. G. Lee "Neural-network-based fuzzy logic control and decision system," *IEEE Trans. Comput.*, Vol. 40, pp. 1320-1336, Dec. 1991.

[26] J. Platt, "A resource allocating network for function interpolation," *Neural Computat.*, Vol. 3, pp. 213-225, 1991.

[27] J. Nie and D. A. Linkens, " Learning control using fuzzified self-organizing radial basis function network," *IEEE Trans. Fuzzy Syst.*, Vol. 40, pp. 280-287, Nov. 1993.

[28] C. T. Lin and C. S. G. Lee, "Reinforcement structure/parameter learning for neural-network-based fuzzy logic control systems," *IEEE Trans. Fuzzy Syst.*, Vol. 2, pp. 46-63, Feb. 1994.

[29] A. Papoulis, *Probability Random Variables and Stochastic Processes*, McGraw

Hill, Inc., 1984.

[30] J. Mercer, “Functions of positive and negative type and their connection with the theory of integral equations,” *Philosophical Transactions of the Royal Society London*, A209, pp. 415-446,1909.

[31] S. Saitoh, *Theory of Reproducing Kernels and Its Application,* Longman Scientific & Technical.

[32] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods,* Cambridge University Press, 2000.

[33] R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge University Press, 1985.

[34] V. Vapnik, *Statistical Learning Theory,* New York: Wiley, 1998.

[35] V. N. Vapnik, *The Nature of Statistical Learning Theory*, New York: Springer-Verlag, 1990.

[36] B. Schölkopf, C. J. C. Burges, and A. J. Smola, *Advances in Kernel Methods—Support Vector Learning,* Cambridge, MA: MIT Press, 1999.

[37] J. Hohensoh and J. M. Mendel, “Two-pass orthogonal least-squares algorithm to train and reduce the complexity of fuzzy logic systems,” *Journal of Intelligent and Fuzzy Systems*, vol. 4, pp. 295-308, 1996.

[38] G. Mouzouris and J. M. Mendel, “A singular-value-QR decomposition based method for training fuzzy logic systems in uncertain environments,” *Journal of Intelligent and Fuzzy Systems*, vol. 5, pp. 367-374, 1997.

[39] J. Yen and L. Wang “Simplifying fuzzy rule-based models using orthogonal transformation methods,” *IEEE Trans. Syst., Man, Cybern. Pt B,* Vol. 29, pp. 13-24, Feb. 1999.

[40] C. J. C. Burges, “Simplified support vector decision rules,” *in Proc. 13th Int.*

Conf. Machine Learning, L. Saitta, Ed. San Mateo, CA:Morgan Kaufmann, 1996, pp. 71-77.

[41] B. Scholkopf, S. Mika, C. Burges, etc "Input space versus feature space in kernel-based methods," *IEEE Trans. Neural Networks,* Vol. 10, pp.1000-1017, Sep. 1999.

[42] C. L. Blake and C. J. Merz, (1998) UCI repository of Machine Learning Databases, Univ. California, Dept. Inform. Comput. Sci., Irvine, CA. [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html.

[43] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, (1994) Machine Learning, Neural and Statistical Classification [Online]. Available: ftp://ftp.stams.strath.ac.uk/pub/.

[44] D. Prokhorov. IJCNN 2001 neural network competition. presented at Slide Presentation in IJCNN'01. [Online] http://www.geocities.com/ijcnn/nncijcnn01.pdf

[45] [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/binary/

[46] C. W. Hsu and C. J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE Trans. Neural Networks,* Vol. 13, pp. 415-525, March 2002.

[47] Y. J. Lee and O. L. Mangasarian, "RSVM: reduced support vector machines," *Proc. 1st SIAM Int. Conf. Data mining*, 2001.

[48] K. M. Lin and C. J. Lin, "A study on reduced support vector machines," *IEEE Trans. Neural Networks,* Vol. 14, pp.1449-1459, Nov. 2003.

[49] H. M. Lee, C. M. Chen, J. M. Chen, and Y. L. Jou "An efficient fuzzy classifier with feature selection based on fuzzy entropy," *IEEE Trans. Syst., Man, Cybern. Pt B,* Vol. 31, pp. 426-432, June 2001.

[50] M. R. Berthold and J. Diamond, "Constructive training of probabilistic neural

networks," *Neurocomputing,* Vol. 19, pp. 167-183, 1998.

[51] D. S. Chen and R. C. Jain, "A robust back-propagation learning algorithm for function approximation," *IEEE Trans. Neural Networks*, Vol. 5, pp. 467-479, May 1994.

[52] K. Liano, "Robust error measure for supervised neural network learning with outliers," *IEEE Trans. Neural Networks*, Vol. 7, pp. 246-647, Jan. 1996.

[53] C. C. Lee, P. C. Chung, J. R. Tsai, and C. I. Chang, "Robust radial basis function neural networks," *IEEE Trans. Syst., Man, Cybern. Pt B*, Vol. 29, pp. 674-685, Dec. 1999

[54] C. C. Chuang, S. F. Su, and C. C. Hsiao, "The annealing robust backpropagation (ARBP) learning algorithm," *IEEE Trans. Neural Networks*, Vol. 11, pp. 1067-1077, Sep. 2000.

[55] C. C. Chuang, J. T. Jeng, and P. T. Lin, "Annealing robust radial basis function networks for function approximation with outliers" *Neurocomputing*, Vol. 56, pp. 123-139, 2004.

[56] M. Figueiredo and F. Gomide, "Design of fuzzy systems using neurofuzzy networks," *IEEE Trans. Neural Networks*, Vol. 10, pp. 815-827, July 1999.

