THESIS

DIMENSIONALITY REDUCTION AND CLASSIFICATION OF TIME EMBEDDED EEG SIGNALS

Submitted by

Mohammad Nayeem Teli

Department of Computer Science

In partial fulfillment of the requirements

for the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Summer 2007

COLORADO STATE UNIVERSITY

July 05, 2007

WE HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER OUR SUPERVISION BY MOHAMMAD NAYEEM TELI ENTITLED DIMENSIONALITY REDUCTION AND CLASSIFICATION OF TIME EMBEDDED EEG SIGNALS BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE.

Committee on Graduate Work

Committee Member

Committee Member

Adviser

Co-Adviser

Department Head

ABSTRACT OF THESIS

DIMENSIONALITY REDUCTION AND CLASSIFICATION OF TIME EMBEDDED EEG SIGNALS

Electroencephalogram (EEG) is the measurement of the electrical activity of the brain measured by placing electrodes on the scalp. These EEG signals give the micro-voltage difference between different parts of the brain in a non-invasive manner. The brain activity measured in this way is being currently analyzed for a possible diagnosis of physiological and psychiatric diseases. These signals have also found a way into cognitive research. At Colorado State University we are trying to investigate the use of EEG as computer input.

In this particular research our goal is to classify two mental tasks. A subject is asked to think about a mental task and the EEG signals are measured using six electrodes on his scalp. In order to differentiate between two different tasks, the EEG signals produced by each task need to be classified. We hypothesize that a bottleneck neural network would help us to classify EEG data much better than classification techniques like Linear Discriminant Analysis(LDA), Quadratic Discriminant Analysis (QDA), and Support Vector Machines.

A five layer bottleneck neural network is trained using a fast convergence algorithm (variation of Levenberg-Marquardt algorithm) and Scaled Conjugate Gradient (SCG). Classification is compared between a neural network, LDA, QDA and SVM for both raw EEG data as well as bottleneck layer output. Results indicate that QDA and SVM do better classification of raw EEG data without a bottleneck network. QDA and SVM always achieved higher classification accuracy than the neural network with a bottleneck layer in all our experiments. Neural network was able to achieve its best classification accuracy of 92% of test samples correctly classified, whereas QDA achieved 100% accuracy in classifying the test data.

Mohammad Nayeem Teli
Department of Computer Science
Colorado State University
Fort Collins, CO 80523
Summer 2007

ACKNOWLEDGEMENTS

To begin with I want to thank ALLAH, my creator. After that, I would like to thank my adviser Dr. Charles Anderson for his directions, suggestions and feedback without which the completion of this thesis would not have been possible. I would also like to thank Dr. Michael Kirby for various insights during our regular EEG group meetings. Additional thanks to Dr. Ross McConnell for accepting a position on my committee.

DEDICATION

This thesis is dedicated to my family, friends and all those who helped me complete it.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

The human brain is a complex organ with approximately 100 billion nerve cells called neurons. Neurons gather and transmit electrochemical signals. No matter what state we are in, whether asleep, awake or anesthetized, our brain produces some kind of waves which can be observed and used for research. One of the most widely used methods of capturing brain activity is using EEG.

Living brains having some kind of brain activity was known to many as early as the nineteenth century. However a German psychiatrist named Hans Berger was the first to record this electrical activity in 1928. Ever since its discovery EEG has been used to diagnose many medical conditions like epilepsy, identify the location of a suspected brain tumor, or a disease in the brain such as Parkinson's disease. It is a non-invasive method of measuring electrical activity of the brain. Electrodes are placed on the scalp and the brainwaves recorded.

A large amount of research is underway to interpret these waves and find ways to utilize them. In computer science EEG is being used for a Brain Computer Interface (BCI). This is a technology which can be very beneficial for people with motor disabilities. It will provide them with a new powerful communication and control technique. People severely disabled by amyotrophic lateral sclerosis (ALS), brain stem stroke, cerebral palsy, and other neuromuscular disorders would greatly benefit by advances in BCI research and development [bci06].

The research in this thesis is an attempt to understand the EEG waves and to classify them. The EEG for this study was observed in subjects performing mental tasks. The results of this research could be integrated with the current Brain Computer Interface (BCI) project being undertaken at the Colorado State University (CSU). It could possibly be used by paralyzed people, or by anyone for rudimentary user-interface actions, like popping up windows or making menu choices. In order for us to classify and visualize EEG these signals first need to go through dimensionality reduction.

1.1 Dimensionality Reduction

Dimensionality reduction is a method of obtaining the information from a high dimensional feature space using fewer intrinsic dimensions. In machine learning it is very important to reduce high dimensional data set for better classification, regression, presentation and visualization of data. It is also useful for better understanding of the correlations within the data. This enables us to find the intrinsic dimensionality of the dataset and better generalization.

Multiple approaches could be used for dimensionality reduction which may be linear or non-linear. One of the most common linear techniques used is the Principal Components Analysis (PCA). It is also known as the Hotelling [Hot53], or Karhunen and Loeve (KL) [Kar47][Loe48] transform. PCA is a linear transformation of a data set from high dimensional space to its principal components representing the data set in lower dimensions. It typically exploits the linear relationship between the data variables. PCA has not been very useful where the relationship between the variables is non-linear. For example, a helix formed by a sine over a cosine wave in $\Re^3$ would be represented by PCA in three dimensions. The real dimensionality of such a helix would be one [DC93].

In order to avoid such situations when the data have non-linear relationships, we use non-linear methods of dimensionality reduction. One way to perform the non-linear methods of dimensionality reduction is by using neural networks. A large amount of research has gone into studying multi-layer neural networks. Kramer [Kra91] proposed a Non Linear Principal Component Analysis (NLPCA) approach for training the feed-forward neural network to obtain an identity mapping. The network has multiple hidden layers with a middle ”bottleneck” layer. The output of the bottleneck layer represents the lower dimensional representation of the input data set. Such a neural network with an identity mapping, $f(x) = x$, has been very successful in obtaining a reduced representation of the input data. The basic goal is to map the input to the output with minimal error. The success of this goal means that the intrinsic dimensionality of the data is equal to the number of units in the bottleneck layer. Oja [Oja91] used a five layer neural network for non-linear dimensionality reduction. He was able to compute smooth nonlinear mapping between the inputs and the central hidden layer, and another mapping between the central hidden layer and the outputs. Oja further further showed that a sigmoid function in the hidden layer units enables learning the structure of complex non-linear inputs embedded within a noisy environment. Usui et al. [UNN91] used a five layer neural network for dimensionality reduction using a non-linear approach as well. They showed that a non-linear representation of original data can be found in lower dimensional space with multiple hidden layers. They successfully used their technique in a color application , which reduces the dimensionality of the color space of a multi-spectral sensor. Their approach uses the fact that color measurements form a manifold of three intrinsic dimensions. Moreover, it has been found that the modes of variation correspond to psychological color attributes of humans.

We have been mentioning the bottleneck layer network but haven’t really explained it. We think this is the right place to present it. A bottleneck layer network has an input layer and an output layer with hidden layers between them. The middle layer characterized as a bottleneck layer has the lowest number of hidden units. Figure 1.1

represents one such network. In this figure the input layer is the input data layer, and

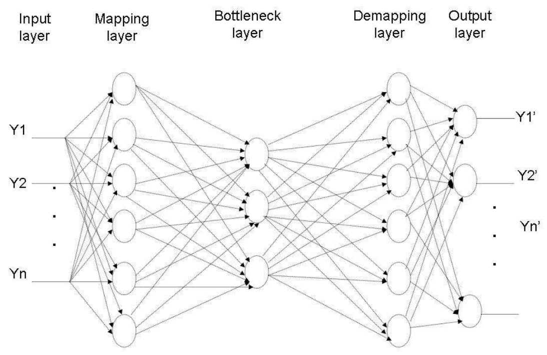


Figure 1.1: Bottleneck Neural network.

the mapping layer reduces the dimensionality to the number of units in the bottleneck layer. The bottleneck layer has the least number of units in the network. The demapping layer changes the lower dimensional output of the bottleneck layer to the dimensionality of the mapping layer and then the output layer finally remaps the dimensionality of the output to that present in the input dataset. Although, this network has only three hidden layers we could have more than that in a neural network depending on the dimensionality reduction technique being used. The dots in the input and the output layers indicate that there are more units in these layers depending on the dimensionality of the dataset. The Output of each unit in the hidden layers is used as an input to all the units of the next layer in this network. The input to each layer from all units of the previous layer is multiplied by the weight of the link between the hidden units, summed up and passed through a transfer function. One of the most widely used transfer function in a non-linear neural network is a sigmoid function. The sigmoid function is a typical non-linear transfer function that helps make outputs reachable. The neural network output could be passed through a sigmoid function or not depending upon the design of the solution.

While all the above approaches have hidden layers with nodes representing real numbers, Kirby et al.[KM96] suggested a circular node representation in the bottleneck layer. This circular node stores and transmits angular information.

One of the earliest approaches in neural network training has been the use of back-propagation algorithm. However, it has been seen to have slow convergence. LeCun et al. [LBOM98] suggested that with an efficient choice of parameters backpropagation algorithm can be used very efficiently. They further suggested that for a large and redundant training set stochastic gradient with careful tuning is a good choice or one could use the Levenberg-Marquardt algorithm. For a small data set they suggested the use of conjugate gradient.

The Levenberg-Marquardt algorithm [Lev44] [Mar63] is an iterative algorithm. It could be referred to as an interpolation of Gauss-Newton and the steepest descent methods. More recently, Hinton et al. [HS06] used a pretraining method to place the weights near good solutions, followed by a gradient descent fine tuning method for training multi-layer neural network. They used restricted Boltzmann machine to pretrain their networks.

1.2 EEG research at Colorado State University

Researchers at CSU have studied various aspects of EEG. Knight [Kni03] analyzed eye movements, eye blinks, muscles, heart beats and other electrical sources affecting EEG recordings. He investigated approaches based on generalized singular value decomposition. Peterson et al. [PKK$^+$05] indicated that Blind Source Separation(BSS) and feature selection could be used to improve the performance of even a direct single-session BCI. They used a modified genetic algorithm wrapped around a support vector machine classifier to search the space of feature subsets. Anderson et al. [AP01] found that auto regressive (AR) models of six channel EEG results in the best classification accuracy. They were able to identify with 70% accuracy of five cognitive tasks a person is doing for two of four subjects tested, and near 40% for the other two. Anderson et al. [ASS98] compared raw data, power spectral densities, Karhunen-Loeve transforms, and AR models as signal representations. They found that AR models performed the best. Anderson et al. [ADS95] compared four representations of EEG signals and their classification by a two-layer neural network with sigmoid activation functions. They concluded that the frequency-based representation results in significantly more accurate classification than the unprocessed or K-L representations.

1.3 Dimensionality Reduction and Classification of EEG Data

EEG data also needs some dimensionality reduction before it could be classified. This dimensionality reduction of EEG data is very important to visualization and representation. Dimensionality reduction also becomes important because the number of electrodes could vary from 6 to 256. EEG signals are noisy. There is a possibility of much correlation because brain electrical activity spreads through the brain volume. In addition there could be correlation among different channels. These correlations might be temporally separated. Dimensionality reduction could remove such correlations. It is also important because an understanding of EEG data would mean that it could help in brain computer interface research.

