AN EFFICIENT PIECEWISE LINEAR NETWORK

by

ROHIT RAWAT

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

December 2009

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr Michael T. Manry, for supervising me on this thesis and helping me in every step of my work.

I wish to thank Dr Babak Fahimi and Dr William E. Dillon for taking time to serve on my thesis committee.

I would also like to thank the University Library for providing excellent templates to work with.

November 25, 2009

ABSTRACT

AN EFFICIENT PIECEWISE LINEAR NETWORK

Rohit Rawat, M.S.

The University of Texas at Arlington, 2009

Supervising Professor: Michael T. Manry

A Piecewise Linear Network (PLN) is a local network that offers the accuracy of higher order networks and the Multi Layer Perceptron (MLP), with the computational simplicity of linear networks. A method to design a PLN is demonstrated and several clustering algorithms, used in the design procedure, are compared. The performance of the Self Organizing Map (SOM) clustering algorithm has been found to be slightly better than the other clustering methods. Methods to determine the appropriate threshold in the Sequential Leader algorithm have been studied. A binary search based approach was found to be the most efficient in terms of the number of trials needed. Methods to delete extra clusters generated have been studied and compared to pruning. Pruning yields the best networks followed by deleting the smallest clusters. Methods of improving PLN

pruning performance have been developed, including segregation of patterns by clusters, the use of partial distances, and redesign of only changed clusters. Results have been presented for several different data files.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Neural Networks

Neural networks consist of highly interconnected processing elements working to solve specific problems. A neural network processing element (a neuron) consists of a number of inputs that are multiplied by gains, a threshold that is added to it, and an activation function that performs a transformation on the result.

Neural networks have an amazing ability to learn. Given an arbitrary data set, a neural network can approximate a mapping from the inputs to the outputs.

Neural networks are used in approximation problems [1] such as stock market forecasting [2], aviation prognostics [3], data mining [4, 5], filtering [6], and control applications [7]. They are also used in classification problems such as speech recognition [8], character recognition [9], fingerprint recognition [10], and face detection [11].

1.1.1 Benefits of Neural Networks

Neural networks have several benefits such as [1]:

1. Nonlinearity: An artificial neuron is nonlinear because of its activation function. A neural network made up of such elements is also nonlinear. This property is essential, especially when modeling nonlinear phenomenon [7, 12].

2. Input-Output Mapping: In the supervised learning paradigm, the synaptic weights of a neural network are modified to minimize the error between the desired output and the actual outputs of the network. The network is trained with several training patterns till the change in weights becomes negligible. The network thus forms a mapping between the input data and the desired outputs. They are thus useful in regression analysis, such as time series prediction, fitness approximation and modeling [13].
3. Adaptivity: Neural networks have an ability to adapt their weights according to changes in operating conditions. In other words, a network can be easily retrained to deal with changes in the operating environment. This adds robustness to the system [14].
4. Evidential Response: Neural network based classifiers can be designed to provide information not only about the decision made, but also the confidence in the decision [15, 16]. This helps in eliminating ambiguous patterns.
5. Contextual Information: Every neuron in the network is potentially affected by the outputs of other neurons in the network. Thus neural networks can grasp contextual information from the data [17].
6. Fault Tolerance: Due to the highly distributed processing in a neural network, the loss of or damage to one neuron does not affect the performance of the whole network drastically. There is a graceful degradation in performance [18].
7. VLSI Implementability: The massively parallel and hierarchical structure of neural networks makes them suitable for hardware implementation using very-large-scale-integration (VLSI) technology [1, 19, 20].

1.1.2 Model of a Neuron

The neuron described in this document has a structure similar to the one shown in the following figure:

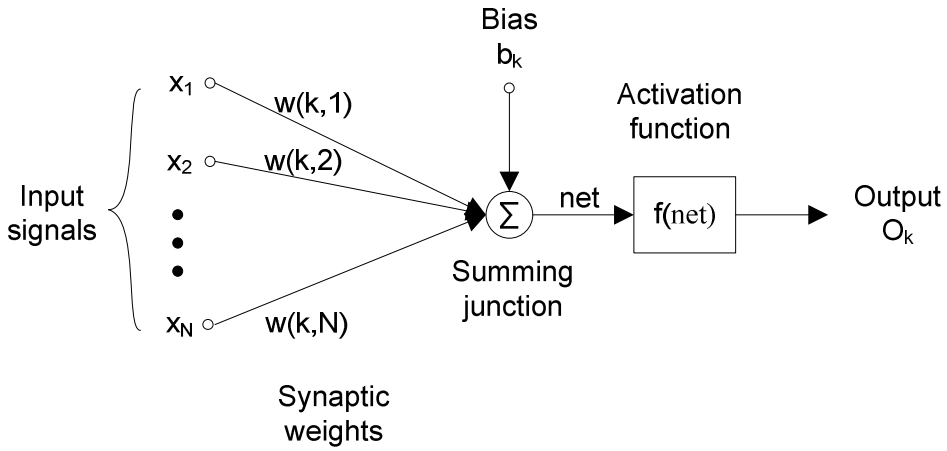

Figure 1.1 Nonlinear model of a neuron.

The components of a neuron are described below:

1. Synapses are a set of interconnecting links, each of which is characterized by a weight that is multiplied to the input to determine the value at its output.
2. An adder sums up the outputs of all the synapses and a fixed bias to produce a value known as the net function. The bias works to increase or decrease the net function.
3. An activation function is applied to the net function to limit its output. The activation function can be used to transform the output to a binary representation if desired, or to introduce nonlinearity in the outputs.

   Typically used activation functions [1] are:

   a. Thresholding Function
   b. Piecewise-Linear Function
   c. Sigmoidal Function

1.2 Types of Neural Networks

Neural networks can be classified on the basis of the structural arrangement of neurons and the way information travels between them. Each arrangement requires a different learning algorithm for training the network. Although there can be any number of arbitrary network configurations, a few prominent ones with well developed learning algorithms are mentioned here.

1.2.1 Single-Layer Feedforward Networks

Single layer feedforward networks are made up of a single layer of neurons. The inputs to the network are projected onto this layer, and the neuron outputs are the final outputs of the network. There is no connection between the output of one neuron and the input of another neuron, so there is strictly unidirectional flow of information.

1.2.2 Multi-Layer Feedforward Networks

Multi-layer feedforward networks have one or more additional hidden layers between the input layer of source nodes and the output layer. The extra layer adds processing power to the network in terms of an additional set of neuron interaction, and increases global connectivity.

Typically, the input layer is connected to the inputs of the first hidden layer. The outputs of the first hidden layer are connected to the inputs of the next layer and so on. There can also be bypass weights directly connecting the input layer to the output layer, or outputs of any layer to the inputs of a higher layer. But there are no feedback loops, i.e., weights connecting outputs of a higher layer to the inputs of a lower layer.

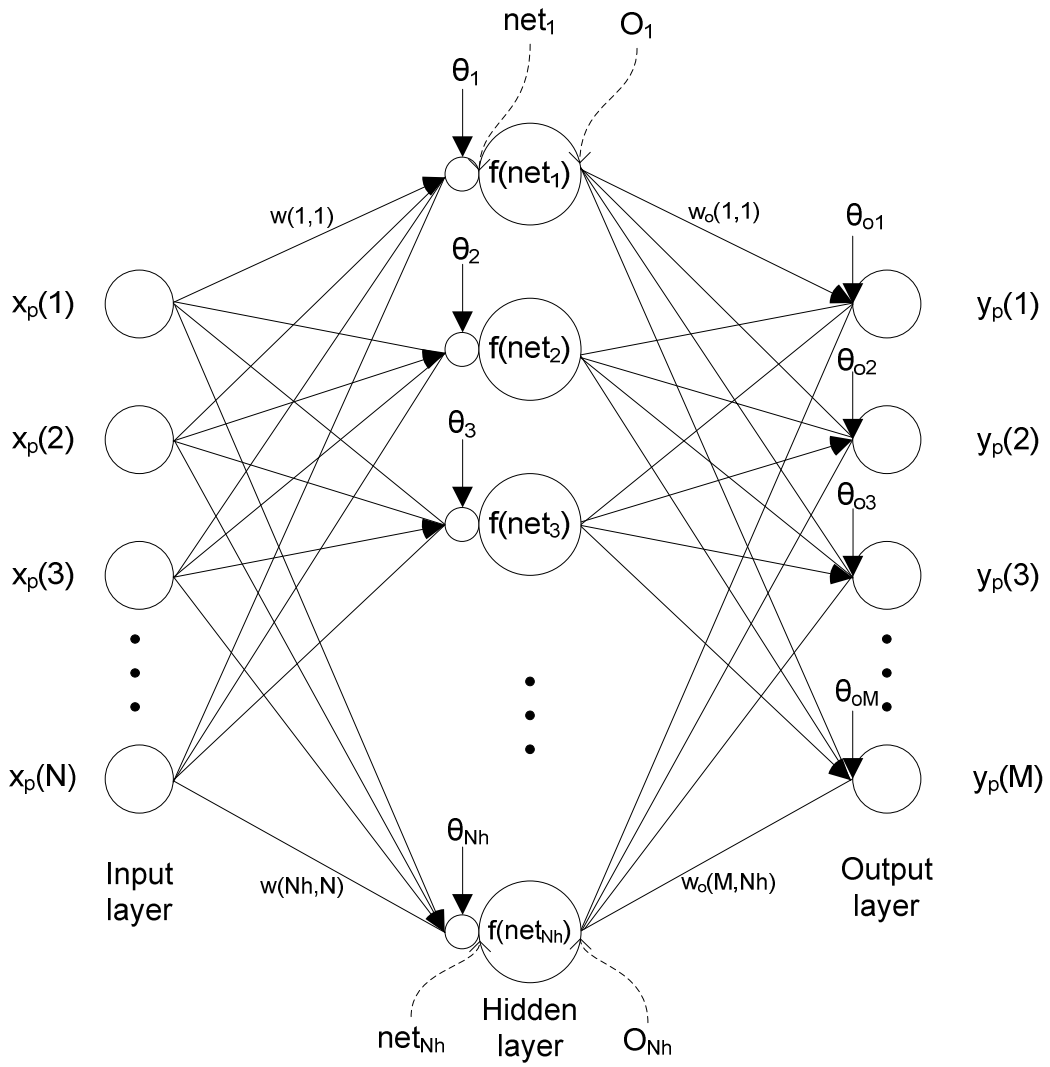

Figure 1.2 A multilayer perceptron with one hidden layer.

A neural network is said to be fully connected if every node in each layer of the network is connected to every node in the adjacent forward layer. If any of these connections are missing, the network is said to be partially connected [1].

1.2.3 Recurrent Networks

The human brain has a highly interconnected network of neurons through synapses comprising many recurrent connections. To model systems which are biologically more realistic, recurrent neural network architectures are considered more suitable [17, 21].

A recurrent network is different from a feedforward network in the sense that it has at least one feedback loop. The feedback links have a unit delay element (denoted by z^{-1}), which results in a nonlinear dynamical behavior.

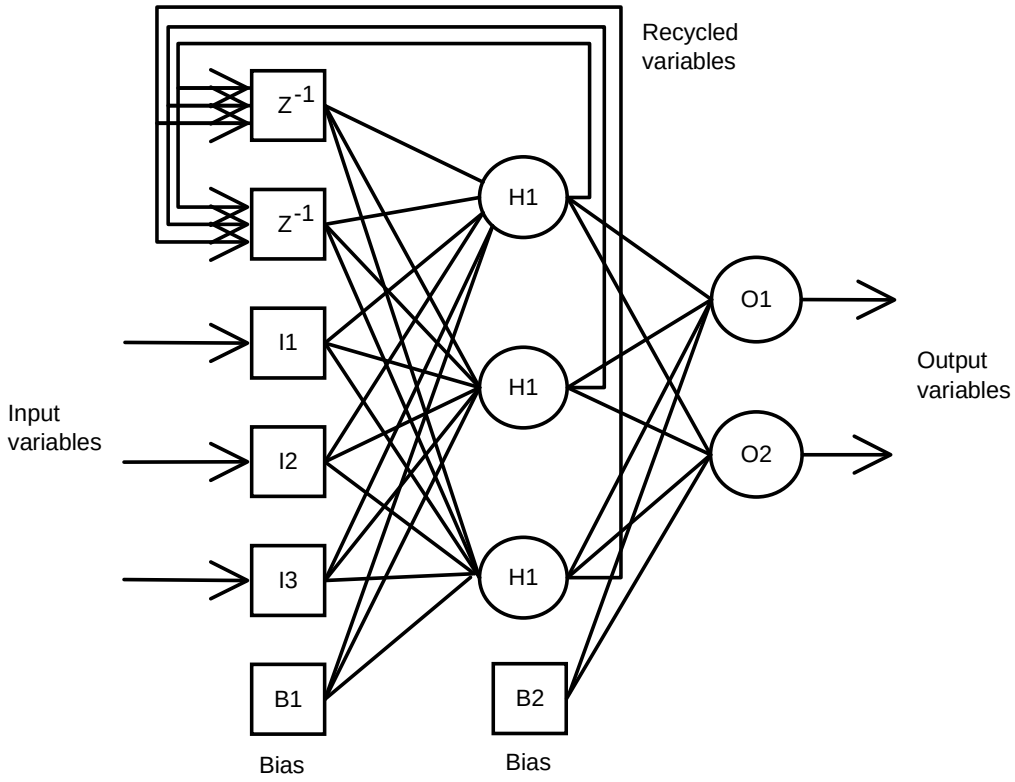

Figure 1.3 Example of a recurrent network.

In the recurrent network shown in Figure 1.3, there is a single layer of neurons with each neuron feeding its output signal back to the inputs of all the other neurons. If the output of a neuron is fed to one of its own inputs, it is said to have a self feedback loop [1].

1.2.4 Radial Basis Function Networks

A radial basis function (RBF) is a function whose value depends only on the distance from an input vector **x** to a center vector $\mathbf{m}_k$.

$$\phi(\mathbf{x}) = \phi(\|\mathbf{x} - \mathbf{m}_k\|)$$

The norm is usually the Euclidian distance, though other distance measures are also used.

Radial basis function networks have neurons with nonlinear RBF activations in the hidden layer. The input layer is made up of sensory neurons. When the hidden layer is larger than the input layer, the network applies a nonlinear transformation from the input space to a hidden space of higher dimensionality. The output layer is just a linear layer generating the output of the network. A pattern classification problem cast in a high dimensional space is more likely to be linearly separable than in low dimensional space [22]. This justifies the linear operation following the nonlinear operation. With a higher dimensional hidden space, the network is better able to approximate a smooth input-output mapping [23, 24].

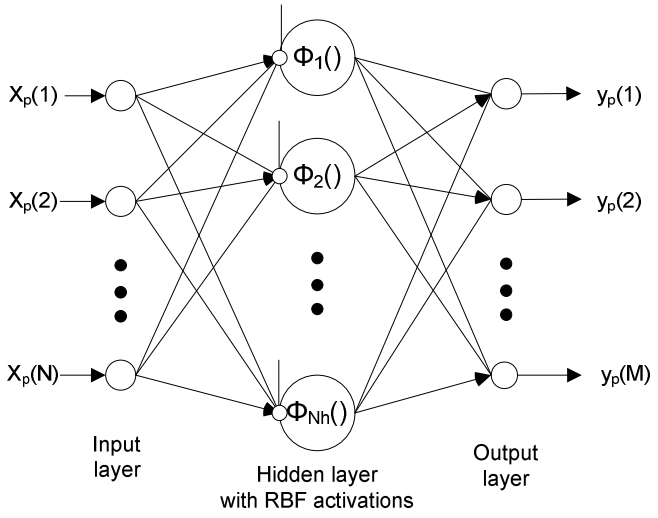

Figure 1.4 Structure of a radial basis function network.

1.2.5 Time-Delay Neural Networks

Time-delay neural networks (TDNNs) are used for processing temporal data such as recognizing phonemes in speech. Delay elements are used to preserve the previous output of a neuron. Several delay units can be aggregated to delay information by different periods. In the TDNN shown in Figure 1.5, the inputs to the hidden layer are the current set of feature inputs, along with several delayed versions of previous inputs and hidden

layer outputs, for multiple delay values. Time-delay neural networks are able to keep track of context, which plays an important role in speech perception [1, 8, 25].

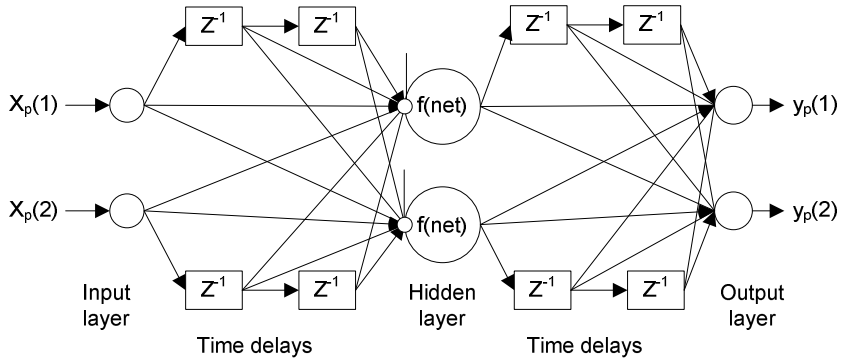

Figure 1.5 Time delay neural network.

