A PIECEWISE LINEAR CLASSIFIER

by

ABDUL AZIZ ABDURRAB

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2007

ACKNOWLEDGEMENTS

It is difficult to overstate my gratitude to my M.S. supervisor, Dr. Michael T. Manry. Throughout my research period, he provided encouragement, sound advice, good teaching, good company, and lots of good ideas that helped make my research easy and interesting. I am thankful for the countless hours he invested in painstakingly revising drafts for the thesis.

I thank Dr. Raymond R. Shoults and Dr William E. Dillon for reviewing my work and also for agreeing to serve on my thesis committee.

I am indebted to my many student colleagues, both graduate and undergraduate, for providing a stimulating environment to learn and grow.

I thank my peers and supervisors at Qualcomm, Inc. for being a constant source of motivation, during my internship.

I would like to express my deepest appreciation to my aunts and uncles - Khadija Khan and Dr. M. A. Sattar Khan, Qudsia Ahmad and Dr. Mohammad Ahmad, Amena Hussaini and Sajjad Hussaini - and Ruqia Ali and Syed H. Ali who helped make my stay miles away from home much easier.

I dedicate this thesis to my brother, sister and parents, Abdurrab Bin Mohammed and Khursheed Unnisa Begum, whose prayers made this possible.

Lastly, I offer my obeisance to Allah S.W.T for guiding me from within.

April 10, 2007

ABSTRACT

A PIECEWISE LINEAR CLASSIFIER

Publication No. _____

Abdul Aziz Abdurrab, MSEE

The University of Texas at Arlington, 2007

Supervising Professor: Dr. Michael T. Manry

A piecewise linear network is discussed which classifies N-dimensional input vectors. The network uses a distance measure to assign incoming input vectors to an appropriate cluster. Each cluster has a linear classifier for generating class discriminants. A training algorithm is described for generating the clusters and discriminants. A pruning algorithm is also described. The algorithm is applied after the network has grown completely, i.e, it has achieved the maximum number of clusters. The pruning algorithm eliminates the least important clusters, one at a time, leading to a more compact network. Theorems are given which relate the network's performance to that of nearest neighbor and k-nearest neighbor classifiers. It is shown that the error approaches Bayes Error as the number of clusters and patterns per cluster approach infinity. The mathematical complexity of the piecewise linear network classifier, in

terms of number of multiplies, is compared against those of classical neural net classifiers, like the multi-layer perceptron and the nearest neighbor classifier. The classifier is also compared with these classifiers with respect to their sizes, i.e, number of clusters or hidden units. It is shown that the piecewise linear network classifier generally outperforms on both fronts.

TABLE OF CONTENTS

Appendix

LIST OF ILLUSTRATIONS

CHAPTER 1

INTRODUCTION

The ease with which humans recognize a face, comprehend spoken words, read handwritten characters, identify and distinguish many things by feel disguises the astonishingly complex process of pattern recognition and classification. Pattern classification is the act of taking in raw data and deciding on the "category" of the data. Over the past many years, humans have evolved highly sophisticated neural and cognitive systems for such tasks. It is only natural that we should seek to design and build machines that can recognize and classify patterns. From automated speech recognition, fingerprint identification, optical character recognition, DNA sequence matching, and much more, it is obvious that reliable, accurate pattern recognition by machines is immensely useful.

1.1 Classifier Design Problem

A problem common to many disciplines is classification of data or patterns based on *a-priori* knowledge and on statistical information extracted from the patterns. The patterns to be classified are usually groups of measurements or observations, defining points in an appropriate multidimensional input space. A complete pattern recognition system consists of a preprocessor, e.g., a sensor that gathers the observations to be classified or described, a feature extraction mechanism that computes

numeric or symbolic information from the observations, and a classification scheme that does the actual job of classifying observations, relying on the extracted features. The task of the classifier component proper of a full system is to use the feature vectors provided by the feature extractor to assign the object to a category. A typical classifier design problem can be split into two parts, training and evaluation. In general, the process of using data to develop the classifier is referred to as training the classifier. The most effective methods for developing classifiers involve learning from a set of example patterns that have already been classified. This set of patterns is termed the training dataset and the resulting learning strategy is characterized as supervised learning [3,4]. A training dataset usually consists of N_v labeled feature vectors $\mathbf{x}_{fp}$, each of dimension N. Each feature vector has its class label, $i_c(p)$, defined, where $1 \leq p \leq N_v$. The goal is to design a classifier that estimates $i_c(p)$ from $\mathbf{x}_{fp}$, given the training data ($\mathbf{x}_{fp}$, $i_c(p)$). Learning can also be unsupervised [3,4], in the sense that the system is not given *a-priori* labeling of patterns, instead it establishes the classes itself based on the statistical regularities of the patterns. Evaluation of the classifier is important both to measure the performance of the system and to identify the need of improvement in its components. While an overly complex system may allow perfect classification of the training samples, it is unlikely to perform well on new patterns. This situation is known as overfitting [4]. One of the most important areas of research in statistical pattern classification is determining how to adjust the complexity of the model- not so simple that it can not explain the differences between the categories, yet not so complex as to give poor classification on novel patterns.

1.2 Conventional and Neural Net Classifiers

Conventional classifiers like the Bayes [3,4] and k-nearest neighbor classifiers (k-NNC) [3,4] have long been used. A Bayes classifier is a simple probabilistic classifier that makes the final decision by combining two sources of information, i.e., the prior and the likelihood, to form a posterior probability using Bayes' rule [5].

The k-NNC classifies objects based on the closest training examples in the *N*-dimensional feature space. In the training phase, the feature space is divided into convex polygons or clusters based on the class labels of the various training patterns. This leads to partitioning of the input space into a Voronoi tessellation [46] as shown in fig. 1.1. In the classification phase, distances from the new test vector to all the stored vectors are computed and the *k* closest samples are selected. The new vector is predicted to belong to the most numerous class labels within this set. The best choice of *k* depends on the data; generally larger values reduce the effect of noise on classification but make the decision boundaries less distinct. The algorithm is easy to implement, but can get computationally intense, especially when the size of the training set increases.

Several artificial neural networks have also been used for classification purposes. Neural nets can either undergo supervised learning or unsupervised learning, as in self-organizing map (SOM) networks [10,18]. In supervised learning, there exist the input feature vector, $\mathbf{x}_{fp}$ and the feature vector's class label, $i_c(p)$. Multi-layer Perceptrons (MLP) [6], radial basis function (RBF) networks [7] and support vector machines (SVM) [8,9] are trained using supervised learning techniques. Neural net

classifiers are usually trained to minimize the Mean-Squared Error (MSE) over a number of iterations.

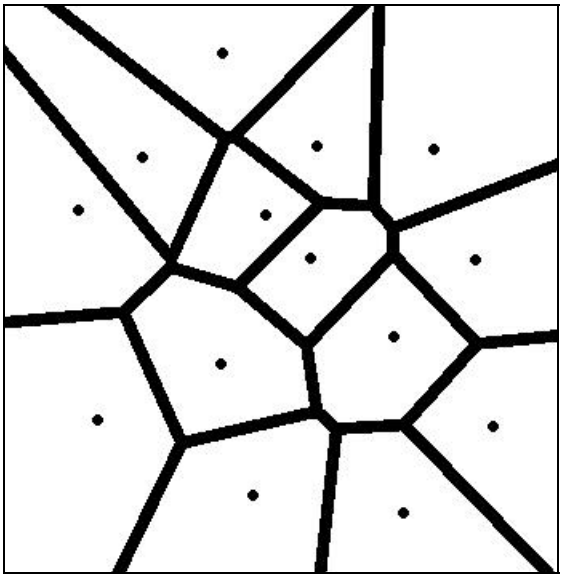

Fig. 1.1 Voronoi tessellation of a two-dimensional space

1.3 Introduction to the Piecewise Linear Classifier

Feedforward neural nets with the universal approximation property [19,20] mimic Bayes discriminants [11,12] and have been successfully used for many classification tasks. However, training time is slow, and convergence of the classification error to Bayes error has not been shown. Such convergence theorems do exist for nearest neighbor classifiers (NNCs) and k-NNCs [3,4], which also have the advantage of being easy to design in a short period of time. However, the NNC and k-NNC are rarely used because they are very time-consuming to apply. Piecewise linear networks (PLNs) have long been used for function approximation and classification

tasks [21-25] where speed of operation and simplicity are very important. One design approach is training an MLP having piecewise linear activations [26,27]. This approach is useful in hardware implementations, speeds up training as compared to backpropagation [47-49], and results in a continuous approximation. If we are willing to give up continuous approximation, a simpler piecewise linear network can be devised. Suppose that a distance measure is used to partition the feature space into a Voronoi tessellation. For each partition or cluster, a linear network maps that clusters members to the output space. This approach has been analyzed in detail for the approximation case [13].

1.4 Objectives and Overview of this Thesis

In this thesis we develop a piecewise linear network classifier (PLNC), based upon the work in [13], which can be quickly designed and applied. Chapter 2 presents a review on various existing neural net classifiers. Chapter 3 discusses the structure, operation and training of the PLNC. It also discusses pruning techniques that can lead to more compact PLNCs. In chapter 4, the estimation of Bayesian *a-posteriori* probabilities by the PLNC is discussed, along with the PLNC's relation to the NNC and k-NNC. Mathematical complexity of the PLNC, in terms of number of multiplies, against the nearest neighbor classifier (NNC) and the MLP are discussed in chapter 5. Simulations and performance comparisons against the NNC and the MLP, with respect to mathematical complexity and network sizes, are presented in chapter 6. Chapter 7 concludes this thesis and puts forth proposals for future research work.

CHAPTER 2

REVIEW OF NEURAL NET CLASSIFIERS

An optimal classifier can be designed based on the Bayes' rule [5] but this would require us to know the prior probabilities, P_i and the class conditional densities, $f(x|i)$. Unfortunately, in pattern classification applications, this kind of complete knowledge about the probabilistic structure of the problem is rarely, if ever, available. In a typical case we merely have some vague, general knowledge about the situation, together with a number of design samples or training data- particular representatives of the patterns we want to classify. The problem, then, is to find some way to use this information to design or train the classifier.

Many approaches have been developed to solve this problem. A few of these include Bayesian estimation [4], maximum likelihood estimation [42], and nonparametric design techniques such as the nearest neighbor rule and artificial neural networks [4]. Neural nets have the advantage that they do not require knowledge of the underlying probability distributions and in this limited sense they can be said to be nonparametric. They are usually trained to minimize the cost function [4], which in most cases is the mean squared error (MSE). In this chapter, we review three neural nets that have been widely used as classifiers and state their advantages and disadvantages.

2.1 Multi-Layer Perceptrons

This class of networks consists of multiple layers of computational units or nodes, usually interconnected in a feed-forward way. In many applications the hidden units of these networks apply a sigmoid function as an activation function at the unit's output. The structure of the multi-layer perceptron (MLP) is shown in fig. 2.1.

Multi-layer perceptrons use a variety of learning techniques, the most popular being backpropagation[47-49]. Here the computed output values are compared with the desired output to compute the value of an error function. The error is then fed back through the network, which is then reduced by adjusting the weights by a general optimization technique like the gradient descent [14].

In general the problem of teaching a network to perform well, even on samples that were not used as training samples, is a quite subtle issue that requires additional techniques. This is especially important for cases where only very limited numbers of training samples are available. The danger is that the network overfits the training data and fails to capture the true statistical process generating the data. Computational learning theory is concerned with training classifiers on a limited amount of data. In the context of neural networks a simple heuristic, called early stopping [50,51], often ensures that the network will generalize well to examples not in the training set. Other typical problems of the backpropagation algorithm are the speed of convergence and the possibility of ending up in a local minimum of the error function.

