THESIS

NON-LINEAR PRINCIPAL COMPONENT ANALYSIS AND CLASSIFICATION OF EEG DURING MENTAL TASKS

Submitted by

Saikumar Devulapalli

Department of Computer Science

In partial fulfillment of the requirements

for the degree of Master of Science

Colorado State University

Fort Collins, Colorado

Summer 1996

COLORADO STATE UNIVERSITY

July 3, 1996

We hereby recommend that the thesis NON-LINEAR PRINCIPAL COMPONENT ANALYSIS AND CLASSIFICATION OF EEG DURING MENTAL TASKS prepared under our supervision by Saikumar Devulapalli be accepted as fulfilling in part requirements for the degree of Master of Science.

Committee on Graduate Work

Committee Member

Committee Member

Adviser

Department Head

ABSTRACT OF THESIS

NON-LINEAR PRINCIPAL COMPONENT ANALYSIS AND CLASSIFICATION OF EEG DURING MENTAL TASKS

This thesis explores the effectiveness of Non-Linear Principal Component Analysis (NLPCA) as a technique for reducing the dimensionality of human electroencephelogram (EEG) for enabling it to be classified into two different metal tasks. EEG signals from a single subject recorded through six channels was studied during the performance of two mental tasks. An NLPCA network was used to reduce the dimensionality of temporal windows of eye-blink free EEG data. A standard backpropagation network was used to classify the reduced dimensionality representation of the original data. The results indicate that the NLPCA method is able to extract distinguishing features from the data that could be classified as belonging to one of the two tasks with an average percentage accuracy of 86.22%. The NLPCA method is found to be a definite improvement over its linear counterpart, the Karhunen-Loève transform, in extracting important features from EEG signals for their classification. The work presented here is part of a larger project whose goal is to be able to classify EEG signals belonging to a varied set of mental activities to investigate the feasibility of using different mental tasks as an alphabet for communication by a physically handicapped person.

Saikumar Devulapalli
Department of Computer Science
Colorado State University
Fort Collins, Colorado 80523
Summer 1996

ACKNOWLEDGEMENTS

This work has been funded by NSF Grant IRI-9202100. I thank Dr. Charles Anderson for guidance and encouragement I received throughout this project. Special thanks to Dr. Michael Kirby whose research lead us to pursue this project. I also thank all my friends and family for the motivation and support they extended to me.

DEDICATION

To my dear parents whose faith in me is unshakable.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

This chapter briefly describes a few methods for dimensionality reduction. The procedure involved in recording EEG data and the structure of the recorded data is dealt with. Then, a brief explanation of the NLPCA technique as a preprocessing step for reducing the dimensionality of the EEG data is given followed by a description of the classification experiments done on the reduced dimensionality representation. The results obtained in classification are given followed by an overview of the remaining chapters.

1.1 Dimensionality Reduction

The analysis and classification of signals involves extracting significant features from the signals which lend themselves to easier interpretation. Dimensionality reduction as a preprocessing step for classification of signals using a neural network could be more effective than trying to classify unprocessed signals because smaller networks can be used and it leads to easier classification [3]. The Karhunen-Loève (K-L) transform performs linear dimensionality reduction by determining the first n principal components. This can be viewed as determining a linear mapping from the original data to a n dimensional subspace. However, non-linear dimensionality reduction of signals may result in a mapping to a lower dimensional subspace. Autoassociative neural networks as an effective way of non-linear dimensionality reduction was first proposed by Kramer [15] among others [18], [23], [16] and [21]. An autoassociative network has a mapping network (for reducing data to a lower dimension), demapping network (for mapping the reduced dimensional data to the original dimension), and a layer connecting the mapping and the demapping networks called the *bottleneck layer*. The reduction onto a lower dimension can be obtained by determining the weights in the mapping network. This is done by training the autoassociative network to replicate the input signal. The number of bottleneck nodes in the network is less than the number of inputs and outputs. Therefore, by training the network, we force it to learn a lower dimensional representation of the data. Using sigmoidal units in the mapping and demapping layers, a non-linear mapping onto a lower dimension is obtained.

An enhancement to the autoassociative network for classifying periodic signals was proposed in [12]. In this method, *circular nodes* are used in the bottleneck layer. Circular nodes provide a direct mapping of a signal onto a circular manifold as opposed to sigmoidal nodes which provide a mapping onto an open interval. Therefore, circular nodes could be very effective in dimensionality reduction of signals which have non-linear components that are homeomorphic to a circle.

Another problem in reducing the dimensionality of a signal using a bottleneck architecture is the likelihood that bottleneck nodes learn the same feature. Previous methods suggested involved pruning the network. On the other hand, the sequential non-linear principal component analysis (NLPCA) architecture introduced in [15] uses a number of small networks each consisting of a single bottleneck node. The networks are connected in a cascade. The first network is trained to replicate the original signal. Therefore, the first network tries to learn the primary feature in the data. The error in replication at the first network is given to the second network as the input data. The second network is trained to replicate this data and its error in replication is passed as input to the third network, and so on. This forces the networks higher in the priority to learn the most significant features. These significant features are filtered (by determining the error in replication in higher priority networks) and this data is supplied to lower priority networks. Therefore, these networks learn less important features because they will need to replicate the data from which the significant features have been filtered. Therefore, sequential NLPCA prioritizes the training of important features among multiple networks.

As a preliminary analysis of the effectiveness of the different techniques discussed above, experiments were conducted using a set of mathematical functions. The set of functions was chosen such that each function was defined in terms of multiple variables (which forms the explicit dimensionality of the function) which were driven by a fewer number of internal parameters (which is the intrinsic dimensionality for the function). The experiments consisted of measuring the replication error using an autoassociative network with sigmoidal and circular bottleneck nodes. The number of bottleneck nodes used was equal to the intrinsic dimensionality of the signal. Circular nodes show better performance in replicating the circle and the trefoil knot.

1.2 EEG Signal Classification Problem

The analysis of EEG signals is driven by the goal of trying to correlate between an external stimulus and the characteristic signals found in the EEG patterns. A larger and more widely applicable method would be to try to determine the mental tasks performed by a subject, from normal EEG data without an external stimulus. Such a goal when achieved, would enable a physically challenged person who has no control over his motor responses to communicate with the outside world based on a set of normal mental activities. Therefore, classification of mental tasks based on normal EEG signals is an important and difficult task to achieve.

This work tries to address the problem of classifying multidimensional EEG signals based on specific tasks by reducing the dimensionality of EEG data and employing this data to classify the signals into one of two different mental activities. The data used in this work was recorded by Keirn and Aunon [10]. In their study, a set of tasks comprising of Baseline (no activity), Mental Arithmetic, Geometric Figure Rotation, Mental Letter Composing and Visual Counting were chosen. EEG signals were recorded by placing electrodes at $O_1, O_2, P_3, P_4, C_3, C_4$, standard electrode positions using a Grass amplifier. Ten sessions were conducted for each task, each session lasting for a period of ten seconds which resulted in 2500 samples per task per session. In the current work, Mental Arithmetic and Mental Letter Composing tasks were chosen for classification based on the assumption that they involve significantly different mental processes.

One set of raw EEG data consists of six dimensions (corresponding to the six channels), with 2500 samples for each dimension. Individual samples of EEG data are very difficult to classify because the temporal correlations in the signals are not utilized in such classification. For this purpose, temporal windows of EEG data are considered. Each window is chosen to be 62 samples long (approximately corresponding to a quarter second) and consecutive windows are overlapped by 31 samples [2]. Since each window consists of 62 samples per channel and there are 6 channels, the dimensionality of the data set is $62 \times 6 = 372$. Raw EEG data contain easily discernible high potential spikes caused due to eye blinks. Eye blinks do not carry any significant features of data and can become the basis of classification. Therefore, sample points falling in the region of eye blinks are removed from the raw EEG data before being subjected to any processing.

The approach followed here is to apply dimensionality reduction as a preprocessing step to windowed EEG data using an autoassociative network consisting of a bottleneck layer having fewer nodes than the input and the output layers. A standard backpropagation algorithm trained with conjugate gradient method is used for faster convergence. The network is trained and tested for replication using various combinations of data belonging to different tasks.

1.3 Classification using Reduced Dimensionality Representation

Once the bottleneck network is trained, the mapping network is detached from the trained autoassociative network and used separately for generating a lower dimensional representation. This representation is used for classification of the tasks using a standard backpropagation network. The classification network would have a number of inputs equal to the reduced dimension and the target values -0.9 and +0.9 corresponding to one of the two tasks into which the signal is being classified. The classification accuracy for the reduced dimensionality representation is compared with that of the K-L representation of the original input vectors.

Bottleneck networks consisting of 10, 20, 30 and 40 nodes in the bottleneck layer were trained using conjugate gradient method. The mapping network was detached after training and was used in generating 10, 20, 30 and 40 dimensional representations of the windowed data vectors. The 30-dimensional representation yielded an average percentage of correctly classified vectors of 86.22% over all the trials. This was followed by the 20-dimensional representation with 76.21%, 40-dimensional representation with 65.83% and 10-dimensional representation with 57.89%. All trial data performed consistently well using the 30-dimensional representation. The input vectors to the classification network (which are the reduced dimensional vectors) as such were uniformly distributed in the n dimensional space ($n = 10, 20, 30, 40$) and the mutual Euclidean distances between the vectors was not directly related to how well they were classified (which would indicate how prominent the distinguishing features are) or which task they belonged to. The classification networks corresponding to all dimensions seem to employ all the hidden unit weights in the network. When the first n best-classified input vectors belonging to both the tasks are projected along the direction of the first two eigenvectors (representing the two orthogonal directions of maximum change in the input vector distribution) onto a two dimensional space, the vectors begin to form independent clusters for each task as the value of n decreases. This indicates that the best classified vectors contain features which strongly differentiate between the two tasks supporting the 86.22% of correctly classified vectors for the 30-dimensional representation. Using a K-L transform representation (using 10, 20, 30 and 40 dimensions) of the original data vectors results in classification accuracy only slightly better than chance indicating that NLPCA method is giving a definite improvement over the K-L transform method in classification of EEG signals.

In all training experiments, performance is measured based on the mean squared error for the test set.

1.4 Overview of the Remaining Chapters

The following is the layout of the rest of the chapters. Chapter 2 gives an extensive description of the background work done in both supervised and unsupervised methods for dimensionality reduction. The reader can skip most of this chapter other than the description of autoassociative networks, circular nodes and NLPCA which are the main approaches employed as part of the present work. Chapter 3 describes the results of preliminary experiments done in reducing the dimensionality of well defined mathematical functions using sigmoidal and circular bottleneck architectures. Chapter 4 describes methodology of classification experiments done using actual EEG data and the procedure in which data sets are chosen from different trails to train and test the bottleneck and classification networks. Chapter 5 discusses the classification results and gives an analysis of the reduced dimensionality representations of the windowed data vectors. Chapter 6 gives the conclusion to the study and the future work in this area.

Chapter 2

BACKGROUND

2.1 Introduction

The study and classification of multidimensional signals involves extraction of important features from poorly known processes. This can be simplified by reducing the dimensionality of the signal. Every n-dimensional signal has an implicit dimension associated with it, which is less than or equal to n. Reducing the dimensionality of a signal makes extraction of features easier. Feature extraction identifies the important properties of a signal which might help in classification of the signal. For example, the rate of arrival of items and the rate of removal of items can be used to describe a queuing system. However, these two values can be replaced by the rate of accumulation of items in the queue. Therefore, in this example only one feature is required instead of two. This reduces the complexity of the system.

In complex systems, the dimensionality of the data is equal to the number of sensors used to measure it. However, the implicit dimensionality is the number of sources in the system. In most cases, no information is provided about the number of sources or their characteristics. Also, the actual application in which data would be used (i.e. the factors that have to be measured using the data) is indeterminate. In these cases, it is not possible to reduce the number of variables by using specific information about the data. For such systems, more general methods (which do not need information about the implicit dimensionality or the applicability of the data) have to be used.

2.1.1 Principal Component Analysis

Principal component analysis (PCA), discussed in [6], reduces the dimensionality of a signal by determining a linear mapping of the signal to a smaller dimensional space.

The feature variables in PCA, called factors, represent linear combinations of original problem variables. The coefficients of this linear transformation are such that if the feature transformation is applied to the data followed by its inverse, there will be minimum sum of squares difference between the original and reconstructed data. If Y represents an $n \times m$ matrix (n = number of observations and m = number of variables), PCA involves determining an optimal factorization of Y into a source matrix T ($n \times f$) and a loading matrix P ($m \times f$) and a matrix of residuals E ($n \times m$) such that

$$Y = TP^T + E. \tag{2.1}$$

Here the dimensionality of the system is reduced from m to f by determining the P matrix and using it to get

$$T = YP. \tag{2.2}$$

Therefore, if the values in the residual matrix E represent permissible errors, then the data set Y consisting of n observations and m variables has been successfully reduced to the data set T consisting of n observations and f variables. By considering values of $f < m$, the dimensionality has been reduced from m to f.

2.1.2 Neural Networks and PCA

One of the first neural network approaches to PCA is the Hebbian learning algorithm [24]. Hebbian learning is a basic unsupervised learning algorithm for feed-forward networks. In Hebbian learning the weight change Δw_{ij}, is proportional to the product of the output, y_i, of the node to which the weight connects and the input, x_j, to that node, i.e.

$$\Delta w_{ij} = \alpha y_i x_j. \tag{2.3}$$

Figure 2.1 shows the network architecture for Hebbian learning with the input, output and the weight values. The output for each node can be given in terms of the I inputs x_i, by

$$y_i = f(\sum_{j=1}^{I} w_{ij} x_j) - 1/2 \tag{2.4}$$

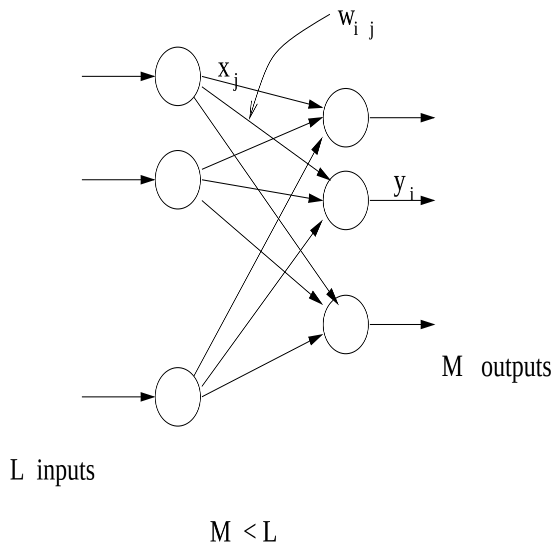

Figure 2.1: Hebbian Network.

where f is the activation function for the network. This algorithm is derived by maximizing an overall response function F given by

$$F = \sum_{i=1}^{N} y_i^2 \tag{2.5}$$

by making the change in w_{ij} proportional to partial derivative of F with respect to w_{ij}, i.e.

$$\Delta w_{ij} \propto \frac{\delta F}{\delta w_{ij}} \tag{2.6}$$

From equation (2.4) we get

$$\frac{\delta F}{\delta w_{ij}} = 2f^{'} y_i x_j \tag{2.7}$$

where $f^{'}$ is the derivative of the activation function f. If no activation function is being used, f will be a linear function and its derivative will be a constant. From equations (2.4) and (2.7) and introducing the learning rate as the proportionality constant, we obtain equation (2.3).

This equation can also be represented in the form of a matrix equation, as given in [17], as

$$W_{k+1} = W_k + \mu_k [I - W_k W_k^T] x_k x_k^T W_k \tag{2.8}$$

where

W_k is a $L \times M$ matrix of weights at the k^{th} iteration,

L is the number of inputs to the network,

M is the number of outputs,

μ_k is the learning rate at the k^{th} iteration,

x_k is the k^{th} input vector.

If $w_k(i)$ is a vector representing the weights connecting to the output node i, the output of that node (assuming linear activation function) is $[x_k^T w_k(i)]$. Therefore, the change in weight is proportional to the product $x_k[x_k^T w_k(i)]$ of the input and output of each neuron. The additive nonlinear term $W_k W_k^T x_k x_k^T W_k$ orthonormalizes the weight vectors. If a sigmoidal nonlinearity is used, the additive term can be given as

$$W_k W_k^T x_k g(x_k^T W_k) \tag{2.9}$$

where g is the sigmoidal function.

It has been proved in [19] that a network that is trained using Hebbian learning will learn the m-dimensional PCA subspace of the input vectors, even though a strict convergence proof is still lacking. Specifically, the weight vectors given in equation (2.8) converge towards the true unnormalized principal eigenvectors of the correlation matrix of the input vectors if a sigmoidal nonlinearity is used in the additive constraint as shown in equation (2.9).

Therefore, a neural network performing Hebbian learning is capable of principal component analysis. This is the fundamental basis on which many other extensions to PCA have been proposed.

PCA gives a linear mapping from the input space to the feature space. However, a linear mapping from the actual signal to a reduced dimension may not exist. A nonlinear mapping may be more beneficial. There are many reasons for this. Principal components are based solely on covariances or correlations. These are second order statistics and can describe Gaussian data and stationary linear processing operations only. However, higher order statistics in data are needed for more accurate feature extraction. These can be introduced by inserting a nonlinearity in the feature extraction. output that result from PCA are often linear combinations of these sub-signals. In some cases, the sub-signals themselves may be desired.

Also, a nonlinear mapping to a much lower dimension might exist. PCA projects the data into a linear subspace with minimum information loss, by multiplying the data by the eigenvectors of the sample covariance matrix. By examining the relative magnitude of the corresponding eigenvalues, the minimum dimensionality of the space into which the data may be projected could be chosen based on the value of permissible error. However, if the data lie on a nonlinear sub-manifold of the feature space, then PCA will over-estimate the dimensionality. For example, the covariance matrix of data sampled from a helix in three-dimensions will have three principal components. However, any point on the helix could be mapped onto an open interval.

2.2 Nonlinear Dimensionality Reduction

Many algorithms have been proposed for nonlinear dimensionality reduction. The algorithms come under two main categories: autoassociative networks, and nonlinear extensions to Hebbian learning. The autoassociative networks use supervised learning and the nonlinear extensions to Hebbian learning use unsupervised learning. A newer technique for signal classification called Independent Component Analysis (INCA) is proposed in [8].

2.2.1 Supervised learning methods

Autoassociative networks were first introduced in [15]. Autoassociative networks are nonlinear feed-forward networks trained using standard back propagation algorithm, where the network is trained to replicate the input signal. The network consists of multiple layers one of which contains fewer nodes than any other layer in the network. This constraint is the bottleneck layer, a layer of hidden nodes smaller in dimension than either the input or output layers. Linear bypasses are allowed in the network between any two layers except across the bottleneck. Therefore, the only communication between the left and the right parts of the network is through the output of these bottleneck nodes. This layer constrains the network to develop a more compact representation of the input at the bottleneck layer.

Conventional techniques for data analysis need a process model which is frequently assumed to be linear. Statistical methods like PCA could be used in the case where a process model does not exist. However, they are based on linear mapping between the m and f dimensional space. Autoassociative networks could be used to study and interpret multidimensional data for which model-based techniques for the data do not exist. Also, standard back propagation with a sigmoidal transfer function could be used to train the network.

In [15], the noise filtering aspect of the autoassociative network is considered. Since the bottleneck is constrained to bring out correlations among the input, the uncorrelated factor of the input (which also consists of uncorrelated noise) is attenuated. During training of an autoassociative network, the training set is approximated using f independent variables. If Γ is the lower dimensional nonlinear manifold for the network, all outputs of the network lie on Γ. The network transforms Y to Γ. If the network has enough representational capability, Γ will appear as a least square fit where the approximations of Y lie very close in the manifold. It has to be also assured that Γ is a smooth fit by cross validation. Given a linear network with k inputs measuring the same quantity, with an added uncorrelated noise whose variance is zero, the noise factor in the output signal is reduced by k times. This is the case only if the error variance is equal for all the input signals. This is similar to the statistical quality control problems. However, for a nonlinear network the noise reduction is difficult to calculate directly. It is however observed in [15] that just as in the linear case, a nonlinear mapping in which the variance of the noise for each input is approximately equal resulted in better noise reduction. The work done in [15] illustrates this by measuring the same signal using two different sensors with some noise induced. A nonlinear network is used to create an identity mapping of the two inputs. A single bottleneck node is shown to be sufficient to replicate the signal. Also, the total noise variance is shown to be reduced for the output signal.

NLPCA using a autoassociative networks was proposed in [14]. The technique is similar to PCA except that a nonlinear vector function is used instead of a linear transformation function. Therefore, we seek a mapping

$$T = G(Y) \tag{2.10}$$

where G is the nonlinear vector function consisting of f nonlinear functions, each corresponding to one variable. If we can determine a vector function H consisting of m nonlinear functions which when applied to T results in a modified data set Y' such that the loss of information given by $E = Y$ - Y' is $\leq$ a prespecified permissible error value, then we can claim that T is a correct representation of Y and that one can be transformed to the other using the nonlinear vector functions G and H. By choosing a value of $f < m$ we can make T lie on a lower dimensional space than m. Therefore, nonlinear principal component analysis consists of determining two nonlinear vector functions to map the data set from a high dimensional space to low dimensional space and vice-versa. In [14], the authors state that the most natural way to come up with such mappings is to use nonlinear neural networks. Therefore, a network consisting of m inputs and f outputs could act as the mapping function and a network consisting of f inputs and m outputs could be used as the demapping function. In order to assign weights to these networks, we need to know the values of the low dimensional data corresponding to the actual data. However, this information is not available. On the other hand, we know that the input to the mapping network and the output of the demapping network are identical. Therefore, the most effective way of assigning weights to the mapping and the demapping networks is to combine them into one network consisting of m inputs and m outputs and training it to replicate the input pattern. The autoassociative network as shown in Figure 2.2 is best suited for this purpose. The use of sigmoidal nodes in the mapping and demapping layers makes the mapping nonlinear. The network can be trained to replicate the input signal. The reduced dimensionality of the signal is equal to the number of bottleneck nodes used. Once the signal is successfully replicated, the network can be dismantled and the mapping network can be used to reduce the dimensionality of similar signals.