Actual speech has various inflections and phonemes that have a variable duration. Thus, it becomes difficult for a TDDN with a fixed structure to accurately model such data. Traditionally, Hidden Markov Models (HMMs) have been used for temporal processing tasks such as speech and handwriting recognition. HMMs are stochastic processes whose underlying Markov chains have unobserved states. Various TDNN/HMM hybrid models have been found to perform temporal pattern recognition tasks well [1, 26].

1.3 Neural Networks as Approximators and Classifiers

1.3.1 Neural Networks as Approximators

While initially developed to mimic brain functions, neural networks can be thought of as another way of developing approximations. A function of many variables needs to be approximated given only values of the function (often perturbed by noise) at various points in the variable space. Typically, a feed-forward network with a single hidden layer, with enough hidden units to approximate the continuous function is selected. The optimum size and topology of the network has been studied in the literature [27]. Neural

networks are being increasingly preferred over polynomial approximations. Unlike polynomial approximations, neural networks can be trained for ill-posed patterns, but the mappings they yield for such cases may not be globally accurate [28].

1.3.2 Neural Networks as Classifiers

A classifier groups items with similar feature vectors into labeled groups. A Nearest Neighbor classifier can be designed by using clustering algorithms, and assigning classes to clusters or groups of clusters. Neural network classifiers are designed by thresholding or segmenting the output of a neural network into classes. Better results can be obtained by training for uncoded outputs instead of a single output [29, 30].

1.4 Piecewise Linear Networks

Piecewise linear networks use the divide and conquer approach in solving a problem. The input vector space is divided into several different clusters and a linear mapping is developed for the patterns in each cluster. When utilizing such a network, the incoming pattern is matched with the cluster closest to it, and the network corresponding to that cluster is used to compute the output of the complete network [31].

1.4.1 Piecewise Linear Multi-layer Perceptron

Traditional MLPs use a sigmoid or step activation function. A sigmoidal activation can be replaced with a piecewise linear activation function which can be continuous or discontinuous. This activation function can be trained using traditional back propagation algorithms [32, 33].

1.4.2 Simplicial Piecewise Linear Approximation

Given real numbers $b_1 < b_2 < \ldots < b_k$, let f be a function on $\Re$ that is linear on each interval $[b_i, b_{i+1}]$, i = 0, 1, … , k. Such a function is called piecewise linear and the numbers b_i are called breakpoints.

A continuous function can be modeled with a piecewise linear model, i.e. by defining it in pieces that are linear. By increasing the number of linear segments, the original function can be approximated very accurately [34].

1.4.3 Discontinuous PLN

In a discontinuous PLN, the training patterns are clustered by the inputs. A linear mapping for each cluster is independently trained. The overall network is a piecewise linear mapping, that may not be continuous [31, 35].

1.5 Properties of Discontinuous PLN

Discontinuous piecewise linear networks have several desirable properties:

1. Since PLN modules are trained by solving linear equations, we can take advantage of fast algorithms like conjugate gradient [36] and the Schmidt procedure [37] to speed up the process.
2. PLNs can approximate a discontinuous mapping. Even though clustering is done only on the inputs, the clusters formed usually have a good input-output mapping as well.
3. A PLN can be pruned down to the required size to suit the processing and accuracy needs of a particular application.
4. PLNs have been successfully used for feature selection [38]. Feature selection is important when there are initially too many features. The goal is then to find the

best small feature subset. Unnecessary features cause several problems such as: 1) increased training time, 2) the *Curse of Dimensionality* [39], and 3) convergence difficulties and 4) poor generalization during training [38].

1.6 PLN Problems

PLNs have several problems including the following:

1. It is difficult to estimate the right number of clusters to approximate a given function. Too many clusters can lead to memorization due to small cluster size, while too few clusters prevent the formation of a good mapping [40].
2. Clustering is not necessarily optimal. The major drawback of the K-means and SOM algorithms is that they often get stuck in a local minima and the result is largely dependent on the choice of initial cluster centers [41, 42].
3. There may be small clusters made up of outlier patterns that do not represent the rest of the data. The PLN may memorize such clusters to reduce the overall error, but will have lower performance during validation. If these clusters are merged with others, they may skew the clusters and introduce error in the mapping [43].
4. Error is not fed back during training. The clustering algorithms currently employed only cluster the inputs and do not consider the output mappings. This can result in the formation of bad clusters which have a large training error.
5. Pruning algorithms for the PLN require at least two complete passes through the data file to prune one cluster. This can significantly slow down pruning, given the slow access speeds of storage devices.
6. Pruning algorithms that prune one cluster at a time do not necessarily produce the optimal set of clusters when used repeatedly to prune more than one cluster.

1.7 Objectives of This Thesis

In chapter 2, we review the basic concepts useful in design of a PLN. In chapter 3, we review the PLN, and the basic PLN pruning algorithm. In chapter 4, we discuss various clustering techniques. In chapter 5, we propose a new threshold estimation technique for the sequential leader clustering algorithm, which yields a good threshold that can be used to quickly arrive at the desired number of clusters. In chapter 6, we compare the performance of various clustering methods in PLN design, and give an optimized pruning algorithm that significantly reduces file access and improves pruning speeds. In chapter 7, we present the results of our algorithm on several data files. In chapter 8, we present our conclusions and possible enhancements to this work.

CHAPTER 2

BACKGROUND FOR PIECEWISE LINEAR NETWORKS

2.1 Introduction

In this chapter we discuss the basics required for the study of PLNs including distance measures, the Schmidt procedure, and methods to validate networks.

2.2 Distance Measures

A distance measure d() is a function of two vectors and yields a value quantifying the distance between the two. Distance measures are used by clustering methods to determining proximity of patterns to each other, and by piecewise linear networks to select a network with which to process an incoming pattern.

The 1-norm distance, also called the Manhattan or city block distance between two real vectors **x** and **m** of dimension N is given by [44]:

$$d(\mathbf{x},\mathbf{m}) = \sum_{n=1}^{N} \left| x(n) - m(n) \right| \tag{2.1}$$

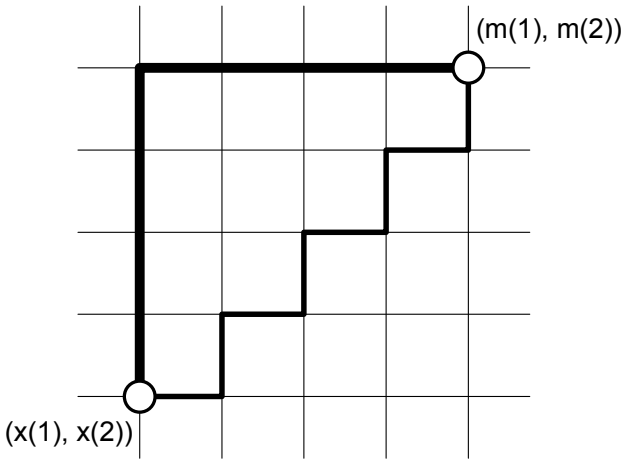

Figure 2.1 Manhattan distance.

The 2-norm distance, often called the Euclidian distance is given by [45]:

$$d(\mathbf{x},\mathbf{m}) = \sum_{n=1}^{N} \left(x(n) - m(n)\right)^2 \tag{2.2}$$

The square root is left out, since that has no effect on the decisions that follow. The weighted Euclidian distance is a modified version of the Euclidian distance which uses different weights in the sum for different dimensions of the vector, based on their importance [46]. It is defined as:

$$d(\mathbf{x},\mathbf{m}) = \sum_{n=1}^{N} \left(x(n) - m(n)\right)^2 \cdot w(n) \tag{2.3}$$

where w(n) is the weight given to the n^{th} dimension of the vectors.

The Mahalanobis distance takes into account the correlations of the data set and is scale invariant. When the features are not correlated, the Mahalanobis distance is equivalent to the squared Euclidian distance. The Mahalanobis distance is defined as:

$$d(\mathbf{x},\mathbf{m}) = \sum_{n=1}^{N}\sum_{m=1}^{N} a(n,m) \cdot \left[x(n) - m(n)\right]\left[x(m) - m(m)\right] \tag{2.4}$$

where $\mathbf{A} = \mathbf{C}^{-1}$, and $\mathbf{C}$ is the covariance matrix defined as $\mathbf{C} = E[\ (\mathbf{x} - \mathbf{m}) \cdot (\mathbf{x} - \mathbf{m})^T\ ]$ where $\mathbf{m} = E[\ \mathbf{x}\ ]$.

If the covariance matrix **C** is the identity matrix, then the Mahalanobis distance reduces to the Euclidian distance [47].

2.3 Schmidt Procedure for Training Linear Networks

Consider a linear system mapping of an (N + 1) dimensional augmented input vector $\mathbf{x}_{ap}$ to an M dimensional output vector $\mathbf{y}_p$. The system parameters are in the form of an M × (N + 1) weight matrix **W**, and $\mathbf{y}_p$ is calculated as:

$$\mathbf{y}_p = \mathbf{W} \cdot \mathbf{x}_{ap}$$

where $\mathbf{x}_{ap}$ is the augmented input vector given by:

$$\mathbf{x}_{ap} = (\mathbf{x}_p^T : 1)^T \tag{2.5}$$

Given training data consisting of inputs $\mathbf{x}_p$ of dimension N and corresponding outputs $\mathbf{t}_p$ of dimension M, to solve for **W** by regression, the following error is minimized:

$$E = \sum_{p=1}^{N_v} \left\| \mathbf{t}_p - \mathbf{W} \cdot \mathbf{x}_{ap} \right\|^2 \tag{2.6}$$

where $\mathbf{x}_{ap}$ is the augmented input vector obtained by equation (2.5). The auto correlation matrix **R** is defined as:

$$r(n,l) = \frac{1}{N_v} \sum_{p=1}^{N_v} x_{ap}(n) \cdot x_{ap}(l) \tag{2.7}$$

The cross-correlation matrix **C** is defined as:

$$c(n,i) = \frac{1}{N_v} \sum_{p=1}^{N_v} x_{ap}(n) \cdot t_p(i) \tag{2.8}$$

The weight matrix **W** can be solved using the following procedure. Writing equation (2.6) in terms of element of **W**:

$$E = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} \left[t_p(i) - \sum_{n=1}^{N+1} w(i,n) \cdot x_{ap}(n) \right]^2 \tag{2.9}$$

Differentiating it with respect to the elements of **W**:

$$\frac{\partial E}{\partial w(m,l)} = \frac{-2}{N_v} \cdot \sum_{p=1}^{N_v} \left[t_p(m) - \sum_{n=1}^{N+1} w(m,n) \cdot x_{ap}(n) \right] \cdot x_{ap}(l) \tag{2.10}$$

This can be simplified as:

$$\frac{\partial E}{\partial w(m,l)} = \frac{-2}{N_v} \cdot \sum_{p=1}^{N_v} \left[t_p(m) \cdot x_{ap}(l) - \sum_{n=1}^{N+1} w(m,n) \cdot x_{ap}(n) \cdot x_{ap}(l) \right] \tag{2.11}$$

$$\frac{\partial E}{\partial w(m,l)} = \frac{-2}{N_v} \cdot \left[\sum_{p=1}^{N_v} \left(x_{ap}(l) \cdot t_p(m) \right) - \sum_{n=1}^{N+1} w(m,n) \cdot \sum_{p=1}^{N_v} \left(x_{ap}(n) \cdot x_{ap}(l) \right) \right] \tag{2.12}$$

Writing in terms of auto and cross correlation matrices:

$$\frac{\partial E}{\partial w(m,l)} = \frac{-2}{N_v} \cdot \left[c(l,m) - \sum_{n=1}^{N+1} w(m,n) \cdot r(n,l) \right] \tag{2.13}$$

To minimize the mean squared error, this derivative is equated to zero, generating M sets of (N+1) linear equations in (N + 1) variables:

$$\begin{aligned} c(1,m) &= \sum_{n=1}^{N+1} w(m,n) \cdot r(n,1) \\ c(2,m) &= \sum_{n=1}^{N+1} w(m,n) \cdot r(n,2) \\ &\vdots \\ c(N+1,m) &= \sum_{n=1}^{N+1} w(m,n) \cdot r(n,N+1) \end{aligned} \quad \text{for } 1 \le m \le M \tag{2.14}$$

These equations can be represented in a compact way as:

$$\mathbf{R} \cdot \mathbf{W}^{T} = \mathbf{C} \tag{2.15}$$

where **R** is the (N+1) × (N+1) autocorrelation matrix, **W** is the M × (N+1) weight matrix, and **C** is the (N+1) × M cross-correlation matrix.

These equations can be solved using conjugate gradient [36] or the Schmidt procedure [37].

2.3.1 Schmidt Procedure

The Schmidt procedure maps the inputs into an orthonormal system which speeds up the computation of weights. For un-ordered basis functions **x** of dimension N_u, where N_u may be (N + 1), the m^{th} orthonormal basis function x'_m is defined as [37, 48]:

$$x'_m = \sum_{k=1}^{m} a_{mk} x_k$$

where **A** is a lower triangular N_u by N_u orthonormalization matrix.

Initially, x'_1 is found as $a_{11}x_1$ where,

$$a_{11} = \frac{1}{\|x_1\|} = \frac{1}{r(1,1)}$$

For $2 \leq m \leq N_u$, we first perform

$$c_i = \sum_{q=1}^{i} a_{iq} r(q,m) \quad ,$$

for $1 \leq i \leq m-1$. Second, we set $b_m = 1$ and get

$$b_k = -\sum_{i=k}^{m-1} c_i a_{ik} \quad ,$$

for $1 \leq k \leq m-1$. Lastly we get coefficients a_{mk} for the lower triangular matrix **A** as

$$a_{mk} = \frac{b_k}{\left(r(m,m) - \sum_{i=1}^{m-1} c_i^{\,2} \right)^{1/2}}$$

Once we have the orthonormal basis functions, the linear mapping weights in the orthonormal system can be simply found as

$$w'(i,m) = \sum_{k=1}^{m} a_{mk} c(i,k) \quad 1 \le \mathrm{i} \le \mathrm{M}$$

The orthonormal system's weights $\mathbf{W}'$ can be mapped back to the original system's weights **W** as

$$w(i,k) = \sum_{m=k}^{N_u} a_{mk} \cdot w_0'(i,m)$$

where $w'(i,k) = 0$, for $1 \le \mathrm{i} \le \mathrm{M}$.

Equation (2.6) can be written for a single output *i* as:

$$E(i) = \sum_{p=1}^{N_v} \left[t_p(i) - \sum_{n=1}^{N_u} w(i,n) \cdot x_{ap}(n) \right]^2 \tag{2.16}$$

In the orthonormal system,

$$E(i) = \left\langle \left[t_p(i) - \sum_{k=1}^{N_u} w'(i,k) \cdot x'_{apk} \right], \left[t_p(i) - \sum_{q=1}^{N_u} w'(i,q) \cdot x'_{apq} \right] \right\rangle \tag{2.17}$$

$$E(i) = \left\langle t_p(i), t_p(i) \right\rangle - \sum_{k=1}^{N_u} \left(w'(i,k) \right)^2 \tag{2.18}$$

The total training error in the orthonormal system is given by:

$$E = \sum_{i=1}^{M} E(i) \tag{2.19}$$

2.3.2 Complexity of the Schmidt Procedure

The number of multiplies $\mathrm{M_{Schmidt}}$ needed to solve for the linear network's weights using Schmidt procedure, given pre-computed **R** and **C** matrices, can be computed as:

$$M_{Schmidt} = {N_u}^3 + (M+1) \cdot {N_u}^2 + \left[M + \frac{11}{6} \right] \cdot N_u - 13 \tag{2.20}$$

where $\mathrm{N_u} = \mathrm{N} + 1$, as before.

2.4 Performance Evaluation of Neural Networks

2.4.1 Training & Validation Error

Training error is defined as the average error produced by the network when it is subjected to all the patterns that it was trained on. Validation error is the average error produced by the network when it is made to process new data not seen during training. Since the network is already optimized to reduce training error, this error is generally smaller than the validation error.

2.4.2 K-fold Validation

K-fold validation [49] is a cross validation technique for assessing how the network will generalize on an independent dataset. The goal is to obtain K training and validation set pairs. The K results from the folds are averaged to obtain a single, more reliable estimate. Given a single data set, randomly divide it into K disjoint subsets D_k of equal size, for $1 \leq k \leq K$. Form K separate training/validation set pairs as:

$$T_k = \left(\bigcup_{j \neq k} D_j \right), V_k = D_k$$

for $1 \leq k \leq K$.

We obtain K training and validation data set pairs $\{T_k, V_k\}$. For each k value where $1 \leq k \leq K$, the network is trained with the k^{th} training data set T_k to obtain the corresponding training error E_{tk}. and then the trained network is validated against the corresponding validation data set V_k, to obtain the corresponding validation error E_{vk}. The average of the K training and validation errors is used as the average error of the network.

$$E_t = \frac{1}{K} \sum_{k=1}^{K} E_{tk} \tag{2.21}$$

$$E_v = \frac{1}{K} \sum_{k=1}^{K} E_{vtk} \tag{2.22}$$

2.4.3 Execution Times

It is a difficult to accurately determine the execution time by calculating metrics on the code. The amount of time taken by a piece of code to run depends on the hardware it is running on, underlying implementation of libraries, and the load conditions on the system. But a fair comparison between two pieces of code can be made if they are both compiled into executable code using the same method, and executed on the same hardware, under identical conditions.

