IMPROVED DESIGN OF A PIECEWISE LINEAR NETWORK

by

SAYANTANI PAL

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 2008

ACKNOWLEDGEMENTS

When I first started working in the Image Processing and neural network laboratory (IPNNL) at UTA, I knew very little about the probable road I have now taken, the people I was going to be best of friends with and most importantly the knowledge I was going to grasp in this extremely short venture called the masters degree. At that point about two years ago, I was unaware how Dr Michael Manry, would lead me to understand better what I have today adopted as my career. He has always taken excessive amounts of time through his terribly busy schedules to help me learn the curves. Through this nurturing process, I have learned more than I could have ever dreamed of and have acquired focus for many other goals in my life. I admire his subject expertise, contribution and devotion to the field of Neural Networks, and unceasing help to his students in discovering the field of Neural Networks

I must also thank Dr.Stephen Gibbs for allowing the room for my growth within UTA, without which it would have been very hard for me to pursue my need to learn and work to my fullest. As a mentor he allowed the flexibility and was a crusade of constant source of excitement. He engaged me often in both intellectual thoughts on the world and on life and also gave me a mammoth chance to work closely with undergrad students in their senior year projects. For the influence and help throughout the last 2 years of my life, I wish to thank him very much.

A very special appreciation to all my teachers in Auxillium convent school, India and at North Bengal University whose strive to educate the students formed the foundation of what I am today and gave me the courage to pursue this degree.

Last but not the least, I thank my friends all over the world and my family for constantly being with me and helping me think big

August 18, 2008

ABSTRACT

IMPROVED DESIGN OF A PIECEWISE LINEAR NETWORK

Sayantani Pal, MS

The University of Texas at Arlington, 2008

Supervising Professor: Michael T. Manry

An efficient design of the Piecewise Linear Network (PLN) which maps the N dimensional input vector space into an M dimensional output vector space is presented. Several clusters are formed in the given datasets and a linear mapping is fitted for each cluster using regression and the total mapping error is calculated. We have designed this network using Self Organizing Map (SOM) clustering which uses the Euclidean distance measure. Again the same has been designed by using various weighted distance measures whose results are compared. It is observed that there is a decrease in Mean Square Error (MSE) of the PLN, when it is finally re-designed with the optimum version of the weighted distance measure.

Design of the PLN using Sequential Leader Algorithm (SLA) has also been implemented in this thesis and the results are compared with the one designed using SOM. It is shown that the SLA outperforms the Self-organizing map clustering. In SLA a threshold to optimize the number of clusters formed is also computed and the PLN is designed accordingly. On deleting of several clusters formed by the Sequential leader algorithm to match the number used in SOM using several heuristics, and on designing the PLN with the final number of clusters, the Mean square error(MSE) of the designed PLN decreases.

A pruning algorithm for pruning the insignificant clusters in designing the PLN is demonstrated. This algorithm designs the network by pruning one cluster at a time, without

significantly decreasing the MSE and hence the design becomes more compact. A second version of the pruning method which makes the design more efficient is also implemented.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

LIST OF NOTATIONS

PLN	Piecewise Linear Network
MLP	MultiLayer Perceptron
FLN	Functional Link Network
SOM	Self Organizing Maps
SLA	Sequential Leader Algorithm
$\mathbf{x}_p$ or $\mathbf{x}(n)$	Input Vector Of The Pattern
$\mathbf{t}_p$	Desired Output Vector Of The Pattern
$\mathbf{y}_p$	Calculated Output Vector Of The Pattern
$\mathbf{m}_\mathbf{k}(n)$	Cluster Mean Vector
$d(\mathbf{x},\mathbf{m}_\mathbf{k})$	Distance measure Between Input Pattern And Cluster Mean

CHAPTER 1

INTRODUCTION

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, etc, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well.

Neural network simulations appear to be a recent development. However, this field was established before the advent of computers, and has survived at least one major setback and several eras. The first artificial neuron was produced in 1943 by the neurophysiologist Warren McCulloch and the logician Walter Pitts [1].

1.1 Neural Networks Types

1.1.1 Single/Multilayer Feed Forward Networks

The feed forward neural network was the first and arguably simplest type of artificial neural network devised. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes[2, 46]. There are no cycles or loops in the network. In this kind of neural network the connections between the units do not form a directed cycle. This is different from recurrent neural networks.

The earliest kind of neural network is a single-layer perceptron network, which consists of a single layer of output nodes; the inputs are fed directly to the outputs via a series of weights. In

this way it can be considered the simplest kind of feed-forward network. The sum of the products of the weights and the inputs is calculated in each node, and if the value is above some threshold (typically 0) the neuron fires and takes the activated value (typically 1); otherwise it takes the deactivated value (typically -1)

Multiple-layer perceptron networks consists of multiple layers of computational units, usually interconnected in a feed-forward way. Each neuron in one layer has directed connections to the neurons of the subsequent layer. In many applications the units of these networks apply a sigmoid function as an activation function [2, 4, 6, 7].

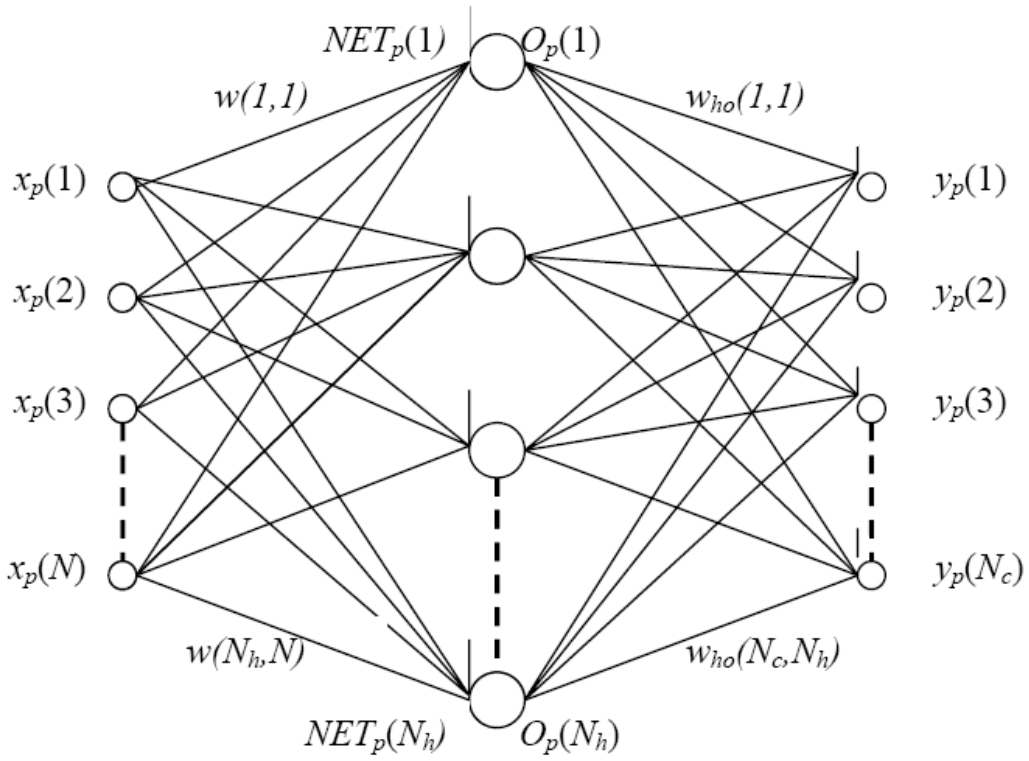

Fig 1.1 Structure Of A Multilayer Perceptron

1.1.2 Radial Basis Function Networks

A RBF is a function which has built into a distance criterion with respect to a centre. Radial basis functions have been applied in the area of neural networks where they may be used as a replacement for the sigmoidal hidden layer transfer characteristic in Multi-Layer Perceptron [49]. RBF networks have two layers of processing: In the first, input is mapped onto each RBF in the 'hidden' layer. The RBF chosen is usually a Gaussian. In regression problems the output layer is then a linear combination of hidden layer values representing mean predicted output. The interpretation of this output layer value is the same as a regression model in statistics. In classification problems [3], the output layer is typically a sigmoid function of a linear combination of hidden layer values, representing a posterior probability. Performance in both cases is often improved by shrinkage techniques and is known to correspond to a prior belief in small parameter values (and therefore smooth output functions) in a Bayesian framework [2].

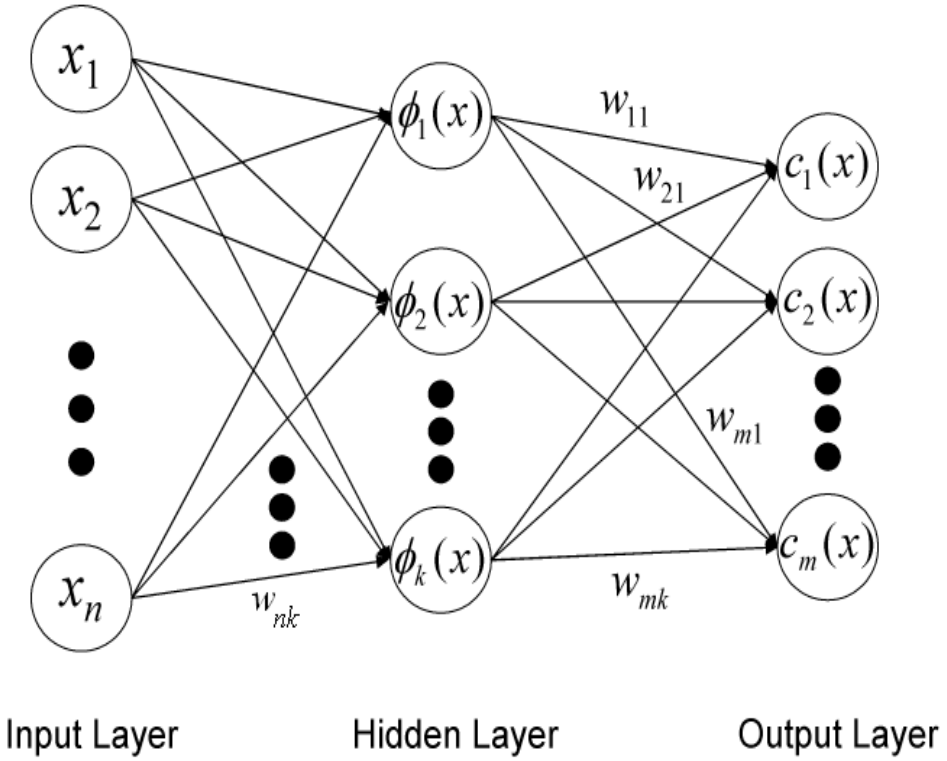

Fig 1.2 Structure Of A Radial Basis Function Network

They have the advantage of not suffering from local minima in the same way as Multi-Layer Perceptron [2, 46, 47]. This is because the only parameters that are adjusted in the learning process are the linear mapping from hidden layer to output layer. Linearity ensures that the error surface is quadratic and therefore has a single easily found minimum. In regression problems this can be found in one matrix operation. In classification problems the fixed non-linearity introduced by the sigmoid output function is most efficiently dealt with using iteratively re-weighted least squares.

Radial Basis Function (RBF) networks have the disadvantage of requiring good coverage of the input space by radial basis functions. RBF centers are determined with reference to the distribution of the input data, but without reference to the prediction task. As a result, representational resources may be wasted on areas of the input space that are irrelevant to the

learning task. A common solution is to associate each data point with its own centre, although this can make the linear system to be solved in the final layer rather large, and requires shrinkage techniques to avoid overfitting.

1.1.3 Recurrent Networks

Contrary to feed forward networks, recurrent neural networks (RNs) are models with bi-directional data flow. While a feed forward network propagates data linearly from input to output, RNs also propagate data from later processing stages to earlier stages, thus the connections between units form a directed cycle [50, 51].

This creates an internal state of the network which allows it to exhibit dynamic temporal behavior. Due to this kind of behavior, can become chaotic, hence dynamical systems theory is used to model and analyze them. While a feed forward network propagates data linearly from input to output, recurrent networks (RN) also propagate data from later processing stages to earlier stages.

