THESIS

SUPERVISED AND UNSUPERVISED TRAINING OF DEEP AUTOENCODER

Submitted by

Tomojit Ghosh

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Fall 2017

ABSTRACT

SUPERVISED AND UNSUPERVISED TRAINING OF DEEP AUTOENCODER

Deep learning has proven to be a very useful approach to learn complex data. Recent research in the fields of speech recognition, visual object recognition, natural language processing shows that deep generative models, which contain many layers of latent features, can learn complex data very efficiently. An autoencoder neural network with multiple layers can be used as a deep network to learn complex patterns in data. As training a multiple layer neural network is time consuming, a pre-training step has been employed to initialize the weights of a deep network to speed up the training process. In the pre-training step, each layer is trained individually and the output of each layer is wired to the input of the successive layers. After the pre-training, all the layers are stacked together to form the deep network, and then post training, also known as fine tuning, is done on the whole network to further improve the solution. The aforementioned way of training a deep network is known as stacked autoencoding and the deep neural network architecture is known as stack autoencoder. It is a very useful tool for classification as well as low dimensionality reduction.

In this research we propose two new approaches to pre-train a deep autoencoder. We also propose a new supervised learning algorithm, called Centroid-encoding, which shows promising results in low dimensional embedding and classification. We use EEG data, gene expression data and MNIST hand written data to demonstrate the usefulness of our proposed methods.

ACKNOWLEDGEMENTS

To begin with I would like to thank Prof. Charles Anderson and Prof. Michael Kirby for their directions, suggestions and feedback without which the completion of this thesis would not have been possible. I would also like to thank Prof. Don Rojas for accepting a position on my committee. Additionally I would like to thank Elliot Forney, a graduate student of Prof. Charles Anderson, for his time to discuss ideas with me.

DEDICATION

I would like to dedicate this thesis to my mother Anjali Ghosh and my brother late Nabendu Ghosh.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Machine learning (ML) has become a widespread technology that influences many aspects of modern life. Its applications include weather forecasting, speech recognition, anomaly detection, image classification, content filtering in social network and genetics. In a broader sense, machine learning (ML) algorithms are used for classification and regression. Typically, classification tasks are supervised learning in the sense that the algorithms need ground truth information (class label/category information) to build (or train) the model. Once the training is done on ground truth data, the model can be used to predict data which the model has never seen before. Similarly regression tasks can be done in a supervised manner. In contrast, machine learning algorithms can be trained in an unsupervised manner in the sense that the models do not need any ground truth information about the data. Clustering algorithms are good examples of unsupervised learning where the algorithms exploit some similarity measures to learn from the data. The similarity between data can be defined as distance where different distance metrics such as Euclidean distance, Mahalanobis Distance, angle between points, angle between subspaces and geodesic distance can be used.

In machine learning there are algorithms which support only supervised learning. For example, Support Vector Machines [1] is a state of the art algorithm that explicitly needs ground truth information about data. On the other hand Artificial Neural Networks [2] can be used in both supervised and unsupervised learning. Unsupervised learning algorithms are particularly useful when the ground truth (class/category label) information is not available. To apply a machine learning algorithm (supervised or unsupervised), one needs to know a great amount of detail about the data; in another sense a good domain expertise is a key factor to apply any machine learning algorithm. The domain knowledge along with careful engineering plays a critical role to extract the latent features from the raw data. For decades machine learning or pattern analysis has been done by following the aforementioned approach.

Then came the age of Deep Learning [3]. Deep learning derives from the concept of representation learning [4], in which the model automatically discovers representations required to extract the features from the data. At its heart, deep learning utilizes this concept through multiple levels of representation where the representation in any current level is achieved by applying some non-linear functions/modules on the representation of the preceeding level. Research in recent years has shown that this hierarchical learning via multiple layer of representation can produce unprecedented results [5, 6].

Autoencoder neural network is one such model that has the built-in capability to model deep learning architecture. An autoencoder maps each data point (a sample) to itself through multiple layers of hidden representations. This mapping is independent of the ground truth information, i.e. autoencoder does not need any class/category label information, thus it is an unsupervised learning algorithm. Typically training an autoencoder involves error backpropagation [7]. The backpropagation algorithm does not work too well for deep autoencoder which consists of multiple hidden layers. Over the years researchers have observed that the gradient tends to get smaller and smaller as we move backward to update the weights in hidden layers. This phenomenon is termed as *vanishing gradient*. More details can be found in [8, 9]. An approach to alleviating this learning problem comes from Geoff Hinton who introduces a greedy layer-wise pretraining algorithm, called Restricted Boltzmann Machine [10], which initializes the parameters of the network near a good solution followed by a gradient based fine tuning method.

In our research work, we explore the layer wise pre-training method in light of training a deep autoencoder. In conventional stacked autoencoder, each hidden layer is pre-trained individually. Once all the hidden layers are pre-trained, they are stacked together to form an autoencoder which is then fine tuned. In the process of pre-training each hidden layer individually, it can add noise, and this noise addition increases as the number of hidden layers increases. We explain this is in detail in Section 3.1. We propose two variations of pre-training for autoencoders. In our proposed approaches we minimize the addition of noise in the pre-training phase. Thus we think our approach will set the parameters of the network close to a good solution when compared to standard stacked autoencoder. The results of our experiments support our hypothesis.

Dimensionality reduction is a key aspect in Machine Learning. Data analysis usually starts with visualization and this visualization is significantly harder for high dimensional dataset. In practice, we often get datasets where the number of samples is much lower than the number of dimensions of each sample. This motivates the practitioners of Machine Learning to come up with algorithms which can represent high dimensional data in low dimensional space, preferably in 2D/3D space for visualization. There are several state of the art techniques that one can employ to visualize his/her data. The most popular and common technique is the Principal Component Analysis (PCA) [11], which is a linear technique that captures the maximum variance of the data. On the other hand, neural networks can be used in an autoencoder setting that can be viewed as nonlinear PCA [12]. Both of these techniques are unsupervised, i.e., they do not need ground truth information about the data.

In our research we propose a new supervised learning algorithm, centroid-encoding, which is similar to autoencoder neural network. But unlike autoencoder, centroid-encoder exploits the class information to find a good representative of a class. In our algorithm, we choose the centroid as being representative of a class. In autoencoder, each data point is mapped to itself via a nonlinear mapping, whereas in centroid-encoder, each data point is mapped to the centroid of the class in which the data belongs via a nonlinear mapping. As the centroid of each class of a dataset are different in their ambient space, we think encoding centroids via low dimensional space (2D/3D) can capture structured information about the data. We use three different datasets to visualize high dimensional data in low dimensional space (2D/3D) using centroid-encoder. Our results show that centroid-encoder can reveal interesting structured information in low dimensional space while PCA and nonlinear PCA failed at this task. We also discover that centroid-encoder can be used for pre-training step in classification. Encoding centroids via nonlinearity helps the network to learn the hidden features which are conducive to distinguish a sample of one class from a sample of another class. After pre-training, a softmax layer can be used for classification. Our result shows that the classification accuracy after pre-training using centroid-encoder is significantly better than that of autoencoder. We also calculate the classification accuracy after fine tuning the network. We observe that centroid-encoder did better compared to autoencoder and standard neural network when the network size is small.

The layout of the rest of the thesis is as follows. Chapter 2 consists of background information of training an autoencoder. Chapter 3 comprises our proposed algorithms. Chapter 4 contains the details of the datasets we explored. Chapter 5 elaborates the details of experiments and the results. In Chapter 6, we make our conclusion based on the results we have obtained. We also mention some of the prospective future work on our proposed algorithms.

Chapter 2

Background

In this chapter we will review the background of the autoencoder, its mathematical formulation and its training using error backpropagation.

2.1 Autoencoder

To be put simply, an autoencoder is a function which tries to learn an approximation of the identity map. For given data $\{X^1, X^2, X^3, ...\}$, where $X^i \in \mathbb{R}^n$, it tries to learn a function $f_{(W)}$ such that $f_{(W)}(X^i) \approx X^i$ where W and b are the model parameters.
One can use the artificial neural network with error backpropagation to learn an autoencoder. In this set up the targets are set to be equal to the inputs as shown in Figure 2.1. The input is a five dimensional vector $\{x_1..x_5\}$ which is passed through three hidden layers. In the first hidden