We compare the execution times of several algorithms in this thesis. For accuracy, each algorithm is executed 3 times on 10-fold validation data sets, and the average execution time is calculated as:

$$T_{execution} = \frac{1}{10} \sum_{k=1}^{10} \frac{1}{3} \sum_{i=1}^{3} T_{ki} \tag{2.23}$$

where T_{ki} is the execution time for the k^{th} 10-fold data set, on the i^{th} repetition.

CHAPTER 3

PIECEWISE LINEAR NETWORKS REVIEW

3.1 Piecewise Linear Networks

Piecewise linear functions are characterized by functional relationships composed of a finite number of linear regions adjoining each other. The changeover from one linear region to another is determined at the point where a quantity becomes greater or less than another quantity. Piecewise linear networks divide the N dimensional input space into K volumes or clusters. A clustering algorithm such as SOM or Sequential Leader is used to obtain K cluster center vectors $\mathbf{m}_k$. For each cluster, a linear network is independently trained, producing K weight matrices. The output vector is obtained by first determining to which cluster the pattern belongs to, and then multiplying the weight matrix with the pattern vector. An additional global linear network can be trained to remove the global mapping from the outputs [35, 50].

As the number of training patterns N_v tends to infinity, partition based estimates of a regression function converge to the true function and the mapping converges to the corresponding Bayes estimate. Piecewise linear networks are thus consistent nonparametric estimators [51, 52].

3.2 Piecewise Linear Network Structure

The structure of a Piecewise linear network is shown in Figure 3.1.

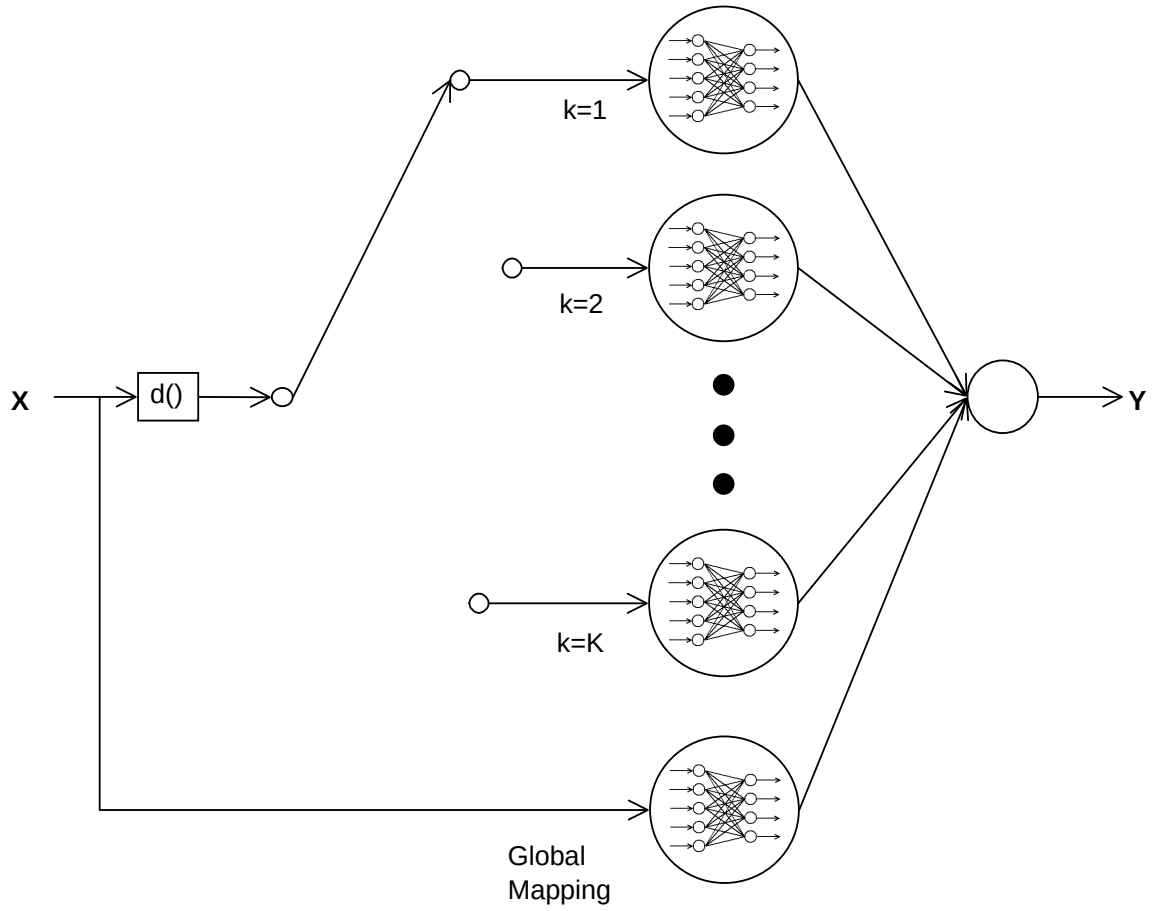

Figure 3.1 Piecewise linear network structure.

It consists of:

1. K cluster center vectors $\mathbf{m}_k$, each of dimension N, where $1 \leq k \leq K$.
2. K weight matrices $\mathbf{W}_k$, each of dimension M by (N+1), for storing trained weights for each cluster.
3. A global linear mapping weights matrix $\mathbf{W}_g$ of size M by (N+1) that stores the global linear mapping from inputs to the outputs.
4. A weighted distance measure d(·) which can be used to determine cluster membership for patterns while deemphasizing the less useful features in the

pattern vector. Weights for the distance measure are stored in an array $\mathbf{w}_{dm}$ of size N.

5. Means and standard deviations of the input patterns represented by vectors **μ** and **σ** of lengths N each.

The storage capacity of a PLN is given by:

$$P_{ab} = K \cdot N + K \cdot (N+1) \cdot M \tag{3.24}$$

3.2.1 Processing Input Patterns

To determine the output for a given pattern,

1. The p^{th} input vector $\mathbf{x}_p$ is augmented as $\{\mathbf{x}_p : 1\}^T$ for $1 \le p \le N_v$, where the extra input 1 handles any bias or threshold in the output.
2. All input pattern elements except for the $(N+1)^{th}$ are normalized by subtracting from them the mean and dividing by the standard deviation,

$$x_p(n) = \left(x_p(n) - \mu(n)\right) / \sigma(n) \quad \text{for } 1 \le n \le N \tag{3.25}$$

3. The global linear mapping $\mathbf{y}_{pg}$ for the pattern is obtained by multiplying it with the global weights matrix $\mathbf{W}_g$.

$$\mathbf{y}_{pg} = \mathbf{W}_g \cdot \mathbf{x}_p \tag{3.26}$$

4. The cluster membership for the pattern is determined by using the distance measure d() to find the k such that:

$$d\left(\mathbf{x}_p, \mathbf{m}_k\right) = \min_n \left(d(\mathbf{x}_p, \mathbf{m}_n)\right) \tag{3.27}$$

where d() is the weighted Euclidian distance measure defined equation 2.3.

5. The pattern is multiplied with the weights matrix $\mathbf{W}_k$ belonging to the k^{th} cluster.

$$\mathbf{y_p} = \mathbf{W}_k \cdot \mathbf{x}_p \tag{3.28}$$

6. The output from the cluster's weight matrix is added to the output from the global linear network $\mathbf{y}_{pg}$ defined in equation (3.26) to produce the final output.

$$\mathbf{y_p} = \mathbf{y_{pg}} + \mathbf{y_p} \tag{3.29}$$

Note that $\mathbf{W}_g$ can be absorbed into each matrix $\mathbf{W}_k$, and therefore eliminated from the calculation.

3.2.2 Equivalent MLP Based on Multiplies

The number of multiplies required to compute the output $\mathbf{y}_p$ from a zero mean normalized input pattern $\mathbf{x}_p$ given an existing network consists of $M \cdot N$ multiplies to compute $\mathbf{y}_{pg}$, $2 \cdot K \cdot N$ multiplies to determine the cluster membership using the weighted distance measure, and $M \cdot N$ multiplies to compute $\mathbf{y}_p$. Thus,

$$M_{PLN} = 2 \cdot M \cdot N + 2 \cdot K \cdot N \tag{3.30}$$

A single layer MLP requires the following number of multiplies to process one input vector:

$$M_{MLP} = N \cdot (N_h + M) + N_h \cdot M + 2 \cdot N_h \tag{3.31}$$

By equating the expressions of (3.30) and (3.31), we find the number of hidden units required on an equivalent MLP as

$$N_h = \frac{2 \cdot N \cdot K + N \cdot M}{N + M + 2} \tag{3.32}$$

Here it is assumed that each evaluation of the sigmoid activation takes 2 multiplies.

3.3 Basic PLN Training

Piecewise linear networks are designed for a given number of clusters. A larger number of clusters usually gives better results than a small number of clusters. The training

process involves clustering the training vectors into the desired number of clusters. To prevent memorization, the number of patterns per cluster $N_v(k)$ should be more than N+1. For each cluster, a linear network is trained for the patterns belonging to that cluster by minimizing the following error equation:

$$E(k) = \sum_{p \in s_K} \left\| \mathbf{t}_p - \mathbf{W}_k \cdot \mathbf{x}_p \right\|^2 \tag{3.33}$$

where S_k is the set of pattern numbers for the k^{th} cluster, and $\mathbf{W}_k$ is an M × (N + 1) weights matrix for the k^{th} cluster.

The weights matrix $\mathbf{W}_k$ for the k^{th} linear network can be solved for by using the procedure described in section 2.3, with the matrix $\mathbf{W}$ replaced by $\mathbf{W}_k$, and pattern indices p referring to patterns belonging to the k^{th} cluster. The autocorrelation matrix $\mathbf{R}_k$ and cross correlation matrix $\mathbf{C}_k$ for the k^{th} cluster are defined as:

$$r_k(n,l) = \frac{1}{N_v(k)} \sum_{p \in s_K} x_p(n) \cdot x_p(l) \tag{3.34}$$

$$c_k(n,i) = \frac{1}{N_v(k)} \sum_{p \in s_K} x_{ap}(n) \cdot t_p(i) \tag{3.35}$$

where S_k is the set of pattern numbers and $N_v(k)$ is the count of patterns for the k^{th} cluster. Equation (2.13) can be rewritten in terms of elements of $\mathbf{R}_k$, $\mathbf{C}_k$ and $\mathbf{W}_k$ as:

$$\frac{\partial E(k)}{\partial w_k(m,l)} = -2 \cdot \left[c_k(l,m) - \sum_{n=1}^{N+1} w_k(m,n) \cdot r_k(n,l) \right] \tag{3.36}$$

For each cluster, this yields M sets of (N + 1) linear equations in (N + 1) variables, similar to those in (2.14) and (2.15), which can be solved using the conjugate gradient method, or the Schmidt procedure described in section 2.3.1.

The total network error is the sum of mapping error in each cluster divided by the number of patterns.

$$E = \frac{1}{N_v} \sum_{k=1}^{K} E(k) \tag{3.37}$$

3.4 PLN Pruning

PLN pruning is the process of deleting less useful PLN modules from a network designed with a large number of modules. Usefulness of a module is measured in terms of the reduction in the global MSE by the presence of that module. Pruning those modules whose removal leads to the least increase in the MSE produces more compact PLN structures [35]. Also, pruning with validation is a way to implement structural risk minimization [53].

3.4.1 Basic Pruning Algorithm

The algorithm described here removes one least useful cluster from the existing cluster set:

1. Let k be the index of the module to be potentially eliminated and E_k the error of the network after module k has been pruned. Set E = 0, E_k = 0, for $1 \leq k \leq K$.
2. For every input vector $\mathbf{x}_p$, p = 1 to N_v:
    a. Find the closest cluster k_1 (the cluster it currently belongs to) and the second closest cluster k_2 (the cluster it will end up in if the closest cluster k_1 is deleted) for the pattern.
    b. Compute the error for the pattern if it belonged to the first cluster as e_1, and if it belonged to the second cluster as e_2.

c. Supposing cluster k_1 was deleted, the pattern will move from cluster k_1 to cluster k_2 since it is the second closest cluster. This causes the pattern to cease contributing to the error for cluster k_1, and instead start contributing to the error in cluster k_2. For k = 1 to K, accumulate errors as:

$$E_k \leftarrow E_k + e_1, k \neq k_1$$
$$E_k \leftarrow E_k + e_2, k = k_1$$

3. Delete the cluster *kmin* with the smallest pruning error E_{kmin}, distributing its patterns among the remaining clusters. Recompute the linear mapping for the modified clusters.

This process can be repeated multiple times to remove multiple clusters, though it does not guarantee that the resulting network is the best resultant network of this size. Optimal pruning of *m* modules from a PLN requires evaluating all possible PLNs with (*K - m)* modules, and finding the one with the least MSE. This involves choosing *m* out of *K* clusters, which can be done in $\binom{K}{m}$ ways. The algorithm can be suitably modified to remove a fixed number of clusters together, such that the resultant MSE increase is the least. But this is not very practical since the number of all possible combinations becomes prohibitive even from modest values of K and m [38].

3.4.2 Algorithm for Pruning Multiple Clusters at a Time

Given K clusters, the following algorithm prunes m clusters from the PLN at a time:

1. Let the number of possible sets of m clusters to be eliminated be U = $\binom{K}{m}$

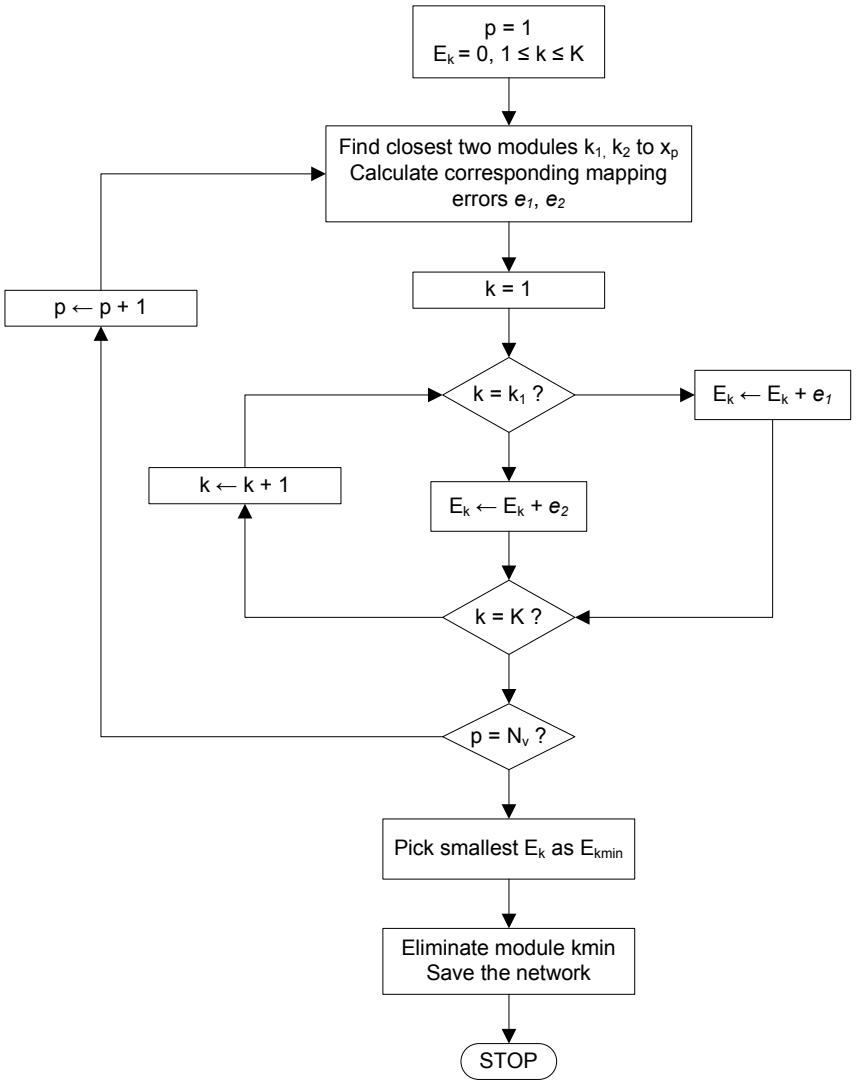

Figure 3.2 PLN basic pruning algorithm.

2. Determine all U possible sets of m cluster indices that can be eliminated as set_u for $1 \le u \le U$
3. Let E_u denote the error when the clusters in set_u are deleted. Set $E_u=0$ for $1 \le u \le U$
4. For each input pattern $\mathbf{x}_p$:
    a. Find indices of the (m+1) clusters closest to $\mathbf{x}_p$ as k_i, such that k_i is the i^{th} closest cluster to $\mathbf{x}_p$, for $1 \le i \le (m+1)$

b. Also find the error for the pattern if it belonged to the k_i^{th} cluster as e_i for $1 \leq i \leq (m+1)$

c. Let u=1

d. if $k_1 \in set_u$,

   i. Let v=2

   ii. If $k_v \notin set_u$, $E_u = E_u + e_v$. Go to f.

   iii. $v \leftarrow v+1$

   iv. If $v \leq (m+1)$, Go to ii.

e. else, $E_u \leftarrow E_u + e_1$

f. $u \leftarrow u+1$

g. If $u \leq U$, Go to d.