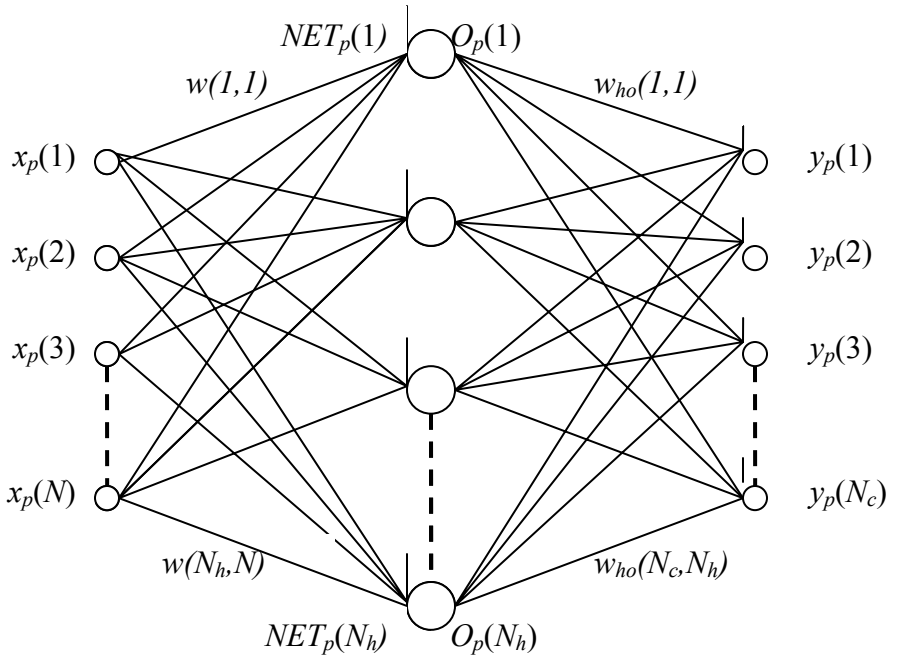

Fig. 2.1 Stucture of a multi-layer perceptron

2.2 Radial Basis Function Networks

A radial basis function (RBF) is a real-valued function whose value depends only on the distance of its input vector from the origin. RBF Networks are used in function approximation, time series prediction, and control. In neural networks, radial basis functions are utilized as hidden units. RBF networks typically have 3 layers, the input layer, the hidden layer with the RBF non-linearity and a linear output layer. The input is first mapped onto each RBF in the hidden layer. The RBF activation chosen is usually a Gaussian. In regression problems the output layer is then a linear combination of hidden layer values representing mean predicted output. The interpretation of this output layer value is the same as a regression model in statistics. In classification

problems the output layer is typically a sigmoid function of a linear combination of hidden layer values, representing a posterior probability.

RBF networks have the advantage of not suffering from local minima in the same way as MLPs. This is because usually the only parameters that are adjusted in the learning process are the weights from the hidden layer to the output layer. Linearity ensures that the error surface is quadratic and therefore has a single easily found minimum.

RBF networks have the disadvantage of requiring good coverage of the input space by radial basis functions. RBF centers are determined with reference to the distribution of the input data, but without reference to the prediction task. As a result, representational resources may be wasted on areas of the input space that are irrelevant to learning. A common solution is to associate each data point with its own center, although this can make the linear system to be solved in the final layer rather large, and requires shrinkage techniques to avoid overfitting [4].

2.3 Support Vector Machines

Support vector machines (SVMs) are a set of related supervised learning networks used for classification and regression. They usually consist of a feature extractor containing RBF hidden units, followed by a classifier that makes decisions based on a linear combination of features. A special property of SVMs is that they minimize the empirical classification error and maximize the geometric margin between the various classes. A typical classification problem involves separating *N*-dimensional data into different classes by an *(N-1)*-dimensional hyperplane. This could be done

using a typical form of linear classifier. Possible boundaries for such a classification problem are shown in fig. 2.2. However, if it is also desired to achieve maximum separation between the different classes, this could be obtained using an SVM or other maximal margin classifier [11]. By maximizing this margin, SVMs avoid overfitting. A maximum-margin hyperplane for an SVM trained with samples from two classes is shown in fig. 2.3. Samples along the hyperplanes are called the support vectors.

The parameters of the maximal margin hyperplane are commonly derived by solving a quadratic programming (QP) optimization problem using Platt's Sequential Minimal Optimization (SMO) algorithm [15,16]. This algorithm breaks the problem down into 2-dimensional sub-problems that may be solved analytically, eliminating the need for a numerical optimization algorithm such as the conjugate gradient method [17].

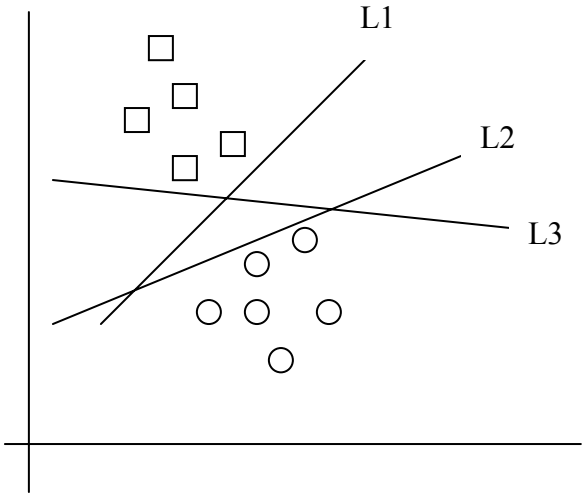

Fig. 2.2 Possible boundaries to a simple classification problem

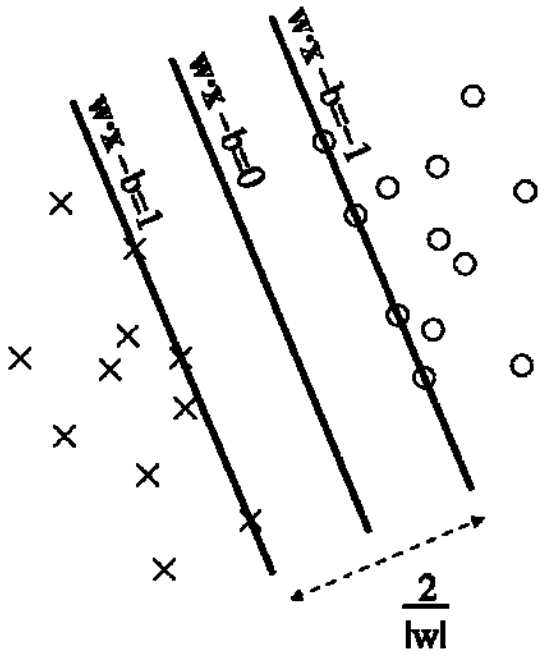

Fig. 2.3 A maximum-margin hyperplane for a support vector machine

2.4 Problems with Classifiers

Neural net classifiers have several problems. Training time for MLP and RBF classifiers can be long and they may suffer from overfitting [4]. SVM classifiers avoid overfitting but usually require several orders of magnitude more hidden units than RBF and MLP networks.

Conventional classifiers also have problems. Bayes-Gaussian classifiers require accurate input statistics and Gaussian input vectors, both of which may be unavailable. Nearest neighbor classifiers, like SVMs, frequently require hundreds or thousands of parameters, and can take too long to apply.

CHAPTER 3

THE PIECEWISE LINEAR CLASSIFIER

In this chapter, a PLN [13] classifier (PLNC) is introduced which solves the problems discussed in section 2.4.

3.1 Network Structure and Operation

The network structure is shown in fig. 4.1. The PLNC consists of three layers: the input elements form the first layer, the hidden units the second and the output units the third. An *(N+1)*-dimensional vector **x** forms the input to the PLNC. Vector **x** is obtained from the *N*-dimensional feature vector $\mathbf{x}_f$. The means and standard deviations of the feature vector elements x_{fn} are respectively μ_n and σ_n, where $1 \le n \le N$. The feature vector elements are first normalized as

$$x_{fn} \leftarrow \left[x_{fn} - \mu_n \right] / \sigma_n \tag{3.1}$$

The normalized feature vector is then augmented as

$$\mathbf{x} = \left(\mathbf{x}_f{}^{\mathrm{T}} : 1 \right)^{\mathrm{T}} \tag{3.2}$$

to form the *(N+1)*-dimensional input, **x**, to the PLNC. The hidden layer consists of *K* clusters, each cluster having its *N*-dimensional cluster mean vector $\mathbf{m}_c$, where $1 \le c \le K$. A Euclidean distance measure d(·)

$$d\left(\mathbf{x}, \mathbf{m}_c\right) = \sum_{n=1}^{N} \left[x_n - m_{cn} \right]^2 \tag{3.3}$$

is used to do the clustering. x_n is the n^{th} element of vector **x**, and m_{cn} is the n^{th} element of the vector $\mathbf{m}_c$. Each cluster also has a weight matrix $\mathbf{A}_c$ of dimension N_c by *(N+1),* where N_c is the number of classes in the classification problem.

The output of the network, **y**, comprises of N_c elements. The vector **y** is calculated by multiplying the input vector with the weight matrix of the cluster it has been assigned to. Given an input vector **x**, we find *c* such that d(**x**, $\mathbf{m}_c$) is minimized. Then we form the output vector **y** as

$$\mathbf{y} = \mathbf{A}_c \cdot \mathbf{x} \tag{3.4}$$

The estimate of the correct class i_c is given by

$$i_c' = \arg\max_i \left[y_i \right] \tag{3.5}$$

where y_i is the i^{th} element of the output vector **y** and $1 \le i \le N_c$.

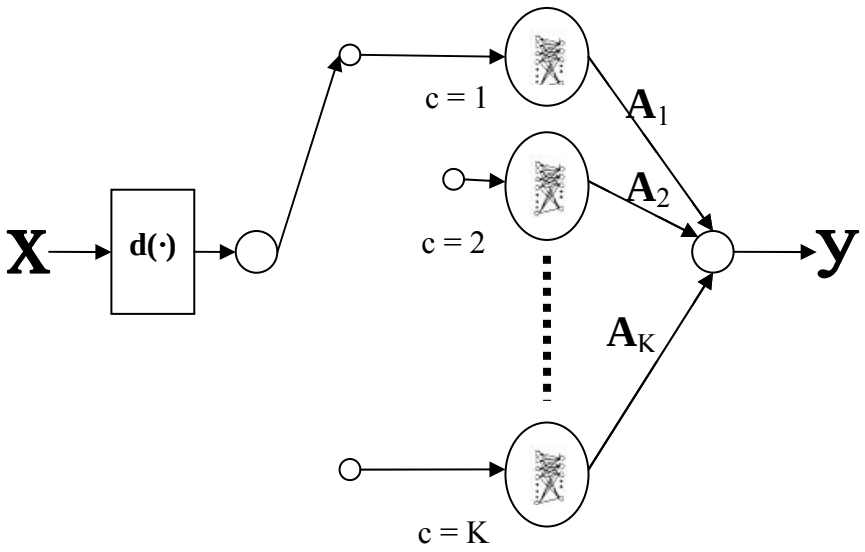

Fig. 3.1 Structure of a Piecewise Linear Classifier

3.2 Network Training

A classification problem typically involves a feature space with numerous feature vectors or samples that have to be classified into various class labels. In supervised learning, the training dataset includes the class label, i_c, of each of the N_v feature vectors. The label is transformed into an N_c-dimensional target vector **t** such that

$$t_{pi} = \begin{cases} +b & i = i_c(p) \\ -b & \text{otherwise} \end{cases} \tag{3.6}$$

where $1 \le p \le N_v$, $1 \le i \le N_c$ and b is any positive integer. Before the network can be used for classification itself, it has to be trained. Training involves designing the PLNC weight matrices given numerous training patterns.