[57] C. C. Chuang, S. F. Su, J. T. Jeng, and C. C. Hsiao, "Robust support vector regression network for function approximation with outliers," *IEEE Trans. Neural Networks*, Vol. 13, pp. 1322-1330, Nov. 2002.

[58] J. H. Chiang, and P. Y. Hao, "Support vector learning mechanism for fuzzy rule-based modeling: a new approach," *IEEE Trans. Fuzzy Syst.,* Vol. 12, pp. 1-12, Feb. 2004.

[59] Z. Sun and Y. Sun, "Fuzzy support vector machine for regression estimation," *IEEE International Conference on Systems, Man, and Cybernetics* (SMC'03),

Vol. 4, pp. 3336-3341, Oct. 2003.

[60] M. Setnes, R. Babuska, and H. B. Verbruggen, “Rule-based modeling: precision and transparency,” *IEEE Trans. Syst., Man, Cybern. Pt C*, Vol. 28, pp. 165-169, Feb. 1998.

[61] C. T. Lin, C. M. Yeh, S. F. Liang, J. F. Chung, and N. Kumar, “Support vector based fuzzy neural network for pattern classification,” *IEEE Trans. Fuzzy Syst.*, Vol. 14, pp. 31-41, Feb. 2006.

[62] B. Schölkopf, P. Simard, A. J. Smola, and V. N. Vapnik, “Prior knowledge in support vector kernels,” *Advances in Neural Information Processing Systems,* Vol. 10, MIT Press, Cambridge, MA.

[63] V. Vapnik, S. Golowich, and A. J. Smola, “Support vector method for function approximation, regression estimation, and signal processing,” in Neural Information Processing Systems. Cambridge, MA: MIT Press, vol. 9. 1997.

[64] K. Liano, “Robust error measure for supervised neural network learning with outliers,” *IEEE Trans. Neural Networks,* Vol. 7, pp.246-250, Jan. 1996.

[65] A. Suarez and J. F. Lutsko, “Globally optimal fuzzy decision trees for classification and regression,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 21, pp. 1297-1311, Dec. 1999

[66] J. S. R. Jang, C. T. Sun, and E. Mizutani, *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*, Prentice-Hall, Upper Saddle River, NJ, 1997.

[67] F. D. Foresee and M. T. Hagan, “Gauss-newton approximation to bayesian learning,” *Proc. International Joint Conference on Neural Networks* (IJCNN’97), Vol. 3, pp. 1930-1935, Jun 1997.

[68] S. R. Gunn. (1999) Support vector regression-Matlab toolbox. Univ.

Southampton, Southampton, U. K.. [Online]. Available: http://kernel-machines.
org.

LISTS OF PUBLICATION

著作目錄

姓名：葉長茂(Chang-Mao Yeh)

已刊登或被接受之期刊論文：(總共 4.2 點)

[1] C.T. Lin, C. M. Yeh, J. F. Chung, S. F. Liang, and H. C. Pu, "Support vector based fuzzy neural networks," *International Journal of Computational Intelligence Research*, Vol. 1, pp. 138-150, Dec. 2005. (1.2 點)

[2] C.T. Lin, C. M. Yeh, S. F. Liang, J. F. Chung, and N. Kumar, "Support vector based fuzzy neural network for pattern classification," *IEEE Trans. Fuzzy Syst.*, Vol. 14, pp. 31-41, Feb. 2006. (2.4 點)

[3] C. T. Lin, K. W. Fan, C. M. Yeh, H. C. Pu, and F. Y. Wu, "High-accuracy skew estimation of document images," *International Journal of fuzzy systems*, Vol. 8, pp. 119-126, Sep. 2006. (0.6 點)

待審之期刊論文：

[1] C. T. Lin and C. M. Yeh, "Self-tuning error correcting output coding support vector machine for multi-class classification," submitted to *IEEE Trans. Systems, Man, and Cybernetics Part B*

[2] C. T. Lin and C. M. Yeh, and S. F. Liang, "Support-vector based fuzzy neural network for function approximation," submitted to *IEEE Trans. Neural Networks*

研討會論文：

[1] C.T. Lin, C. M. Yeh, and C. F. Hsu, "Fuzzy neural network classification design using support vector machine," *IEEE International Conference on Circuits and Systems*, pp.724-727, May 2004.

[2] C.T. Lin, C. M. Yeh, H. C. Pu, and S. F. Liang,"Skew estimation of document images using fuzzy c-regression models," *Conference of computer vision, Graphics, and image processing*, pp.38-43, Aug. 2004.

[3] C.T. Lin, S. F. Liang, C. M. Yeh, and K. W. Fan, “Fuzzy neural network design using support vector regression for function approximation with outliers,” *IEEE International Conference on Systems, Man, and Cybernetics*, pp2763-2768. Oct. 2005.

VITA

博士候選人學經歷資料

姓名: 葉長茂

性別: 男

生日: 中華民國 60 年 7 月 24 日

籍貫: 台灣省彰化縣

論文題目: 中文: 支持向量模糊神經網路及其應用

英文: Support-Vector based Fuzzy Neural Networks and its applications

學歷:

1. 民國 83 年 6 月 國立雲林科技大學 電子系畢業。
2. 民國 86 年 6 月 國立雲林科技大學 電子與資訊研究所畢業。
3. 民國 90 年 9 月 國立交通大學電機及控制工程研究所博士班。

經歷:

1. 民國 86 年起至今 中州技術學院 講師