THESIS

P300 CLASSIFICATION USING DEEP BELIEF NETS

Submitted by

Amin Sobhani

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Summer 2014

Master's Committee:

Advisor: Charles Anderson

Asa Ben-Hur
Chris Peterson

ABSTRACT

P300 CLASSIFICATION USING DEEP BELIEF NETS

Electroencephalogram (EEG) is measure of the electrical activity of the brain. One of the most important EEG paradigm that has been explored in BCI systems is the P300 signal. The P300 wave is an endogenous event-related-potential which can be captured during the process of decision making as a subject reacts to a stimulus. One way to detect the P300 signal is to show a subject two types of visual stimuli occurring at different rates. The event occurring less frequently than the other elicits a positive signal component with a latency of roughly 250-500 ms. P300 detection has many applications in the BCI field. One of the most common applications of P300 detection is the P300 speller which enables users to type letters on the screen.

Machine Learning algorithms play a crucial role in designing a BCI system. One important purpose of using the machine learning algorithms in BCI systems is the classification of EEG signals. In order to translate EEG signals to a control signal, BCI systems should first capture the pattern of EEG signals and discriminate them into different command categories. This is usually done using different machine learning-based classifiers. In the past, different linear and nonlinear methods have been used to discriminate the P300 signals from nonP300 signals. This thesis provides the first attempt to implement and examine the performance of the Deep Belief Networks (DBN) to model the P300 data for classification. The highest classification accuracy we achieved with DBN is 97 percent for testing trials. In our experiments, we used EEG data collected by the BCI lab at Colorado State University on both healthy and disabled subjects.

ACKNOWLEDGEMENTS

I would like to express my appreciation to my advisor Professor Dr. Charles Anderson for his reviews, guidance and thoughtful comments to improve this work. I would like also to thank Dr. Asa Ben-Hur and Dr. Chris Peterson for serving on my research committee.

TABLE OF CONTENTS

CHAPTER 1

Introduction

1.1. EEG

Electroencephalogram (EEG) is measure of the electrical activity of the brain. Many practical imaging techniques like nuclear imaging and MRI have been developed over the recent years. However, neurologists still use EEG as a complementary tool for assessment of neurological disorders since EEG records electrical activity of the brain over a period of the time while other imaging techniques do not provide such advantage. EEG has a variety of applications in neurology including diagnosing sleep disorders, infections, tumors [17].

EEG can be recorded using EEG electrodes. EEG electrodes measure the electric potential difference of the electric field created by brain neural activities, at the scalp. This can be done in two ways. The first way can be performed by choosing an arbitrary referencing electric potential on the head. This is considered as a zero level potential. Having chosen this referencing point, electric voltage of any interest points on the scalp will be calculated by subtracting the recording electrode from the reference electrode. The resulting value shows electrical potential difference of the interest point on the scalp. In the case of multichannel EEG recording, rather than choosing a referencing potential electrode, we can average among different recording electrodes and subtract this average from each electrode [17].

Researchers use standardized locations for EEG electrodes. This facilitates comparison of different experiment results and studies. Exact locations are different from subject to subject. Figure 1.1 shows the eight standardized locations we used to collect the EEG signals.

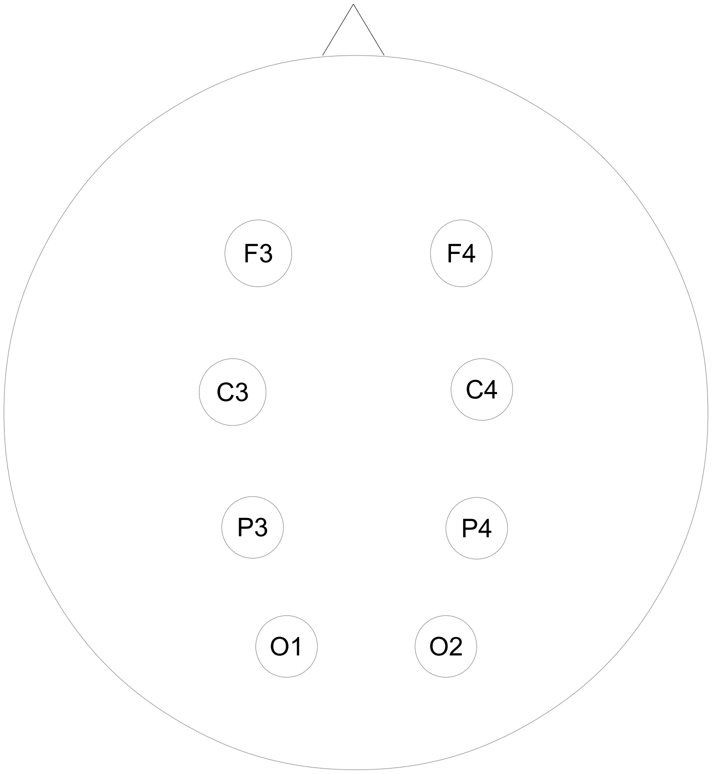

Figure 1.1. Standardized locations used to collect the EEG signals

.

Physicians are not the only ones who use the EEG to diagnose different neurological disorders. Brain Computer Interface (BCI) scientists use the EEG as a way of communication between digital devices and the brain. The EEG is not the only brain function measurement technology. However, researchers widely believe that the EEG is the best technology among all of them. By using the EEG as an alternative way of communication, the BCI provides a significant impact on the assistive care. For every activity, feeling or any decisions we make, there are associated neural activities in our brain. Different neural activities result in different EEG signals. Discriminating these different EEG signals provides unique commands from the brain to the external world. For example, someday a completely paralyzed patient may be able to control robotic arm movement by just thinking about it. BCI applications are not limited to assistive care. BCI systems can also be applied to control a computer application such as word processor or Graphic User Interface. Nowadays, gaming industries use BCI technologies. However, it's in an early stage [11].

1.2. EEG Paradigm

One of the most important EEG paradigms that has been explored in the BCI systems is the P300 signal. The P300 wave is an endogenous event-related-potential which can be captured during the process of decision making as a subject reacts to a stimulus. A usual way to detect the P300 signal is to show a subject two types of the events occurring at different rates. The event occurring less frequently than the other elicits a positive signal component with a latency of roughly 250-500 ms.

The BCI lab at Colorado State University detected and recorded the elicitation of the P300 signals by asking subjects to look at a computer screen in which different letters randomly appeared in the middle. Then the subjects were asked to count how many times a specific letter (target letter) was shown in the middle of the screen. The Figure 1.2 shows P300 and nonP300 signals recorded by the BCI lab at Colorado State University.

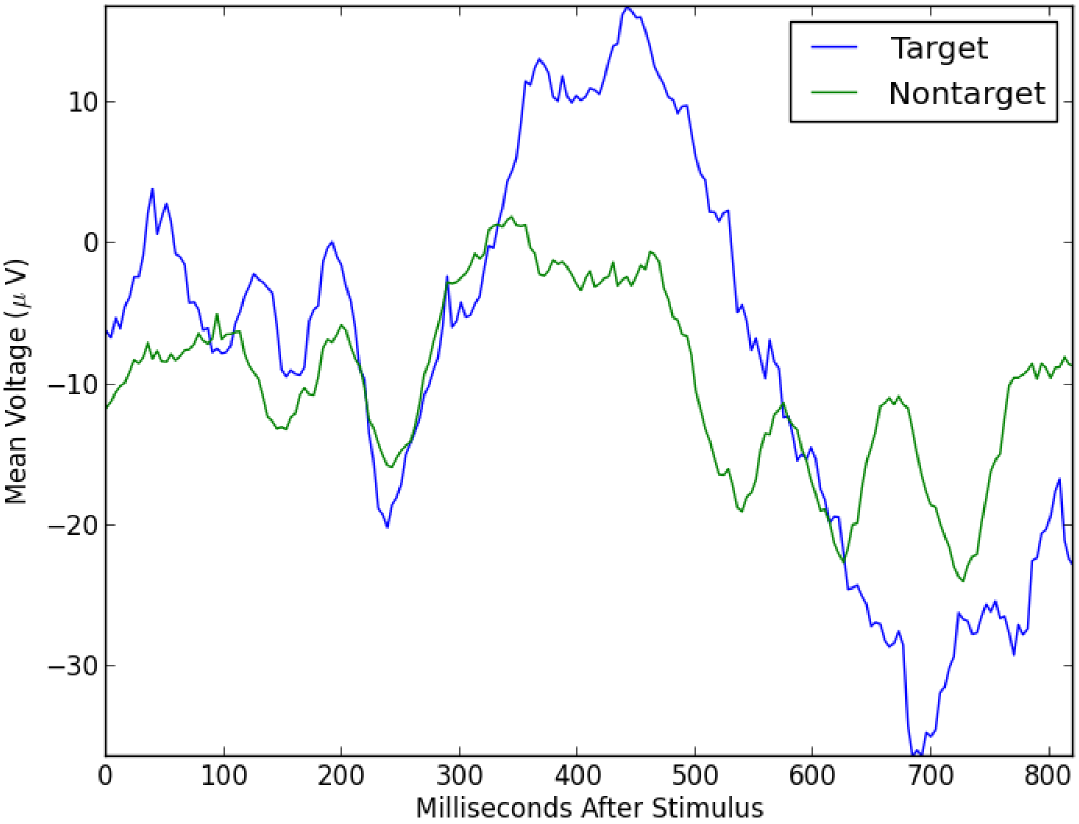

FIGURE 1.2. The P300 signal

P300 detection has many applications in the BCI field. One of the most common applications of P300 detection is the P300 speller which enables users to type letters on the screen [5]. In the mid 1970s, P300 detection was also used for lie detection.

Machine Learning algorithms play a crucial role in designing a BCI system for a variety of reasons. First of all, the machine learning algorithms enable a BCI device to extract useful features for discrimination. The EEG recorded at the scalp is usually noisy for a variety of reasons. The feature extraction module makes it easier for the classifier to discriminate the EEG signals into its corresponding brain activity pattern. In fact the feature extraction module tends to remove the noise, artifacts and other unnecessary features of the raw data. The second purpose of using the machine learning algorithm in BCI systems is the classification of EEG signals. In order to translate EEG signals to a control signal, BCI systems should first capture the pattern of EEG signals and discriminate them into different command categories. This is usually done using different machine learning-based classifiers [3].

1.3. Related Work

In the past, different linear and nonlinear methods have been used to discriminate the P300 signals from nonP300 signals. In this section, we briefly describe some of the popular classifiers which have been used for the P300 classification. We do not provide classification accuracy here since the classification accuracy that each method can achieve changes from one data set to another data set.

Fisher's Linear Discriminant Analysis (FLDA) [12] is one of the widely used methods for P300 classification [15, 19, 20]. Many experimental results suggest that FLDA works better than the other classifiers. The main difference between FLDA and Linear Discriminant Analysis is that FLDA does not assume that the data is normally distributed or that the classes have equal covariance.

Stepwise Linear Discriminant Analysis (SLDA) [18] is an extension of Fisher's Linear Discriminant Analysis which incorporates filter feature selection. This method has been widely used for P300 classification and there are demonstrations that in many cases it works better than several other classifiers such as Linear and Nonlinear Support Vector Machines, Bayesian Linear Discriminant Analysis [21, 20]. SLDA fits the model to the training data set by adding and removing the terms from linear discriminant analysis based on their significance on the discrimination [20].

Linear Support Vector Machines (LSVM) [24] find the hyperplane that maximizes the margins of the two classes in the case of the binary classification. In the past, many attempts have been made to classify P300 signals using LSVM [20, 21]. Since sometimes the separating boundary between two classes is not linear, most of the researchers have used a regularizer term to increase the power of this classifier. It has been reported in the literature that LSVM works very well for P300 detection [20, 18].