The process of training a PLNC is divided into two parts. The first part involves partitioning of the input feature space into K clusters. The second part involves the calculation of the network weights by solving a set of linear equations whose solution minimizes the MSE of the network. The techniques used to perform these two parts of the training algorithm are mentioned in subsections 3.2.1 and 3.2.2 respectively.

3.2.1 Clustering

The idea behind performing clustering is to group like feature vectors together, based on their inherent statistical properties, so that a simple linear classifier could be used for classification within every cluster. This would result in an overall complex decision boundary. Of the various techniques available [53-55], we use Kohonen's self organizing maps (SOM) to do clustering. SOM partitions feature vectors

into clusters based on their distances from the cluster mean vectors. More information on SOM is available in appendix A.

3.2.2 Solving for $\mathbf{A}_c$

Clustering results in partitioning of the feature space into K clusters. Each cluster has its own weight matrix $\mathbf{A}_c$, where $1 \le c \le K$. For a PLNC having linear output layer activations, the output vector is calculated by simply multiplying the input vector and the weight matrix of the cluster to which the pattern was assigned, as described in section 3.1. Hence, only the winning cluster's weight matrix is turned on in order to generate the output vector. Elaborating on (3.4), the elements of the output vector $\mathbf{y}$, are calculated as

$$y_{pi} = \sum_{n=1}^{N+1} a_{cin} \cdot x_{pn} \tag{3.7}$$

where a_{cin} is the element of the weight matrix $\mathbf{A}_c$, belonging to row i and column n. The elements of the output vector evaluated in (3.7) are compared with the N_c-dimensional desired output vector $\mathbf{t}_p$, described in (3.6). The mean squared error (MSE) for the c^{th} cluster is given by

$$E_c = \frac{1}{N_v(c)} \sum_{q=1}^{N_v(c)} \sum_{i=1}^{N_c} \left[t_{qi} - \sum_{n=1}^{N+1} a_{cin} x_{qn} \right]^2 \tag{3.8}$$

where $N_v(c)$ is the number of feature vectors in cluster c and t_{qi} is the actual output which could be $+b$ or $-b$. The error gradient with respect to elements of the weight matrix $\mathbf{A}_c$ is given by

$$\frac{\partial E}{\partial a_{cjm}} = -2\frac{1}{N_v(c)}\sum_{q=1}^{N_v(c)}\left[t_{qj} - \sum_{n=1}^{N+1} a_{cjn}x_{qn}\right]x_{qm} \tag{3.9}$$

On further simplification we get

$$\frac{\partial E}{\partial a_{cjm}} = -2\frac{1}{N_v(c)}\left[\left(\sum_{q=1}^{N_v(c)} t_{qj}x_{qm}\right) - \left(\sum_{q=1}^{N_v(c)}\sum_{n=1}^{N+1} a_{cjn}x_{qn}x_{qm}\right)\right] \tag{3.10}$$

Equation (3.10) can be written in terms of auto-correlation and cross-correlation elements, and respectively, as

$$\frac{\partial E}{\partial a_{cjm}} = -2\left[c(j,m) - \sum_{n=1}^{N+1} a_{cjn}r(n,m)\right] \tag{3.11}$$

The PLNC is designed by minimizing the training MSE. Hence, equating the gradient in (3.11) to zero, in order to minimize the error, yields

$$c(j,m) = \sum_{n=1}^{N+1} a_{cjn}r(n,m) \tag{3.12}$$

Several techniques [14,17,56-59] can be used to solve the set of equations in (3.12). The technique described in appendix B is a modified version of the Gram-Schmidt procedure [28,29].

3.2.3 Classification Error of the PLNC

Upon obtaining the weight matrices for all clusters, the training is complete and these weights can be used to compute the outputs of the network in response to the input vectors. The predicted class label, $i_c'(p)$ for vector $\mathbf{x}_p$ satisfies $1 \le i_c'(p) \le N_c$. If $i_c'(p)$ is not equal to the correct class label, $i_c(p)$, the PLNC is said to have incorrectly classified

the feature vector $\mathbf{x}_f$. The classification error percentage of the PLNC is one hundred times the total number of such incorrect classifications, divided by N_v.

3.3 Review of Output Reset Algorithm

The output of the PLNC, at a single node, can be considered to be the sum of an optimal output and noise, which is usually zero-mean. The noise contributes to the error in the classifier. Each instance of residual error contains at least two types of bias. The first type of bias a_p is an additive constant, inherent to each output vector. The second type of bias d_{pi} has error components due to individual output values having the correct sign but magnitude larger than b. Removal of these biases has no immediate effect on the class recognition error count. However, after removing a_p and d_{pi}, over several training iterations, Output Reset (OR) [30-32] training error E' more closely models the class recognition error rate.

We mechanize the removal of biases by introducing a new desired output vector, $\mathbf{t}_p'$, which accounts for both a_p and d_{pi}. The OR training error to be minimized is

$$E' = \frac{1}{N_v} \sum_{i=1}^{N_c} \sum_{p=1}^{N_v} \left[t'_{pi} - y_{pi} \right]^2 \tag{3.13}$$

where $t_{pi}' = t_{pi} + a_p + d_{pi}$ and d_{pi} is a function of p and i. Our goal now is to find a_p, d_{pi} and y_{pi}, that minimize E'.

A sufficient condition for finding a_p is that the gradient of E' with respect to z_p be zero, yielding

$$a_p = \frac{1}{M} \sum_{i=1}^{N_c} \left[z_{pi} - d_{pi} \right] \tag{3.14}$$

Values for d_{pi} are found by minimizing square error term $[d_{pi} + a_p - z_{pi}]^2$, yielding $d_{pi} = z_{pi} - a_p$. But, in order to keep $d_{pic} - d_{pid}$ greater than or equal to zero, we must also constrain d_{pi} such that $d_{pic} \geq 0$ and $d_{pid} \leq 0$. A non-zero value for d_{pi} is said to be an *active* bias whereas a zero value is said to be *inert*. The values of d_{pic} and d_{pid} are, hence, given by:

$$d_{pi_c} = \begin{cases} z_{pi_c} - a_p & \text{if } a_p < z_{pi_c} \quad \text{:active} \\ 0 & \text{otherwise} \quad \text{: inert} \end{cases} \tag{3.15}$$

$$d_{pi_d} = \begin{cases} z_{pi_d} - a_p & \text{if } a_p > z_{pi_d} \quad \text{:active} \\ 0 & \text{otherwise} \quad \text{: inert} \end{cases} \tag{3.16}$$

According to (3.15) and (3.16), $d_p(i)$ is active if the sign of $z_p(i)$ is correct but has magnitude larger than a_p.

3.4 Network Validation

The PLNC is trained using the feature vectors in the training file. Had there been access to an infinite number of training samples, it would be possible to use the trained network for classifying any feature vector, of similar nature of course. But unfortunately, in real examples, we only have access to a finite set of examples, usually smaller than desired. It is possible to use the entire training data to train the classifier and then use it on new test data. This approach, however, has a fundamental problem. The final trained classifier will overfit [4] the training data and have poor generalization capability when used on new test data.

A much better approach is to split the training data into disjoint subsets or files, one used for training, or the training file, and the other for validation, or the validation

file. An important reason for using the validation data is to test the generalization capability of the trained network and determine an optimum network size and a stopping point for the iterations involved in calculation of weights.

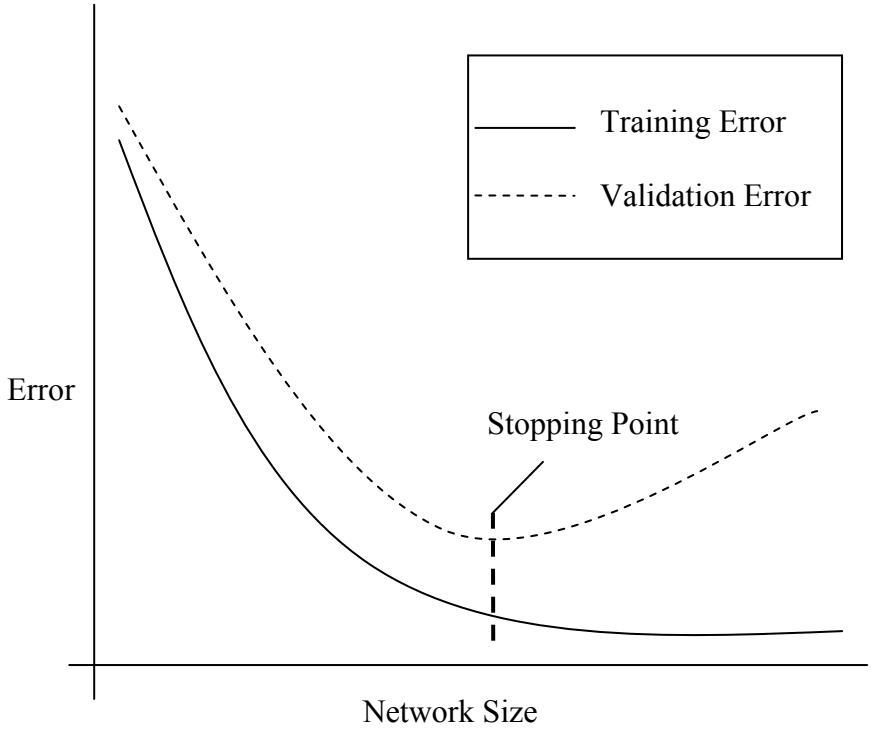

Fig 3.2 Training and Validation Errors

3.5 Network Pruning

Setting the ideal network size of a PLNC's topology is usually a major problem. Both, large and small networks have their own advantages and disadvantages, making it difficult to select an optimum size. Different applications would further require different sized networks to yield best results. Large networks may fit a given data set, but often

result in poor generalization. Too many clusters and/or weights in a network tend to overfit the data set. Large networks are also more computationally intense and more difficult to analyze. Smaller networks, on the other hand, can fail to learn a data set. It is often unclear on how to choose an optimal trade-off between a large and a small network.

Although the training algorithm, mentioned in section 3.2, designs the PLNC by adding the desired number of clusters, it does not always lead to the most compact network with the most optimal cluster mean vectors. The validation error generally tends to decrease with increasing size of the network, but only to a certain extent. After this, the validation error starts increasing even though the training error still continues to decrease. This is the portion where the network overfits the training data set and yields poor validation error. This is shown in fig. 3.2. Pruning of those clusters whose elimination causes the least increase in classification error leads to a more compact PLNC. Optimal removal of a cluster from a network with K clusters would involve calculating the possible increase in error from all possible eliminations – one cluster at a time – and selecting the cluster that causes the least increase in error.

There are a variety of methods available for pruning nodes and connections in neural networks, ranging from computationally extensive ones to much simpler ones. Examples are the Optimal Brain Damage [33], Optimal Brain Surgeon [34] and Skeletonization [35] techniques. A simple approach is described in this thesis; one that identifies and eliminates the least useful cluster in a completely trained network without significantly degrading the classifier's performance. This approach is derived by

replacing MSEs in the method of [13] with classification errors. The algorithm consists of the following steps:

1. Let k be the index of the cluster to be potentially eliminated and E_k the error of the network after cluster k has been pruned. Set $E = 0$ and $E_k = 0$, $1 \leq k \leq K$.