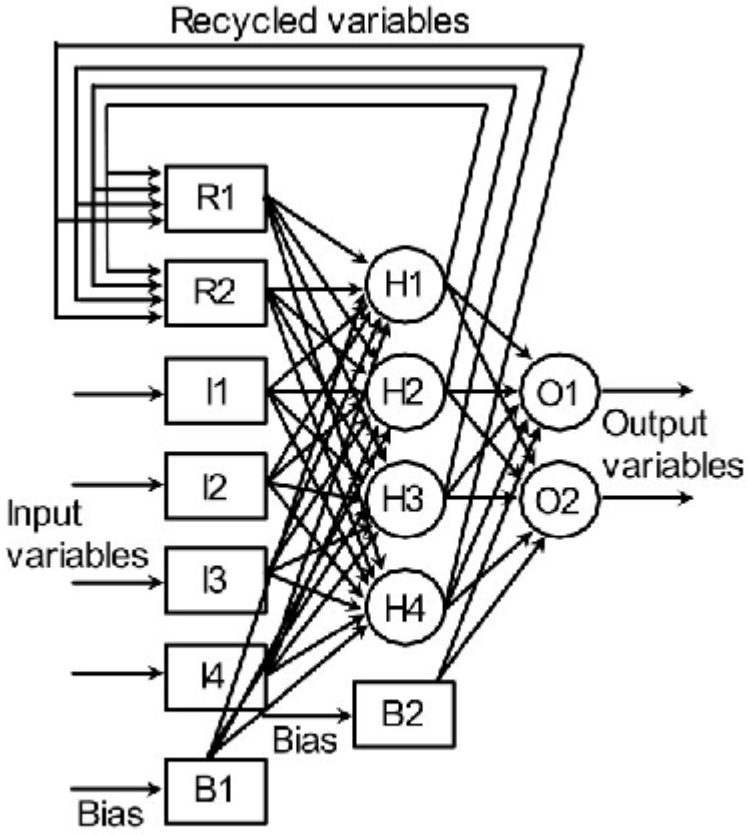

Fig 1.3 Structure of a Recurrent Neural Network

1.2 Neural networks As Function Approximators And Classifiers

Neural networks have greatly been used as function approximators and classifiers

1.2.1 Neural Networks As Function Approximators

To approximate a function is to determine the closest match of a task specific function in a task specific way, i.e. to estimate the closest functional relationship between a N-dimensional input vector space and M-dimensional output vector space [3, 6, 7].They were first proposed as function approximators based on the Kolmogorov theorem. Since then, it has been recognized

that multilayer Artificial Neural Networks, with an arbitrary squashing function in the hidden layer, can approximate any multivariate function to any degree of accuracy. Based on this, backpropagation networks have been trained to realize any arbitrary function. The ability of neural networks as function approximators has been reviewed in detail in chapter 3.

1.2.2 Neural Networks As Classifiers

Classification is a procedure in which individual items are placed into groups based on quantitative information on one or more characteristics inherent in the items (referred to as traits, variables, characters, etc) and based on a training set of previously labeled items. Thus pattern classification is the process of taking in raw data and deciding on the “category or class” of the data.

Formally, the problem can be stated as follows: given training data $(\mathbf{x}_p, \mathbf{t}_p)$ where $\mathbf{x}_p$ is the input vector of the pattern and $\mathbf{t}_p$ is the output vector(since we are doing a classification, this is the class ID number) of the training pattern; to produce a classifier $\mathbf{H} : \mathbf{x}_p \rightarrow \mathbf{t}_p$ which maps an object $\mathbf{x}_p \in \mathbf{X}$ to its classification label or class ID number $\mathbf{t}_p \in \mathbf{T}$. Statistical classification algorithms are typically used in pattern recognition systems.

The utility of artificial neural network models lies in the fact that they can be used to infer a function (approximate it or classify it) from observations. This is particularly useful in applications where the complexity of the data or task makes the design of such a function by hand impractical [2].

1.3 Objectives and Organization Of The Thesis

Function approximators are briefly reviewed in chapter 3, which talks about why a local NN like the PLN should be used over global NN like MLP or FLNs. A review of various clustering techniques is given in chapter 2. It discusses various conventional clustering algorithms; and also describes others like Kohonen's Self Organizing Map (SOM), K-means and Sequential leader follower algorithm (SLA). A brief review of function approximators is done in chapter 3. Chapter 4 deals with the design of a Piecewise Linear Network (PLN) as function approximators; basic version is done implementing the SOM by using Euclidean distance measure and also a brute force re-design of the PLN using various weighted Euclidean distance measure is employed. Also SLA is also used to design the network and is compared to the basic design. An efficient algorithm for pruning the least significant clusters in the PLN without degrading the mean square error of the network is drawn out in chapter 5.

CHAPTER 2

REVIEW OF VARIOUS CLUSTERING TECHNIQUES

A cluster is a set of similar objects. The goal of clustering is to reduce the amount of data by categorizing or grouping similar data items together. Such grouping is pervasive in the way humans process information, and one of the motivations for using clustering algorithms is to provide automated tools to help in constructing categories or taxonomies [9, 31, 32].

Data clustering is a common technique for statistical data analysis, which is used in many fields, including machine learning, data mining, pattern recognition, image analysis and bioinformatics. The computational task of classifying the data set into k clusters is often referred to as **K**-clustering.

2.1 Applications Of Clustering

The principle functions of clustering are to name, to display, to summarize, to predict and to require explanation. Thus all objects in the same cluster will be given the same name and will be expected to have the same property. Thus if some objects in a cluster have a certain property, all the other objects will be expected to have the same property.

Hence the similarity of properties of objects / data in a cluster calls in for immense number of applications in every field of life. One has to cluster a lot of things on the basis of similarity either consciously or unconsciously. In the field of computers too, use of data clustering has its own value. Especially when it comes information retrieval data clustering plays an important role. Some of the applications are listed below [18, 19].

1. Image processing and pattern recognition: The ease with which humans recognize a face, comprehend spoken words, read handwritten characters, identify and distinguish many things by feel disguises the astonishingly complex process of pattern recognition and

classification. Clustering is needed to be done for all types of pattern classification and recognition. Thus clustering is the act of taking in raw data and deciding on the “category” of the data. Over the past many years, humans have evolved highly sophisticated neural and cognitive systems for such tasks. It is only natural that we should seek to design and build machines that can recognize and classify patterns, by means of forming similar types of clusters. Hence clustering is the “means” for classification. From automated speech recognition, fingerprint identification, optical character recognition, DNA sequence matching, and much more, it is obvious that reliable, accurate pattern recognition by machines is immensely useful.

2. Data Mining: Clustering is often one of the first steps in data mining analysis. It identifies groups of related records that can be used as a starting point for exploring further relationships. This technique supports the development of population segmentation models, such as demographic-based customer segmentation. Additional analyses using standard analytical and other data mining techniques can determine the characteristics of these segments with respect to some desired outcome.

3. In the WWW: used for document classification and clustering different kids of weblogs for similar kind of data.

2.2 Distance Measures

An important step in any clustering is to select a distance measure, which will determine how the similarity of two elements is calculated. This will influence the shape of the clusters, as some elements may be close to one another according to one distance and further away according to another.[31,32] Common distance functions are given below:

i. The Euclidean distance: this is the most common distance measure in published studies and most research areas is the Euclidean distance or the squared Euclidean distance.

$$d(\mathbf{x},\mathbf{m}_\mathbf{k}) = \frac{1}{N_v}\sum_{n=1}^{N}[\mathbf{x}(n)-\mathbf{m}_\mathbf{k}(n)]^2 \qquad (1.1)$$

Squared Weighted Euclidean distance measures are also common which is defined as

$$d(\mathbf{x},\mathbf{m}_\mathbf{k}) = \sum_{n=1}^{N}[\mathbf{x}(n)-\mathbf{m}_\mathbf{k}(n)]^2 / \boldsymbol{\sigma}^2{}_n \qquad (1.2)$$

Where $\boldsymbol{\sigma}^2{}_n$ is the Variance Vector of the Input Pattern Vector.

ii. The Manhattan distance (also called taxicab norm or 1-norm) which computes the distance that would be traveled to get from one data point to the other if a grid like path is followed.

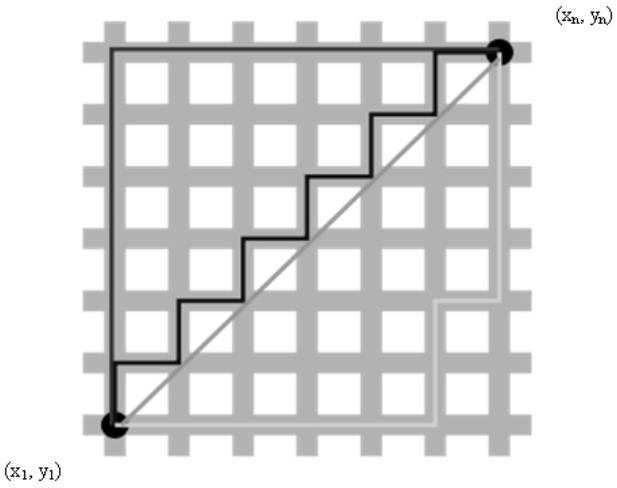

Fig 2.1 Manhattan Distance Grid

The taxicab distance between two points in a Euclidean space with fixed Cartesian coordinate system is the sum of the lengths of the projections of the line segment between the points onto the coordinate axes. For example, the taxicab distance between the point P_1 with coordinates (x_1, y_1) and the point P_2 at (x_2, y_2) is

$|x_1 - x_2| + |y_1 - y_2|$

iii. The Mahalonobis distance corrects data for different scales and correlations in the variables. This is given by the equation

$d(\mathbf{x}, \mathbf{m}_\mathbf{k}) = \sum_{n=1}^{N} \sum_{m=1}^{N} \mathbf{a}_\mathbf{k}(n,m)[\mathbf{x}(n) - \mathbf{m}_\mathbf{k}(n)][\mathbf{x}(n) - \mathbf{m}_\mathbf{k}(n)])$ where

$\mathbf{a}_\mathbf{k}(n,m) = \mathbf{C}^{-1}{}_k$ and

$\mathbf{C}_k = E[\mathbf{x} - \mathbf{m}_\mathbf{x}][\mathbf{x} - \mathbf{m}_\mathbf{x}]^T$.

iv. The angle between two vectors can be used as a distance measure when clustering high dimensional data.

v. The Hamming distance (sometimes edit distance) measures the minimum number of substitutions required to change one member into another.

2.3 Types Of Clustering Methods

There are many clustering methods/ algorithms available, and each of them may give a different grouping of a dataset. The choice of a particular method will depend on the type of output desired. The known performance of method with particular types of data, the hardware and software facilities available and the size of the dataset. In general, clustering methods may be divided into two categories based on the cluster structure which they produce.

Again the clustering methods can also be hierarchical The hierarchical methods produce a set of nested clusters in which each pair of objects or clusters is progressively nested in a larger cluster until only one cluster remains. The hierarchical methods can be further divided into agglomerative or divisive methods. In agglomerative methods, the hierarchy is build up in a series

of N-1 agglomerations, or Fusion, of pairs of objects, beginning with the un-clustered dataset. The less common divisive methods begin with all objects in a single cluster and at each of N-1 steps divide some clusters into two smaller clusters, until each object resides in its own cluster.

In other methods, we divide a dataset of N objects into M clusters, with or without overlap [34, 35].These methods are sometimes divided into partitioning methods, in which the classes are mutually exclusive, and the less common clumping method, in which overlap is allowed. Each object is a member of the cluster with which it is most similar; however the threshold of similarity has to be defined.

In some Divisive/ Partitioning clustering algorithms like SOM assume that the number of partitions is pre-known while in Sequential leader the total numbers of clusters are formed during data pass based on a definitive threshold. Unfortunately no unified theory for determining the total number of optimum clusters has been fully developed and accepted.

The conventional clustering methods like the Sequential leader clustering and K-Means clustering are described in section 2.4 and 2.5

2.4 Sequential Leader Or Leader – Follower Algorithm

This algorithm does not have a pre defined number of clusters but the clusters are formed during data pass through the algorithm based on a pre defined threshold. Also there are certain ways of calculating the optimum threshold which would be described in details in section The performance of Sequential Leader has been found to be faster than SOM or other conventional clustering algorithms [31, 35].

Given the time and resources, the best way to assign all feature vectors $\mathbf{x}_p$ where p = 1 to N_v, to clusters would be to identify all the possible partitions and to select the most sensible one according to a pre-selected criterion. However this is not possible even for moderate values

of N_v. Let S (N_v , N_c) be the number of all possible clustering of N_v vectors into k groups where k=1 to N_c. By definition no luster is empty. Thus the p[th] vector will either be added to the preformed clusters or it will form a new cluster to the existing clusters.

The algorithm is defined as follows:

The given inputs are a predefined threshold T, N_v training / feature vectors $\mathbf{x}_p$ each of dimensions N, a distance measure d. In this case the general Euclidean distance given previously by $d(\mathbf{x},\mathbf{m_k})$ can be used.

To start T can be chosen as $\frac{d(\mathbf{x},\mathbf{m_k})}{10}$

and modified when the optimum threshold is calculated

(1) Initialization; N_c (number of clusters) = 1.

m1 (the first cluster center vector) = x1,

p (the training vector index) = 1,

$N_v(k)$ (Number of members of kth cluster) = 1. m (1) = 1 (cluster membership of p[th] training vector)

(2) p = p + 1. If p > N_v, stop

(3) d = min $d(\mathbf{x},\mathbf{m_k})$. 1 ≤ k ≤ K where k denotes index of closest center vector.

(4) If d ≤ T, then

$N_v(k)$ = $N_v(k)$ + 1 (increment size of kth cluster), m (p) = k (cluster membership of p[th] training vector)

Else

N_c = N_c + 1 (increment number of clusters m (p) = K (cluster membership of p[th] training vector)

$\mathbf{m}_k$ = $\mathbf{x}_p$ (choose center of new cluster)

$N_v(k)$ =1

(5) Go to (2) (examine next training vector)

This algorithm is quite straightforward and fast method in which all the feature vectors are presented to it once after the threshold has been calculated earlier. The final result is usually dependant on the order in which the vectors are presented to the algorithm. The schemes tend to produce compact hyper-spherically or hyper-ellipsoidally shaped clusters, depending on the distance measure used.

2.4.1 Calculation Of The Optimum Threshold.

The threshold to determine whether a feature vector belongs to a pre formed cluster or becomes a new cluster mean is calculated in the following way. At first data is passed through the algorithm to get the initial number of clusters K_0.