5. Find the smallest E_u as E_{umin}.

6. Eliminate the clusters in set_{umin}. Recompute the linear mapping for the modified clusters.

We can see that the number of possible sets U can be very large. For example, with K=50 and m=5, the algorithm requires 2,118,760 sets of 5 elements each, which needs a lot of memory.

3.4.3 Computational Complexity of the Basic Pruning Algorithm

The computational complexity of the pruning process can be represented by the number of multiplies required to delete one cluster to obtain K-1 clusters starting from K clusters, denoted by M_{PPLN}.

Let K denote the current number of cluster and $M_{Schmidt}$ denote the number of multiplies required to solve a linear network using the Schmidt procedure.

For each pattern, we need:

2 · K · N multiplies to determine new cluster membership using weighted Euclidian distance measure

N · (N + 1) / 2 + N multiplies for computing incremental auto-correlation matrix $\mathbf{R_k}$ (lower or upper half of the symmetric matrix)

N · M multiplies for computing incremental cross correlation matrix $\mathbf{C_k}$

M multiplies for computing $\mathbf{E_t}$, the energy at each output

(K - 1) · $M_{Schmidt}$ to compute K-1 linear mappings for the new network

Therefore, for N_v patterns:

$$M_{PPLN}(K) = N_v \cdot (2 \cdot K \cdot N) + 2 \cdot (N+1) \cdot M + N_v \cdot (K \cdot N + N \cdot (N+1)/2 + N + N \cdot M + M) + (K-1) \cdot M_{Schmidt} \quad (3.38)$$

where $M_{Schmidt}$ has been developed in Section 2.3.2.

CHAPTER 4

CLUSTERING TECHNIQUES

4.1 Clustering

Clustering is the process of grouping together related mathematical objects such as vectors. There is no universally agreed upon definition. Most researchers describe a cluster by considering the internal homogeneity and the external separation [45, 54]. Clustering is useful in designing nearest neighbor classifiers, and important to the subject of this thesis, in the design of piecewise linear networks. The goal of clustering in PLN pruning is to obtain K good initial clusters which can be pruned down to desired number of clusters $K_{desired}$. We may need to specify the desired number of clusters in algorithms such as SOM and K-means, while we may obtain an unknown number of clusters on using Sequential Leader clustering [55, 56].

4.2 Types of Clustering Methods

Clustering is ubiquitous, and a wealth of clustering algorithms has been developed to solve different problems in specific fields. However, there is no clustering algorithm that can be universally used to solve all problems [45, 57].

Clustering techniques are mostly classified into partitional or divisive clustering and hierarchical clustering. Hierarchical clustering aims to build up a nested hierarchy of clusters, starting from either singleton clusters to a cluster containing all the patterns, or vice versa. Partitional clustering directly divides data into a pre-specified number of clusters [45, 54, 55].

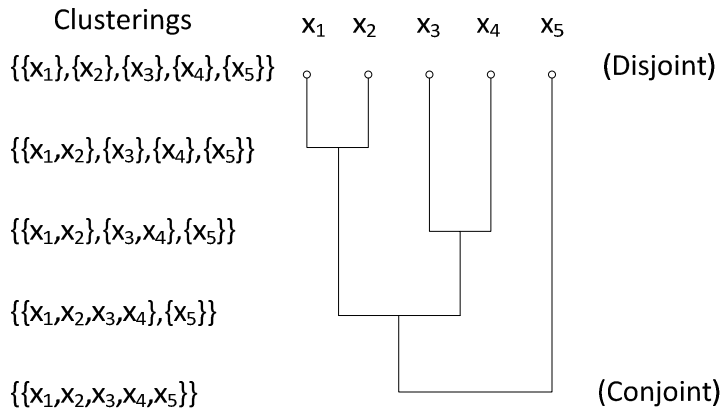

Figure 4.1 Example of hierarchical clustering.

Some clustering algorithms such as SOM and K-means require the number of clusters to be pre-determined, while in Sequential leader, the number of clusters obtained depends on the threshold used for determining the boundaries of a cluster. It is possible for some SOM clusters to end up with no patterns [58].

We have considered several partitional clustering algorithms such as SOM, K-means and sequential leader, because of their simplicity and speed.

4.3 SOM Clustering

SOM clustering was devised by Teuvo Kohonen [58] as a tool for visualizing high dimensional data in low dimensions. Using a 2-D grid of nodes, a 2-dimensional topological view of the multidimensional data is obtained. The clusters so formed contain

similar patterns. SOM differs from vector quantization in that the clusters formed themselves are ordered. Another important observation is that the cluster centers need not be vectors themselves. There only needs to be defined a distance measure from the input pattern space to the cluster entities [1, 58].

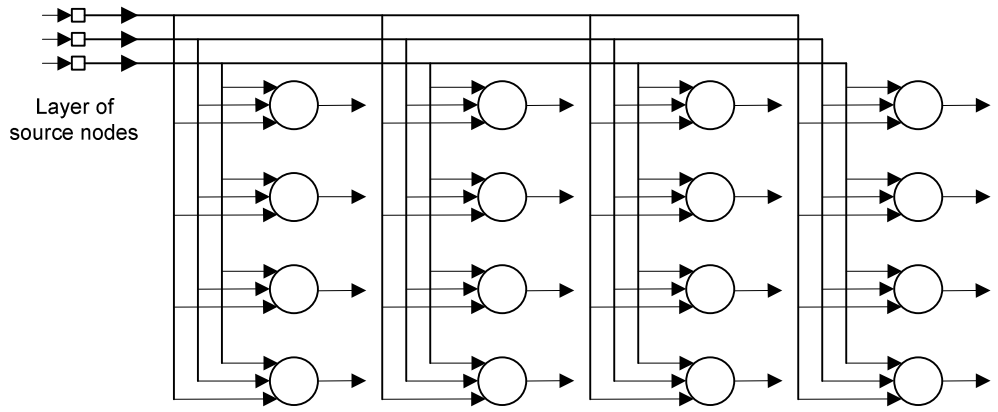

Figure 4.2 Mapping input patterns to a two dimensional lattice of neurons with SOM clustering.

The number of clusters or nodes in a two or higher dimensional map, are pre-known. Associated with each node are a weight vector or the center vector, and a position in the map. Input patterns are classified to one of the nodes, and the nodes themselves change their weights and positions on the map.

4.3.1 SOM Algorithm for 1-D cluster indexing

The SOM algorithm for clustering vectors into a one dimensional map is described [59]. Given K, N_v vectors $\mathbf{x}_p$ of dimension N, decreasing functions z(t) and N(t), and N_{it}.

1. Find the means **m** and standard deviations **σ** of each of the N inputs/features as:

$$m(n) = \frac{1}{N_v} \sum_{p=1}^{N_v} x_p(n) \tag{4.1}$$

$$\sigma(n) = \sqrt{\frac{1}{N_v} \sum_{p=1}^{N_v} (x_p(n) - m(n))^2} \quad (4.2)$$

2. Initialize K mean vectors $\mathbf{m}_k$ with random numbers having the same means and standard deviations as that found in step 1.
3. Let *t* be a measure of time proportional to the number of patterns processed in all iterations up to that time. For iteration number i_t, and pattern index *p* in the current iteration, *t* can be defined as

$$t = p + (i_t - 1) \cdot N_v \quad (4.3)$$

4. For each pattern $\mathbf{x}_p$ in the data file, find the cluster index *n* of the cluster $\mathbf{m}_n$ closest to it. Then modify all clusters $\mathbf{m}_k$ having cluster index *k* in the decreasing neighborhood of *n*, such that $|n - k| \leq N(t)$, by shifting them towards the pattern by a distance proportional to the distance between the pattern and the cluster, and a decreasing learning rate z(*t*).

$$\mathbf{m}_k = \mathbf{m}_k + z(t) \cdot [\mathbf{x}_p - \mathbf{m}_k] \text{ for } |k-n| \leq N(t) \quad (4.4)$$

5. Repeat step 4 for the desired number of iterations or till z(*t*) becomes 0.

4.3.2 Choice of Decreasing Functions z(t) and N(t)

With each clustering iteration, the clusters re-align themselves as new patterns are learned and new passes are made through the data file. To provide stability to the algorithm and make the clusters converge, the learning rate is decreased as a function of "time", defined earlier in equation (4.3).

z(t) is a learning rate for the cluster centers and is commonly chosen to be a decreasing function. Thus with increasing time, the changes made to the cluster centers become

smaller and smaller, tending towards zero for very large time, causing the learning to stop. One choice for the learning rate function is:

$$z_1(t) = a_1 \cdot e^{-t/T_1}$$

N(t) is an exponentially decreasing radius function which controls the neighborhood of clusters that get deformed when a pattern is added to a cluster. The natural tendency is for the neighboring clusters to move towards the cluster to which the pattern was added.

$$N(t) = a_2 + a_3 \cdot e^{-t/T_2}$$

One suitable set of values of a_1, a_2, a_3, T_1 and T_2 can be obtained as:

$$a_1 = \frac{K}{N_v}, \quad a_2 = 0, \quad a_3 = \frac{K}{10}$$

$$T_1 = \frac{N_v \cdot N_{it}}{3}, \quad T_2 = \frac{N_v \cdot N_{it}}{10}$$

4.4 K-means Clustering

K-means clustering starts with a set of K initial cluster centers. It then repeatedly reclassifies patterns to these clusters and recomputes the cluster centers till a stable distribution of clusters is obtained [54, 56].

4.4.1 K-means Clustering Algorithm

Given the number of desired clusters K, the number of iterations N_{it}, N_v training vectors $\mathbf{x_p}$ of dimension N, a distance measure d(), and an initial set of means $\mathbf{m_k}$ selected randomly or by some heuristic:

1. $i_t = 0$
2. $i_t = i_t + 1$

3. Calculate means as

$$\mathbf{m_k} = \frac{1}{N_v(k)} \sum_{p:m(p)=k} \mathbf{x_p}$$

4. Reclassify $\mathbf{x_p}$s, in one data pass. If $\mathbf{x_p}$ belongs to the k^{th} cluster, then set m(p)=k. Thus, m(p) specifies the cluster membership of the p^{th} pattern. If any clusters change and $i_t < N_{it}$, then go to step 2. Otherwise stop.

4.4.2 Analysis

The error function being minimized for K-means clustering is:

$$E = \frac{1}{N_v} \sum_{p=1}^{N_v} d(\mathbf{x_p}, \mathbf{m}_{m(p)}) \tag{4.5}$$

or

$$E = \frac{1}{N_v} \sum_{k=1}^{K} E_k \tag{4.6}$$

where,

$$E(k) = \sum_{p:m(p)=k} d(\mathbf{x_p}, \mathbf{m_k}) \tag{4.7}$$

The mean recalculation step minimizes the distance between the cluster mean and the cluster members, thus minimizing the error within each cluster E(k).

The reclassification step moves a pattern from cluster k1 to cluster k2 if $d(\mathbf{x_p},\mathbf{m}_{k2})$ is less than $d(\mathbf{x_p},\mathbf{m}_{k1})$. This results in $d(\mathbf{x_p},\mathbf{m}_{k1})$ being subtracted from E(k1) and $d(\mathbf{x_p},\mathbf{m}_{k2})$ being added to E(k2). Since a larger value is being subtracted than what is being added, the net error E comes down [60, 61].

CHAPTER 5

IMPROVED SEQUENTIAL LEADER TRAINING

5.1 Introduction

Sequential Leader or simply the Leader algorithm is a fast clustering method that requires only a single pass through the data, if any number of resultant clusters is acceptable [54]. Most applications have a requirement for a predefined number of clusters. Thus it can take many trials to arrive at the optimal threshold to be used for this algorithm [56]. In this chapter, we develop several threshold estimation techniques which yield clusters very close to the desired count.

Even when using a good threshold estimate, the number of clusters rarely matches the desired number exactly. To overcome this problem, the threshold can be estimated for a number of clusters slightly larger than originally desired, and the extra clusters deleted by a suitable method. A few methods to select the extra clusters from the pool of initially generated clusters are presented and compared.

5.2 Basic Sequential Leader algorithm

The first pattern is normally chosen to be the first cluster. The algorithm then makes one pass through the data, assigning patterns to the first cluster leader from which its distance

is under the distance threshold, and forms a new leader for patterns that are not close to any of the existing leaders [56].

Algorithm:

1. Given patterns to be clustered $\{\mathbf{x}_p\}_{p=1}^{N_v}$, a threshold T, and a distance measure d().
2. Start with K = 1 clusters, and pattern index p = 1. Assign the first pattern $\mathbf{x}_1$ as the first cluster center $\mathbf{m}_1$.
3. Increment p and read the next pattern $\mathbf{x}_p$.
4. Start with cluster index k = 1.
5. If $d(\mathbf{x}_p, \mathbf{m}_k) \leq T$, then assign pattern $\mathbf{x}_p$ to cluster k. Go to step 9.
6. If $d(\mathbf{x}_p, \mathbf{m}_k) > T$ then increment k.
7. If $k \leq K$, go to step 5.
8. If $d(\mathbf{x}_p, \mathbf{m}_k) > T$ for all k, then a new cluster is created with K increased by 1. Set $\mathbf{m}_K = \mathbf{x}_p$.
9. If $p = N_v$, stop. Else, go to step 3.

5.3 Properties

The advantage of the sequential leader algorithm is that it is fast, requiring only a single pass through the data when T is known. Since the patterns are read one by one sequentially, it is not required to store all patterns in main memory.

This algorithm also has several disadvantages

1. The clusters that are formed depend on the ordering of patterns in the input file. The first pattern is always the first cluster.
2. The starting clusters tend to have more patterns than the later clusters. This is because a pattern is allocated to the first cluster it is close enough to. This does

not give the remaining clusters an opportunity to absorb the pattern even though the pattern may be closer to them than the one acquiring the pattern. It should be noted that this does not affect the clusters that are generated. This problem can be remedied by modifying the algorithm such that patterns are allocated to the closest cluster leader from which the distance is less than the threshold, rather than the first.

3. The number of clusters that are formed is determined by the threshold and cannot be controlled when using an arbitrary threshold. To avoid several trial and error iterations for trying to come up with the desired number of clusters, a good threshold estimate can be tried.

5.4 Threshold Estimation

Since the distribution of patterns in the N dimensional vector space and their average distances is not known apriori, it is difficult to make a good estimate of T. Using a brute force technique starting with an arbitrary threshold requires several increments or decrements of variable step size before arriving at the correct threshold. Each iteration requires an extra pass through the data file, which can be slow when the size of the data files is large or the media access speeds are slow. A few statistical and systematic methods for threshold estimation are explained. In all methods described except for binary search and brute force algorithms, brute force was ultimately applied starting with the original estimate, if the original estimate itself was not an acceptable threshold.

5.4.1 Brute Force

A brute force approach for threshold estimation is simplest, but several trial iterations of sequential leader may be required to arrive at the correct threshold. A reasonable initial estimate for the threshold can be made as follows:

1. Make one pass through the data file to compute the mean vector **m**.
2. Given the total number of patterns N_v and a distance measure *d(·)*, measure the average distance of each pattern from the mean vector as:

$$d = \frac{1}{N_v} \sum_{p=1}^{Nv} d(\mathbf{x_p}, \mathbf{m}) \tag{5.1}$$

3. Calculate the initial threshold as $T_{est} = d/10$.

Using the calculated threshold, the correct threshold can be arrived at as follows:

Given the desired number of clusters $K_{desired,}$

1. Obtain the initial threshold estimate $T = T_{est}$.
2. Cluster the data file using the sequential leader algorithm and the current threshold to obtain the cluster count K.
3. If $K > K_{desired}$, increment the threshold as $T = T + T/10$.
4. If $K < K_{desired}$, decrement the threshold as $T = T - T/10$.
5. If $K = K_{desired}$, then stop. Else, go to step 2.

A problem with this approach is that it may not converge or may take a very large number of iterations, especially with the floating point precision limits.

5.4.2 Volumetric Approach

The volumetric approach to threshold estimation can be used when the patterns are uniformly distributed in the N dimensional feature space. Then we can use the equation

$$\frac{V}{V_c} = K \tag{5.2}$$

where V denotes the total volume containing the patterns and V_c is the maximum volume per cluster.

Since the sample space can extend to infinity along each dimension, the patterns lying at the outer extremes of the pattern space are used to compute the total volume. If the Squared Euclidean Distance measure is used in the Sequential Leader algorithm, the total volume in the enclosing hypercube is given by:

$$V = \prod_{n=1}^{N} \left[x_{\max}(n) - x_{\min}(n) \right] \tag{5.3}$$

The presence of outliers can make the above estimate incorrect. Other heuristics can be used to improve this result. If this approach is applied to a data set where each dimension has a normal distribution, then a length of 4 times the standard deviation is assumed to contain all the patterns along a given dimension. Thus the volume can be estimated as:

$$V = 4^N \cdot \prod_{n=1}^{N} \sigma(n) \tag{5.4}$$

where $\sigma(n)$ is the standard deviation of the pattern distances from the mean along the n^{th} dimension.