However, it is a difficult problem. The problem of unfolding the inherent representation of the brain activity using the EEG signals is complicated. Dimensionality reduction and classification of EEG using neural networks has been done before by many researchers. The problem that has been tried by researchers earlier is the classification of mental tasks. In order to classify the mental tasks, the EEG signals corresponding to them are first fed to a bottleneck network and the bottleneck output fed to a classifier. Keirn et al. [KA90] studied a set of tasks comprised of baseline (no activity), mental arithmetic, geometric figure rotation, mental letter composing and visual counting. Each task had ten trials. In each trial 2500 samples were observed over 10 seconds.

Devulapalli [Dev96] used the same data as used by them and classified mental arithmetic and mental letter composing tasks. This research also used the same dataset and it is a follow-up to what Devulapalli did. He obtained good results and our goal would be to repeat what was done and go beyond by using better training algorithms. In his conclusion he reported that NLPCA requires a long training time because of the sheer number of weights. We shall use a training algorithm that is known to be faster than a simple conjugate gradient algorithm used by Devulapalli. We will discuss more about Devulapalli's approach and how the approach of this research is different from his, in the next section.

1.4 Prior Results with Bottleneck Network on EEG Classification

Devulapalli [Dev96] used Non-Linear Principal Components Analysis (NLPCA) for dimensionality reduction. In his experiments he used temporal windows of EEG data. Each window was 62 samples long (approximately corresponding to a quarter second) and consecutive windows were overlapped by 31 samples. Since each window consisted of 62 samples per channel and 6 channels were used, the dimensionality of the dataset used was, 62 x 6 = 372. 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. However, sample points corresponding to all six channels falling in the region of eye blinks were removed from the raw EEG data before being subjected to any processing.

Ten sessions were conducted for each task with each session lasting for a period of ten seconds which results in 2500 samples per task per session. This was followed by a classification of the mental tasks using a standard backpropagation neural network trained using conjugate gradient algorithm. Devulapalli reported a classification accuracy of 86.22% for a 30-dimensional bottleneck layer representation of the EEG data 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%.

In the current work our goal is to build upon this previous research and to classify EEG signals using the same approach and data as used earlier. However, we will be using a variant of Levenberg-Marquardt algorithm [WIKE01], and Scaled Conjugate gradient (SCG) [Møl93] as preprocessing steps to train the neural network. For classification we will try to classify different tasks using neural network, Linear Discriminant Analysis (LDA), Quadratic discriminant Analysis (QDA) and Support Vector Machine (SVM). The number of tasks to be classified would depend on how well our network represents the EEG data. We will be training our five layer neural network with a hidden bottleneck layer. The number of bottleneck units will be varied to find the correct number of units in the bottleneck layer for an efficient representation of the input data. The number of units will be chosen on the basis of how well the bottleneck network maps the input to the output data.

1.5 Organization of the Rest of Thesis

The rest of the thesis layout is as follows: Chapter 2 covers the mathematical background of steepest descent, Gauss-Newton, Levenberg-Marquardt algorithm and its variant, fast convergence algorithm, which is the algorithm being used in this work. The second neural network training algorithm SCG is also covered. LDA and QDA and SVM are also discussed.

In Chapter 3 the methods for describing the data, the tasks and the experiments are presented. Chapter 4 contains the results of the experiments. Chapter 5 is comprised of conclusions and discussion of the results.

Chapter 2

Mathematical Background

In this chapter we describe the background of the algorithms we used for this work. For training the multi-layer neural network we used two different algorithms: a Fast Convergence Algorithm Based on Levenberg-Marquardt (FCALM) [WIKE01] and Scaled Conjugate Gradient algorithm (SCG) [Møl93]. For classification, we used Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA) and Support Vector Machines (SVM). The following sections describe the basis and the mathematics behind the algorithms.

The training of a bottleneck multi-layer neural network involves a search for an optimized set of weights between the units of different layers in the network. An optimized set of weights is such that the squared residual error between the input and the output is minimal. It would vary from one problem set to another but when the output is nearly equal to the input, it means we have reached close to the intrinsic dimensionality at the bottleneck layer and the weights are optimized. So our objective function is the squared error between the input and the output and we want to minimize it. This requires that we find a good set of weights. In order to find this set of good weights the training algorithm has to traverse through a complex residual error landscape. The goal is to find a global minima of this landscape. Typically, the problem is reduced to minimizing a function

F(w), which is a sum of squares of nonlinear functions

$$F(w) = \frac{1}{2}\sum_{i=1}^{m} f_i(w)^2 = \frac{1}{2}||f(w)||^2 \tag{2.1}$$

The *i*-th component of the m-vector *f(w)* is the function $f_i(w)$, and $\| \ f(w) \ \|$ is termed the residual at *w*. The fraction $\frac{1}{2}$ has been included to avoid the appearance of a factor of two in the derivatives [GMW81].

In order to solve the least squares problem of (2.1) we will need to find the derivatives of *F* [GMW04].[1] Provided that *f* has continuous second partial derivatives, we can write its *Taylor expansion* as

$$f(w+h) = f(w) + J(w)h + O(||h||^2), \tag{2.2}$$

where $J \epsilon \mathbb{R}^{m*n}$ is the *Jacobian*. This is a matrix containing the first partial derivatives of the function components,

$$(J(w))_{ij} = \frac{\partial f_i}{\partial w_j}(w). \tag{2.3}$$

As regards F:$\mathbb{R}^n \rightarrow \mathbb{R}$, it follows from the first formulation in (2.1), that

$$\frac{\partial F(w)}{\partial w_j} = \sum_{i=1}^{m} f_i(w)\frac{\partial f_i}{\partial w_j}(w). \tag{2.4}$$

Thus, the gradient is

$$F'(w) = J(w)^T f(w). \tag{2.5}$$

We also need the Hessian of *F*. From (2.4) we see the element in position (*j*,*k*) is

$$\frac{\partial^2 F}{\partial w_j \partial w_k}(w) = \sum_{i=1}^{m}(\frac{\partial f_i}{\partial w_j}(w)\frac{\partial f_i}{\partial w_k}(w) + f_i(w)\frac{\partial^2 f_i}{\partial w_j \partial w_k}(w)), \tag{2.6}$$

showing that

$$F''(w) = J(w)^T J(w) + \sum_{i=1}^{m} f_i(w) f_i''(w). \tag{2.7}$$

[1]Most of the equations here are based on [GMW04]

2.1 The Gauss-Newton Method

The Gauss-Newton method is the basis of the Levenberg-Marquardt algorithm and its variation. It is based on a linear approximation to the components of **f** (a linear model of **f**) in the neighborhood of **w**: For small ||**h**|| we see from the Taylor expansion (2.2) that

$$f(w+h) \simeq \ell(h) \equiv f(w) + J(w)h. \tag{2.8}$$

Inserting this in the definition (2.1) of F we see that

$$\begin{aligned} F(w+h) \simeq L(h) &\equiv \frac{1}{2}\ell(h)^T\ell(h) \\ &= \frac{1}{2}f^Tf + h^TJ^Tf + \frac{1}{2}h^TJ^TJh \\ &= F(w) + h^TJ^Tf + \frac{1}{2}h^TJ^TJh \end{aligned} \tag{2.9}$$

(with **f** = **f(w)** and **J** = **J(w)**). The *Gauss-Newton step* h_{gn} minimizes L(h),

$$h_{gn} = \text{argmin}_h\{L(h)\}.$$

It is easily seen that the gradient and the Hessian of L are

$$L'(h) = J^Tf + J^TJh, \quad L''(h) = J^TJ. \tag{2.10}$$

Comparison with (2.5) shows that $L'(0) = F'(w)$. Further, we see that the matrix $L''(h)$ is independent of h. It is symmetric and if J has full rank, i.e., if the columns are linearly independent, then $L''(h)$ is also positive definite. This implies that $L(h)$ has a unique minimizer, which can be found by solving

$$(J^TJ)h_{gn} = -J^Tf. \tag{2.11}$$

This is a descent direction for F since

$$h_{gn}^TF'(w) = h_{gn}^T(J^Tf) = -h_{gn}^T(J^TJ)h_{gn} < 0. \tag{2.12}$$

The typical step is

$$\begin{aligned} &Solve \quad (J^T J)h_{gn} = -J^T f \\ &w := w + \alpha h_{gn} \end{aligned} \tag{2.13}$$

where α is found by line search. The classical Gauss-Newton method uses α = 1 in all steps. The method with line search can be shown to have guaranteed convergence, provided that $\{w|F(w) \le F(w_0)\}$ is bounded, where w_0 is the previous position, and the Jacobian $J(w)$ has full rank in all steps. Unlike Newton's method, Gauss-Newton does not have quadratic convergence. One shortcoming of the Gauss-Newton method is that the matrix $J(w)^T J(w)$ may not be positive definite. This could be avoided by using Levenberg [Lev44] and Marquardt [Mar63] modification described in the next section.

2.2 The Levenberg-Marquardt Method

For continuity we will explain this method based on [GMW04] and follow it by the definition provided by [WIKE01] and their modification. Levenberg [Lev44] and later Marquardt [Mar63] suggested to use a damping term in the Gauss-Newton method which modifies (2.11) to

$$(J^T J + \mu I)h_{lm} = -g \quad \text{with} \quad g = J^T f \quad \text{and} \quad \mu \ge 0. \tag{2.14}$$

Here, $J = J(w)$ and $f = f(w)$. The damping parameter μ has several effects:

1. For all $\mu > 0$ the coefficient matrix, $(J^T J + \mu I)$, is positive definite, and this ensures that h_{lm} is a descent direction,

2. For large values of μ we get

$$h_{lm} \simeq -\frac{1}{\mu} g = -\frac{1}{\mu} F'(w),$$

i.e., a short step in the steepest descent direction. This is good if the current iterate is far from the solution.

3. If μ is very small, then $h_{lm} \simeq h_{gn}$, which is a good step in the final stages of the iteration, when w is close to w^*, where w^* is the minima. If $F(w^*) = 0$ (or very small), then we can get (almost) quadratic final convergence.

Thus, the damping parameter influences both the direction and the size of the step, and this leads us to make a method without a specific line search. The choice of initial μ-value should be related to the size of the elements, $a_{ij}^{(0)}$, A_0 = $J(w_0)^T J(w_0)$, for example by letting

$$\mu_0 = \tau \cdot \max_i \{a_{ii}^{(0)}\}, \tag{2.15}$$

where τ is chosen by the user. During iteration the size of μ can be updated. The updating is controlled by the gain ratio

$$\rho = \frac{F(w) - F(w + h_{lm})}{L(0) - L(h_{lm})},$$

where the denominator is the gain predicted by the linear model

$$\begin{aligned} L(0) - L(h_{lm}) &= -h_{lm}^T J^T f - \frac{1}{2} h_{lm}^T J^T J h_{lm} \\ &= -\frac{1}{2} h_{lm}^T (2g + (J^T J + \mu I - \mu I) h_{lm}) \\ &= \frac{1}{2} h_{lm}^T (\mu h_{lm} - g). \end{aligned}$$

Note that both $h_{lm}^T h_{lm}$ and -$h_{lm}^T g$ are positive, so $L(0) - L(h_{lm})$ is guaranteed to be positive.