Now we know that the overall volume of the feature vector space V is given by

$$V = [\sqrt{\sum_{n=1}^{N} [\mathbf{x}(n)_{\max} - \mathbf{x}(n)_{\min}]^2}]^N \quad (2.1)$$

The total volume of the clusters V_c is given by

$V_c = \sqrt{T_0^N}$ where (2.2)

$$T_0 = d(\mathbf{x}, \mathbf{m_k}) = \frac{1}{N_v} \sum_{k=1}^{Nv} [\mathbf{x}(n) - \mathbf{m_k}(n)]^2$$

i.e. T_0 is the threshold which is used to find the number of clusters formed in SLA during data pass.

Thus the estimated numbers of clusters K_{est} are given by

$$K_{est} = \frac{V}{V_c}. \quad (2.3)$$

On substituting the value of V_c from above we get

$$K_{est} = \frac{V}{T_o^{\frac{N}{2}}} \tag{2.4}$$

Solving for T_o we get,

$$T_o = (\frac{V}{K_{est}})^{\frac{2}{N}} \tag{2.5}$$

In order to normalize the value of T_o we will multiply it with a normalizing coefficient C.

$$T_o = (\frac{V}{K_{est}})^{\frac{2}{N}} C \tag{2.6}$$

Thus solving for the value of C we get,

$$C = T_0 (\frac{K_0}{V})^{\frac{2}{N}} \tag{2.7}$$

i.e. C is the "normalizing coefficient" in SLA

where K_0 is the number of clusters formed by passing the data through the Sequential leader once for the first time.

Thus we can evaluate an "optimal threshold" T_{opt} as

$$T_{opt} = C(\frac{V}{K})^{\frac{2}{N}} \tag{2.8}$$

where K is the number of clusters wanted by the user which is the same as the pre defined number of clusters used in SOM.

After the Optimum threshold is calculated we pass the data through the algorithm once more to get K_{Leader}, where K_{Leader} is the number of clusters formed by running SLA now. If $K_{Leader} > K$, some clusters are deleted, according to some heuristic functions to bring the number of clusters

down to K, which is the same as that is pre defined in SOM. Then a piecewise linear network is designed.

2.5 K-Means Clustering

K-means [35, 36, 52] is one of the simplest unsupervised learning algorithms that solve the well known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed a priori. The main idea is to define k centroids, one for each cluster. These centroids should be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest centroid. When no point is pending, the first step is completed and an early groupage is done. At this point we need to re-calculate k new centroids as barycenters of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new centroid. A loop has been generated. As a result of this loop we may notice that the k centroids change their location step by step until no more changes are done Finally, this algorithm aims at minimizing an *objective function*, in this case a squared error function. The objective function

$$J = \sum_{k=1}^{N_c} \sum_{p=1}^{N_v} [\mathbf{x_p}(n) - \mathbf{m_k}(n)]^2 ,$$

where $[\mathbf{x_p}(n) - \mathbf{m_k}(n)]^2$ is a chosen distance measure between a data point $\mathbf{x_p}(n)$ and the cluster centre $\mathbf{m_k}(n)$, is an indicator of the distance of the N_v data points from their respective cluster centres.

2.6 Kohonen's Self Organizing Maps

Self-organizing maps (SOMs) are a data visualization technique invented by Professor Teuvo Kohonen which reduces the dimensions of data through the use of self-organizing neural networks [8].

This is a type of unsupervised learning method used to produce low dimensional representation of the training samples while presenting the topological features of the input features. The problem that data visualization attempts to solve is that humans simply cannot visualize high dimensional data as is so techniques are created to help us understand this high dimensional data. The way SOMs go about reducing dimensions is by producing a map of usually 1 or 2 dimensions which plot the similarities of the data by grouping similar data items together. So SOMs accomplish two things, they reduce dimensions and display similarities. Hence, we can say that SOM has the special property of effectively creating spatially organized "internal representation" of various features of input signals and their abstractions After supervised fine tuning of its weight vectors, the Self Organizing Maps have been particularly successful in various pattern recognition tasks involving very noisy signals.

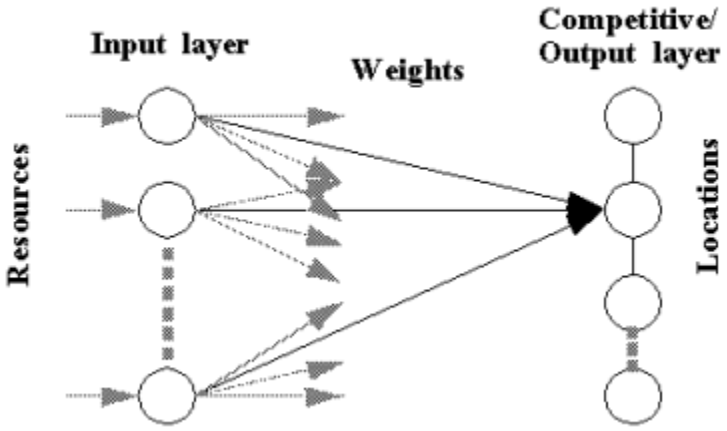

Fig 2.2 Kohonen's Self Organizing Map Structure

How the input features cluster themselves according to similarity , in the form of clusters in shown in the figure 2.2

Thus the goal of SOM is to associate different feature vectors to respond similarly to certain input patterns or to group the feature vectors with similar properties into several clusters. In particular, these maps are used in speech recognition, robotics, process control, telecommunications etc

2.6.1 Kohonen's Self-Organizing Feature Map (SOM) For 1-D Clusters

Given: K clusters, Nv vectors x_p of dimension N, decreasing functions $z(t)$ and $N(t)$,and N_{it} [10,11,12].

(1) Find the mean $\mathbf{m_n}$ and standard deviation $\boldsymbol{\sigma}_\mathbf{n}$ of each of the N inputs, 1 ≤ n ≤ N.

(2) Initialize the weights (cluster means) $\mathbf{m_k}$ as random vectors with all nth elements generated from a random number generator using parameters $\mathbf{m_n}$ and $\boldsymbol{\sigma}_\mathbf{n}$. Initialize the iteration number as it = 0.

(3) it = it + 1. Quit if it > N_{it} t.

(4) For 1 ≤ p ≤ N_v ,

t = p + (it-1) N_v ,

Find k such that $d(\mathbf{x},\mathbf{m_k})$ is minimum,

Update $\mathbf{m_n}$ as $\mathbf{m_n} = \mathbf{m_n} + z(t)\,[\mathbf{X}_p - \mathbf{m_n}]$ for |k-n| ≤ $N(t)$.

End

(5) Go to (3)

i. Exponential gains and neighborhood functions are defined as follows

$$z(t) = a_1 e^{\frac{-t}{T_1}}$$

and the time constant $N(t) = a_2 + a_3 e^{\frac{-t}{T_2}}$.

Also

$$T_1 = \frac{N_v N_{it}}{3}$$

And

$$T_2 = \frac{N_v N_{it}}{10}.$$

In all the above equations N_v is defined by the number of patterns in the dataset and N_{it} is the number of iterations made to design the network

ii Linearly decreasing gains and neighborhoods are given as

$$z(t) = a_1(1 - \frac{t}{N_v N_{it} + 1})$$

and

$$N(t) = a_2(1 - \frac{t}{N_v . N_{it} . (0.8)}) \text{ for } 1 \le t \le N_v N_{it}.$$

2.6.2 Kohonen's Self-Organizing Feature Map (SOM) For 2-D Clusters

Given: Clusters K, K1, K2 (K = K1,K2), N_v training vectors xp of dimension N, a distance measure d, decreasing functions $z(t)$ and $N(t)$, and the number of iterations N_{it}.

(1) Initialize the weights (cluster means) $\mathbf{m_{ku}}$ (1 ≤ k ≤ K1 and 1 ≤ u ≤ K2) as random vectors. Initialize the iteration number as it = 0.

(2) it = it + 1. Quit if it > N_{it}.

(3) For 1 ≤ p ≤ N_v, t = p + (it-1) N_v,

Find (k, u) such that $d(\mathbf{x}, \mathbf{m}_{\mathbf{ku}})$ is minimum,

Update $\mathbf{m}_{\mathbf{mn}}$ as $\mathbf{m}_{\mathbf{mn}} = \mathbf{m}_{\mathbf{mn}} + z(t)$ [xp - $\mathbf{m}_{\mathbf{mn}}$] for |m-k| ≤ $N(t)$ and |n-u| ≤ $N(t)$.

(4) Go to (2)

CHAPTER 3

REVIEW OF FUNCTION APPROXIMATORS

3.1 Mapping (Function Approximation)

A problem common to many disciplines is that of adequately approximating a function of many variables, given only the value of the function (which may be perturbed by noise) at various points in the dependant variable space. A typical mapping or function approximation problem can be formulated in terms of an N-dimensional feature or input vector $\mathbf{X}_p$, an M- dimensional response or target vector $\mathbf{T}_p$, a data collection **D** of observed ($\mathbf{X}_p$, $\mathbf{T}_p$) pairs and the goal of learning to estimate t from **x**. The pair (**x**, **t**) obeys some unknown joint probability distribution ***P***. Often , the physical characteristics and limitations of the devices used for observation and the media, from which information is obtained , makes **D** noisy. The mapping problem is to construct a function ***f*(X)** based on the data **D** so that ***f*(X)** approximates or estimates the desired response $\mathbf{T}_p$. Conventionally ***f*** is chosen to minimize some cost function, usually the sum of squared errors. This kind of problem arises frequently in adaptive control systems, communications, and signal processing and pattern recognition.

One can distinguish two major classes of function approximation problems: First, for known target functions approximation theory, is the branch of numerical analysis that investigates how certain known functions (for example, special functions) can be approximated by a specific class of functions (for example, polynomials or rational functions) that often have desirable properties (inexpensive computation, continuity, integral and limit values, etc.).

Second, the target function, call it ***f***, may be unknown; instead of an explicit formula, only a set of points of the form (**x**, ***f*(x)**) is provided. Depending on the structure of the domain and co domain

of ***f***, several techniques for approximating ***f*** may be applicable. For example, if ***f*** is an operation on the real numbers, techniques of interpolation, extrapolation, regression analysis, and curve fitting can be used. If the co domain of ***f*** is a finite set, one is dealing with a classification problem instead.

To some extent the different problems (regression, classification) have received a unified treatment in statistical learning theory, where they are viewed as supervised learning problem.

The function approximation methods fall broadly into two categories: local and global. A global approximation is one in which all the network coefficients are used to process and input vector to an output vector; these can be made with polynomial networks like Functional Link Networks (FLN) and multilayer perceptron (MLP) feed forward networks.[6,7]

3.1.1 Global Neural Networks

Global neural networks approximate or interpolate a continuous, multivariate function **f**(**x**) by an approximating function **f**(**x, w**), where w is a parameter vector such that

w = w*, defines the specific function **f**(**x, w***) in {**f**(**x, w**)}. For a choice of specific **f** the problem is then to find the set of parameters **w** that provides the best approximation of **f** on the set of data. This is called the learning or training process.

The problem of which approximation to use (i.e. which classes of function **f**(**x**) can be effectively approximated by which approximating functions **f**(**x,w**)), is dependant on which architecture is chosen . For MLP neural networks, the representation problem is choosing the right activation function and the correct number of hidden units. For FLN (volterra filter type) networks this problem is translated to choosing the correct degree and the number of polynomial of polynomial terms in the network.

In training the feed forward neural networks, the goal is to achieve the minimum mean square error (MSE). The function to be minimized is often

$$\mathbf{E}(w) = \frac{1}{N_v} \sum_{p=1}^{Nv} \sum_{k=1}^{M} [\mathbf{t}_{pk} - f(\mathbf{x}_p, \mathbf{w})]^2 \qquad (3.1)$$

Where $\mathbf{E}(w)$ is the MSE.

Shortcomings of the MLP networks

1. MLP training algorithms are very time consuming, making them impractical to be used in some applications. Again the training process can be extremely slow to converge, depending on the particular training algorithm and the dataset that is being looked into.

2. Determining the network size for the MLP becomes extremely difficult. Given the input and output layers of the MLP, the optimal number of hidden layers and also the number of neurons in the hidden layer is yet in the stage of experimentation.

3.1.2 Local Neural Networks

Local neural networks often employ various kinds of clustering methods like Kohonen's self organizing map (SOM), sequential leader- follower algorithm, nearest neighbor, to partition the input space into a hierarchy of regions, where simple hyper surfaces are fit to the local data. Thus local neural networks construct local approximations to non linear input-output mappings. Some example of these are Cerebellar model articulation controllers (CMAC), piecewise constant networks (PCN), piecewise linear networks (PLN) and piecewise polynomial networks. The PLNs use local linear models [10, 12]. The PLN, in details is talked about in chapter 4.

3.1.2.1 Piecewise Linear Functions

In mathematics, a piecewise linear function $f : \Omega \rightarrow V$, where V is a vector space

and Ω is a subset of a vector space, is any function with the property that Ω can be decomposed into finitely many convex polytopes, such that **f** is equal to a linear function on each of these polytopes.

A special case is when **f** is a real-valued function on an interval [x_1, x2]. Then f is piecewise linear if and only if [x_1, x2] can be partitioned into finitely many sub-intervals, such that on each such sub-interval I, **f** is equal to a linear function **f(x)** = $a_I x + b_I$. The absolute value function f(x) = | x | is

a good example of a piecewise linear function. Other examples include the square wave, the sawtooth function, and the floor function.