Although researchers have examined the performance of different linear and nonlinear methods, still no one has attempted to examine the performance of deep learning algorithms, such as Deep Belief Networks as a P300 classifier. This thesis provides the first attempt to implement and examine the performance of the Deep Belief Networks to model the P300 data for classification. We will show that this sophisticated nonlinear technique will deliver a superior classifier for P300 classification.

1.4. Contributions

This work contributes to the Brain Computer Interface field since it provides the first attempt to model the P300 data for classification using the Deep Belief Networks. In this thesis, we will show that the Deep Belief Networks works very well as a P300 classifier. As mentioned in the related work section, most of the previous experiments show that linear classifiers work better than nonlinear classifiers and many researchers concluded that using nonlinear classifiers may result in over fitting the training data sets and will result in a poor generalization. In this thesis, we will show that a nonlinear classifier such as deep belief networks could work even better than the linear classifiers. The highest accuracy we have achieved for classifying the P300 data is 97 % for one subject, though the average across 13 subjects was 76.4 %. In the following chapters, we will thoroughly discuss why Deep Belief Networks works well. However, in brief, we can point to two unique features of the DBN. The first feature is unsupervised pre-training using the Restricted Boltzmann Machine. The second feature is using a deep architecture which leads to a lower number of the learning modules, and as a result improved generalization.

1.5. Overview

In Chapter 2, we first motivate using deep architectures and then describe the deep belief networks algorithm proposed by G. Hinton. We also provide our implementation method in python. In Chapter 3, we discuss and provide the results we have achieved for P300 classification using our implementation of deep belief networks. In this chapter, we also discuss the parameter values and visualizing the higher layer units of the deep belief networks. Finally, in Chapter 4, we provide some possible paths for future work and provide a conclusion and a brief summary of what we did in this thesis.

CHAPTER 2

Background and Methods

2.1. Deep Architecture Motivation and Challenges

In this section, we talk about the advantages of using deep architectures for modeling the data and the challenges we face in training the deep architectures. The depth of an architecture is defined as the longest path between any inputs and any outputs of the system. For example, in neural networks, the depth of the architecture is the number of the hidden layers plus one. Theoretical results suggest that some complex functions may not be represented efficiently by shallow architectures [6]. This is due to the fact that deep architectures can provide a more compact representation of a function compared with shallow architectures. By compact representation here we mean that a few computational elements need to be tuned in order to train the system. This will improve the generalization especially if the number of the training data samples is not large.

On the other hand, training deep architectures is more challenging than training shallow architectures [6, 7, 10]. Experimental results show that training deep neural networks with random initialization of the weights and biases can become stuck in local minima [6, 7, 10]. However, unsupervised pre-training of each layer can improve the training results for deep neural networks [6, 7, 10]. This unsupervised training phase is done in a greedy manner (we start by pre-training of the first layer and we continue this layer by layer unsupervised pre-training toward the last layer). Having finished unsupervised pre-training, the training is followed by a supervised training with parameters initialized to the values optimized from the pre-training phase.

One reason that this unsupervised pre-training improves the training result is that it directs the parameters of each layer toward a better region for initialization in the supervised training phase [6]. On the other hand, when in neural networks the number of hidden layers increases, the top layers will fit the data set and as a result the lower layers will be trained poorly [10, 16]. This will result in poor generalization. One advantage of deploying unsupervised pre-training is that it improves the training of the lower layers and hence will result in better generalization [10]. In the next sections, we summarize the deep learning architecture and training algorithm proposed by Hinton.

2.2. Energy Based Models

This section has been summarized from the textbook of “Predicting Structured Data”[4]. Machine Learning methods aim to encode the dependencies between the independent variables and the target variables. This can be seen as learning a complex function which approximates the value of the target variables given the value of the independent variables. Different methods perform this in different ways. Energy Based Models (EBM) associate different scalar energy to different configuration of variables. The learning process in energy based models will lead to finding optimal parameter values (i.e., weights and biases) which associate the minimum energy to the correct value of the target variables and higher energy to the incorrect value of target variables. This is the only constraint we have in training energy based models and as a result these models are very flexible for training. Training can be modeled by searching through the set of energy functions indexed by weights, W, as the parameters of the system. This can be mathematically expressed as

$$\{E(X,Y,W)|W \in w\}. \tag{1}$$

In Equation (1), X is the vector of observed values, Y is the target vector. After training, the resulting value of W is used to infer the target value Y^* as

$$Y^* = \underset{Y \in y}{\arg\min}\, E(X, Y, W). \tag{2}$$

In Equation (2), y is the set of the all possible values that can be assigned to the target variable Y.

The flexibility that EBMs offer us makes it hard to combine separately built energy based models. In order to be able to use the output of energy based models as input to another separately built EBM, we use the Gibbs distribution to provide the conditional distribution of the output given input $(P(Y|X))$

$$P(Y|X) = \frac{e^{-E(X,Y,W)}}{\int_{Y \in y} e^{-E(X,Y,W)}}. \tag{3}$$

In Equation (3), the numerator is a continuous function. So the existence of the conditional probability distribution is dependent on the convergence of the integral function in the denominator of the Equation (3). In order to train the EBMs as probabilistic models, negative log likelihood loss function is the best possible choice for loss function. In fact, probabilistic models are a kind of energy based models which are trained by the negative log likelihood as loss function. In order to explain the negative log likelihood, first we define the training data set consisting of the K samples formally as

$$S = \{(X^i, Y^i), i \in N \text{ and } 0 < i \leq K\}. \tag{4}$$

Having given the training data set as defined in Equation (4) and conditional probability distribution as given in Equation (3), our goal is to increase the conditional probability of the target variables given the independent variables by changing the parameter values (i.e., weights and biases). To express this formally, we have

$$P(Y^1, Y^2 ..., Y^i | X^1, X^2 ..., X^i, W) = \prod_{i=1}^{K} P(Y^i | X^i, W). \tag{5}$$

In Equation (5), we have assumed that the samples in the training data sets are independent from each other, i.e., $P(Y^i | X^1, X^2, X^3, ..., X^i) = P(Y^i | X^i)$. By taking the negative log from two sides of the Equation (5) and applying Equation (3), we will derive

$$L(W, S) = -\log \prod_{i=1}^{K} P(Y^i | X^i, W) = \sum_{i=1}^{K} (E(X^i, Y^i, W) + \log \int_{Y \in y} e^{-E(X^i, Y, W)}). \tag{6}$$

In Equation (6), K is the number of the training data samples. In Equation (6), the derivative of the integral term (second term), can be calculated as

$$\frac{\partial \log \int_{Y \in y} e^{-E(X^i, Y, W)}}{\partial W} = \int \frac{\frac{\partial \int_{Y \in y} e^{E(X^i, Y, W)}}{\partial W}}{\int_{Y \in y} e^{-E(X^i, Y, W)}} = -\int_{Y \in y} \frac{e^{-E(X^i, Y, W)} \frac{\partial E(X^i, Y, W)}{\partial W}}{\int_{Y \in y} e^{-E(X^i, Y, W)}}. \tag{7}$$

Having considered the Equation (7) and (3), we can calculate the derivative of Equation (6) for a single sample i as

$$\frac{\partial L(W, X^i, Y^i)}{\partial W} = \frac{\partial E(X^i, Y^i, W)}{\partial W} - \int_{Y \in y} \frac{\partial E(X^i, Y, W)}{\partial W} P(Y | X^i, W). \tag{8}$$

In the above equation, approximating the second term, called the contrastive term, is not always easy. In fact, for many models it is very difficult and sometimes impossible to approximate or calculate the integral term in Equation (8) since the conditional probability and the energy function should have a very specific distribution in order to approximate the integral. Here we introduce the contrastive divergence method proposed by Hinton to approximate the contrastive term in Equation (8) for the Restricted Boltzmann Machine which is an undirected graphical model. In the Restricted Boltzmann Machine, the second term in Equation (8) is approximated using a Gibbs sampling method with the chain starting from the data. We will discuss this in the next session. Using contrastive divergence, Equation (8) can be approximated as

$$\frac{\partial L(W, X^i, Y^i)}{\partial W} = \frac{\partial E(X^i, Y^i, W)}{\partial W} - \frac{\partial E(X^i, \tilde{Y}, W)}{\partial W}. \tag{9}$$

In Equation (9), $\tilde{Y}$ is the approximation of the desired target value. In the next section, we will explain how to approximate $\tilde{Y}$ by Gibbs sampling in order to train the Restricted Boltzmann Machine. Using the gradient descent algorithm, the weight update rule would be

$$W = W - \eta(\frac{\partial E(X^i, Y^i, W)}{\partial W} - \frac{\partial E(X^i, \tilde{Y}, W)}{\partial W}). \tag{10}$$

In Equation (10), the first term will cause the energy surface to be pushed down at the desired target variable and the second term will cause the energy surface to be pulled up around the target variable. By doing this, we hope the desired target will eventually become the local minima. Stochastic gradient descent is often suggested for optimization of the loss function for high dimensional data [4, 8], so in this thesis we have used stochastic gradient descent to optimize the loss function.

2.3. Restricted Boltzmann Machine

The Restricted Boltzmann Machine (RBM) is a type of graphical undirected energy based model. RBMs are the building block of the Deep Belief Networks [6]. Figure 2.1 shows the architecture of the restricted boltzmann machine:

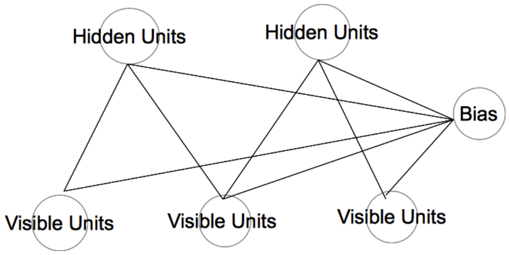

Figure 2.1. Restricted Boltzmann Machine

As you can see in the Figure 2.1, the hidden units are not connected to each other, nor the visible units. As we show here, this makes the energy function simpler and makes the inference easy. In this thesis, we have used two kinds of RBMs, Bernoulli-Bernoulli RBM and Gaussian-Bernoulli RBM. In the next sections, we briefly describe them. Please note that in this thesis, we have implemented the algorithm proposed by Hinton [13, 6].

2.4. Bernoulli-Bernoulli RBM

Bernoulli-Bernoulli RBM is a kind of RBM in which hidden units and visible units have stochastic binary values [13]. Energy of the Bernoulli-Bernoulli RBM can be calculated as [6]

$$E(x, h) = -b'x - c'h - h'Wx. \tag{11}$$

In (11) b is the bias vector for visible units, c is the bias vector for hidden units, x is the vector of visible units values, h is the vector of hidden units values and W is the weight matrix in which w_{ij} shows the edge value between Visible Unit i and Hidden Unit j. b', c' and h' are column vectors. Please note that b' is the transpose of vector b, c' is the transpose of vector c and h' is the transpose of vector h.