2. For $p = 1$ to N_v

   a) Identify two nearest clusters to input vector $\mathbf{x}_p$ with indices i, j and compute the two classification errors e_1 and e_2 if $\mathbf{x}_p$ were assigned to clusters i and j respectively.

   b) For $k = 1$ to K, accumulate errors as

$$\begin{aligned} E_k &\leftarrow E_k + e_1 \quad k \neq i \\ E_k &\leftarrow E_k + e_2 \quad k = i \end{aligned} \tag{3.17}$$

3. Find the smallest E_k as $E_{k_{\min}}$ and eliminate cluster k_{min}. The number of clusters K, now decreases by one, i.e.,

$$K \leftarrow K - 1 \tag{3.18}$$

4. Pass the data through the pruned network, redesign all the clusters and calculate the training error E. Save the network.

5. Repeat the above steps if further pruning is required.

Pruning results are discussed in chapter 6.

CHAPTER 4

NEURAL NETS AND THE BAYES OPTIMAL DISCRIMINANT

Previous research into the area of pattern recognition has shown that multi-layer perceptron classifiers and conventional nonparametric Bayesian classifiers yield the same classification accuracy, statistically speaking [43-45]. These results were empirical and, hence, were dependent on the data sets used. However, the consistently similar performances led to investigations into the theoretical connections and it has been proved [12] that MLPs approximate the Bayes optimal discriminant function when used for classification. In this chapter we carry the investigation further and explore the relationship between the PLNC and Bayes discriminant.

4.1 Neural Networks and Bayes Discriminant

Over the past several years, neural networks have been used for many tasks, including classification and function approximation. There have been many useful theoretical results concerning their capabilities. One of these theorems [12] follows.

Theorem 1: When neural net classifiers are trained to minimize the mean-squared error (MSE), the MSE approaches a constant value plus the expected squared error between the neural net output and Bayes discriminant, as the number of training patterns approaches infinity. Specifically,

$$\lim_{N_v \to \infty} \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{N_c} \left[t_{pi} - y_i(\mathbf{x}_p) \right]^2 = \sum_{i=1}^{N_c} \mathrm{E}\left[\left(b_i(\mathbf{x}) - y_i(\mathbf{x}) \right)^2 \right] + a \tag{4.1}$$

where a is a constant, independent of p, t_{pi} is defined by (3.6), and $y_i(\mathbf{x}_p)$ is the i^{th} output of the network. The Bayes discriminant $b_i(\mathbf{x})$, is the probability that the i^{th} class is correct, given $\mathbf{x}$, which is written as $P(i|\mathbf{x})$. The above theorem, however, leaves room for some doubts:

1. It does not give any bounds on the neural network's probability of error. Neither does it state an inequality relating the neural network's probability of error to Bayes probability of error.
2. The mean-squared error in the theorem treats positive and negative errors the same. However, it is good if the correct class discriminant is larger than desired, but bad if it is smaller. This problem leads to sub-optimal networks. However, this is fixed in the PLNC using output reset [30-32].
3. The theorem makes no use of the neural network's structure. It applies equally well to any discriminant designed by minimizing the MSE. Neural networks with different structures could lead to different results.

4.2 Nearest Neighbor and the Piecewise Linear Classifiers

The nearest neighbor classifier (NNC) classifies objects based on the closest training pattern in the feature space. A test sample or feature vector in the input space is assigned to class i if its nearest training sample, usually determined via the Euclidean distance, belongs to that class. The feature space is divided into convex partitions or clusters based on the class labels of the various training patterns. Specifically, the distance measure, which is usually Euclidean, performs a Voronoi tessellation [46] of the N-dimensional input feature space.

Now we consider the relationship between a PLNC and an NNC. As K approaches infinity, the convex Voronoi cells in the feature space get smaller in volume and the optimal decision boundaries in each cluster become linear. Hence, each cluster can have its own linear discriminant and overall, a more complex decision boundary can be achieved. Therefore, for a given value of K, the PLNC should perform better than the NNC. We begin to address this idea in the following lemmas.

Lemma 1: If a PLNC and NNC have the same distance measure and identical cluster mean vectors, the PLNC has at least as good a performance as the NNC.

Proof: Since the PLNC's augmented input vector **x** includes the constant 1, the PLNC's output vector **y** can have a 1 for the same class as picked by the NNC. Other class outputs can be 0.

Suppose that the c^{th} NNC cluster belongs to class i_c. If we would like the PLNC's c^{th} cluster to always map patterns to class i_c, the elements of the N_c by $(N+1)$ matrix, $\mathbf{A}_c'$, would be chosen such that its elements are defined as

$$a_c'(m,n) = \begin{cases} 1 & m = i_c \text{ and } n = N+1 \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

There can be occurrences when a PLNC's $\mathbf{A}_c$ matrices could initially perform worse than the NNC for the same c^{th} cluster. In such cases we can, however, guarantee that the PLNC performance is at least as good as the NNC by doing the following. If the PLNC's performance is worse than that of the NNC for the c^{th} cluster and the weight matrix obtained for this cluster is $\mathbf{A}_c$, replace it with $\mathbf{A}_c'$ as defined in (4.2).

Lemma 2: If a PLNC and a NNC have the same distance measure and identical cluster mean vectors, then as K approaches infinity,

$$P_{eB} \leq P_{e(PLNC)} \leq P_{e(NNC)} \leq 2 \cdot P_{eB} \tag{4.3}$$

where $P_{e(PLNC)}$, $P_{e(NNC)}$ and P_{eB} respectively denote the PLNC, NNC and Bayes probabilities of error.

Proof: As K approaches infinity, we know that [11,12]

$$P_{eB} \leq P_{e(NNC)} \leq 2 \cdot P_{eB} \tag{4.4}$$

Using lemma 1 and (4.4) yields the result of (4.3).

4.3 Convergence of the PLNC Error Probability

We now wish to be more specific and evaluate the average probability of error for the PLNC, as the amount of training data increases. First, recall the following lemma describing the convergence of the k-Nearest Neighbor Classifier (k-NNC) probability of error to Bayes error, $P_{e(k\text{-}NNC)}$.

Lemma 3 [4]: As k and (N_v/k) approach infinity, the k-NNC can be viewed as an attempt to estimate the *a-posteriori* probabilities from the training samples. Under this condition, k-NNC hence becomes optimal and

$$\lim_{k, \frac{N_v}{k} \to \infty} P_{e(k\text{-}NNC)} = P_{eB} \tag{4.5}$$

Here k is the number of nearest training vectors or samples to the test sample **x**. As k increases, the upper bound of the probability of error gets closer to the lower bound – the Bayes rate. In the limit as k goes to infinity, the two bounds meet and the k-NNC becomes optimal. We want to use a large value of k to obtain a reliable estimate. On the

other hand, we want all of the k neighbors to be very near to the test sample, **x**. This forces us to choose a compromise k that is a small fraction of the total number of training samples, N_v. It is only in the limit as N_v goes to infinity that we can be assured of the nearly optimal behaviour of the k-NNC.

We now investigate the relationship between Bayes discriminant and the PLNC in more detail. We start by stating the following theorem.

Theorem 2: As K and $N_v(c)$ approach infinity, the output of a PLNC approximates the *a-posteriori* probability functions of the class labels, given the input vector. Under this condition, the PLNC hence becomes optimal and

$$\lim_{K, N_v(c) \to \infty} \mathrm{P}_{\mathrm{e(PLNC)}} = \mathrm{P}_{\mathrm{eB}} \tag{4.6}$$

Proof: We start by observing that the classification error for the c^{th} cluster of a PLNC is given by (3.8). Let us start by taking into consideration only the feature vectors that have been assigned class label i. In this case, the MSE would be given by

$$E_{c,i} = \frac{1}{N_v(c)} \sum_{q=1}^{N_v(c)} \left[t_{qi} - \sum_{n=1}^{N+1} a_{cin} x_{qn} \right]^2 \tag{4.7}$$

where x_{qn} is the n^{th} element of the vector $\mathbf{x}_{\mathrm{q}}$, and $1 \le q \le N_v(c)$. The partial derivative of the error with respect to the elements of weight matrix $\mathbf{A}_{\mathrm{c}}$ is given by

$$\frac{\partial E}{\partial a_{cim}} = \frac{-2}{N_v(c)} \sum_{q=1}^{N_v(c)} \left[t_{qi} - \sum_{n=1}^{N+1} a_{cin} x_{qn} \right] x_{qm} \tag{4.8}$$

Before going any further, we consider how letting K and $N_v(c)$ approach infinity effects the range and variability of the input feature vectors **x**. Assuming the feature space is a

bounded compact subset of R^N, each cluster falls within a cell of the Voronoi tessellation. As K and $N_v(c)$ simultaneously increase towards infinity, the maximum radius of each cluster decreases towards zero. The range of values the elements of a feature vector in a cluster can take, decreases so much that they become constant, i.e, variability within a cluster approaches zero. Within the c^{th} cluster, as **x** becomes independent of the pattern number q, we can replace x_{qn} with mean vector element m_n in (4.8). We also equate the gradient to zero, so as to minimize the MSE. Equation (4.8) reduces to

$$\frac{-2}{N_v(c)} \sum_{q=1}^{N_v(c)} \left[t_{qi} - \sum_{n=1}^{N+1} a_{cin} m_n \right] m_m = 0 \tag{4.9}$$

which becomes

$$\sum_{q=1}^{N_v(c)} \left[t_{qi} - \sum_{n=1}^{N+1} a_{cin} m_n \right] = 0 \tag{4.10}$$

Since the terms a_{cin} and m_n are constants, we can replace the sum over n by the single constant $a_{ci(N+1)}$ which yields

$$\sum_{q=1}^{N_v(c)} \left[t_{qi} - a_{ci(N+1)} \right] = 0 \tag{4.11}$$

This can be written as

$$\sum_{q=1}^{N_v(c)} t_{qi} = N_v(c) \cdot a_{ci(N+1)} \tag{4.12}$$

which yields

$$a_{ci(N+1)} = \frac{\sum_{q=1}^{N_v(c)} t_{qi}}{N_v(c)} \tag{4.13}$$

Equation (3.6) gives the values of the elements of the desired output vector **t**. Without loss of generality, though, we can represent t_{qi} as

$$t_{qi} = \delta\left(i_c(q) - i\right) \tag{4.14}$$

where δ is the Kronecker delta function. In turn, (4.13) becomes

$$a_{ci(N+1)} = \frac{\sum_{q=1}^{N_v(c)} \delta\left(i_c(q) - i\right)}{N_v(c)} \tag{4.15}$$

$\delta(i_c(q) - i)$ equals 1 if $i_c(q)$ equals i, so, the numerator summation equals the number of training vectors in cluster c that belong to class i. This number could be represented by $N_v(c,i)$, hence reducing (4.15) to

$$a_{ci(N+1)} = \frac{N_v(c,i)}{N_v(c)} \tag{4.16}$$

which converges to the *a-posteriori* probability P(i|**x**). Thus theorem 2 is proved.

For a given cluster c, $a_{ci(N+1)}$ is largest for the class i which has the most input vectors in the cluster. This is precisely how the k-NNC makes decisions. So, each PLNC cluster with $N_v(c)$ members emulates a k-NNC decision with $k = N_v(c)$.

CHAPTER 5

MATHEMATICAL COMPLEXITY

Different classification applications have different requirements; speed may be the top priority in a real-time application while accuracy could be of prime importance in another. In order to find the best suited classifier for a particular application, performances of different kinds of classifiers can be compared. The basis for comparison could range from the speed of the network to the number of hidden units to the number of multiplies. In this chapter, we compare the number of multiplies in a piecewise linear network classifier against two other classifiers – the MLP and the NNC.