3.2 Observations By Comparing Global And Local Neural Networks

Global neural networks take long to train, thus making them impractical in some applications. Also, there is no usable formal theory to determine the optimal network topology. In this case the hidden networks should be determined experimentally; which is a very time consuming task [43].

Local networks, on the other hand, train faster and provide adequate approximations to many mapping problems. Hence the Piecewise Linear network (PLN) is looked into in details in this thesis

CHAPTER 4

PIECEWISE LINEAR NETWORKS

In this chapter, a piecewise linear network is reviewed. At first SOM clustering is carried out using the Euclidean distance measures and then various versions of the weighted distance measures are used for the design of the PLN. Finally, as discussed in chapter 2, sequential leader clustering is carried out to get the number of clusters, before deleting the lesser useful clusters according to some given heuristics, in order to match the number of clusters defined in SOM[10,12,28,42].

4.1 Piecewise Linear Network Theory [48]

The fundamental properties of piecewise-linear systems are:

1. They are characterized by functional relationships composed of a finite number
of linear regions adjoining one another.
2. The change-over from one linear region to the next is determined by the point at which some quantity becomes greater or less than some other quantity.

Although those systems appear to be closely related to linear systems, it is clear that the superposition principle is not valid in piecewise-linear systems. This fact alone increases enormously the difficulties of analysis and synthesis, and makes the development of an algebraic method of handling them, which differs from conventional techniques, worth while. It may be observed from property 2 that the words "greater" and “less, " i. e., inequalities, play important roles in these systems. It was the recognition of this fact that led to the development of a symbolism that would enable the handling of such concepts algebraically.

The basic design of a piecewise linear networks (PLN), when it pertains to the field of Neural Networks is comprised of two parts. The first part involves clustering the data into K clusters. The second part involves calculation of network weights by solving a set of linear equations whose solution minimizes the Mean Square Error (MSE) of the network. Here each partition or cluster is fitted with a linear model. The output vector is calculated by multiplying the input vector and the weight matrix $\mathbf{A}_k$, of the partition to which the input vector is assigned.

As the number of training patterns N_v tends to infinity, partition based estimates of the regression function converge to the true function and the MSE of the mapping converges to that corresponding to Bayes rule. Hence the Piecewise linear networks are consistent nonparametric estimators.

4.2 Structure Of The Piecewise Linear Network

Consider a training set consisting of N_v patterns $(\mathbf{x}_p, \mathbf{t}_p)$ where the pth example input vector $\mathbf{x}_p$ and the pth example desired output vector $\mathbf{t}_p$ have dimensions N and M, respectively [43]. The PLN architecture consists of

- N_c N-dimensional cluster center vectors $\mathbf{m}_k$, where $1 \leq k \leq N_c$, where N_c is the number of clusters,
- An M (N+1) matrix, $\mathbf{W}_g$ representing the global linear mapping between input and output vectors,
- N_c matrices $\mathbf{W}_k$ of dimension M.(N+1), where $\mathbf{W}_k$ represents the linear mapping for the kth cluster
- A distance measure d ( ), which de-emphasizes random or less useful features, and

• 2N parameters $\mu(i)$, $\sigma(i)$ or 1 $\leq$ i $\leq$ N representing the mean and standard deviation of the input vector elements.

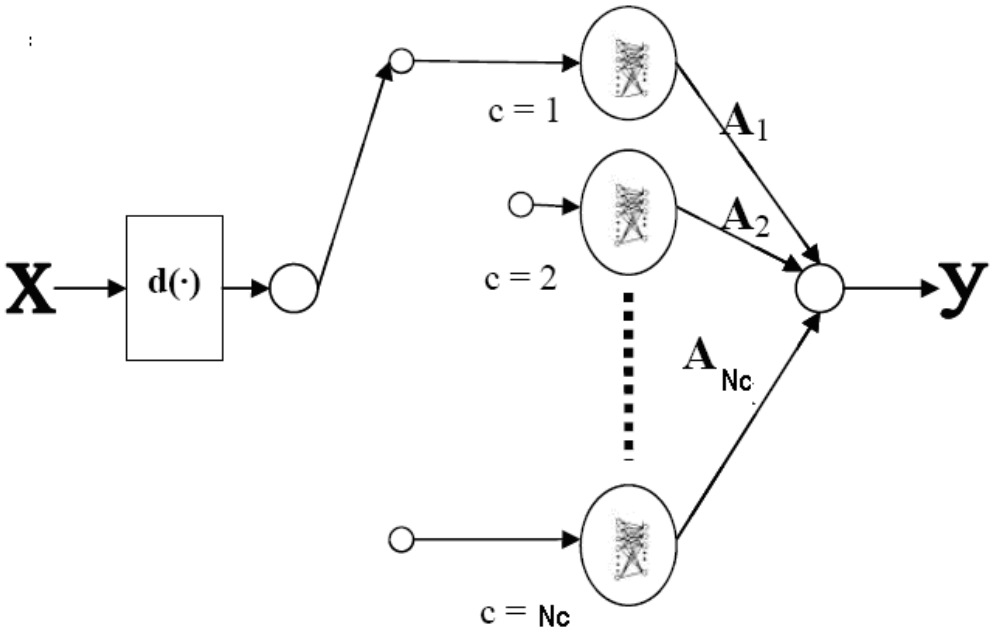

Fig 4.1 Structure Of A Piecewise Linear Network

4.3 Training The Piecewise Linear Network

A global linear mapping between input and the output is obtained .The clustering is performed using Euclidean distance measure given in eq. (1.1) on the normalized inputs using SOM in the first iteration. An input pattern is given to the network; it is assigned to the cluster for which $d\left(\mathbf{x},\mathbf{m}_{\mathbf{k}}\right)$ is a minimum. The p[th] input vector $\mathbf{x}_p$ is augmented as (x^T : 1), where the constant 1 handles the constant component of the mapping. All input vector elements except $\mathbf{x}_{\mathrm{p}}(\mathrm{N+1})$ are normalized as

$$\mathbf{x}'_{\mathrm{p}}(\mathrm{i}) = \frac{(\mathbf{x}_{\mathrm{p}}(\mathrm{i})\boldsymbol{\mu}(\mathrm{i}))}{\boldsymbol{\sigma}(\mathrm{i})} \tag{4.1}$$

$\mathbf{x}_{\mathrm{p}}(\mathrm{N}+1)$ is kept unchanged as 1.

After the clustering is completed, each cluster is fitted with a piecewise linear mapping. The output vectors in each cluster is found by

$$\mathbf{y}_p = \mathbf{A}_k \cdot \mathbf{x}_p \tag{4.2}$$

Where $\mathbf{A}_k$ is the weight matrix of each cluster to which the input patterns belong.

The MSE $\mathbf{E}_k(j)$ for the jth output and the kth cluster is given by

$$\mathbf{E}_k(j) = \sum_{p \in S(k)} [\mathbf{t}_p(j) - \mathbf{y}_p(j)]^2 \tag{4.3}$$

where $\mathbf{y}_p(j)$ is the calculated output. The linear mapping weights $\mathbf{A}_k$ are found by minimizing the error function $\mathbf{E}_k(j)$ for each output by the conjugate gradient algorithm.

4.3.1 Solving For $\mathbf{A}_k$

Clustering results in partitioning of the feature space into K clusters. Each cluster has its own weight matrix $\mathbf{A}_k$, where 1 ≤ k≤ K. For a PLN having linear output layer, the output vector is calculated by simply multiplying the input vector and the weight matrix of the cluster to which the pattern was assigned, as described above, only the winning cluster's weight matrix is turned on in order to generate the output vector. Elaborating on (4.2) the elements of the output vector y, are calculated as

$$\mathbf{E}_k = \frac{1}{N_v(k)} \sum_{q=1}^{Nv(k)} \sum_{i=1}^{Nc} [\mathbf{t}_{qi} - \sum_{n=1}^{N+1} \mathbf{a}_{kin} \mathbf{x}_{qn}]^2 \quad (4.4)$$

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

$$\mathbf{E}_k = \frac{1}{N_v(k)} \sum_{q=1}^{Nv(k)} \sum_{i=1}^{Nc} [\mathbf{t}_{qi} - \sum_{n=1}^{N+1} \mathbf{a}_{kin} \mathbf{x}_{qn}]^2 \quad (4.5)$$

where $N_v(k)$ the number of feature vectors in cluster k and $\mathbf{t}_p$ is the actual output which needs to be read from the data file.

The error gradient with respect to elements of the weight matrix $\mathbf{A}_k$, is given by

$$\frac{\delta \mathbf{E}}{\delta \mathbf{a}_{kjm}} = -2 \frac{1}{N_v(k)} \sum_{q=1}^{Nv(k)} [\mathbf{t}_{qj} - \sum_{n=1}^{N+1} \mathbf{a}_{kjn} \mathbf{x}_{qn}] \mathbf{x}_{qm} \quad (4.6)$$

On further simplification we get

$$\frac{\delta \mathbf{E}}{\delta \mathbf{a}_{kjm}} = -2 \frac{1}{N_v(k)} [\sum_{q=1}^{Nv(k)} \mathbf{t}_{qj} \mathbf{x}_{qm} - \sum_{q=1}^{Nv(k)} \sum_{n=1}^{N+1} \mathbf{a}_{kjn} \mathbf{x}_{qn} \mathbf{x}_{qm}] \quad (4.7)$$

Equation (4.7) can be written in terms of auto-correlation and cross-correlation elements, and respectively, as

$$\frac{\delta \mathbf{E}}{\delta \mathbf{a}_{kjm}} = -2[\mathbf{c}(j,m) - \sum_{n=1}^{N+1} \mathbf{a}_{kjn}\mathbf{r}(n,m)] \quad (4.8)$$

The PLN is designed by minimizing the training MSE. Hence, equating the gradient in (4.8) to zero, in order to minimize the error, yields

$$\mathbf{c}(j,m) = \sum_{n=1}^{N+1} \mathbf{a}_{kjn}\mathbf{r}(n,m) \quad (4.9)$$

Several techniques can be used to solve the set of equations in (3.9), conjugate gradient and Gram-Schmidt orthonormalization being some of them.

Finally we get,

$$\mathbf{A_k} = \mathbf{C^{-1}R}$$

4.3.2 Using Various Versions Of The Weighted Euclidean Distance Measure For PLN Design

The PLN is designed as described above apart from a few variations. $\mathbf{M}_{ik}(n)$ is defined as the inverse of the variance $\boldsymbol{\sigma}^2(n)$ of the input vector $\mathbf{x}(n)$ and $\boldsymbol{\varepsilon}(n)$ is defined as one- tenth of the variance $\boldsymbol{\sigma}^2(n)$ of the input vector $\mathbf{x}(n)$.

In this case the various versions of weights $\mathbf{w}(n)$ which is used for the distance measure is given 5 versions according to

$$\mathbf{w}_0(n) = \mathbf{M}_{ik}(n) \tag{4.10a}$$

$$\mathbf{w}_1(n) = \mathbf{M}_{ik}(n) + 2.\boldsymbol{\varepsilon}(n) \tag{4.10b}$$

$$\mathbf{w}_2(n) = \mathbf{M}_{ik}(n) + 5.\boldsymbol{\varepsilon}(n) \tag{4.10c}$$

$$\mathbf{w}_3(n) = \mathbf{M}_{ik}(n) - 3.\boldsymbol{\varepsilon}(n) \tag{4.10d}$$

$$\mathbf{w}_4(n) = \mathbf{M}_{ik}(n) - 6.\boldsymbol{\varepsilon}(n) \tag{4.10e}$$

Where $\mathbf{M}_{ik}(n) = \dfrac{1}{\boldsymbol{\sigma}^2(n)}$ (4.11)

And $\boldsymbol{\varepsilon}(n) = \dfrac{1}{10\boldsymbol{\sigma}^2(n)}$ (4.12)

And the distance measure used is given by

$$d(\mathbf{x_p}, \mathbf{m_k}) = \sum_{n=1}^{N} \mathbf{w}_m(n)[\mathbf{x}_p(n) - \mathbf{m}_k(n)]^2 \tag{4.13}$$

Thus each input has five versions of distance measures and we get the corresponding n*5 versions of distance measures, where n is the input $1 \le n \le N$. In designing the network, we iterate over the input vector $\mathbf{x}_p(n)$, by finding the minimum MSE of the designed PLN among each version of the weighted distance measure

Hence, we propose n times 5 networks and by brute force the network which gives the minimum MSE is chosen for each iteration when the value of the nth input is changed.

Once the distance which gives the minimum MSE is found, re clustering is done and the A_k, matrix is re calculated to redesign the PLN. The flowchart for the above Brute Force redesign of the PLN is shown in figure 4.2.

The goal for the above method is to minimize the MSE of the PLN in each iteration wherein the weighted distance measure which gives the minimum MSE for the network is chosen and the PLN is designed according to that distance. Thus we find a change in the MSE per iteration which leads to an absolute decrease in the MSE of the PLN when compared to the basic network designed with the weighted Euclidean distance measures given by eq. (1.2).