Considering Gibbs sampling and Equation (11), we would have [6]

$$\begin{aligned} P(h|x) &= \frac{e^{b'x+c'h+h'Wx}}{\sum_{\tilde{h}} e^{b'x+c'\tilde{h}+\tilde{h}'Wx}} \\ &= \prod_i \frac{e^{c_i h_i + h_i W_i x}}{\sum_{\tilde{h_i}} e^{c_i \tilde{h_i} + \tilde{h_i} W_i x}}. \end{aligned} \tag{12}$$

In Equation (12), $\tilde{h}$ is the set of all possible values can be assigned to the hidden vector h, h_i is the ith hidden unit, c_i is the bias unit for the ith hidden unit and W_i is the ith row of the weight matrix. The sigmoid function can be calculated as $sig(x) = \frac{1}{1+e^{-x}}$. Considering Equation (12) and assuming hidden units are Bernoulli, for Hidden Unit i we would have

$$P(h_i = 1|x) = sig(c_i + W_i x). \tag{13}$$

Equation (13) shows how to infer each hidden unit value given the visible units values, x. W_i is the ith row of the weight matrix. Assuming each visible unit is Bernoulli, in the similar way to Equation (13), for Visible Unit j we can derive

$$P(x_j = 1|h) = sig(b_j + W_j' h). \tag{14}$$

Equation (14) shows how to infer each visible unit value given hidden units vector. W_j is the jth row of the W. Using Gibbs distribution we can derive [13]

$$P(v) = \frac{1}{Z} \sum_h e^{-E(v,h,W)}. \tag{15}$$

Equation (15) shows the probability that RBM assigns to visible vector v. By taking derivative of log probability of the training vector v and using contrastive divergence (Equations (10) and (8)), we can train RBM using the following update rules for weights [13, 6, 4]

$$W_{ij} \leftarrow W_{ij} - \eta(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}). \tag{16}$$

The main question that Equation (16) raises is that how to collect the statistics for $\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}$. The statistics for $\langle v_i h_j \rangle_{data}$ is collected by driving the training datas and the statistics for $\langle v_i h_j \rangle_{model}$ is collected using the first iteration of Gibbs sampling. Based on the recipe that Hinton [13] provides for collecting the statistics for Equation (16), we use probability values for both visible units and hidden units rather than sampled values. However, when the data is driven by hidden units, we use sampled values as opposed to probability values. The statistics for h_j in $\langle v_i h_j \rangle_{data}$ is calculated as

$$P(h_j = 1|v) = sig(b_j + W_j' v). \tag{17}$$

In Equation (17), b_j is the bias unit for the hidden units j and v is the data vector. The statistics for $\langle v_i h_j \rangle_{model}$ is calculated as

$$\begin{gathered} P(h|v) = sig(b + W\prime v) \\ P(v_i = 1|h) = sig(c_i + W_i' h) \\ P(h_j = 1|v) = sig(b_j + W_j' v). \end{gathered} \tag{18}$$

In Equation (18), b is the hidden bias vector, b_j is the bias unit for the jth hidden unit and c_i is the bias unit for the ith visible unit.

We have used momentum and weight decay in implementation of the RBM. Momentum will increase the speed of the learning in RBM training [13]. Weight decay will result in improved generalization and shrinking the large weights to get more smooth output [13].

As mentioned in the energy based models section, contrastive divergence attempts to approximate the negative log likelihood derivative with respect to the weight . Intuitively speaking, we aim to push down the energy surface at the desired target and pull up the surface around the desired target value. By doing this we hope to create a local minima at the desired target. However, it is not guaranteed that we create a global minima at the desired target value. That is a theoretical problem but works well for training RBM.

Since hidden units and visible units are independent from each other, we use Gibbs sampling in order to sample the hidden units and visible units. In fact, Equation (13) shows how to infer the value of each hidden unit or each visible unit while Gibbs sampling is a way to infer visible vector or hidden vector out of each visible unit or hidden unit.

2.5. Reconstruction of MNIST Data Set in Bernoulli-Bernoulli RBM

In this section, we show how Bernoulli-Bernoulli RBM can be used to reconstruct the MNIST digit [2]. MNIST data sets are composed of 60000 training data samples and 10000 testing data samples. Each sample is a hand written digit with 28 pixels in each row and 28 pixels in each column. We have trained an RBM with 784 visible units and 500 hidden units. The reason we have chosen Bernoulli-Bernoulli RBM for reconstruction of MNIST digit data set is that each data vector is binary. In this section, we do not discuss the parameter values. We are just showing that our implementation works. In order to reconstruct the MNIST digits, first we have trained the Bernoulli-Bernoulli RBM with the training data sets. Having trained the RBM, first we calculate the values of the hidden vector using the Equation (13).

Then, we recalculate the values of the visible vector using the Equation (14) to reconstruct the MNIST digit. Finally, we sample the probability values resulted from (14). Here we have provided 100 samples of MNIST data set and their corresponding reconstruction with Bernoulli-Bernoulli RBM. Figure 2.2 shows the MNIST digits and Figure 2.3 shows the reconstructed MNIST digits by Bernoulli-Bernoulli RBM.

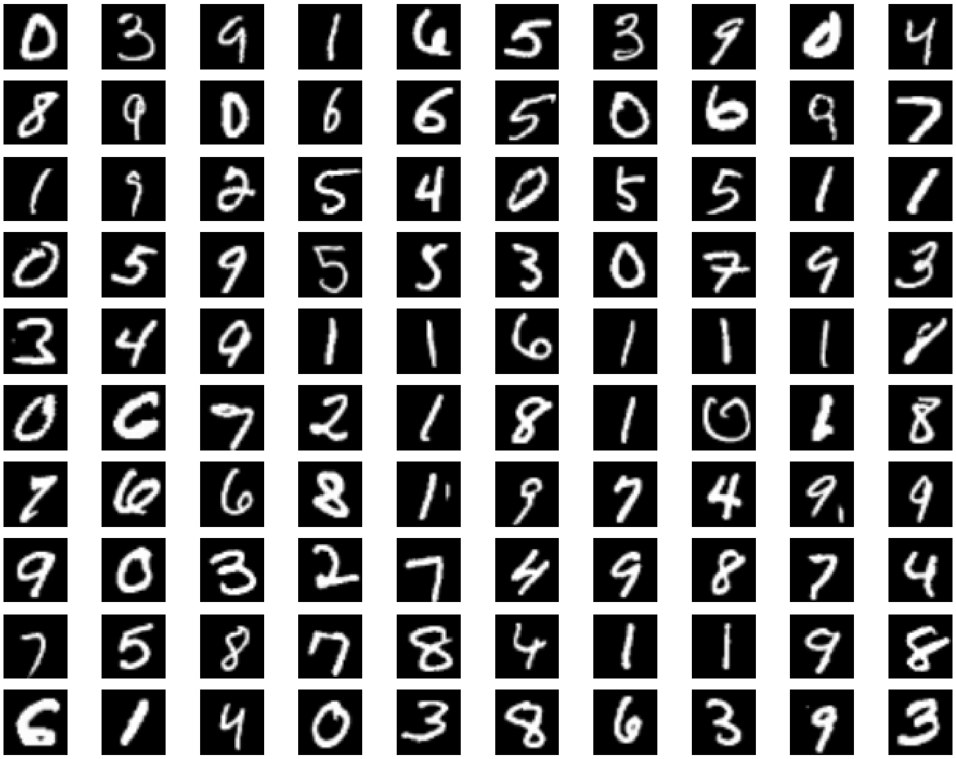

FIGURE 2.2. MNIST digits

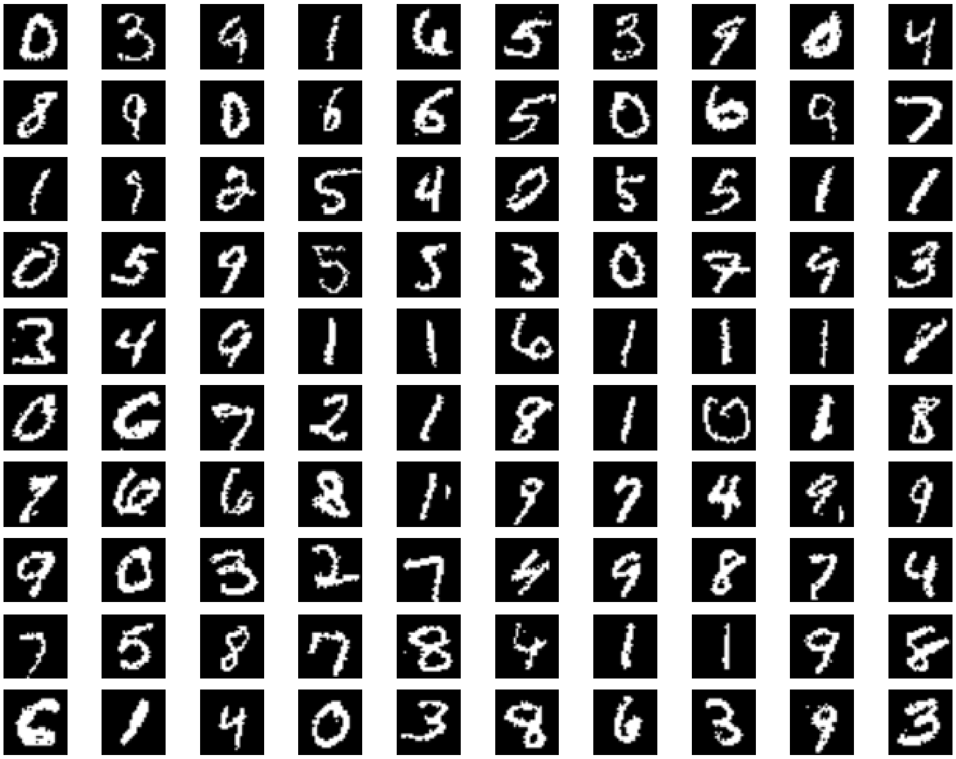

FIGURE 2.3. Reconstruction of MNIST digits

As can be seen in Figure 2.2 and Figure 2.3, the training leads to a successful reconstruction of MNIST digit. By increasing the number of the iterations or the number of the hidden units we can even achieve a more accurate reconstruction of the MNIST data sets.

2.6. Gaussian-Bernoulli RBM

In Gaussian-Bernoulli RBM, visible units are linear with independent gaussian noise and hidden units are stochastic binary variables as before [13]. The energy of the Gaussian-Bernoulli RBM can be calculated as [13]

$$E(v,h,w) = \frac{\sum_{i \in visible}(v_i - a_i)^2}{\sigma^2} - \sum_{j \in hidden} b_j h_j - \sum_i \sum_j \frac{v_i}{\sigma_i} h_j w_{ij}. \tag{19}$$

In Equation (19), σ_i is the standard deviation of the Visible Unit i and a_i is the mean of the ith visible unit. We can simplify the energy function in Equation (19) by normalizing the data set to have mean of zero and variance of 1. Given the energy function and Gibbs distribution, we can derive the inference rule as

$$x_i = c_i + W_i h. \tag{20}$$

Please note that in Equation (20), we have assumed that the data has mean of zero and variance of 1. The inference rule for hidden units are the same as Bernoulli-Bernoulli RBM. Please note that since visible units can get arbitrarily large, we should use a much smaller learning rate in order to train Gaussian-Bernoulli RBM in comparison with Bernoulli-Bernoulli RBM [13]. For Gaussian-Bernoulli RBM, we also use Gibbs sampling in order to sample for training and use contrastive divergence to approximate the negative log likelihood derivative with respect to the weight.

2.7. Reconstruction of P300 Signal Using Gaussian-Bernoulli RBM

In this section, we show that Gaussian-Bernoulli RBM can be used to reconstruct a P300 signal. Since the training data is not binary, we use Gaussian-Bernoulli RBM for reconstruction of the P300 data. Having trained the Gaussian-Bernoulli RBM, first we calculate the values of the hidden vector using the Equation (13). Then, we recalculate the values of the visible vector using the Equation (20) to reconstruct the P300 signal. Figure 2.4 shows the P300 signal and Figure 2.5 shows the reconstructed P300 signal using Gaussian-Bernoulli RBM.