5.1 Comparison of Number of Multiplies against the NNC

We consider a trained PLNC that is given the task of classifying an N-dimensional test vector, $\mathbf{x}_t$, into one of N_c classes. As mentioned in section 3.2, it first assigns the vector to one of K clusters, using distance measure, and then assigns the vector a class ID using a simple linear classifier. It takes N multiplies to calculate the distance measure, given by

$$d\left(\mathbf{x}_t, m_k\right) = \sum_{n=1}^{N} \left[x_{tn} - m_{kn}\right]^2 \tag{5.1}$$

from the vector $\mathbf{x}_t$ to a cluster center vector. Since a total of K clusters exist, the test vector's distance from each of these clusters has to be measured. This results in a total of $K \cdot N$ multiplies. Once the test vector is assigned to a particular cluster, the

corresponding $\mathbf{A}_c$ matrix is activated to calculate the output vector $\mathbf{y}$, according to the formula

$$\mathbf{y} = \mathbf{A}_c \cdot \mathbf{x} \tag{5.2}$$

This calculation, in turn, requires $(N+1) \cdot N_c$ multiplies. Hence, the total number of multiplies required to apply a PLNC to a test vector is given by:

$$M_{PLNC} = \left[K \cdot N\right] + \left[\left(N+1\right) \cdot N_c\right] \tag{5.3}$$

A trained NNC, on the other hand, would first have to calculate the distance of the test vector from K_{NNC} clusters. This process requires a total of $K_{NNC} \cdot N$ multiplies. After all the distance measures are calculated, the closest cluster is found and test vector $\mathbf{x}_t$ is assigned its class ID. Therefore, the total number of multiplies required to apply a NNC to a test vector is given by:

$$M_{NNC} = \left[K_{NNC} \cdot N\right] \tag{5.4}$$

In order to compare the two classifiers based on the number of multiplies, we equate (5.3) and (5.4) to obtain

$$K \cdot N + (N+1) \cdot N_c = K_{NNC} \cdot N \tag{5.5}$$

This leads us to the following equation

$$K_{NNC} = K + \frac{\left(N+1\right) \cdot N_c}{N} \tag{5.6}$$

which provides the number of clusters in a NNC that would have the same number of multiplies as that of a PLNC.

Performances of the classifiers, with respect to the number of multiplies, are compared in chapter 6.

5.2 Comparison of Number of Multiplies against the MLP

Equation (5.3) states the number of multiplies required to apply a fully trained PLNC to a test vector. We now consider a trained MLP that is assigned the same task. We consider an MLP with one hidden layer, containing N_h hidden units. The input to the MLP is an *(N+1)*-dimensional vector, **x**, which is created by augmenting the test vector, $\mathbf{x}_p$, with 1. The i^{th} input unit is connected to the j^{th} hidden unit via weight *w(j,i)*. The MLP first calculates net functions that form the input to the hidden units. Net Function at the j^{th} hidden unit would be

$$NET(j) = \sum_{i=1}^{N+1} \left[w(j,i) \cdot x_i \right] \tag{5.7}$$

where $1 \leq j \leq N_h$. Equation (5.7) seems to require *N+1* multiplies, but since the last multiplication is just an addition – the last element of vector **x** being a 1 – it effectively requires only *N* multiplies. Calculation of N_h such net functions would hence require $N_h \cdot N$ multiplications. Each hidden unit applies an activation function, usually a sigmoidal function, to its input net function to obtain the output, *O(j)*. Each sigmoidal activation can be considered equivalent to two multiplies, hence N_h hidden units would require $2.N_h$ multiplies. Finally, the output layer of the MLP calculates the output vector elements y_k as

$$y_k = \sum_{i=1}^{N+1} \left[w_{io}(k,i) \cdot x_i \right] + \sum_{j=1}^{Nh} \left[w_{ho}(k,j) \cdot O(j) \right] \tag{5.8}$$

where $w_{io}(k,i)$ is the weight connecting the i^{th} input unit to the k^{th} output unit, $w_{ho}(k,j)$ is the weight connecting the output of the j^{th} hidden unit, *O(j)*, to the k^{th} output unit and$1 \leq$

$k \leq N_c$. The calculation of the output vector, **y**, hence requires $N_c.(N+N_h)$ multiplies. Therefore, a trained MLP would require a total of

$$N_h \cdot N + 2 \cdot N_h + N_c \left(N + N_h \right) \tag{5.9}$$

multiplies. In order to compare the PLNC and the MLP based on the number of multiplies, we equate (5.3) and (5.9) to obtain

$$N \cdot N_h + 2 \cdot N_h + N_c \cdot \left(N + N_h \right) = K \cdot N + N_c \cdot \left(N + 1 \right) \tag{5.10}$$

The number of clusters, K, hence required for a trained PLNC to have an equal number of multiplies as that of an MLP is given by

$$K = \frac{N_h \cdot \left(N + 2 \right) + N_c \cdot \left(N_h - 1 \right)}{N} \tag{5.11}$$

Performances of the classifiers, with respect to the number of multiplies, are compared in chapter 6.

CHAPTER 6

PERFORMANCE COMPARISONS AND SIMULATION RESULTS

The primary objective of performing these experiments is to examine and evaluate the performance of PLNC in light of the lemmas in chapter 4. In this chapter, we compare the performance of a PLNC with that of an NNC and an MLP using different data sets. Each data set consists of a training data file and a validation data file. The former is used to train the network and the latter is used to validate the design, i.e., test the generalization capability of the network. Performances are compared with respect to two parameters as mentioned below.

1. In section 6.1, the classification error percentages are plotted as function of the number of clusters, for an NNC and a PLNC, or the number of hidden units, for an MLP.
2. In section 6.2, the classification error percentages are plotted as a function of the number of multiplies required to apply the network.

6.1 Training and Pruning Results

In this section, we compare the performance of a PLNC with that of the NNC and an MLP using different data sets. The network is first trained using the training data file; the weights of the trained network are stored; the weights are then used to apply the network to a validation data file. During training, once the network is grown to the desired maximum size (in terms of number of clusters or hidden units) it is then pruned

down and error percentages are calculated at the end of each iteration. Similarly, during validation, the trained network is pruned down and error percentages calculated at the end of each iteration. The user has the ability to choose the size of the network based on the results.

6.1.1 PLNC vs NNC

Three data sets are used for the comparison.

1. grng [36] – This is a geometric shape recognition data set for of four shapes - ellipse, triangle, quadrilateral, and pentagon. Each shape consists of a matrix of size 64 x 64 pixels. For each shape, training and test patterns were generated using different degrees of deformation. The deformations included rotation, scaling, translation, and oblique distortions. The feature set has 16 features. For this comparison, the NNC and PLNC, both use the same set of cluster center vectors for classification. The results of this comparison are shown in fig. 6.1. The poor performance of the NNC occurs because it is forced to use the same clusters as the PLNC, for this example only.

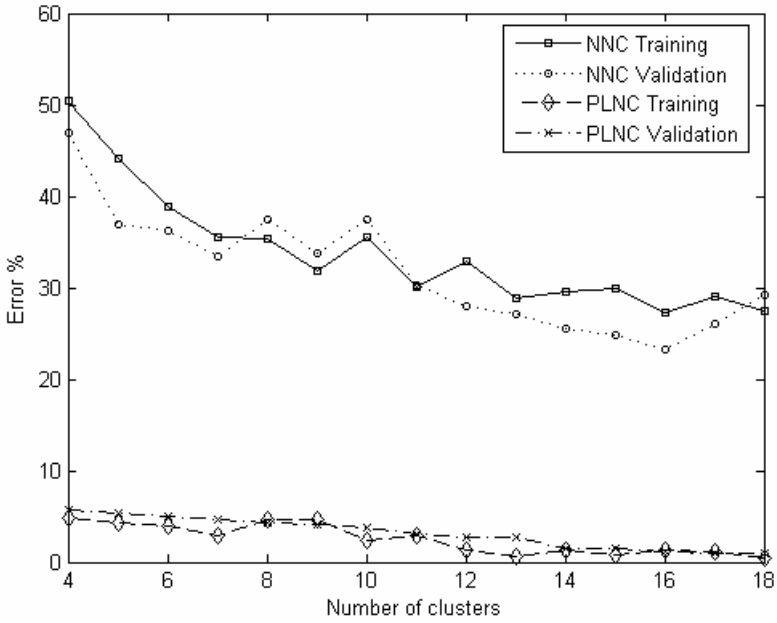

Fig. 6.1 Comparison of equivalent sized PLNC and NNC for shape recognition data

2. gong [37] – The raw data consists of images from hand printed numerals collected from 3,000 people by the Internal Revenue Service. Images are 32 by 24 binary matrices. An image scaling algorithm is used to remove size variation in characters. The feature set contains 16 elements. The 10 classes correspond to the 10 Arabic numerals. For this comparison, the PLNC and NNC are allowed to design and use their own sets of cluster center vectors. The results of this comparison are shown in fig. 6.2. The linear classifier in each PLNC cluster allows the PLNC to outperform the NNC.

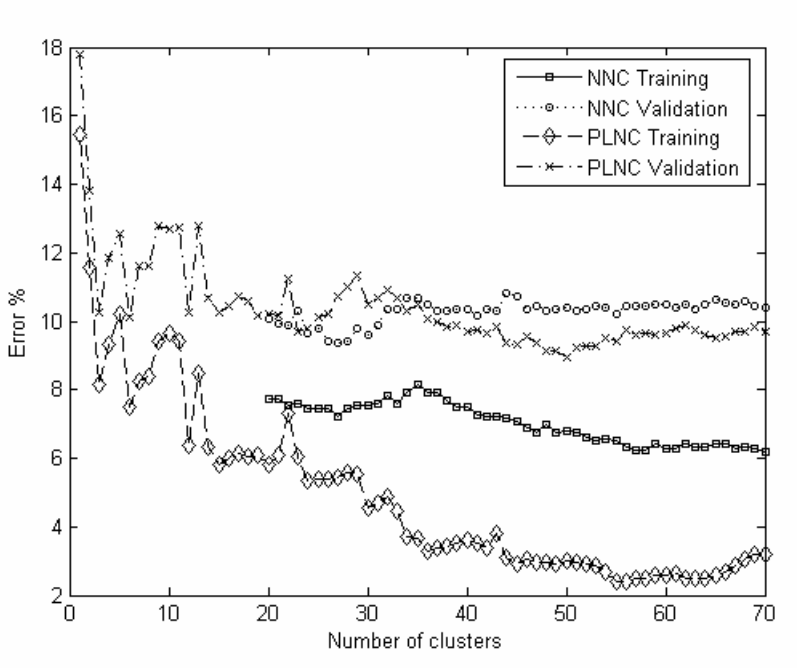

Fig. 6.2 Comparison of equivalent sized PLNC and NNC for numerical recognition problem

3. comf18 [38] – This data set consists of texture features corresponding to an image segmentation problem. Each segmented region is separately histogram equalized to 20 levels. Then the joint probability density of pairs of pixels separated by a given distance and a given direction is estimated. We use 0, 90, 180, 270 degrees for the directions and 1, 3, and 5 pixels for the separations. The density estimates are computed for each classification window. For each separation, the co-occurrences for the four directions are folded together to form a triangular matrix. From each of the resulting three matrices, six features are computed: angular second moment, contrast, entropy, correlation, and the sums of the main diagonal and the first off diagonal. This results in 18 features for

each classification window. For this comparison too, the PLNC and NNC are allowed to design and use their own set of cluster center vectors. The results of this comparison are shown in fig. 6.3. As in the previous examples, the PLNC outperforms the NNC.