The authors show the application of NLPCA by training the network on a two-dimensional input signal representing a circle using a single sigmoidal node in the bottleneck. The input variables were given by

$$y_1 = 0.8\sin(\theta); \qquad y_2 = 0.8\cos(\theta); \qquad (\theta = [0, 2\pi]) \tag{2.11}$$

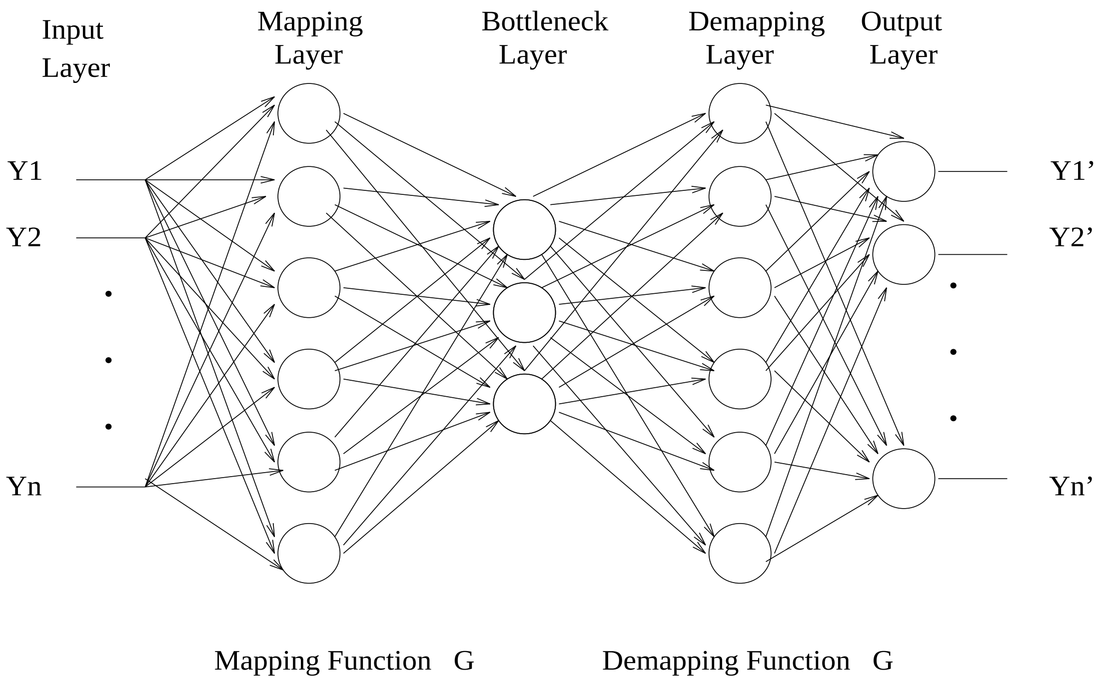

Figure 2.2: Basic bottleneck architecture.

The data set consisted of 100 data vectors. Since both the input variables are driven by a single underlying parameter, the network should be able to model the data with one nonlinear factor. Three methods were applied to this data to determine the best one-factor representation: PCA, an autoassociative network with no mapping and demapping nodes (ANN-1HL), NLPCA network using a single sigmoidal bottleneck node and two thru ten mapping and demapping nodes. It is illustrated that both PCA and ANN-1HL resulted in output along one of the diagonals on the circle whereas the NLPCA method resulted in the output that closely followed in the input except for a few vectors corresponding to the beginning and end of the $[0, 2\pi]$ interval. In another application for NLPCA, data from a simulated batch reactor in which four simultaneous first order reactions occur, is considered. A batch consists of one set of reactions where a raw material A gives rise to products T, U, R and S (corresponding to one reaction each). All four reactions are guided only by two initial conditions : the initial temperature T_0 and the value of an impurity α. Each batch of reactions ran for 30 minutes during which the concentration of the desired product R is measured for 100 times. 25 such batches of reactions were conducted. Since each batch consisted of a temporal window of 100 measurements of concentration of the desired product R, the superficial dimensionality of the data is 100. However, since each batch is governed by precisely two initial conditions T_0 and α the implicit dimensionality of each batch is two. The three methods of PCA, ANN-1HL and NLPCA were applied to reduce the dimensionality of the data to 1 and 2 variables. The PCA and ANN-1HL are shown to perform almost identically, whereas the NLPCA is shown to result in error an order of magnitude less than the PCA and ANN-1HL.

NLPCA using the bottleneck architecture forces the network to learn a compact representation of the input at the bottleneck layer. Since the network is highly symmetric, each node in the bottleneck tries to learn principal features in the data independently. No effort is made to learn features in a complimentary way. For example, two nodes in the bottleneck layer might try to learn the same features in the data. This results in duplication of effort and the effective number of bottleneck nodes is reduced by one. Sequential NLPCA tries to solve this problem by assigning hierarchies to the nodes in the bottleneck layer. This procedure can be called a nonlinear extension of the recursive procedure of factor calculation used in linear PCA.

In the recursive procedure for factor calculation, the secondary factor $p_2(Y_1)$ of the original data matrix Y_1 is the primary factor $p_1(Y_2)$ of the residual matrix $E_1 = Y_2$. Therefore extending the rule to the first i factors:

$$p_i(Y_1) = p_{i-1}(Y_2) = p_{i-2}(Y_3) = ... = p_1(Y_i) \tag{2.12}$$

Therefore, the factors of the data matrix can be obtained as follows :

- The primary factor is extracted from the data matrix (using eigenvector analysis).
- This is subtracted from the data matrix to obtain the residual matrix.
- The primary factor is extracted for the residual matrix and becomes the secondary factor for the data matrix.

This procedure is repeated until the first n factors for the data matrix are determined.

The sequential NLPCA given in [14] is similar to this procedure. The first nonlinear component of the data is learned using an autoassociative network with only one bottleneck layer. The output from the bottleneck has only the principal component of the input.

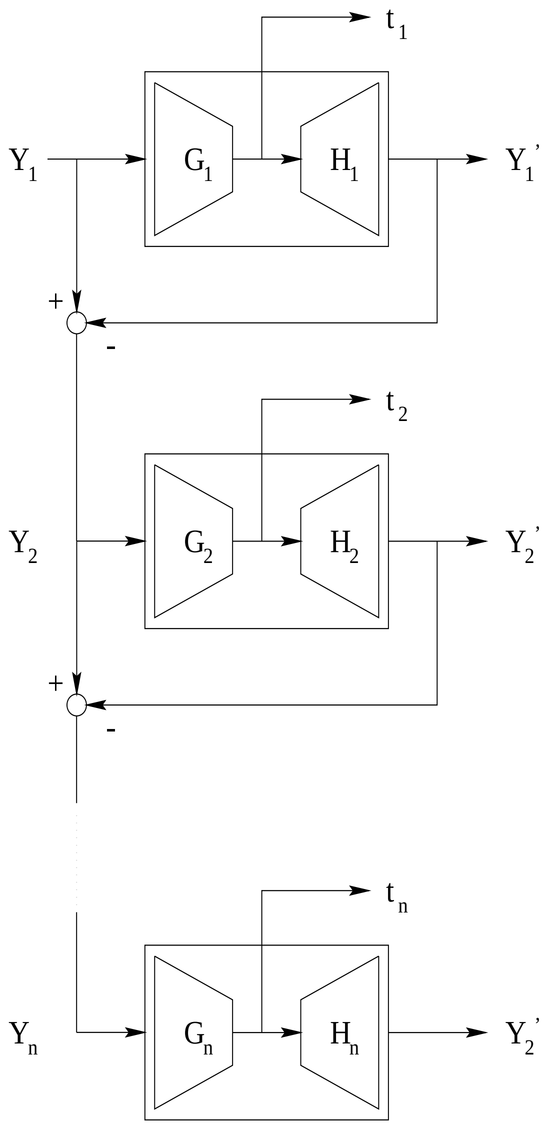

Figure 2.3: Sequential Determination of Nonlinear factors.

This is subtracted from the input signal and the residual signal forms the input to the next autoassociative network which learns the second nonlinear principal component. The process is repeated for the desired number of nonlinear components needed. In this method, the nodes are forced to learn the components in a specific order. The network architecture for sequential NLPCA is shown in Figure 2.3. Sequential NLPCA tries to eliminate the contention among nodes to learn the primary features. Each node is forced to learn a specific factor. The residual inputs get progressively smaller for higher orders. Therefore, each residual input can be rescaled to lie within a specific interval. Rescaling magnifies the values of the residuals and eases the learning of higher factors which may expedite convergence. There are however, some potential drawbacks in sequential NLPCA. Using the residual matrix to determine the next eigenvector is acceptable in sequential PCA because the feature extraction involves only linear transformations. However, extending the same principal to its non-linear counterpart may result in an ill-defined solution. Also, there is no criterion enforced in sequential NLPCA which forces the learned features to be mutually orthogonal.

The method proposed in [14] tries to train a bottleneck network with a given number of bottleneck nodes. Therefore, the minimum dimension to encode the signal has to be determined by trying various dimensions. However, ideally, the minimum dimensionality of the encoding should be provided by the data itself. In [3], a method of pruning the network is proposed in which a greedy algorithm successively eliminates bottleneck units based on low variances in the output of the units. The training begins with a prespecified number of bottleneck units which should be more than the implicit dimensionality of the system (otherwise the network will never train) and small enough so that the dimensionality could be reduced quickly. A reasonable choice for the initial number of bottleneck units would be a number slightly smaller than the explicit dimensionality of the signal. Once this number is chosen, the bottleneck network is trained to replicate the signal. During training, the node with the least variance in the output is considered for pruning. The learning rate is increased for this node and its effect on the training error is studied. If the error can be maintained at an acceptable level at the same time decreasing the variance for a bottleneck node beyond a threshold, then the bottleneck node can be removed out of the network. This reduces the dimension of the signal by one. This way an attempt is made to reduce the dimensionality by studying the effect of reduction of variance of outputs of bottleneck layer on the squared error until any further reduction of variance of output results in unacceptable error.

The authors demonstrate their method on various examples. The pruning of the bottleneck layer resulted in the mapping of a helix onto a one-dimensional manifold. PCA on the other hand resulted in no reduction in dimensionality. In other experiments done in [22] and [11], an 8-bit, 64×64 pixel image was considered for reduction. This image can be assumed to be a point in 4096 dimensional "pixel space". The data is preprocessed by reduction to the first 50 principal components of the image. The reduced representations were processed by using a bottleneck network consisting of 30 units in encoding and decoding layers, an initial representation layer of 20 units and a bottleneck layer of five units. This five-dimensional data was used to train a feed forward network to recognize the identity of the subjects. 120 images were used in training and 40 for testing. The paper reports a classification accuracy of 98% on the training set and 95% on the test set.

A study was conducted in [4] on dimensionality reduction of biological neuron models which belong to the class of dissipative dynamical systems, using bottleneck neural networks. The authors state that trajectories of dissipative dynamical systems usually fall into some low-dimensional manifold of the state space after some transient time. In these cases, it is convenient to define a lower-dimensional coordinate systems and a compact form of the vector field on this manifold. The paper discusses experiments in which a four-dimensional Hodgkin-Huxley model was successfully reduced to two dimensions and a six dimensional bursting neuron model was reduced to three dimensions. The authors analyze the reduced dimensional signal by graphically investigating the structure of the reduced models.

2.2.2 Circular nodes in neural networks

The previous studies indicate that given an n-dimensional signal, a neural network can be used to force a mapping onto an m-dimensional manifold ($m \leq n$) corresponding to m outputs from the bottleneck layer. Essentially, each output corresponds to a one-dimensional manifold. Conventionally, an open interval is used as the one-dimensional manifold by having sigmoidal/linear nodes in the bottleneck layer. However, open intervals do not have the property of periodicity, i.e., they provide a one-one mapping from a point on a n-dimensional signal to a point within the open interval. If the n-dimensional signal is periodic, then any two points on the signal which are very close to each other but are off by a period are mapped to totally unrelated points on the open interval. This is because a mapping onto an open interval does not allow points separated by a multiple of the period to be mapped to the same point. In order to overcome this problem, a circular manifold was suggested in [12] in order to map a periodic signal onto one dimension.

More specifically, it is more appropriate to map points on a periodic signal onto a circle so that all points at a distance of one period from each other are mapped onto the same point on the circle. Also, it is not possible to map a periodic signal onto an open interval and extract the signal from the open interval values. Consider a periodic signal that is mapped onto an open interval so that for each point on the signal there exists a point in the open interval. In order to de-map the interval, we need to determine the point on the signal corresponding to a point in the open interval. However, as was mentioned before, all points separated by one phase length on the signal are mapped onto the same point in the open interval. Therefore, the mapping of a periodic signal onto an open interval is irreversible. Hence we will not be able to truly replicate a periodic signal with a bottleneck layer consisting only of nodes with output in the open interval. This problem was encountered in [14] where two-dimensional data consisting of a set of points on the circle was being analyzed using NLPCA with a single sigmoidal bottleneck unit. The vectors corresponding to the beginning and end of the $[0, 2\pi]$ interval for θ did not find a good fit in a single dimension. A circular node is both capable of and required for encoding angular information in a signal.

The concept of circular nodes was first presented in [12]. A circular node which provides mapping onto a circular interval is implemented as a pair of coupled nodes so that the output of the two nodes have values that are constrained to lie on a unit circle. If $N_j^{(i)}$ is the jth node in the ith layer of the network, the node coupled to it can be given by $N_{\tau(j)}^{(i)}$. Therefore, the function $\tau(j)$ gives the node coupled with the jth node in the same layer. The prestate value of the network at a node is the output of the node at a given time before the activation function is applied. The state value of the network at a node is the output of the node at a given time after the activation function is applied. The prestate value and the state value of the network at node $N_j^{(i)}$ are denoted by $P_j^{(i)}$ and $S_j^{(i)}$ respectively. From the implementation of a circular node, it follows that

$$(S_j^{(i)})^2 + (S_{\tau(j)}^{(i)})^2 = 1 \tag{2.13}$$

A variation to standard back-propagation which imposes the above constraint is used to train the network. The forward propagation is similar to that of the Standard back-propagation, with an activation function that enforces the constraint in equation (2.13). Therefore, the prestate value of the network at node $N_j^{(i)}$ is given by the formula

$$P_j^{(i)} = b_j^{(i)} + \sum_{k=0}^{N^{(i-1)}-1} w_{kj}^{(i-1)} S_k^{(i-1)}, \tag{2.14}$$

where $b_j^{(i)}$ is the bias at node $N_j^{(i)}$, $w_{kj}^{(i-1)}$ is the weight connecting the nodes $N_k^{(i-1)}$ and $N_j^{(i)}$, $S_k^{(i-1)}$ is the state value of the node $N_k^{(i-1)}$.

The state value of a node can be determined from its prestate value based on whether the node is sigmoidal or circular as follows:

$$S_j^{(i)} = \begin{cases} P_j^{(i)} / \sqrt{(P_j^{(i)})^2 + (P_{\tau(j)}^{(i)})^2}, & N_j^{(i)} \text{ is circular;} \\ \sigma_j^{(i)}(P_j^{(i)}), & N_j^{(i)} \text{ is sigmoidal,} \end{cases} \tag{2.15}$$

where $N_{\tau(j)}^{(i)}$ is the node coupled with $N_j^{(i)}$ and $\sigma_j^{(i)}$ is the activation function use for the jth node in the ith layer.

The term $\sqrt{(P_j^{(i)})^2 + (P_{\tau(j)}^{(i)})^2}$ in the above equation can be treated as the *radial value* at that node denoted by $R_j^{(i)}$.

The equations for the error propagation can be determined in the conventional way for standard back-propagation by equating the change in weight to the partial differential of the error with respect to that weight using equation (2.15) as the activation function. The total squared error for the network is given by

$$E = \frac{1}{2} \sum_{j=0}^{N_{L-1}-1} (S_j^{(L-1)} - G_j)^2 \tag{2.16}$$

where L is the number of layers in the network. The partial derivatives of the weights (and biases) with respect to E can then be derived resulting in

$$\frac{\delta E}{\delta b_j^{(i)}} = \begin{cases} \frac{\delta E}{\delta S_j^{(i)}} \frac{(P_{\tau(j)}^{(i)})^2}{(R_j^{(i)})^3} + \frac{\delta E}{\delta S_{\tau(j)}^{(i)}} \frac{-P_j^{(i)} P_{\tau(j)}^{(i)}}{(R_j^{(i)})^3}, & \text{if } N_j^{(i)} \text{ is circular;} \\ \frac{\delta E}{\delta S_j^{(i)}} \sigma'^{(i)}_j (P_j^{(i)}), & \text{if } N_j^{(i)} \text{ is sigmoidal ,} \end{cases} \tag{2.17}$$

and

$$\frac{\delta E}{\delta w_{jk}^{(i)}} = \begin{cases} [\frac{\delta E}{\delta S_k^{(i+1)}} \frac{(P_{\tau(k)}^{(i+1)})^2}{(R_k^{(i+1)})^3} + \frac{\delta E}{\delta S_{\tau(k)}^{(i+1)}} \frac{-P_k^{(i+1)} P_{\tau(k)}^{(i+1)}}{(R_k^{(i+1)})^3}] S_j^{(i)}, & \text{if } N_k^{(i+1)} \text{ is circular;} \\ \frac{\delta E}{\delta S_k^{(i+1)}} {\sigma'}_k^{(i+1)}(P_k^{(i+1)}) S_j^{(i)}, & \text{if } N_k^{(i+1)} \text{ is sigmoidal,} \end{cases} \tag{2.18}$$

where $\frac{\delta E}{\delta S_j(i)}$ is the partial derivative of total squared error with respect to the state value of the network at node $N_j^{(i)}$ given by

$$\frac{\delta E}{\delta S_j(i-1)} = \begin{cases} S_j^{(i-1)} - G_j, & \text{if } (i = L); \\ \sum_{k=0}^{N^{(i)}-1} \frac{\delta E}{\delta S_k^{(i)}} \frac{\delta S_k^{(i)}}{\delta S_k^{(i-1)}}, & \text{if } (i < L)\ , \end{cases} \tag{2.19}$$

where

$$\frac{\delta S_j^{(i)}}{\delta S_k^{(i-1)}} = \begin{cases} \frac{(P_{\tau j}^{(i)})^2}{(R_j^{(i)})^3} w_{kj}^{(i-1)} + \frac{-P_j^{(i)} P_{\tau(j)}^{(i)}}{(R_j^{(i)})^3} w_{k\tau(j)}^{(i-1)}, & \text{if } N_j^{(i)} \text{ is circular;} \\ {\sigma'}_j^{(i)}(P_j^{(i)}) w_{kj}^{(i-1)}, & \text{if } N_j^{(i)} \text{ is circular,} \end{cases} \tag{2.20}$$

Using these equations, a network consisting of circular and sigmoidal nodes (each corresponding to a couple of nodes) can be trained using training and testing sets. In [12], the authors discuss various applications of circular nodes. Circular nodes can be used in compressing periodic data by training the network to determine a one-to-one mapping from a periodic locus Γ onto a circle. Functions which have both periodic and aperiodic components could be subject to feature extraction by determining a mapping from the pattern set Γ to $I^n \times (S^1)^k$ using a neural network consisting of sigmoidal nodes (which provide a mapping of the amplitude features of the signal to a set of n interval coordinates) and circular nodes (which provide a mapping of the angular features of the signal to the set of k circular coordinates).

Circular nodes naturally fit into the NLPCA architecture proposed in [14] which is used for non-linear reduction in dimensionality. By using circular nodes in the bottleneck layer of the NLPCA network, the network can be constrained to learn a non-linear mapping onto a circular manifold. This could lead to very effective reduction of dimensionality of signal which have components that are intrinsically homeomorphic to a circle. In [12], bottleneck networks consisting of circular nodes in the bottleneck layer are used in reducing the dimensionality of various loci whose implicit dimensionality is less than the number of variables by which the loci are defined. These experiments are discussed in detail in the next chapter.

Another study was done in [1] where a complex mapping network uses the phase information in signals consisting of complex valued data for their nonlinear classification. In this study, a network was proposed in which all the weights, biases and outputs from each layer are complex numbers. A learning algorithm is presented for complex data using back propagation. However, the output of the nodes are complex numbers and do not necessarily lie on a circle. Therefore, each output has a two-dimensional manifold in the complex plane. The study was however confined only to signals consisting complex valued data.

A very similar set of experiments were conducted in [7] where a complex perceptron with complex weights and autocorrelation associative memories was introduced to represent phase information. The complex perceptron was obtained by replacing the weighting coefficients of the conventional one layer perceptron by complex numbers. The discriminant power of the complex perceptron is nearly twice that of the normal perceptron because linear separability associated with the normal perceptron corresponds to quadratic separability by the complex perceptron. A learning rule based on delta rule is proposed where complex numbers are used for learning rate and error. It is proposed that the complex component in the weights learn the phase information in the signal. Experiments were done using bit patterns into one of the two classes (yielding 1 and 0). The percent correct classification was found to be more for complex perceptrons.

The above studies were all related to the application of autoassociative bottleneck networks using sigmoidal and complex nodes for nonlinear dimensionality reduction.

2.2.3 Unsupervised learning methods

A different approach that has been followed by others is based on a nonlinear extension to the standard Hebbian learning. Some of the approaches followed are presented here.

Sanger [20] introduced a method of reducing the number of hidden units in a network to reduce the number of modifiable parameters and, thereby, reduce the training time. An assumption is made that any function can be approximated as a weighted sum of N nonlinear basis functions

$$f(x) = \sum_{i=1}^{N} c_i \sigma_i(x) \tag{2.21}$$

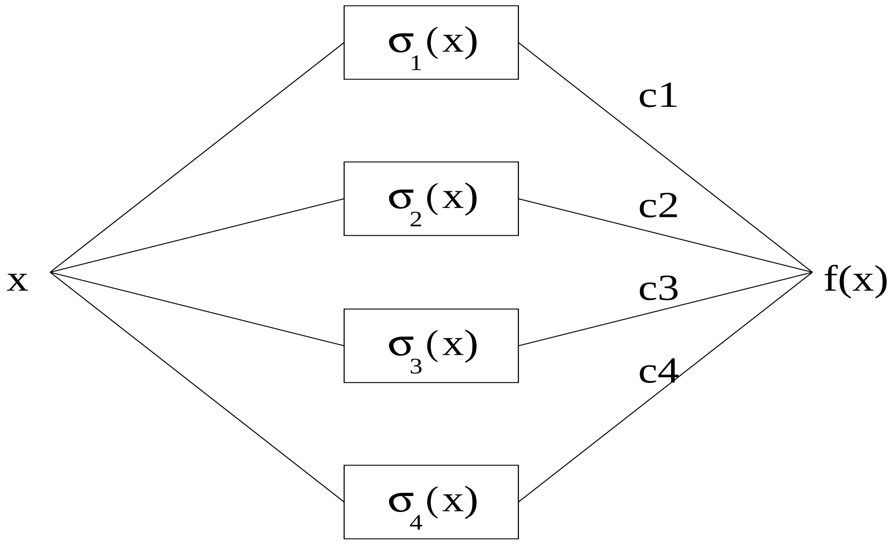

Figure 2.4: The function f represented as a linear combination of nonlinear basis functions $\sigma_j(x)$.

The neural network that evaluates this expression is given in Figure 2.4. The network is assumed to contain a large set of N nonlinear basis functions whose span includes all the functions that might be approximated. This kind of an architecture is suitable if a known function needs to be trained. However, if there are a set of functions that need to be trained or the function that needs to be trained constantly changes, then the use of a small set of features may reduce the learning effort and time. [20] presents an architecture where optimal features of the data are needed before the function can be approximated.

For networks that compute linear functions of their input, optimal features can be defined as those which maximize the mutual information between input and output. The linear principal component analysis helps in evolving such an optimal set of features. A neural network with Hebbian learning can be made to learn the principal components in the data. However, an equivalent learning procedure is lacking for the nonlinear case. In [20], the author defines an optimal set of features for the nonlinear case as one that minimizes the mean-squared approximation error over the distribution of the functions.

Therefore, to determine the optimal features, a large set of N nonlinear basis functions is assumed to exist. The task would be to find a smaller set M of basis functions which is optimal.