2.8. Minibatch Stochastic Gradient Descent

In mini batch stochastic gradient decent, we start by random initialization of the weights and the biases of each layer. We denote the initialized value of the weight and bias with W_0. Having initialized the weights and biases, we calculate the gradient of error function with respect to the weights and biases. We then update the weights and biases by moving a step in the opposite direction of the gradient. We continue this process iteratively until we reach the satisfactory condition. This can be mathematically expressed as

$$\Delta W^r = -\eta \nabla E. \tag{21}$$

In Equation (21), r is the iteration number. Having chosen the sufficiently small η which is called learning rate, the error will be decreased at each step. The convergence of the

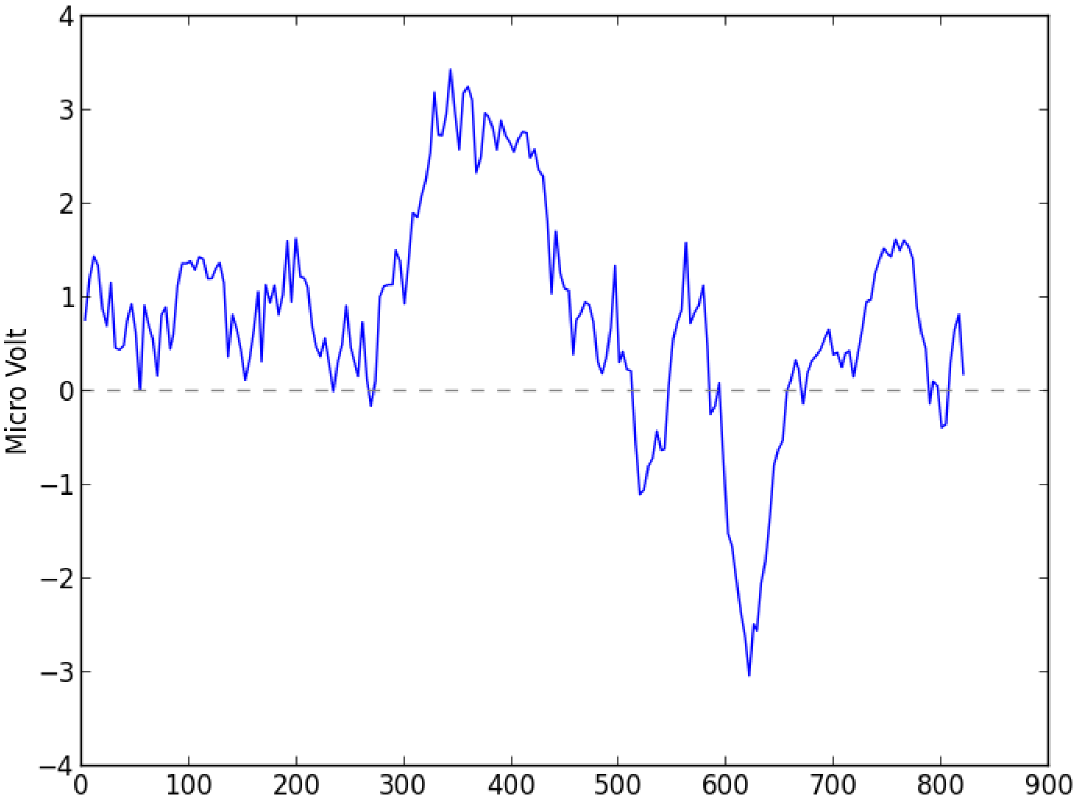

FIGURE 2.4. P300 signal

Equation (21) is guaranteed if we choose a sufficiently small η and $\eta \propto \frac{1}{step-number}$. In this algorithm, using mini batch is specially helpful if the training data sets consist of the redundant samples. Then at each step we can update the parameters based on the gradient values calculated for each mini batch, which includes samples from all classes.

2.9. Back Propagation

The back propagation technique provides an efficient way to calculate the derivative of the error with respect to the weights and biases of each layer. Optimizing the error function using back propagation is an iterative process which is composed of two phases. In the first phase, we calculate the error derivatives with respect to the weights and biases of each layer.

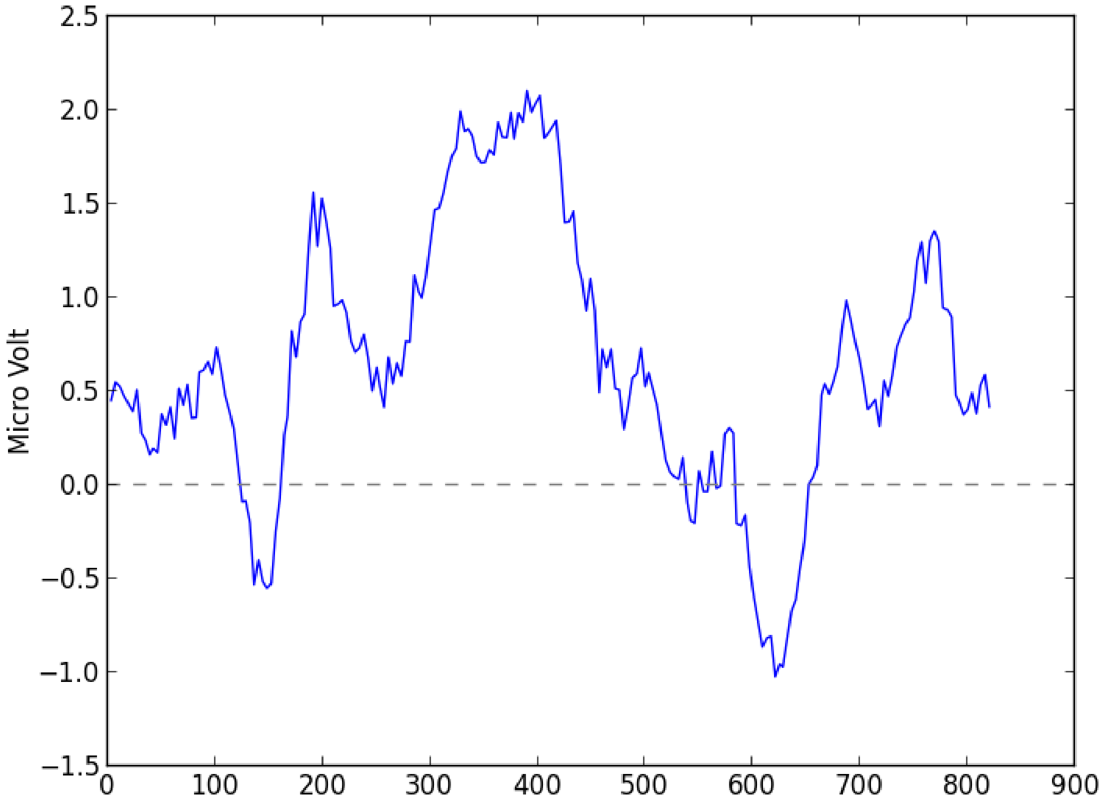

Figure 2.5. Reconstructed image of the P300 signal

In the second phase, we update the weights and biases by moving in the opposite direction of the gradient. Please note that one important restriction that the back propagation algorithm has is that the activation functions of all layers should be differentiable. The overall error can be calculated as

$$E = \sum_{n=1}^{NumberofTrainingSamples} E^n. \tag{22}$$

In Equation (22), the error is considered to be a sum of all the errors, each calculated separately for a training data pattern. As mentioned in the previous paragraph, E^n should be differentiable with respect to the weights and biases. In neural network, the activation of unit j can be calculated as

$$a_j = \sum_i W_{ji} z_i. \tag{23}$$

In Equation (23), z_i is the activation of the ith unit of the previous layer or the input. Assuming that $g(x)$ is a activation function, z_i can be calculated as

$$z_i = g(a_i).$$

Using the chain rule, we can write the derivative of the nth training sample based on w_{ji} as

$$\frac{\partial E^n}{\partial w_{ji}} = \frac{\partial E^n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}.$$

Here we define the δ notation as

$$\delta_j = \frac{\partial E^n}{\partial a_j}.$$

Please note that δ_j is referred to error. Using Equation (23), we would have

$$\frac{\partial a_j}{\partial w_{ji}} = z_i.$$

δ for the output units can be calculated as

$$\delta_k = \frac{\partial a_j}{\partial w_{ji}} = g'(a_k) \frac{\partial E^n}{\partial y_k}. \tag{24}$$

In Equation (24), y_k is the output of the system. Having calculated the δ for the output units using Equation (24), we can back propagate the δ for the hidden units using the following formula

$$\delta_j = g'(a_j) \sum_k w_{kj} \delta_k. \tag{25}$$

In Equation (25), δ_k is the error for the kth unit of the following layer. Having introduced the back propagation formula, we summarize each iteration in the back propagation algorithm as follows:

(1) Feed forward the input

(2) Calculate the δ for the output units using the Equation (24)

(3) Calculate the δ for the hidden units using Equation (25)

(4) Update the weights and biases using the mini batch stochastic gradient descent as introduced in the previous section.

(5) If training needs to be continued, repeat the above steps.

2.10. Logistic Regression

Logistic Regression is a probabilistic model which is widely used for discrimination. In this method, the number of the target variables are equivalent to the number of different class labels. For example, for hand written digit recognition, since we have 10 different numbers, the number of the target variables would be 10. Given the data vector X, we can calculate the distribution of vector X belonging to different classes using this method. This is called the $1 - of - the - C$ coding scheme. One restriction we have here is that the sum of all probability values for a given data vector X must be 1. This can be mathematically expressed using the following formula

$$P(C = k|X) = g_k(X) = \begin{cases} \dfrac{f(X, W_k)}{1 + \sum_{m=1}^{K-1} f(X, W_m)}, & k < K \\ \dfrac{1}{1 + \sum_{m=1}^{K-1} f(X, W_m)}, & k = K. \end{cases} \tag{26}$$

Assuming we have K class labels, $P(C = k|X)$ in Equation (26) is the probability of belonging vector X to class k. W represents the parameter of function f and is tuned during the training. In Equation (26), $f(X, W)$ can be calculated as

$$f(X, W) = e^{WX}.$$

We chose the negative log likelihood loss function to measure how good our model is during the training. As discussed in energy based model section, logistic regression can be seen as an energy based model with negative log likelihood as the loss function. Given the Equation (26), the loss function can be calculated as

$$L(W) = \prod_{n=1}^{N} \prod_{k=1}^{K} p(C = k|x_n)^{t_{n,k}}. \tag{27}$$

In Equation (27), W is the parameter of function f and $t_{n,k}$ is a binary value(i.e. it is always 0 or 1). $t_{n,k}$ is 1 if and only if the nth training sample belongs to the kth class. Using stochastic gradient descent as optimization algorithm, we have the following rule for updating W

$$W \leftarrow W + \alpha X^T (T - g(X)).$$

W is updated in a direction which minimizes the loss function. Given X and W, the way that we infer the target in linear logistic regression is as

$$T = \underset{k \in 1,\ldots,K}{\operatorname{argmax}} P(C = k|X). \tag{28}$$

In (28) equation, we have assumed that there are K distinct classes. Among different probability values assigned to vector X belonging to different classes, we infer the target value as a class label for which system assigned highest probability.

2.11. Deep Belief Network Architecture

In this section, we describe the architecture of the DBN implemented to model the P300 data. We have used the architecture and the training algorithm suggested by [14, 6]. Figure 2.6 shows the architecture of the deep belief networks implemented in this thesis.

We start by introducing the activation functions of different layers and the input data pattern. The input data is standardized to have mean of 0 and standard deviation of 1. The activation function for hidden layers is sigmoid function and the last layer is logistic regression. In this thesis, we use the deep belief networks for classification purposes. The parameters including the number of hidden layers will be discussed in the next chapter. However, two output units were used since we have two class labels for the data. The number of the input units is equivalent to the data dimension.

The training has two phases. The first phase is unsupervised pre-training of each layer. We use Gaussian-Bernoulli RBM to pre-train the first layer and we use Bernoulli-Bernoulli RBM to pre-train the rest of the layers except the final layer. The final layer, which is the classifier, is trained using linear logistic regression. In pre-training phase, we pass the probability values as opposed to sampled values to the next layer as suggested by Hinton [13].