If a weighted squared Euclidean distance measure is used, these formulae become:

$$V = \prod_{n=1}^{N} \sqrt{w(n)} \cdot \left[x_{\max}(n) - x_{\min}(n) \right] \tag{5.5}$$

and

$$V = 4^N \cdot \prod_{n=1}^{N} \sqrt{w(n)} \cdot \sigma(n) \tag{5.6}$$

where w(n) is the weight for the n^{th} dimension.

Dimensions that are very narrow can dominate the computation and quickly diminish the total volume. Thus, the dimensions which do not have a significant distribution of patterns along them are discarded from the computation. In this implementation, if a dimension is smaller than $1/10^{th}$ of the largest dimension, it is not included in the volume computation. The reduced count of dimensions is denoted by N_1.

This volume is divided by the desired number of clusters to obtain the volume in each cluster V_c. Then the radius of this N_1 dimensional volume is used as the threshold.

$$V_c = \left(2 \cdot \sqrt{T}\right)^{N_1} \tag{5.7}$$

Plugging this in equation (5.2), we get:

$$\frac{V}{\left(2 \cdot \sqrt{T}\right)^{N_1}} = K \tag{5.8}$$

And upon solving for T we get:

$$T = \frac{1}{4}\left(\frac{V}{K}\right)^{2/N_1} \tag{5.9}$$

This often gives a good threshold for initializing the brute force algorithm.

5.4.3 Linear and Quadratic Fits

Since the number of clusters decreases monotonically with increasing threshold, linear interpolation was applied to approximate the correct threshold. Two thresholds T_1, T_2, and their corresponding number of clusters K_1, K_2 are required. To fix two thresholds, some statistics related to pattern distance from the mean vector were calculated.

If the mean vector of all patterns is denoted by **m** and d() is the weighted Euclidian distance measure,

$$d_{mean} = \frac{1}{N_v} \sum_{p=1}^{N_v} d(\mathbf{x}_p, \mathbf{m})$$

$$d_{\max} = \max_{1 \le p \le N_v} d(\mathbf{x}_p, \mathbf{m})$$

$$d_{\min} = \min_{1 \le p \le N_v} d(\mathbf{x}_p, \mathbf{m})$$

$$d_{median} = \frac{d_{\max} + d_{\min}}{2}$$

T_1 and T_2 are chosen to be the mean and median respectively:

$$T_1 = d_{mean} \tag{5.10}$$

$$T_2 = d_{median} \tag{5.11}$$

The corresponding number of clusters K_1 and K_2 were obtained by two passes through the sequential leader algorithm. The unknown threshold T that would yield $K_{desired}$ clusters was obtained as:

$$T = T_1 + \frac{T_2 - T_1}{K_2 - K_1}(K_{desired} - K_1) \tag{5.12}$$

Computation of K_1 and K_2 require 2 passes through the data, which were added to the number of trials required for this method in Table 5.1.

To approximate the nonlinear relationship between the threshold and the number of clusters, a quadratic fit was also tried. Three T, K pairs are required to perform a quadratic interpolation. T_1 and T_2 were chosen to be the same as before, and a third threshold T_3 was chosen as:

$$T_3 = d_{\min} + \frac{d_{mean} - d_{\min}}{8} \tag{5.13}$$

The number of clusters produced using threshold T_3 was stored as K_3. The unknown threshold T was determined as:

$$T = A \cdot K_{desired}^2 + B \cdot K_{desired} + C \tag{5.14}$$

where,

$$A = \frac{T_3 - T_2}{(K_3 - K_2)(K_3 - K_1)} - \frac{T_1 - T_2}{(K_1 - K_2)(K_3 - K_1)}$$

$$B = \frac{T_1 - A \cdot (K_2^2 - K_1^2)}{K_1 - K_2}$$

$$C = T_1 - A \cdot K_1^2 - B \cdot K_1$$

Computation of T_1, T_2, and T_3 require 3 passes through the data, which are added to the number of trials required for this method in Table 5.1.

5.4.4 Binary Search

Binary search is a fast method to converge to the correct threshold. The approximation techniques described above seldom arrive at the correct threshold, and the deviation can sometimes be too large to use them as a seed for a brute force search. The numbers of clusters for three different thresholds are obtained, and binary search is performed in one of the four intervals thus formed. The three thresholds used in the computation are:

$$T_1 = \frac{(d_{\max} + d_{\min})}{2} \tag{5.15}$$

$$T_2 = d_{mean} \tag{5.16}$$

$$T_3 = d_{\min} + \frac{(d_{\max} + d_{\min})}{4} \tag{5.17}$$

5.4.5 Results

Equations (5.12) and (5.14) fail to give a solution if any two of K_1, K_2 or K_3 are equal to one another. Here, thresholds were adjusted till distinct values of K_1, K_2 and K_3 were obtained. The extra passes required were added to the numbers in Table 5.1.

Due to memory constraints, the algorithm was able to store a limited number of cluster center vectors as they were formed. In cases when the threshold was too low and producing more clusters than what could be stored, the threshold was incremented iteratively till a valid number of clusters were obtained. These extra passes through the data are also reflected in Table 5.1.

In cases where any of K_1, K_2 or K_3 was equal to $K_{desired}$, further computations were not performed.

In all methods described except for binary search and brute force algorithms, brute force was ultimately applied starting with the original estimate from the method, if the original estimate itself was not an acceptable threshold. This added a large number of data passes for the cases where the estimated threshold deviated significantly from the ideal value.

Table 5.1 Number of data passes required to obtain the correct threshold.

Data Set / Algorithm	Twod.tra	Single2.tra	Oh7.tra	Fmtrain.tra	Mattrn.tra	Mean
Brute Force	9.58	9.13	9.70	9.43	9.98	9.56
Volumetric Estimate	11.50	11.30	15.63	8.93	10.68	11.61
Linear Interpolation	19.43	39.83	27.34	20.83	18.24	25.13
Quadratic Interpolation	19.93	46.60	36.45	21.26	17.34	28.32
Binary Search	8.91	8.69	9.00	8.52	8.55	8.73

Table 5.1 compares the performance of the threshold estimation techniques. The values in each column are the average number of passes computed over 10-fold validation data sets for that file. For each data set, the number of passes required to obtain $K_{desired}$=1 to $K_{desired}$=100 were averaged. It is observed that the binary search approach works best

across all data sets. Brute force technique surprisingly gives very good results, closely approaching the binary search method. The volumetric estimate, linear and quadratic interpolation fail as the initial estimate is too far away from the ideal threshold, requiring several brute force iterations at the end. Linear and quadratic interpolation consistently yielded negative thresholds for higher number of clusters, suggesting that both types of mappings did not approximate the relationship between T and K adequately.

5.5 Deleting Extra Clusters

One way to avoid repeatedly adjusting the threshold and passing through the data for obtaining exactly the desired number of clusters is to allow the algorithm to generate slightly more clusters than needed, and then deleting the extra clusters by a suitable method.

5.5.1 Eliminate Large Clusters

Here, we delete the $(K - K_{desired})$ extra clusters, by removing one largest cluster at a time. As the largest cluster is found and deleted, its patterns migrate to other clusters affecting their size. The largest cluster formed after this redistribution of patterns is selected as the next target for deletion. The process is repeated till $K_{desired}$ clusters are obtained.

Theoretically, eliminating a larger cluster can reduce the global error by a greater margin than deleting a small cluster. Since clustering does not take into consideration the mapping error, a highly correlated cluster could still have a poor mapping. Eliminating such clusters reduces the global mean square error.

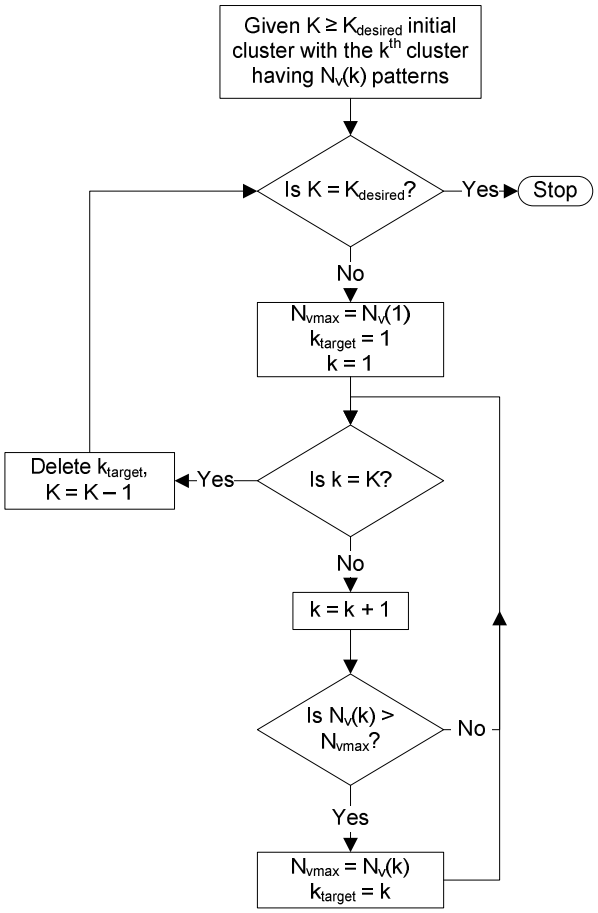

Figure 5.1 Deleting the largest sized extra clusters.

5.5.2 Eliminating Large Clusters Based on Initial Size

An alternate approach to deleting the largest clusters is to identify and list the largest clusters initially formed during clustering. Only the biggest ($K - K_{desired}$) clusters belonging to this pool are deleted, irrespective of whether patterns migrating to other clusters have made the other clusters bigger in size.

Given $K \geq K_{desired}$ initial clusters, select the indices of top ($K - K_{desired}$) clusters into the array $\mathbf{K}_{target}$ with the maximum number of clusters, without attempting to delete any clusters in the process. The clusters in $\mathbf{K}_{target}$ are deleted at the end.

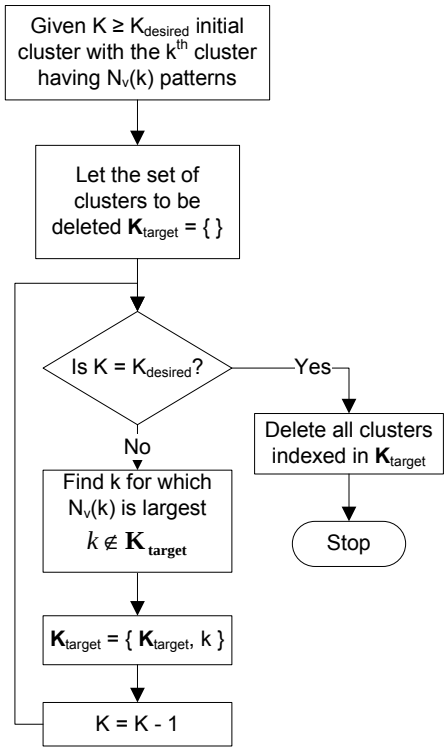

Figure 5.2 Deleting the largest sized extra clusters initially formed.

5.5.3 Eliminate Small Clusters

Another way to delete the ($K - K_{desired}$) extra clusters is by removing one smallest cluster at a time. As the smallest cluster is found and deleted, its patterns migrate to other clusters affecting their size. The cluster with the smallest size after this redistribution of patterns is selected as the next target for deletion. The process is repeated till $K_{desired}$ clusters are obtained.

Small clusters may contain very few patterns, thus contribute marginally to the global mapping error. If such clusters are deleted, there is only a small increase in the error of the cluster to which the patterns migrate to.

An outlier is a pattern sufficiently removed from the rest of the clusters to suspect that it was included by error. Outliers are usually introduced by noise or observation error.

Forcing an outlier to belong to a cluster distorts its shape. It is best to identify and remove such outliers [54].

Since outliers are produced by noise or error, they tend to occur sporadically and far out. Algorithms such as sequential leader will most certainly classify them as independent clusters, since their distances are out of the threshold. Deleting such small clusters helps in removing outliers and improving the clustering.

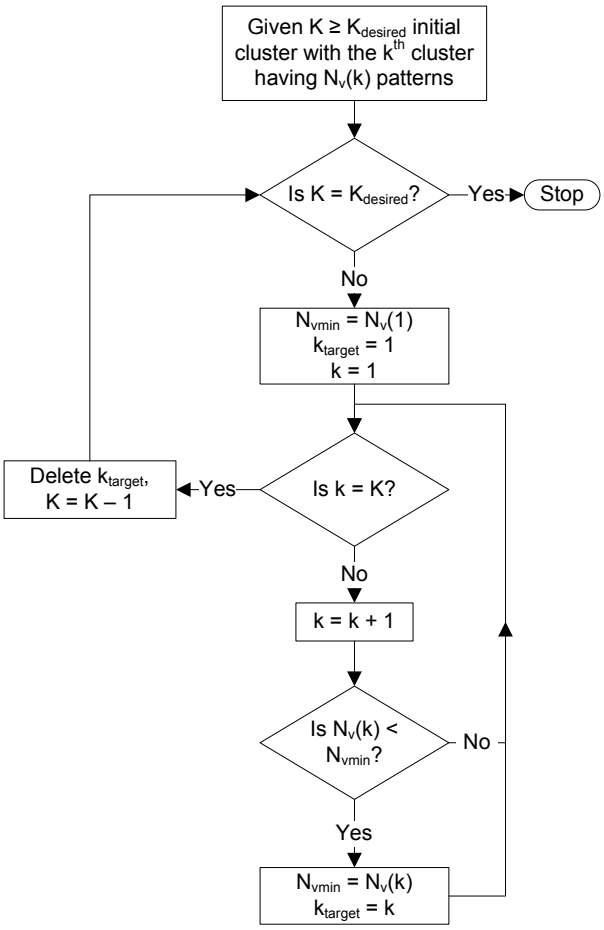

Figure 5.3 Deleting the smallest sized extra clusters.

5.5.4 Eliminating Small Clusters Based on Initial Size

Like in the case of deleting large clusters, there is also an alternative way of deleting the smallest clusters. The ($K - K_{desired}$) smallest clusters are identified and listed from

amongst the initial clusters generated. Then all clusters belonging to this pool are deleted, irrespective of whether other patterns migrating to clusters within this pool have made these clusters bigger in size than other clusters.

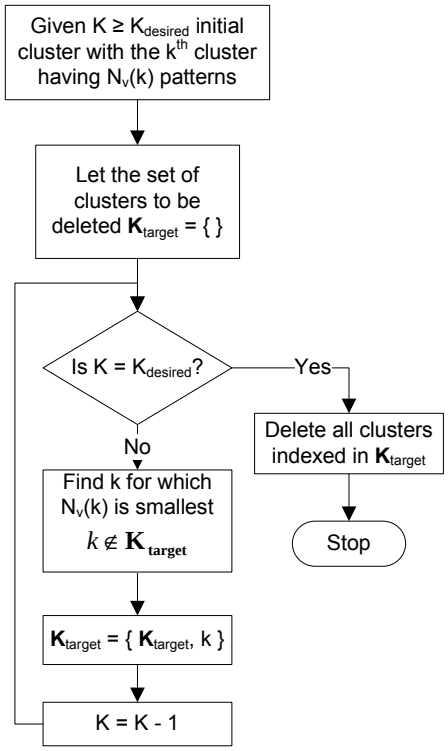

Figure 5.4 Deleting the smallest sized extra clusters initially formed.

5.5.5 Pruning the Clusters

Pruning in the simplest case guarantees the minimum increase in error upon deletion of one cluster. Thus it is very close to the ideal. But it cannot guarantee that the increase in error would still be the smallest over the deletion of several clusters. Table 5.2 compares the global mapping error when extra clusters were pruned with each of the techniques described in this section.

5.6 Comparing Performance of Different Cluster Elimination Techniques

To compare the performance of the different cluster deletion techniques described here, 10-fold validation using four data sets was performed. When SOM clustering was used, exactly $K = 1.1 \cdot K_{desired} + 50$ clusters were initially generated, and with sequential leader clustering, K between $K_{desired} + 5$ and $1.1 \cdot K_{desired} + 50$ clusters were initially generated. These were pruned down to $K_{desired}$ clusters using each of the described methods, and the average clustering error at this stage was noted over the 10 validation data sets.

Table 5.2 Initial error after deleting extra clusters using several techniques.