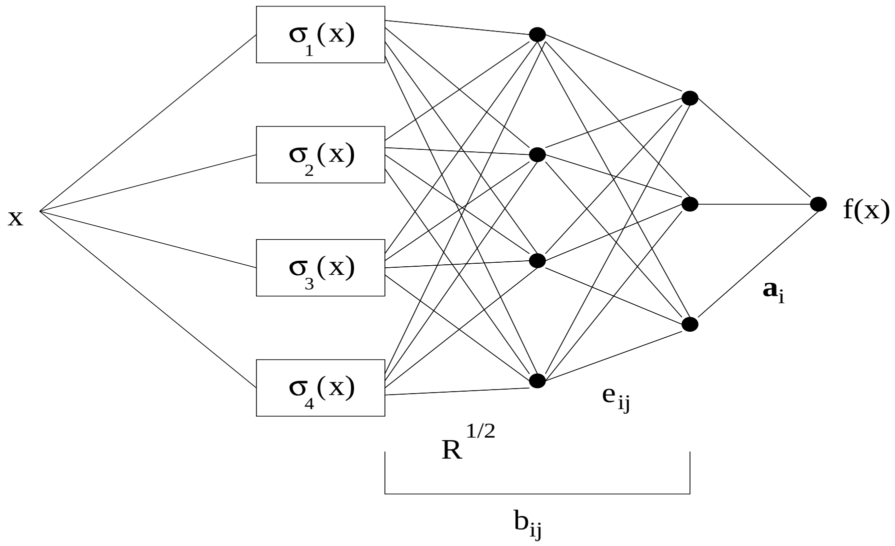

Figure 2.5: The modified two stage network consisting of the prior information matrix R, eigenvectors e and output weights a.

The author proposes a nonlinear unsupervised learning algorithm as an extension to Hebbian learning for extraction of optimal nonlinear features and a supervised learning algorithm for function evaluation. The network architecture proposed is shown in Figure 2.5. A new set of nonlinear basis functions $(\phi_i(x), i = 1, \ldots, N)$ is determined in which the elements are a linear combination of the original basis functions $(\sigma_i(x), i = 1, \ldots, M)$ where $N \leq M$. The weights used in the linear combination are the eigenvectors obtained by training the network using Hebbian learning. Therefore, the set of new basis functions represent the optimal set of features. Once these features are determined, the desired function to be trained can be chosen and the weights of the output layer can be computed by a supervised learning algorithm such as the Widrow-Hoff LMS rule. Therefore, training is separated into two stages: an unsupervised step to learn the features, and a supervised step to learn the output function.

Such two step training actually increases the training time because the original network had N weights whereas the extended network has $NM + M$ weights to be learned. If the output function is known in advance, this technique is more inefficient. However, when the function f is not known in advance, but the input statistics for the network remain constant, the unsupervised portion of the network can be trained without knowledge of the functions. Once a mapping is determined from N basis functions to the M basis functions, the training of any new function (described by the same input statistics) only involves the determination of M weights.

Therefore, the efficiency and usefulness of this method is based on training using a two pass method where the first pass captures the features in the input data and is independent of the function(s) to be trained. The second pass is used to train the network to evaluate one or more functions using fewer weights than the conventional method.

The paper presents an example of the inverse kinematics problem for a two joint planar robot arm which moves in the $x - y$ plane. Given the (x, y) coordinate of the tip of the arm, the angles subtended at the two joints has to be determined. The neural network therefore consists of two inputs and outputs. The end point positions are coded using 64^2 basis functions evenly spaced in the work space with 64 each in x and y directions.

The generalized Hebbian learning technique was used to reduce the number of basis functions to 16. Therefore, before evaluating the inverse kinematic function, the 16 primary features in the input data are extracted. Now the function can be evaluated for a fixed basis function width for each arm, using the Widrow-Hoff rule. If the basis function width is changed, the primary features need not be recalculated. Only the supervised learning has to be repeated. Therefore, the number of weights to be trained for every new basis width is only the number of primary features, and not the total number of basis functions in the work space.

When a Hebbian network is trained to learn the component features of the data, the individual nodes should respond to independent components of the input signal for learning the optimal features in the data. A modified Hebbian learning algorithm is presented in [24] where the hidden nodes are discouraged to learn the same features. The paper presents an extension to the Hebbian learning by introducing an algorithm which does gradient descent on the sum of squares of the outputs and an orthogonality constraint. This algorithm can be implemented in a strictly feed-forward multinode network similar to Hebbian learning.

The algorithm is based on the assumption that if the outputs of the network are constrained to be orthogonal to each other, then the features learned will most probably be complimentary. The standard Hebbian learning rule

$$\Delta w_{ij} = \alpha y_i x_j \tag{2.22}$$

can be derived using a gradient descent on the overall response function F given by

$$F = \sum_{i=1}^{N} y_i^2 \tag{2.23}$$

i.e.. the sum of squares of the outputs.

In the proposed algorithm, a set of constraints given by

$$g_{ik} = (y_i y_k)^s = 0 \tag{2.24}$$

is used to constrain the outputs from nodes y_i and y_k to be orthogonal. s is a constant which is chosen to be a small even integer so that the sign of the node activity becomes insignificant.

An algorithm is needed that forces the set of N nodes to respond to distinct components of that input. In other words, a training algorithm has to be evolved that forces the node outputs to be close to orthogonal to each other. Since $g_{ik} = 0$ is the criterion for orthogonality between two nodes i and k, the criterion for orthogonality between any two nodes can be determined by a weighted sum of g_{ik} where the weights are the Lagrange multipliers λ_{ik} [24]. Therefore, a modified response function G given by

$$G = F + \sum_{i=1}^{N} \sum_{k=i+1}^{N} \lambda_{ik} g_{ik} \tag{2.25}$$

gives the measure of orthogonality between all possible pairs of nodes and has to be minimized. To determine the learning rule, a gradient descent is done on G. Therefore

$$\Delta w_{ij} \propto \frac{\delta G}{\delta w_{ij}} \tag{2.26}$$

and

$$\frac{\delta G}{\delta w_{ij}} = f_i^{'}[2y_i x_i + S(y_i)^{S-1} x_j \sum_{k \neq i} \lambda_{ik} (y_k)^S] \tag{2.27}$$

Therefore

$$\delta w_{ij} = \alpha y_i x_j [1 + (S/2)(y_i)^{S-2} \sum_{k \neq i} \lambda_{ik} (y_k^S)] \tag{2.28}$$

Using the smallest positive even integer value for S (=2) makes the constraint g_{ik} independent of the sign of the node activity and also simplifies the above equation to

$$\delta w_{ij} = \alpha y_i x_j [1 + \sum_{k \neq i} \lambda_{ik}(y_k^2)] \tag{2.29}$$

The Lagrange multipliers have now to be determined. We can assume that all λ_{ik}'s are equal because we have no reason to expect some pair of outputs to be less mutually orthogonal than others (because of the symmetry in the network).

The authors state in the paper that a value of -4 for all λ_{ik}'s results in a simple form symmetric to the second order in y_i for δg_{ik}. Also, the effect of change in weights on each δg_{ik} is uniform because of a constant λ_{ik}. By substituting this value in the above equation, we get

$$\delta w_{ij} = \alpha [y_i x_j][1 - 4 \sum_{l \neq i} y_l^2] \tag{2.30}$$

which is the competitive Hebbian learning rule.

Also, a limit on the weights can be imposed in order to make the learning more effective. For example, a constraint like

$$\sum_{j=1}^{L} w_{ij}^2 \leq Constant \tag{2.31}$$

where L is the number of input nodes makes sure that all the weights leading to a particular node do not exceed a particular value simultaneously.

The effectiveness of competitive Hebbian learning was demonstrated in the paper using the height of points on a regular 10 x 10 lattice as input. For each input vector, a random point is chosen to posses the maximum height, which is considered as the center of a Gaussian spot and the elevation of other points is determined using their distance from the center of the Gaussian.

Experiments illustrate that with competitive Hebbian learning, the multiple hidden nodes learned to share the input space effectively. The use of a single Hebbian node resulted in an output distribution with maximum height at the center of the 10 x 10 grid and minimum height along the edges indicating that the network has generalized on the data set of random Gaussian spots. With two hidden nodes, the network learned to divide the input space along the two diagonal axes (which are orthogonal) of the input square. For a network with three hidden nodes, the three nodes have maximum response at three different points in the input square which are separated by a large distance.

An image compression application is also illustrated where an image is divided into 8 x 8 pixel squares each of which is given as an input vector to the neural network. The network is trained using the competitive learning rule given in equation (2.30). The network consists of 0 - 63 nodes. The 8 x 8 image can be reconstructed after the output values are calculated using the equation

$$\hat{x}_j = C \sum_{i=1}^{N} w_{ij} y_i \tag{2.32}$$

A network with 0 nodes does not undergo training. The average pixel value of the 8 x 8 block is returned as output. A network with one node undergoes standard Hebbian training (because there is no competition among nodes). The paper illustrates that a network with more than one node results in better reconstruction using the competitive Hebbian learning.

By far the most generalized treatment of extensions to Hebbian learning is given in [9]. This paper discusses a nonlinear extension to Hebbian learning. According to the standard Hebbian learning as given in equation (2.8) the weight changes is proportional to the product $[x_k^T w_k(i)]x_k$ of input and output of each neuron. The additive nonlinear term $W_k W_k^T x_k x_k^T W_k$ orthonormalizes the weight vectors.

The terms used here have the following meaning:

- W_k is a $L \times M$ matrix of weights at the k^{th} iteration,
- $w_k(i)$ gives the i^{th} row of W_k, i.e. set of weights going into the i^{th} output node.
- L is the number of inputs to the network,
- M is the number of hidden nodes,
- x_k is the k^{th} input vector.

This algorithm was first presented in [17] where it is proved that a network which is trained using this algorithm learns the linear principal components of the data. A nonlinear extension to the algorithm is given by the same authors in [19] where the following three kinds of extensions are proposed.

$$W_{k+1} = W_k + \mu_k[x_k x_k^T W_k - W_k W_k^T x_k g(x_k^T W_k)] \tag{2.33}$$

$$W_{k+1} = W_k + \mu_k[I - W_k W_k^T] x_k g(x_k^T W_k) \tag{2.34}$$

$$W_{k+1} = W_k + \mu_k[x_k g(x_k^T W_k) - W_k g(W_k^T x_k) g(x_k^T W_k)] \tag{2.35}$$

These variants were obtained only heuristically, simply by replacing either one, two or three of the products $W_k^T x_k$ or $x_k^T W_k$ by their corresponding nonlinearities $g(W_k^T x_k)$ and $g(x_k^T W_k)$. In [19], a very general network architecture is defined by two functions f_1 and f_2. f_2 is the function applied to the output of each node in the network. If f_2 is linear for all nodes, then it is a linear network architecture. f_1 is a function of the error which has to be minimized. If $f_1(x) = x^2$ then squared error has to be minimized. The network topology (shown in figure figure 2.6) is similar to that used for standard Hebbian learning. For such a network, the estimation of x is a weighted sum of the M basis vectors $w(1), \ldots, w(M)$, where the weights are given by the outputs from each hidden node. More formally,

$$x = \chi + e = \sum_{i=1}^{M} f_2[x^T w(i)] w(i) + e \tag{2.36}$$

where χ is the approximation of x.

To begin with, a nonlinear statistical error criterion $J(W) = E\{f_1(e)|W\}$ is assumed. This gives the conditional probability of the error function with respect to the weights in the network. A gradient descent is done on this criterion resulting in the algorithm

$$w_{k+1}(m) = w_k(m) + \mu_k\{g_1(e_k^T) w_k(m) g_2[x_k^T w_k(m)] x_k + f_2[x_k^T w_k(m)] g_1(e_k)\} \tag{2.37}$$

where g_1 and g_2 are the derivatives of f_1 and f_2 respectively. It is next demonstrated that this general algorithm can reduce into the three extensions presented in [19] by making some simplifications. If a linear network is being used, $f_2(t) = t$ and $g_2(t) = 1$. Also, if the error criterion in this case is quadratic, ie. $f_1(t) = t^2$ and $g_1(t) = 2t$. Substituting these values in the above equation, we get

$$W_{k+1} = W_k + \mu_k[x_k e_k^T W_k + e_k x_k^T W_k]. \tag{2.38}$$

Therefore, for a given weight vector $w(m)$ the update equation is

$$w_{k+1}(m) = w_k(m) + \mu_k[e_k^T w_k(m) x_k + x_k^T w_k(m) e_k] \tag{2.39}$$

However, the values of e_k are much less than x_k. Therefore, the coefficient of $w(m)$ in the first part can be ignored. Therefore, the weight update equation can be rewritten as

$$W_{k+1} = W_k + \mu_k[e_k x_k^T W_k] \tag{2.40}$$

Substituting $e_k = (I - W_k W_k^T)x_k$ in this equation gives

$$W_{k+1} = W_k + \mu_k(I - W_k W_k^T)x_k x_k^T W_k \tag{2.41}$$

which is the standard Hebbian learning equation. Therefore, standard Hebbian learning is a special case of equation (2.37) where a linear network with quadratic error criterion is used.

Another simplification to equation (2.37) can be considered by assuming a nonlinear network (i.e. $f_2(t)$ is nonlinear) and a quadratic error criterion (i.e. $f_1(t) = t^2$). This results in a simplified learning rule

$$W_{k+1} = W_k + \mu_k[x_k e_k^T W_k G_2(x_k^T W_k) + e_k f_2(x_k^T W_k)] \tag{2.42}$$

The first term in the above equation does not contribute significantly to the weight update because the value of e_k is much less than that of x_k. Therefore, the above equation can be simplified as

$$W_{k+1} = W_k + \mu_k e_k f_2(x_k^T W_k) \tag{2.43}$$

Substituting for e_k gives

$$W_{k+1} = W_k + \mu_k[x_k - W_k f_2(W_k^T x_k)] f_2(x_k^T W_k) \tag{2.44}$$

This is the same equation (2.35). Therefore, a nonlinear network with quadratic error criterion reduces to a modified Hebbian learning equation proposed in [19]. Another statistical optimization criterion is that of feature extraction defined as

$$J(W) = \sum_{i=1}^{M} Ef_1[x^T w(i)]|w(i) + \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M} \lambda_{ij}[w(i)^T w(j) - a_{ij}] \tag{2.45}$$

The sum of conditional expectations that depend nonlinearly on the output of the neurons has to be maximized. The constraint $w(i)^T w(j) - a_{ij}$ keeps the weights bounded. This constraint is weighted by the Lagrange multipliers and included in the optimality criteria. If matrix a is assumed to be an identity matrix, then the constraint $w(i)^T w(j) - 1$ where $i \neq j$ imposes orthogonality on the weight vectors. By applying gradient ascent on the above equation results in the algorithm

$$W_{k+1} = W_k + \mu_k[I - W_k W_k^T]x_k g(x_k^T W_k) \tag{2.46}$$

which is one of the extensions proposed to Hebbian learning.

Hence, the generalized algorithms presented in this paper can be reduced into various important extensions of Hebbian learning. The training is done by updating the weights locally depending on the error vector e_k , the input vector x_k and the weight vector $w_k(m)$ of the neuron to be updated. The network topology for all the algorithms presented here is the same, as shown in Figure 2.6.

This algorithm is also completely symmetric because the weight updates are uniform for all the nodes. It is important for the neuron weight vectors to have some order. Asymmetry can be introduced in the training by using only one neuron to begin with and introducing other neurons after some convergence is achieved. Also, the weights connect a particular output node should be constrained to lie within a value to assure convergence. This aspect was incorporated in [24] and further scrutinized in the sections to follow.

2.2.4 Independent Component Analysis

The ideal classification algorithm should perform what is called the blind separation. Blind separation aims at decomposing the signal into statistically independent components (i.e. extracting the original signal sources from their mixture) where there is little

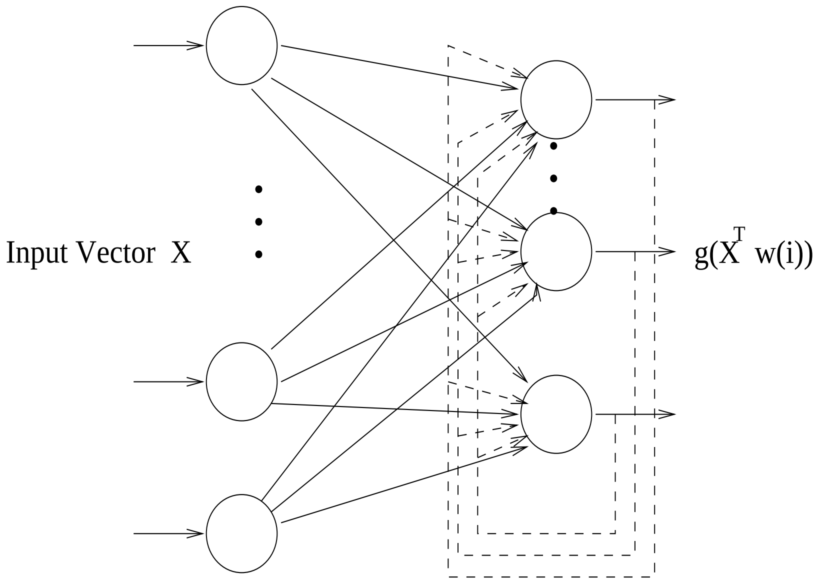

Figure 2.6: The network topology for extended Hebbian learning algorithms.

information about the sources themselves. A complex algorithm called INCA (INdependent Component Analysis) was proposed in [8] which attempts to do blind separation. In this algorithm a mixture model is assumed. If $E_i(i = 1 \ldots n)$ represent the signals from n sensors, and $X_i(i = 1 \ldots m)$ represent the m sources, then we need to determine the elements of a n x m matrix A such that $E(t) = AX(t)$. For this, a neural network is used whose architecture is shown in figure 2.7. The learning rule used to train this network is

$$S_i(t) = E_i(t) - \sum_{k \neq i} c_{ik} S_k(t), 1 \leq i \leq n \tag{2.47}$$

where, $E_i(t)$ is the ith input, $S_i(t)$ is the ith output, c_{ik} are the network coefficients. Therefore, the output $S_i(t)$ of the ith node is the weighted sum of the input signal $E_i(t)$ and other outputs $S_k(t)(k \neq i)$. In matrix form, this equation can be written as

$$S(t) = E(t) - CS(t) \tag{2.48}$$

Assuming that the recursive network is stable, it computes the function

$$S(t) = (I + C)^{-1} E(t) = (I + C)^{-1} AX(t) \tag{2.49}$$

Therefore, if the network is trained so that the coefficients given by matrix C converge to values such that

$$(I + C)^{-1} A = I \tag{2.50}$$

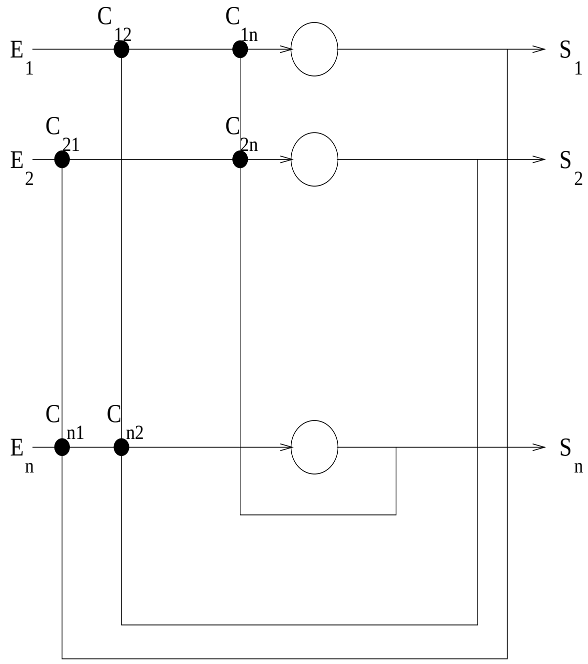

Figure 2.7: The network topology for INCA.

then we have S(t) = X(t), and the network can be used to extract the source signals. To obtain such a convergence (i.e. to determine the matrix C), it is assumed that the network is close to a solution: $n-1$ outputs $S_k(t)$ are already equal to $X_k(t)$, the last output can be given by

$$S_n(t) = E_n(t) - \sum_{k \neq n} c_{nk} S_k(t) \tag{2.51}$$

Since $E(t) = AX(t)$ we have

$$S_n(t) = \sum_{k \neq n} (a_{nk} - c_{nk} a_{kk}) X_k(t) + a_{nn} X_n(t) \tag{2.52}$$

A term $s_n(t)$ is then defined as

$$s_n^2(t) = \sum_{k \neq n} (a_{nk} - c_{nk} a_{kk})^2 X_k^2(t) + a_{nn}^2 X_n^2(t) \tag{2.53}$$

The values of c_{ij} can then be determined by a gradient descent on $s_n^2(t)$ This results in a network trained to transform the input signal $E(t)$ into its independent components $X(t)$.

2.3 Analysis

The above discussion primarily focuses on three important methods for nonlinear feature extraction: unsupervised Hebbian learning extensions, supervised autoassociative bottleneck architectures and independent component analysis. Some aspects of learning which are common to both these categories are: symmetry of the architecture, complementary learning of nonlinear components, convergence of weights.

Symmetrical architectures are easier to implement in hardware because they do not need external hardwiring and can be generalized to any size. Symmetry also allows simultaneous and uniform update of weight vectors. However, symmetrical networks do not perform well especially for linear PCA. There should be some hierarchy among the weight vectors. This is true for both supervised (autoassociative) learning and Unsupervised (Hebbian) learning. This aspect is closely related to that of complementary learning of nonlinear components. In supervised learning, asymmetric learning can be realized using the sequential NLPCA architecture. In Hebbian learning, competitive learning among hidden nodes can be enforced (as described in [24] and [9]) to make the learning asymmetric. In any case, some constraint (as given in equation (2.31)) should be placed on the sum of weights connecting to any node, to ensure convergence.

The following text analyzes the various algorithms discussed above for their network symmetry and complementary learning features.

The autoassociative network has a highly symmetric architecture, because the learning rule is identical for all nodes in the network. As a result, more than a single node in the bottleneck may compete together to learn a single component of the input data effectively reducing the number of bottleneck nodes. Sequential NLPCA avoids the problem by ordering the nodes according to the principal components that they need to learn. Another method would be to introduce only a single bottleneck node and let it train a while until it learns the primary principal component. Other nodes can be slowly introduced in the bottleneck as the training proceeds. This way, the order of learning of principal components may be controlled.

The competitive Hebbian learning presented in [24] uses a nonlinear network where the output of the nodes is calculated using a squashing function. It assumes that if the outputs are constrained to be orthogonal, the set of nodes respond to different components of the input. The change in weight for the modified Hebbian rule consists of a multiplicative factor $[1 - 4\sum_{l \neq i} y_l^2]$ which has a low value if outputs from all nodes other than the node into which w_{ij} connects are high. During training, if the output of other nodes is high, then the weight change in the given node is less than what it would be using standard Hebbian learning. Therefore, if different nodes have different responses, the learning for different nodes will be significantly different. Hence, the network behaves asymmetrically. The weight limit makes sure that the network weights do not become exceedingly high.