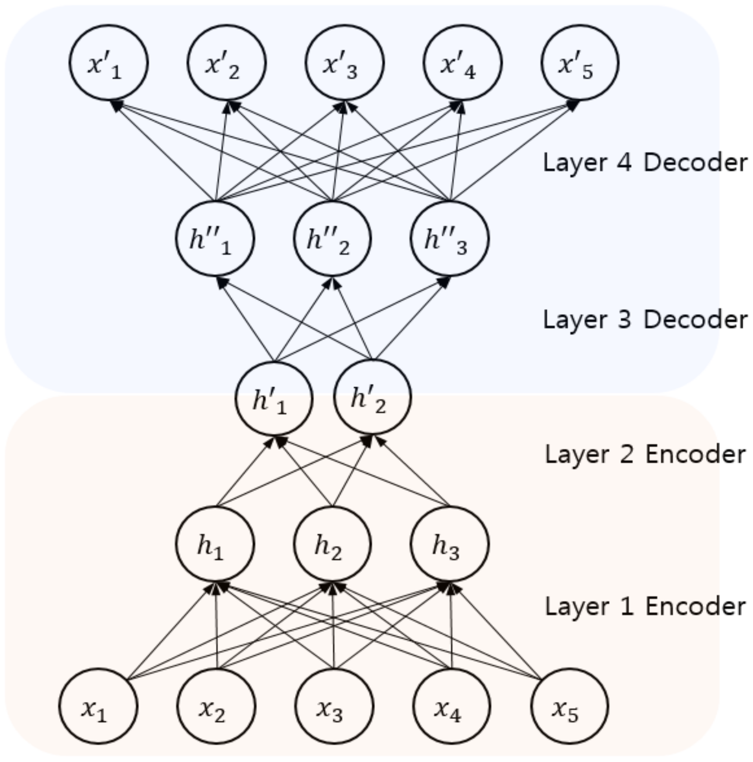


Figure 2.1: An example of an autoencoder with three hidden layers.

layer we have three hidden units $\{h_1..h_3\}$ and in the second hidden layer we have two hidden units $\{h'_1, h'_2\}$. These two hidden layers can be thought as encoding layers. The third hidden layer $\{h''_1..h''_3\}$ takes the two dimensional encoded data, decodes it and then pass it on to the output layer which decodes the data again to produce the output $\{x'_1..x'_5\}$. The number of nodes in the input layer needs to be same as in the output layer. At this point the error is calculated from the original input $\{x_1..x_5\}$ and reconstructed input $\{x'_1..x'_5\}$. Then this error is back propagated through the network using an algorithm called error backpropagation. So an autoencoder reconstructs a data point via some hidden representation.

For an autoencoder neural network the function $f_{(W)}$ is a nonlinear one. This type of neural network is generally used on unsupervised data where the labels are not available. Figure 2.1 represents an autoencoder with three hidden layers, but there can be numerous hidden layers especially in a deep autoencoder. A deep autoencoder can be thought of as a multi-layer artificial neural network. So the training procedure of autoencoder is the same as a multi-layer neural network. Typically the training is done by an algorithm called error backpropagation [7] which we briefly discuss here.

2.2 Error Backpropagation

In a feed-forward neural network each hidden unit computes a weighted sum of its input in the form

$$a_j^l = \sum_i w_{j,i}^l z_i^{l-1} \tag{2.1}$$

where z_i^{l-1} is the activation i^{th} hidden unit of $(l-1)^{th}$ layer or input layer, which is connected via wight $w_{j,i}^l$ to the j^{th} unit of l^{th} layer. For simplicity, the bias in each unit is considered to be an element of w^l and the corresponding input in z^{l-1} is a constant 1. A non-linear activation function (g) is applied to give the activation or output (z_j^l) of unit j in layer l in the following form:

$$z_j^l = g(a_j^l) \tag{2.2}$$

We seek to adjust the parameters of the network by minimizing an appropriate error function in the following form:

$$E = \sum_{n=1}^{N} E^n$$

In the above equation we are calculating the sum of all errors defined on each pattern n. Most of the error functions in practice takes this form. Examples are the mean squared error (MSE) and cross entropy error (CE). We are also assuming that the error E^n can be expressed as a differentiable function of the output variables of the network. Now we need to calculate the derivative of E^n with respect to some network parameter $w^l_{j,i}$. We note that E^n depends on the $w^l_{j,i}$ only via summed input a^l_j of $j\,th$ unit in layer l . Applying the chain rule of derivatives we get:

$$\frac{\partial E^n}{\partial w^l_{j,i}} = \frac{\partial E^n}{\partial a^l_j}\frac{\partial a^l_j}{\partial w^l_{j,i}} \tag{2.3}$$

Define :

$$\delta^l_j \equiv \frac{\partial E^n}{\partial a^l_j} \tag{2.4}$$

Using equation 2.1 we can write:

$$\frac{\partial a^l_j}{\partial w^l_{j,i}} = z^{l-1}_i \tag{2.5}$$

Substituting equation 2.4 and equation 2.5 into equation 2.3, we get:

$$\frac{\partial E^n}{\partial w^l_{j,i}} = \delta^l_j z^{l-1}_i$$

So to calculate the derivatives we need to multiply the δ of unit j of layer l to the output of unit i in layer $l-1$. The calculation of δ^o_k for the output layer is as follows:

$$\delta^o_k \equiv \frac{\partial E^n}{\partial a^o_k} = g^{'}(a^o_k)\frac{\partial E^n}{\partial y_k}$$

where we used equation 2.2 with z_k denoted by y_k. To calculate the δs for hidden units in a hidden layer l we can use the equation 2.4 as follows:

$$\delta_j^l \equiv \frac{\partial E^n}{\partial a_j^l} = \sum_k \frac{\partial E^n}{\partial a_k^{l+1}} \frac{\partial a_k^{l+1}}{\partial a_j^l}$$

where the sum runs over all the units k in layer $l+1$ to which unit j of layer l sends connections. Substituting the definition of δ_j^l from equation 2.4 and using equations 2.1 and 2.2 we can write:

$$\delta_j^l = g^{'}(a_j^l) \sum_k w_{kj} \delta_k^{l+1} \tag{2.6}$$

Equation 2.6 tells us that the values of δ of a hidden unit can be calculated by propagating δs backwards from units higher up in the network. This is the key to the error backpropagation algorithm.

2.3 Pre-training Autoencoder

Updating the weights in the network by propagating the error backwards works well for a shallow network where the number of hidden layers are few. But for a deep network where the number of hidden layers is quite large, the learning saturates due to a phenomenon called *vanishing gradient* [8,9,13,14]. In a deep architecture the gradient tends to get smaller as we move backward from the output layer. This means the weights associated with the hidden layers close to the output layer will be updated by large values, whereas the weights associated with the hidden layers close to input layer will be updated by small values. This effectively slows down the overall learning process in a deep neural network.

Around 2005 Geoff Hinton came up with the idea of greedy layer wise pre-training which initializes the parameters in the network close to a good solutions. They used a probabilistic approach called Restricted Boltzmann Machine (RBM) to initialize the parameters in the network followed by a gradient based fine tuning [10, 15]. Due to this breakthrough people started to use deep neural network architecture which started to produce good results in different applications.

Restricted Boltzmann Machine is a probabilistic model to initialize the parameters in the network. The idea of pre-training was soon extended with continuous values [16, 17].

To speed up the training of deep autoencoder a greedy layer wise pre-training approach is useful. The main objective of this pre-training is to initialize the weights of a deep autoencoder. A stacked autoencoder is a multi-layer artificial neural network which is built by stacking multiple layers of autoenoder where the output of one autoencoder is fed as the input to the successive layers as shown in Figure 2.2. Once the pre-training is done all the layers of autoencoders will be stacked

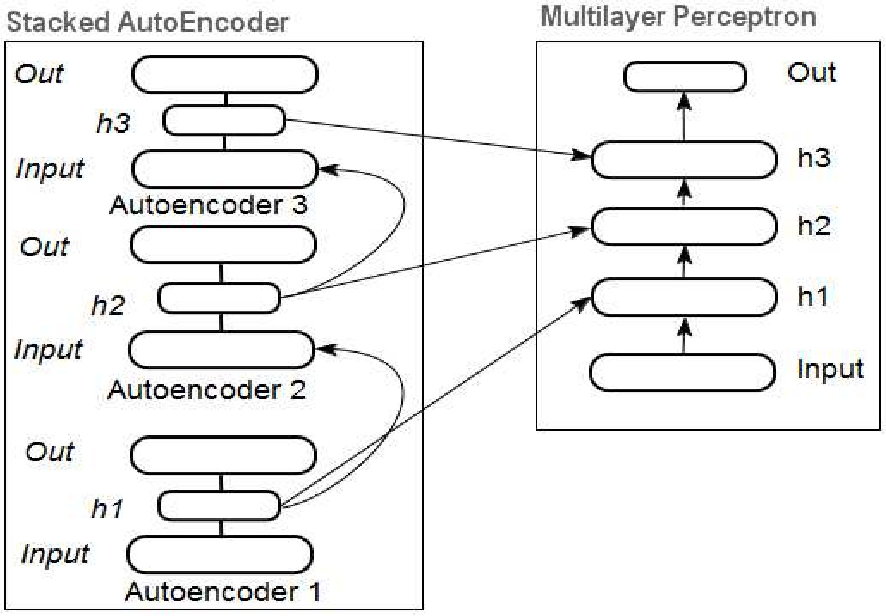


Figure 2.2: An example of a stacked autoencoder using three hidden layers. On the left hand side we have three autoencoders trained separately. The first autoencoder maps the original input to itself via the hidden layer 'h1'. The second autoencoder takes the output of the hidden layer 'h1' from the first autoencoder and then maps the data to itself via the hidden layer 'h2'. Similarly the third autoencoder maps the output of hidden layer 'h2' to itself via the hidden layer 'h3'. This process is known as pre-training. After the pre-training, all the three autoencoders are stacked together to form a multiple layer perceptron (multi-layer ANN) to do a further refinement of the network parameters.

together to form the deep autoencoder and then the post training (backpropagation) will be done on the deep network. The post training of the deep autoencoder will be faster as we already have a good initialization of weights from the stacked autoencoder.