Having finished the pre-training phase, we start the second phase of training called fine-tuning. In this phase, we first initialize the parameters of each layer with the values learned from RBM pre-training. The loss function we use for this section is negative log likelihood. We use mini batch stochastic gradient descent in order to optimize the loss function. We divide the training data sets into smaller sections called mini batch. In each iteration, we update the weights and biases based on the gradient calculated for each mini batch. We also use back propagation algorithm in order to calculate the error in different layers. We propagate the data exactly the same as the usual neural network.

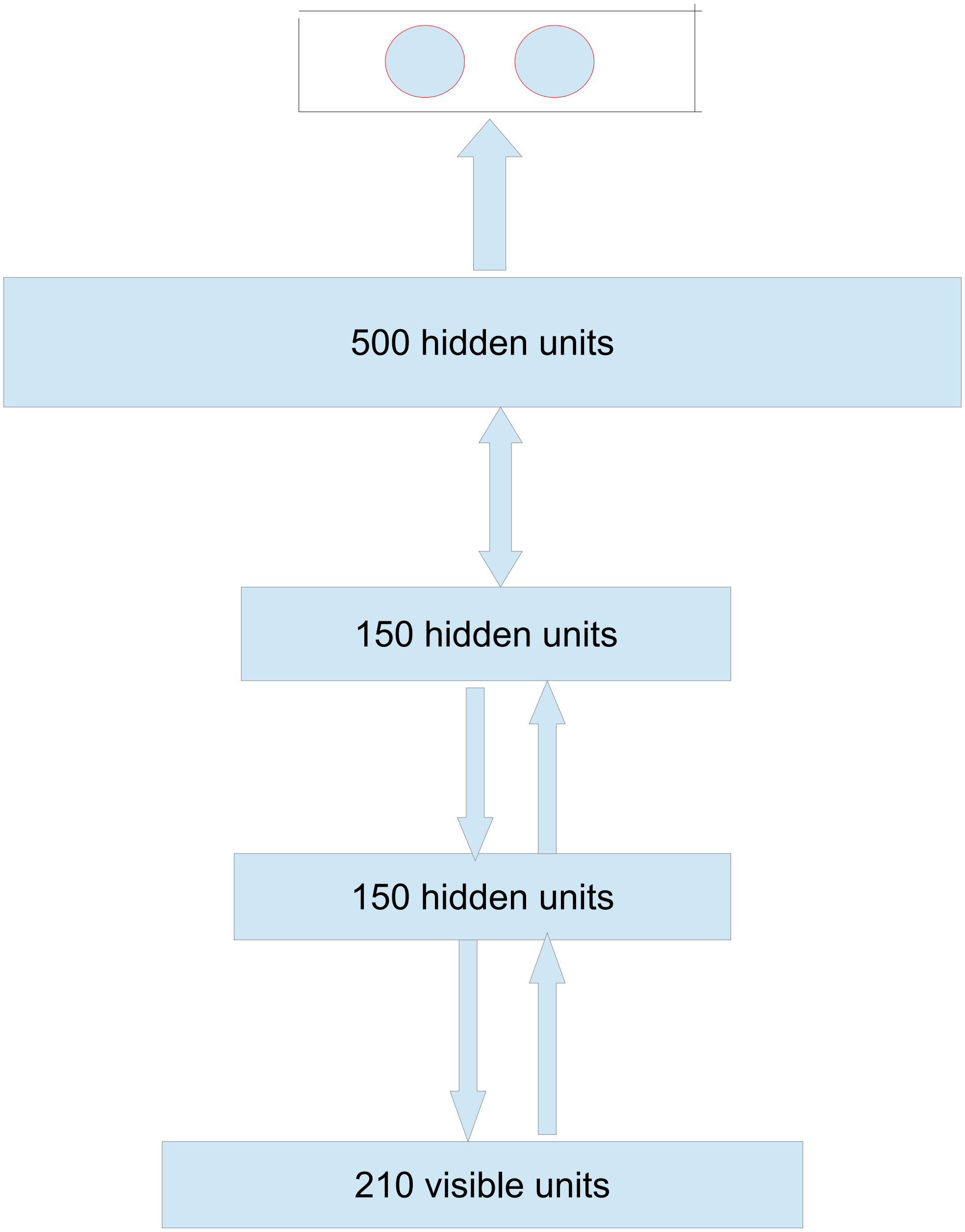

Figure 2.6. The Deep Belief Network Architecture

CHAPTER 3

Experiments and Results

In this chapter, we discuss the results we have achieved by applying deep belief networks to model P300 signal for classification. In the following sections, first we introduce our algorithm for classification and then we present the classification accuracy we have achieved. Finally, we analyze the effect of the unsupervised pre-training and fine-tuning on classification and will discuss two testing data samples one of which has been classified correctly and the other has been misclassified.

3.1. Experimental Data

In this section, we introduce the data sets we used to conduct our experiments. Colorado State University has collected P300 data sets from healthy subjects in the lab and disabled subjects at home. The data set is available at [1]. In this thesis, we have modeled the data recorded at home and the lab by g.GAMMAsys EEG recording system. The data sample rate is 256 HZ. The Brain Computer Interface Lab at Colorado State University [1] provides us with the data recorded from nine different subjects at the lab and four different subjects at home. The DBN was trained and tested on each subject separately. Although [1] provides a tutorial for how to preprocess the data, we reiterate some part of this tutorial here since it is crucial for the reader to understand. For each subject, the EEG data is collected under different protocols. In this thesis, we have just conducted experiments for the EEG data collected under letter-b, letter-p and letter-d protocols. Letter-b, letter-p and letter-d protocol are visual stimuli protocols designed to elicit the P300 signal. During each protocol, letters are presented about every second and each trial consists of 210 samples. For example for the letter-d protocol, the subjects were asked to count the number of the times that letter d appeared in the middle of the screen. Figure 3.1 shows the EEG signals collected by eight different channels under the letter-d visual stimuli protocol.

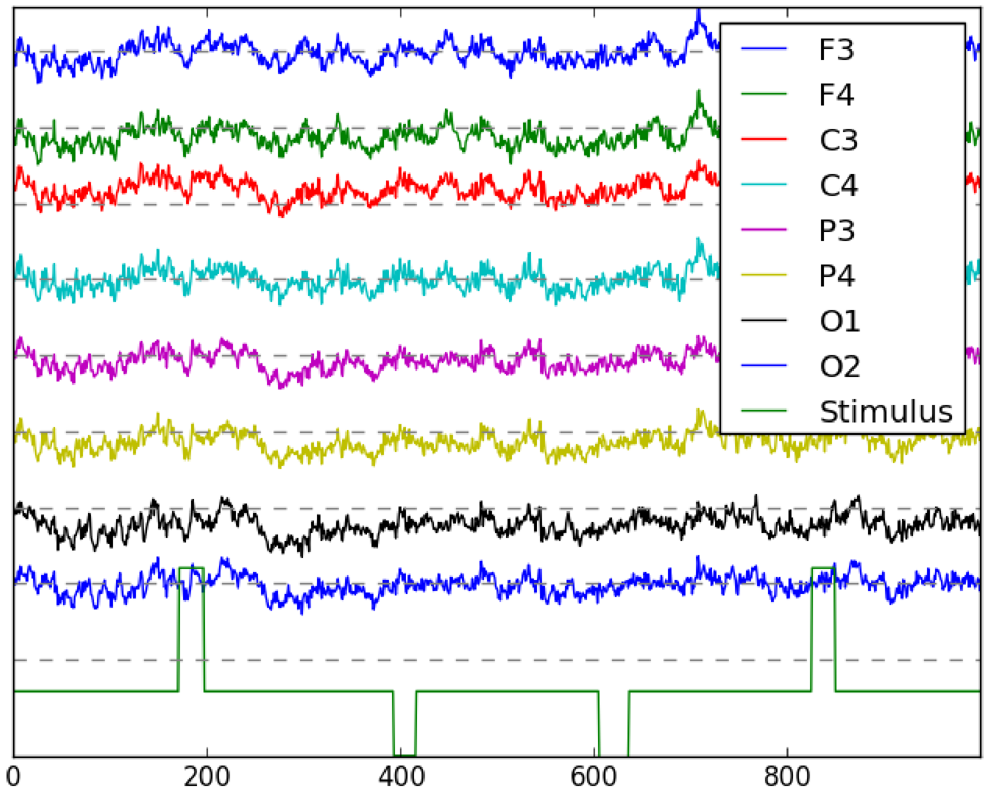

Figure 3.1. EEG signals collected by 8 different channels under the letter-d visual stimuli protocol

In Figure 3.1 the green line at the bottom shows the ASCII number of the figure displayed on the screen. The negative values represent the ASCII of the non-target letters and the positive values represent the ASCII of the target letter.

3.2. Parameter Selection

In this section, we discuss the parameter values used in our experiments. As discussed in the previous chapter, deep belief networks training has two main sections. The first section is the pre-training section and the next section is the fine-tuning section. In the pre-training section, we have chosen most of the parameter values as advised by Hinton [13]. However, we have initialized some parameters intuitively and we have not completely followed [13]. The weights and biases are initialized to have a mean of 0 and a standard deviation of 0.01. The bias values are initialized to 0. Hinton suggests to set the mini batch size to the number of different class labels. Although we have just two class labels here, since the EEG data is very noisy, we have set mini batch size to 10. We have set the learning rate for Bernoulli-Bernoulli RBM to 0.1 and Gaussian-Bernoulli RBM to 0.001. As far as momentum values go, we have set the momentum value to be 0.5 for the first five iterations and 0.9 for the rest of the training. We have used the value of 0.0002 for the weight decay. Table 3.1 and Table 3.2 summarizes the parameter values we chose for pre-training the DBN.

TABLE 3.1. pre-training parameter values for Bernoulli-Bernoulli RBM

parameter	value
mini batch size	10
momentum value	[0.5-0.9]
learning rate	0.1
number of iterations	200
weight decay	0.0002

Table 3.2. pre-training parameter values for Gaussian-Bernoulli RBM

parameter	value
mini batch size	10
momentum value	[0.5-0.9]
learning rate	0.001
number of iterations	500
weight decay	0.0002

Our DBN architecture consists of four different layers. The first three layers RBMs machines and the last layer is the classifier layer. The first layer is believed to capture the low level feature of the P300 data. The second layer is believed to capture the higher level feature of the P300 data or in other words, the second layer captures the feature of the features of the P300 data. Finally, the third layer is believed to represent the high level features in a high dimensional space which makes the classification simpler. Hinton suggests that the number of hidden units in the first and the second layer should be the same. So we have chosen the same number of hidden units for the first and the second layers. We have not made experiments to choose the number of the hidden units in different layers and we have done this intuitively. Since the data is noisy, we have chosen the number of hidden units to be 150 for the first two layers and 500 for the third layer. The reason that we have chosen 500 hidden units for the third layer is that we want to represent the data in a high dimensional space before classification. We have chosen the learning rate to be 0.001 since in the previous chapter we mentioned that if we use a small enough learning rate, the stochastic gradient descent optimization algorithm will converge. So we chose the learning rate to be as small as 0.001. We set the number of iterations to be very large. We stop training using validation data sets. When the training likelihood of the validation data sets start decreasing for a certain number of iterations, we stop the training. This is the way suggested by Stephen Marsland [23].

3.3. Classification Accuracy

In this section, we provide the results we achieved for the classification accuracy for different subjects. In our experimental data, we have a total of 80 trials for each subject. Each trial, consists of eight EEG channels of data. We have divided the data sets into the disjoint training, testing and validation data subsets. Our training data sets consist of 70 percent of all of the data and each of the validation and the testing data set consist 15 percent of all of the data sets. Our training algorithm is as follows:

(1) We randomly choose 70 percent of the target trials and 70 percent of the non-target trials as training trials. Since we have 80 trials, we would have 56 training trials for each subject.