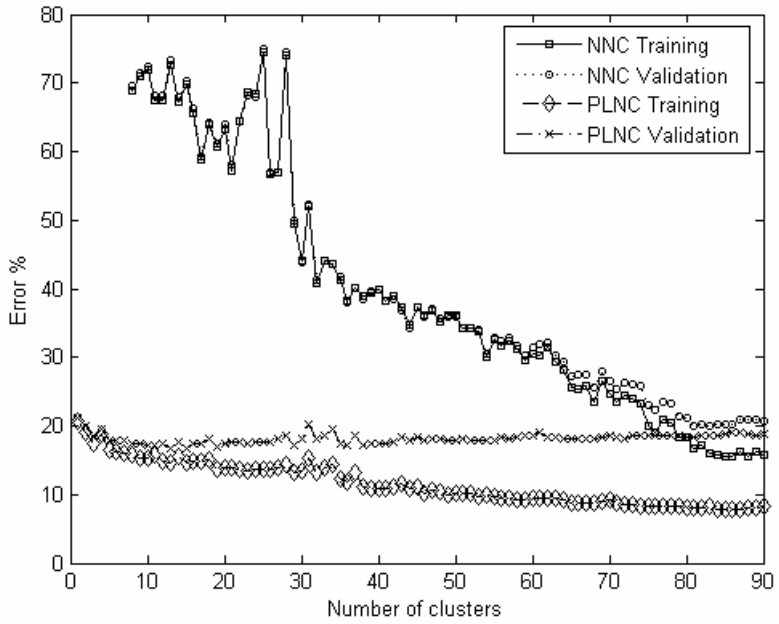

Fig. 6.3 Comparison of equivalent sized PLNC and NNC for segmentation problem

6.1.2 PLNC vs MLP

Two data sets are used for the comparison.

1. F17C – This data set consists of parameters that are available in the basic health usage monitoring system (HUMS). The data was obtained from the M430 flight load level survey conducted in Mirabel Canada in early 1995. Each input vector contains 17 elements. The 39 classes represent different maneuvers of the flight like taking off, landing, turning right or left etc. This is an application for

prognostics or flight condition recognition. The results of this comparison are shown in fig. 6.4. In this example, the PLNC outperforms the MLP.

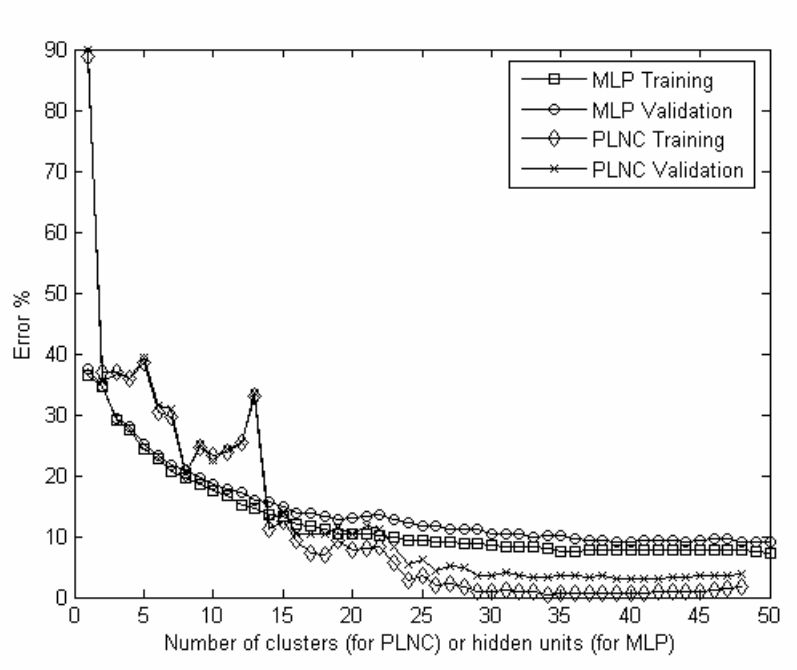

Fig. 6.4 Comparison of equivalent sized PLNC and MLP for prognostics data

2. mushroom - This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family. Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one. The Guide clearly states that there is no simple rule for determining the edibility of a mushroom; no rule like ``leaflets three, let it be'' for Poisonous Oak and Ivy. The results of this comparison are shown in fig. 6.5. The PLNC outperforms the MLP for larger networks.

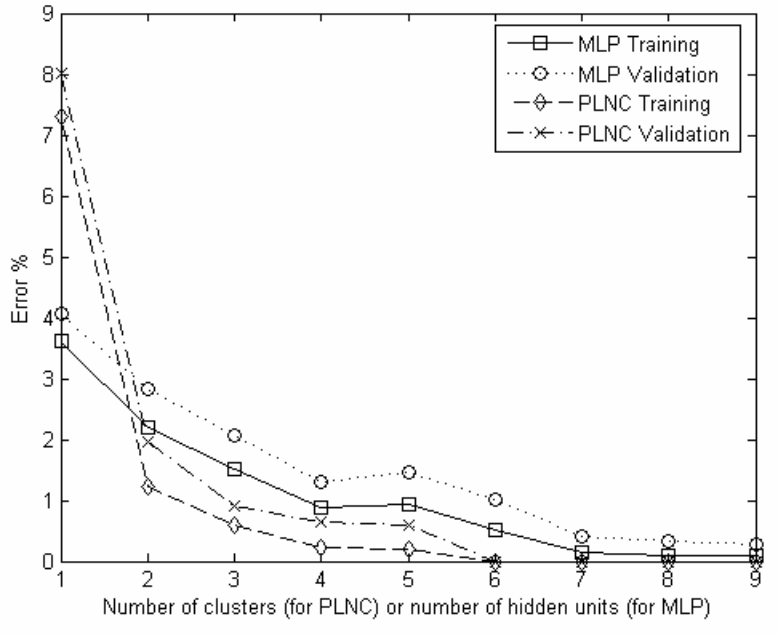

Fig. 6.5 Comparison of equivalent sized PLNC and MLP for mushroom edibility data

6.2 Mathematical Complexity Comparisons

Based on the work done in chapter 5, we compare the performance of a PLNC with that of the NNC and an MLP with respect to the number of multiplies required to apply each of these networks. Data points obtained from the training and pruning results, in section 6.1, are used to calculate the number of multiplies required to apply networks of various sizes. The classification errors are then plotted as a function of the number of multiplies.

6.2.1 PLNC vs NNC

Two data sets are used for the comparison – gong [37] and comf18 [38]. The results are shown in fig. 6.6 and fig. 6.7 respectively.

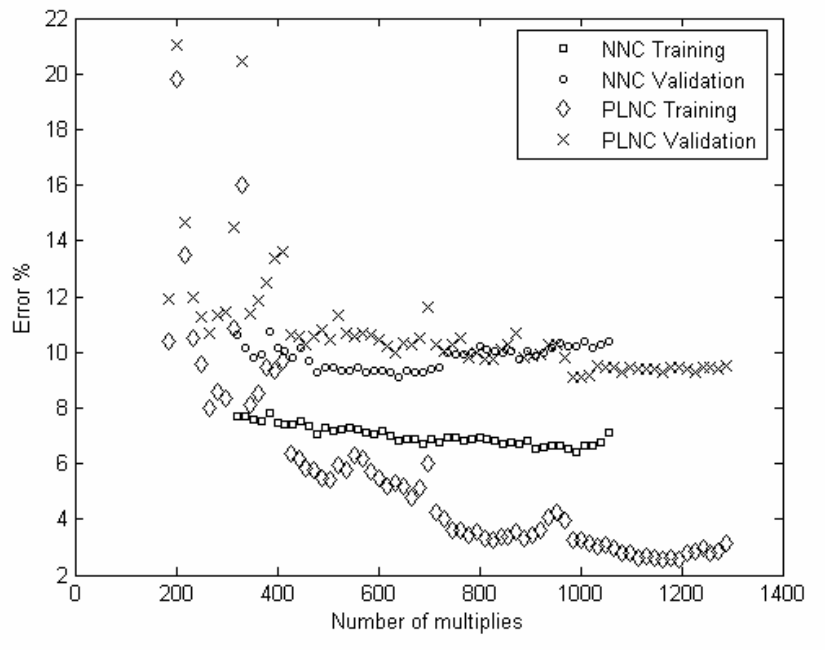

Fig. 6.6 Comparison of PLNC and NNC for numerical recognition problem based on the number of multiplies

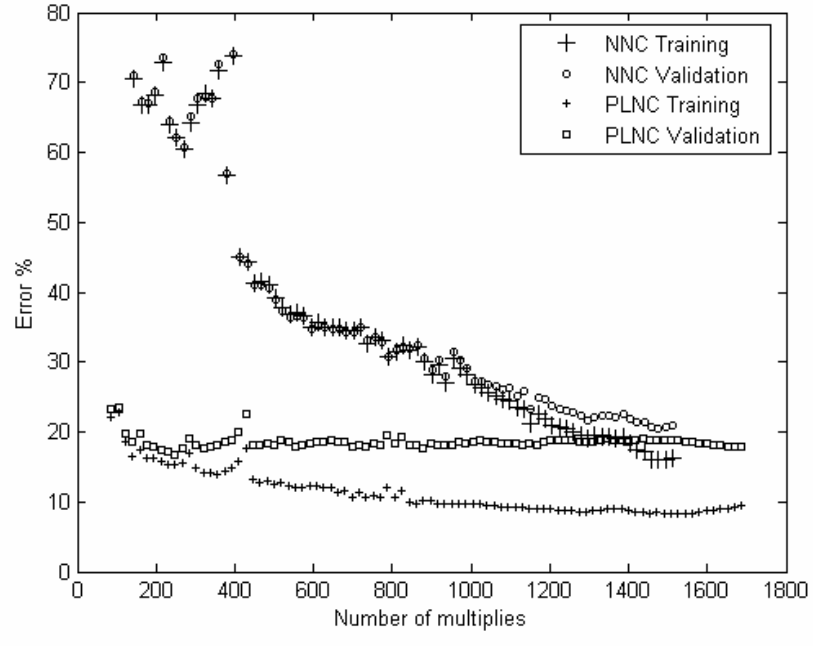

Fig. 6.7 Comparison of PLNC and NNC for segmentation problem based on the number of multiplies

6.2.2 PLNC vs MLP

Two data sets are used for the comparison – F17C and mushroom. The results are shown in fig. 6.8 and fig. 6.9 respectively.

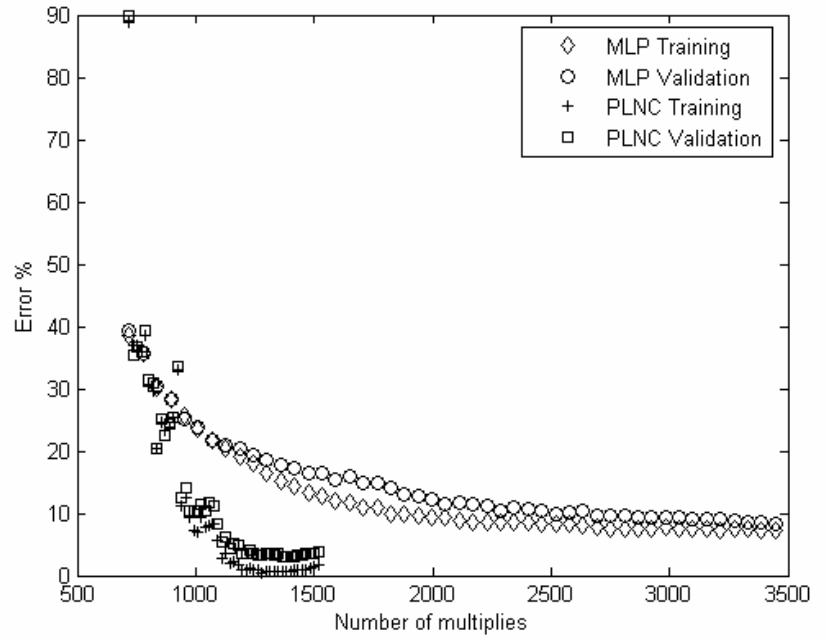

Fig. 6.8 Comparison of PLNC and MLP for prognostics data based on the number of multiplies

Figures 6.6 through 6.9 show that a PLNC requires fewer multiplications to obtain similar classification error percentage as compared to an MLP or an NNC. This is why the PLNC is much faster and far less mathematically complex than the classical neural net classifiers.