Chapter 3

Methods

Before describing our proposed methods we point out an issue with stacked autoencoder.

3.1 Issue of Pre-training with Stacked Autoencoder

The autoencoder neural network tries to approximate the identity map such that for a given input X, where $X \in \mathbb{R}^n$, $f_{(W)}(X) \approx X$ where $f_{(W)}$ is a non-linear function. Therefore the reconstruction of the input X by the non-linear function $f_{(W)}$ will have an error in it. So it can be written

$$f_{(W,b)}(X) = X + \epsilon_{error} \tag{3.1}$$

The reconstruction error (ϵ_{error}) will be added by each layer while doing the pre-training by stacked autoencoder. Moreover in each successive layer the autoencoder will reconstruct the input which has the error ϵ_{error} in it. This way the initial weights learned by stacked autoencoder will be misleading and this may cause reaching the local minimum faster in the post training.

To tackle this potential issue with stacked autoencoder, the following pre-training approach is being proposed.

3.2 Proposed Pre-training

In the proposed approach, pre-training will be done by adding the new hidden layers while reconstructing the original input. To pre-train each layer error backpropagation will be used with some non-linear activation function. The main difference of this approach with the standard stacked autoencoder is that in the standard stacked autoencoder the weights in each layer are learned by reconstructing the input which is the output of previous layer, but in the proposed approach the weights of the newly added layer will be learned by reconstructing the original input. Thus the reconstruction error ϵ_{error} of each layer will not be propagated.

Two different variations are being proposed while updating the weights of the newly added hidden layer. In the first approach weights are only updated to the newly added hidden layer using error backpropagation. Weights of the previously added hidden layers will be frozen during backpropagation. We will call this approach ***Stacked Autoencoder with Layer Freeze*** or simply ***SAE with Layer Freeze***. In the second approach, weights of the newly added hidden layer will be updated along with the weights of the previously added layers. We will call this approach ***Stacked Autoencoder without Layer Freeze*** or simply ***SAE without Layer Freeze***. In Figure 3.1 we describe our pre-training approaches. We want to build a stacked autoencoder with n hidden layers. The first step is similar to standard stacked autoencoder. In this step we add the first hidden

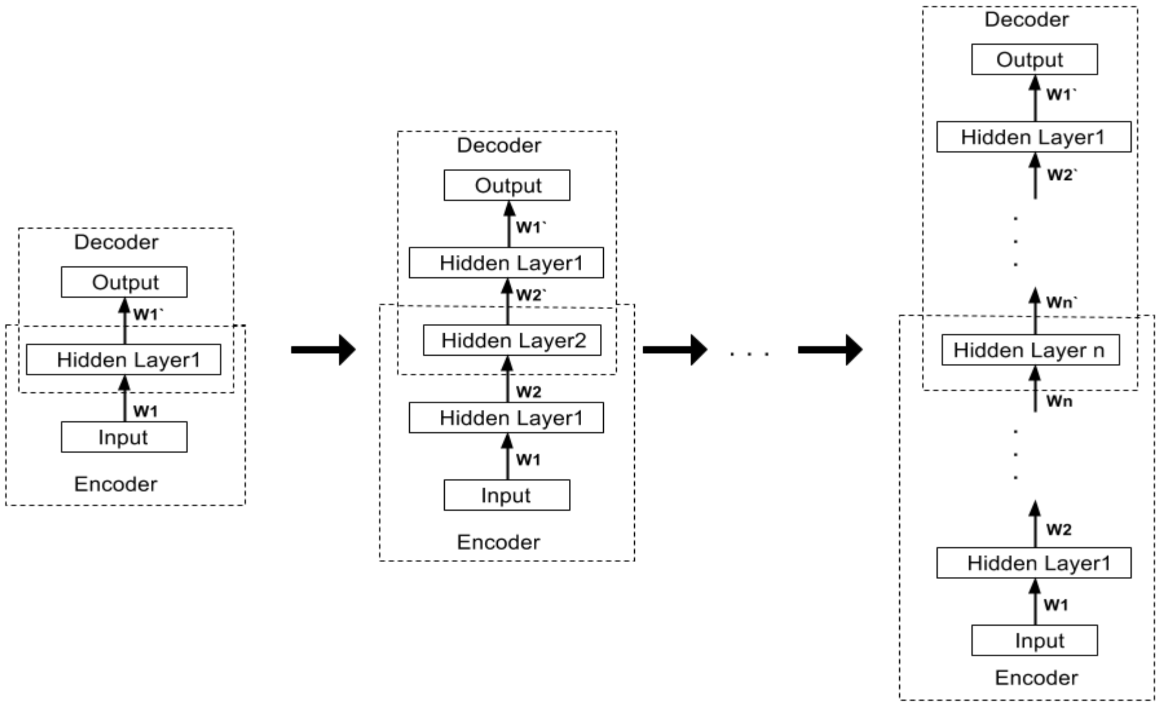


Figure 3.1: Pre-training a deep autoencoder by the proposed approaches. In the first step, an autoencoder with the first hidden layer is pre-trained (left diagram). The pre-trained weights are $w1, w1'$. In the next steps, a new hidden layer is added by extending the network architecture of the autoencoder. In the *SAE with Layer Freeze* approach, weights $(w2, w2')$ associated with the new hidden layer are updated while keeping the other weights $(w1, w1')$ fixed. In contrast in the *SAE without Layer Freeze* approach, weights $(w2, w2')$ associated with the new hidden layer are updated along with the other weights $(w1, w1')$. This process is repeated to add more hidden layers.

layer and map each data point to itself as shown in the left hand side of the Figure3.1. After this step we get weight matrix $w1$ which connects the original input to the first hidden layer and $w1'$ which connects the hidden layer to the output layer. In the next step we want to bring the next hidden layer. Unlike the standard stacked autoencoder, we extend the network by adding the second hidden layer. Now we have two new weight matrix $w2, w2'$. These two weight matrices connect the second hidden layer with the first one as shown in the middle diagram of Figure 3.1. Now we need to train the extended network. Here we take two separate approaches. In the *SAE with Layer Freeze* approach, we only update the weights $w2, w2'$ and we keep the $w1, w1'$ frozen, hence we call this approach *SAE with Layer Freeze*. So we only want to train the newly added hidden layer by allowing the corresponding weight matrices to change. In other words, we do not want to change the learning of first hidden layer and we accomplish this by freezing the weight matrices $w1, w1'$. In contrast, we do not freeze the weights of the first hidden layer in *SAE without Layer Freeze* approach. In this approach we train the extended network by changing the weight matrices $w1, w1'$, associated with first hidden layer, and $w2, w2'$, associated with second hidden layer. As we can see in *SAE without Layer Freeze* approach, the first hidden layer will loose some of its learning which was learnt in first step. We need to repeat this process n times to pre-train our network.

Both of our pre-training approaches are different from standard stacked autoencoder except the first step. In standard stacked autoencoder, we pre-train n different autoencoders separately, whereas in both of our approaches we extend an autoencoder by adding new hidden layers. In standard stacked autoencoder, we use the output of a hidden layer to pre-train the next autoencoder, but in both of our approaches we use the original input to pre-train the newly added hidden layers.

3.3 Supervised Nonlinear Centroid-Encoder

Now we propose a new supervised learning algorithm. A modification to the autoencoder neural network is being proposed to incorporate supervised learning. Like most of the supervised learning, the label information will be used in training the model. As described above, a traditional autoencoder maps each input $x^{(i)}$ to itself without considering the label information of each data point. In the proposed algorithm, the supervised autoencoder maps each data point of a class to a representative of that class while passing the data through two or three dimensions for visualization. We take the representative C_j of $Class_j$ to be the centroid of the class:

$$C_j = \frac{1}{|Class_j|} \sum_{x^i \in Class_j} x^i$$

The rest of the training is similar to autoencoder: hence, this method can be viewed as a *centroid-encoder*. The steps of this new method is described in Algorithm 1

```
Algorithm 1: Supervised Nonlinear Centroid-encoder.
  Input Data: Data $x^{(\mu)} \in \mathbb{R}^m, \mu = 1, \dots, P$.
  Output Data: Neural network output layer $\tilde{x}^{(\mu)} \in \mathbb{R}^m, \mu = 1, \dots, P$.
  Non-linear function: $f : \mathbb{R}^m \to \mathbb{R}^m$
  Result: Nonlinear embedding of data in $d$ dimensions (user selected).
  Initialization: Set $d$ and number of layers and nodes per layer. Set neural network
                  architecture including error tolerance $\tau$. Calculate the centroid
                  $C_j = \frac{1}{|Class_j|} \sum_{x^i \in Class_j} x^{(i)}$ for each class from input data.
  Define: Centroid-encoder output error E = $\sum_j \sum_{x^i \in Class_j} \|f(x^i) - C_j\|_2$
1 while centroid-encoder output error $> \tau$ do
2     Train centroid-encoder such that $E \leq \tau$ where $x^\mu \in Class_j$.
3 end
```

Chapter 4

Data Description

In this chapter we present the descriptions of different datasets which are used in different experiments.