(2) Each trial consists of eight channels. We combine the different channels of the various trials together. These are our training data sets. Since we have 56 training data trials, we would have $56 * 8 = 448$ different training samples for each subject. Each training sample has a dimension of 210.

(3) We build our validation data sets similarly to our training data sets. The only difference is that we randomly choose 15 percent of the target trials and 15 percent of the non-target trials as the validation data set. So we would have 12 trials as validation trials and the number of the validation data samples would be $12*8 = 96$.

(4) We build our testing data sets in the same way as the validation data sets.

(5) We start training (pre-training and fine-tuning) the system using training data samples. We stop training when the likelihood of the validation data sets decreases for a certain number of the iterations.

(6) After training the system, we classify testing data samples using DBN.

(7) The class labels for all 8 channels are collected. The predicted label that results most often among the 8 outputs is selected, performing a majority vote. If there are equal number of P300 and nonP300 labels, then the trial will be classified as nonP300. For example, if 5 channel are classified as P300 and 3 channels are classified as nonP300, then we classify that trial as a P300 trial.

Using the above algorithm, the highest percent count we achieve for testing individual channels is 85.33 and combining individual channel results with majority vote gives testing trial accuracy of 83.33. The average classification accuracy we achieved across all subjects is 76.38 for testing individual channels and 75.6 for testing trials. Please note that if all the trials are classified as non-target, we will get 75% accuracy. The Figures 3.2, 3.3 and 3.4 show the results of our experiments using the above algorithm. In Figures 3.2, 3.3 and 3.4, the x-axis shows the subject number and the y-axis shows the classification accuracy percentage. As you can see, the highest accuracy we achieved belonged to the first subject. Figure 3.2 shows the training likelihood for each subject. Figure 3.3 shows the classification accuracy percentage of the EEG channels for each subject, while Figure 3.4 shows the classification accuracy of the trials for each subject.

In order to increase the accuracy of our trial classification, we use only the four channels at the top of the head which are more sensitive to capturing the P300 signals. The rest of the algorithm is similar to what we explained above. Our new algorithm would be as follows:

(1) - We randomly choose 70 percent of the trials as training trials. Since we have 80 trials, we would have 56 training trials

(2) - Each trial consists of eight channels. We throw four channels out and we just continue our experiments with C3, C4, P3 and P4 channels. We combine the different channels of the different trials together. These our training data sets. Since we have

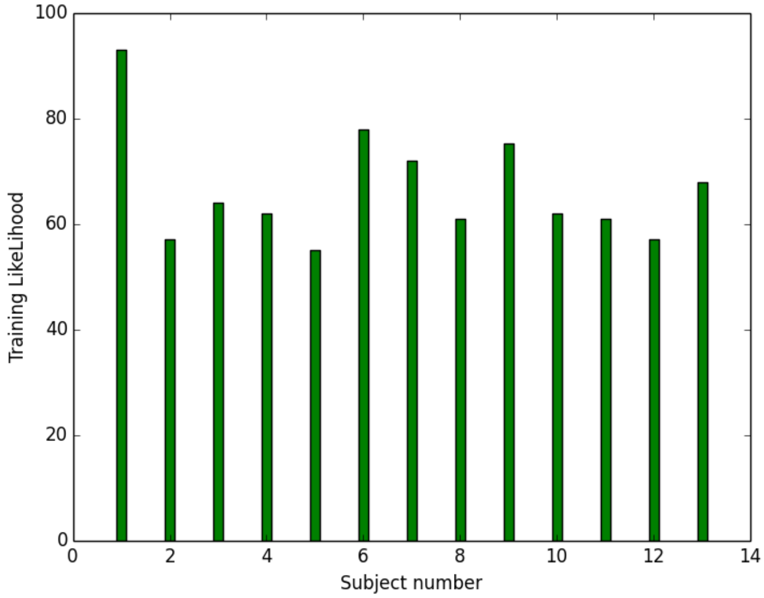

Figure 3.2. The training likelihood of the data for each subject

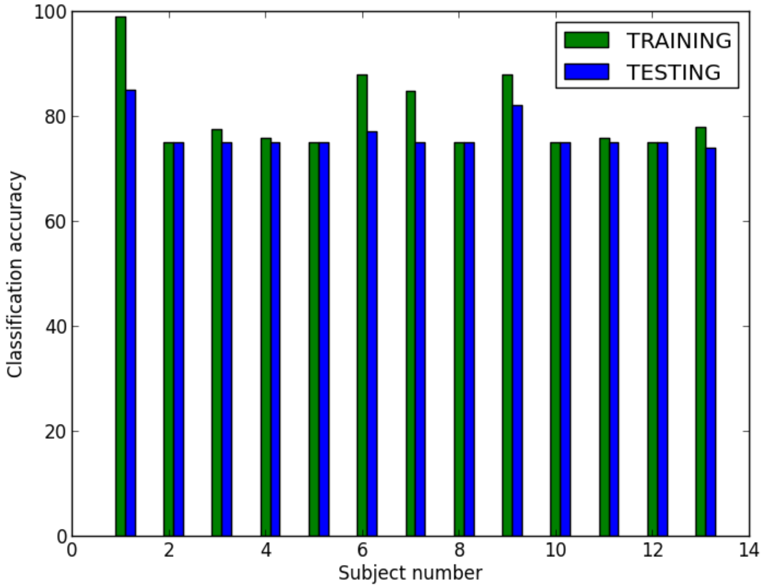

Figure 3.3. The classification accuracy of the EEG channels for each subject

56 training data trials, we would have $56 * 4 = 224$ different training samples. Each training sample has a dimension of 210.

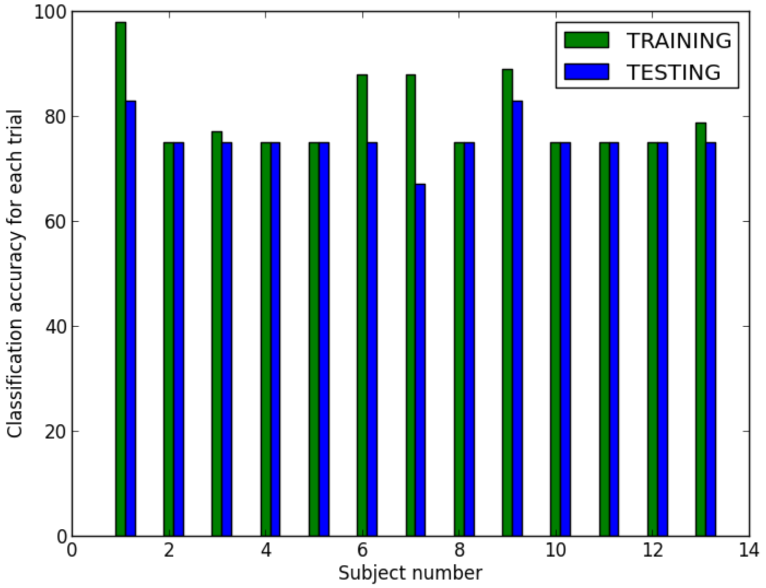

FIGURE 3.4. The classification accuracy of the trials for each subject

(3) - We build our validation data sets similarly to our training data sets. The only difference is that we randomly choose 15 percent of the trials as the validation data set. So we would have 15 trials as validation trials and the number of the validation data samples would be $12 * 4 = 48$.

(4) - We build our testing data sets in the same way as the validation data sets.

(5) - We start training the system using training data samples. We stop training when the likelihood of the validation data sets starts decreasing for a certain number of the iterations.

(6) - After training the system, we classify each channel using DBN.

(7) - The channel output with majority vote will determine the trial label. If there are equal number of P300 and nonP300 labels, then the trial will be classified as nonP300. For example, if three channels are classified as P300 and one channel is classified as nonP300, then we classify that trial as P300 trial.

The Figure 3.5 show the result of our experiments for different subjects. In Figure 3.5, the x-axis shows the subject and the y-axis shows the classification accuracy for testing and training samples. In Figure 3.5, the left column of plots show he classification accuracy achieved for each channel and the right column shows the claasification accuracy achieved for each trial. In Figure 3.5 each row is for a different letter. Table 3.3 summarizes the average classification accuracy we achieved for each subject across letter b, letter d and letter p.

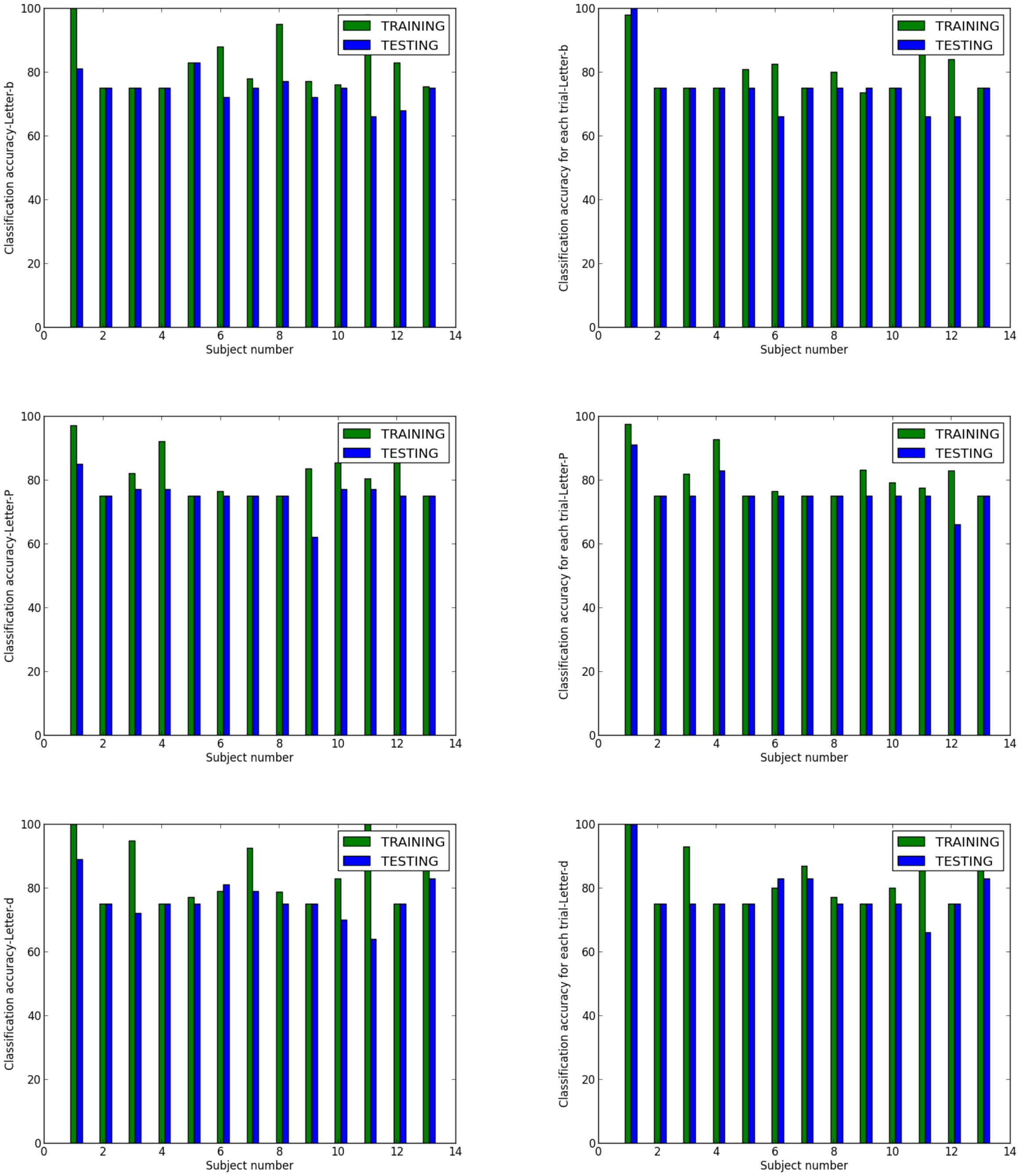

FIGURE 3.5. Classification accuracy achieved for different subjects

TABLE 3.3. Average classification accuracy achieved for each subject across letter b, letter p and letter d

subject	percentage
subject-1	97
subject-2	75
subject-3	75
subject-4	77.6
subject-5	75
subject-6	75
subject-7	77.6
subject-8	75
subject-9	75
subject-10	75
subject-11	69.6
subject-12	69.6
subject-13	77.6

In the next section, we will look at the samples that DBN could classify correctly and the samples that DBN failed to classify correctly and we try to justify the response.