This algorithm could give a definite improvement over Hebbian learning because of the nonlinearity involved in the network and the orthogonality constraint imposed during training. Convergence could also be faster if the nodes learn different components of the input. However, the network might become unstable very easily. Equation (2.30) suggests that a node which has the largest output has the largest weight increment for all weights which connect to the input of this node. Similarly, a node which has the lowest output has the lowest weight change for all weights which connect to the input of the node. As a result, the output for such a node does not change significantly. Therefore, the weights connecting to the node which has a large output might increase exceedingly. Hence, applying a weight limit on the overall strength of the input to each node is very crucial. There is no direct method to calculate an optimal weight limit for this purpose. Therefore, the practicality of this algorithm greatly rests on the weight limit chosen. The weight limit has to be chosen empirically. If the input signal is very complex, then the empirical determination of weight limit may be very difficult.

The algorithm proposed in [20] could be used to increase the efficiency in training a network to a set of functions by precalculating the optimal features in the input. This is done by using unsupervised Hebbian learning to determine the eigenvectors for the input data. A set of basis functions which span all the functions that need to be approximated, has to be determined. However, for complex signal in which the function to be approximated is complex and unknown, it is not possible to define a set of basis functions that span all the functions to be approximated. Therefore, this algorithm is useful in approximating small functions efficiently. However, it might not be able to approximate or extract optimal features from complex functions.

In [9], a generalized and broad treatment of network architectures for nonlinear principal component analysis is given. A symmetric network is assumed with constrains given by the two functions f_1 (error criterion function) and f_2 (the output function). The learning algorithms are obtained using two approaches: by doing gradient descent on error criterion; by doing gradient ascent on statistical criterion for feature extraction.

The illustration given in the paper where the generalized algorithms are reduced to standard nonlinear extensions proposed in [19] shows that they are consistent. By varying the functions f_1 and f_2, the effect of nonlinearity and error criterion on the performance of various kinds of networks can be studied. The algorithms that can be constructed from the first statistical optimization criterion result in interesting nonlinear extensions to Hebbian learning while the feature extraction statistical criterion encompasses the concept of competitive learning by introducing the constraint that the weight vectors be orthonormal. The feature extraction algorithm is very similar to the competitive Hebbian algorithm presented in [24]. In [24] the orthogonality constraint is enforced among the nodes by considering a weighted sum of g_{ik} which is the product of two output vectors raised to a positive power.

The nonlinear functions used in the algorithms proposed here introduce higher order statistics into the learning equation producing more accurate uncorrelated outputs than the PCA learning algorithms which utilize only second order statistics.

The INCA algorithm tries to solve the blind separation problem by using an adaptive rule for determining the network coefficients by using high-order statistical moments (with non-linear functions). This involves complex matrix operations for every iteration. Also, the number of learning steps necessary before convergence, is unknown in the adaptive algorithm. The problem of network stability is not addressed in the algorithm.

The above section discussed the studies done in the area of NLPCA using 1. bottleneck architectures and complex sigmoidal nodes to store phase information, 2. nonlinear extensions to unsupervised Hebbian learning, 3. INCA. These algorithms, though more complex than the PCA, are a definite improvement over it as shown in [3], [4], [8] and [14]. The next chapter focuses on the bottleneck architectures and their effectiveness in reducing the dimensionality of well defined signals. Chapter 3 presents the experiments conducted using bottleneck architectures for feature extraction from complex EEG signals.

Chapter 3

PRELIMINARY EXPERIMENTS

This chapter discusses experiments done using the bottleneck architecture and sequential bottleneck networks in reducing the dimensionality of both periodic and non-periodic mathematical functions.

The goal is to optimally compress a data set $\Gamma \subset \Re^N$, which is the locus of a well specified function. The data set for each function has a dimensionality N which is the number of observed variables, and an implicit dimensionality which is equal to the number of independent variables used to generate the data. Therefore, an autoassociative network in which the number of bottleneck nodes equals the number of independent variables in the function should be able to learn to replicate the function.

An autoassociative network as dealt with in [15] is a symmetric network consisting of 5 layers (including input and output layers). Using standard backpropagation, it is difficult to train any but the most simple functions with a large network size. The use of a large training set also makes learning difficult. Therefore, an extension to standard backpropagation called conjugate gradient method, which was first presented in [5], is used to train the autoassociative network.

3.1 Conjugate Gradient Method

In standard back propagation, the steepest descent direction m_p associated with the training example p is given by

$$m_{ijk} = \delta_{i+1,k} \times z_{ij} \tag{3.1}$$

where m_{ijk} is steepest descent direction for the unit j in layer i, to unit k in layer $i+1$, $\delta_{i+1,k}$ is the back-propagated error value and z_{ij} is the output of the unit j in layer i.

The weight change is then calculated using the formula

$$\Delta w = \mu m_p + \alpha \Delta w \tag{3.2}$$

where μ is the learning rate or step size, α is the momentum.

In standard backpropagation, an estimate of the true gradient is determined and applied separately for each training pattern. The true gradient is the direction of steepest descent with respect to all the patterns considered simultaneously. Batching is a technique in which the direction of steepest descent is calculated as the sum of the gradients for individual training examples, ie.

$$M_P = \sum_{p=1}^{P} m_p. \tag{3.3}$$

Another improvement that has been suggested to backpropagation is line search. Given the weights of the network at a specific epoch, the learning rate to be used is determined by considering each learning rate and determining the squared error that would result in the next epoch if that learning rate is used. Given the error function

$$\psi(w) = \frac{1}{2N} \sum_{n=1}^{N} \sum_{j \in C} (d_j(n) - y_i(n))^2 \tag{3.4}$$

where N is the number of train patterns, C is the set of neurons in the output layer, $d_j(n)$ is the actual output of jth neuron using nth train pattern, $y_j(n)$ is the desired output of jth neuron using nth train pattern. The learning rate can be determined as

$$\mu(n) = \min_{\mu} \{\psi(w(n) + \mu p(n))\} \tag{3.5}$$

where $p(n)$ denotes the gradient at nth epoch. The learning rate which results in the least error is used for the current epoch. Therefore, an adaptive learning rate is used in training.

Batching and line search provide consistent improvement over backpropagation. However, steepest descent is a poor optimization method. In [13], the authors propose to combine batching of patterns and line search with gradient descent in conjugate directions.

The conjugate gradient method can be viewed as batch BP with dynamic adjustment of the learning rate and momentum.

If $p(n)$ denotes the gradient at nth epoch, the weight vectors are updated according to the equation

$$w(n+1) = w(n) + \mu(n)p(n), \tag{3.6}$$

where $\mu(n)$ is the learning rate for the nth epoch determined using line search. The direction vector $p(n)$ is computed using the previous direction vector $p(n-1)$ and the direction of steepest descent M_P determined from equation 3.3 as follows:

$$p(n) = \begin{cases} -M_P(n) + \beta(n-1)p(n-1), & \text{if } n > 0; \\ -M_P(n), & \text{if } n = 0, \end{cases} \tag{3.7}$$

where $\beta(n)$ is the adaptive momentum for the nth epoch. This is calculated by using the Fletcher-Reeves formula given in [5] as:

$$\beta(n-1) = \frac{M_P^T(n)M_P(n)}{M_P^T(n-1)M_P(n-1)}. \tag{3.8}$$

Therefore, the conjugate gradient method uses batch BP with adaptive learning rate and momentum. In [13], the authors compare the conjugate gradient method with conventional BP using an XOR network as example. The conjugate gradient method is shown to result in faster convergence.

3.2 Performance tests

In order to compare the effectiveness of sigmoidal versus circular bottleneck nodes, a set of functions was chosen whose implicit dimensionality is less than the dimensionality of the data. Some functions are periodic, some are aperiodic and some functions consist of a mixture of periodic and aperiodic variables. The following subsections describe the performance of the different network architectures on examples of each kind of function.

3.2.1 Circle

This is a periodic function of two variables a and b which are generated using a single parameter θ as

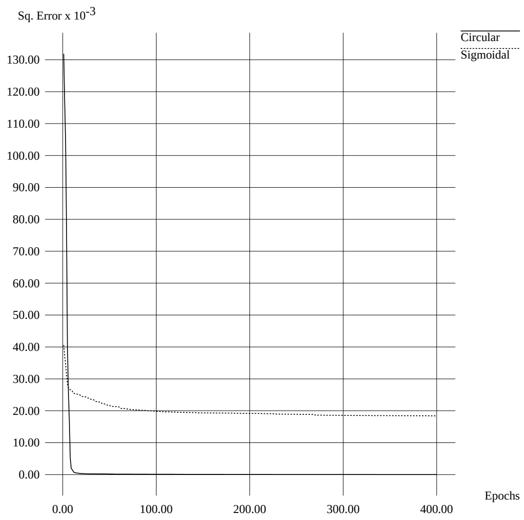

Figure 3.1: Error in replicating data on a circle using one circular/sigmoidal bottleneck node.

$$a = \sin(\theta); \qquad b = \cos(\theta) \qquad (\theta = [0, 2\pi])$$

Therefore, the implicit dimensionality of this function is one. A network consisting of one bottleneck node should be able to replicate this function, ie., given input values of a and b its output should be approximately the same values. Therefore, the network should learn the underlying parameter θ.

Figure 3.1 shows typical learning curves (in terms of replication error across epochs) using a $2-6-1-6-2$ network with one sigmoidal bottleneck node and a $2-6-1-6-2$ network with one circular bottleneck node. Both networks had 6 sigmoid nodes in the mapping and demapping layer and were trained for 400 epochs.

As shown in Figure 3.1, the test error using sigmoidal bottleneck settles at a higher value than that using the circular node. This reinforces the fact that it is easier to map periodic data onto a circular manifold. A mapping of a periodic signal onto an open interval is irreversible. Therefore, a sigmoidal bottleneck network does not consider the periodicity in the network. This is illustrated in Figures 3.2 and 3.3 which show the expected output versus actual output of a network which is made to learn θ, given $\sin(\theta)$ and $\cos(\theta)$ as inputs. When using sigmoidal nodes, the network has difficulty in mapping the points on the circle which are close to 0 or 2π, as shown in Figure 3.3 because of

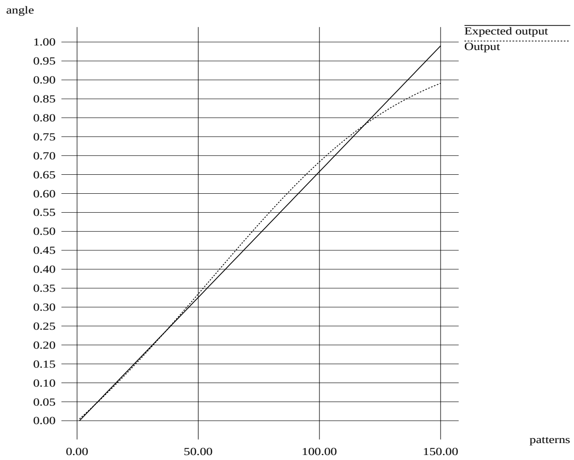

Figure 3.2: Using circular nodes to learn θ corresponding to a point on the circle.

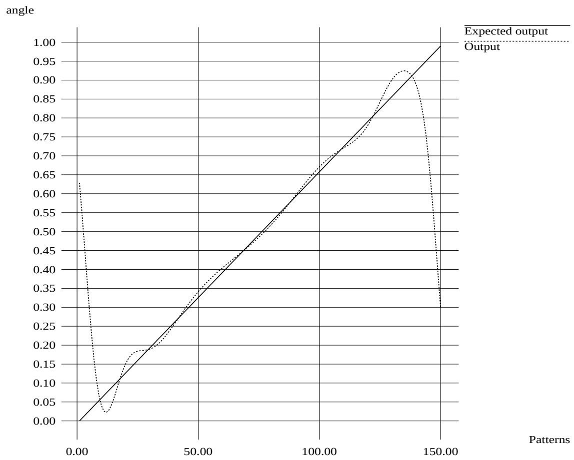

Figure 3.3: Using sigmoidal nodes to learn θ corresponding to a point on the circle.

the discontinuity in data. However, a good mapping is obtained when the network uses circular nodes.

3.2.2 Trefoil knot

This is a periodic function of three variables x, y and z which are determined using a single parameter θ as

$$x = \cos(2\theta)(2 + \cos(3\theta)) \tag{3.9}$$

$$y = \sin(2\theta)(2 + \cos(3\theta)) \tag{3.10}$$

$$z = \sin(3\theta) \tag{3.11}$$

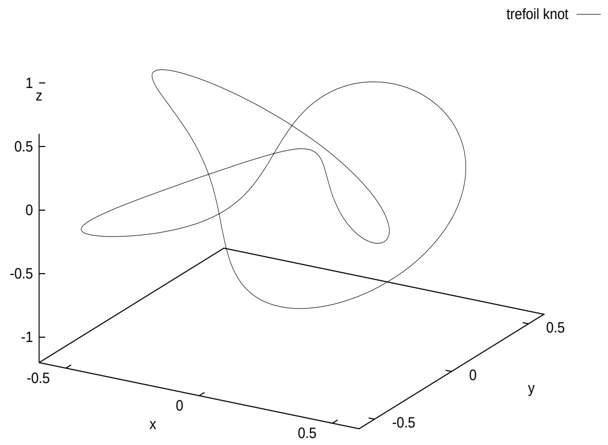

Figure 3.4: The 3-dimensional trefoil knot.

The implicit dimensionality of this function shown in Figure 3.4 is one.

An autoassociative network with one bottleneck node should be able to replicate the trefoil knot. As in the previous function, a single circular bottleneck node should be able to replicate a trefoil knot better than the sigmoidal node because the curve is intrinsically homeomorphic to a circle, i.e, a reversible mapping exists from the knot to a unit circle. The network has to determine this mapping in order to replicate the knot.

Figure 3.5 shows the test inputs and outputs to a $3-15-1-15-3$ network with one circular node in the bottleneck layer. Figure 3.6 shows the test inputs versus outputs for the three dimensions of the trefoil knot.

Figure 3.7 shows the test input versus output for each of the three dimensions of the trefoil knot using a trained $3-15-1-15-3$ network consisting of one sigmoidal node in the bottleneck layer. It can be observed that the single sigmoidal node in the bottleneck layer is forcing the network to learn only one of the dimensions in the data. This could be also seen from figure 3.8 where the error in replication settles to a high value and remains constant.

3.2.3 Square and reciprocal functions

An aperiodic function of two measured variables x and y which are determined using a single parameter a as $x = a^2$ and $y = 1/a^2$, has an implicit dimensionality of one. An

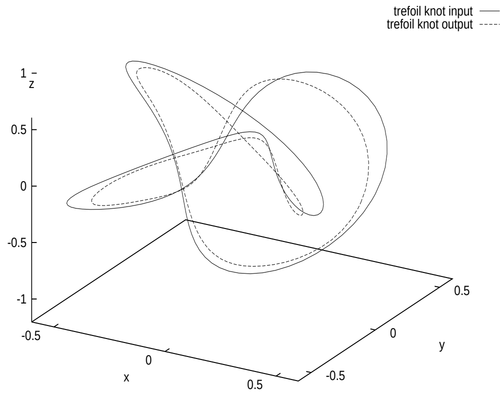

Figure 3.5: Input and output of a network having one circular bottleneck node trained on the trefoil knot.

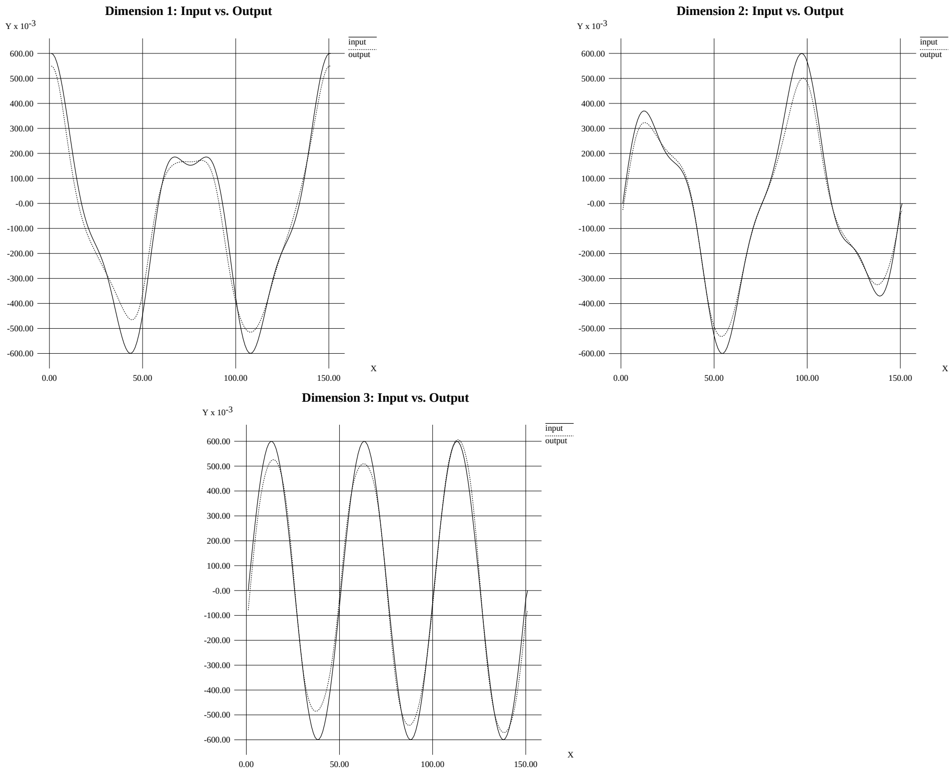

Figure 3.6: Using one circular bottleneck node to replicate a trefoil knot.

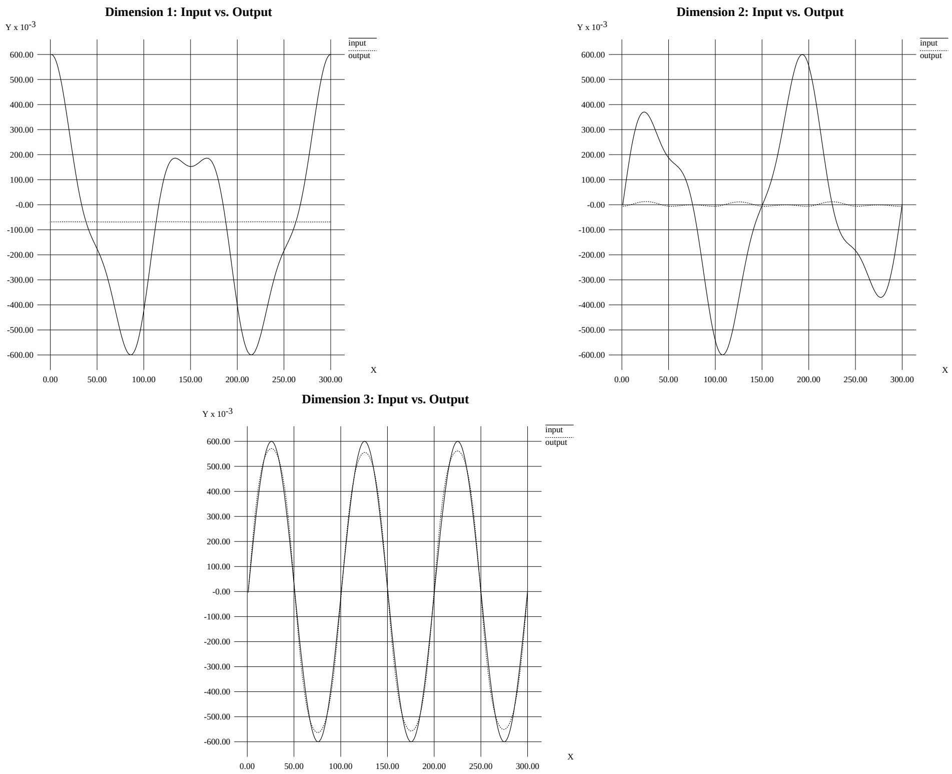

Figure 3.7: Using one sigmoidal bottleneck node to replicate a trefoil knot.

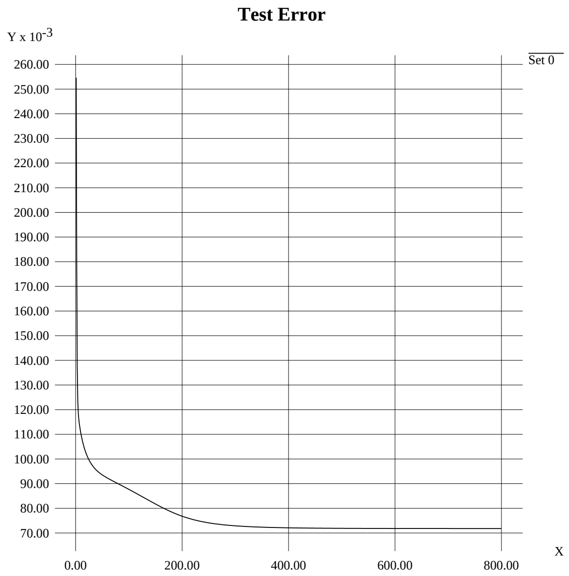

Figure 3.8: Test error across epochs in training a $3-15-1-15-3$ network with one sigmoidal node in the bottleneck layer.

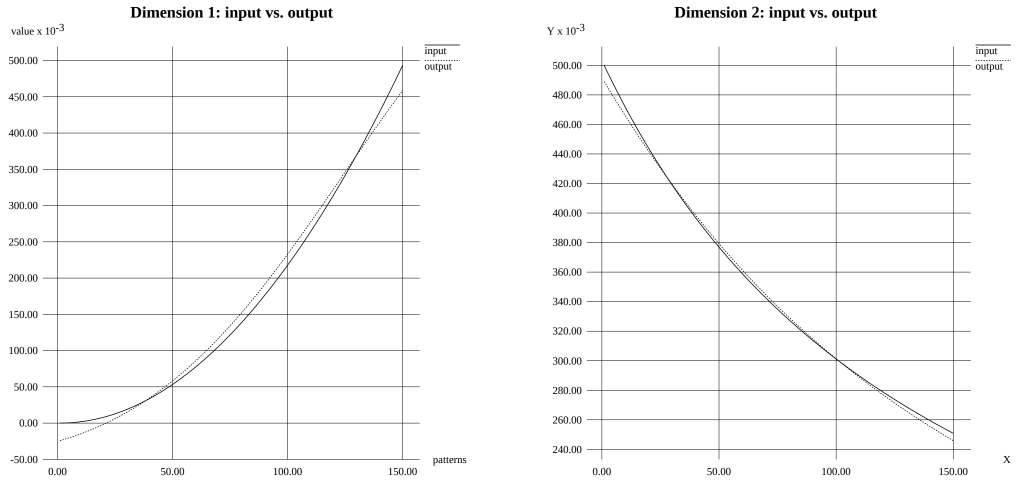

Figure 3.9: Using one sigmoidal bottleneck node to replicate a square and reciprocal function.

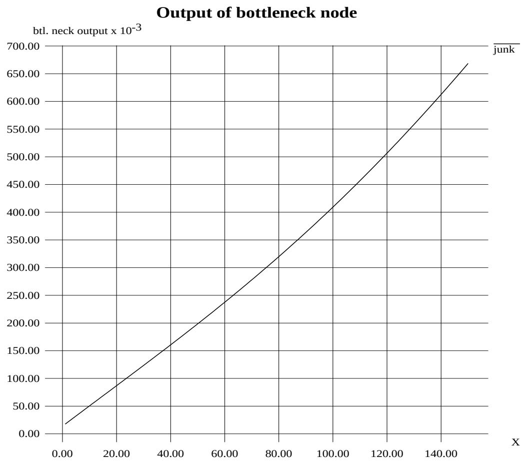

Figure 3.10: Output of the sigmoidal bottleneck node when learning the square and reciprocal function.

autoassociative network with a single bottleneck node is used to replicate the function. Figure 3.9 shows the input and output to the network in both the dimensions, using sigmoidal nodes. Eight sigmoidal nodes were used in the mapping and demapping layers. Sigmoidal nodes perform well because the function is aperiodic. Therefore a mapping onto an open interval is easily obtained. Figure 3.10 shows the output of the bottleneck node. The output is a linear function of x, the implicit variable involved in the function.