4.1 MNIST dataset

The MNIST (Mixed/Modified National Institute of Standards and Technology) database is a collection of images of handwritten digits (0 .. 9). This is a standard dataset to train and test machine learning algorithm. This dataset is originally created from larger sets available in NIST (National Institute of Standards and Technology) dataset. There are a total of 60,000 training samples and 10,000 test samples in this dataset. Samples are the black and white image of the digits 0 to 9 and these images are normalized to fit into a 28 x 28 bounding box. Therefore each image in this dataset is represented by a collection of 28 x 28 = 784 pixels where the values in each pixel represents the intensity in gray scale.

The training set contains ten different classes (digits) of data. The distribution of digits from different classes is almost equal. In the training set we have 5923 digits of class 0, 6742 digits of class 1, 5958 digits of class 2, 6131 digits of class 3, 5842 digits of class 4, 5421 digits of class 5, 5918 digits of class 6, 6265 digits of class 7, 5851 digits of class 8 and 5949 digits of class 9. The distribution of ten different classes in test data is almost equal. In the test set we have 980 digits of class 0, 1135 digits of class 1, 1032 digits of class 2, 1010 digits of class 3, 982 digits of class 4, 892 digits of class 5, 958 digits of class 6, 1028 digits of class 7, 974 digits of class 8 and 1009 digits of class 9.

The MNIST dataset is originally created by taking handwritten samples from NIST's Special Dataset 3 and Special Dataset 1. The SD-1 contains 58,527 digits which are collected from 500 different writers and the data is collected around 1992. NIST collected this data from high school students. In contrast the SD-3 data was collected from the employees of Census Bureau. NIST originally released SD-3 data as training set and SD-1 data as test set. The samples from SD-3 set is much clearer that the samples from SD-1. One reason could be the difference of participants in those two datasets. Therefore the performance (classification rate) of any machine learning algorithm will depend on the choice of training and test set. Hence it is required to create a new dataset by mixing the samples from SD-1 and SD-3 set.

The MNIST training set is comprised of 30,000 samples from SD-1 and 30,000 samples from SD-3. The test set is composed of 5,000 samples from SD-1 and 5,000 samples from SD-3. The writers of training and test set are disjoint. The original pixel values of MNIST data is in the range 0..255. These values were divided by 255 to fit in the range 0..1. This is a standard practice in Machine Learning to work with image dataset. Samples of MNIST data are shown in Figure 4.1. Over the years many machine learning methods have been applied on this dataset. In many cases the training set, which has 60,000 samples, has been distorted artificially (random combinations of shifts, scaling, skewing etc) to improve the performance. In this thesis we do not apply any distortion of the on the training data.

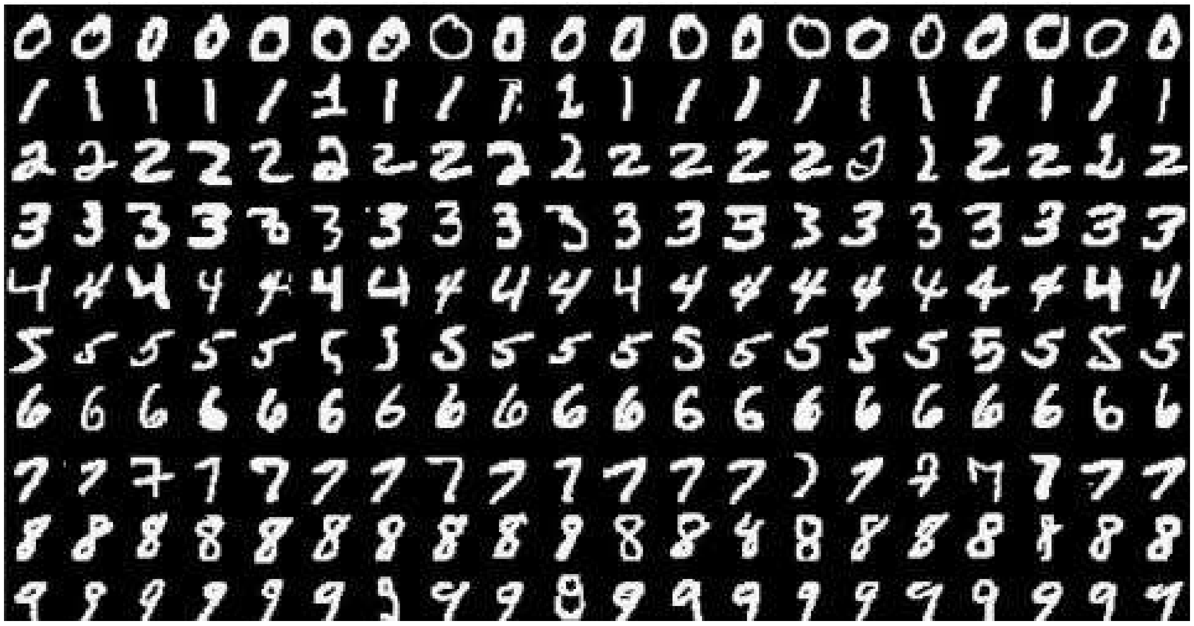


Figure 4.1: Samples of MNIST dataset

4.2 Microarray gene expression data of crab-eating macaque

Technological advancements over the last decade have shown a tremendous improvement of acquisition of biological datasets. Among the many techniques, DNA microarrays has emerged as a new technology that allows to measure thousands of genes simultaneously. This technology, known as DNA microarrays or DNA chips, helps to measure the mRNA levels in particular cells or tissues for many genes at once. Essentially, a microarray is made up with multiple DNA spots on a solid surface. Each of these spot is capable of estimating expression level of a gene. Generally, a microarray is composed with thousands of DNA spots, hence by a microarray experiment one can measure multiple genes at once. This technology, emerged in late 90s, helps to measure gene expression at a higher temporal resolution, enables us to make discoveries related to human and animal diseases. In our research we have analyzed such a microarray dataset which contains the host immune response (gene expressions), taken from the blood samples of crab eating macaques, in response to infection to Ebola Virus.

The second dataset that has been explored is provided by Rubins et al. [18]. In this study fifteen macaques are used. Peripheral blood mononuclear cells (PBMCs) are collected from crab eating macaques infected with ZEBOV (Zaire ebolavirus). Subjects are killed on days 1, 2, 3, 4, 5 and 6 after infection to collect the peripheral blood samples which we consider as infected samples. Blood samples are also taken on days 1, 4, 6 before infection which we consider as baseline (pre-bleed samples). After collecting the PBMCs, microarray analysis was done to get the gene expression and then the gene expressions are assembled in pathways. For this study gene expression is measured from 18,000 unique genes. A pathway is a collection of genes that work together towards a common goal. There are 1469 genetic pathways in the Macaque dataset. We considered the T Cell Receptor Signaling pathway for our study as this is a key pathway for early warning. This pathway has 96 genes, so the ambient space is $\mathbb{R}^{96}$. We studied the pre-infection data (pre-bleed samples) and post-infection data where we considered only three time points (day 1 to day 3) in post-infection. There are thirty control samples in total. Beside that we have seven samples from day one, twelve samples from day two and seven samples from day three. By sample we mean expression of a genetic pathway of a subject at a particular time point. So from a subject we may have multiple samples. For our study we used 80% of total data in training and the remaining 20% in testing. More information on data can be found in [18].

4.3 EEG Mental Task dataset

EEG (Electroencephalogram) is the electrical activity of brain. In the year of 1928 the German psychiatrist Hans Berger was the first to record EEG signal. 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. EEG has been used heavily to build BCI (Brain Computer Interface) systems.

Collection of EEG data is done in two different ways: invasive and non-invasive. In invasive EEG recording, electrodes are implanted inside the brain of a subject. This technique typically requires surgery and it is costlier. On the other hand, non-invasive EEG recording is obtained through electrodes attached to the scalp surface by an EEG cap. This technique is cheaper than the other one. Depending upon the nature of the study, one has to decide the type of EEG recording. Although the invasive technique is costlier, this allows the collection of EEG signal with considerably higher amplitude and spatial resolution. Because this technique allows the electrodes to be placed much closer to the brain than the scalp electrodes, the EEG recording will be more clearer and less noisy. On the other hand, the non-invasive EEG technique allows us to collect data with less cost. Because non-invasive EEG systems are portable, it allows us to collect data from different places which gives one a better reach to a variety of subjects. In non-invasive technique the electrodes are attached to the scalp surface via an EEG cap. The position of electrodes is determined by the type of study. Figure 4.2 shows the positions of electrodes / channels in an EEG cap. In this research we have collected EEG signals from the electrodes F3, F4, C3, C4, P3, P4, O1 and O2.