3.4. Analyzing Classified Samples

In this section, we will have a closer look at the samples that DBN classified correctly and the samples that DBN misclassified. However, first we analyze the effect of pre-training and fine-tuning on the classification results of the first subject. After pre-training, the DBN classifies all of the testing data samples as nonP300 signals and it does not classify any P300 signals correctly. This is expected since with unsupervised pre-training, the system does not have any information to discriminate the signals from each other. The system is just able to capture the features of the input at different layers waiting for supervised fine-tuning to learn how to correspond higher level feature representation of the input to the correct labels. So the classification accuracy we achieve after the pre-training is 75 percent. After the fine-tuning, the DBN can correctly discriminate all of the P300 signals.

Figure 3.6 shows a P300 signal which has been classified correctly. Figure 3.7 shows a P300 signal which has been misclassified as nonP300 signal using DBN. As you can see, the P300 signal in Figure 3.7 is much noisier compared with the P300 signal showed in Figure 3.6. Another possible reason that the DBN misclassified this P300 sample is that this sample may actually be missing the expected P300, perhaps the subject was not concentrating. One possible way for DBN to classify the P300 signal, represented in Figure 3.7, is to increase the number of the layers or de-noising the signals before the classification. In this case, the higher level representation of the signal might contain more of the P300 signal features and as a result would be easier for the classifier to classify correctly.

As stated in the previous chapter, in RBMs hidden units represent some features of the data. In this section we visualize what features each hidden unit represents by determining what maximizes the activation of each hidden unit. We have done this in two ways. In the next sections, we briefly describe the algorithms and represent the results we achieve using each method.

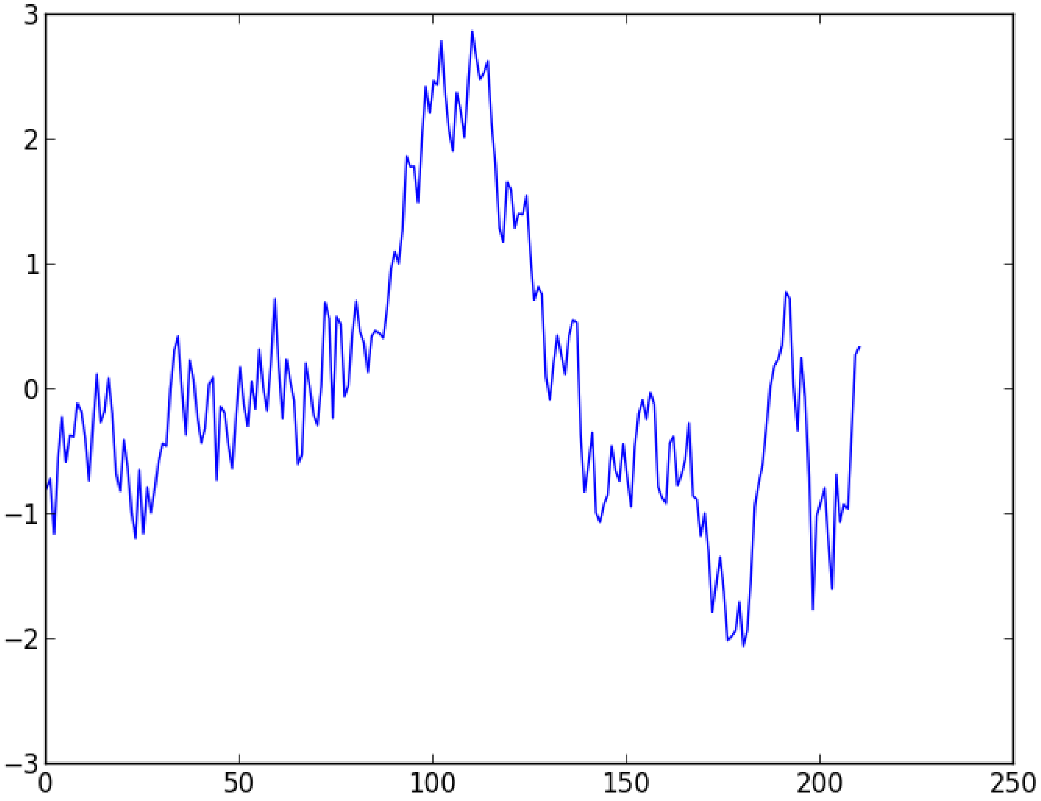

FIGURE 3.6. A correctly classified P300 sample

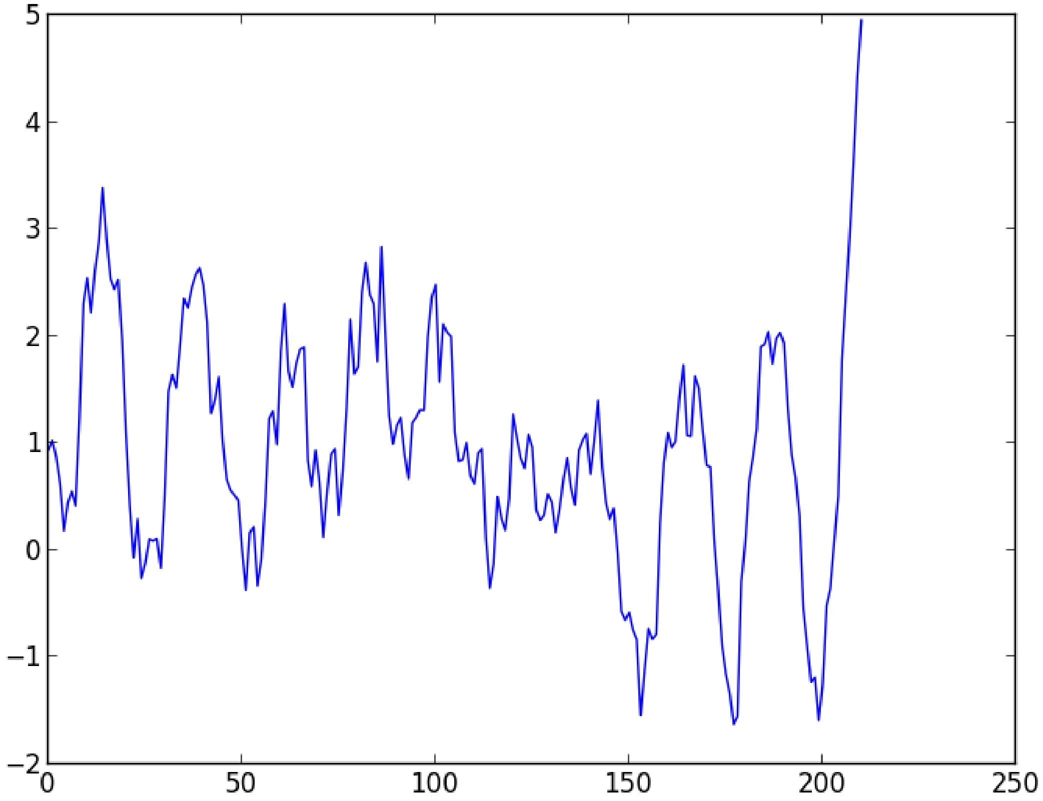

Figure 3.7. A misclassified P300 sample

3.5. Hidden Unit Visualization by Solving an Optimization Problem

One way to determine what aspects of the input data maximizes the activation of each hidden unit is solving the optimization problem proposed by Bengio [9]. In this algorithm, we first train the system using the algorithm mentioned in the previous chapter. Having trained the system, we have the values of the weights and biases of the DBN. In order to determine the input pattern which maximizes the hidden unit of interest, we can solve the optimization problem with respect to the input vector. In order to do that, first we write the activation of the hidden unit of interest with respect to the input vector. Since in our architecture we have used three layers of the RBMs, we provide the activation of the hidden units of each layer with respect to the input pattern. For the first layer we would have

$$H_j = W_{1j}X + b_{1j}. \tag{29}$$

In Equation (29), H_j is the activation of the hidden units of the interest in the first layer and W_{1j} is the jth column of the weights in the first layer and b is the bias for the jth hidden units. The derivative of the H_j in Equation (29) with respect to X can be calculated as

$$\frac{\partial H_j}{\partial X} = W_{1j}. \tag{30}$$

In order to calculate the activation of the hidden unit of interest in the second layer, we would have

$$H_j = W_{2j} sigm(W_1 X + b_1) + b_{2j}. \tag{31}$$

In Equation (31), W_1 is the matrix weight of the first layer and W_{2j} is the jth column of the second layer weight matrix. b_1 is the bias for the hidden units of the first layer and b_{2j} is the bias of the jth hidden units of the second layer. The derivative of the activation with respect to the input vector can be calculated as

$$\frac{\partial H_j}{\partial X} = W'_{2j} diag(sigm(W_1 X + b_1) * (1 - sigm(w_1 X + b_1)) W'_1. \tag{32}$$

In Equation (32), $*$ implies element-wise matrix multiplication. In order to calculate the activation of the hidden unit of interest in the second layer, we would have

$$H_j = W_{3j} sigm(W_2 sigm(W_1 X + b_1) + b_2) + b_{3j}. \tag{33}$$

In Equation (33), W_1 and W_2 are the weights of the first and the second layer and W_{3j} is the jth column of the matrix weight in the third layer. b_1 and b_2 are the hidden biases of the first and the second layer and b_{3j} is the bias of the jth hidden units in the third layer. The derivative of the activation with respect to the input vector can be calculated as

(34)

$$\frac{\partial H_j}{\partial X} = W'_{3j} diag(sigm(W_2 sigm(W_1X + b_1) + b_2) * (1 - sigm(W_2 sigm(W_1X + b_1) + b_2)))$$

$$W'_2 diag(sigm(W_1X + b_1) * (1 - sigm(w_1X + b_1))W'_1.$$

In Equation (34), $*$ implies element-wise matrix multiplication. In our experiment, first we initialized each visible unit independently to a have normal distribution with mean of 0 and variance of 1. We then optimized the input vector using stochastic gradient ascent to maximize the activation of each unit. In Figure 3.8, we have provided the input patterns which maximize the activation of 100 hidden units, randomly chosen from the third layer. As you can see in Figure 3.8, the patterns that maximize the activations are very similar to each other. The first reason for this is that the number of the training data samples is too low compared to the number of the hidden units in the third layer. The second reason is that there is no guarantee that the pattern we find to maximize the hidden units is among the training data samples. This could be also the reason that the data looks very noisy.