A large value of ρ indicates that $L(h_{lm})$ is a good approximation to $F(w + h_{lm})$, and we can decrease μ so that the next Levenberg-Marquardt step is closer to the Gauss-Newton step. If ρ is small (maybe even negative), then $L(h_{lm})$ is a poor approximation, and we should increase μ with the twofold aim of getting closer to the steepest descent direction and reducing the step length.

The stopping criteria for the algorithm is such that at global minimum we have $F'(w^*) = g(w^*) = 0$, so we can use

$$\|g(w^*)\|_\infty \leq \epsilon_1, \tag{2.16}$$

where ϵ_1 is a small, positive number, chosen by the user. Another relevant criterion is to stop if the change in w is small,

$$\|w_{new} - w\| \leq \epsilon_2(\|w\| + \epsilon_2). \tag{2.17}$$

This expression gives a gradual change from relative step size ϵ_2 when $\|w\|$ is large to absolute size ϵ_2^2 if w is close to 0. In the above equation w_{new} is the new position.

The above description is a more general representation of Levenberg-Marquardt algorithm. Below, we will reproduce how [WIKE01] presented the algorithm and their modifications. The variable w will represent the weights of the links between the units of different layers of the neural network. Wilamowski et al. [WIKE01] presented the optimization as a performance index in the form

$$F(w) = \sum_{p=1}^{P} \left[\sum_{k=1}^{K} (d_{kp} - o_{kp})^2 \right] \tag{2.18}$$

where $w = [w_1 w_2 ... w_N]^T$ consists of all weights of the network, d_{kp} is the desired value of the k^{th} output and the p^{th} pattern, o_{kp} is the actual value of the k^{th} output and the p^{th} pattern, N is the number of weights, P is the number of patterns, and K is the number of the network outputs. Equation (2.18) can be written as

$$F(w) = E^T E \tag{2.19}$$

where

$$E = [e_{11} \quad ... \quad e_{K1} \quad e_{12} \quad ... \quad e_{K2} \quad ... \quad e_{1P} \quad ... \quad e_{KP}]^T$$

$$e_{kp} = d_{kp} - o_{kp}, \quad k = 1, ..., K, \quad p = 1, ..., P$$

where E is the cumulative error vector (for all patterns). From equation (2.19) the Jacobian matrix is defined as

$$J = \begin{bmatrix} \frac{\partial e_{11}}{\partial w_1} & \frac{\partial e_{11}}{\partial w_2} & \cdots & \frac{\partial e_{11}}{\partial w_N} \\ \frac{\partial e_{21}}{\partial w_1} & \frac{\partial e_{21}}{\partial w_2} & \cdots & \frac{\partial e_{21}}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_{K1}}{\partial w_1} & \frac{\partial e_{K1}}{\partial w_2} & \cdots & \frac{\partial e_{K1}}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_{1P}}{\partial w_1} & \frac{\partial e_{1P}}{\partial w_2} & \cdots & \frac{\partial e_{1P}}{\partial w_N} \\ \frac{\partial e_{2P}}{\partial w_1} & \frac{\partial e_{2P}}{\partial w_2} & \cdots & \frac{\partial e_{2P}}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_{KP}}{\partial w_1} & \frac{\partial e_{KP}}{\partial w_2} & \cdots & \frac{\partial e_{KP}}{\partial w_N} \end{bmatrix} \tag{2.20}$$

and the weights are calculated using the following equation

$$w_{t+1} = w_t - \left(J_t^T J_t + \mu_t I\right)^{-1} J_t^T E_t \tag{2.21}$$

where **I** is identity unit matrix, μ is a learning parameter and **J** is a Jacobian of m output errors with respect to n weights of the neural network. Jacobian **J** is calculated at each iteration step and so is the inversion of $J^T J$ square matrix with dimension $N \times N$. In order to reduce this dimensionality Wilamowski et al. [WIKE01] suggested a modification which is described in the section below.

2.2.1 Wilamowski et al. Modification of Levenberg-Marquardt Algorithm

Wilamowski et al. [WIKE01] changed the performance index equation (2.18) to

$$F(w) = \sum_{k=1}^{K} \left[\sum_{p=1}^{P} (d_{kp} - o_{kp})^2 \right]^2 \tag{2.22}$$

This represents the global error and reduces the dimensionality of the matrix to be inverted at each iteration step. Equation (2.22) can also be written as:

$$F(w) = \hat{E}^T \hat{E} \tag{2.23}$$

where

$$\hat{E} = [\hat{e_1} \quad \hat{e_2} \quad \cdots \quad \hat{e_K}]^T \quad and \quad \hat{e_k} = \sum_{p=1}^{P} (d_{kp} - o_{kp})^2$$

with $k = 1, \ldots, K$. The modified Jacobian matrix $\hat{J}_t$ as reported by Wilamowski et al. [WIKE01] is represented as

$$\hat{J}_t = \begin{bmatrix} \frac{\partial \hat{e_1}}{\partial w_1} & \frac{\partial \hat{e_1}}{\partial w_2} & \cdots & \frac{\partial \hat{e_1}}{\partial w_N} \\ \frac{\partial \hat{e_2}}{\partial w_1} & \frac{\partial \hat{e_2}}{\partial w_2} & \cdots & \frac{\partial \hat{e_2}}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial \hat{e_K}}{\partial w_1} & \frac{\partial \hat{e_K}}{\partial w_2} & \cdots & \frac{\partial \hat{e_K}}{\partial w_N} \end{bmatrix} \tag{2.24}$$

Using the modified Jacobian matrix equation (2.21) can rewritten as

$$w_{t+1} = w_t - \left(\hat{J}_t^T \hat{J}_t + \mu_t I \right)^{-1} \hat{J}_t^T \hat{E}_t \tag{2.25}$$

Wilamowski et al. [WIKE01] report that the advantage of this modification is that $\hat{J}_t$ is now K by N matrix. However, since the above equation still needs to invert an N by N matrix they used Matrix Inversion Lemma, according to which if a matrix **A** satisfies

$$A = B^{-1} + CD^{-1}C^T \tag{2.26}$$

then

$$A^{-1} = B - BC \left(D + C^T BC \right)^{-1} C^T B. \tag{2.27}$$

Applying this lemma to the term within the parentheses of equation (2.25) we get,

$$\left(\hat{J}_t^T \hat{J}_t + \mu_t I \right)^{-1} = \frac{1}{\mu_t} I - \frac{1}{\mu_t^2} \hat{J}_t^T \left(I + \frac{1}{\mu_t} \hat{J}_t \hat{J}_t^T \right)^{-1} \hat{J}_t \tag{2.28}$$

Now the right side of equation (2.28) reduces to the size K by K. According to Wilamowski et al. [WIKE01], this significantly reduces the computational complexity of the weight adaptation problem. Therefore, with this change the weight update equation becomes

$$w_{t+1} = w_t - \left[\frac{1}{\mu_t} I - \frac{1}{\mu_t^2} \hat{J}_t^T \left(I + \frac{1}{\mu_t} \hat{J}_t \hat{J}_t^T \right)^{-1} \hat{J}_t \right] \hat{J}_t^T \hat{E}_t \tag{2.29}$$

2.3 Scaled Conjugate Gradient

Scaled conjugate gradient (SCG) [Møl93] method belongs to the class of conjugate gradient methods. It uses second order information from the neural network but requires only $O(N)$ memory usage, where N is the number of weights in the network. It avoids time consuming line search of other conjugate gradient methods and finds good solutions even in an error landscape with ravines.

Let w_i be a vector from the space $\mathbb{R}^N$, where *N* is the sum of the number of weights and of the number of the biases of the network. Let *E* be the error function we want to minimize. This algorithm differs from the other conjugate gradient methods in the following ways.

- Each iteration *k* of a conjugate gradient method (CGM) computes $w_{k+1} = w_k + \alpha_k p_k$, where p_k is a new conjugate direction, and α_k is the size of the step in this direction. Actually α_k is a function of $E''(w_k)$, the Hessian matrix of the error function, namely the matrix of the second derivatives. In contrast to the other CGMs which avoid the complex computation of the Hessian and approximate α_k with a time consuming line search procedure, SCG makes the following simple approximation of the term s_k, a key component of the computation of α_k:

$$s_k = E''(w_k)p_k \approx \frac{E'(w_k + \sigma_k p_k) - E'(w_k)}{\sigma_k}, 0 < \sigma_k \ll 1$$

- As the Hessian is not always positive definite, which prevents the algorithm from achieving good performance, SCG uses a scalar λ_k which is regulates the indefiniteness of the Hessian. This is done by a setting:

$$s_k = \frac{E'(w_k + \sigma_k p_k) - E'(w_k)}{\sigma_k} + \lambda_k p_k$$

  and adjusting λ_k at each iteration. This is the main contribution of SCG to both fields of neural learning and optimization theory.

2.4 Linear Discriminant Analysis

In this section the discussion with be based mostly on [HTF01]. Linear Discriminant Analysis (LDA) is used to find a linear combination of features to classify two or more different classes. It assumes that each class is multivariate Gaussian with the distribution [HTF01]

$$f_k(x) = \frac{1}{(2\pi)^{\frac{p}{2}}|\Sigma_k|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k)}, \tag{2.30}$$

where mean of class k, $\mu_k = \sum_{g_i=k} \frac{x_i}{N_k}$, x_i represents the *i-th* observation of class k, N_k is the number of class k observations, K is the number of classes, N is the total number of observations, and Σ_k is the covariance of class k.

For optimal classification we need to find the class posteriors $Pr(G|X)$. Suppose in class $G = k$, the class-conditional density of X be defined by f_k in equation (2.30) with $\sum_{k=1}^{K} \pi_k = 1$ where π_k is the mean of class k. An application of Bayes theorem gives us

$$Pr(G = k|X = x) = \frac{f_k(x)\pi_k}{\sum_{l=1}^{K} f_\ell(x)\pi_\ell}. \tag{2.31}$$

LDA assumes that the classes have a common covariance matrix $\Sigma_k = \Sigma, \forall k$. In order to compare two classes k and l, we look at the log-ratio

$$\begin{aligned}\log \frac{Pr(G = k|X = x)}{Pr(G = \ell|X = x)} &= \log \frac{f_k(x)}{f_l(x)} + \log \frac{\pi_k}{\pi_\ell} \\ &= \log \frac{\pi_k}{\pi_l} - \frac{1}{2}(\mu_k + \mu_\ell)^T\Sigma^{-1}(\mu_k - \mu_\ell) + x^T\Sigma^{-1}(\mu_k - \mu_\ell),\end{aligned} \tag{2.32}$$

which is a linear equation in x. The assumption in linear log odds ratio is that the decision boundary between classes k and ℓ is linear [HTF01]. In $\mathbb{R}^P$ the classes will be separated by hyperplanes. Based on equation (2.32) the linear discriminant functions

$$\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \log \pi_k \tag{2.33}$$

are an equivalent description of the decision rule, with $G(x)$ = argmax$_k\delta_k(x)$. The parameters of the Gaussian distribution are calculated from the training data, with $\Sigma = \sum_{k=1}^{K}\sum_{g_i=k}(x_i-\mu_k)(x_i-\mu_k)^T/(N-K)$.