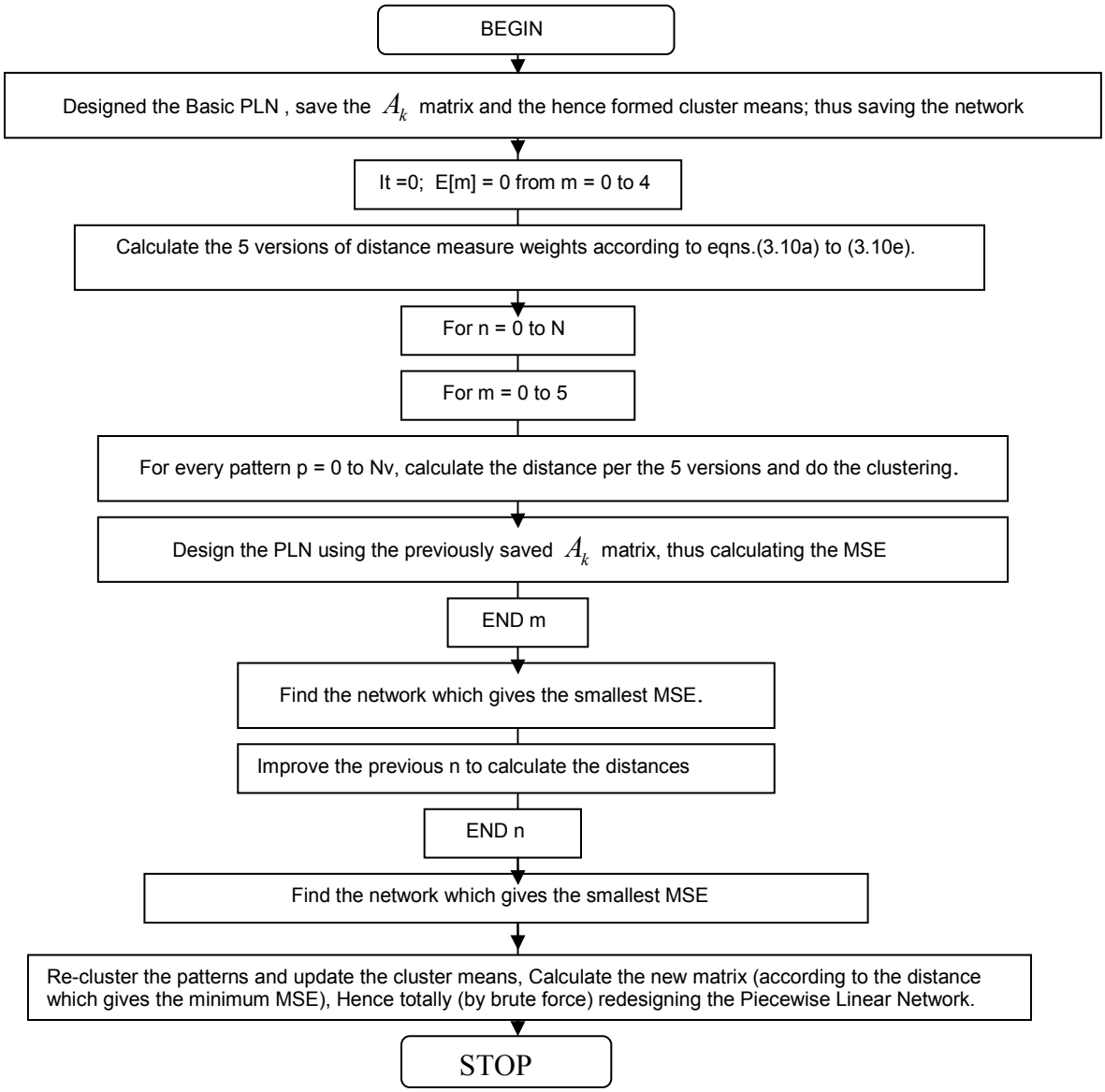

Fig 4.2 Flowchart For Brute Force Redesign Of PLN

4.3.3 Simulation Results

The results for the above algorithm are shown below. The total MSE of the network has been compared against the number of iterations. Each iteration corresponds to a change in the number of inputs.

1.Dataset fmtrain.dat:

There are 5 inputs and 1 output and 1024 training patterns trains a neural network to perform demodulation of an FM (frequency modulation) signal containing a sinusoidal message. The data is generated from the equation

r(n) = Camp * cos[2* PI* n* Cfreq + Mamp *sin(2* PI* n* Mfreq)] where Camp = Carrier Amplitude, Mamp = Message Amplitude, Cfreq = normalized Carrier frequency, Mfreq = normalized message frequency. In this data set, Camp = .5, Cfreq = .1012878, Mfreq = .01106328, and Mamp=5. The five inputs are r(n-2), r(n-1), r(n), r(n+1), and r(n+2). The output is Cos(2* PI* n* Mfreq). In each consecutive pattern, n is incremented by 1.

In this case we have 5 versions of the weighted distance measure. Hence we iterate 5 times with 5 weighted distance measure versions and design the network using 5 versions of the weighted distance measure, choose the one which gives the smallest per iteration and plot it in the graph. In the following graph, the MSE of the PLN versus the number of iterations is shown. It is observed that after redesigning the PLN the overall MSE decreases.

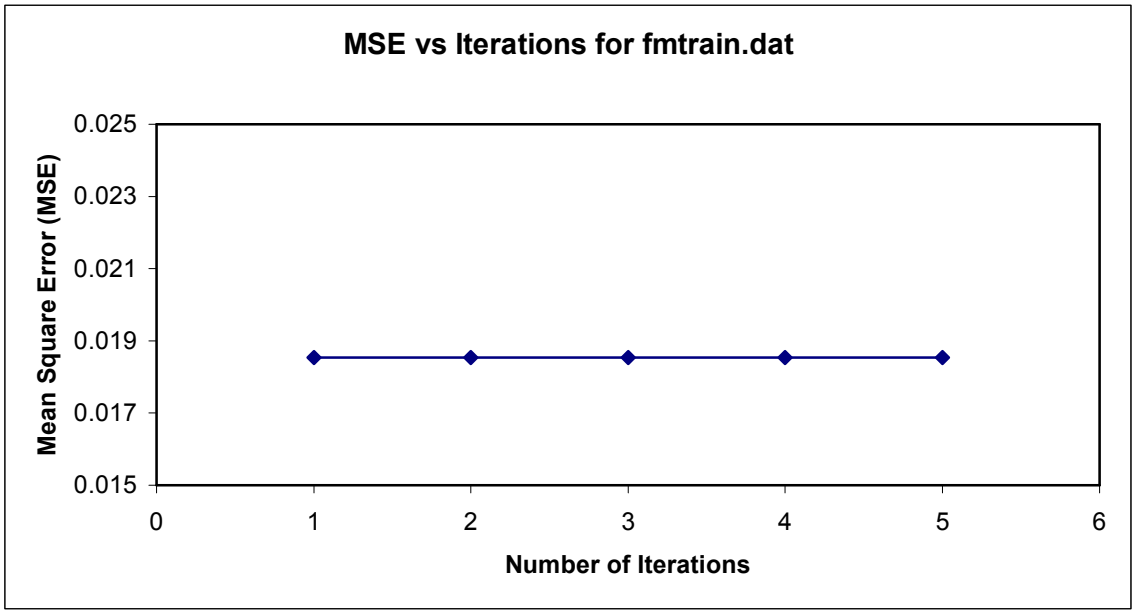

Fig 4.3 Graph For MSE Versus Number Of Iterations For Dataset fmtrain.dat

1. Dataset Mattrn.dat :

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

Here too we have 5 versions of the distance measures and since there are 4 inputs we will iterate 4 times with each of the 5 versions of the measures. The simulation results are shown below.

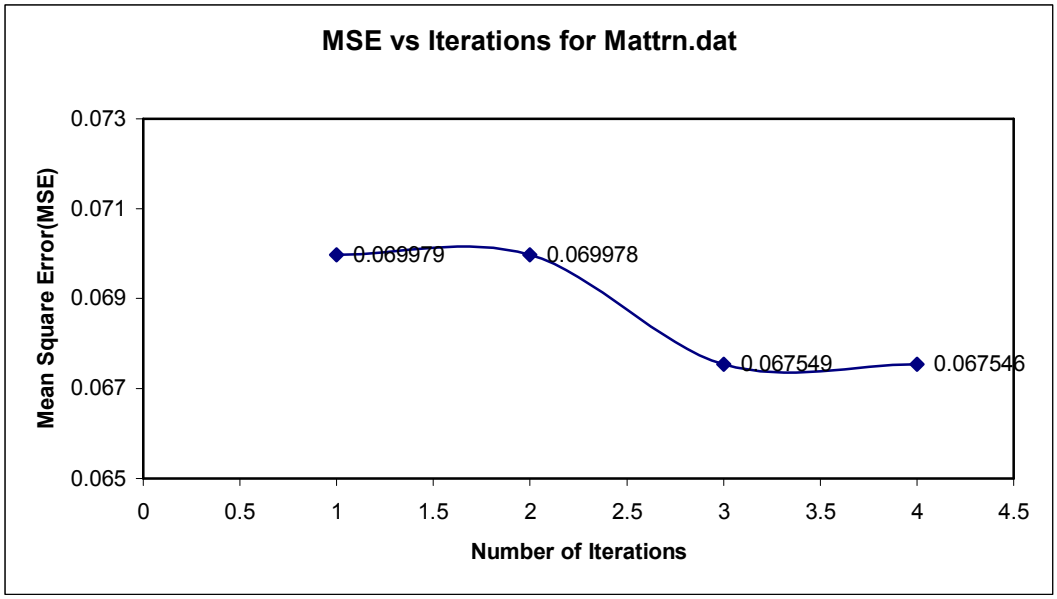

Fig 4.4 Graph For MSE Versus Number Of Iterations For Dataset mattrn.dat

3. Dataset f17.dat

The FLS data was obtained from the M430 flight load level survey conducted in Mirabel Canada in early 1995 by Bell Helicopter Textron. The input features, which are measurements available in the cockpit, include:

(1) CG F/A load factor,

(2) CG lateral load factor,

(3) CG normal load factor,

(4) pitch attitude,

(5) pitch rate,

(6) roll attitude,

(7) roll rate,

(8) yaw rate,

(9) corrected airspeed,

(10) rate of climb,

(11) longitudinal cyclic stick position,

(12) pedal position,

(13) collective stick position,

(14) lateral cyclic stick position,

(15) main rotor mast torque,

(16) main rotor mast rpm,

(17) density ratio,

(18) F/A acceleration, transmission,

(19) lateral acceleration, transmission,

(20) vertical acceleration, transmission,

(21) left hand forward pylon link,

(22) right hand forward pylon link,

(23) left hand aft pylon link, and

(24) right hand aft pylon link.

The desired outputs are the helicopter loads which are the strains on mechanical parts. These are not available in a typical helicopter, but need to be estimated throughout the flight so that mechanics know when to replace critical parts. During test flights, the loads are measured by strain gauges which are temporarily attached to the critical parts. The desired output loads are as follows:

(1) fore/aft cyclic boost tube oscillatory axial load (OAL),

(2) lateral cyclic boost tube OAL,

(3) Collective boost tube OAL,

(4) Main rotor (MR) pitch link OAL,

(5) MR mast oscillatory perpendicular bending smoothed time average (STA),

(6) MR yoke oscillatory beam bending STA,

(7) MR blade oscillatory beam bending STA,

(8) MR yoke oscillatory chord bending STA, and

(9) Resultant mast bending STA position

Three training data files were produced. File F17.dat used features 1 through 17 and all nine output loads and this is the file which has been used for the simulation. Thus in this case, we will be iterating 17 times over all the inputs and use 5 versions of the distance measures. After all the iterations, the PLN is redesigned and we notice a decrease in the overall MSE. The graph for the simulation is shown below

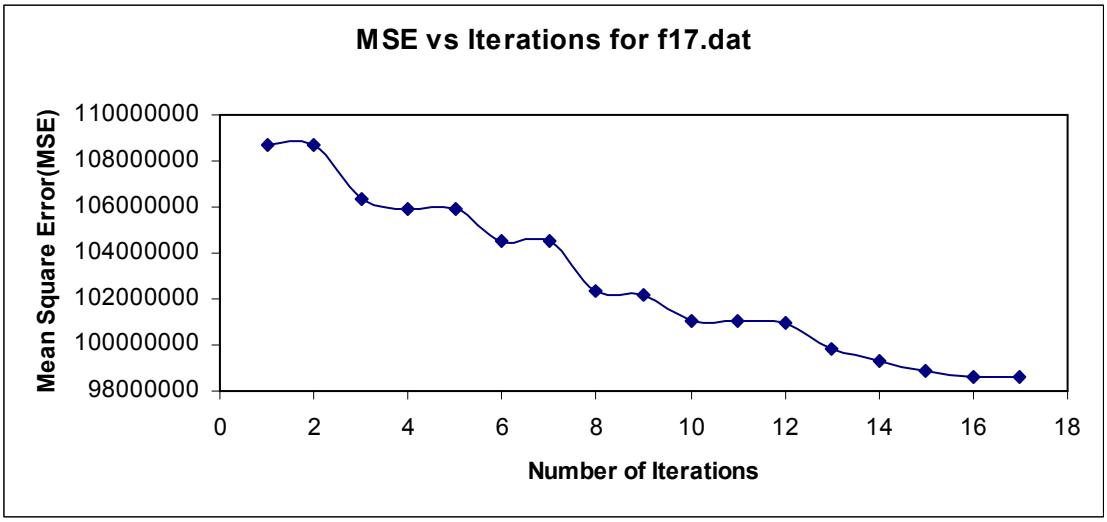

Fig 4.5 Graph For MSE Versus Number Of Iterations For Dataset f17.dat

4. Dataset twod.tra

We have 8 outputs and 5 versions of distance measures. It 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 1768 patterns.