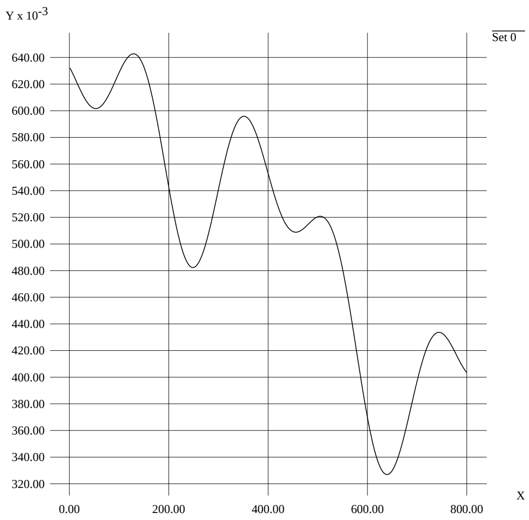

Figure 3.11: A weighted sum of sine waves used as the noise free source signal.

3.2.4 Replicated signals

In [15], it is stated that the autoassociative network should be capable of producing a model of measurements that fits the systematic correlation in the data, yet excludes random variations due to noise. This can be demonstrated by using an n-dimensional signal where each dimension corresponds to a sensor reading of the same source signal. Therefore, the value of the signal in each dimension is identical expect for a noise term which is uncorrelated. The network should be able to reduce the dimensionality of such a system to one, and filter the noise in the signals. For this purpose, a signal as shown in Figure 3.11, which is a weighted sum of a few sine waves with different phase and frequencies, is assumed to be the source signal. N identical copies of the signal with different white noise added is used as input to a autoassociative network with one bottleneck node since the implicit dimensionality for the set of signals is one. Figure 3.12 shows one such input signal. Figure 3.13 shows the output of the single bottleneck node given 6 signals identical to the one shown in Figure 3.11 with different white noises added. The output of the bottleneck node filters out the uncorrelated noise in the signal. However, as Figure 3.13 shows, any correlations in the signals (including noise correlations) are retained in the signal.

This chapter discussed the results of reducing the dimensionality of well defined mathematical functions, both periodic and aperiodic. Functions that are homeomorphic to a

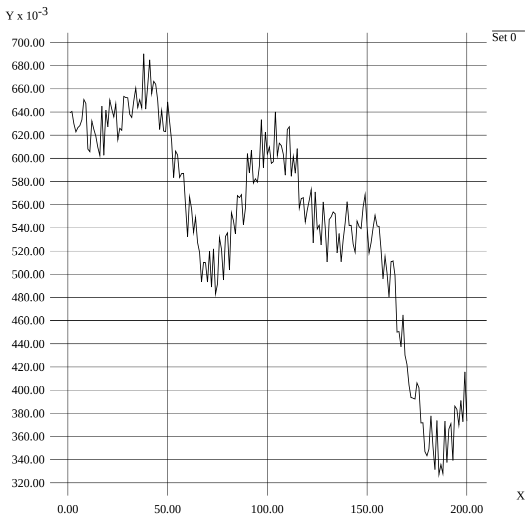

Figure 3.12: Some white noise added to the same signal.

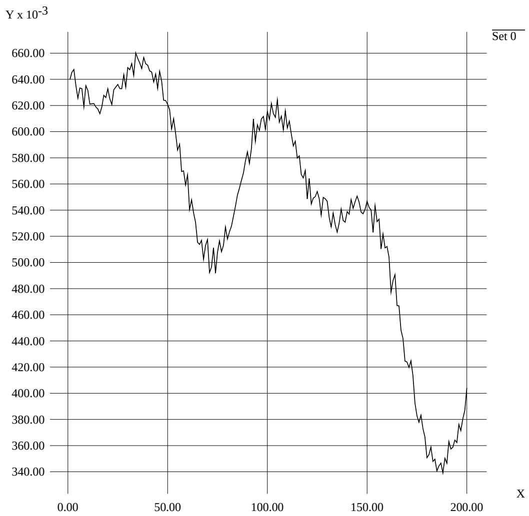

Figure 3.13: Output of the sigmoidal bottleneck node.

circle could be directly mapped onto a unit circle with a single circular node in the bottleneck layer. Chapter 3 gives a description of the EEG signals, their representation in the form of input vectors and the methods used in reducing the dimensionality of the input vectors.

Chapter 4

METHODOLOGY OF EXPERIMENTS

This chapter describes the collection of EEG data from the subject which consists of recording the multiple channels of the signal, removing eye-blinks from the data and restructuring the data into temporal windows. The NLPCA and classification network topologies and training strategies are given. Finally, the distribution of EEG data belonging to Mental Arithmetic and Mental Letter Writing into testing and training sets for both the NLPCA and the classification networks is presented.

4.1 EEG Data Collection

The EEG data used in all the experiments was recorded by Keirn and Aunon [10]. In their study, a set of tasks comprising of Baseline (representing no mental activity), Mental Arithmetic, Geometric Figure Rotation, Mental Letter Composing and Visual Counting were chosen for analysis. The EEG data was recorded from subjects seated in a sound-proof dimly lit room with 6 electrodes placed at standard electrode positions $O_1, O_2, P_3, P_4, C_3, C_4$. A Grass amplifier was used as a bandpass filter which extracted only the 0.1-100 Hz bandwidth in the signals. The current work specifically focuses on classification between mental arithmetic and letter writing tasks based on the assumption that they involve significantly different mental processes.

- **Mental Arithmetic** A non-trivial multiplication problem is presented to the subject who is asked to solve it mentally without vocalizing or overt movements. The numbers involved in the multiplication are different for each trial and are designed so that immediate answers are not apparent.

- **Letter Composing** The subject is asked to compose a letter to a friend or relative without vocalizing. For the successive trials, the subject is asked to pick up from where he left before.

Ten sessions were conducted for each task each session lasting for a period of ten seconds which resulted in 2500 samples per task per session. Therefore, one set of raw EEG data consisted of six dimensions (corresponding to the six channels), with 2500 samples for each dimension. A separate seventh channel (called the eye-blink channel) was used to record eye-blink information. A high potential spike in the eye blink channel (greater than 100 μVolts) lasting up to 10 milliseconds was considered as an eye-blink. Eye blinks do not carry any significant features of data and can become the basis of classification. Therefore, sample points corresponding to all six channels which fall in the region of eye blinks were removed from the raw EEG data before being subjected to any processing.

Figure 4.1 shows the raw data values for letter and math task data with the samples falling in eye-blink region removed.

EEG data recorded using multiple channels could have correlations among the different channels. These correlations might be temporally separated. If individual samples of six channel EEG data are considered as separate input vectors for classification, the temporal correlations among the channels will be completely disregarded. This was verified to be true from attempts in the current study to classify individual samples of EEG data resulted in classification close to chance. For this purpose, temporal windows of EEG data are employed in all experiments. Each window is chosen to be 62 samples long (approximately corresponding to a quarter second) and consecutive windows are overlapped by 31 samples [2]. Since each window consists of 62 samples per channel and there are 6 channels, the dimensionality of the data set is $62 \times 6 = 372$. Therefore we have input vectors of dimensionality 372 belonging to 10 trials for math and letter tasks.

4.2 Dimensionality Reduction and Classification

In [3], it was suggested that dimensionality reduction as a preprocessing step for classifying signals using neural networks could lead to better classification results because

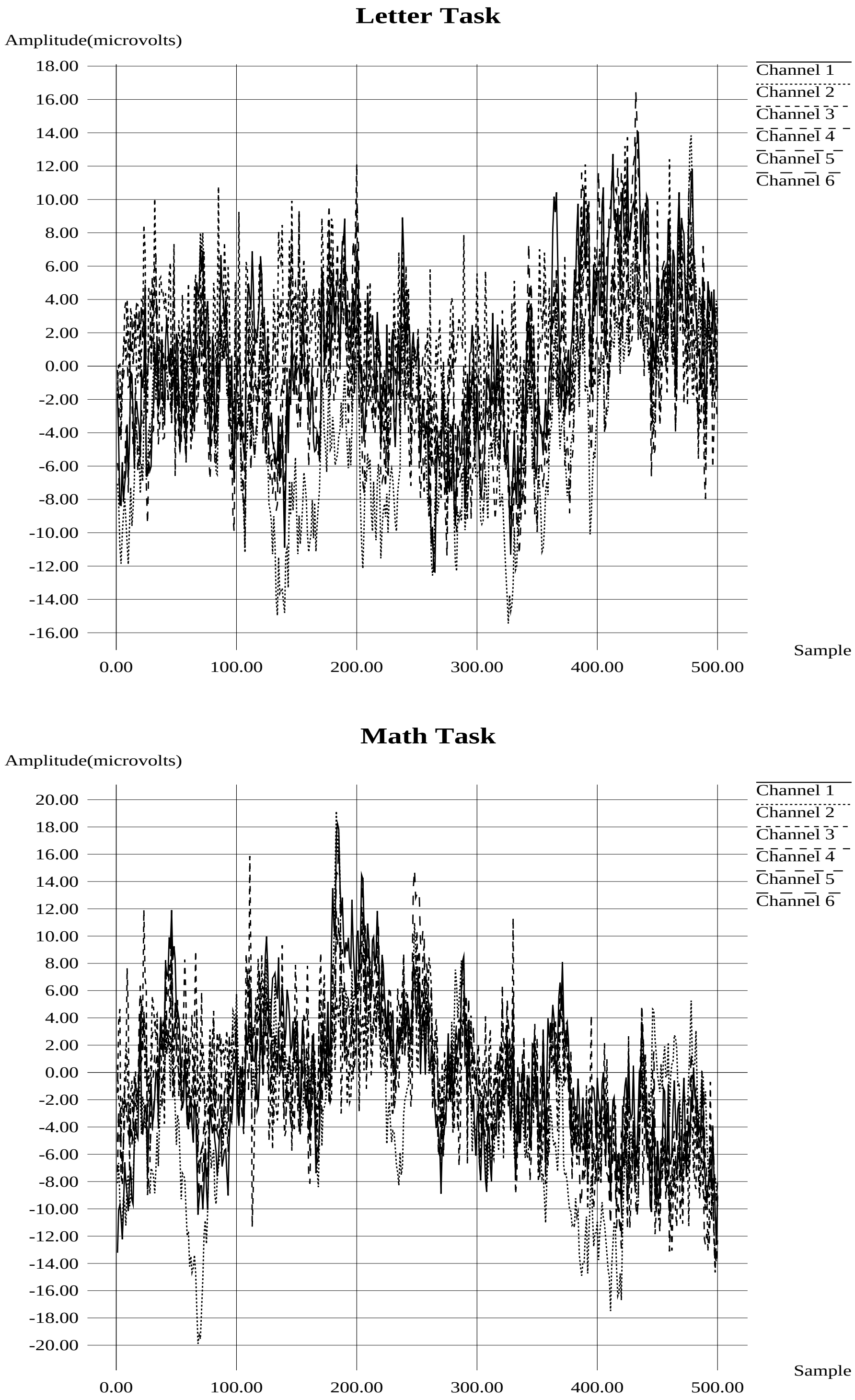

Figure 4.1: Raw six channel eye-blink free data belonging to trial 1 for subject 3 (first 500 samples).

it may extract significant features in the data that will be helpful in its classification. In Chapter 2, autoassociative networks were discussed as an effective technique for non-linear dimensionality reduction. The technique followed here is to use an autoassociative network to extract significant features from the data vectors (corresponding to the signal) by providing a non-linear mapping of the input vectors onto a lower dimension, as a preprocessing step to the classification of the signal. Since this work addresses classification of the EEG signals into one of math and letter composition tasks, there are two steps associated with this:

- using a bottleneck network to determine a reduced dimensionality representation for the input vectors,
- using a classification network to classify the reduced dimensionality representation into one of math and letter composition tasks.

Therefore, a bottleneck network has to be initially trained in order to provide a function to reduce the dimensionality of the input vectors to a specific value. This function has to be used to generate a lower dimensional representation of all the input vectors which will be employed in training and testing the classification network to classify the input vectors into math and letter tasks. The available data should be used in training and testing both the bottleneck and the classification networks. Therefore, input vectors belonging to the ten trials should be separated into a training set and a test set so that the training set is used in training both networks and the test set is used in testing both networks. The number of inputs to the classification network should equal the number of nodes in the bottleneck layer of the bottleneck network (which will be the reduced dimensionality of the signal).

The K-L transform discussed in Chapter 2 is the linear counterpart of the NLPCA method. In order to compare the effectiveness of K-L transform in extracting significant task dependent features from the signal with that of NLPCA method, the reduced dimensionality K-L representation of the EEG data is subject to classification. Therefore, there are two steps associated with this:

- projecting the input vectors to first n principal eigenvectors to obtain a reduced n dimensional K-L representation for the input vectors.
- using a classification network to classify the K-L representation into one of math and letter composition tasks.

The number of inputs to the classification network should equal the dimensionality of the K-L representation of the EEG data.

4.3 Implementation

The bottleneck network denoted by $372-k-n-k-372$ consisting of 372 nodes in the input and output layers, a prespecified number of nodes k in the mapping and demapping layers and n nodes in the bottleneck layer (where n is the dimensionality to which the data vectors will be reduced to) is trained to replicate the 372 dimensional input vectors belonging to the training set. The table below shows the values of n and k used in the experiments.

n	k
10	20
20	30
30	40
40	60

The weights of the network are stored at each epoch which results in the least error in replication of the test set until that epoch. The trained weights (and biases) of the mapping and the bottleneck layers form the mapping function denoted by $372 - k - n$, which takes a 372 dimensional data vector as input and produces an n dimensional vector as output as shown in figure 4.2. This mapping function will be used in converting all the input vectors into the corresponding reduced dimensionality representations.

The training and test sets for the classifier network have an input vector dimensionality of n and are obtained by applying the mapping function to the corresponding training and test sets for the bottleneck networks, respectively. Using the same training sets for both networks will prevent any two training vectors to the bottleneck network from having reduced dimensional representations one of which belongs to the test set and the other to

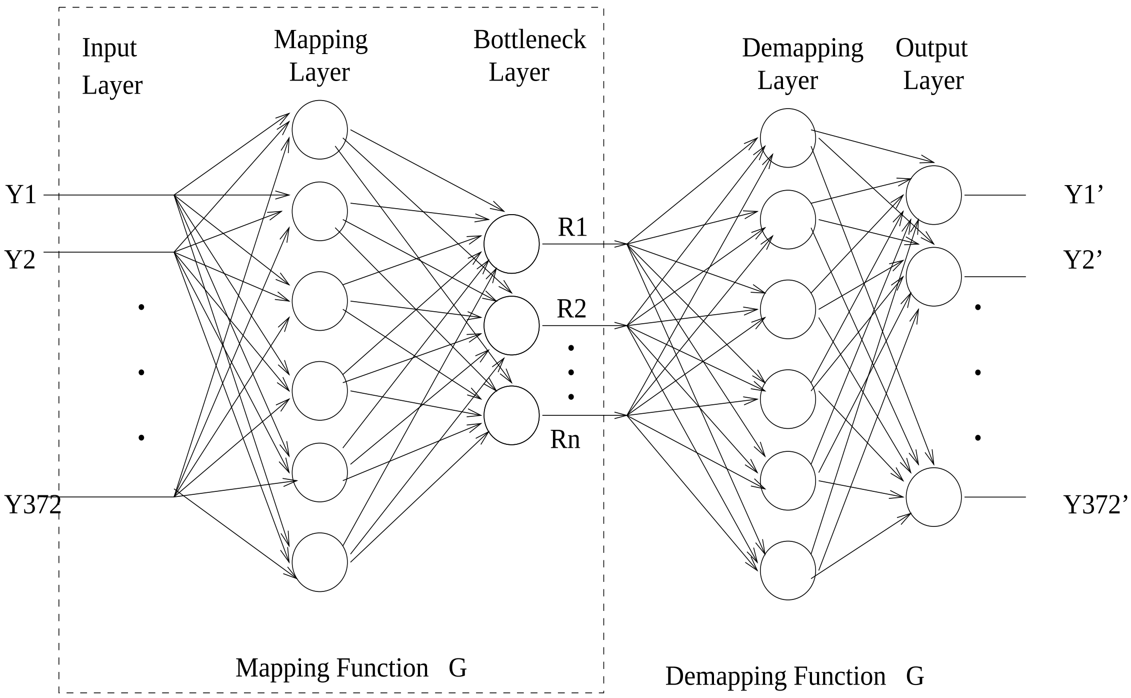

Figure 4.2: Mapping function in the trained bottleneck network.

the training set of the classifier network and ensures generalization in learning. The classifier network has only one output whose target is -0.9 for an input vector corresponding to the letter task and 0.9 for an input vector corresponding to the math task. Therefore, the classification network can be denoted by $n - p - 1$ where p is the number of hidden units in the classification network. The values of p used in the experiments are 30, 60 and 80. Both the bottleneck and the classification networks are trained using standard back-propagation using conjugate gradient method for faster convergence.

Since there are 10 trials of data corresponding to both math and letter tasks for a specific subject, the testing and training sets for classification experiments can be obtained in 10 different ways (referred to as RUNS) by choosing data vectors from one trial as the testing set and data vectors from the rest of the trials as training set. Figure 4.4 shows how data vectors belonging to ten trials are distributed into training and testing sets and used in training and testing the bottleneck and classification networks for the i^{th} run.

$DSET_i$ is the set of input vectors (of dimensionality 372) corresponding to math and baseline tasks for the i^{th} trial. The ordering of input vectors within $DSET_i$ is immaterial but a fixed convention is followed for simplicity so that input vectors belonging to the

DSET(i) : Data vectors belonging to math and letter tasks corresponding to ith trial.
Data vectors for Letter task precede those of Math task.

TEST(i) : ith test set consisting of data vectors belonging to DSET(i)

TRAIN(i) : ith train set consisting of data vectors belonging to DSET(k) (k = 1..10, k != i)

CTRAIN(ni) : Output of the nodes in the bottleneck layer in a trained bottleneck network consisting of "n" bottleneck nodes when TRAIN(i) is given as input to the network

CTEST(ni) : Output of the nodes in the bottleneck layer in a trained bottleneck network consisting of "n" bottleneck nodes when TEST(i) is given as input to the network

Figure 4.3: Distribution of data vectors into training and test sets.

letter task precede those of math task. $TEST_i$ is the set of input vectors belonging to trial i and is used as the test set for the i^{th} run. $TRAIN_i$ is the set of input vectors from all trials but the i^{th} one and forms the training set for the i^{th} run.

The training set $TRAIN_i$ is used to train a bottleneck network $372 - k - n - k - 372$ to replicate the input. Once the bottleneck network is trained, the mapping network is detached from the trained autoassociative network and used separately as a mapping function for generating n dimensional representations from input vectors. $CTRAIN_{ni}$ is the result obtained by applying the mapping function to $TRAIN_i$ and forms the training set for the classification network and $CTEST_{ni}$ is the result obtained by applying the mapping function to $TEST_i$ and forms the test set for the classification network. The classification network has n nodes in the input layer and one node in the output layer.

Since the conjugate gradient method (described in Chapter 3) is used in training both networks, the learning rate and momentum are not specified for training. The maximum number of epochs to train the network was decided based on the number of epochs needed for the networks to converge in a few experimental runs.

The K-L representation of data was also used in classification. Here, the K-L transform is applied to the 372 sample windows of input data belonging to all trials of both the tasks to reduce its dimensionality to n (where $n = 10$, 20, 30 and 40). This is done by projecting the data onto the first n principal eigenvectors. The eigenvalues and eigenvectors of the covariant matrix of 372 dimensional patterns are determined using the Jacobi

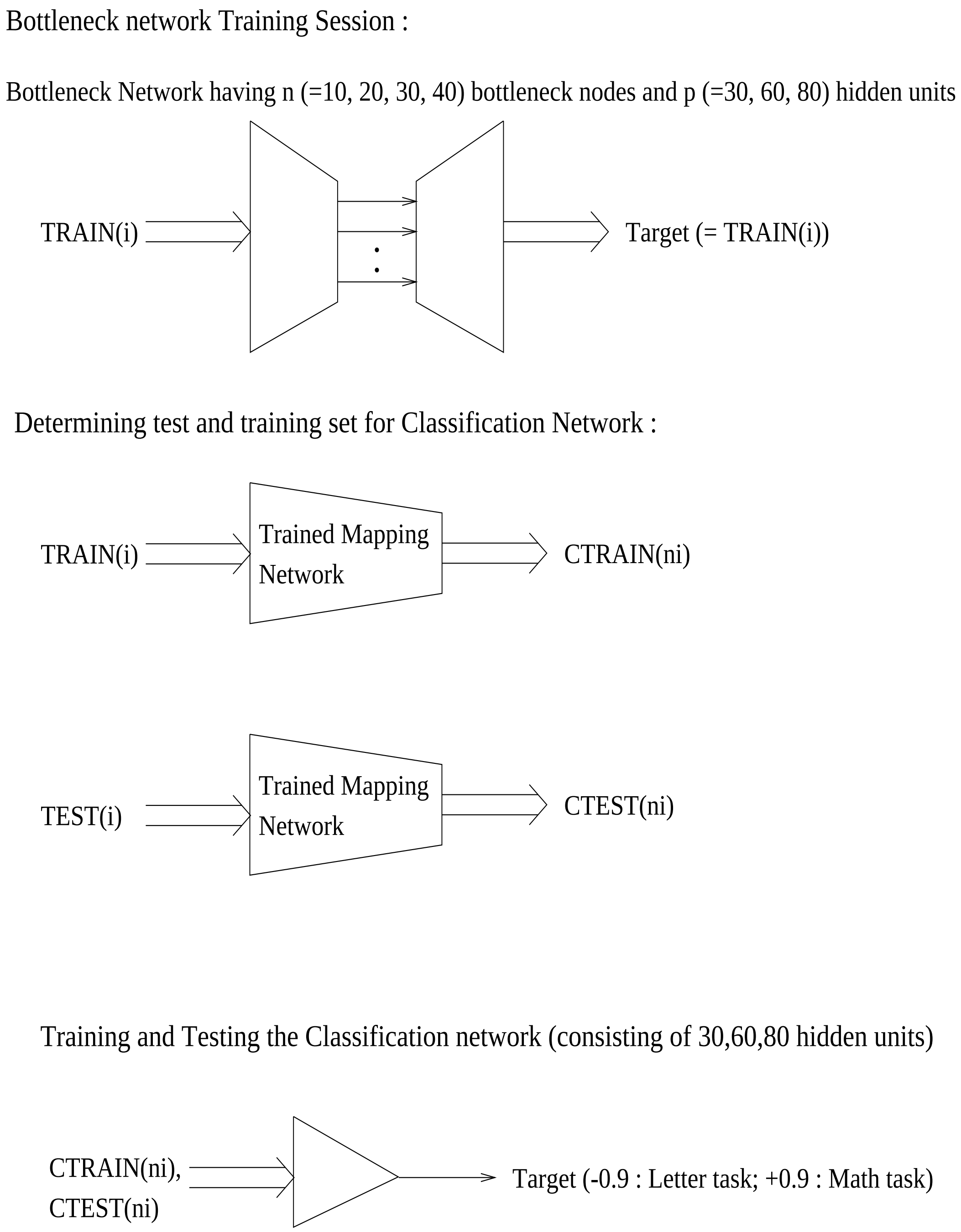

Figure 4.4: Training and testing methodology for Bottleneck and Classification networks.

method in which a sequence of similarity transformations are applied to the input data to reduce the off-diagonal elements to zero. The product of all the transformation matrices is the matrix of eigenvectors (one in each column). The K-L representation of data is then given as input to train an $n - 60 - 1$ classifier network having 60 sigmoidal bottleneck nodes using standard back-propagation. The target for the network is -0.9 for letter task data vectors and 0.9 for math task data vectors. The test and training sets for the classifier network are chosen in the same way as for the classification network in the NLPCA method. The K-L representation of data belonging to trial one was used as test set and the K-L representation of data belonging to trials two thru ten was used as the training set. The NLPCA and K-L transform method are compared for $n = 10$, 20, 30, 40 in terms of percentage correctly classified test vectors.