Data Set		Oh7.tra				Single2.tra			
K		20		50		20		50	
Clustering	Clusters selected for deletion	Training Error	Validation Error	Training Error	Validation Error	Training Error	Validation Error	Training Error	Validation Error
SOM	Group of largest generated clusters	2.022707	2.382119	1.585041	2.10325	0.031084	0.12371	0.016088	0.040618
	One largest cluster at a time	2.045838	2.428807	1.515710	2.049238	0.030970	0.124550	0.016340	0.064812
	Group of smallest generated clusters	2.062976	2.377584	1.730798	2.271263	0.032098	0.073335	0.022934	0.041664
	One smallest cluster at a time	2.102140	2.346228	1.800588	2.392646	0.047400	0.111698	0.020659	0.047319
	Pruning extra clusters	2.004188	2.322848	1.486143	1.988834	0.030980	0.051329	0.016882	0.037403
Sequential Leader	Group of largest generated clusters	1.785185	2.248760	1.463296	2.698309	0.037723	0.080616	0.021461	0.056046
	One largest cluster at a time	1.759999	2.466585	1.469269	5.116379	0.035101	0.675261	0.017456	0.066077
	Group of smallest generated clusters	1.806505	1.987828	1.468349	4.789066	0.033159	0.042774	0.017162	0.046756
	One smallest cluster at a time	1.930246	2.144635	1.467419	2.104229	0.033032	0.041232	0.016464	0.036162
	Pruning extra clusters	1.795159	2.027868	1.461367	5.346219	0.033261	0.042838	0.017573	0.041262

Table 5.2 – *Continued*

Data Set		Twod.tra				Mattrn.tra			
K		20		50		20		50	
Clustering	Deletion Method	Training Error	Validation Error	Training Error	Validation Error	Training Error	Validation Error	Training Error	Validation Error
SOM	Group of largest generated clusters	0.194424	0.257178	0.142301	0.713996	0.025468	0.029388	0.010854	0.018351
	One largest cluster at a time	0.194281	0.255438	0.143796	0.698954	0.028565	0.033629	0.016131	0.026015
	Group of smallest generated clusters	0.190405	0.247962	0.142212	0.366240	0.023202	0.029962	0.010661	0.017253
	One smallest cluster at a time	0.190265	0.245559	0.140269	0.316514	0.023469	0.028909	0.009932	0.016884
	Pruning extra clusters	0.188396	0.242877	0.141451	0.348277	0.018454	0.023697	0.007988	0.014705
Sequential Leader	Group of largest generated clusters	0.197268	0.567863	0.155876	2.887054	0.025855	0.031004	0.011043	0.019828
	One largest cluster at a time	0.206905	1.776535	0.166569	4.880402	0.023797	0.026813	0.012352	0.021264
	Group of smallest generated clusters	0.193487	1.509204	0.150356	2.295173	0.025035	0.029907	0.011388	0.019644
	One smallest cluster at a time	0.190865	0.257002	0.142951	4.475504	0.024480	0.029255	0.010729	0.017656
	Pruning extra clusters	0.191748	0.285933	0.150678	4.837354	0.021424	0.026131	0.008919	0.075440

CHAPTER 6

IMPROVED PLN TRAINING

6.1 Overhead in Conventional Pruning Methods

Training data files can be overwhelmingly large in size, and reading through the whole file can take up a significant amount of processing time even with a very fast hard disk drive (HDD). This is because processors have become increasingly fast, but mass storage technology has not been able to maintain the same pace. This results in the CPU waiting to read from or write to a storage device very often. In most computers and for most data files, the main memory or random access memory (RAM) is not large enough to hold the complete data file in its entirety. Thus, the file must be read back again from the disk for every pass. The conventional pruning method makes at least 2 passes through the whole data file for every pruning iteration. We have proposed a way to significantly reduce the amount of file access performed.

6.2 Storing Patterns of Each Cluster Separately

After the initial set of clusters has been obtained from a clustering algorithm, the patterns are read from the training data file, and written to a file corresponding to the cluster to which the pattern belongs to. Modern operating systems have no problems having a large number of files open at once, and can easily open files for hundreds of clusters. The files

are created in binary format to conserve space and reduce the number of bytes to be read or written.

6.2.1 Multiplies Saved in Computing Distance Measures

Once a cluster is selected for deletion, the original data file need not be traversed to seek out patterns belonging to the target cluster, since the patterns belonging to this cluster are already segregated. Thus, it is not necessary to compute $N_v \cdot K$ distances to determine cluster memberships for all the patterns.

When pruning from K clusters down to 2 clusters, the number of multiplies needed to compute additional distance measures when using a single file, which are not needed when using multiple files are:

$$M_{saved} = N_v \cdot M_{DM} \cdot \sum_{k=3}^{K} k \tag{6.1}$$

where M_{DM} is the number of multiplies required to compute one distance measure. For the weighted Euclidian distance measure, where $M_{DM} = 2 \cdot N$,

$$M_{saved} = 2 \cdot N_v \cdot N \cdot \sum_{k=3}^{K} k \tag{6.2}$$

$$M_{saved} = 2 \cdot N_v \cdot N \cdot \left(\frac{K \cdot (K+1)}{2} - 3 \right) \tag{6.3}$$

6.2.2 File I/O Operations Saved

File I/O operations are a considerable bottleneck in modern day program execution speeds. The number of file reads and writes required when pruning from K clusters down to 2 clusters is:

$$N_{IO}(single\ file) = \sum_{k=3}^{K} N_v = (K-2) \cdot N_v \tag{6.4}$$

$$N_{IO}(multiple\ files) = \sum_{k=3}^{K} 2 \cdot N_v(k) \quad (6.5)$$

Since the target cluster k_{target} generally has very few patterns $N_v(k_{target})$ compared to the original data file N_v, this results in considerable time savings.

If the complexity of one I/O operation is equivalent to M_{IO} multiplies, the total savings in the number of multiplies is:

$$M_{saved} = 2 \cdot N_v \cdot N \cdot \left(\frac{K \cdot (K+1)}{2} - 3 \right) + (K-2) \cdot N_v \cdot M_{IO} - \sum_{k=3}^{K} 2 \cdot N_v(k) \cdot M_{IO} \quad (6.6)$$

Table 6.1 presents some results obtained on some test files for multiplies and I/O operations saved, computed by using these formulas. The results shown are for pruning from 50 clusters down to a single cluster.

Table 6.1 Savings due to use of multiple files.

Data Set	N_v	N	Number of multiplies			Number of IO operations		
			Using a Single file	Using Multiple Files	Percentage Savings	Using a Single file	Using Multiple Files	Percentage Savings
twod	1768	8	35982336	1414400	96.07	84864	7736	90.88
single2	10000	16	407040000	16000000	96.07	480000	15858	96.70
oh7	15000	20	763200000	30000000	96.07	720000	44082	93.88
fmtrain	1024	5	13025280	512000	96.07	49152	4678	90.48
mattrn	2000	4	20352000	800000	96.07	96000	9444	90.16
power12	1414	12	43166592	1696800	96.07	67872	5814	91.43

Actual time savings in seconds are illustrated in Table 6.2. The results shown are for pruning from 50 clusters down to a single cluster. To increase timing accuracy, the values have been averaged across 3 trials on 10-fold validation data sets.

Table 6.2 Comparison of time taken using separate files for each cluster to working with a single file.

Data Set	Execution Time using a Single File	Execution Time using Separate Files	Time Saved	Percentage Time Savings
twod	3.9779	3.4009	0.5769	14.50
single2	14.5581	10.7003	3.8578	26.50
oh7	28.9881	20.8993	8.0888	27.90
fmtrain	1.3229	1.1038	0.2191	16.56
mattrn	2.3954	1.9230	0.4723	19.72
power12	3.0986	2.3475	0.7510	24.24

6.3 Redesign Only the Changed Clusters

When a cluster is deleted, patterns migrate to other clusters. It is likely that the patterns from the deleted cluster will be absorbed by the clusters surrounding it, and no patterns will be added to many clusters that are at a distance. When redesigning the PLN at this stage, only the clusters which received new training patterns need to be redesigned. This leads to considerable time savings compared to redesigning all modules of the PLN. In the case of the PLN, a linear fitting algorithm using the Schmidt procedure is applied. If the modules are to have quadratic or higher order mappings, the time savings increase. In some cases, unnecessary cluster fitting was avoided 90% of the time. The previous optimization of using multiple files was retained when implementing this optimization.

A flag variable can be set up for the clusters to which new patterns have been added. This information can be later used to decide if the cluster needs to be redesigned or not. Table 6.3 shows the percentage of time clusters were unmodified and did not need to be redesigned. The results shown are for pruning from 50 clusters down to a single cluster.

Table 6.3 Percentage of times call to OLS was avoided
for unmodified clusters.

Data Set	Percentage of times call to OLS was avoided
twod	90.39
single2	88.06
oh7	87.88
fmtrain	92.39
mattrn	83.33
power12	92.66

The number of times calls to OLS were avoided is quite significant, ranging above eighty percent. But due to fast speeds of today's processors, this leads to only a modest reduction in execution time. Table 6.4 shows the time savings in seconds for pruning from 50 clusters down to a single cluster.

Table 6.4 Time saved by avoiding calls to OLS for unmodified clusters.

Dataset	Execution Time when Redesigning All Clusters	Execution Time when Redesigning Only Changed Clusters	Time Saved	Percentage Time Savings
twod	3.977857	3.921943	0.055914	1.41
single2	14.558098	14.355004	0.203094	1.40
oh7	28.988145	28.600798	0.387347	1.34
fmtrain	1.322890	1.298520	0.024371	1.84
mattrn	2.395359	2.348174	0.047185	1.97
power12	3.098569	2.964425	0.134145	4.33

IO operations can take up a significant amount of the total execution time, which make the improvements in execution speed less noticeable. The scheduler often tries to execute interleaving IO and processing operations in parallel, which can introduce some variability in results. To work around these problems, a separate version of the pruning

software was developed which did not work with the original data file, but instead worked on a copy stored in the main memory thus avoiding most IO operations. Table 6.5 shows execution times in seconds for this in-memory implementation when pruning from 50 clusters down to a single cluster.

Table 6.5 Time saved by avoiding calls to OLS for unmodified clusters with the in-memory implementation.

Dataset	Execution Time when Redesigning All Clusters for in-memory implementation	Execution Time when Redesigning Only Changed Clusters for in-memory implementation	Time Saved	Percentage Time Savings
twod	1.000563	0.960174	0.040389	4.04
single2	4.752369	4.634026	0.118343	2.49
oh7	9.938447	9.750855	0.187592	1.89
fmtrain	0.336549	0.329262	0.007287	2.17
mattrn	0.655931	0.641581	0.014350	2.19
power12	0.850312	0.790640	0.059672	7.02

6.4 Reassign Patterns and Change R & C

As patterns from the deleted cluster are added to other clusters, the autocorrelation matrices $\mathbf{R}_k$, and the cross correlation matrices $\mathbf{C}_k$ of the clusters receiving the patterns are affected. Instead of recalculating the matrices by parsing the whole data file, the **R** and **C** matrices are updated incrementally as soon as a new pattern is added. To facilitate this, the **R** & **C** matrices are not normalized with respect to the number of patterns in the cluster. This does not affect the computation of weights, as the linear equation solver does not depend on the scaling of the matrices. This eliminates the need to go through all the patterns in the cluster to obtain the **R** and **C** matrices. The previous optimizations of

using multiple files and redesigning only the changed clusters still remain when implementing this optimization.

6.5 Partial Distances

Partial distances have been used in vector quantization and video coding algorithms for improving speeds [62, 63]. The concept is that the decision whether a pattern does not belongs to a cluster can be made before the squared distance along all the dimensions has been added to the norm. If for any $n < N$, the partial distance is greater than the minimum distance yet found in the search, the next cluster can be considered. This certainly reduces the number of multiplies required to determine cluster memberships. The algorithm is described in Figure 6.1.

Partial distance measures have been used to improve the network design and input processing speeds in the following ways:

1. When training the PLN, patterns need to be clustered. This requires determining the closest cluster to each pattern.
2. The pruning algorithm needs the closest two clusters to each pattern. A modified form of partial distance (Figure 6.2) is used to determine both in a single pass.
3. In pruning, when a cluster is selected for deletion, its patterns are reassigned to other clusters.
4. When processing input patterns with a PLN, the first step is to determine the cluster to which the pattern belongs.

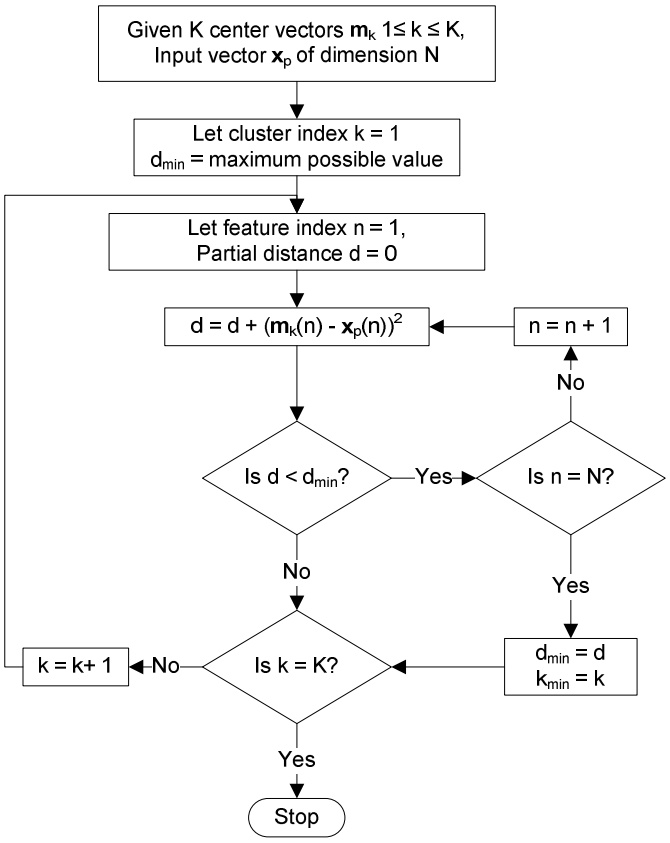

Figure 6.1 Using partial distance to find nearest cluster.

Use of partial distances helped with reducing the amount of computation required for distance measurement. Table 6.6 shows the reduction in computation with the use of partial distance when pruning from 50 clusters down to a single cluster. The optimizations of using multiple files, redesigning only the changed clusters, and incrementally updating R and C matrices were retained.

Table 6.7 shows the execution time savings in seconds when using partial distances for pruning from 50 clusters down to a single cluster. Using partial distance reduced execution time in most cases, except where the number of inputs was already very small. The savings in time become more pronounced when using the in-memory implementation as seen in Table 6.8.

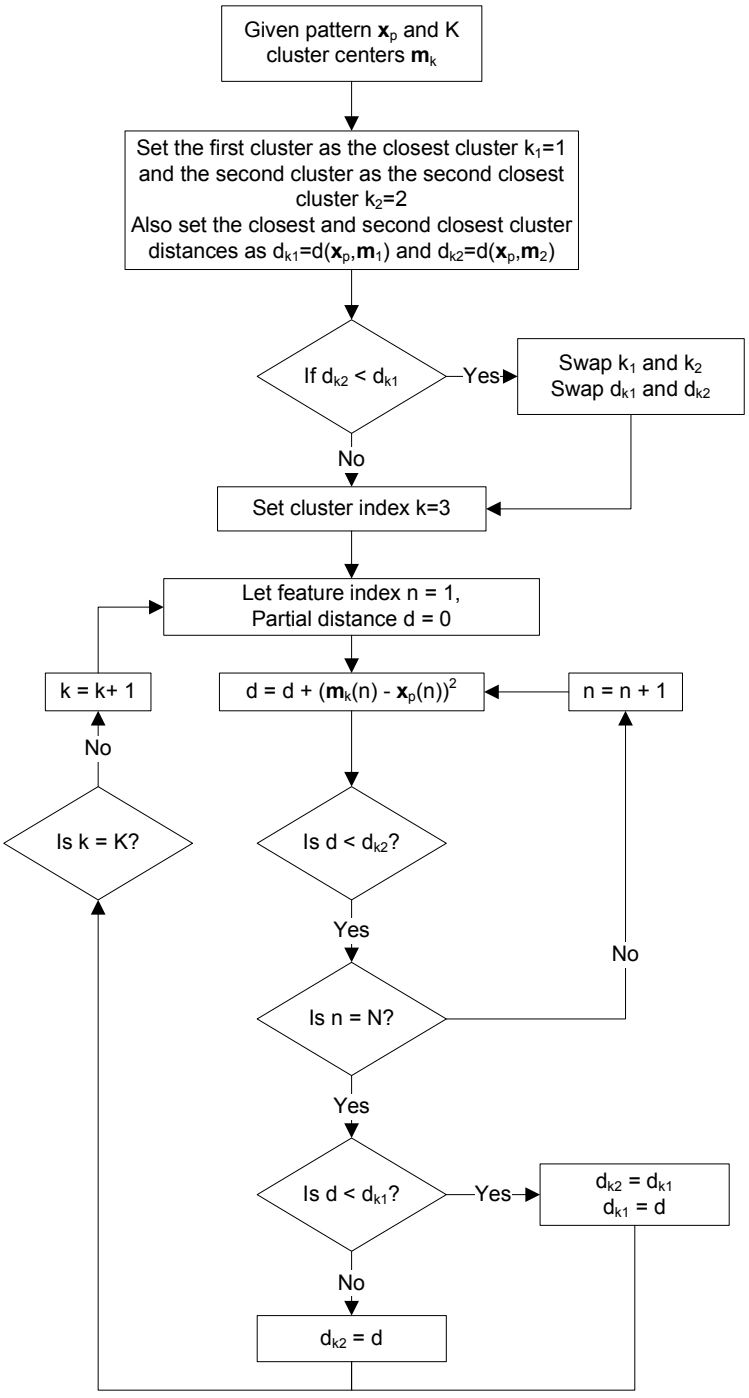

Figure 6.2 Modified partial distance used in pruning.

Table 6.6 Reduction in computation with partial distance measure.

Data Set	Original Feature Length	Average Feature Length With Partial Distances	Percentage of multiplies avoided by using Partial Distances
twod	8	2.8994	63.76
single2	16	6.2643	60.85
oh7	20	11.3243	43.38
fmtrain	5	2.7565	44.87
mattrn	4	2.3470	41.33
power12	12	4.4931	62.56

Table 6.7 Execution time saved using partial distance.