Thus we will be iterating 8 times, with 5 weighted distance measure versions. This results in a decrease in the MSE of the overall network.

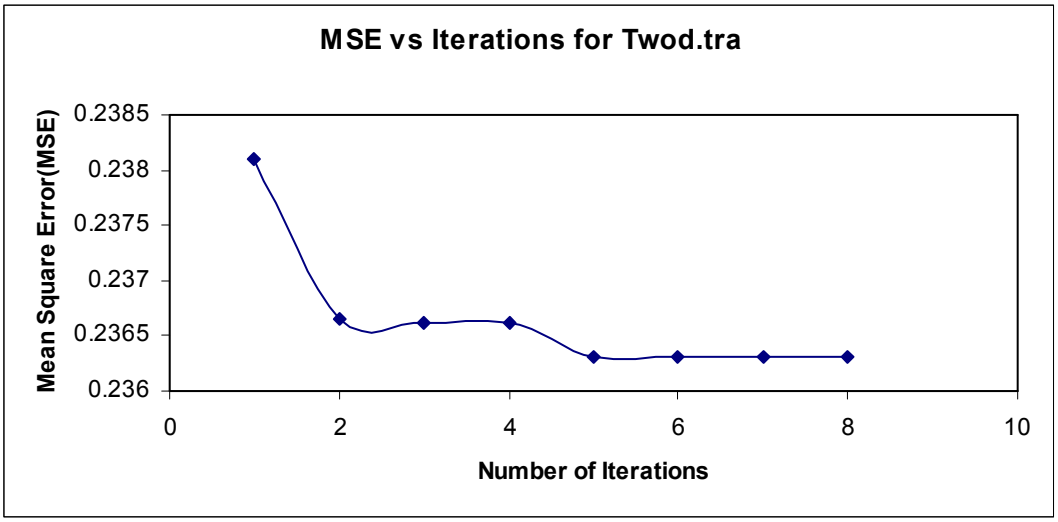

Fig 4.6 Graph For MSE Versus Number Of Iterations For Dataset Twod.tra

In the above graph, the MSE of the PLN versus the number of iterations is shown. It is observed that after redesigning the PLN the overall MSE decreases.

4.4 Deletion Of Clusters

As described in section 2.4 , sequential leader algorithm(once or a couple of times to get more number at least those many number of clusters wanted by SOM. Different data files act differently, hence this needs to be taken care of) is carried out to get the initial number of clusters. When $K_{Leader} > K$, some clusters are deleted, according to some heuristic functions to bring the number of clusters down to K , which is the same as that is pre defined in SOM.

The goal is to delete those clusters which show up to be insignificant i. e. MSE does not increase with the deletion of those clusters. Also this is done in so that the user can choose the number of clusters (K) .

In order to do this total number of patterns in each formed cluster is calculated. Also the cumulative error of each cluster is calculated.

We delete the clusters according to the following four ways till we get only those many clusters used in SOM clustering.

1. Keep clusters which have the lowest number of patterns, thus keeping k number of clusters.

2. Keep clusters which have the highest clustering error, thus keeping k number of clusters.

3. Keep clusters which have the lowest clustering error, thus keeping k number of clusters.

4. Keep clusters which have the highest number of patterns, thus keeping k number of clusters.

Where k is the number of clusters used in SOM clustering. Finally after deleting those clusters, the matrix is recalculated to redesign the Piecewise linear network using all the four above mentioned ways and compared in order to find which one gives the minimum Mean Square Error(MSE).

4.4.1 Simulation Results From Deleting Clusters Formed In Sequential Leader

The simulation results are calculated in MS-C++. The PLN is designed after deleting those clusters according to the heuristics given in the left hand column. It was seen that Sequential leader formed clusters about 8 to 10. For data files which formed less than 5 clusters, the threshold was optimized to form at least that many number of clusters given by the user.

Table 4.1 For Dataset Twod.tra: Comparison Of MSE For Several Heuristics To Delete The Number Of Clusters Formed

	MSE of PLN
Keeping clusters with lowest # of patterns	0.326347
Keeping clusters with highest clustering error	0.319930
Keeping clusters with lowest clustering error	0.290054
Keeping clusters with highest # of patterns	0.259001

Total number of clusters = 5

Table 4.2 For Dataset oh7.tra: Comparison Of MSE For Several Heuristics To Delete The Number of Clusters Formed

	MSE of PLN
Keeping clusters with lowest# of patterns	2.789474
Keeping clusters with highest clustering error	2.649127
Keeping clusters with lowest clustering error	2.529347
Keeping clusters with highest # of patterns	2.509768

Total number of clusters = 5

Table 4.3 For Dataset mattrain.dat: Comparison Of MSE For Several Heuristics To Delete The Number Of Clusters Formed

	MSE of PLN
Keeping clusters with lowest # of patterns	0.174455
Keeping clusters with highest clustering error	0.187269
Keeping clusters with lowest clustering error	0.186015
Keeping clusters with highest # of patterns	0.162988

Total number of clusters = 5

Table 4.4 For Dataset fmtrain.dat: Comparison Of MSE For Several Heuristics To Delete The Number Of Clusters Formed

	MSE of PLN
Keeping clusters with lowest # of patterns	0.026009
Keeping clusters with highest clustering error	0.025989
Keeping clusters with lowest clustering error	0.025564
Keeping clusters with highest #of patterns	0.023971

Total number of clusters = 5

It can be concluded from the above tables that designing the PLN by keeping the clusters with the highest number of patterns result in a decrease in the MSE of the PLN while doing the same

by keeping those clusters with the lowest number of clusters results in a hike in the MSE. Also keeping those clusters which have the highest total clustering error result in the MSE to be comparably high.

4.5 Comparison Of SOM Clustering And Sequential Leader Clustering Based On The Computational Complexity

Different clustering algorithms have different requirements; speed may be the top priority in a real-time application while accuracy could be of prime importance in another. In order to find the best suited clustering technique for a particular application, performances of different kinds of techniques can be compared. The basis for comparison studied here is the number of multiplies taken for the clustering algorithms used to design a Piecewise Linear Network. In this section, we compare the number of multiplies in a Piecewise Linear Network (PLN) using the Self Organizing Maps (SOM) clustering and the Sequential leader Algorithm (SLA)

For SOM Clustering, we consider designing a trained PLN that clusters $\mathbf{x}_p$ input vectors into N_c clusters. It first assigns one input vector to one of the N_c clusters , using a distance measure and then designs the network by fitting a linear network though every cluster in order to find the total Mean Square Error(MSE) of the whole network.

It takes N multiplies to calculate the distance measure given in eq. (1.1)

Since a total of N_c clusters exist, the test vector's distance from each of these clusters has to be measured. This result in a total of N_c.N multiplies. Once the test vector is assigned to a particular cluster, the corresponding $\mathbf{A}_k$ matrix is activated to calculate the output vector y, according to the formula

$$\mathbf{y}_p = \mathbf{A}_k . \mathbf{x}_p$$

This calculation, in turn, requires (N+1) N_c .M multiplies. Hence, the total number of multiplies required to design a PLN using SOM clustering is given by:

$M(PLN)_{SOM}$ = N_c . N + (N+1). N_c .M (4.14)

On the other hand, during Sequential Leader clustering, we first calculate a threshold value which is the "sum" of the distances of all N_v input patterns $\mathbf{x}(n)$ from the overall mean $\mathbf{m_n}(n)$ of the patterns. The distance used is given in eq. (1.1). which we calculate for all N_v patterns $\mathbf{x}(n)$, each of which are N dimensional , from the overall mean $\mathbf{m_n}(n)$, which is also N dimensional.

This requires a total of N.N multiplies.

After the threshold is calculated, the first $\mathbf{x}(n)$ is initialized as the first cluster means. Henceforth all the rest $\mathbf{x}(n)$ patterns forms new cluster if its distance from the previously formed closest cluster mean is greater than the threshold , but the patterns belong to a previously formed cluster if the computed distance is smaller than the threshold. Hence N_{cSLA} numbers of clusters are formed. This distance is given by eq (1.1)

Hence this computation requires N. N_{cSLA} number of multiplications

If the PLN is designed using these N_{cSLA} numbers of clusters, solving for the $\mathbf{A}_k$ matrix would require (N+1). N_{cSLA} number of multiplications.

Thus for the whole PLN design we have

$M(PLN)_{SLA}$ = N.N + N. N_{cSLA} + (N+1). N_{cSLA} (4.15)

number of multiplications.

But many a time, the user would like to have some specified number of clusters denoted by N_c

In this case some clusters are deleted to match the number of clusters formed in SLA. So for designing the PLN, SOM is carried out to re-assign the patterns according to the N_c number of clusters. Thus the total number of multiplications for this is

N. N_c + (N+1). N_c from eq (4.14)

Thus the total number of multiplications for the whole PLN would be

$M(PLN)_{SLA}$ = N.N + N. N_{cSLA} + N. N_c + (N+1). N_c

CHAPTER 5

PRUNING THE PIECEWISE LINEAR NETWORK

As was discussed earlier global function approximators like the Multilayer perceptron (MLP) had several performance problems and hence local networks like the Piecewise Linear Networks (PLN) are chosen. There might arise performance problems with the PLN which are discussed in this chapter and an improved design of the PLN is presented.

5.1 Design Of The Piecewise Linear Networks (PLN)

The Piecewise Linear Networks (PLN) is designed as discussed in details in chapter 3.

5.2 Network Pruning

After a basic PLN is designed with a given number of clusters N_c, pruning is done in order to make the network more compact, by removing those modules/clusters whose elimination causes the least increase in MSE. Optimal pruning of k clusters from a PLN with a total of N_c clusters involves calculating all possible combinations of the PLNs and then finding the minimum MSE. This is done by pruning one module at a time till we are left with only one cluster.

5.2.1. Pruning the Piecewise Linear Networks (PLN) only one cluster at a Time.

The target is to identify and prune the least useful cluster in a completely trained network without significantly degrading the mapping error performance. The algorithm is discussed as in the following steps:

1. We have the input $\mathbf{x}_p$ and $\mathbf{t}_p(m)$ where $1 \le n \le N$ and $1 \le m \le M$. Also we have the number of clusters k where $1 \le k \le N_c$. Previously saved from the PLN training is the $\mathbf{A}_k$ matrix for all the N_c clusters.

2. Let $k_{\min}$ be the index of the cluster to be eliminated and E be the MSE of the network after the particular cluster has been deleted.

Initialize E = 0 and $\mathbf{E}(k)$ = 0

3. For pattern p = 1 to N_v,

i. Find 2 closest clusters k = i, j closest to the pattern $\mathbf{x}_p$ where i and j denote the first and second closest clusters respectively.

ii. Calculate $E_1 = \| \mathbf{t}_p - \mathbf{A}_i \mathbf{x}_p \|^2$ (5.1)

$$E_2 = \| \mathbf{t}_p - \mathbf{A}_j \mathbf{x}_p \|^2 \quad (5.2)$$

Where $\mathbf{A}_i$ and $\mathbf{A}_j$ are the $\mathbf{A}$ matrices associated with the ith and the jth cluster respectively.

iii. For k = 1 to N_c,

accumulate the errors $\mathbf{E}(k)$ as the following

$$\mathbf{E}(k) \rightarrow \mathbf{E}(k) + E_1 \text{ for k} \neq 1 \quad (5.3)$$

$$\mathbf{E}(k) \rightarrow \mathbf{E}(k) + E_2 \text{ for k = 1} \quad (5.4)$$

4. Find $E_{\min} = \mathbf{E}(k)$ where $E_{\min}$ the smallest error is. The cluster to be eliminated is $k_{\min}$ and hence this cluster is pruned.

5. Pass the data through the pruned network, re-clustering and recalculating the $\mathbf{A}_k$ matrix, thus redesigning the PLN.

6. The whole process above is continued till it ends up with only one luster (all patterns belong to this one), redesigning the PLN in every iteration.

Every time a cluster is deleted, and the patterns corresponding to that cluster are re-assigned to the remaining ones, a linear mapping is fitted to every cluster in order to design the PLN. Thus this design causes the network to become more compact

The flowchart for the above algorithm is shown below. This final Piecewise Linear Network is compared to the basic linear mapping network and the results are shown in chapter 6.