2.5 Quadratic Discriminant Analysis

Quadratic Discriminant Analysis (QDA) is a classification technique which separates two or more classes using a quadratic surface. Unlike LDA, there is no assumption that the covariance Σ_k of different classes is equal. The discriminant function is defined as

$$\delta_k(x) = -\frac{1}{2}\log|\Sigma_k| - \frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) + \log\pi_k. \tag{2.34}$$

The decision boundary between each pair of classes *k* and ℓ is described by a quadratic equation $x : \delta_k(x) = \delta_\ell(x)$. For a detailed discussion on LDA and QDA refer to [HTF01].

2.6 Support Vector Machines

Support Vector Machines (SVM) are classifiers which produce non linear boundaries by constructing a linear boundary in a large, transformed version of the feature space [HTF01]. Here we will present the details as described by Hastie et al. [HTF01]. For an exhaustive treatment see [Vap95].

Suppose we want to separate two classes,

$$(x_1, y_1), \ldots, (x_l, y_l), x\epsilon\mathbb{R}^n, y\epsilon\{-1,+1\}, \tag{2.35}$$

with a hyperplane

$$\{x : f(x) = x^T\beta + \beta_0 = 0\}. \tag{2.36}$$

Optimal separation requires that the group of vectors be optimally separated without error. This implies that

$$\min_{\beta,\beta_0} \|\beta\|$$
$$\text{subject to} \quad y_i(x_i^T\beta + \beta_0) \geq 1, \quad i = 1, \ldots, N.$$

The distance $d(\beta, \beta_0; x)$ of a point *x* from the hyperplane (β, β_0) is,

$$d(\beta, \beta_0; x) = \frac{|\beta \cdot x + \beta_0|}{\|\beta\|} \tag{2.37}$$

The optimal hyperplane is obtained by maximizing the margin, $\rho(\beta, \beta_0)$. The margin is given by,

$$\begin{aligned}
\rho(\beta, \beta_0) &= \min_{\{x_i : y_i = 1\}} d(\beta, \beta_0; x_i) + \min_{\{x_j : y_j = -1\}} d(\beta, \beta_0; x_j) \\
&= \min_{\{x_i : y_i = 1\}} \frac{|\beta \cdot x_i + \beta_0|}{\|\beta\|} + \min_{\{x_j : y_j = -1\}} \frac{|\beta \cdot x_j + \beta_0|}{\|\beta\|} \\
&= \frac{1}{\|\beta\|} \left(\min_{\{x_i : y_i = 1\}} |\beta \cdot x_i + \beta_0| + \min_{\{x_j : y_j = -1\}} |\beta \cdot x_j + \beta_0| \right) \\
&= \frac{2}{\|\beta\|}
\end{aligned} \tag{2.38}$$

Therefore, the hyperplane that separates the data is the one that minimizes

$$\phi(w) = \frac{1}{2}\|\beta\|^2. \tag{2.39}$$

In order to understand how equation (2.39) is equivalent to finding the optimal hyperplane, suppose that the following bound holds,

$$\|\beta\| \leq C. \tag{2.40}$$

Then,

$$d(\beta, \beta_0; x) \geq \frac{1}{C}. \tag{2.41}$$

where C is the gap between the optimized hyperplane and the closest class observation on either side of the separating hyperplane. Accordingly, the hyperplanes cannot be nearer than 1/C to any of the data points. The solution to the optimization problem of equation (2.39) is given by the saddle point of the Lagrange functional

$$L(\beta, \beta_0, \alpha) = \frac{1}{2}\|\beta\|^2 - \sum_{i=1}^{l} \alpha_i[(x_i \cdot \beta) + \beta_0]y_i - 1. \tag{2.42}$$

where α_i are the Lagrange multipliers. We minimize this Lagrange function w.r.t. β and β_0. Setting the respective derivatives to zero, we get

$$\beta = \sum_{i=1}^{N} \alpha_i y_i x_i, \tag{2.43}$$

$$0 = \sum_{i=1}^{N} \alpha_i y_i, \tag{2.44}$$

By substituting (2.43) and (2.44) in (2.42), we obtain the Lagrangian (Wolfe) dual objective function

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} x_i^T x_{i'}, \tag{2.45}$$

and the constraints are

$$\begin{aligned} &\alpha_i \geq 0, \quad i = 1, \dots, N \\ &\sum_{i=1}^{N} \alpha_i y_i = 0 \end{aligned} \tag{2.46}$$

Solving eq (2.45) with constraints of (2.46) determines the Lagrange multipliers, and the optimal separating hyperplane is given by,

$$\hat{\beta} = \sum_{i=1}^{N} \hat{\alpha}_i y_i x_i, \hat{\beta}_0 = -\frac{1}{2}\hat{\beta}\cdot [x_r + x_s] \tag{2.47}$$

where x_r and x_s are any support vector from each class satisfying,

$$\hat{\alpha_r}, \hat{\alpha_s} > 0, \quad y_r = 1, \quad y_s = -1. \tag{2.48}$$

The hard classifier is then,

$$\begin{aligned} \hat{G}(x) &= \text{sign}[\hat{f}(x)] \\ &= \text{sign}[x^T\hat{\beta} + \hat{\beta}_0]. \end{aligned} \tag{2.49}$$

Alternatively, a soft classifier may be used which linearly interpolates the margin,

$$\hat{G}(x) = H(\hat{\beta}\cdot x + \hat{\beta}_0) \quad \text{where} \quad H(x) = \begin{cases} -1, & x < 1; \\ x, & -1 \leq x \leq 1; \\ 1, & 1. \end{cases} \tag{2.50}$$

This soft classifier produces a real valued output between -1 and 1 when the classifier is queried within the margin, where no training data resides.

Chapter 3

Methods

In this chapter our main focus will be to describe the various methods that were used in our experiments.

3.1 Electroencephalogram (EEG) data

In this research work we used the data collected by Keirn et al. [KA90]. This data was collected using the 10-20 system of electrode placement [Jas58]. These positions are shown in Figure 3.1 This is a a standardized system of measuring EEG based on area

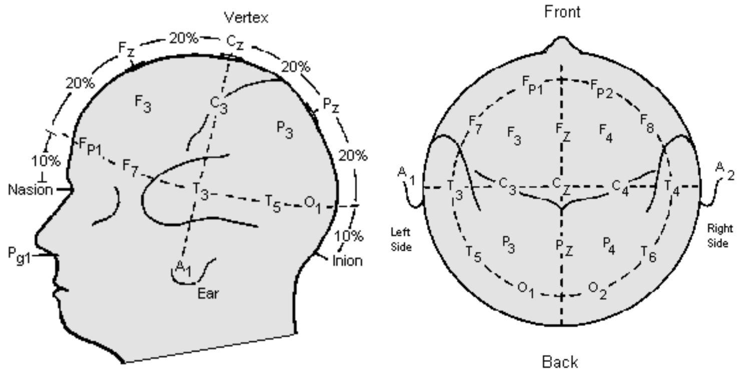


Figure 3.1: 10-20 system of electrode positions for EEG [KA90].

of the cerebral cortex and the position of the electrodes on the scalp. Each letter represents the location of the brain with the number indicating the hemisphere, and z the mid line. The letters F, T, C, P, and O represent frontal, temporal, central, parietal and occipital parts of the brain. The data for this study was measured at six sites: C3, C4, P3, P4, O1, and O2. The data was stored at a frequency of 250 samples per second. Sets of ten second trials were recorded for each of five mental tasks: resting task, imagined letter writing, mental multiplication, visualized counting, and geometric object rotation. These tasks were chosen to exhibit different spatial patterns for the purpose of classification.

In this research we undertake the study of only two tasks: letter writing and mental arithmetic. For the imagined letter writing task, the subject was asked to compose a letter without vocalizing it. In subsequent trials the subject was asked to resume the letter from a previous starting point. In the mental arithmetic task each subject was asked to multiply two numbers, with the numbers being different in the different trials. The subjects were asked not to vocalize the numbers and to start each trial in the letter composition task from where the previous trial had left off [Kni03].

3.1.1 Data Representation

The tasks data could be graphically represented in Figures 3.2. In each figure the letters with the numbers under the signals represent the electrodes from which the brainwaves were measured. The electrooculograph (EOG) shown in each plot represent the eye movements.

3.1.2 Neural Network

The neural network we used for this thesis is a five layer network represented in Figure 3.3. The dots indicate that there could be many more units in each one of those layers. In our research the input is a six dimensional dataset. The number of units in the hidden layers and the bottleneck layer are chosen through pilot experiments to get the best configuration. The main focus of the experiments, however, was to find the best number of the bottleneck units.

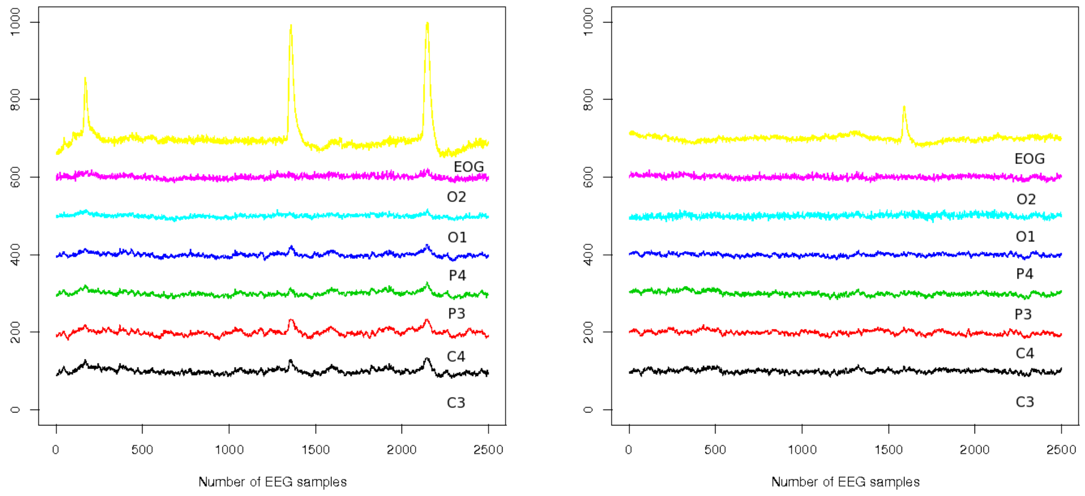


Figure 3.2: The figure on the left represents letter composition task and the one on the right is the mental arithmetic task. Each figure shows the six electrode outputs and EOG. These output values vary between -50 to +50 micro volts.