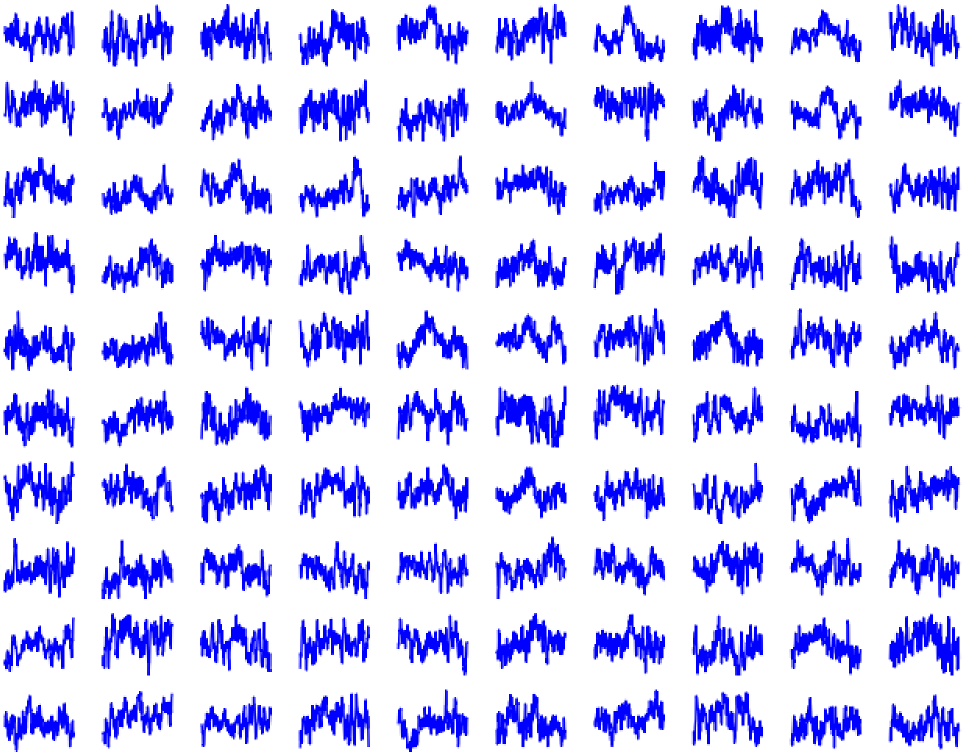

FIGURE 3.8. Solving optimization problem method- the patterns which maximize the activation of 100 hidden units.

3.6. Hidden Unit Visualization by Weighted Average of Inputs

In this section we provide another way to visualize the hidden units in the third layer. In this method, first we calculate the probability value that each hidden unit represents for each training data sample. Then, we calculate the weighted average of the training data samples. The formula for calculating the weighted average is as

$$\frac{\sum_{i=0}^{n} A_{ji} X_i}{\sum_{i=0}^{n} A_{ji}}. \tag{35}$$

In Equation (35), A_{ij} is the probability value of the jth hidden unit for the ith training sample, X_i is the ith training sample and n is the number of the training samples. The result we get for visualizing the 100 hidden units, randomly chosen from the third layer is shown in Figure 3.9. As you can see in Figure 3.9, the patterns that maximize the activation of hidden units are very similar to each other. Many patterns contain the P300 signal, but some patterns do not contain P300 signals.

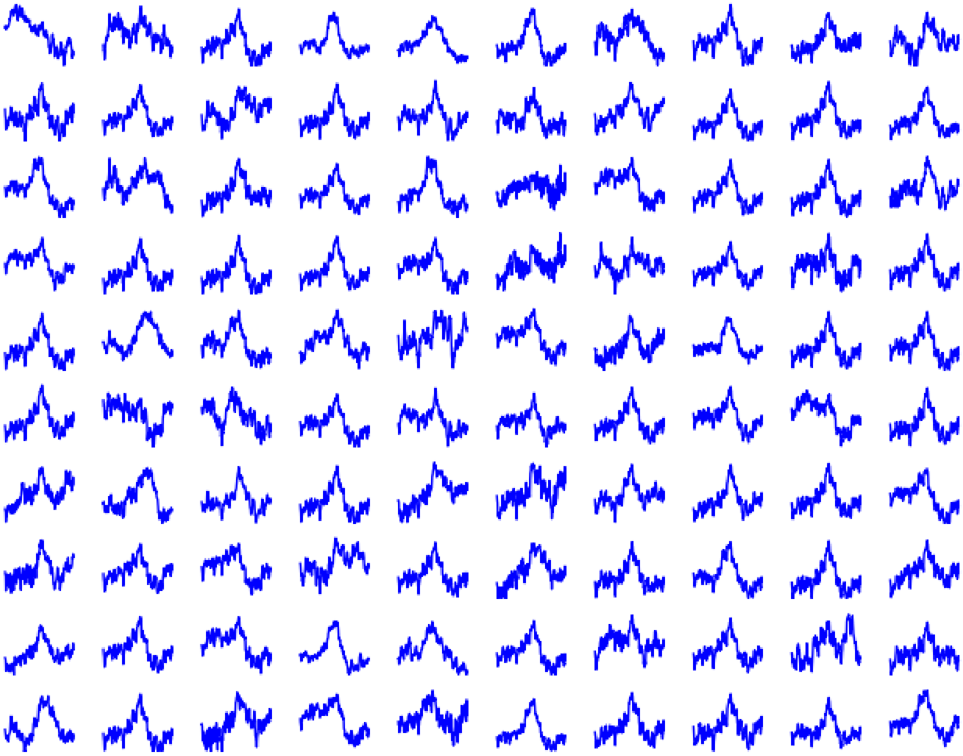

FIGURE 3.9. Weighted-Average Method-the average of training samples weighted by hidden unit activations.

CHAPTER 4

Conclusion

In this thesis, we provided the first attempt to model P300 data for classification using deep belief networks. In order to do that, we implemented a DBN package in python based on the algorithm that Hinton provided for training the DBN. Our DBN consists of four layers. The final layer is the classifier layer and we have implemented logistic regression for this purpose. The first three layers are pre-trained using RBMs. We pre-trained the first layer using Gaussian-Bernoulli RBM and pre-trained the two following layers using Bernoulli-Bernoulli RBM. We used weight decay and momentum in our implementation of the RBM. For the fine-tuning section, we used negative log likelihood as loss function and used back propagation algorithm to calculate the derivative of the error with respect to weights and biases. We optimized the parameters using mini batch stochastic gradient descent algorithm.

In the past, researchers have examined different classifiers to discriminate P300 signal from nonP300 signals. However, no one has tried any kind of deep learning algorithms for P300 classification. The application of deep learning algorithms on EEG time series are limited to analysis of epilepsy [25], but this article has not paid much attention to model P300 waves for classification. Researchers mostly have applied DBN for image classification, voice recognition, or other high dimensional data. This thesis, provides the first attempt to implement and apply the DBN to model the P300 signals for classification. In this thesis we showed that DBN works very well in modeling the P300 signals for classification. As summarized in the review by Sharma [22], other methods on average achieve between 60 % to 90 % classification accuracy for a single P300 trial. Our method on average achieves between 69 % and 97 % classification accuracy for a single P300 trial. The average classification accuracy we achieved across all subjects was higher for testing individual channels than testing trials. This suggests that maybe the majority vote is not the best way to determine the label of each trial. In our experiments, we used EEG data set collected by the BCI lab at Colorado State University on both healthy and disabled subjects.

The deep belief networks is a young algorithm and there is a lot to be done in this field to improve the algorithm. In this thesis, we focused on reporting classification accuracy we achieved for subjects. However, we did not investigate the training and the classification time of this algorithm which is a crucial factor for building a real time BCI system. In fact, the training and classification time of this algorithm seem to be higher than most of the other machine learning classifiers.

We also used a logistic regression as the final layer of our DBN. Researchers have also proposed other options such as Support Vector Machines and RBM to be used in the final layer of the DBN as a classifier. Investigating these options may lead to a faster and more accurate implementation of DBN for modeling P300 data.

We also did not investigate the effects of the different parameters such as the number of the hidden layers, learning rate, momentum value and weight decay values formally. We used Hinton's guide to set up most of the parameters in pre-training. The rest of the parameters were set up intuitively. We believe it is worth trying to examine the effect of different parameters throughly to improve the performance.

One important question is how would a deep architecture model P300 data for classification with supervised pre-training or semi-supervised fine-tuning. As stated earlier, in this thesis, we used unsupervised pre-training. However, supervised pre-training of deep architectures may lead to a faster and more accurate deep learning algorithm.

As stated earlier, the training of the deep belief network is time consuming. One approach that could decrease the training time is to transfer the learning between two independently trained DBN. For example, we may initialize the DBN parameter values using the result of another subject's pre-training. This approach may result in faster training of the DBN. This is an approach which has not gained much attention in the literature.

BIBLIOGRAPHY

[1] Experimental data. `http://www.cs.colostate.edu/eeg`.

[2] The mnist database of handwritten digits. `http://yann.lecun.com/exdb/mnist/`.

[3] A. E. Selim and M. Abdel and Y. M. Kadah. Machine learning methodologies in brain computer interface systems. *Biomedical Engineering Conference*, pages 1–5, 2008.

[4] G. Bakir, T. Hofman, B. Scholkopf, A. Smola, and B. Taskar. *Predicting Structured Data*. MIT Press, 2006.

[5] A. H. Barnett, D. Cohen, and E. J. Heller. Talking off the top of your head: toward a mental prosthesis utilizing event-related brain potentials. *Electroencephalography and Clinical Neurophysiology*, 70:510–523, 1988.

[6] Y. Bengio. Learning deep architectures for AI. *Foundation and Trends in Machine Learning*, 2:1–127, 2009.

[7] Y. Bengio, P.Lamblin, D. Papovici, and H.Larochelle. Geedy layer-wise training of deep networks. In B. Scholkopf, J.Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems*, volume 19, pages 153–160. MIT Press, 2007.

[8] Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Clarendon Press, Oxford, 1995.

[9] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualizing a higher-layer features of a deep network. Technical Report 1341, University of Montreal, June 2009.

[10] D. Erhan, P. Manzagol, Y. Bengio, S. Bengio, and P.Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *Proceedings of Twelfth International Conference on Artificial Intelligence and Statistics*, pages 153–160, 2009.

[11] V. Erp, F. Lotte, and M. Tangermann. Brain-computer interface: Beyond medical applications. *Computer*, 45:26–34, 2012.

[12] R. A. Fisher. The use of multiple measurments in taxonomic problems. *Ann. Eugenics*, 7:179–188, 1936.

[13] G. Hinton. A practical guide for training restricted boltzmann machine. Technical Report 2010-003, Department of computer Science, University of Toronto, August 2010.

[14] G. E. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

[15] D. J. Krusienski, E. W. Sellers, and F. Cabestaing. A comparsion of classification techniques for the p300 speller. *Journal of Neural Engineering*, 3:299–305, 2006.

[16] H. Larochelle, Y. Bengio, J. Louradour, and P.Lamblin. Exploring strategies for deep neural networks. *Journal of Machine Learning Research*, 10:1–40, 2009.

[17] Mark Libenson. *Practical approach to Electroencephalography.* Saunders, 2009.

[18] D. N and S. H. Applied regression analysis. *Wiley : New York*, 2:307–312, 1981.

[19] R. Panicker, S. Puthusserypady, and Y. Sun. Adaptation in p300 brain computer interfaces: A two classifier cotraining approach. *IEEE Transactions on Biomedical Engineering*, 57:2927–2935, 2010.

[20] R. Panicker, S. Puthusserypady, and Y. Sun. Comparison of classification methods for p300 brain computer interface on disabled subjects. *Computational Intelligence and Neuroscience*, vol. 2011:1–12, 2011.

[21] E. W. Sellers and E. Donchin. A p300-based brain-computer interface: initial tests by als patients. *Clinical Neurophysiology*, 117:538–548, 2006.

[22] N. Sharma. Single-trial p300 classification using pca with lda and neural networks. Master's thesis, Colorado State University, December 2013.

[23] Stephen Marsland. *Machine Learning: An algorithmic perspective.* Taylor and Francis, 2009.

[24] Vapnik V. *Statistical Learning Theory.* Wiley, 1998.

[25] D. F. Wulsin, J. R. Gupta, R. Mani, J. A. Blanco, and B. Litt. Modeling electroencephalography waveforms with semi-supervised deep belief nets: fast classification and anomaly measurement. *Journal of Neural Engineering*, 8:1741–2552, 2011.