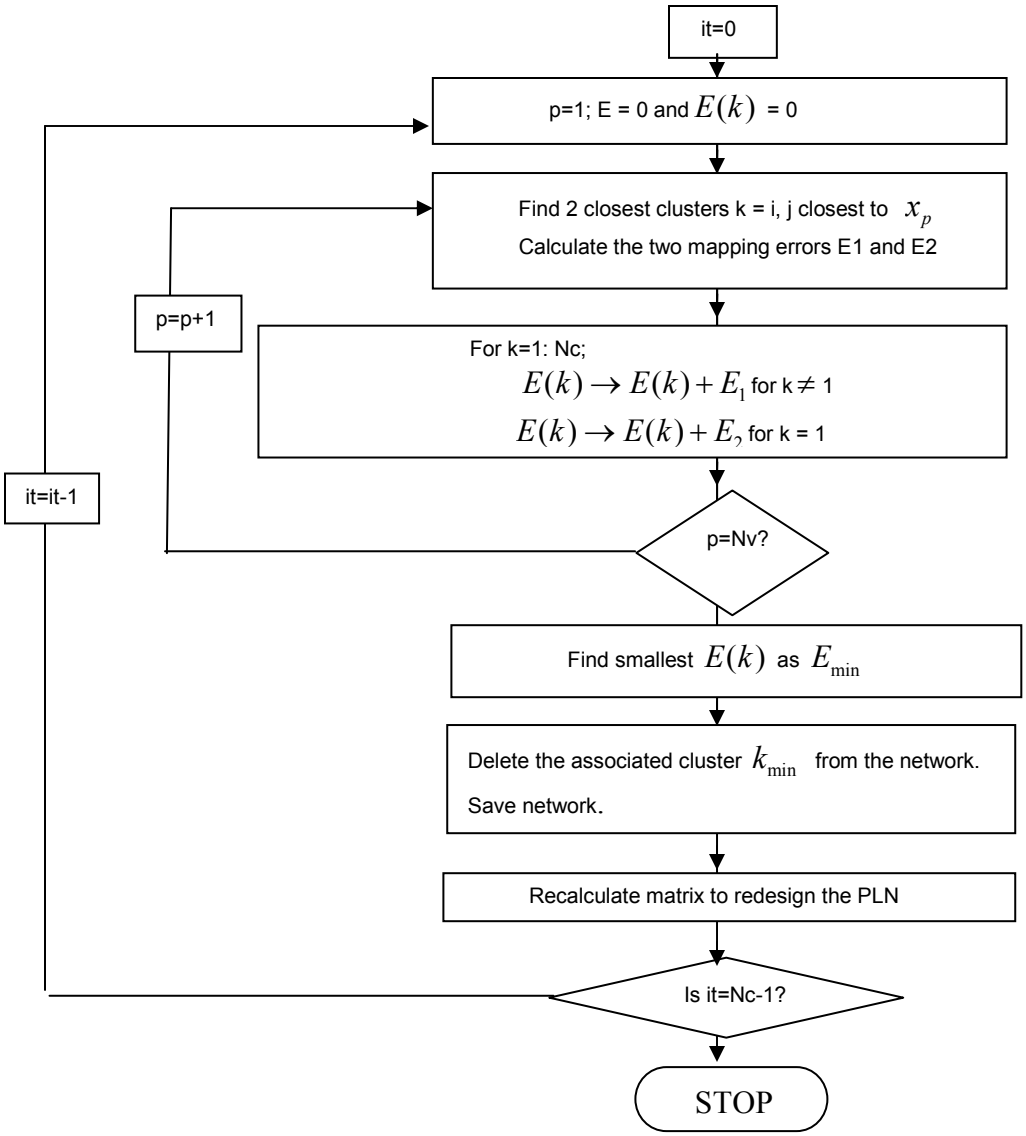

Fig 5.1 Flowchart For The PLN Pruning Algorithm

5.3 An Efficient Version Of The Pruning Algorithm

In redesigning the Piecewise Linear Network (PLN) by pruning the least significant clusters in every iteration / pass though the data, we first write the patterns into separate files according to the cluster it belongs. For example, if a particular pattern belongs to the second cluster, it would be written into the file with index 2. Thus if the n^{th} pattern belongs to the k^{th} cluster, then the pattern would be written into the k^{th} file. The autocorrelation matrix **R** and the cross correlation matrix **C** are calculated by reading from those cluster files. The $\mathbf{A}_k$ matrix is calculated thus designing the PLN.

When a cluster is deleted, we re-cluster the patterns belonging only to the deleted cluster and append them to remaining cluster file. The **R** and **C** matrices are read from the remnant files recomputing the $\mathbf{A}_k$ matrix and thus re-design the PLN.

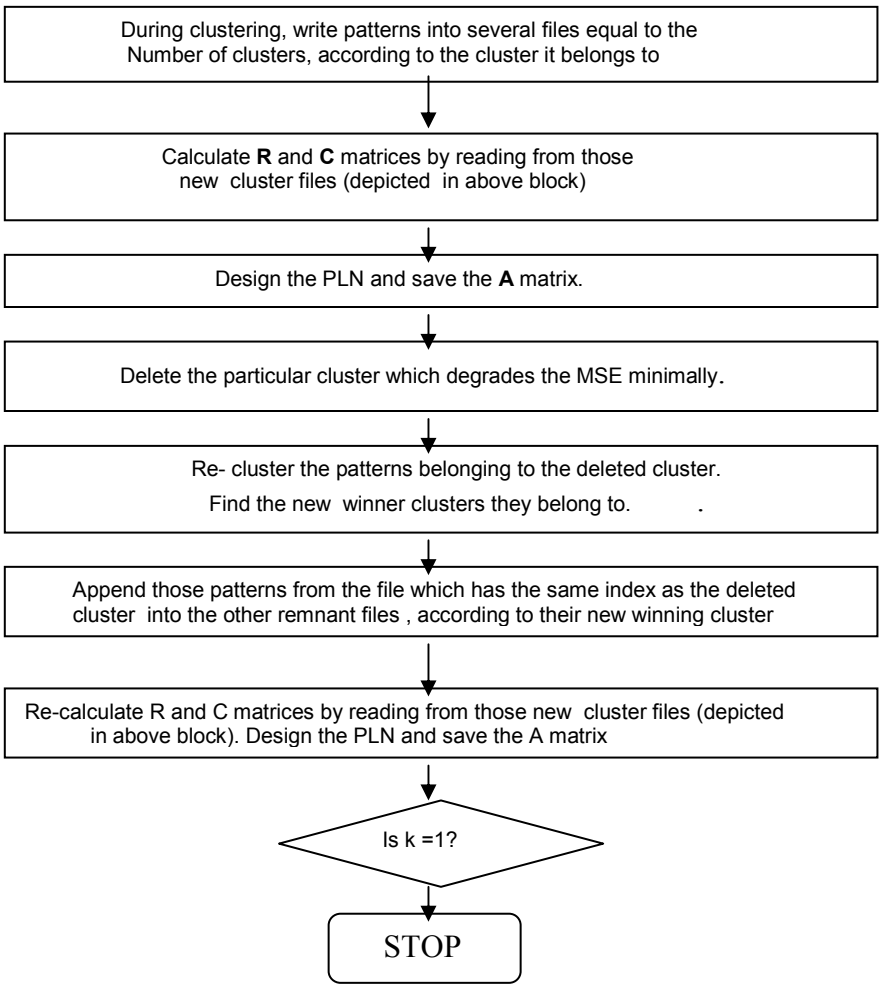

Fig 5.2 Flowchart For The PLN Pruning Algorithm (IMPROVED)

The whole algorithm is described in details in the following steps:

1. . We have the input $\mathbf{x}_p(n)$ and $\mathbf{t}_p(m)$ where $1 \le n \le N$ and $1 \le m \le M$. Also we have the number of clusters k where $1 \le k \le N_c$ .During the basic design of the PLN we write the input patterns into several N_c number of data files, depending on the cluster input $\mathbf{x}_p(n)$ belongs to.
2. The **R** and **C** matrices and the $\mathbf{A}_k$ matrix is calculated for all N_c clusters. Saved the $\mathbf{A}_k$ matrices for all the N_c clusters.
3. Deletion of clusters is done as shown in steps 2 to 4 in the section 5.2.1 , given by eq . (5.1) to (5.4).
4. The cluster $k_{\min}$ is pruned.
5. Recluster the patterns belonging only to the $k_{\min}$ cluster, from the $k_{\min}$ cluster file. This is denoted by $N_v(k_{\min})$.
6. Write those $N_v(k_{\min})$ patterns into the remnant N_c -1 files
7. Recalculate the **R** and **C** matrices of the clusters and also the corresponding $\mathbf{A}_k$ matrix for the remnant clusters .
8. Steps 3 to 7 are done till we are left with only one cluster and all the N_v patterns belong to that cluster.

   Three situations may occur in this case and needs to be taken care of:

   i. During Re-clustering all the patterns go to only one cluster among the remnant ones , then we will re-calculate the **R** and **C** matrices and also the $\mathbf{A}_k$ matrix of that cluster

   ii. During Re-clustering all the patterns go to some clusters out of the remnant ones, then we will re-calculate the **R** and **C** matrices and also the $\mathbf{A}_k$ matrix of those clusters.

iii. When the patterns go to all the remaining clusters, the **R** and **C** matrices and the corresponding $\mathbf{A}_k$ matrices of all the remnant clusters need to be re-calculated.

On implementing the PLN designed as described above, we find that the computation becomes more efficient as the R and C matrices are directly read from the files into which the patterns are written according to the cluster it belongs. The results are shown in chapter 6.

5.4 Comparison Of The Two Pruning Algorithms

For comparing the two versions of the pruning algorithm, we calculate the computational complexity of each algorithm based on the number of multiplies required and applied to one test vector $\mathbf{x}(n)$.

5.4.1 The First Version Of Pruning Algorithm

In this version, during the design of the basic PLN we calculate the distances given by eq. (1.1) for every input vector $\mathbf{x}(n)$ from N_c cluster means and find the shortest distance among them. Here the number of multiplies are N. N_c. We have N_v patterns, the total number of multiplies required are $N_v\ N_c$ N.

Once the test vectors are assigned to particular clusters, the $\mathbf{A}_k$ matrix is calculated using the formula $\mathbf{y}_p = \mathbf{A}_k.\mathbf{x}_\mathrm{p}$. And the PLN is designed.

This calculation requires N_c. (N+1).M multiplies.

Now we delete the cluster which is the most insignificant one. In doing this we accumulate errors E_1 and E_2 defined earlier. Such calculation requires (M+1) $N_v\ N_c$ multiplies.

This is repeated till we are left with only one cluster to design the PLN. Hence the total number of multiplies required for the whole design of the PLN and the pruning algorithm is given in the following equation

$$N_v N[\frac{N_c.(N_c-1)}{2}]+N_v(M+1)[\frac{N_c.(N_c-1)}{2}-1]+(N+1)M[\frac{N_c.(N_c-1)}{2}]$$

Thus we can say the complexity of the algorithm is in terms of

$O(N_v.N.N_c^{\,2})+O(N.M.N_c^{\,2})$.

5.4.2 The Second Improved Version Of The Pruning Algorithm

In this version, the total number of multiplies required in designing the basic PLN is the same as above. Also the cluster is deleted in the same way as in the basic design and hence requires the same number of multiplies.

These two parts for the whole design and pruning results in

$$N_v(M+1)[\frac{N_c.(N_c-1)}{2}-1]+(N+1)M[\frac{N_c.(N_c-1)}{2}]$$

But after deletion of every cluster, we re-cluster the patterns into the other remaining clusters and also write the patterns from that deleted cluster/file into those remnant cluster data files according to the patterns new winner cluster.

Let the number of patterns which belong to the deleted cluster be $Nv[k]_{it}$, where $it=0:N_c-1$. Also $Nv[k]_{it}$ << N_v where N_v is the total number of patterns in the given data file. Thus in this case only $Nv[k]_{it}$ distance measures need to be calculated for the re-assignment of these patterns. Hence the total number of multiplies will be smaller that that

required for computation of distance for all the N_v patterns in the dataset in each iteration and thus the total number of multiplies will be decreasing .

Even it is noticed during simulation , that the run time is smaller and hence this version of the pruning software is a lot more efficient.

5.5 Equivalent MLP Based On Multiplies

When the trained PLN network is employed to process one input vector into an output vector, it requires $2.N.N_c + N.M$ multiplies. In contrast a single hidden layer fully connected MLP requires $N(N_h + M) + N_h.M$ multiplies and N_h nonlinear sigmoid function calculations. On equating the number of multiplies required to process an input vector, the equivalent MLP network is found as $N_h = \frac{2.N.N_c}{N + M + 2}$. We assumed that each sigmoid function calculation takes time as much as two multiplies.

CHAPTER 6

SIMULATION EXAMPLES AND RESULTS

Training and Pruning results simulated on various data sets are shown below. The algorithms were implemented in MS C++. MSE from NuMap is compared to the MSE from the designed software. M is the total number of outputs.

1. Inversion of the surface scattering parameters: Dataset twod.tra

This training file consists of 8 inputs and 7 outputs and 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 1768 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.

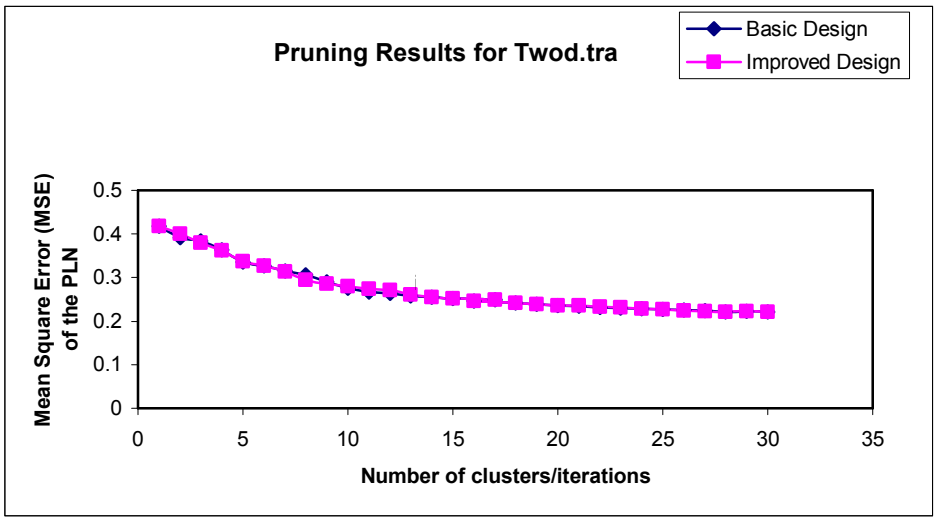

Fig 6.1 Pruning results for Dataset Twod.tra

2 Matrix inversions: Dataset mattrain.dat

This training file consisting 4 Inputs, 4 Outputs, 2000 Training Patterns 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.