In this thesis EEG signals for different mental tasks have been analyzed. The EEG data has been recorded in the BCI lab of Colorado State University. The EEG signal was captured by g.Tec system which has the 8 electrode on the cap. The sampling frequency is 256 Hz. We have four

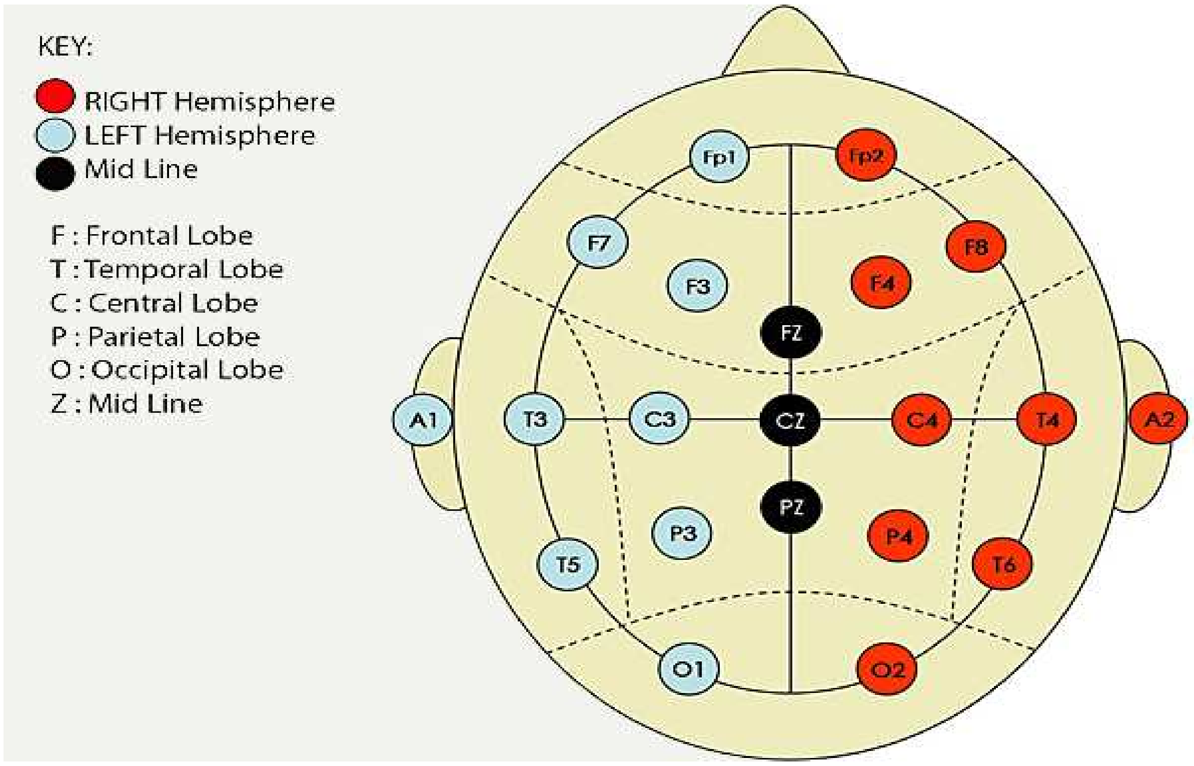


Figure 4.2: EEG cap showing position of different electrodes..

different mental tasks: visual task, counting task, fist movement task and singing task. To collect mental task data subjects are asked to imagine four different mental task one after another with a little gap in between. In the visual task the participant has been asked to imagine a rubric cube, in the counting task the participant has been asked to count from 100 to 1 mentally, in fist movement task the participant has been asked to imagine moving his/her fist and in the singing mental task the participant has been asked to remember a favorite song in his/her mind. It is noteworthy to mention that in none these mental tasks the participants do the task physically, rather they have just imagined the task being performed in their mind. For each mental task, we have collected 10 seconds of continuous EEG data and we have collected the data in five different trials. As we have four different mental tasks so we have collected 40 seconds of EEG data in each trial. So in five trials, we have collected 200 seconds of EEG data. To be more precise we have collected 50 seconds of EEG data for each mental task. Sample of EEG signal is shown in Figure 4.3.

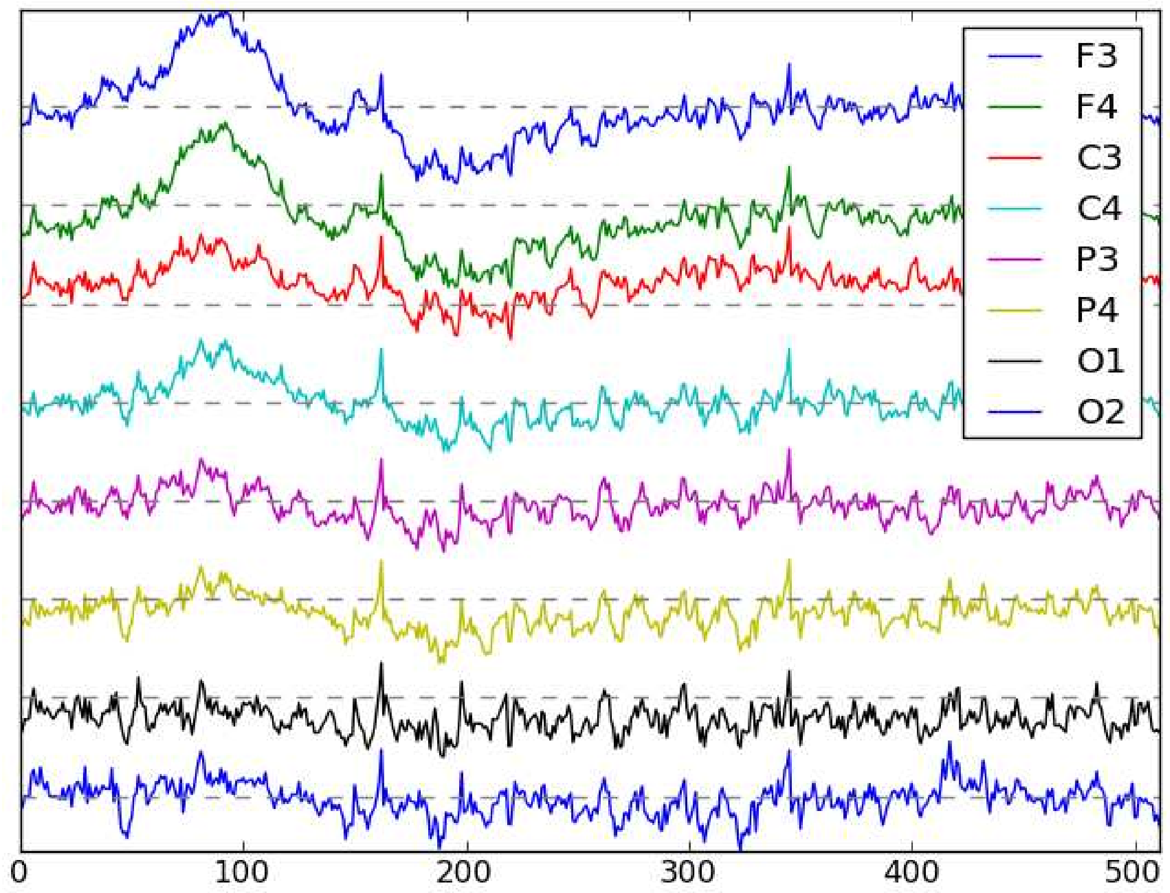


Figure 4.3: EEG data recorded from scalp electrodes at eight standard positions labeled F3, F4, C3, C4, P3, P4, O1 and O2.

Chapter 5

Results

In this chapter we will discuss the details of our experiments and the corresponding results. Our experiments fall into three categories. In the first category of experiments we will focus on the proposed pre-training. We will demonstrate the performance of the proposed pre-training with respect to the standard practice. The experiments in the second category explore the effectiveness of centroid-encoder for low dimensional data visualization. We will compare our low dimensional visualization with PCA and autoencoder. In the third category we will describe classification using centroid-encoder, focusing on how centroid-encoder can be used as a pre-training step. Again we compare our result with autoencoder and standard neural network.

5.1 Experiments with proposed pre-training

Now we describe the experimental setup and associated results on the proposed pre-training. We use the MNIST hand written digit dataset for these experiments.

5.1.1 Experiments

The objective of this experiment is to compare the performance of our proposed pre-training algorithms with stacked autoencoder. To make the comparison fair, we keep the network architecture fixed in all three algorithms. Our network architecture is comprised of four hidden layers where the first layer has 1000 hidden units, the second layer has 500 hidden units, the third layer has 250 hidden units and the fourth layer has 30 hidden units. This structure will be denoted as [1000 -> 500 -> 250 -> 30].

In stacked autoencoder, we pre-train four different autoencoders. First we train 784 -> [1000] -> 784 where the input layer has 784 units, the hidden layer has 1000 units and the output layer has 784 units. We then take the output of the hidden layer and pass it to the next autoencoder 1000 -> [500] -> 1000. This way we train the next two autoencoders: 500 -> [250] -> 500 and 250 ->

[30] -> 250. After the pre-training, we stack together all the four autoencoders to form the deep autoencoder (784 -> [1000 -> 500 -> 250 -> 30 -> 250 -> 500 -> 1000] -> 784) which is then post-trained for fine tuning. Figure 5.1 illustrates this process.