In this chapter, the collection of EEG data from subjects and the conversion to sampled windowed data vectors was discussed. This was followed by a description of how the data was separated into training and test sets and how the bottleneck and the classification networks were trained and tested. The next chapter discusses the results of training and testing both these network and an analysis of the distribution of input vectors and their classification accuracies.

Chapter 5

RESULTS AND ANALYSIS

This chapter presents the dimensionality reduction and classification results obtained by training the NLPCA and classification networks, respectively. This is followed by an analysis of the networks weights and the reduced dimensionality representations of EEG data vectors using different methods such as clustering the vectors, projecting the vectors along the first two eigenvectors etc. Finally, the classification results obtained using a reduced dimensionality K-L representation of the EEG data are compared with those of its non-linear counterpart, the NLPCA method.

5.1 Dimensionality Reduction

The first phase of the experiments consists of training a $372-k-n-k-372$ bottleneck network to replicate input vectors belonging to math and letter tasks. An NLPCA network consisting of random weights was trained to replicate the test set. Figure 5.1 shows the mean squared error in replicating the test set at each epoch in training. The error at any epoch is the average over the training sessions corresponding to all the trials. The NLPCA network was trained with conjugate gradient method and it was observed that the replication error settles at a constant value for all numbers of bottlenecks nodes at about 150 epochs. Therefore, a maximum of 200 epochs was used in all training sessions for the bottleneck network.

The outputs of the bottleneck nodes of the trained bottleneck network correspond to the n dimensional representation of the signal. It was determined that the reduced dimensional representations for the input vectors were uniformly distributed in the n dimensional space. This was done by determining the average mutual Euclidean distance within the set of vectors in the reduced dimensional representation and comparing it

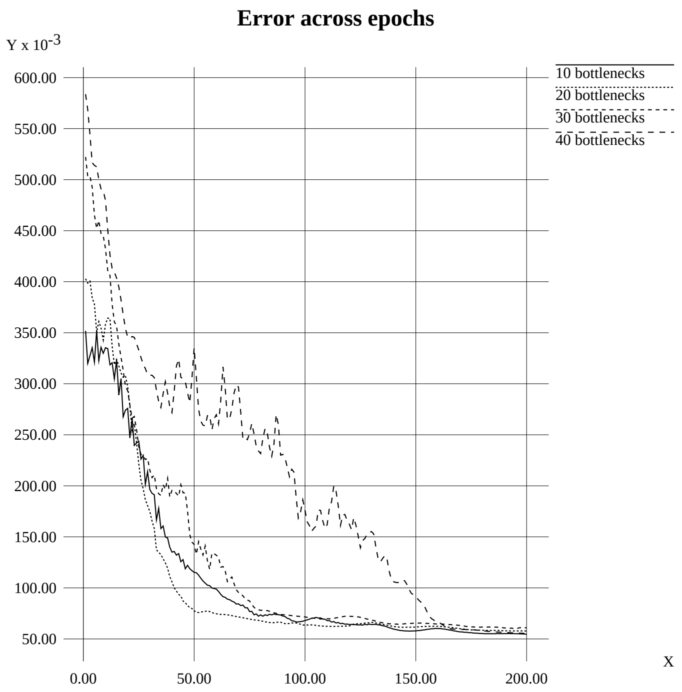

Figure 5.1: Mean Squared error across epochs in training the $372 - k - n - k - 372$ bottleneck network (averaged across all trials).

with that of a set of same number of vectors which have a uniform distribution in the n dimensional space. Table 5.1 shows the values of average mutual distances for n = 10, 20, 30, 40 corresponding to the reduced dimensional representation and a set of n dimensional vectors forming a uniform distribution.

# bottlenecks	*math*	*letter*	*random*
10	2.7151	2.7679	2.5281
20	3.4985	3.5176	3.6126
30	4.9539	4.9159	4.8649
40	6.6917	6.7382	6.8021

Table 5.1: Average mutual distance between a specific number of n dimensional vectors belonging to letter task, math task and a uniform distribution, respectively.

5.2 Classification

This consisted of classifying the reduced n-dimensional representation of original input data into math and letter tasks using a classification network as described in Chapter 4. The targets used for math and letter data vectors are 0.9 and -0.9 respectively. Figures 5.2, 5.3, 5.4 and 5.5 show the classification errors (averaged over test data set) across training epochs using all ten combinations of trial data (as described in Chapter 5) for 10,20,30 and 40 dimensional representations of the original data. From each figure, it is evident that classification error across epochs follows the same trend for different combinations of trial data for the 10 sessions belonging to both the tasks. For 10 dimensional representation, the classification network does not seem to be doing any learning which is evident from the flat structure of the learning curve. This is shown in Table 5.2 where the percentage of correctly classified vectors for $n = 10$ is only slightly better than chance. For the other representations, the classification error in general seems to reduce constantly until approximately 700 epochs and then increase. Among $n = 10$, 20, 30 and 40, the worst classification error is for n=10 and the best classification error is for n=30. This implies that the first ten significant features (that were extracted by training the bottleneck network) are not being sufficiently useful in classifying the vectors as belonging to one of math and letter tasks. However, the 30 dimensional representation of data vectors (belonging to the letter and math tasks) is able to store enough significant features that will enable the data to be distinguished as belonging to one of the two tasks. The 40 dimensional representation is classified better than the 10 dimensional representation but worse than the 30 dimensional representation. This means that the additional features contained in the 40 dimensional representation are making it difficult to differentiate the data vectors as belonging to one of the two tasks. The NLPCA method extracts important features in the data without any knowledge of the type of task that the data belongs to. Therefore, it is possible that the additional features contained in the 40 dimensional representation are common to both the tasks and are making it more difficult to be classified.

Table 5.2 gives the number of vectors and percent vectors correctly classified corresponding to 10, 20, 30 and 40 dimensional representations respectively for the trained classification network. A data vector is assumed to be correctly classified if the single output of the classification network is > 0 for a target value of 0.9 and < 0 for a target value of -0.9.

Figures 5.6, 5.7, 5.8 and 5.9 show the number of incorrectly classified vectors across training epochs for the 10, 20, 30 and 40 dimensional representations of data vectors. The training and test sets for trials 1 through 10 have different sizes because they are obtained

Trial	*# Data Vectors*	*Correctly Classified*				*Percent Correct*			
		n				n			
		10	20	30	40	10	20	30	40
1	136	81	119	127	97	59.5	87.5	93.3	71.3
2	118	56	87	98	76	47.4	73.7	83.0	64.4
3	154	108	133	143	114	70.1	86.3	92.8	74.0
4	136	95	98	111	82	69.8	72.0	81.6	60.3
5	106	46	75	95	52	43.4	70.7	89.6	49.0
6	130	67	93	109	90	51.5	71.5	83.8	69.2
7	154	86	121	130	116	55.8	78.5	84.4	75.3
8	142	84	111	126	81	59.1	78.1	88.7	57.0
9	134	81	91	107	83	60.8	67.9	79.8	61.9
10	148	83	107	125	103	56.0	72.3	84.4	69.6
Total	1358	623	1035	1171	894	57.9	76.2	86.2	65.8

Table 5.2: Number and Percent correctly classified vectors of 10, 20, 30 and 40 dimensions.

from 10 different sessions (as described in Chapter 5) which have unequal number of samples based on the number of samples which do not fall under an eye blink. It can be noticed that the number of test vectors incorrectly classified directly corresponds with the average error in classification of the test vectors.

Figures 5.4, 5.11 and 5.12 show the distribution of output values for the classification network for 20, 30 and 40 dimensional data corresponding to target values of -0.9 and 0.9 for letter and math data respectively. The symbol "$\diamond$" corresponds to the output of the trained classification network which was given an input vector belonging to letter data and "+" corresponds to the output of the trained classification network which was given an input vector belonging to math data. The outputs correspond to input vectors belonging to all the trials. Since trials one through ten (for both the tasks) are tested on separately trained networks (chapter 5 to be referred for detailed explanation of how data from different trails is used in training and testing separate networks) the outputs shown in the figures 5.4, 5.11 and 5.12 belong to different classification networks based on the trail to which the individual data vectors belong. The segregation of output values around the targets is more pronounced in 30 dimensional representation versus 20 and 40 dimensional representation which is reflected in the values of percentage of total vectors correctly classified given in table 5.2.

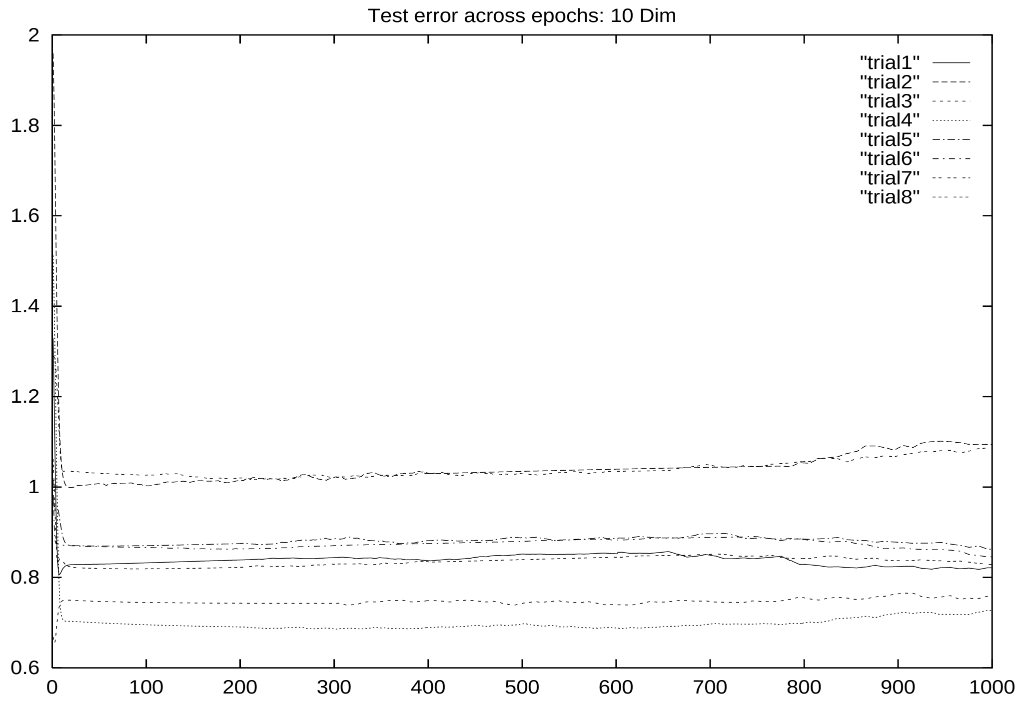

Figure 5.2: Average test error across training session for the classification network for 10 dimensional representation.

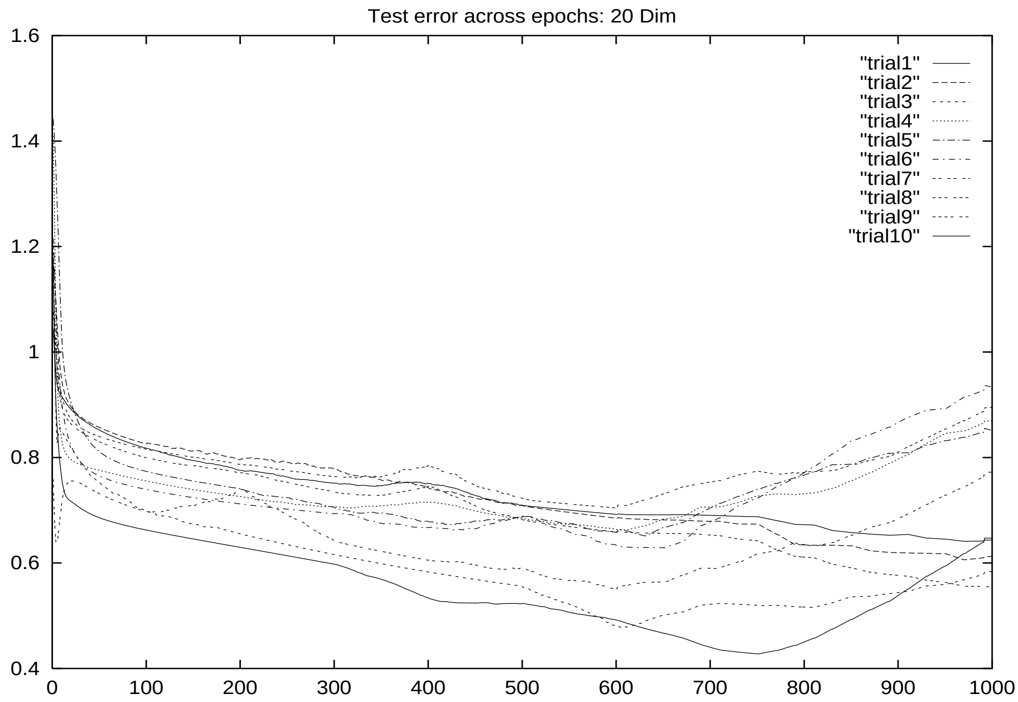

Figure 5.3: Average test error across training session for the classification network for 20 dimensional representation.

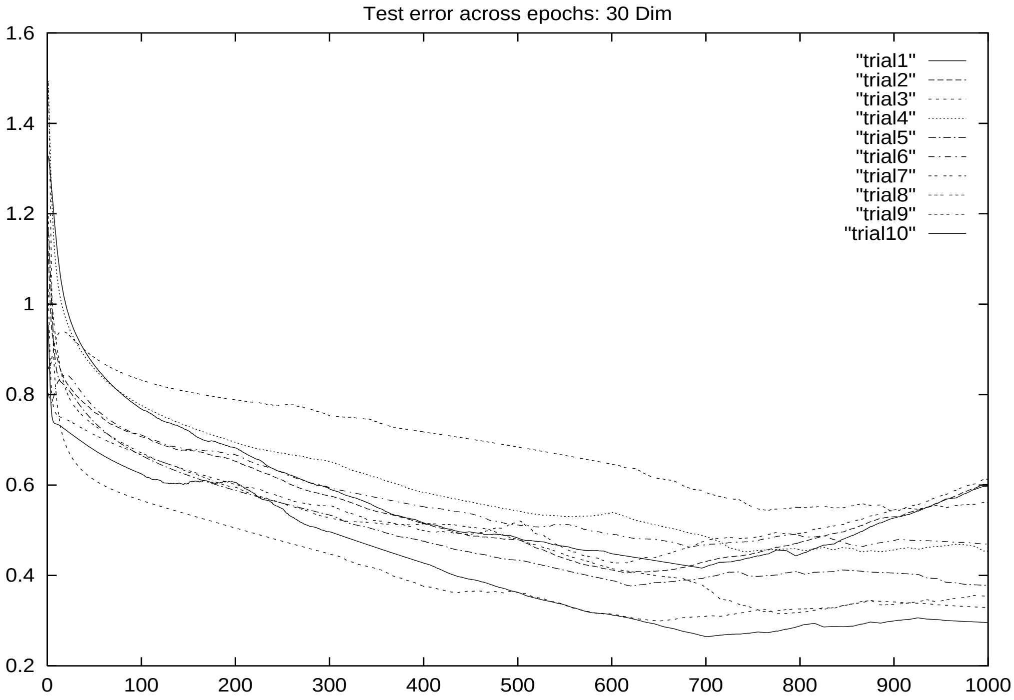

Figure 5.4: Average test error across training session for the classification network for 30 dimensional representation.

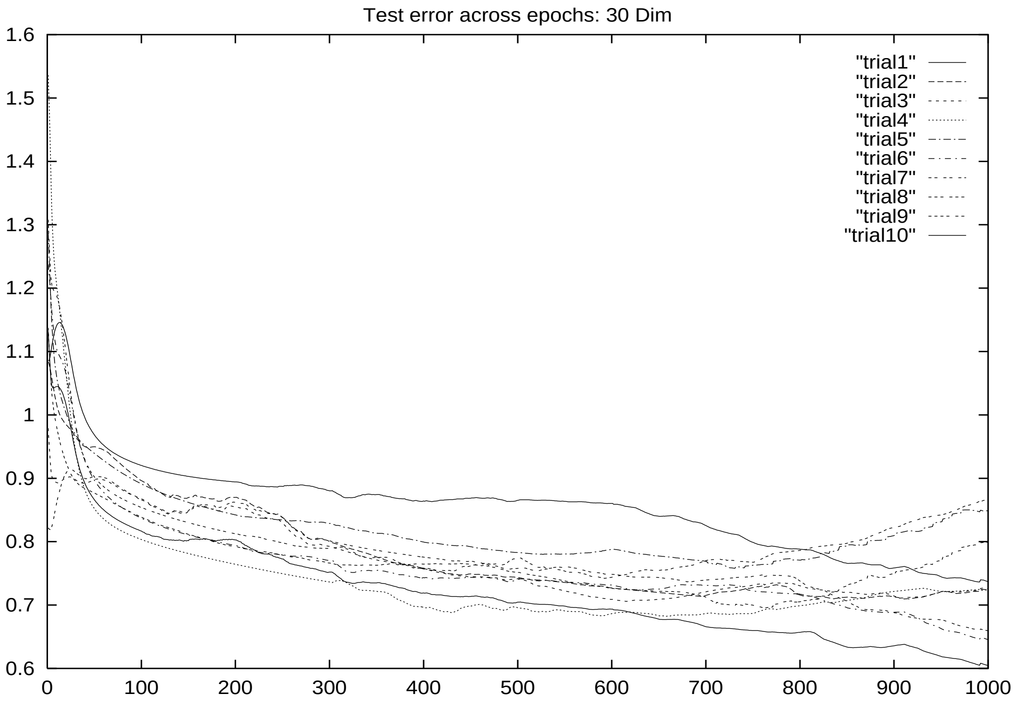

Figure 5.5: Average test error across training session for the classification network for 40 dimensional representation.

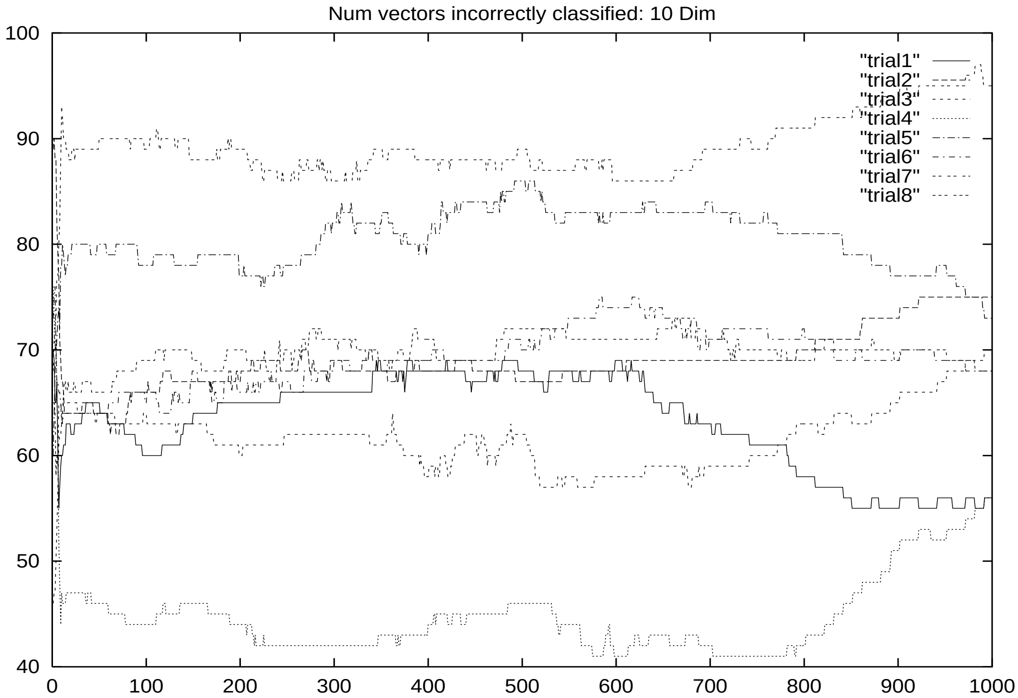

Figure 5.6: Number of test vectors correctly classified across training session for 10 dimensional representation.

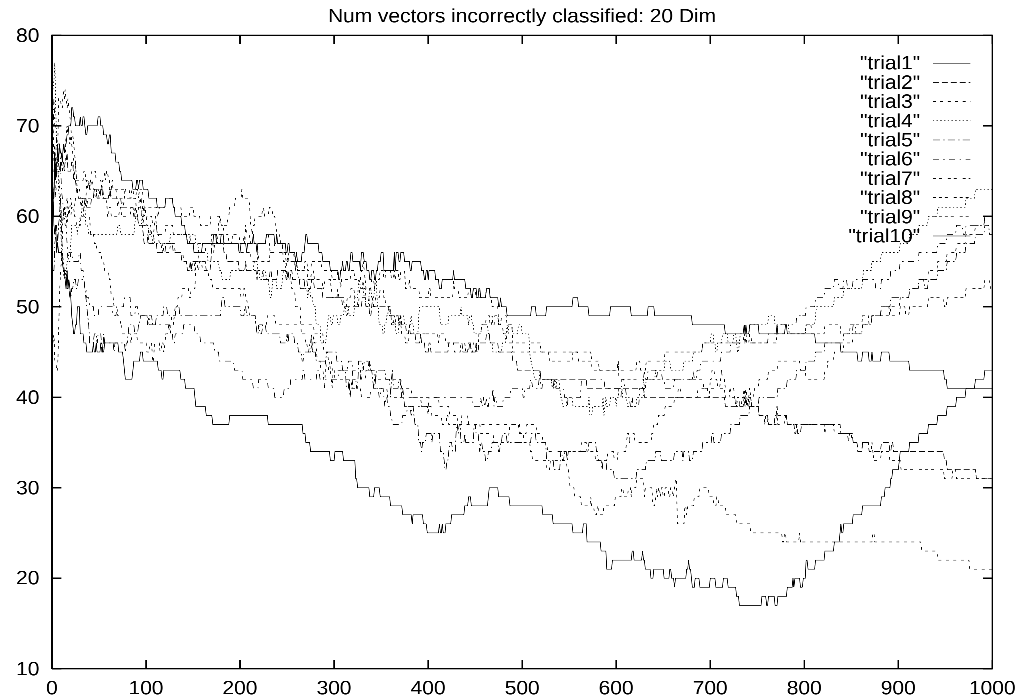

Figure 5.7: Number of test vectors correctly classified across training session for 20 dimensional representation.