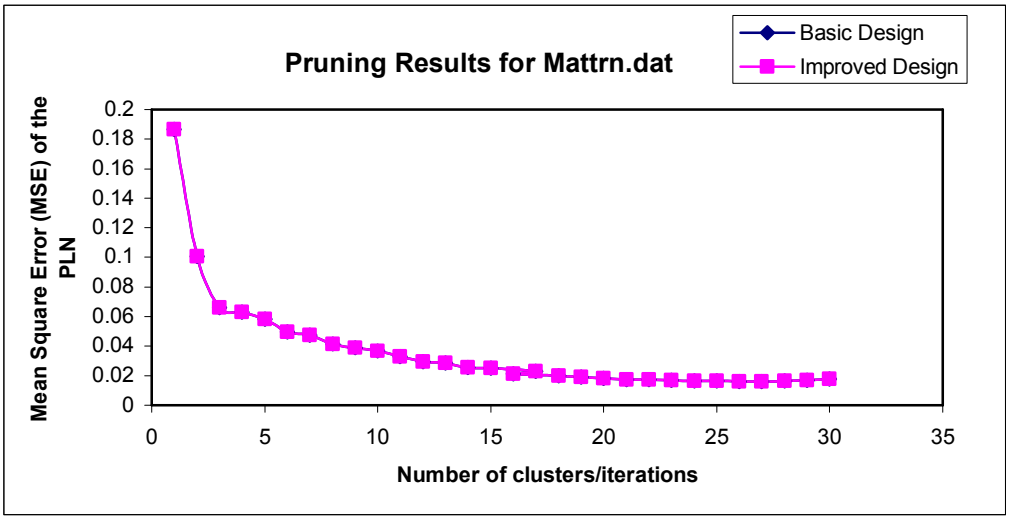

Fig 6.2 Pruning results for Dataset mattrain.dat

3. Demodulation of a Frequency modulated signal: Dataset fmtrain.dat

This training file consisting of 5 inputs, 1 output and 1024 training patterns trains 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) = Camp * cos[2* PI* n* Cfreq + Mamp *sin(2* PI* n* Mfreq)]

where Camp = Carrier Amplitude, Mamp = Message Amplitude, Cfreq = normalized Carrier frequency, Mfreq = normalized message frequency. In this data set, Camp = .5, Cfreq = .1012878, Mfreq = .01106328, and Mamp=5. The five inputs are r(n-2), r(n-1), r(n), r(n+1), and r(n+2). The output is Cos (2* PI* n* Mfreq). In each consecutive pattern, n is incremented by 1.

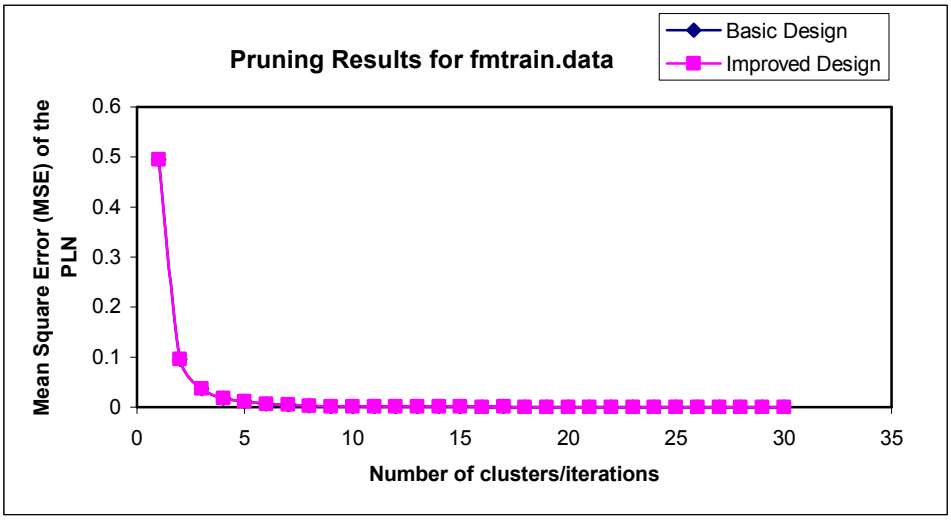

Fig 6.3 Pruning results for Dataset fmtrain.dat

4.Rx/Tx Polarization: Dataset oh7.tra

The training set file consists of 20 Inputs, 3 Outputs and 15,000 Training Patterns. It collects 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.

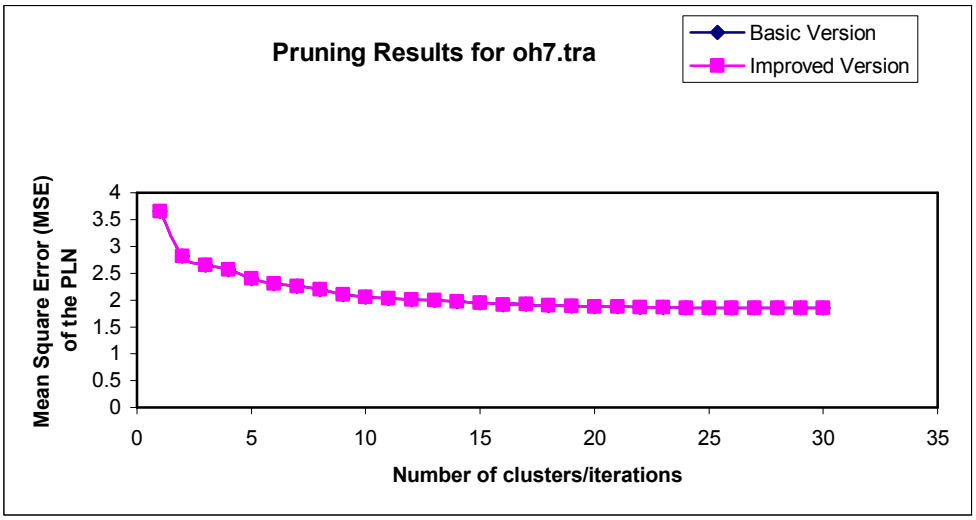

Fig 6.4 Pruning results for Dataset oh7.tra

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

In this thesis, we have dealt about the improved design of the Piecewise Linear Network (PLN) and two algorithms to make the network more compact. Also Sequential leader algorithm (SLA) is implemented to cluster the patterns in the data and PLN is designed. It has been shown that SLA gives improved performance than doing the clustering using SOM.

7.2 Suggested Future Work

1) In order to solve the linear network weights per cluster, Gram Schmidt orthonormalization can be implemented.

2) While designing the PLN using various versions of the weighted Euclidean Distance measure, the minimum number of iterations can be equal to M. W where M is the number of outputs per pattern and W is the number of distance measure versions. The number of iterations can be optimized by changing the W or using X.W versions, where X is a non-negative integer. That is to say, instead of one iteration per distance measure version, we can use X number of iterations.

3) A PLN classifier can be designed using SLA and the performance can be studied. Also other networks which require clustering of the input patterns, SLA can be implemented and the results could be studied.

REFERENCES

[1] Warren McCulloch and Walter Pitts, “A Logical Calculus of Ideas Immanent in Nervous Activity”, Bulletin of Mathematical Biophysics, 1943, 5:115-133.

[2] Simon Haykin, Neural Networks: A Comprehensive Foundation, Prentice Hall PTR, 1998.

[3] Christopher M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, 1995.

[4] Minsky, M. and Papert, S., Perceptrons, MIT Press, Cambridge,1969

[5] P. Werbos, “Beyond regression: New tools for prediction and analysis in the behavioral sciences,” Ph.D. dissertation, Committee on Appl. Math., Harvard Univ., Cambridge, MA, Nov.1974.

[6] K. Fukushima, "Cognitron: A Self-Organizing Multilayered Neural Network”, Biological Cybernetics, Vol. 20, No. 3/4, NOV. 1975, pp. 121-136.

[7] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," Proc. N&l. Acud. Sci., vol. 81, pp. 3088- 3092, 1984.

[8] T. Kohonen, Associative Memory: A System-Theoretical Approach, Springer- Verlag, Berlin, 1977.

[9] S. Grossberg, "Adaptive Pattern Classification and Universal Recoding, I: Parallel Development and Coding of Neural Feature Detectors", Biological Cybernetics, Vol. 23, 1976, pp.121-134.

[10] Hema Chandrasekaran, Jiang Li, W.H. Delashmit, P.L. Narasimha, Changhua Yu, Michael T. Manry, "Convergent design of piecewise linear neural networks", Neurocomputing, vol. 70, pp. 1022–1039, 2007

[11] Jiang Li, Michael T. Manry, Pramod L. Narasimha, and Changhua Yu, "Feature Selection Using a Piecewise Linear Network", IEEE Transactions on Neural Network, vol. 17, no. 5, September 2006, pp. 1101 – 1115

[12] Hema Chandrasekaran and Michael T. Manry, "Convergent Design of a Piecewise Linear Neural Network", Proceedings of IJCNN'99

[13] Tae Kim, M. T. Manry, J. Maldonado, "New learning factor and testing methods for conjugate gradient training algorithm", Proceedings of the International Joint Conference on Neural Networks, July 2003, vol. 3, pp. 2011 - 2016.

[14] A. Papoulis, Probability, Random Variables, and Stochastic Processes, McGraw- Hill Book Company, New York, 1965.

[15] R. Battiti, “First- and second-order methods for learning: Between steepest descent and Newton’s method,” Neural Computation,1992, vol. 4, pp.141-166.

[16] D. G. Luenherger, Linear and Nonlinear Programming, 2nd ed., MA: Addison-Wesley, 1989.

[17] Gilbert Strang, Introduction To Linear Algebra, Wesley-Cambridge Press, 1993

[18] R. O. Duda, P. E. Hart, D. G. Stork, Pattern Classification, 2nd Ed,Wiley Interscience, 2000.

[19] K. Fununaga, Statistical Pattern Recognition, 2nd Ed., Academic Press, NY, 1990.

[20] M.D. Srinath, P.K. Rajasekaran, An Introduction to Statistical Signal Processing With Applications, John Wiley and Sons, 1979.

[21] V. Vapnik, The Nature of Statistical Learning Theory, Springer, 1995

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

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

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

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

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

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

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

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

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

[31] Rui Xu and Donald Wunsch II, IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 16, NO. 3, MAY 2005, Survey of Clustering Algorithms

[32] Christopher F. Eick, Nidal Zeidat, and Zhenghong Zhao, Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004), Supervised Clustering – Algorithms and Benefits

[33]Andreas Rauber, Jan Paralic, Elias Pampalk, Work supported by the Austrian Institute for East- and Southeast Europe as part of the CLUE Project, Empirical Evaluation of Clustering Algorithms.

[34]Glenn Fung, A Comprehensive Overview of Basic Clustering Algorithms

[35] Hartigan, J.A, Clustering Algorithms

[36] Hartigan, J.A, Clustering Algorithms with Probability and Mathematical Statistics

[37] Xiao Liu and Tulay Adalz, Recurrent Canonical Piecewise Linear Network for Blind Equalization.

[38]S. Subbarayan, K. Kim, M. T. Manry, V. Devarajan, and H Chen, ”Modular Neural network architecture using piecewise linear mapping”, Thirtieth Asilomar Conference on Signals, systems & Computers ,vol 2, pp 1171-1175, Nov 1996.

[39] Hema Chandrasekaran, PhD Dissertation, May 2000, Analysis and convergent Design of Piecewise Linear Networks.

[40]W. Li, J.N. Lin and R. Unbehauen,” Canonical representation of Piecewise polynomial functions with non degenerate linear domain partitions”, IEEE Transactions on Circuits and Systems i: Fundamental theory and Applications, vol 45, no8, pp838-848, Aug 1998

[41] W. H Press, B.P. Flannery, S.A. Teukolskky, Numerical Recipes in C: The Art of scientific computing.

[42] Gurney, Kevin, An Introduction to Neural Networks.

[43] Orr, Mark J.L, Introduction to Radial Basis function Networks.

[44] Stern, T. E, Piecewise Linear Network Theory.

[45] Bors, A.G, Introduction to Radial Basis Function Networks.

[46] Kuhn, Simone and Cruse, Holk, Mental representation and cognitive behavior – a recurrent neural network approach.

[47] Rainer W. Paine, Jun Tani, Motor primitive and sequence self-organization in a hierarchical recurrent neural network.

[48] Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability", J. B. Macqueen, Some Methods for classification and Analysis of Multivariate Observations.

BIOGRAPHICAL INFORMATION

Sayantani Pal received her BSEE degree from North Bengal University in 2003 and is currently pursuing her MS degree from University of Texas at Arlington.

She was involved in several projects in mobile games and applications development with Mobile Technologies (Kolkatta, India) from 2003 to 2005.She is currently with the Image processing and neural network laboratory (IPNNL) in University of Texas at Arlington. Her research interests include the theories of several artificial neural networks and machine learning and their applications in various fields, specifically robotics.