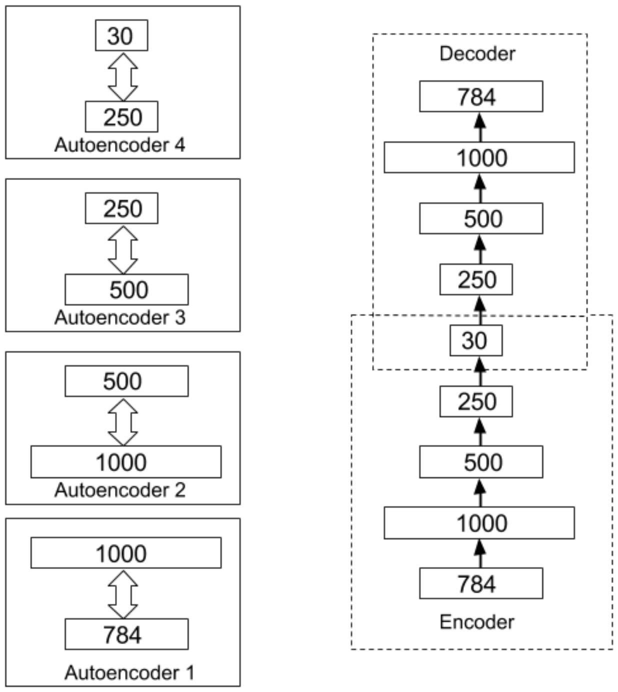


Figure 5.1: Layer wise pre-training with standard stacked autoencoder. Four different autoencoders are pre-trained as shown in the left diagram. After the pre-training, all the four autoencoders are stacked together to form a deep autoencoder architecture as shown in right diagram.

We pre-train the autoencoder with our proposed methods in four steps as shown in Figure 5.2. The first step is essentially the same with the stacked autoencoder when we pre-train the autoencoder 784 -> [1000] -> 784. In the second step we add the next hidden layer which has 500 hidden units and pre-train the autoencoder 784 -> [1000 -> 500 -> 1000] -> 784 by reconstructing the input using error backpropagation. When we pre-train with layer freeze method, we do not update the weights associated with the hidden layer which we have added in step one. We only update the weights associated with the newly added hidden layer. Therefore in the SAE with Layer Freeze method, we do not change the weights connecting layers 784 -> 1000 and 1000 ->

784; we only update the weights connecting layers 1000 -> 500 and 500 -> 1000. But in SAE without Layer Freeze method we do not freeze weights associated with any layer, we update all the weights. Therefore in SAE without Layer Freeze we update the weights connecting all the layers: 784 -> 1000, 1000 -> 500, 500 -> 1000 and 1000 -> 784.

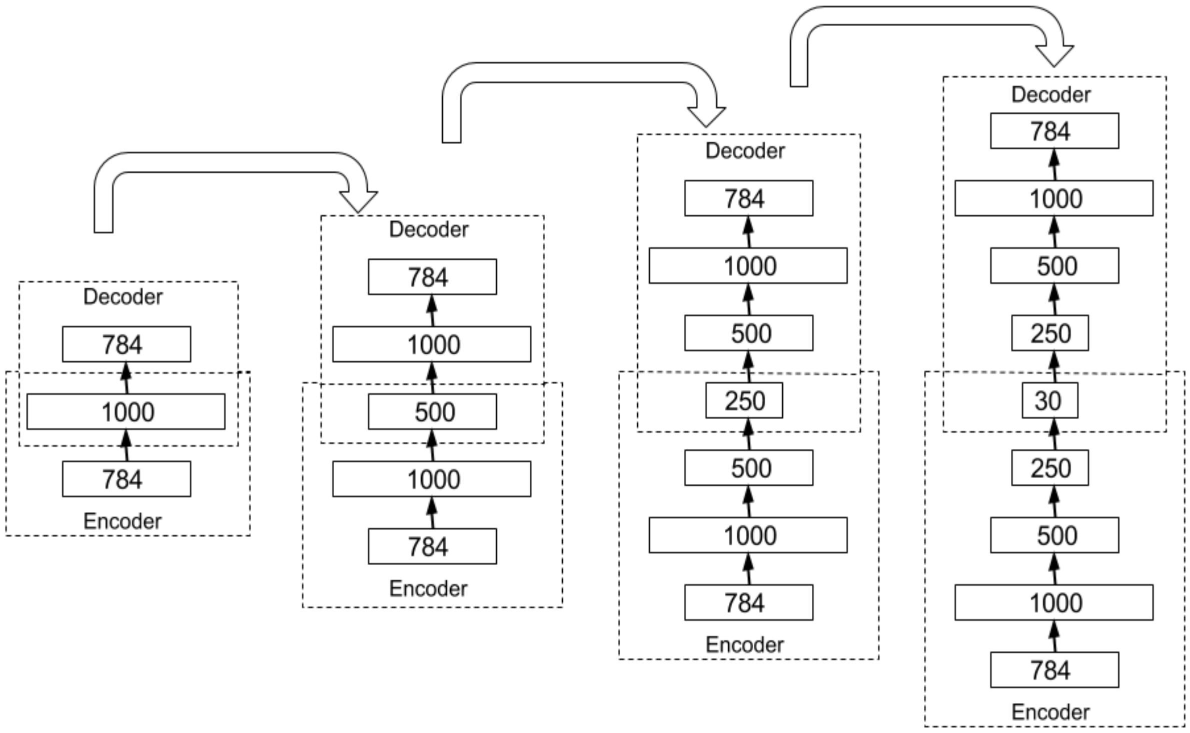


Figure 5.2: Layer wise pre-training with proposed stacked autoencoder. In the first step an autoencoder with 1000 hidden units is trained. The next hidden layer with 500 units, is added by extending the network architecture of the autoencoder. In the SAE with Layer Freeze method, weights associated with the newly added hidden layer (second one) are updated, whereas in the SAE without Layer Freeze method all the weights in the network are updated together. Similarly this process is repeated to add the next two hidden layers as shown in the figure.

In all of the hidden units we use hyperbolic tangent (tanh) as the activation function which is defined in Equation 5.1. This is a commonly used activation function which keeps the output value between -1 to +1.

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{5.1}$$

For error backpropagation we use the Scaled Conjugate Gradient Descent (SCG) algorithm [19] in both pre-training and post-training.

We use mean square error (MSE) as the error function which is described in Equation 5.2.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\bar{Y_i} - Y_i)^2 \tag{5.2}$$

where n is the total number of samples, Y_i is the i^{th} observed sample and the $\bar{Y_i}$ is the i^{th} predicted sample. MSE is calculated on the test data after the post training for each methods.

In the pre-training, we determine the optimum number of SCG iterations by a statistical measure called *Coefficient of Variation (C_v)* which is defined as below:

$$C_v = \frac{\sigma}{\mu}$$

where σ represents the standard deviation and μ represents the mean. For a given list of numbers C_v tells us the amount of variability relative to the mean. During pre-training of each hidden layer, we calculate MSE on the training data of five consecutive iterations to calculate C_v. We stop the pre-training when this value reaches a value of of 0.1.

We use early stopping to find out the optimal number of training iterations during post-training. We calculate validation error (MSE) of twenty consecutive iterations. If the mean of the last ten validation error is greater than the mean of first ten then we stop the post training. Among the twenty consecutive iterations we pick the one which gives minimum validation error to select best parameters (weights and biases).

We use MNIST hand written digit dataset for this experiment because this dataset has been used very extensively in the research of deep learning. This dataset has 60,000 training samples and 10,000 test samples. As it is quite time consuming to train a deep autoencoder on this dataset, we divide the data into small subsets and run our experiments on these small subsets. From the original training dataset we form ten disjoint subsets which we refer as **data chunks**. Each data chunk contains 6000 samples drawn randomly without replacement from the original MNIST training set. We also make sure to keep equal number of samples from each class in a data chunk. As MNIST dataset has ten different classes, we pick 600 random samples for each class. We use half of the data in a data chunk as a training set and the rest as a validation set which we use for early-stopping as a regularization step in post-training. Both the training and the validation set contain equal number of samples (300) from each class. In the first set of experiments, we run three different pre-training algorithms ten times on a single data chunk and report the average error (MSE) on the test set. In the second set of experiments, we run all the three pre-training algorithm on all ten different data chunks only one time and we calculate the average error (MSE) on the test set.

5.1.2 Results

In the first set of experiments we run three algorithms on the same data. By data we mean a data chunk which is defined in the previous section. We calculate the mean squared error (MSE) after pre-training and post-training on training data as well as on test data. Table 5.1 and Table 5.2 give the details of the error statistics of our proposed pre-training algorithms, whereas Table 5.3 gives the error statistics of the standard stacked autoencoder algorithm. We see that after pre-training the mean MSE on test data is the lowest for SAE with Layer Freeze algorithm, whereas the mean MSE on test data after pre-training is the highest for standard stacked autoencoder. If we consider the post-training error we observe that SAE with Layer Freeze has the best result with mean MSE at 16.58, whereas SAE without Layer Freeze and standard SAE algorithms performed poorly. It is noteworthy to mention that mean MSE of these two methods are very similar.