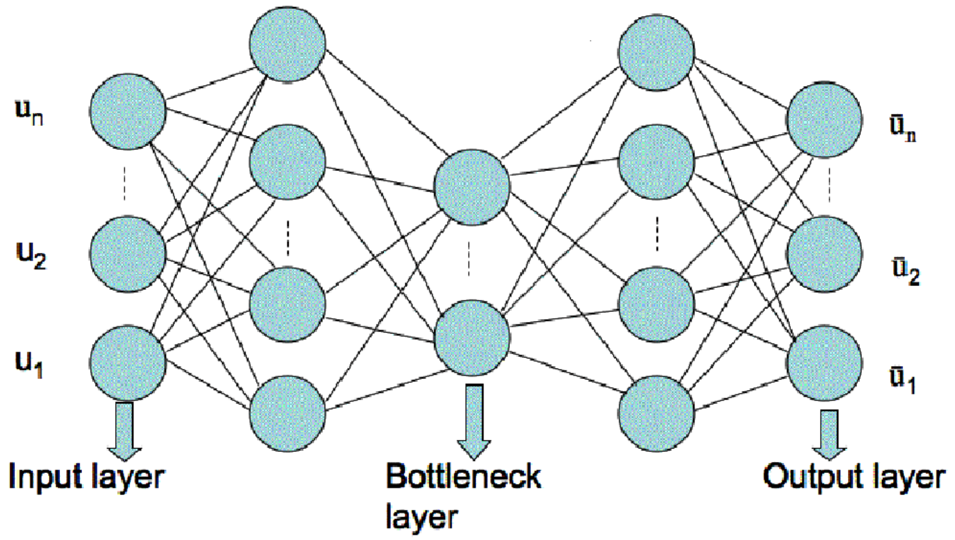


Figure 3.3: Neural network.

3.2 Data Partitioning

In this research we used the data observed on subject 1. A subject is defined as a person from whom the EEG data is observed. Each task has ten trials. Each trial consists of 2500 observations per session with each session lasting for ten seconds. 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 μ Volts) lasting up to 10 milliseconds was considered as an eye-blink. The 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.

For our training dataset we used trials 1 to 4 of tasks 3 (mental arithmetic) and 4 (letter-composing). Therefore our training dataset had 20000 observations. The test dataset comprised of trials 5 for tasks 3 and 4 and therefore, 5000 samples. The data from these two tasks was concatenated in both training and test datasets.

3.3 Data Lagging

EEG data used in this research was collected using 6 electrodes connected to the subject's scalp. Each electrode data output represented a dimension in the dataset making it 6-dimensional. This dataset was preprocessed using lagging.

Lagging was carried out by combining multiple samples together, thereby increasing the dimensionality of the neural network input dataset. For example, a lag of 1 meant that the dataset retained its dimensionality of 6. A lag of 2 implied that successive samples were combined, which means, row 1 of the dataset was combined with row 2 making row 1, 12 dimensional, original row 2 was combined with original row 3 making a new row 2 with 12 dimensions, so on and so forth. Thereby, a lag of 2 means that the dimensionality of the input dataset was changed from 6 to 12, a lag of 3 implies the new dimensionality of the dataset is 18, and so on. A lag of two also meant that the number of samples has reduced by 1, a lag of three means the number of samples has reduced by 2, and so on.

Lagging was done on both training as well as testing datasets. Therefore, lagging implied that although the data dimensionality was increased, the number of training and testing samples decreased, however, not at the same rate.

3.4 Bottleneck Algorithm Parameters

In this research we used a five layer neural network with three hidden layers and the middle one being the bottleneck. Using initial pilot experiments we held the hidden layers static with 30 hidden units. After this was chosen our main focus was to determine the number of bottleneck units. In each of our experiments, we varied the number of bottleneck units from 1 to 6 and also 10 and 20, holding the number of hidden layer units to be 30 on either side of our bottleneck layer. With these units we ran our final set of experiments using both SCG and modified LM algorithms. However, some pilot experiments were run with different combinations of hidden layer and bottleneck layer units as well. We reported the results for the above set of units in this research. The amount of data that was read for training was also varied using a variable which could be tuned. Although, pilot experiments were run using a smaller fraction of data, however, the results reported in this research are for the whole set of training and test data described in the previous section.

The default number of iterations for each training algorithm was set to be 1000, however, pilot experiments determined that 500 iterations yielded equally better results. All our results reported are for 500 iterations. Some of the parameters used in SCG algorithm were the same as reported in Nabney's netlab matlab library [Nab07] because our

SCG algorithm implementation was adapted from the matlab code of this library. The algorithm was terminated either at the end of the maximum iterations or if the second derivatives of the location in the weight search space would reach the machine precision, whichever happened sooner. The search direction was updated using Polak-Ribiere formula or we would restart in the direction of negative gradient after a fixed number of steps determined by pilot experiments. The initial search direction was chosen to be the negative of the gradient with lower bound on scale to be 1.0e-15 and the upper bound on scale as 1.0e100.

For modified LM algorithm, the maximum number of iterations was again chosen through pilot experiments and set to 200. There are not many parameters that required to be tuned for this algorithm. The algorithm was terminated either at the end of the maximum number of iterations or when the machine precision was reached. The initial weight multiplication factor was chosen to be 2 after testing some other values in the pilot experiments. All the parameters were chosen based on values reported by Wilamowski et al. If the RMSE increased in an epoch, the learning factor of the algorithm would be decreased by a tenth and if the error decreased then the learning factor would be multiplied by a factor of 10 to move faster in that direction of weight change. The Jacobian was still computed in each iteration, however, the inversion matrix had the dimension equal to the number of outputs rather than the number of weights.

3.5 Training and Testing Classification Algorithms

Some of the classification algorithms that we used in this research were the neural network, LDA, QDA and SVM. Classification using neural network was done by training it using SCG and modified LM algorithms followed by classification of the test data. The classification was done for both raw EEG input data and the reduced data. The reduced data was the output of the bottleneck layer obtained after first training the whole neural network, followed by removing the mapping network (the input layer, hidden layer and the bottleneck layer) and using the output of the bottleneck layer as input data for classification using the classification network. The classification network consisted of a network with only one hidden layer with 50 units. This could be represented as a network like n-p-1, where n is the bottleneck layer output dimensionality representing the classification network input dimesnionality, and p the hidden layer. There is only one output with target values 0 and 1 for math and letter writing tasks respectively. The reduced data for classification was obtained from NN with hidden layer units being 30 units and the number of bottleneck units being varied from 1-6,10, and 20. For each different value of the number of bottleneck units in the NN, the bottleneck output was used as an input to the three layer classification network and the results reported for all the input datasets.

LDA, QDA and SVM algorithms were also used for classification of EEG data and that of the reduced data. In each algorithm the training dataset is first learned and then the results reported for the test dataset for the classification into letter writing or the math tasks. Therefore, the training dataset helps in the feature selection and the test dataset is used for classification. In each algorithm the training and the test dataset is partitioned as explained in the data partitioning section. The classification was done using these algorithms for raw EEG data as well as the reduced data obtained from the bottleneck layer. These classification results are reported for reduced dataset obtained by training NN with the number of bottleneck layer units varied from 1-6, 10 and 20. The number of units in the other two hidden layers was set at 30 in the training neural network.

Chapter 4

Results

This chapter reports the results of various experiments carried out in this research. A number of experiments were carried out to determine the optimal number of bottleneck units depending upon the output error of the test data set. The 5 layer neural network was first trained using the fast convergence algorithm [WIKE01] followed by scaled conjugate gradient algorithm separately [Møl93]. Classification results reported were obtained using neural network, LDA, QDA and SVM.

4.1 Fast Convergence Algorithm

In order to determine the accuracy with which the data was being replicated at the output layer, we studied the RMSE of our EEG data set with a lag of 5. The number of units in the hidden layers was kept static at 30 and the number of units at the bottleneck layer was varied.

In our pilot experiments the test data RMSE was compared with the training data RMSE by varying the number of bottleneck units from 1-100. Our results indicated that the test RMSE for the number of bottleneck units between 10 and 20 was below the training RMSE. With the increase in the number of bottleneck units between 30 and 100, the test RMSE started showing up above the training RMSE. This observation encouraged us to use bottleneck units from 1 to 20. In all our subsequent experiments the bottleneck units have been varied from 1 to 20. Following these results we trained our neural network for 10 and 20 lags using fast convergence algorithm. Again, the number of bottleneck units was varied from 1 to 20. Fast convergence algorithm because of its hessian matrix could not handle data with more than 20 lags, so we restricted our results up to those lags. The RMSE for the reconstructed test data using fast convergence algorithm is shown in Figure 4.1

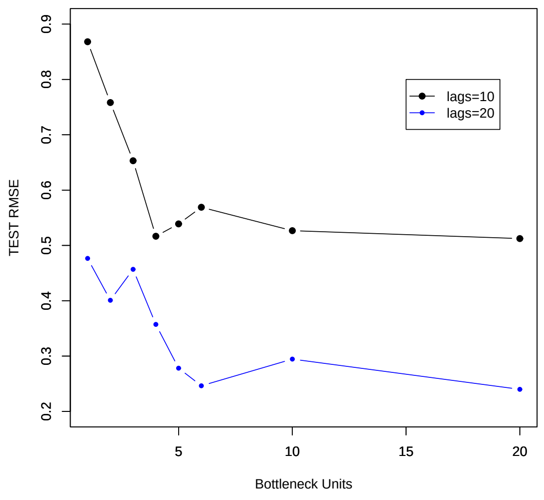


Figure 4.1: RMSE of reconstructed test data vs. the number of bottleneck units for different data lags.

Based on Figure 4.1, the least RMSE was obtained with 20 lags for a network containing 20 bottleneck units. This value is 0.239939. This indicates the best layer structure for our neural network would be a network with 30-20-30 as the hidden layers and a data lag of 20. It also indicated that higher the number of lags the lower the RMSE. It could also be noticed that the RMSE kept decreasing with the increase in the number of bottleneck units. It may be possible that we could achieve better results with more bottleneck units. However, it was not pursued in this research. We were also intrigued by the decrease in the RMSE with the increase in the number of lags, however, our training algorithm could not handle higher lags.

After the RMSE experiments, the test data set was classified using different algorithms and the results reported in Table 4.1. We used LDA, QDA and SVM for classifying the test data set. Neural network trained using fast convergence algorithm was also used for classification. The table also includes results for the classification of the bottleneck output. The neural network was first trained and then the first half of the neural network detached. The test data at the output of the bottleneck layer of the de-

nLags	nBottle	nOther	LDA Test	QDA Test	NN Test	Svm Test	R-NN Test	R-LDA Test	R-QDA Test	R-Svm Test
10			0.48	0.93	0.50	0.73				
10	1	30					0.50	0.51	0.51	0.50
10	2	30					0.50	0.51	0.51	0.49
10	3	30					0.50	0.51	0.51	0.51
10	4	30					0.48	0.51	0.51	0.51
10	5	30					0.50	0.51	0.51	0.49
10	6	30					0.50	0.51	0.51	0.54
10	10	30					0.50	0.51	0.51	0.54
10	20	30					0.50	0.51	0.51	0.57
20			0.50	0.97	0.51	0.76				
20	1	30					0.50	0.54	0.54	0.52
20	2	30					0.50	0.54	0.54	0.51
20	3	30					0.50	0.54	0.54	0.51
20	4	30					0.50	0.54	0.54	0.46
20	5	30					0.50	0.54	0.54	0.51
20	6	30					0.49	0.54	0.54	0.53
20	10	30					0.52	0.54	0.54	0.52
20	20	30					0.50	0.54	0.54	0.56

Table 4.1: Classification results of various methods. nlags: number of lags in the data, nBottle: number of units in the bottleneck layer of the neural network, nOther: number of units in the hidden layers, LDA Test: Classification results of using LDA for test data classification, QDA Test: Classification of test data using QDA, NN Test: Classification of test data using neural networks, R-NN Test: Classification of neural network bottleneck output using neural networks, Svm Test: Classification of Test data using Support Vector Machines.

tached network was used as an input for the classification algorithms for classifications. These results have been reported in Table 4.1 under the columns with prefix ’R’ before the algorithm name. The letter ’R’ represents the reduced neural network. The results indicate that QDA did the best for the raw test data. In this table the classification results in the NN test column represents the classification results for the neural network trained using the corresponding bottleneck network and not the classification network. The classification network has only one hidden layer with 50 units in it. A box plot comparison of these classification results is presented in Figure 4.2. Although, we observed the least RMSE in the bottleneck network with hidden layers 30-20-30, the best classification using neural network was obtained for the hidden layers, 30-6-30. The best data lag is 20. The reduced neural network classification results using neural network was obtained for the network 30-10-30, although there isn’t any significant difference between this result and the one obtained for original test dataset using 30-6-30 network. However, there is a significant difference between the original dataset classification results using QDA and other algorithms. For the original test dataset SVM was the second best with significantly better than LDA and the neural network. None of the algorithms produced encouraging results using the reduced neural network test output. There was no significant difference between the Reduced LDA (RLDA), Reduced QDA(RQDA) and Reduced SVM(RSVM) results, but they were still significantly better than Reduced NN(RNN) results, although the difference wasn’t huge.