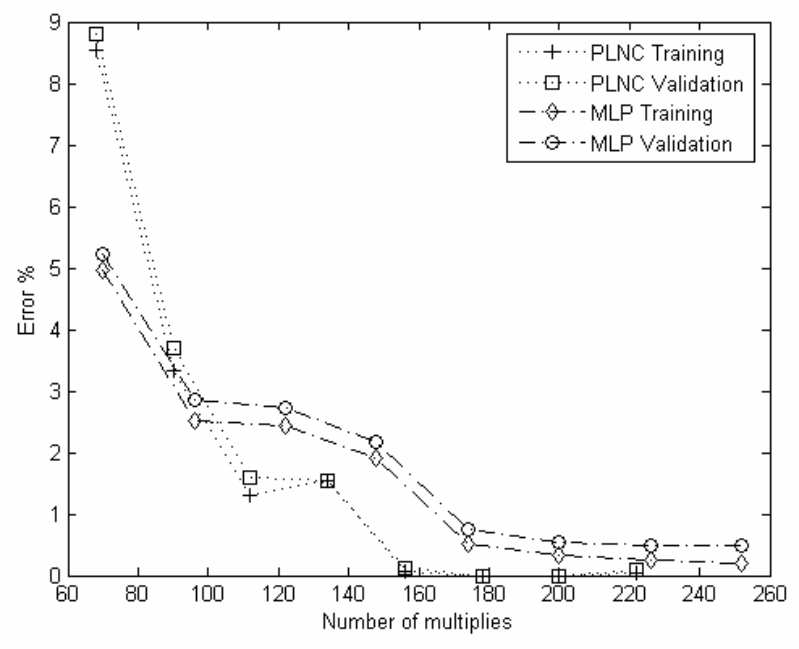

Fig. 6.9 Comparison of PLNC and MLP for mushroom edibility data based on the number of multiplies

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

In this thesis we first presented a brief overview of a typical classification problem, followed by a review of few of the existing classical and neural net classifiers along with their advantages and disadvantages. We then described a classifier based on a piecewise linear network and gave a design algorithm for it. The training algorithm has fast convergence because sets of linear equations are solved in each cluster. Due to the unavailability of theorems relating the mean squared error of existing neural networks and Bayes probability of error, we investigate the relastionship between the PLNC and Bayes probabilities of error. We started by proving that the performance of a PLNC is at least as good as an NNC, given the same distance measure and identical cluster mean vectors. Using existing theorems for the NNC and k-NNC, we then proved that the output of the PLNC approximates a Bayes optimal discriminant when trained to minimize the mean square error (MSE), leading to an optimal classifier. We have shown that the mathematical complexity of PLNCs is usually less than that other classifiers like MLPs and NNCs. It can, thus, be stated that the PLNC always trains faster than the MLP and often faster than the NNC, which usually requires a lot more clusters. Using several data sets, we have also shown that the PLNC often performs better than equivalent sized NNCs and MLPs.

7.2 Suggested Future Work

Future work will involve training the linear classifiers using alternative methods and comparing the results. Implementing multiple distance measures, instead of only one, can prove effective during the clustering stage and is worth exploring in the future. Better stopping criterion for the output-reset method should also be a topic for future research. Means and standard deviations of the PLNC classification error should be compared against those of classical and neural net classifiers, so that their performance could be better generalized.

APPENDIX A

REVIEW OF SELF ORGANIZING MAPS

Kohonen's SOM [10,18], in itself, is a type of unsupervised learning used to produce low dimensional representation of the training samples while preserving the topological properties of the input space. The goal of learning in SOM is to associate different feature vectors to respond similarly to certain input patterns or to group feature vectors with similar properties into clusters.

Clustering of the N_v input vectors, $\mathbf{x}_p$, $(1 \leq p \leq N_v)$ is performed using the SOM technique with the Euclidean distance measure $d(\cdot)$. At the end of each iteration, i.e., when feature vectors have been assigned to different clusters, we calculate the mean cluster vector $\mathbf{m}_c$ of every cluster. The training algorithm, in brief, does the following:

1. Let K be the total number of clusters and cluster mean vectors. Cluster mean vectors are first initialized as random vectors using parameters μ_n and σ_n as described in section 3.1. We denote a cluster mean vector by $\mathbf{m}_c$, where $1 \leq c \leq K$.
2. Training patterns are then fed to the network multiple times over a total of N_{it} iterations. During a single iteration it, where $1 \leq it \leq N_{it}$, a pass is made over the entire training data set of N_v vectors. When an input feature vector, $\mathbf{x}_p$, is fed to the SOM, its Euclidean distance to all the cluster mean vectors is computed. The cluster mean vector that is the closest to the input vector, say the k^{th} cluster in this case, along with its few neighboring clusters, is adjusted towards the input vector as

$$\mathbf{m}_c = \mathbf{m}_c + z(t)\left[\mathbf{x}_p - \mathbf{m}_c\right] \tag{1}$$

for

$$| k - c | \leq N(t) \tag{2}$$

Here *z(t)* is the gain function that decides the amount by which the vectors are adjusted, *k* and *c* are cluster mean vector indices and *N(t)* is the neighborhood function that decides the neighborhood of cluster mean vectors that is adjusted. The closest cluster mean vector, $\mathbf{m}_k$, is at the center of the neighborhood. In our experiments, we have used a decreasing exponential gain function given by

$$z(t) = a_1 . e^{-t/T_1} \tag{3}$$

and a decreasing neighborhood function given by

$$N(t) = a_2 . e^{-t/T_2} \tag{4}$$

where

$$t = p + (it \cdot N_v) \tag{5}$$

$$T_1 = \frac{N_v \cdot N_{it}}{3}, \tag{6}$$

$$T_2 = \frac{N_v \cdot N_{it}}{10}, \tag{7}$$

$$a_1 = \frac{K}{N_v}, \tag{8}$$

and

$$a_2 = \frac{K}{10} \tag{9}$$

APPENDIX B

REVIEW OF MODIFIED GRAM-SCHMIDT PROCEDURE

The Gram-Schmidt procedure [60] is a method for orthonormalizing a set of vectors in an inner product space, most commonly the Euclidean space, $\mathbf{R}^n$. The Gram-Schmidt process takes a finite, linearly independent set $S = \{v_1, ..., v_n\}$ and generates an orthonormal set $S' = \{u_1, ..., u_n\}$ that spans the same subspace as S.

In the context of a PLNC, basis functions for the network are the inputs and the constant 1. These are stored in a random vector **x**, which has a dimension $N_u=N+1$. Let $\mathbf{x}_m$ denote the m^{th} unit of this vector, where $1 \leq m \leq N_u$. The Schmidt procedure, for orthonormalization of basis functions can be described [39] as follows:

Given the basis functions x_m, form the first orthonormal basis function as

$$x_1' = \frac{x_1}{\|x_1\|} \tag{10}$$

The second orthonormal basis function is found as

$$c_1 = \left\langle x_1', x_2 \right\rangle \tag{11}$$

$$x_2' = \frac{x_2 - c_1 \cdot x_1'}{\left\| x_2 - c_1 \cdot x_1' \right\|} \tag{12}$$

The remaining orthonormal basis functions are calculated in a similar fashion. This process requires at least one pass through the training data for each new basis function, in order to calculate inner products of the form $<x_m, x_j'>$. However, it can be noted that each x_m' is a weighted sum of the x_j. It is therefore possible to reformulate the procedure so that all inner products $<x_m, x_j'>$ are calculated as weighted sums of $<x_m, x_k>$, meaning that only one pass through the training data is necessary.

The normal Gram-Schmidt procedure is refined to obtain a more useful form [28,29], wherein the orthonormal system is represented in terms of autocorrelation elements.

The m^{th} orthonormal basis function x_m', can be expressed as

$$x_m' = \sum_{k=1}^{m} g_{mk} \cdot x_k \tag{13}$$

From (3.30), for $m = 1$, the first basis function is obtained as

$$x_1' = \sum_{k=1}^{1} g_{1k} \cdot x_k = g_{11} \cdot x_1 \tag{14}$$

$$g_{11} = \frac{1}{\|x_1\|} = \frac{1}{r(1,1)^{1/2}} \tag{15}$$

where

$$r(i,j) = \langle x_i, x_j \rangle = \frac{1}{N_v} \sum_{p=1}^{N_v} x_{pi} \cdot x_{pj} \tag{16}$$

For values of m between 2 and N_u, c_i is first found for $1 \le i \le m\text{-}1$ as

$$c_i = \sum_{q=1}^{i} g_{iq} \cdot r(q,m) \tag{17}$$

Then m coefficients b_k are obtained as

$$\begin{aligned} b_k &= -\sum_{i=k}^{m-1} c_i \cdot g_{ik} \qquad 1 \le k \le m-1 \\ b_m &= 1 \end{aligned} \tag{18}$$

Finally for the m^{th} basis function the new a_{mk} coefficients (for $1 \le k \le m$) are found as

$$g_{mk} = \frac{b_k}{\left[r(m,m) - \sum_{i=1}^{m-1} c_i^{\,2} \right]^{1/2}} \tag{19}$$

Equation (3.4) can also be written as

$$y_i = \sum_{n=1}^{N+1} a_{cin} \cdot x_n \tag{20}$$

Equating y_i in (20) to

$$y_i = \sum_{q=1}^{N_u} a'_{ciq} \cdot x_q' \tag{21}$$

where the cluster weights in the orthonormal system are

$$a'_{ciq} = \sum_{k=1}^{q} g_{qk} \cdot \langle x_k, t_i \rangle = \sum_{k=1}^{q} g_{qk} \cdot c(i,k) \tag{22}$$

and using (13) we obtain cluster weights for the system as

$$a_{cin} = \sum_{q=n}^{N_u} a'_{ciq} \cdot g_{qn} \tag{23}$$

A few of the advantages of using the modified Gram-Schmidt procedure over other linear equation solving techniques are:

1. Since the modified Gram-Schmidt procedure is a noniterative technique, it is much faster [40] than other iterative techniques like the steepest descent and conjugate gradient.
2. This method has low storage requirements [40] since the cluster weights can be calculated recursively.

3. If the locations of the basis functions are distinct, then the set of equations (3.12) is linearly independent and has a unique solution [41]. This is a single globally optimum solution to (3.13).

REFERENCES

[1] C.M. van der Walt and E. Barnard,"Data characteristics that determine classifier performance," *Proceedings of the Sixteenth Annual Symposium of the Pattern Recognition Association of South Africa*, pp.160-165, 2006.

[2] Schalkoff, Robert, *Pattern recognition - statistical, structural and neural approaches*, John Wiley & Sons, 1992.

[3] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, 2nd ed., Academic Press, 1990.

[4] R.O. Duda, P.E. Hart, and D.G. Stork, *Pattern Classification*, 2nd ed., John Wiley & Sons, 2001.