Data Set	Number of Inputs	Execution Time Without Using Partial Distances	Execution Time Using Partial Distances	Time Saved	Percentage Savings
twod	8	3.400945	2.982096	0.418849	12.32
single2	16	10.700347	9.145168	1.555180	14.53
oh7	20	20.899322	19.823897	1.075425	5.15
fmtrain	5	1.103798	1.115989	-0.012191	-1.10
mattrn	4	1.923028	1.924957	-0.001930	-0.10
power12	12	2.347528	2.082255	0.265273	11.30

Table 6.8 Execution time saved using partial distance for in-memory implementation.

Data Set	Number of Inputs	Execution Time Without Using Partial Distances	Execution Time Using Partial Distances	Time Saved	Percentage Savings
twod	8	0.960174	0.823122	0.137052	14.27
single2	16	4.634026	3.258940	1.375086	29.67
oh7	20	9.750855	8.678893	1.071962	10.99
fmtrain	5	0.329262	0.330343	-0.001081	-0.33
mattrn	4	0.641581	0.673092	-0.031511	-4.91
power12	12	0.790640	0.666757	0.123884	15.67

6.6 Partial Distances with an Order Function

The dimensions with a larger variance are expected to create larger increments to the partial distance, causing distant clusters to be rejected more quickly. An order function maps the natural order of inputs in the data file to a different sequence. Distance components were summed up in this order in the partial distance, and when the sum exceeded the minimum distance observed yet, further computation of the distance measure was aborted.

An ordering function *o(n)* was determined by sorting the input variances in decreasing order.

$$o(1) = \arg\min_{1 \le i \le N} \left[\frac{1}{\sigma(i)^2} \right]$$
$$o(n) = \arg\min_{1 \le i \le N,\, i \notin \{o(1), o(2) \ldots o(n-1)\}} \left[\frac{1}{\sigma(i)^2} \right] \tag{6.7}$$

A second order function was also used which also accounted the variances of cluster centers. A variance measure *f(n)* was defined as:

$$f(n) = \sigma^2(n) + \sum_{k=1}^{K-1} \sum_{l=k+1}^{K} \left(m_k(n) - m_l(n) \right)^2 \tag{6.8}$$

The order function in this case was determined by sorting *f(n)* in decreasing order:

$$o(1) = \arg\min_{1 \le i \le N} \{ f(i) \}$$
$$o(n) = \arg\min_{1 \le i \le N,\, i \notin \{o(1), o(2) \ldots o(n-1)\}} \{ f(i) \} \tag{6.9}$$

Table 6.9 shows the reduction in computation with the use of both the order functions. Using an order function further reduced the computation in most cases over using only partial distance. In the few cases where the use of an order function increased the amount of computation needed, it can be inferred that the order function used was not the ideal

one for the scenario, and the natural ordering of inputs in the data file was better. The first order function based on input variance performed better than the one given in (6.9).

Table 6.9 Reduction in computation with partial distance measure and an order function.

Data Set	Original Feature Length	Average Feature Length With Partial Distances Only	Order Function 1		Order function 2	
			Average Feature Length With Partial Distance and Order Function	Percentage of multiplies avoided by using Order Function	Average Feature Length With Partial Distance and Order Function	Percentage of multiplies avoided by using Order Function
twod	8	2.8994	2.8860	0.17	4.08202	-14.78
single2	16	6.2643	7.5146	-7.81	5.237872	6.42
oh7	20	11.3243	9.4453	9.39	10.44815	4.38
fmtrain	5	2.7565	2.6056	3.02	2.84444	-1.76
mattrn	4	2.3470	2.1948	3.80	2.428342	-2.03
power12	12	4.4931	4.7576	-2.20	5.567546	-8.95

The savings in execution time for the first order function (decreasing order of variance) are reported in Table 6.10. All previous optimizations, namely, using multiple files, redesigning only the changed clusters, incrementally updating R and C matrices, and using partial distances were retained when implementing the order function.

Table 6.10 Comparison of execution times when using partial distances and an order function.

Data Set	Number of Inputs	Execution Time Using Partial Distance	Execution Time Using Partial Distance and Order Function	Time Saved	Percentage Savings
twod	8	0.823122	0.819107	0.004015	0.49
single2	16	3.258940	3.457075	-0.198135	-6.08
oh7	20	8.678893	7.859646	0.819247	9.44
fmtrain	5	0.330343	0.310786	0.019557	5.92
mattrn	4	0.673092	0.614167	0.058925	8.75
power12	12	0.666757	0.640978	0.025779	3.87

CHAPTER 7

SIMULATIONS

In this chapter, results from pruning PLNs using the improved algorithm of CHAPTER 6 are presented. The algorithm was implemented in Microsoft Visual C++ compiler version 6.0.

7.1 Pruning Simulations

The improved PLN was tested on 6 data sets, pruning down from 50 clusters to 1 cluster. The clustering algorithm used is SOM. Training and validation errors were averaged over 10-fold validation data sets created from the original data files. For more information about the data files, please refer to the appendix.

7.1.1 Twod Data Set

The monotonic increase in training error when reducing the number of clusters proves the effectiveness of the pruning algorithm. But the validation error shows that the initial number of clusters was too big, and caused memorization of patterns. Thus the network could not respond well to the unseen validation data. The lowest validation error of 0.2434 was seen for K=14 clusters, though any number of clusters between K=5 and K=27 would have yielded a similar error. The sharp increase in the validation error when

going from K=47 to K=46 clusters shows the deletion of a useful cluster. Since the pruning algorithm aims at reducing the total MSE, a bigger cluster with a good mapping may get more preference over smaller cluster with poor mapping for deletion. Similarly, the fall in error when going from K=37 to K=36 clusters could be due to the deletion of a poor cluster that had been memorized, relinquishing its patterns to other clusters where they fit the mapping well.

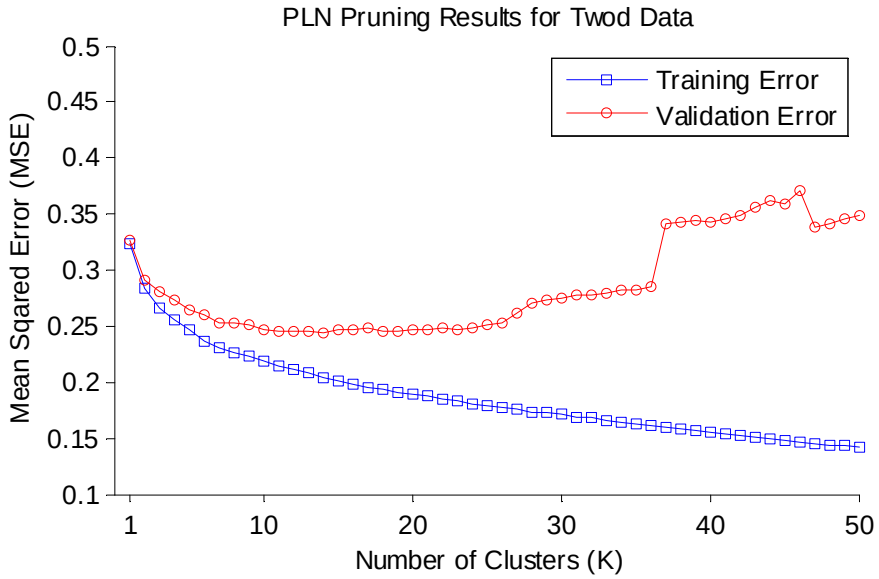

Figure 7.1 PLN pruning results for twod data.

7.2 Single2 Data Set

This data file was mapped very well by the piecewise linear network. The lowest validation error of 0.0363 was noted for K=44 clusters, and went up to only 0.0426 if half the number of clusters were used.

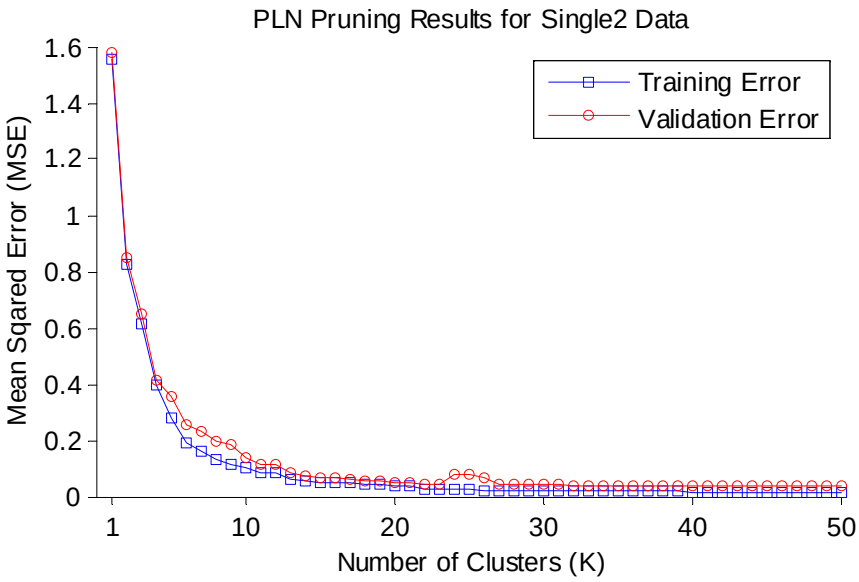

Figure 7.2 PLN pruning results for single2 data.

7.2.1 Oh7 Data Set

Here, a good number of clusters for the PLN comes out to be K=20. Upon increasing the number of clusters beyond 20, the validation error does not decrease, showing that the clusters have already mapped the function adequately well.

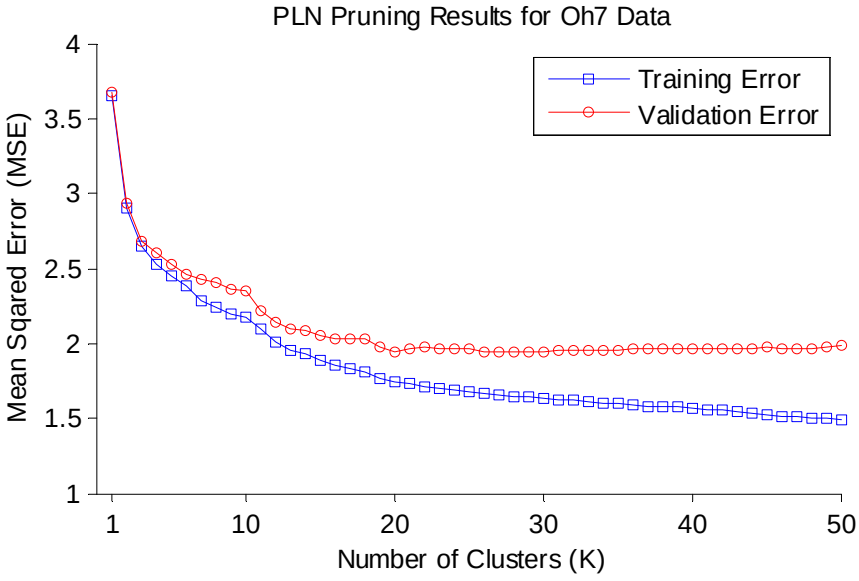

Figure 7.3 PLN pruning results for oh7 data.

7.2.2 Fmtrain Data Set

This data file contains synthetic data generated from a mathematical relation having five inputs and one output. The relatively low dimensionality and simplicity of the mapping helps in obtaining good results with a very small number of clusters.

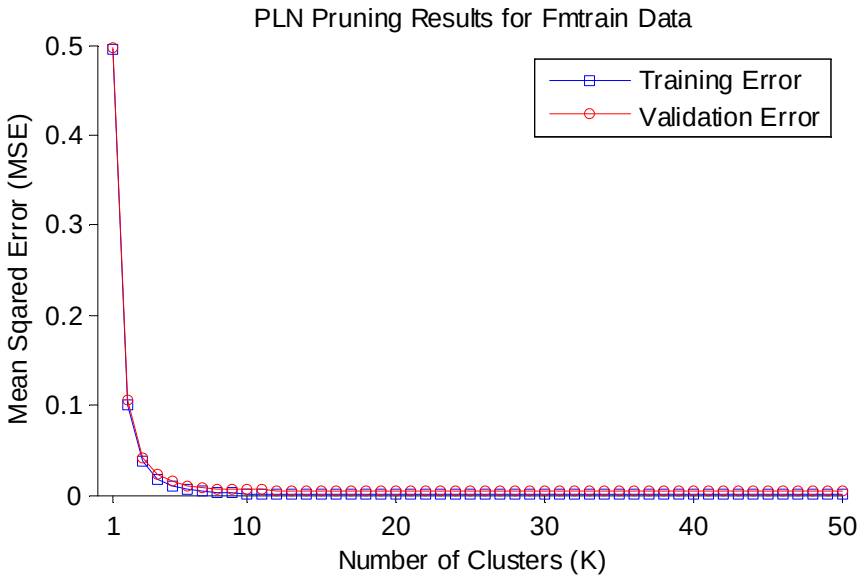

Figure 7.4 PLN pruning results for fmtrain data.

7.2.3 Mattrn Data Set

This data set contains data for inversion of 4x4 matrices. The input matrices are restricted to uniform distribution between 0 and 1, suggesting that the inverted matrices are likely to have values with magnitude more than 1. Even with a single cluster, the mean squared error is restricted to 0.2, which is relatively small. As the number of clusters is increased, both the training and validation errors go down almost monotonically, although there is not much difference in the error for K=20 and for K=50 clusters.

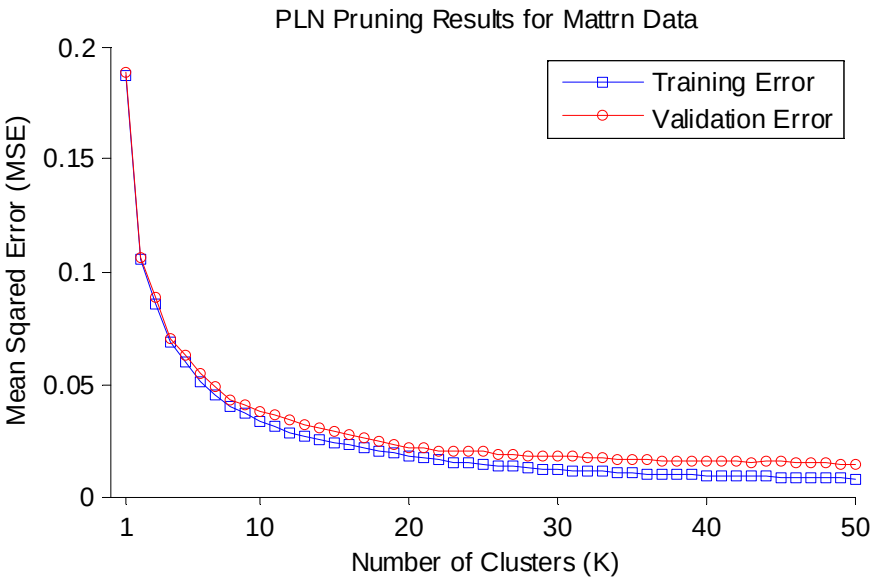

Figure 7.5 PLN pruning results for mattrn data.

7.2.4 Power12 Data Set

This file contains power load forecasting data, having twelve input parameters and one predicted output. For this file, the validation error seems to go up as the number of clusters increase. This can be attributed to poor data for approximation, or a poor choice of distance measure that yields clusters with a poor mapping.

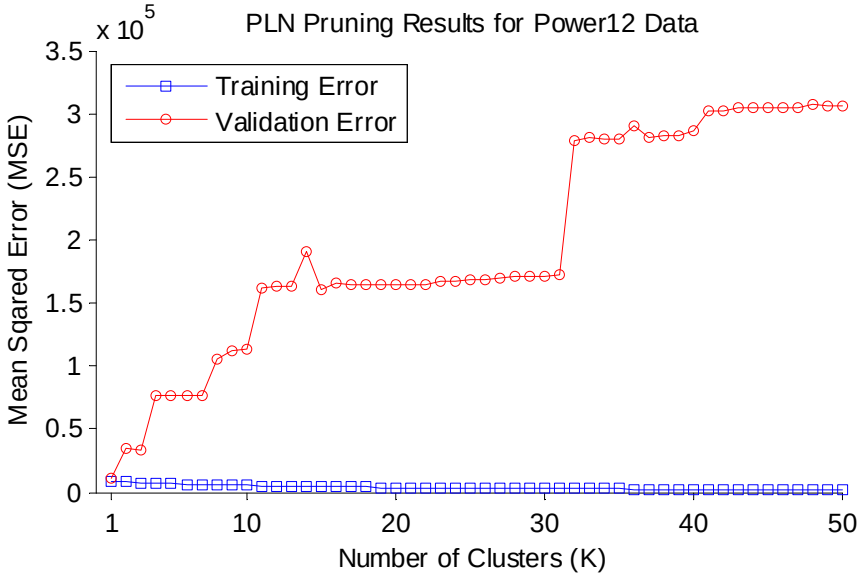

Figure 7.6 PLN pruning results for power12 data.