In order to visualize the results of the RNN we present the results in Figure 4.3. This was done so that we have a clear picture of the neural network behavior. This figure shows the variation of classification results of the test dataset at the output of the bottleneck units. The classification network was trained using the fast convergence algorithm (modified LM). The graph indicates that a bottleneck layer with 10 units and a data lag of 20 was able to obtain the best classification accuracy of the test dataset. This

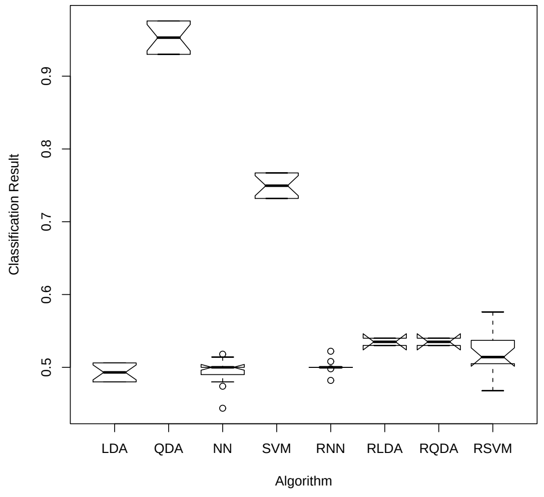


Figure 4.2: Classification results of different algorithms for the test dataset. NN represents the neural network classification of the actual test data and algorithms with a prefix R means the classification of the bottleneck output of the neural network. Neural network was trained using fast convergence algorithm(modified LM).

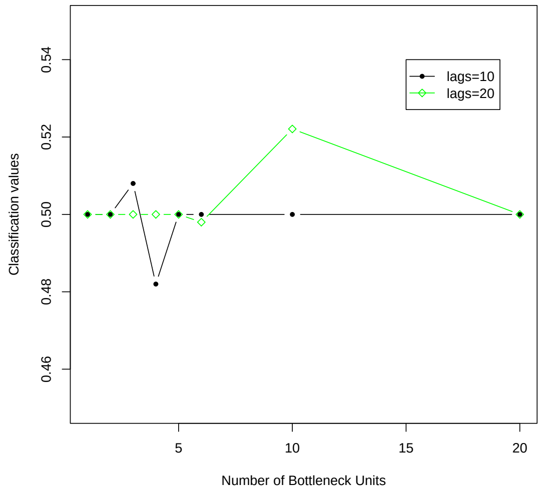


Figure 4.3: Classification results of the test data output at the bottleneck layer. Classification was done using Neural Network. The neural network was trained using fast convergence algorithm(modified LM).

indicates that a higher lag in the data might have yielded better classification accuracy of the test dataset, however, the training algorithm was unable to handle dataset with lags higher than 20. This indicates that with the increase in the number of bottleneck units, the additional features obtained using those units were making it difficult to differentiate between the two tasks.

4.2 Scaled Conjugate Gradient

We also trained our neural network using scaled conjugate gradient algorithm using various combinations of the number of units in the hidden layers. The RMSE of the reconstructed test data of the neural network trained using SCG is represented in Figure 4.4.

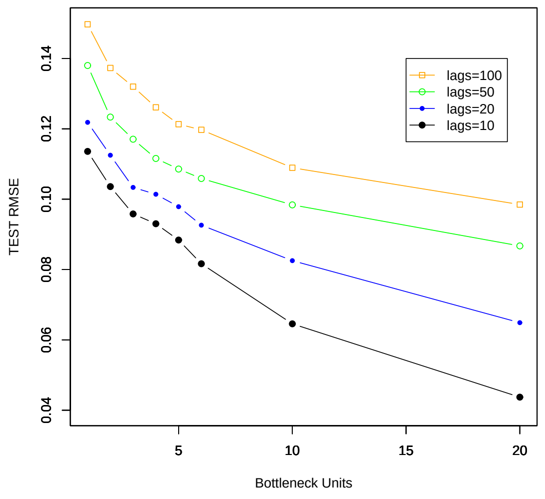


Figure 4.4: RMSE of reconstructed test data vs. the number of bottleneck units for different data lags.

These plots indicate that the least RMSE is for a data with lag 10 and 20 units in the bottleneck layer. Therefore, the best performing network has hidden layers with 30-20-30 units. This plot also shows that lower lag and higher number of bottleneck units helps in achieving a lower RMSE using SCG. It also leads us to think that a further reduction in the number of lags or an increase in the number of bottleneck units would yield a much lower RMSE. However, a further decrease in the number of lags would bring the gap, between the number of units in the bottleneck layer and the input data dimensionality, closer. Another, important observation is that the RMSE in this figure is lower than that in Figure 4.1. It might be possible that SCG is able to learn the weights between the hidden layers much better than the fast convergence algorithm (modified LM). It seems that although both algorithms learn the features equally well initially, SCG is able to fine tune the weight selection much better than the fast convergence algorithm (modified LM).

Since, SCG did better in terms of RMSE reduction than fast convergence algorithm (modified LM), we wanted to investigate how the data was being reconstructed at the output of our bottleneck neural network. Figure 4.5 shows the reconstructed data output compared to the input dataset for each channel. Different plots of this figure indicate that the reconstruction is fairly well except for channels 3 and 5. However, overall the bottleneck neural network output is able to mimic the input very well. This also indicates that the bottleneck layer is able to distinguish between the various channels input signals. In order to verify that we investigated the output of the bottleneck units of our bottleneck network. These outputs are shown in Figure 4.6. These figures represent the outputs at the bottleneck network, showing only a part of the dataset. In the left plot of Figure 4.6 we are trying to represent a quarter of the input samples to see the covariance of different channels at the bottleneck layer. In order to further visualize with a smaller sample size, the plot on the right in the same figure shows the bottleneck output for only 100 samples. These plots further illustrate that there is at the most either very little or no correlation between the bottleneck unit outputs. Overall, a no correlation output at the bottleneck layer indicates that the neural network is learning the dataset quite well. This would

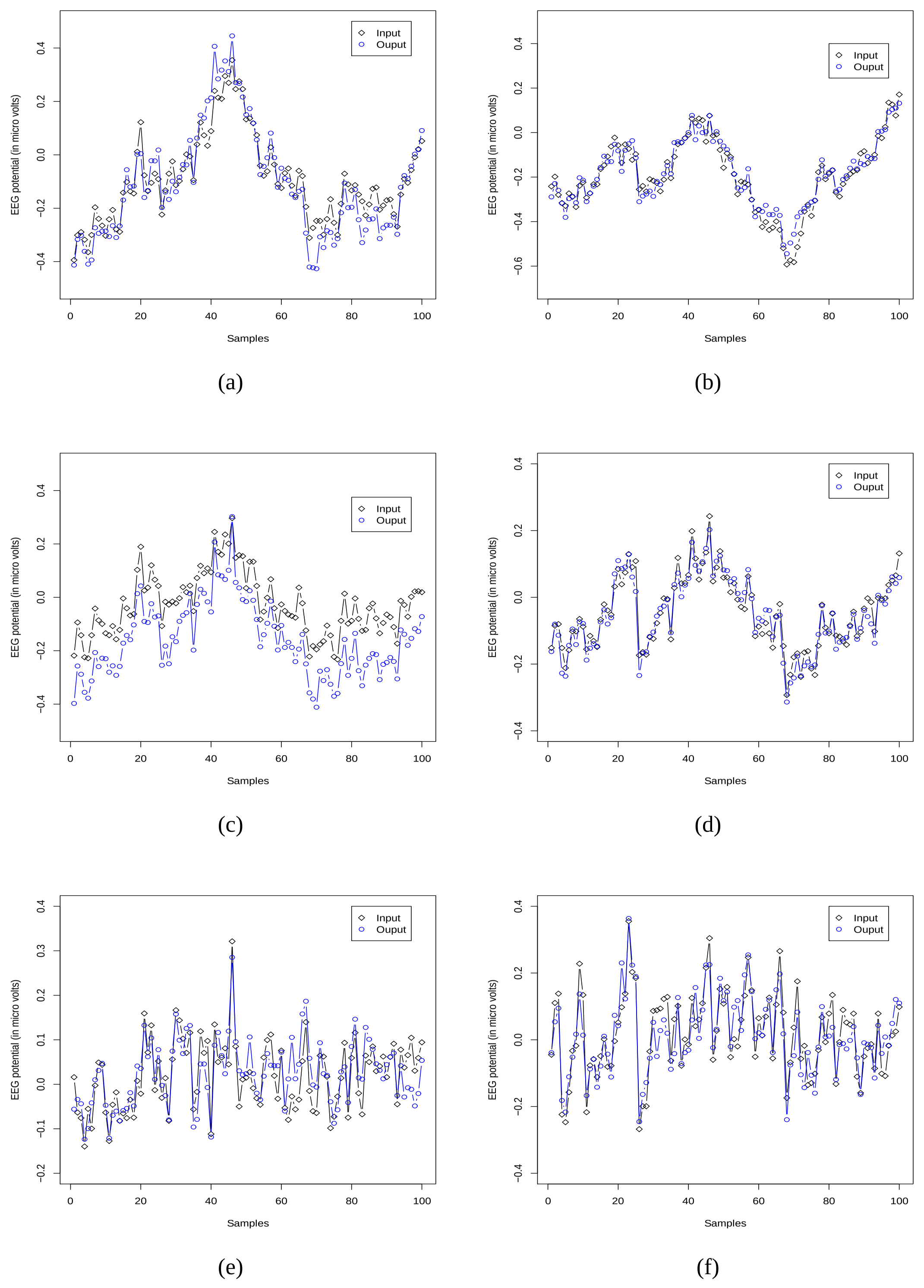
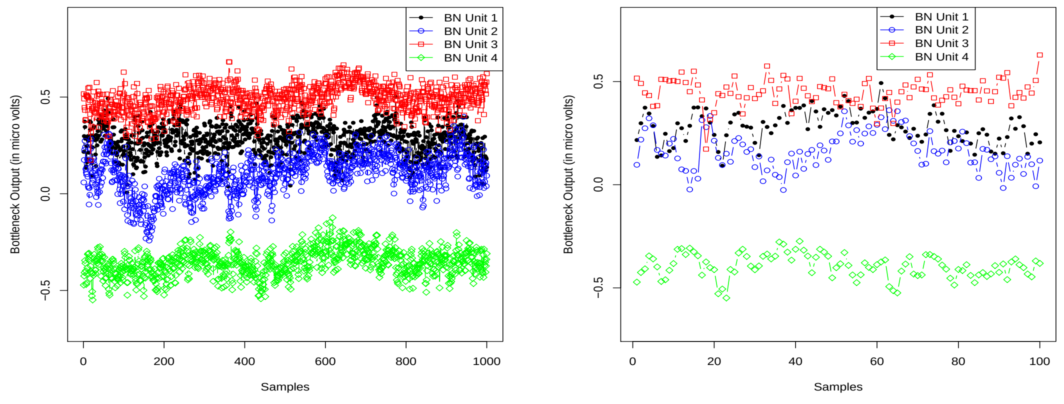


Figure 4.5: Each figure represents the reconstructed test data at the output of the bottleneck NN versus the input test dataset for an EEG channel. Figures a-f correspond to channels 1-6, respectively.