Table 5.1: Error statistics for SAE with Layer Freeze over multiple runs

	Pre Training		Post Training	
Run	Training Data	Test Data	Training Data	Test Data
1	24.07	25.12	8.51	16.39
2	24.89	25.90	7.72	16.19
3	24.71	25.76	7.93	16.08
4	24.99	26.02	17.87	20.29
5	25.03	25.89	8.24	15.93
6	25.64	26.44	8.34	16.33
7	25.37	26.30	7.96	16.17
8	24.70	25.77	8.39	16.31
9	26.37	27.15	8.08	15.98
10	25.27	26.21	8.70	16.17
Mean	25.10	26.06	9.17	16.58

Table 5.2: Error statistics for SAE without Layer Freeze over multiple runs

	Pre Training		Post Training	
Run	Training Data	Test Data	Training Data	Test Data
1	27.04	27.99	18.07	19.80
2	26.36	27.19	18.43	20.21
3	43.49	43.52	18.12	19.82
4	29.31	29.90	18.02	19.88
5	43.70	43.93	18.48	19.93
6	28.84	29.57	17.97	19.72
7	29.52	29.90	18.17	19.61
8	31.59	32.31	18.14	19.82
9	29.10	29.84	18.07	19.79
10	29.45	30.31	18.32	20.14
Mean	31.84	32.45	18.18	19.87

Now we discuss the performance of these algorithms on multiple data chunks. In this set of experiments we have run all three algorithms on each of those ten data chunks only once and then we have calculated the average MSE. Table 5.4 lists the MSE after pre-training and post-training on ten different data chunks for the algorithm SAE with Layer Freeze. Table 5.5 and 5.6 show the similar error statistics for the algorithm SAE without Layer Freeze and Standard SAE respectively. The MSE on each different data chunk is considerably lower for the algorithm SAE with Layer Freeze compared to the other two. This observation is consistent in both pre-training as well as

Table 5.3: Error statistics for Standard SAE over multiple runs

	Pre Training		Post Training	
Run	Training Data	Test Data	Training Data	Test Data
1	39.44	39.69	17.82	20.15
2	38.67	39.23	18.15	20.27
3	38.80	39.12	8.73	17.05
4	39.30	39.43	18.43	20.53
5	40.86	41.01	18.37	20.52
6	40.41	40.62	7.94	16.55
7	39.50	39.91	18.24	20.32
8	39.24	39.76	18.25	20.23
9	40.54	40.63	18.00	20.15
10	40.74	41.17	18.0114	20.1476
Mean	39.75	40.06	16.19	19.59

post-training. The mean MSE after post training on test data is 16.70 for the algorithm SAE with Layer Freeze. This number is considerably lower than 19.66 and 19.53 which are the corresponding mean MSE for algorithms SAE without Layer Freeze and Standard SAE. In terms of error after post training, algorithms SAE without Layer Freeze and Standard SAE have a similarly result.

Table 5.4: Error statistics for SAE with Layer Freeze over multiple data chunks

	Pre Training		Post Training	
Data Chunk	Training Data	Test Data	Training Data	Test Data
1	25.42	26.34	10.94	17.25
2	25.55	26.29	8.69	16.39
3	26.22	26.77	11.00	17.34
4	25.71	26.15	9.16	16.45
5	25.42	25.86	10.20	16.41
6	26.01	26.75	9.05	16.52
7	25.67	26.42	10.19	16.75
8	25.49	25.86	12.54	17.72
9	25.61	26.35	8.36	16.01
10	25.86	26.09	8.89	16.19
Mean	25.70	26.29	9.90	16.70

Table 5.5: Error statistics for SAE without Layer Freeze over multiple data chunks

	Pre Training		Post Training	
Data Chunk	Training Data	Test Data	Training Data	Test Data
1	33.23	33.65	18.30	19.80
2	32.82	33.36	18.08	19.71
3	32.72	32.63	18.13	19.66
4	34.45	34.63	18.49	19.56
5	31.18	31.39	18.19	19.63
6	36.39	36.98	18.27	19.71
7	32.13	32.83	18.14	19.72
8	35.67	35.55	18.17	19.50
9	33.00	33.58	18.42	19.70
10	35.69	35.91	18.58	19.63
Mean	33.73	34.05	18.28	19.66

Table 5.6: Error statistics for Standard SAE over multiple data chunks

	Pre Training		Post Training	
Data Chunk	Training Data	Test Data	Training Data	Test Data
1	39.70	40.12	17.92	20.22
2	40.24	40.36	18.27	20.28
3	40.11	39.56	17.31	19.84
4	39.83	40.15	17.43	19.66
5	40.11	39.43	17.43	19.84
6	39.56	40.00	13.65	18.39
7	39.55	40.34	17.43	19.78
8	40.64	40.51	15.52	19.00
9	39.83	40.44	17.40	19.81
10	39.76	39.72	13.76	18.47
Mean	39.93	40.06	16.61	19.53

We have also plotted the change of error (MSE) over iterations during the post training. Figure 5.3 compares the decrease in error for the three methods. In this experiment we have used the entire MNIST training dataset and we have run all the algorithms for 2500 iterations. At the early phase of the post training, the decrease of error for SAE with Layer Freeze method is more compared to the other two. At the later phase of training the proposed methods (SAE with Layer Freeze and SAE without Layer Freeze) show the similar performance in terms of decrease in error. On the other hand, the decrease of error in standard stacked autoencoder is slow compared to our proposed

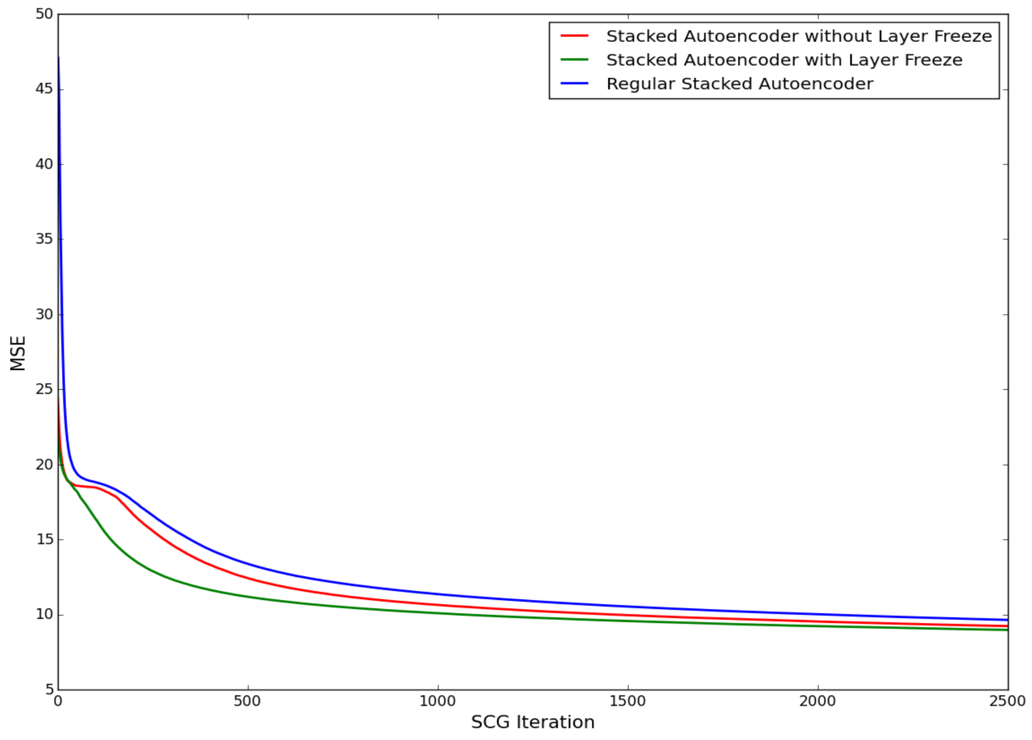


Figure 5.3: Post Training Error over SCG Iteration

algorithms. From this observation we can say our proposed algorithms have initialized the weights and biases close to the optimal value compared to standard stacked autoencoder. Among our proposed algorithms SAE with Layer Freeze has found a better initial condition than SAE without Layer Freeze.

5.2 Dimensionality Reduction using centroid-encoding

Dimensionality reduction is one of the key aspects of Machine Learning. It is especially useful to visualize high dimensional data in two/three dimensional space. In this section we describe the details of our experiments and the visualization using our proposed algorithm, centroid-encoding. We also discuss the effectiveness of our algorithm with respect to PCA and autoencoder. We use all three datasets to compare the algorithms.

5.2.1 Experiments

The purpose of this experiment is to compare the low dimensional encoding of high dimensional data using the methods autoencoder and centroid-encoder. The algorithm for centroid-encoder is defined in Chapter 3. We use three different datasets: gene expression data of Crab eating Macaque, MNIST handwritten digit data and EEG metal task data.

MNIST is a hand written digit dataset which contains all the digits 0..9. Each of the hand written digit is 28 x 28 image. Thus the ambient space of each data point is $X^i \in \mathbb{R}^{784}$. We have projected the MNIST data on a 2D plane using centroid-encoder and autoencoder. The two dimensional visualization is formed by the output of the bottleneck layer using the architecture $784- > [500- > 250- > 125- > 2- > 125- > 250- > 500]- > 784$. We randomly select 50,000 samples from the original training set to train the autoencoder and centroid-encoder. We show the 2D projection of training data and test data for both methods.