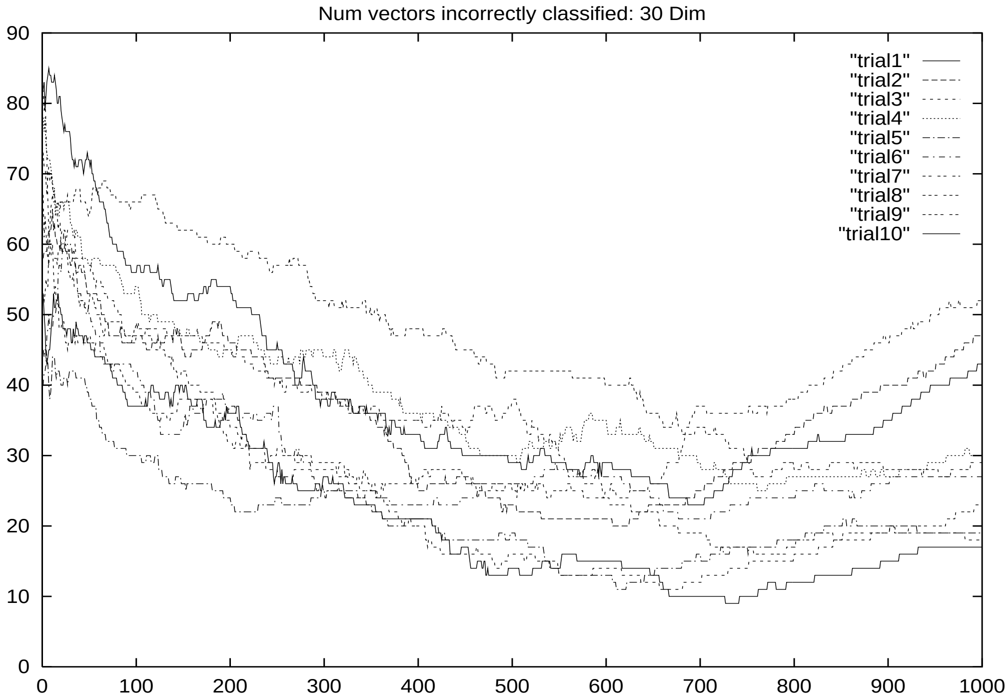

Figure 5.8: Number of test vectors correctly classified across training session for 30 dimensional representation.

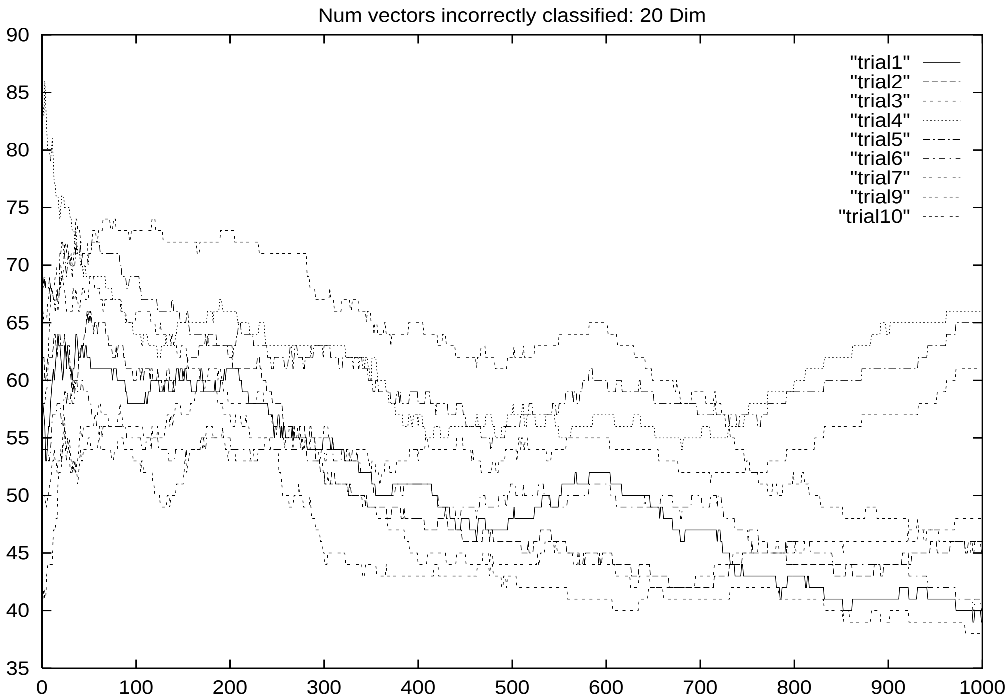

Figure 5.9: Number of test vectors correctly classified across training session for 40 dimensional representation.

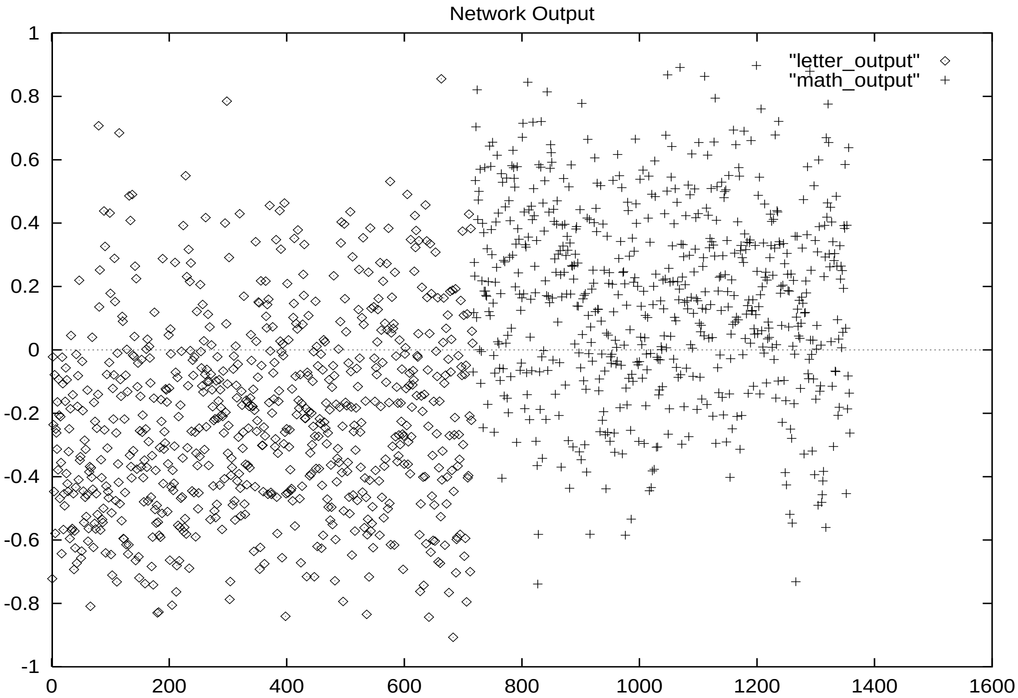

Figure 5.10: Distribution of Output of a trained Classification network for 20 dimensional representation of math and letter task data, all trials.

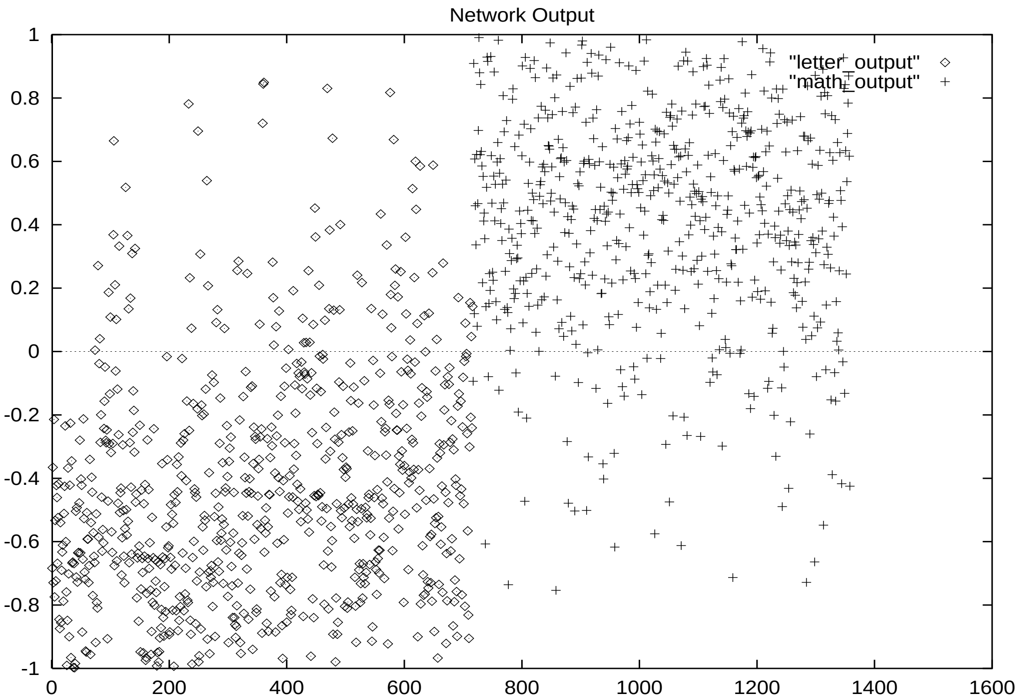

Figure 5.11: Distribution of Output of a trained Classification network for 30 dimensional representation of math and letter task data, all trials.

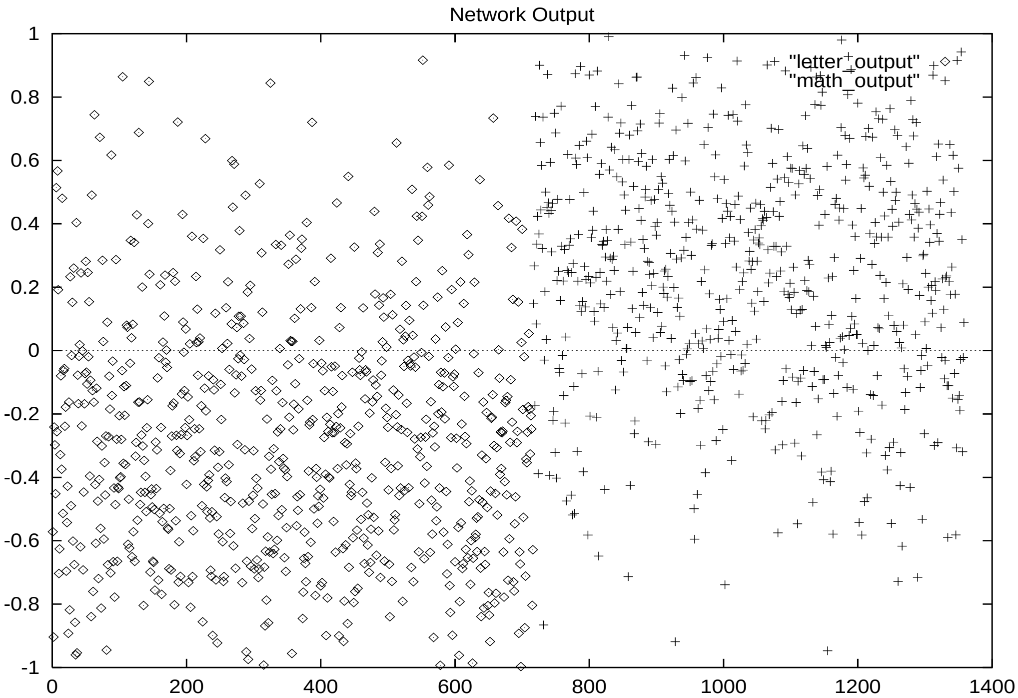

Figure 5.12: Distribution of Output of a trained Classification network for 40 dimensional representation of math and letter task data, all trials.

Figure 5.13 shows the average and range of percentage of correctly classified vectors across trials 1 through 10 using 10, 20, 30 and 40 dimensional vectors. It can be noticed that the range is smallest for 30 dimensional representation and largest for 10 dimensional representation implying that the classification percentages are more consistent for the 30 dimensional representation over all the trials.

5.3 Analysis

The previous section has shown that the percentage of math and letter task vectors correctly classified using the 30 dimensional representation averaged across all the trials is 86.22%. This section discusses a few methods applied to analyze the 30 dimensional representation in an effort to interpret the results of classification. The objective is to study the properties of well classified input vectors (belonging to the 30 dimensional representation of the raw EEG data) for significant features which distinguish between the two tasks.

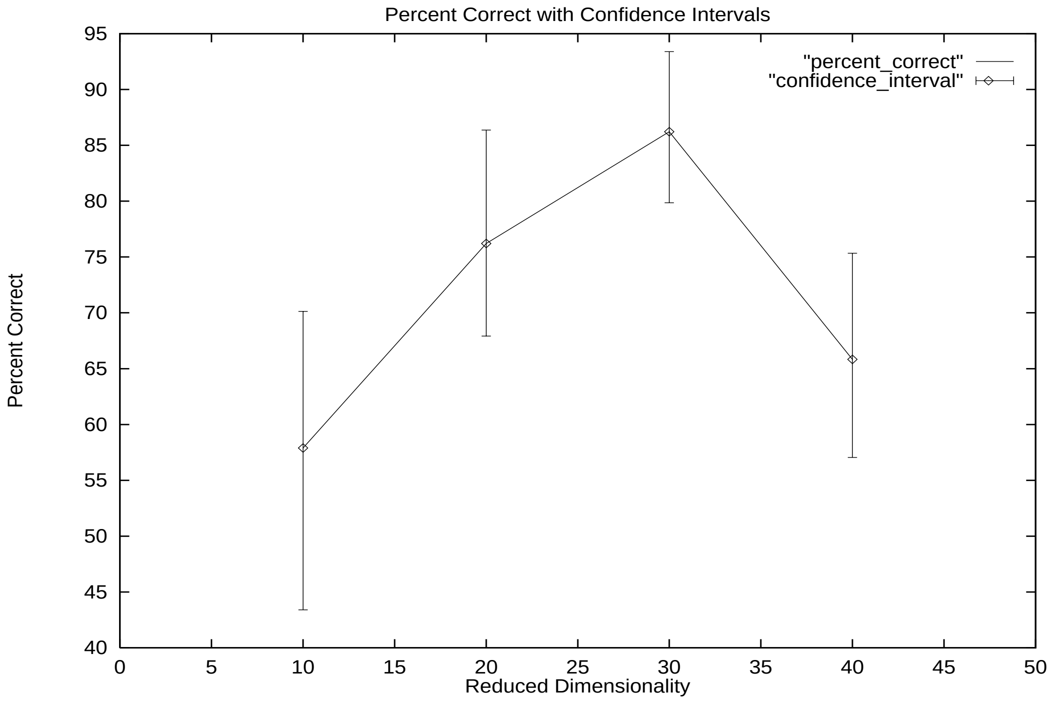

Figure 5.13: Percentage of correctly classified input vectors for different reduced dimensionality representations obtained by NLPCA method.

5.3.1 Ideal Vector analysis

Given the trained weights of the classification network $30-60-1$, a set of n "ideal" input vectors can be obtained for each target by choosing small random values as inputs to the network and training on the inputs. This can be done using standard back-propagation by by adjusting the inputs to the network (rather than the weights) by propagating the difference of the output and target value for the network back to the inputs. The set of n best classified vectors and a set of n worst classified vectors belonging to the actual input data is considered for each task and the average mutual Euclidean distance between the ideal and the best sets and the ideal and the worst sets is compared.

Since a separate trained classification network is associated with each trial, the set of ideal, best and worst classified vectors have to be considered on an individual trial basis.

Table 5.3 shows the values of average mutual distances between the set of ideal vectors and best classified vectors and the set of ideal vectors and worst classified vectors for each trial. Averaged across all the trials, the set of best classified vectors are not significantly closer than the set of worst classified vectors to the set of ideal vectors. Therefore,

Trial #	*Best Classified*	*Worst Classified*
1	4.045	3.788
2	3.872	4.364
3	3.972	3.667
4	3.960	4.079
5	3.701	4.105
6	3.515	4.234
7	3.102	4.259
8	3.749	3.470
9	4.077	4.361
10	3.565	3.844
Average	3.756	4.017

Table 5.3: Average mutual distances between the ideal and the best and worst classified input vectors for both the tasks combined.

Euclidean distances between vectors might not be a measure of how well the vectors are classified. This implies that input vectors which contain prominent features of a particular task may be distributed throughout the k-dimensional space ($k = 30$, because the current analysis deals with 30 dimensional representation of the data).

5.3.2 Input Clustering

Since all input vectors belonging to a task are not necessarily grouped together, clusters of input vectors were considered to determine if the membership of an input vector to a particular cluster is related to how well the input vector is classified as belonging to a particular task.

Therefore, input vectors belonging to math and letter tasks for each trial were divided into n clusters based on their mutual distances and the center of each cluster was determined. The average classification error for input vectors was determined for each individual cluster. This value was compared to classification error for the corresponding center for the cluster.

It was determined that there was not any significant difference in average error of classification for different clusters. Where some difference did exist, no relation was found between the average error in classification for input vectors belonging to a cluster and the error in classification for the cluster center. From this, it can be concluded that the classification of the input vectors was not directly related to the mutual Euclidean distances with other vectors.

Task	*Cluster Number*	*Avg. Error*	*Cluster Center Error*
Letter	1	0.120	0.130
	2	0.111	0.216
	3	0.189	0.214
	4	0.115	0.104
Math	1	0.222	0.149
	2	0.370	0.196
	3	0.336	0.189
	4	0.231	0.159

Table 5.4: Average Classification Error and classification error for the cluster center (trial 1 data vectors) .

5.3.3 Weights of the trained classification network

The weights in a trained classification network can yield useful information about the learning pattern of each node in the network and any easily discernible relationships between components of the input vectors. For example, if the weights into all the hidden nodes from a specific input node are positive, then that input is contributing positively to the output of the network. If there is more than one node in the hidden layer for which all the weights into and out of the node are positive (or negative), then a single node may be able to replace all such nodes.

Figure 5.14 shows the weights of a classification network trained on trial data two through ten. A hollow box represents positive weights and a filled box represents negative weights. The magnitude of the weight is represented by the size of the box. Each row corresponds to all the weights from an input unit or all the weights into an output unit. Each column corresponds to all the weights into and out of a specific hidden unit. According to the previous argument, if multiple columns of weights have the same sign, then the hidden units corresponding to those columns can be replaced by a single hidden unit. From figure 5.14, it can be observed that no two columns come under this category. Therefore, it can be concluded that all the weights are being used in training the classification network. This is true even for 20 and 10 dimensional representations whose weights are shown in figures 5.15 and 5.16 respectively.

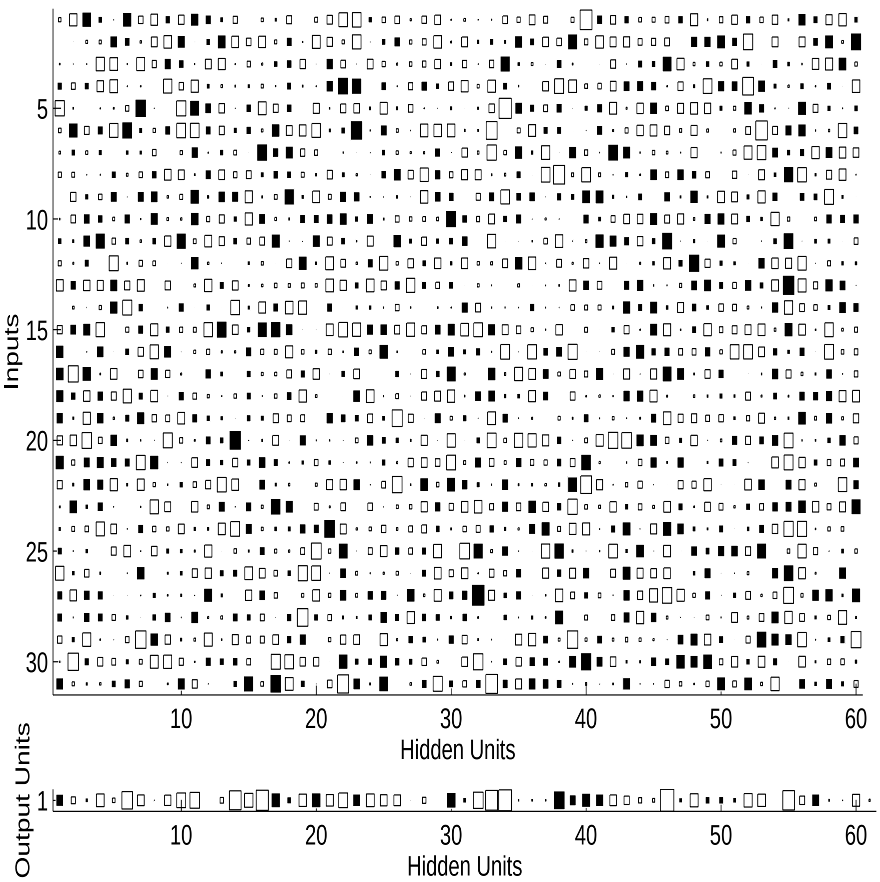

Figure 5.14: The weights of 30-60-1 classification network trained on trials two through ten.

5.3.4 Clustering using Eigenvector Analysis

In the previous section, clustering of inputs according to their mutual Euclidean distances did not yield groups of vectors which have significantly different classification accuracies. This led to the conclusion that classification error does not directly depend on the mutual Euclidean distances between input vectors. Another way of clustering input vectors is by regarding a limited number of mutually orthogonal directions of maximal change in the input vector distribution and to perform clustering in the space formed by using these directions as the primary axes. This can be done by determining the eigenvalues for the covariance of the set of input vectors. These eigenvectors represent the directions of maximum change in input vector distribution. For analyzing the clusters only the first two principal eigenvectors were considered since the input vectors can then

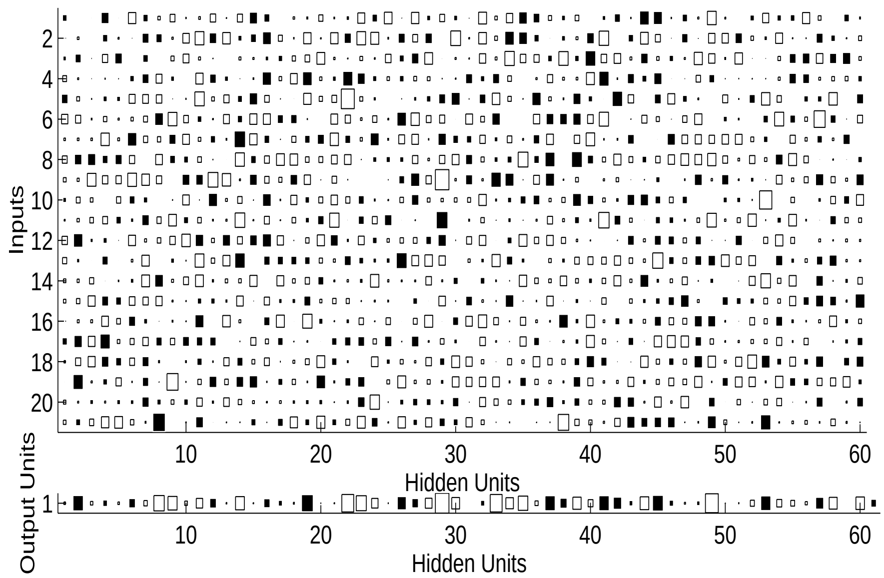

Figure 5.15: The weights of 20-60-1 classification network trained on trials two through ten.