(a) (b)

Figure 4.6: The figure on the left represents bottleneck outputs with 1000 samples and the one on the right shows the same output for only 100 samples using 4 bottleneck units.

mean that classification accuracy of this output should be higher.

These classifications results of the bottleneck output have been presented in Figure 4.7. This plot indicates that reduction in RMSE observed in Figure 4.4 is directly proportional to the classification results. A neural network with 10 lags and 20 bottleneck units had the lowest RMSE and this network obtained the best classification results. These two figures indicate that SCG performed the best with a 30-20-30 network and data lag of 10.

We represented various combinations of lags, the number of bottleneck units and various classification results using neural networks and other algorithms. The results are represented in Table 4.2.

This table indicates that the neural network performed the best classification for a data lag of 20 and hidden layers with 30-6-30 units. This is not the same as was seen to have the least RMSE. LDA performed the best when data lag was 100 and so did QDA although there wasn't much difference in the results with the change in the number of data lags. SVM did the best with data lag of 20. In classification of bottleneck units

nLags	nBottle	nOther	LDA Test	QDA Test	NN Test	Svm Test	R-NN Test	R-LDA Test	R-QDA Test	R-SVM Test
10			0.46	0.93	0.92	0.87				
10	1	30					0.50	0.49	0.50	0.50
10	2	30					0.47	0.45	0.50	0.49
10	3	30					0.49	0.44	0.50	0.49
10	4	30					0.52	0.44	0.47	0.51
10	5	30					0.51	0.45	0.45	0.49
10	6	30					0.54	0.44	0.49	0.51
10	10	30					0.52	0.45	0.48	0.50
10	20	30					0.68	0.45	0.70	0.66
20			0.45	0.99	0.92	0.87				
20	1	30					0.49	0.50	0.51	0.50
20	2	30					0.50	0.50	0.51	0.50
20	3	30					0.47	0.46	0.50	0.49
20	4	30					0.52	0.43	0.48	0.52
20	5	30					0.51	0.44	0.46	0.50
20	6	30					0.52	0.45	0.46	0.49
20	10	30					0.51	0.46	0.47	0.53
20	20	30					0.53	0.46	0.52	0.54
50			0.46	0.99	0.74	0.86				
50	1	30					0.50	0.51	0.50	0.49
50	2	30					0.50	0.49	0.50	0.49
50	3	30					0.52	0.46	0.50	0.48
50	4	30					0.51	0.47	0.51	0.48
50	5	30					0.51	0.44	0.46	0.51
50	6	30					0.54	0.44	0.46	0.51
50	10	30					0.55	0.46	0.49	0.53
50	20	30					0.53	0.45	0.48	0.51
100			0.48	1.00	0.58	0.77				
100	1	30					0.48	0.47	0.48	0.47
100	2	30					0.46	0.48	0.54	0.48
100	3	30					0.46	0.44	0.52	0.47
100	4	30					0.50	0.43	0.54	0.46
100	5	30					0.54	0.44	0.54	0.45
100	6	30					0.53	0.41	0.51	0.52
100	10	30					0.50	0.46	0.51	0.48
100	20	30					0.50	0.46	0.51	0.52

Table 4.2: Classification results of various methods. nlags: number of lags in the data, nBottle: number of units in the bottleneck layer of the neural network, nOther: number of units in the hidden layers, LDA Test: Classification results of using LDA for test data classification, QDA Test: Classification of test data using QDA, NN Test: Classification of test data using neural networks, Svm Test: Classification of Test data using Support Vector Machines, R prefix to each algorithm: Classification of neural network bottleneck output using the corresponding algorithm

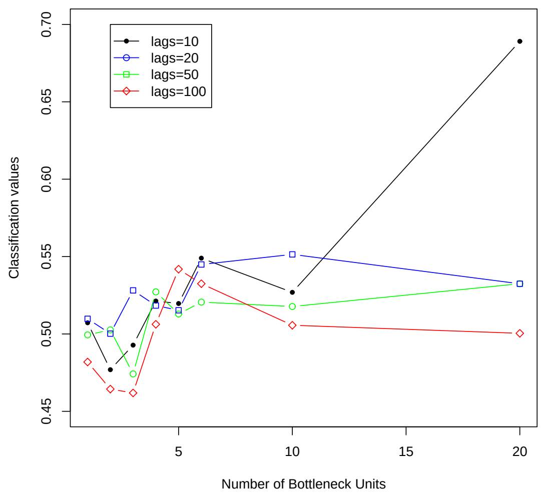


Figure 4.7: Classification results of the test data output at the bottleneck layer. Classification was done using Neural Network. The neural network was trained using Scaled Conjugate Gradient(SCG).

output, neural network with a data lag of 10 and 20 bottleneck units obtained the best classification and so did QDA and SVM. LDA obtained the best classification results on the bottleneck units output with a data lag of 50 and 10 bottleneck units. Although these results show some variation in the results based on the number of hidden layer units there still appears to be some uniformity. The higher the number of lags the better an algorithm performs on the data. Classification of bottleneck units output, however, indicated that a lower number of bottleneck units would obtain better results. Overall, QDA performed the best again with performance becoming better with the increase in the number of data lags. The column NN Test in this table has the classification accuracy values for the neural network trained using the corresponding bottleneck units. These bottleneck units do not correspond to the classification network, which has only one hidden layer with 50 units. In order to depict the variation of the classification results, Figure 4.8 represents the box plots of various algorithms used. These results clearly

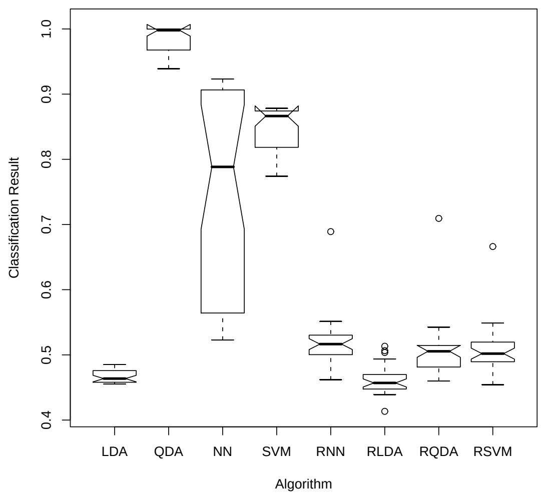


Figure 4.8: Classification results of different algorithms for the test dataset. NN represents the neural network classification of the actual test data and R prefix implies the classification of the bottleneck output of the neural network using the corresponding algorithm. Neural network was trained using Scaled Conjugate Gradient algorithm.

indicate that QDA outperformed the rest of the algorithms and was statistically significant. A neural network trained using SCG performed better than the one trained using fast convergence algorithm. There was no statistical significance between the results obtained using neural network and SVM. However, both of them performed better than the rest of the algorithms. Once again classification of bottleneck unit output was worse than that of the input data.

In order to have a comparison with Devulapalli's results we tried to train a neural network with the same number of lags and the hidden and bottleneck layer units as used by him. These results were run with a lag of 62 as used by Devulapalli. The number of bottleneck layer units was held constant at 30 as was used by Devulapalli. Also, the number of hidden layer units was varied to choose 40,50 and 60 units. These results are reported in Table 4.3. These results have been reported for a neural network trained using SCG algorithm because it proved to be having higher accuracy in classifying the

nLags	nBottle	nOther	LDA Test	QDA Test	NN Test	Svm Test	R-NN Test	R-LDA Test	R-QDA Test	R-Svm Test
62			0.47	1	0.58	0.83				
62	30	40					0.52	0.47	0.52	0.55
62	30	50					0.52	0.47	0.52	0.53
62	30	60					0.50	0.47	0.49	0.54

Table 4.3: Classification results of various methods using 62 lags and the neural network used by Devulapalli. nlags: number of lags in the data, nBottle: number of units in the bottleneck layer of the neural network, nOther: number of units in the hidden layers, LDA Test: Classification results of using LDA for test data classification, QDA Test: Classification of test data using QDA, NN Test: Classification of test data using neural networks, R-NN Test: Classification of neural network bottleneck output using neural networks, Svm Test: Classification of Test data using Support Vector Machines.

test dataset than the fast convergence algorithm (modified LM). However, the results indicate that the training algorithm was unable to achieve classification accuracy as good as Devulapalli obtained using his training approach. SVM and QDA performed significantly better again than any other approach whether it be ours or Devulapalli's. The neural network again did not provide any advantage for classification accuracy over the classification of raw EEG data. A poor result using LDA as well indicates that while there could be some non-linearity in the dataset, neural network trained using SCG could not learn those non-linear features as well as Devulapalli's approach did. A further tuning of our algorithm might help to improve our classification accuracy. However, all these results that have been presented above were obtained by running the experiments on subject 1 of our EEG dataset. It was observed that Devulapalli used subject 3. Therefore in order to exactly replicate Devulapalli's experiment setup we used subject 3 with his experiment parameters. The set up that was chosen includes; a training dataset with trials 2-10 of tasks 3 and 4, a test dataset of the first trial for both tasks, these datasets were lagged by 62, with 30 bottleneck layer units and 40 hidden layer units. The number of training epochs was chosen to be 200. For the three layer classification network the number of hidden units was chosen to be 60 with a sigmoid transfer function in the output layer. The number of training epochs was again chosen to be 200. In addition the target values were chosen to be -0.9 and 0.9 for the mental arithmetic and the letter writing tasks respectively. The classification results for this setup have been shown in Table 4.4.

nLags	nBottle	nOther	LDA Test	QDA Test	NN Test	Svm Test	R-NN Test	R-LDA Test	R-QDA Test	R-Svm Test
62			0.59	0.93	0.57	0.73				
62	30	40					0.50	0.56	0.80	0.76

Table 4.4: Classification results of various methods using 62 lags and the neural network used by Devulapalli for subject 3. nlags: number of lags in the data, nBottle: number of units in the bottleneck layer of the neural network, nOther: number of units in the hidden layers, LDA Test: Classification results of using LDA for test data classification, QDA Test: Classification of test data using QDA, NN Test: Classification of test data using neural networks, R-NN Test: Classification of neural network bottleneck output using neural networks, Svm Test: Classification of Test data using Support Vector Machines.