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

8.1 Conclusions

In this thesis, several prospects for improving a piecewise linear network are explored. SOM clustering is found to perform better than sequential leader clustering when compared by the initial error of the network. Several methods to estimate the threshold for SL were developed, and a binary search based approach was found to be the most efficient. To deal with the problem of SOM clustering yielding empty clusters and SL clustering failing to be precise with the number of clusters, it was suggested to generate a higher number of clusters than desired. Several methods to delete the extra clusters were considered, and pruning – although slower, was expectedly found to be the best, closely followed by the deletion of the smallest clusters. Several methods to improve PLN pruning performance are discussed and their effectiveness demonstrated. It is found that separating patterns to different files based on their cluster membership can be used to improve performance. The gains in execution speed were more noticeable on larger data files because the number of file IO operations, which are the biggest speed bottlenecks, saved also increased proportionally. The Schmidt procedure was implemented for solving linear network weights, instead of using conjugate gradient. Significant reduction in computation was observed during pruning when only clusters that received new patterns

during the process were re-fitted. Partial distance measure based on an ordering function was also found to reduce the amount of computation.

8.2 Future Work

Given the drawbacks of SOM clustering – namely dependence on the initial set of cluster center vectors provided, and slower performance as compared to Sequential Leader clustering, it necessary to find better applications of Sequential Leader clustering to PLN problems. The performance skew in favor of SOM clustering can be attributed to the fact that it begins with a significantly larger number of clusters than SL clustering, pruning down across a larger number of clusters to get to the desired number of clusters. Since it is difficult to control precisely the number of clusters generated by SL algorithm, it is made to generate any number of clusters between a window having boundaries slightly over the desired number of clusters and less than the maximum number of clusters. Because the number of clusters initially generated with SL is almost always less than the initial number of clusters generated in SOM, it is difficult to make a precise comparison of the quality of generated clusters. Thus it is suggested that a thorough comparison be made by forcing both SL and SOM to yield the same number of initial clusters. This may require several brute force attempts on the SL threshold.

APPENDIX A

DESCRIPTION OF DATA SETS USED FOR TRAINING AND VALIDATION

Twod - Inversion of surface scattering parameters

This training file is used in the task of inverting the surface scattering parameters from an inhomogeneous layer above a homogeneous half space, where both interfaces are randomly rough. The parameters to be inverted are the effective permittivity of the surface, the normalized rms height, the normalized surface correlation length, the optical depth, and single scattering albedo of an inhomogeneous irregular layer above a homogeneous half space from back scattering measurements.

The training data file contains 1,768 patterns. The inputs consist of eight theoretical values of back scattering coefficient parameters at V and H polarization and four incident angles. The outputs were the corresponding values of permittivity, upper surface height, lower surface height, normalized upper surface correlation length, normalized lower surface correlation length, optical depth and single scattering albedo which had a joint uniform PDF [64, 65].

Single2 – Inversion of back scattering parameters

This training data file consists of 16 inputs, 3 outputs and 10,000 training patterns, and represents the training set for inversion of surface permittivity, the normalized surface rms roughness, and the surface correlation length found in back scattering models from randomly rough dielectric surfaces. The first 16 inputs represent the simulated back scattering coefficient measured at 10, 30, 50 and 70 degrees at both vertical and horizontal polarization. The remaining 8 are various combinations of ratios of the original eight values. These ratios correspond to those used in several empirical retrieval algorithms [66, 67].

Oh7 - Radar Scattering from Bare Soil Surfaces

This data set is given in [68]. The training set contains VV and HH polarization at L 30, 40 deg, C 10, 30, 40, 50, 60 deg, and X 30, 40, 50 deg along with the corresponding unknowns rms surface height, surface correlation length, and volumetric soil moisture content in g / cubic cm. The file has 20 inputs, 3 outputs and 10,453 training patterns.

Fmtrain – FM demodulation data

This training file is used to train a neural network to perform demodulation of an FM (frequency modulation) signal containing a sinusoidal message. The data are generated from the equation

$$r(n) = A_c \cos\left[(2\pi f_c n) + A_m \sin(2\pi f_m n)\right]$$

where A_c = Carrier Amplitude, A_m = Message Amplitude, f_c = normalized Carrier frequency, f_m = normalized message frequency. In this data set, A_c = .5, f_c = .1012878, A_m = 5, and f_m = .01106328. The five inputs are r(n-2), r(n-1), r(n), r(n+1), and r(n+2). The output is the sinusoidal message $\cos(2\pi f_m n)$. In each consecutive pattern, n is incremented by 1 [69]. The file has 2000 training patterns.

Mattrn – Matrix inversion data

This training file provides the data set for inversion of random two-by-two matrices. Each pattern consists of 4 input features and 4 output features. The input features, which are uniformly distributed between 0 and 1, represent a matrix and the four output features are elements of the corresponding inverse matrix. The determinants of the input matrices are constrained to be between .3 and 2. the file has 2,000 training patterns.

Power12 – Power load forecasting

This training file was generated using data obtained from TU Electric Company in Texas. The file has 12 inputs, 1 output, and 1,414 training patterns. The first ten input features are last ten minutes power load in megawatts for the entire TU Electric utility, which covers a large part of north Texas. The output is power load fifteen minutes in the future from the current time. All powers were originally sampled every fraction of a second, and averaged over 1 minute to reduce noise. The last two inputs are respectively, the "True Area Control Error" (TACE) and the "Filtered Area Control Error" (FACE). The FACE is a combination of exponentially filtered TACE and moving average filtered TACE [70, 71].

REFERENCES

[1] S. Haykin. (1994, *Neural Networks a Comprehensive Foundation.*

[2] H. White. Economic prediction using neural networks: The case of IBM daily stock returns. Presented at Proceedings of the IEEE International Conference on Neural Networks.

[3] T. Brotherton and T. Johnson. Anomaly detection for advanced military aircraft using neural networks. Presented at Proceedings of 2001 IEEE Aerospace Conference.

[4] M. W. Craven and J. W. Shavlik. (1997, Using neural networks for data mining. *FGCS.Future Generations Computer Systems 13(2-3),* pp. 211-229.

[5] H. Lu, R. Setiono and H. Liu. (1996, Effective data mining using neural networks. *IEEE Trans. Knowled. Data Eng. 8(6),* pp. 957-961.

[6] O. Nerrand, P. Roussel-Ragot, L. Personnaz, G. Dreyfus and S. Marcos. (1993, Neural networks and nonlinear adaptive filtering: Unifying concepts and new algorithms. *Neural Comput. 5(2),* pp. 165-199.

[7] F. L. Lewis, S. Jagannathan and A. Yeşildirek. (1998, *Neural Network Control of Robot Manipulators and Nonlinear Systems.*

[8] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano and K. J. Lang. (1989, Phoneme recognition using time-delay neural networks. *Readings in Speech Recognition* pp. 393–404.

[9] I. Guyon. (1991, Applications of neural networks to character recognition. *INT.J.PATTERN RECOG.ARTIF.INTELL. 5(1),* pp. 353-382.

[10] C. L. Wilson, G. T. Candela and C. I. Watson. (1994, Neural network fingerprint classification. *J. Artif. Neural Networks 1(2),* pp. 203-228.

[11] S. Lawrence, C. L. Giles, A. C. Tsoi and A. D. Back. (1997, Face recognition: A convolutional neural-network approach. *IEEE Trans. Neural Networks 8(1),* pp. 98-113.

[12] S. Chen and S. A. Billings. Neural networks for non-linear dynamic system modelling and identification. *Advances in Intelligent Control*

[13] T. Y. Kwok and D. Y. Yeung. (1997, Constructive algorithms for structure learning in feedforward neural networks for regression problems. *IEEE Trans. Neural Networks*

[14] R. D. Beer and J. C. Gallagher. (1992, Evolving dynamical neural networks for adaptive behavior. *Adapt. Behav. 1(1),* pp. 91.

[15] M. Weintraub, F. Beaufays, Z. Rivlin, Y. Konig and A. Stolcke. Neural-network based measures of confidence for word recognition. Presented at IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS SPEECH AND SIGNAL PROCESSING.

[16] H. A. Rowley, S. Baluja and T. Kanade. (1998, Neural network-based face detection. *IEEE Trans. Pattern Anal. Mach. Intell. 20(1),* pp. 23-38.

[17] T. Robinson. (1994, An application of recurrent nets to phone probability estimation. *To Appear in IEEE Transactions on Neural Networks 5(3),*

[18] C. H. Sequin and R. D. Clay. Fault tolerance in artificial neural networks. Presented at Neural Networks, 1990., 1990 IJCNN International Joint Conference on.

[19] H. P. Graf, L. D. Jackel, R. E. Howard, B. Straughn, J. S. Denker, W. Hubbard, D. M. Tennant and D. Schwartz. VLSI implementation of a neural network memory with several hundreds of neurons. Presented at AIP Conference Proceedings.

[20] E. A. Vittoz. Analog VLSI implementation of neural networks. Presented at IEEE International Symposium on Circuits and Systems, 1990.

[21] S. Kühn. (2006, *Simulation of Mental Models with Recurrent Neural Networks*

[22] T. M. Cover. (1965, Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers 14(3),* pp. 326–334.

[23] H. N. Mhaskar. (2004, When is approximation by gaussian networks necessarily a linear process? *Neural Networks 17(7),* pp. 989-1001.

[24] P. Niyogi and F. Girosi. (1996, On the relationship between generalization error, hypothesis complexity, and sample complexity for radial basis functions. *Neural Comput. 8(4),* pp. 819-842.

[25] K. Lang and G. Hinton. (1988, The development of the time-delay neural network architecture for speech recognition. *Dep.Comput.Sci., Carnegie Mellon Univ., Tech.Rep.CMU-CS-88-152*

[26] L. R. Rabiner, S. E. Levinson, A. E. Rosenberg and J. G. Wilpon. (1990, Speaker-independent recognition of isolated words using clustering techniques. *Readings in Speech Recognition, CA* pp. 166-179.

[27] M. Attik, L. Bougrain and F. Alexandre. (2005, Neural network topology optimization. *Lecture Notes in Computer Science 3697*pp. 53.

[28] W. C. Carpenter and J. F. Barthelemy. (1994, Common misconceptions about neural networks as approximators. *J. Comput. Civ. Eng. 8(3),* pp. 345-358.

[29] A. A. Abdurrab, M. T. Manry, J. Li, S. S. Malalur and R. G. Gore. A piecewise linear network classifier. Presented at International Joint Conference on Neural Networks.

[30] T. Cover and P. Hart. (1967, Nearest neighbor pattern classification. *IEEE Trans. Inf. Theory 13(1),* pp. 21-27.

[31] S. Subbarayan, K. K. Kim, M. T. Manry, V. Devarajan and H. H. Chen. Modular neural network architecture using piece-wise linear mapping. Presented at Signals, Systems and Computers, 1996. 1996 Conference Record of the Thirtieth Asilomar Conference on.

[32] J. N. Lin and R. Unbehauen. (1995, Canonical piecewise-linear networks. *IEEE Trans. Neural Networks 6(1),* pp. 43-50.

[33] D. R. Hush and B. Horne. (1998, Efficient algorithms for function approximation with piecewise linear sigmoidal networks. *IEEE Trans. Neural Networks 9(6),* pp. 1129-1141.

[34] D. M. W. Leenaerts and W. M. G. van Bokhoven. (1998, *Piecewise Linear Modeling and Analysis.*

[35] H. Chandrasekaran, J. Li, W. H. Delashmit, P. L. Narasimha, C. Yu and M. T. Manry. (2007, Convergent design of piecewise linear neural networks. *Neurocomputing 70(4-6),* pp. 1022-1039.

[36] M. R. Hestenes and E. Stiefel. (1952, Methods of conjugate gradients for solving linear systems. *J*

[37] J. W. Dettman. (1988, *Mathematical Methods in Physics and Engineering.*

[38] J. Li, M. T. Manry, P. L. Narasimha and C. Yu. (2006, Feature selection using a piecewise linear network. *IEEE Trans. Neural Networks 17(5),* pp. 1101.

[39] R. Bellman. (1957, Dynamic programming, princeton. *NJ: Princeton UP*

[40] G. W. Milligan and M. C. Cooper. (1985, An examination of procedures for determining the number of clusters in a data set. *Psychometrika 50(2),* pp. 159-179.

[41] S. Bandyopadhyay and U. Maulik. (2002, An evolutionary technique based on K-means algorithm for optimal clustering in RN. *Inf. Sci. 146(1-4),* pp. 221-237.

[42] S. Z. Selim and M. A. Ismail. (1984, K-means-type algorithms: A generalized convergence theorem and characterization of local optimality. *IEEE Trans. Pattern Anal. Mach. Intell. 6(1),* pp. 81-87.

[43] J. Hardin and D. M. Rocke. (2004, Outlier detection in the multiple cluster setting using the minimum covariance determinant estimator. *Computational Statistics and Data Analysis 44(4),* pp. 625-638.

[44] E. F. Krause. (1986, *Taxicab Geometry: An Adventure in Non-Euclidean Geometry.*

[45] R. Xu and D. Wunsch. (2005, Survey of clustering algorithms. *IEEE Trans. Neural Networks 16(3),* pp. 645-678.

[46] S. Aksoy, R. Haralick, F. Cheikh and M. Gabbouj. A weighted distance approach to relevance feedback. Presented at International Conference on Pattern Recognition.

[47] P. C. Mahalanobis. On the generalized distance in statistics. Presented at Proceedings of the National Institute of Science, Calcutta.

[48] F. J. Maldonado and M. T. Manry. Optimal pruning of feedforward neural networks based upon the schmidt procedure. Presented at ASILOMAR CONFERENCE ON SIGNALS SYSTEMS AND COMPUTERS.

[49] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. Presented at International Joint Conference on Artificial Intelligence.

[50] T. E. Stern. (1956, Piecewise-linear network theory.

[51] L. Breiman, J. H. Friedman, R. A. Olshen and C. J. Stone. (1984, Classification and regression trees. *Wadsworth, Belmont, CA*

[52] J. H. Friedman. (1991, Multivariate adaptive regression splines. *The Annals of Statistics 19(1),* pp. 1-67.

[53] V. N. Vapnik. (1999, An overview of statistical learning theory. *IEEE Trans. Neural Networks 10(5),* pp. 988-999.

[54] A. Jain and R. Dubes. (1988, Algorithms for clustering data, prentice hall. *Englewood Cliffs*

[55] A. K. Jain, M. N. Murty and P. J. Flynn. (1999, Data clustering: A review. *ACM Comput.Surv. 31(3),* pp. 264-323.

[56] J. A. Hartigan. (1975, *Clustering Algorithms.*

[57] K. Fukunaga. (1990, *Introduction to Statistical Pattern Recognition.*

[58] T. Kohonen. (1988, Self-organization and associative memory.

[59] R. P. Lippmann. An introduction to computing with neural nets. *ARIEL 209*pp. 115.245.

[60] J. B. MacQueen. Some methods for classification and analysis of multivariate observations.

[61] M. Sabin and R. Gray. (1986, Global convergence and empirical consistency of the generalized lloyd algorithm. *IEEE Trans. Inf. Theory 32(2),* pp. 148-155.

[62] C. D. A. BEI and R. M. Gray. (1985, An improvement of the minimum distortion encoding algorithm for vector quantization. *IEEE Trans. Commun. 33(10),* pp. 1121-1133.

[63] K. Lengwehasarit and A. Ortega. (2001, Probabilistic partial-distance fast matching algorithms for motionestimation. *IEEE Transactions on Circuits and Systems for Video Technology 11(2),* pp. 139-152.

[64] M. S. Dawson, A. K. Fung and M. T. Manry. (1993, Surface parameter retrieval using fast learning neural networks. *Remote Sens. Rev. 7(1),* pp. 1-18.

[65] M. S. Dawson, J. Olvera, A. K. Fung and M. T. Manry. Inversion of surface parameters using fast learning neural networks. Presented at IGARSS'92.

[66] A. K. Fung, Z. Li and K. S. Chen. (1992, Backscattering from a randomly rough dielectric surface. *IEEE Trans. Geosci. Remote Sens. 30(2),* pp. 356-369.

[67] A. K. Fung. (1994, Microwave scattering and emission models and their applications. *Norwood, MA: Artech House, 1994.*

[68] Y. Oh, K. Sarabandi and F. T. Ulaby. (1992, An empirical model and an inversion technique for radar scattering from bare soil surfaces. *IEEE Trans. Geosci. Remote Sens. 30(2),* pp. 370-381.

[69] K. Rohani, M. T. Manry, M. Inc and F. Worth. The design of multi-layer perceptions using building blocks. Presented at International Joint Conference on Neural Networks.

[70] K. Liu, S. Subbarayan, R. R. Shoults, M. T. Manry, C. Kwan, F. I. Lewis and J. Naccarino. (1996, Comparison of very short-term load forecasting techniques. *IEEE Trans. Power Syst. 11(2),* pp. 877-882.

[71] M. T. Manry, R. Shoults and J. Naccarino. Automated system for developing neural network short term load forecasters. Presented at PROC AM POWER CONF.

BIOGRAPHICAL INFORMATION

Rohit Rawat was born in India in 1986. He did his Bachelor of Technology in Electronics and Communication Engineering from Guru Gobind Singh Indraprastha University, New Delhi in May 2007. He obtained his Master of Science degree from the University of Texas at Arlington in December 2009. He is also a member and active officer of the UTA chapters of Tau Beta Pi and Eta Kappa Nu engineering honor societies. He worked for Motorola as an intern in the summers of 2008 and 2009, where he worked on developing embedded software. His current research interests include image processing, pattern and speech recognition.