[5] Thomas Bayes, "An Essay towards solving a Problem in the Doctrine of Chances", *Philosophical Transactions*, 1763

[6] W. H. Delashmit and M. T. Manry, "Recent Developments in Multilayer Perceptron Neural Networks", *Proceedings of the 7th annual Memphis Area Engineering and Science Conference (MAESC)*, 2005.

[7] T. Poggio and F. Girosi, "Networks for approximation and learning," *Proc. IEEE 78 (9)*, pp. 1484-1487, 1990.

[8] Nello Cristianini and John Shawe-Taylor, A*n Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*, Cambridge University Press, 2000.

[9] B. E. Boser, I. M. Guyon, and V. N. Vapnik, “A training algorithm for optimal margin classifiers”, *5th Annual ACM Workshop on COLT*, pp. 144-152, Pittsburgh, PA, ACM Press, 1992.

[10] Simon Haykin, *Neural networks - A comprehensive foundation*, 2nd ed., Prentice-Hall, 1999.

[11] Michael D. Richard and Richard P. Lippman, “Neural Network Classifiers estimate Bayesian a-posteriori probabilities,” *Neural Computation*, vol. 3, no. 4, pp. 461-483, 1991.

[12] Dennis W. Ruck, Steven K. Rogers, Matthew Kabrisky, Mark E. Oxley and Bruce W. Suter, “The Multilayer Perceptron as an approximation to a Bayes optimal discriminant function,” *IEEE Trans Neural Networks*, TNN-1(4):296-298, 1990.

[13] H. Chandrasekaran, J. Li, W.H. Delashmit, P.L. Narasimha, C. Yu and M.T. Manry, “Convergent Design of Piecewise Linear Neural Networks”, *NeuroComputing*, vol. 70, pp. 1022-1039, 2007.

[14] Jan A. Snyman, *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms*. Springer Publishing, 2005.

[15] J. Platt, “Fast Training of Support Vector Machines using Sequential Minimal Optimization,” *Advances in Kernel Methods - Support Vector Learning*, MIT Press, 1998.

[16] J. Platt, “Using Sparseness and Analytic QP to Speed Training of Support Vector Machines,” *Advances in Neural Information Processing Systems 11*, MIT Press, 1999.

[17] M. T. Manry, S. J. Apollo, L. S. Allen, W. D. Lyle, W. Gong, M.S. Dawson, and A. K. Fung," Fast Training of Neural Networks for Remote Sensing," *Remote Sensing Reviews*, vol. 9, pp. 77-96, 1994.

[18] T. Kohonen, *Self-Organization and Associative Memory*, 2nd ed., Springer-Verlag, 1987.

[19] K. Hornik, M. Stinchcombe, and H. White, “Multilayer Feedforward Networks Are Universal Approximators,” *Neural Networks*, Vol. 2, No. 5, pp. 359-366, 1989.

[20] K. Hornik, M. Stinchcombe, and H. White, “Universal Approximation of an Unknown Mapping and its Derivatives Using Multilayer Feedforward Networks,” *Neural Networks,* vol. 3, pp. 551-560, 1990.

[21] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*, Wadsworth, Belmont, CA, 1984.

[22] D. B. Fogel, “An information criterion for optimal neural network selection,” *IEEE Trans. Neural Networks*, vol. 2, no. 5, pp. 490-497, Sept.1991.

[23] J. H. Friedman, “Multivariate adaptive regression splines,” *Annals of Statistics*, vol. 19, no. 1, pp. 1-141, 1991.

[24] S. Subbarayan, K. Kim, M. T. Manry, V. Devarajan and H. Chen, “Modular neural network architecture using piecewise linear mapping,” *30th Asilomar Conference on Signals, Systems & Computers*, vol. 2, pp. 1171-1175, Nov. 1996.

[25] W. Li, J.-N. Lin, and R. Unbehauen, “Canonical representation of piecewise polynomial functions with nondegenerate linear domain partitions,” *IEEE Trans. Circuits and Systems I: Fundamental Theory and Applications*, vol. 45, no. 8, pp. 838-848, Aug. 1998.

[26] D.R. Hush and B. Horne, “Efficient algorithms for function approximation with piecewise linear sigmoidal networks,” *IEEE Trans. Neural Networks*, Vol. 9, No. 6, pp. 1129-1141, 1998.

[27] E.F. Gad, A.F. Atiya, S. Shaheen, A. El-Dessouki, “A new algorithm for learning in piecewise-linear neural networks,” *Neural Networks 13*, pp. 485–505, 2000.

[28] F. J. Maldonado, M. T. Manry, Tae-Hoon Kim, "Finding optimal neural network basis function subsets using the Schmidt procedure", *Proceedings of the International Joint Conference on Neural Networks*, vol. 1, pp. 444 – 449, July 2003.

[29] F. J. Maldonado, M. T. Manry, "Optimal Pruning of Feed-forward Neural Networks Based upon the Schmidt Pro*cedure," The 36th Asilomar Conference on Signals, Systems, & Computers*, pp. 1024 – 1028, 2002.

[30] L-M Liu, M.T. Manry, F. Amar, M.S. Dawson, and A.K. Fung, "Iterative Improvement of Image Classifiers Using Relaxation," *Conference Record of the Twenty Eighth Annual Asilomar Conference on Signals, Systems, and Computers, vol. 2*, 10/31/94 to 11/2/94, pp. 902-906.

[31] Jiang Li, Michael T. Manry, Li-Min Liu, Changhua Yu, and John Wei, “Iterative Improvement of Neural Classifiers”, *Proceedings of the Seventeenth International Conference of the Florida AI Research Society*, pp. 700-705, May 2004.

[32] R.G. Gore, Jiang Li, Michael T. Manry, Li-Min Liu, Changhua Yu, and John Wei, "Iterative Design of Neural Network Classifiers through Regression", *International Journal on Artificial Intelligence Tools*, vol. 14, nos. 1&2 pp. 281-301, 2005.

[33] LeCun, J. S.Denker, and S. A.Solla, "Optimal brain damage," *Advances in Neural Information Processing Systems 2*, pp. 598-605, 1990.

[34] Hassibi, B., Stork, O.G., and WOLFF, G.J., "Optimal brain surgeon and general network pruning," *Proceedings of International Conference on Neural Networks*, San Francisco, CA (Los Alamitos, CA: IEEE), pp. 293-299, 1993.

[35] C.Mozer and P.Smolensky, "Skeletonization: A technique for trimming the fat from a network via relevance assessment," *Connection Sci.*, vol. 1, no. 1, pp. 3-26, 1989.

[36] H. C. Yau, M. T. Manry, "Iterative Improvement of a Nearest Neighbor Classifier," *Neural Networks*, vol. 4, pp. 517-524, 1991.

[37] W. Gong, H. C. Yau, and M. T. Manry, "Non-Gaussian Feature Analyses Using a Neural Network," *Progress in Neural Networks*, vol. 2, pp. 253-269, 1994.

[38] R.R. Bailey, E. J. Pettit, R. T. Borochoff, M. T. Manry, and X. Jiang, "Automatic Recognition of USGS Land Use/Cover Categories Using Statistical and Neural Network Classifiers," *Proceedings of SPIE OE/Aerospace and Remote Sensing*, Orlando Florida, April 12-16, 1993.

[39] J.W. Dettman, *Mathematical Methods in Physics and Engineering*, 2nd ed., McGraw-Hill, 1962.

[40] Wladyslaw Kaminski, Pawel Strumillo, "Kernel Orthonormalization in Radial Basis Function Neural Networks," IEEE Transactions on Neural Networks, vol. 8, no. 5, pp. 1177-1183, 1997.

[41] M. Bianchini, P. Frasconi, and M. Gori, "Learning without local minima in radial basis function networks," *IEEE Trans. Neural Networks*, vol. 6, pp. 749–755, May 1995.

[42] Kay, Steven M., *Fundamentals of Statistical Signal Processing: Estimation Theory*, Prentice Hall, ch. 7, 1993.

[43] D. W. Ruck, S. K. Rogers, and M. Kabrisky, "Target recognition: Conventional and neural network approaches," *Proc. EEEE/INNS Int. Joint Conf. Neural Networks*, Abstract, 1989.

[44] S. K. Rogers, D. W. Ruck, M. Kabrisky, and G. L. Tarr, "Artificial neural networks for automatic target recognition," *Proc. SPIE Conf. Appl. Artif. Neural Networks*, Bellingham, WA, 1990.

[45] D. W. Ruck, S. K. Rogers, and M. Kabrisky, "Tactical target recognition: Conventional and neural network approaches," *Proc. 5th Ann. Aerospace Appl. Artif. Intell. Conf.*, Oct. 1989.

[46] Mohammed Kolahdouzan and Cyrus Shahabi, "Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases," *Proceedings of the 30th Very Large Data Bases (VLDB) Conference*, Toronto, Canada, 2004.

[47] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning representations by back-propagating errors*, Nature, pp. 533-536, 1986.

[48] Y.Hirose, K.Yamashita, and S.Hijiya, “Back-propagation algorithm which varies the number of hidden units,” *Neural Networks*, vol. 4, no. 1, pp. 61-66, 1991.

[49] R. Hecht-Nielsen, *Theory of the backpropagation neural networks, Proceedings of the international joint conference on neural networks*, Washington DC, vol. 1. IEEE Press, New York, pp. 593–605, 1989.

[50] L. Fausett, *Fundamentals of Neural Networks : architectures, algorithms,and applications*, Prentice-Hall, 1994.

[51] L. Prechelt, Automatic Early Stopping Using Cross Validation: Quantifying the criteria, *Neural Networks 11*, pp. 761-767, 1998.

[52] B. E. Boser, I. M. Guyon, and V. Vapnik, “A training algorithm for optimal margin classifiers,” *Fifth Annual Workshop on Computational Learning Theory*, Pittsburgh, 1992.

[53] Kendall A. Atkinson, *An Introduction to Numerical Analysis*, 2nd ed., John Wiley & Sons, New York, 1989.

[54] Gene H. Golub and Charles F. Van Loan, *Matrix computations*, 3rd ed., section 4.2, Johns Hopkins University Press, 1996.

[55] J. More, “The Levenberg-Marquardt Algorithm, Implementation, and Theory,” *Conference on Numerical Analysis*, 1977.

[56] R. P. Lippman, "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine,* April 1987.

[57] J. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observations," *Proc. 5th Berkeley Symp. on Mathematical Statistics and Probability*, pp. 281, 1967.

[58] Y. Linde, A. Buzo and R. Gray, "An Algorithm for Vector Quantizer Design," *IEEE Transactions on Communications*, Jan. 1980.

[59] S. Lloyd, "Least-squares quantization in PCM", *IEEE Transactions on Information Theory,* IT-28(2), pp. 129, 1982.

[60] Å. Björck, "Solving linear least squares problems by Gram-Schmidt orthogonalization," *BIT Numerical Mathematics*, vol. 7, no.1, pp. 1-21, March 1967.

BIOGRAPHICAL INFORMATION

Abdul Aziz Abdurrab was born in Hyderabad, India in 1980. He received the Bachelor of Technology in Electronics and Communication Engineering from Jawaharlal Nehru Technological University in 2002 and the Master of Science in Electrical Engineering from the University of Texas at Arlington (UTA) in 2007. While studying at UTA, he did an internship at Qualcomm, Inc., San Diego, CA, where he was responsible for testing firmware updates on real-time platforms and tuning an advanced acoustic echo-canceller for CDMA chipsets.

His research interests include multimedia signal processing, neural network algorithms and embedded microcontroller systems.