This table shows that classification accuracy for the reduced dataset improved significantly from 54%, the best obtained using SVM in table 4.2 for subject 1 using a setup similar to Devulapalli's, to the best of 80% for subject 3 using QDA. These results were obtained for the reduced dataset. We also saw an improvement in the classification accuracy using our approaches on subject 1. Table 4.2 shows that the best we had obtained using any of our methods for the reduced dataset was 70% classification accuracy using QDA. However, we still obtained better results on the raw EEG data using QDA.

The improvement in the accuracy results after dimensionality reduction indicates the effect of the bottleneck layer. It means that dimensionality reduction would play a significant role depending on the parameter set chosen. It also demonstrates that dataset also plays a very significant role in the classification results we obtain. It might be that subject 1 data are difficult to classify than subject 3 data.

Chapter 5

Conclusions

This research was based on the hypothesis that a bottleneck neural network would classify EEG data better than classification techniques like NN, QDA, LDA and SVM without the dimensionality reduction. However, the results indicate that a bottleneck net did not provide any advantage in classification, in our set up and our algorithm implementation. However, there was a marked improvement in the classification accuracy when the test subject was changed from 1 to 3. Overall, in almost all our experiment runs QDA performed the best followed by SVM for both raw as well as reduced EEG test datasets. Neural network performed either worst of all or slightly better than LDA.

These results show that for our setup of the neural network and our training algorithms, a bottleneck network would be the right choice for EEG data classification only after more parameter tuning. Amongst the training algorithms, SCG seems to be better both in terms of the computation time, the reconstruction RMSE and the classification. Figures 4.1 and 4.2 show a comparison of reconstruction RMSE using fast convergence and SCG training algorithms. Figures 4.1 and 4.8 show how SCG obtained better classification results than the neural network trained using fast convergence algorithm.

Since this work is built upon the previous research by Devulapalli [Dev96], it would be worthwhile here to compare our results with his. Devulapalli reported a classification accuracy of 86.22% for a reduced neural network with 30 bottleneck units and a window size of 62. This window size would correspond to a lag of 62 in our experiments.

Although we used a higher lag in our experiments, we did not increase the number of bottleneck units higher than 20 except while training a neural network similar to that used by Devulapalli. In general, our classification accuracy of test dataset using the reduced bottleneck network output data was far less than Devulapalli's results. We achieved a best of 70% classification accuracy using SCG training algorithm while it did not go beyond 52% using fast convergence algorithm (modified LM). For raw EEG data, Table 4.2 shows that we obtained a classification accuracy of 92% using 10 lags and 2 bottleneck units and also with data having 20 lags and, 3 and 6 bottleneck units. These results were also obtained by training a neural network using SCG for raw EEG data. In addition Table 4.3 presents our results for a data lag of 62. We used a neural network similar to the one used by Devulapalli with 30 bottleneck units, and trained it using SCG. However, our results were still worse than Devulapalli's results. We could achieve only a best of 58% classification accuracy using neural network while only 52% using a reduced neural network in comparison to Devulapalli's 86.22% classification accuracy for a similar reduced neural network. However, while these results were for subject 1, our classification accuracy using subject 3 and all the parameters exactly as Devulapalli's, achieved improved results. We got a classification accuracy of 80% for the reduced dataset using QDA against Devulapalli's 86.22% for a similar network. This indicates that SCG might need further parameter tuning to achieve better results than Devulapalli's results. Further experimentation on the number of bottleneck units in combination with the total number of input data lags in our training could improve our test data classification results. In addition, a larger neural network with more hidden layers could help learn the features of the dataset much better and hence achieve a better classification accuracy.

The fast convergence algorithm (modified LM) could not achieve classification accuracy results better than 51% for neural network with 6 and 10 bottleneck units and data lag of 20 for the raw EEG data. Our observations indicate that a data lag of 20 proved to be a good choice. However, neither our algorithms nor Devulapalli's approach was able to beat the classification accuracy results obtained by QDA. QDA was able to achieve far better test data classification accuracy and always achieved results of more than 90% with the best being 100% when there was a lag of 100 in the input data. SVM also did better than the rest of the approaches except QDA.

Our hypothesis was based on the intuition that a bottleneck network would be able to retrieve the features of the dataset much better than any other algorithm which proved to be difficult. On the other hand QDA was much more efficient in successfully differentiating between the data features and be able to classify the data much better than any other method. In training neural network with a further variation in the number of bottleneck units or the number of hidden layers we might be able to obtain results which are competitive with those of QDA. It also seems that we might have missed some of the aspects of the approach used by Devulapalli, which might explain some of the reasons why his neural network was able to achieve better results than the neural network trained using our methods. One of the reasons we found was the dataset. Subject 3 dataset was classified with higher accuracy than subject 1. Although, this indicates that the dimensionality reduction could be effective for classification accuracy, it also raises some questions. One of the important observations is that classification accuracy is data dependent and sensitive to parameters in this set up. Further experimentation will be required to reach a good conclusion as to how these parameters really affect the classification. Subject 1 seemed to be a very tough dataset, while our neural network was able to learn subject 3 features very well. This could mean there are differences between subjects 1 and 3. However, one other aspect is the parameters. We used a sigmoid transfer function in the output layer for subject 3 classification, and a linear transformation for subject 1. It is possible that, it could be the reason. We might need to test our methods on many more subjects to really draw a more meaningful conclusion regarding the factors that affect our classification accuracy. Some factors could be extraneous like the subjects state of mind while the EEG is being recorded.

In our approaches, the inability of the neural network to provide more advantage in classification indicates that the weights of the neural network might not have been learned efficiently. It appears that our weight search space has a very high concentration of local minima which make it very difficult to find a global minimum of our objective function of residual error minimization. As a result of this we are unable to find a good set of weights. This inability influences the dimensionality reduction that we are achieving. It appears that finding a good set of weights might help design a better performing neural network. In a recent paper by Hinton et al. [HS06] a pretraining approach was used to choose an initial set of neural network weights and training algorithms were used to fine tune these weights for an efficient dimensionality reduction. This indicates that it could be possible that if we start with a good set of initial weights our algorithms might be able to fine tune those weights easily. This could enable us to take advantage of the dimensionality reduction using neural network such that we learn the dataset features and achieve a higher classification accuracy. However, this would be a part of future research.

Overall, we can still say the dimensionality reduction for classification seems to be a very promising technique as indicated by ours and Devulapalli's results. Further investigation needs to be carried out to fully exploit the potential of dimensionality reduction using neural networks for a higher classification accuracy.

REFERENCES

[ADS95] C. Anderson, S. Devulapalli, and E. Stolz. Eeg signal classification with different signal representations. In F. Girosi, J. Makhoul, E. Manolakos, and E. Wilson , editors, *Neural Networks for Signal Processing V*, pages 475–483. IEEE Service Center, Piscataway, NJ, 1995.

[AP01] C. Anderson and D. Peterson. Recent advances in eeg signal analysis and classification. In R. Dybowski and V. Gant, editors, *Clinical Applications of Artificial Neural Networks*, pages 175–191. Cambridge University Press, UK, 2001.

[ASS98] C. Anderson, E. Stolz, and S. Shamsunder. Multivariate autoregressive models for classification of spontaneous electroencephalogram during mental tasks. *IEEE Transactions on Biomedical Engineering*, 45(3):277–286, 1998.

[bci06] Guest editorial the third international meeting on brain-computer interface technology: Making a difference. In *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, volume 14, pages 126–127, 2006.

[DC93] David DeMers and Garrison Cottrell. Non-linear dimensionality reduction. In Stephen José Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 580–587. Morgan Kaufmann, San Mateo, CA, 1993.

[Dev96] Saikumar Devulapalli. Non-linear principal component analysis and classification of eeg during mental tasks. Master's thesis, Colorado State University, 1996.

[GMW81] Philip E. Gill, Walter Murray, and Margaret H. Wright. *Practical Optimization*. Academic Press Inc., 111 Fifth Avenue New York, NY, USA, 1981.

[GMW04] Philip E. Gill, Walter Murray, and Margaret H. Wright. Methods for non-linear least squares problems. Technical University of Denmark, 2004.

[Hot53] Harold Hotelling. New light on the correlation coefficient and its transform. *Journal of The Royal Statistical Society*, 15:193–232, 1953.

[HS06] Geoff Hinton and Ruslan Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313:504–507, 2006.

[HTF01] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2001.

[Jas58] H.H. Jasper. The ten-twenty electrode system of the international federation. *Electroencephalography and Cinical Neurophysiology*, 10:371–373, 1958.

[KA90] Zachary A. Keirn and Jorge I. Aunon. A new mode of communication between man and his surroundings. *IEEE Transactions on Biomedical Engineering*, 37(12):1209–1214, December 1990.

[Kar47] K Karhunen. Ueber lineare methoden in der wahrscheinlichtskeitsrechnung. *Annals Acad. Sci. Fennicae Series*, 37, 1947.

[KM96] Michael J. Kirby and Rick Miranda. Circular nodes in neural networks. *Neural Computation*, 8(2):390–402, 1996.

[Kni03] James N. Knight. Signal fraction analysis and artifact removal in eeg. Master's thesis, Colorado State University, Fort Collins, CO-80523, December 2003.

[Kra91] Mark A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37:233–243, 1991.

[LBOM98] Yann LeCun, Lean Bottou, Genevieve B. Orr, and Klaus-Robert Muller. Efficient backprop. In Genevieve B. Orr and Klaus-Robert Muller, editors, *Neural Networks: Tricks of the trade*. Springer, 1998.

[Lev44] K. Leveberg. A method for the solution of certain problems in least squares. *Quart. Appl. Math.*, 2:164–168, 1944.

[Loe48] M. M. Loeve. Fonctions aleatories de seconde ordre. In *Process Stochastiques et Movement Brownien*, Hermann, Paris, 1948.

[Mar63] D. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM J. Appl. Math.*, 11:431–441, 1963.

[Møl93] Martin F. Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6:525–533, 1993.

[Nab07] Ian Nabney. Netlab neural network software. http://www.ncrg.aston.ac.uk/netlab/, 2007.

[Oja91] Erkki Oja. Data compression, feature extraction, and autoassociation in feedforward neural networks. In *Artificial Neural Networks*, pages 737–745, 1991.

[PKK+05] D. Peterson, J. Knight, M. Kirby, C. Anderson, and M. Thaut. Feature selection and blind source separation in an eeg-based brain-computer interface. *EURASIP Journal on Applied Signal Processing*, 19:3128–3140, 2005.

[UNN91] Shiro Usui, Shigeki Nakauchi, and Masae Nakano. Internal color representation acquired by a five-layer neural network. In *Artificial Neural Networks*, pages 867–872, 1991.

[Vap95] Vladimir N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[WIKE01] Bogdan M. Wilamowski, Serdar Iplikci, Okyay Kaynak, and M. Onder Efe. An algorithm for fast convergence in training neural networks. In *International Joint Conference on Neural Networks*, volume 3, pages 1778–1782, 2001.