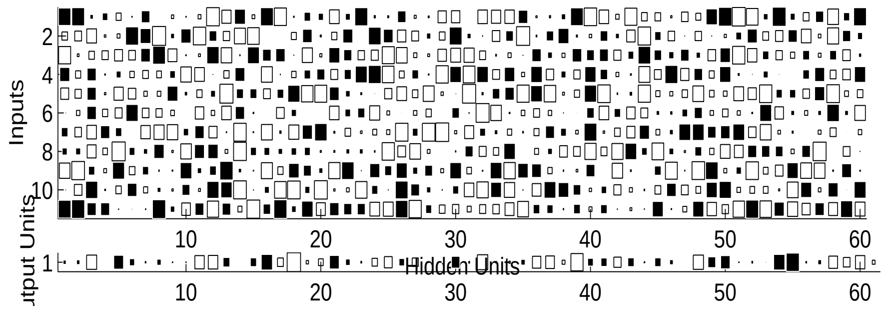

Figure 5.16: The weights of 10-60-1 classification network trained on trials two through ten.

be projected onto a two-dimensional space. The values of the x and y components are the dot products of the input vector with the first two eigenvectors.

Figure 5.17 shows the projection of all the input vectors belonging to trial 1 onto two dimensions. Both the math and letter data vectors seem to be distributed uniformly in space and there is no evident clustering of input vectors according to the task.

However, a definite clustering pattern was observed if only the first few BEST classified vectors are considered for each task. Figures 5.18 and 5.19 show the eigenvector projection of the 10 best classified input vectors belonging to both the tasks for all the 10 trials. The input vectors belonging to math and letter tasks appear to lie in different regions of the 2-dimensional space. This behavior is pronounced for trials 1, 2, 4, 6 and

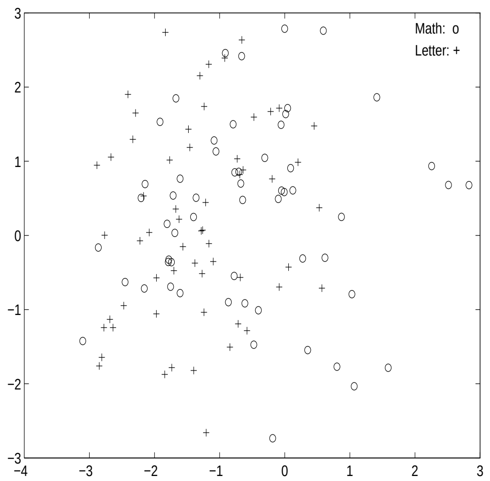

Figure 5.17: Eigenvector projection of input vectors belonging to both tasks for trial one.

7. This shows that the 30 dimensional representation contains some information about the significant features of the tasks which is enabling the projection of input vectors to be clustered according to the tasks. The classification network should be learning the distinction between the input vectors based on these features resulting in an average of 86.22% correctly classified vectors over all the 10 trials.

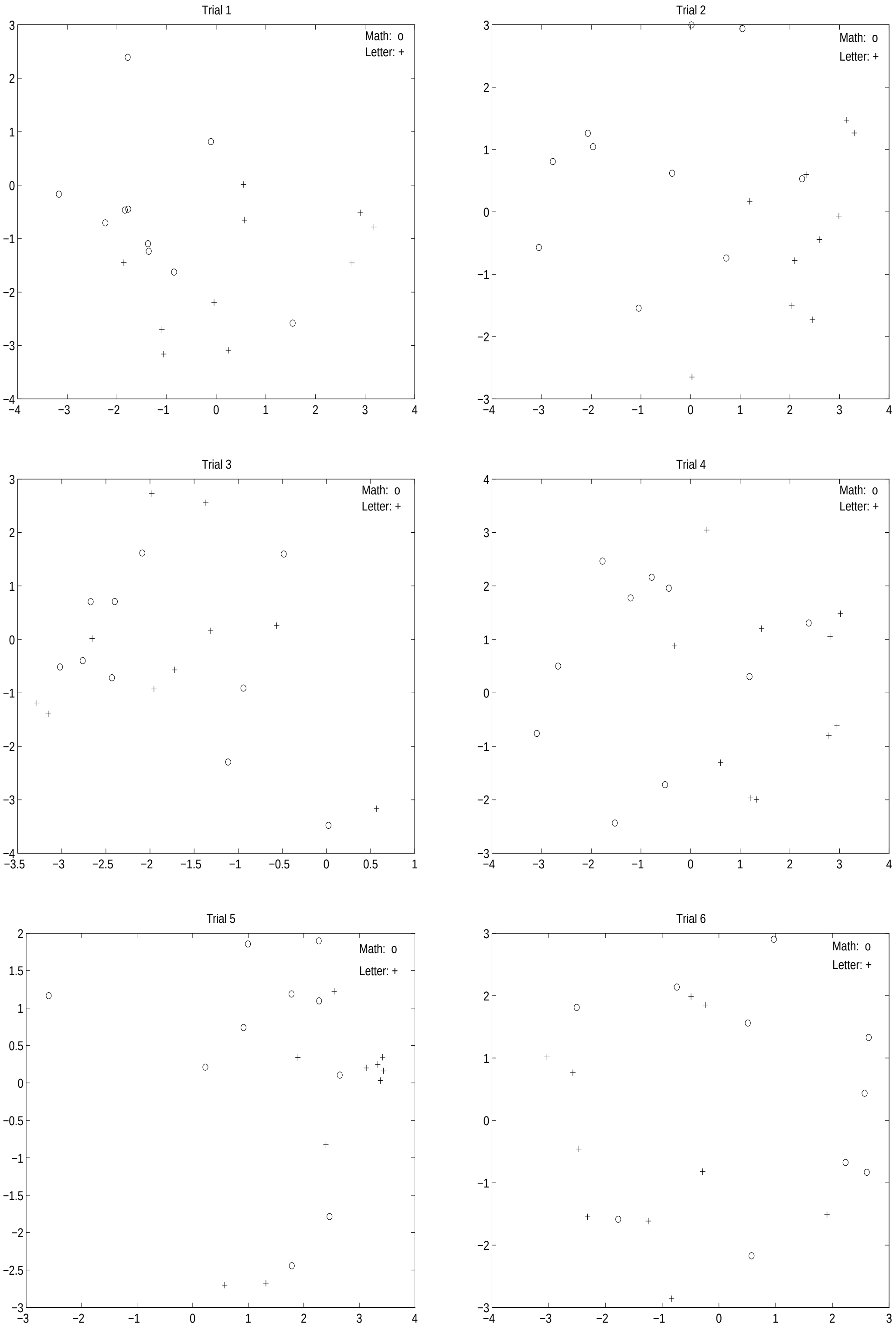

Figure 5.18: Eigenvector projection of 10 best classified input vectors belonging to both tasks .

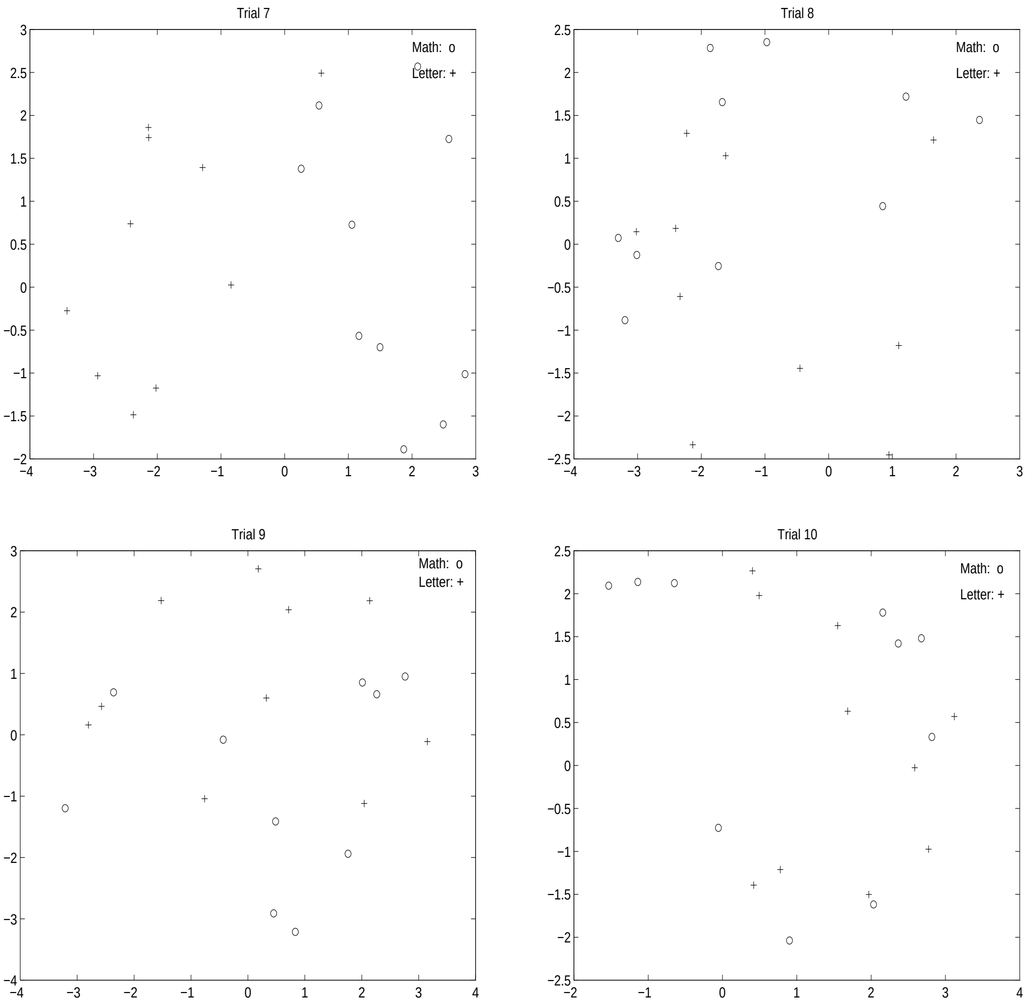

Figure 5.19: Eigenvector projection of 10 best classified input vectors belonging to both tasks .

5.4 K-L representation

A matrix consisting of 1358 input vectors corresponding to quarter second windows of eye-blink free data belonging to letter and math tasks was subject to K-L transform to obtain a reduced dimensionality n ($n = 10, 20, 30, 40$) K-L representation of the data. The classification network $n - 60 - 1$ was trained using K-L representation of trial one data as testing set and K-L representation of trials two through ten as training set. It was observed that there was a bias for the K-L representation vectors to be classified as letter task (66.87% correctly classified vectors) as opposed to math task (39.85% correctly classified vectors). The average correctly classified vectors was only slightly better than chance (55.14%).

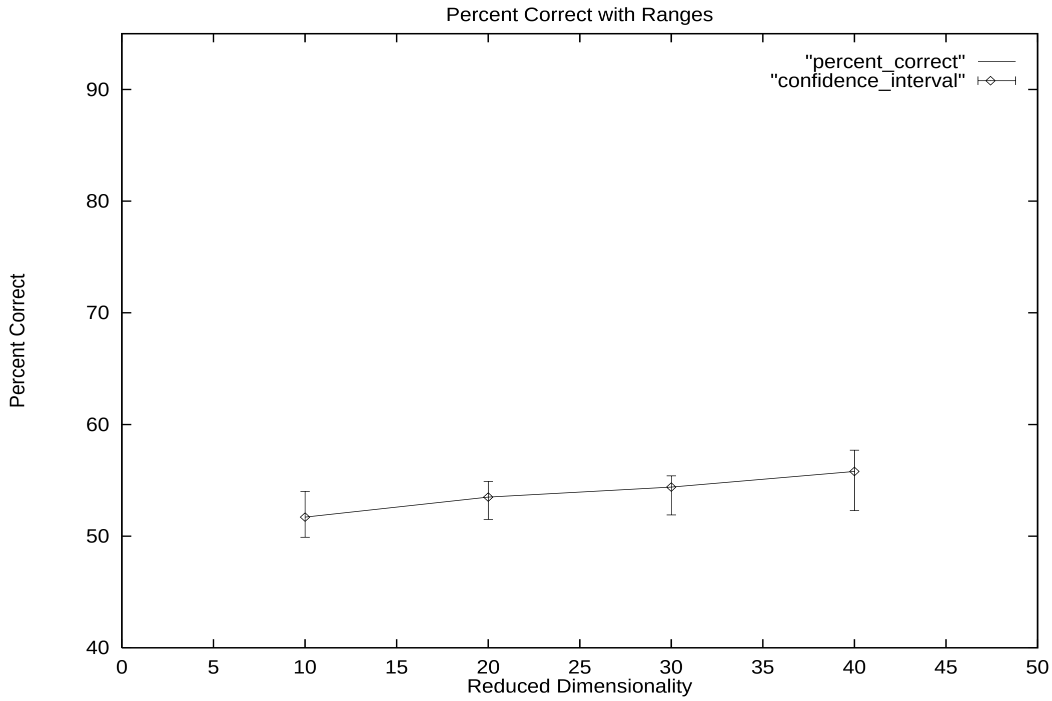

Figure 5.20: Percentage of correctly classified input vectors for different reduced dimensionality representations obtained by K-L transform method.

Table 5.5 shows the percentage correctly classified vectors using NLPCA and K-L Transform representations of input data vectors for reduced dimensions of 10, 20, 30 and 40. It could be notices from Figures 5.13 and 5.20 that the performance of NLPCA is more sensitive to dimensionality of input vectors than is the performance of the K-L representation. The NLPCA method shows a definite improvement in classification over the linear K-L transform method for dimensionality of 30. The K-L representation has a low percentage of correctly classified vectors for all values of n used here and also does not show much variation across different trials.

# bottlenecks	*NLPCA*	*K-L Transform*
10	57.8	51.7
20	76.2	53.5
30	86.2	54.4
40	65.8	55.8

Table 5.5: Percentage of correctly classified vectors using NLPCA and K-L Transform representations of input data vectors.

Chapter 6

CONCLUSION

The purpose of this project was to investigate the effectiveness of dimensionality reduction using autoassociative networks as a preprocessing step for classification of EEG signals into one of two mental tasks. This is part of a larger project whose objective is to determine the feasibility of using mental tasks as an alphabet for controlling a device such as a wheelchair by a physically handicapped person.

General methods applied in classifying EEG signals involve K-L Transform and frequency analysis. The K-L Transform determines a linear mapping of raw EEG data onto a reduced dimensional space. In this work, the NLPCA method was applied to determine a non-linear mapping onto a reduced dimensional space which lead to lower dimensional data consisting of significant features which make it easier to classify the EEG signal into two different tasks. The tasks chosen were "letter composition" and "arithmetic problem solving" because they were considered to involve significantly different mental processes. An autoassociative network was first trained to generate a reduced dimensional representation of the original EEG data, each data vector consisting of a temporal window of raw 6-channel data. The reduced dimensional representation was then used in classifying the original data into one of the two tasks using a standard back propagation network. Both the autoassociative and the classification networks used the conjugate gradient method for faster convergence. Multiple trials of data were considered and in all training experiments, the training and testing sets were formed using data belonging to different trials so that classification is based on features that exist across all the trials. The reduced dimensionality representation of the original data was subject to clustering and eigenvector analysis to study the behavior of the best classified input vectors belonging to both the tasks.

The 30-dimensional representation of the EEG data vectors yielded an average percentage of correctly classified vectors of 86.22% over all the trials. This was followed by the 20-dimensional representation with 76.21%, 40-dimensional representation with 65.83% and 10-dimensional representation with 57.89%. All trial data performed consistently well using the 30-dimensional representation. The input vectors to the classification network (which are also the reduced dimensional representations of the EEG data vectors), as such were uniformly distributed in the n dimensional space ($n = 10, 20, 30, 40$) and the mutual Euclidean distances between the vectors was not directly related to how well they were classified (Indicative of how prominent the distinguishing features are, in the input vectors) or which task they belonged to. The classification networks corresponding to all dimensions seem to employ all the hidden unit weights in the network. When the first n best classified input vectors belonging to both the tasks are projected along the direction of the first two eigenvectors (representing the two orthogonal directions of maximum change in the input vector distribution) onto a two dimensional space, the vectors begin to form independent clusters for each task as the value of n decreases. Chapter 5 illustrates the clustering of vectors for value of $n = 10$. This indicates that the best classified vectors contain features which strongly differentiate between the two tasks supporting the 86.22% of correctly classified vectors for the 30-dimensional representation. The K-L representation of original data vectors however, gives an average percent correctly classified vectors of 55.14%.

The NLPCA method performed well in generating a reduced dimensional representation of raw EEG data which resulted in a high percentage of correctly classified vectors for math and the letter composition tasks. The performance of the autoassociative network in extracting the discriminatory features was found to be very sensitive to the reduced dimensionality used. For a 10-dimensional representation, the classification network performed only slightly better than chance. The conjugate gradient method helped the NLPCA network converge in a reasonably small number of epoch. The NLPCA method proved to be a definite improvement over the K-L transform method of preprocessing signals before classification which yielded a percent correctly classified vectors of only slightly better than chance. The use of windows of EEG data enabled the temporal correlations between the different channels to be considered during the reduction of dimensionality of the EEG data. Previous attempts to apply dimensionality reduction to individual samples of 6-dimensional raw data (corresponding to 6 channels) using an NLPCA network made it impossible for the network to converge.

One of the drawbacks of the NLPCA method as opposed to other preprocessing techniques is the large training time needed because of the large number of weights. The work presented here focuses only on two tasks and a single subject. A more rigorous test for classification should involve training and testing across multiple subjects and using multiple tasks. There are other novel enhancements to the NLPCA network like circular nodes which determine mapping of the signal onto a circular interval which show definite improvement in mapping functions that are homeomorphic to a circle. Future work could focus on employing circular nodes in bottleneck layer of the NLPCA networks to obtain a reduced dimensionality representation of the EEG signal. This could lead to much better classification results if the EEG signals have some components that are homeomorphic to a circle. Sequential NLPCA which enforces the order in which the bottleneck units are trained could be extremely helpful in avoiding the competition between the bottleneck nodes to learn the same features. Another technique that could be applied during dimensionality reduction is pruning in which the bottleneck nodes could be removed if the weights do not contribute to the training of the network. This could be a solution to the large training time for the NLPCA network and also lead to the determination of the "optimal" dimensionality of the windowed EEG data. Future work can also be focused on improved techniques for data collection by increasing the number of channels to 19 given by the 10/20 system enabling signals to be recorded from more areas of the brain than the standard electrode positions used in this work. Future research is needed to employ and develop new techniques in investigating the problem of classifying multiple tasks across different subjects.

REFERENCES

[1] Donald L. Birx and Stephen J. Pipenberg. A complex mapping network for phase sensitive classification. *IEEE transactions on Neural Networks*, 4(1):127–135, 1993.

[2] Erik A. Stolz Charles W. Anderson, Saikumar V. Devulapalli. Determining mental state from EEG signals uing parallel implementations of neural networks. *Scientific Programming*, 4:171–183, 1995.

[3] David DeMers and Garrison Cottrell. Non-linear dimensionality reduction. *Neural Information Processing Systems*, 5:582, 1992.

[4] Kenji Doya. Dimension reduction of biological neuron models by artificial neural networks. *Neural Computation*, 6:696–717, 1994.

[5] R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *Computer Journal*, 7:149–154, 1964.

[6] Fukunaga and Koontz. Application of Karhunen-Loève expansion to feature selection and ordering. *IEEE transactions on Computers*, C(19), 1974.

[7] Nemoto Iku and Kono Tomoshi. Complex neural networks. *Systems and Computers in Japan*, 23(8):75–83, 1992.

[8] C. Jutten and J. Herault. Blind seperation of sources, part i: An adaptive algorithm based on neuromimetic architecture. *Signal Processing*, 24:1–10, 1991.

[9] Juha Karhunen and Jyrki Joutsensalo. Representation and separation of signals using nonlinear PCA type learning. *Neural Networks*, 7(1):113–127, 1994.

[10] Z. A. Keirn and J. I. Aunon. A new mode of communication between man and his surroundings. *IEEE Transactions on Biomedical Engineering*, 37:1209–1214, 1990.

[11] M. Kirby and L. Sirovich. Application of the Karhunen-Loève procedure for the characterization of human faces. *IEEE trans. PAMI*, 12(1):103., 1990.

[12] Michael J. Kirby and Rick Miranda. Circular nodes in neural networks. *Neural Computation*, 8(2):390–402, 1996.

[13] M A Kramer. Improvement of the backpropagation algorithm for training neural networks. *Computer in Chemical Engineering*, 14(3):337–341, 1990.

[14] M A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *American Institute of Chemical Engineering Journal*, 37(2):233–243, 1991.

[15] M A Kramer. Autoassociative neural networks. *Computers and Chemical Engineering*, 16(4):313–328, 1992.

[16] S. Nakagawa, Y.Ono, and Y. Hirata. Dimensionality reduction of dynamical patterns using a neural network. In B.H. Huang, S.Y. Kung, and C.A. Kamm, editors, *Neural Networks for Signal Processing I*, pages 256–265, 1991.

[17] E. Oja. Neural networks, principal components and subspaces. *International Journal of Neural Systems*, 1:61–68, 1989.

[18] E. Oja. Data compression, feature extraction, and autoassociation in feedforward neural networks. In T. Kohonen, K. Mäkisara., O. Simula, and J. Kangas, editors, *"Artificial Neural Networks"*, pages 737–745, NY, 1991. Elsevier Science Publishers.

[19] E. Oja. Principal components, minor components and neural networks. *Neural Networks*, 5:927–935, 1992.

[20] Terence D. Sanger. Optimal hidden units for two-layer nonlinear feedforward neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 5(4):545–461, 1991.

[21] Eric Saund. Dimensionality-reduction using connectionist networks. *IEEE Transactions on Pattern Analysis and Machine Intellignence*, 11(3):304–314, 1989.

[22] L. Sirovich and M. Kirby. A low-dimensional procedure for the characterization of human faces. *J. of the Optical Society of America A*, 4:529., 1987.

[23] Shiro Usui, Shigeki Nakauchi, and Masae Nakano. Internal color representation by a five-layer neural network. In T. Kohonen, K. Mäkisara, O.Simula, and J. Kangas, editors, *Artifical Neural Networks*, pages 867–872. Elsevier Science Publishers, North Holland, 1991.

[24] Ray White. Competitive hebbian learning: Algorithm and demonstrations. *Neural Networks*, 5:261–275, 1992.