The Macaque dataset has 1469 number of biological pathways. We picked the T-cell Receptor Signaling pathway for our study because this is one of the top pathways which show early biological signal in the presence of Ebola virus. This pathway has 96 genes, so the ambient space is $\mathbb{R}^{96}$. We projected the data on 3 dimensional space using PCA, autoencoder and centroid-encoder. In the case of PCA, we have projected the data on the first three principal components. We have used a three layer *bottleneck* architecture $96- > [25- > 3- > 25]- > 96$ to project the data on 3D space using autoencoder and centroid-encoder. The data is comprised of four classes: pre-bleed (before inoculation), infection after day one, day two and day three. The pre-bleed class has 30 samples, the sick day one class has seven samples, the sick day two class has twelve samples and the sick day three class has seven samples. As centroid-encoder is a supervised algorithm, we split the data into training and test set. We have used 80% of total samples from each class to train the model and the rest to project on the trained model. In contrast, we have not sequestered a test set for PCA and autoencoder because they are both unsupervised algorithms.

We have also applied centroid-encoding method on the EEG mental task data. Our aim is to understand the spatio-temporal relationship among four different mental tasks by visualizing them in low dimensional projection. We also want to compare the visualization between autoencoder and centroid-encoder. The EEG data is collected for four different mental tasks: visual task, counting task, fist movement task and singing task. The data is collected from an EEG cap which has eight channels. For each mental task, we have collected 10 seconds of continuous EEG data in each trial and we have collected the data in five different trials. As we have four different mental tasks so we have collected 40 seconds of EEG data in each trial. So in five trials, we have collected 200 seconds of EEG data. To be more precise we have collected 50 seconds of EEG data for each mental task. As the sampling frequency of our EEG device is 256 Hz, we have (256 x 50) samples from each electrode / channel for a given mental task. If we consider all the eight electrodes, we have (256 x 50 x 8) samples for a given mental task. For this experiment we have stacked a quarter of a second data (64 time points) from each channel to constitute a data point. Therefore each data point is a vector of 64 x 8 = 512 elements. So the ambient space of each data point $X^i \in \mathbb{R}^{512}$. As autoencoder is an unsupervised algorithm, we have used all the five trials to constitute the training set. As mentioned earlier, we have 50 seconds of EEG data for each mental task and (256 x 50) samples from each channel, therefore we have (256 x 50 / 64) = 200 data points to train autoencoder. On the other hand, we have sequestered a test set for centroid-encoder as it is a supervised algorithm. We have used the EEG data from first four trials to train centroid-encoder and the data from the fifth trial to test it. Hence, we have (256 x 40 / 64) = 160 points in training set and (256 x 10 / 64) = 40 points in test set. We then we project the data on 3 dimensional space using autoencoder and centroid-encoder. We have used different network architecture to see how the embedding changes.

5.2.2 Results

First we discuss the dimensionality reduction on MNIST data. Figure 5.4 and Figure 5.5 show the two dimensional embedding of MNIST data using centroid-encoder and autoencoder, respectively. We have added the two dimensional visualization of training data as well as test data. As centroid-encoder is a supervised learning algorithm we like to consider the embedding on the test

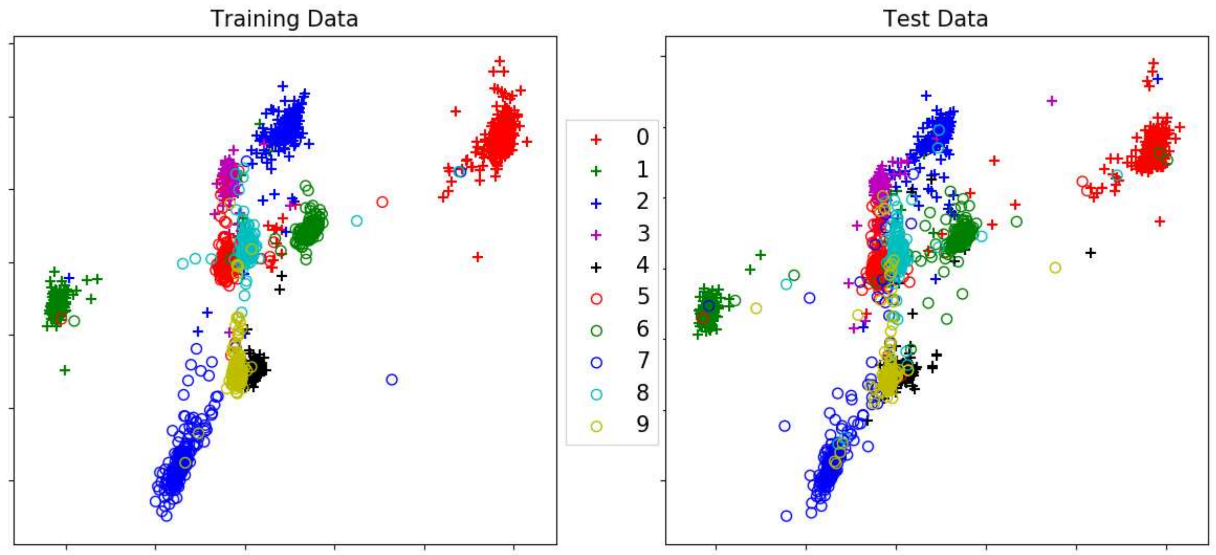


Figure 5.4: A two dimensional embedding of 500 digits of each class of MNIST data using a 784-500-250-125-2 centroid-encoder.

data as well. MNIST comprises with handwritten from digit 0..9. These digits have both similarities and dissimilarities among them. It is quite natural to expect that the digits which are similar to each other will be embedded more closely in the 2D plane. For example, digits 4 and 9 have many similarities whereas digits 1 and 0 are completely dissimilar. So in the 2D plane we can expect digits 4 and 9 will be embedded more closely than digits 0 and 1. Now let's look at Figure 5.4. On the left side we have the training data and on the right side we have the test data. There are ten different classes, so ideally one should see 10 different clusters. One could easily figure out 7 different clusters without looking at the colors. The 0s and 1s are put far apart. Similarly the 7s and 0s are also put far apart. On the other hand digit 9 and digit 4 are close in the embedding plane. So far the embedding of digits 0, 1, 7, 4 and 9 makes sense. Now if we look at the projection of the digits 8, 5 and 3 on 2D plane, we observe that they are close to each other; although these three digits have similarities in them. It is also possible that the human hand writing of three digits are very similar. From this observation we can say that the algorithm probably have given high importance on this common feature to embed the digits 8, 5 and 3.

Now we look into the embedding of autoencoder. As autoencoder is an unsupervised method, it does not matter which data we are looking at. The separation between the digits 1 and 0 is very

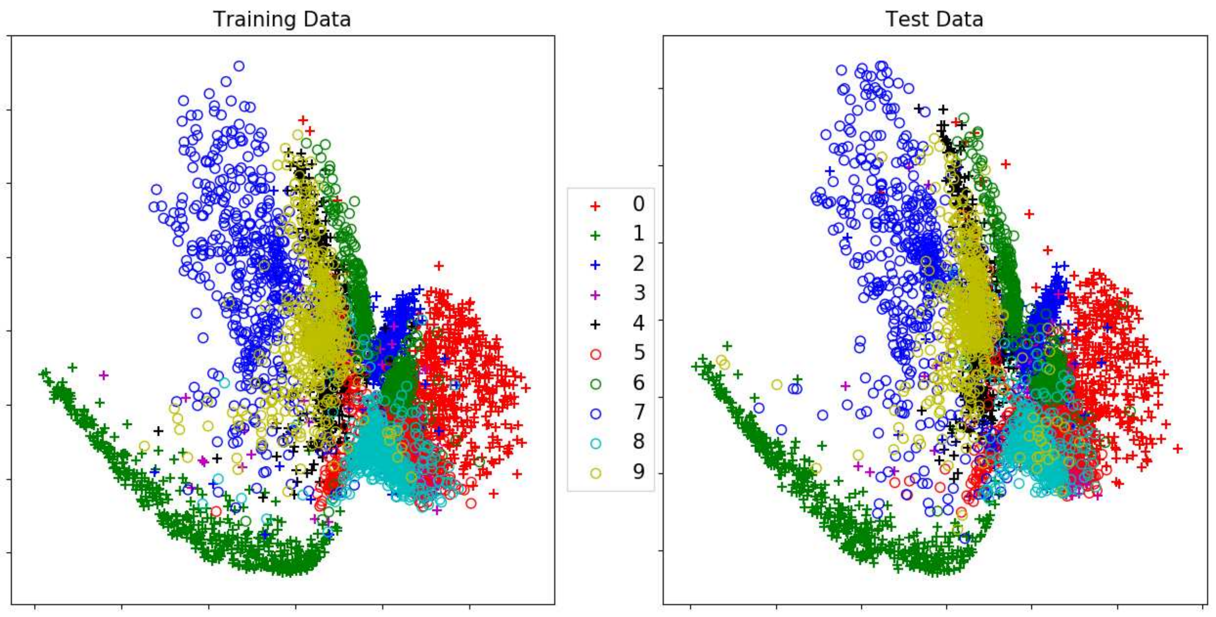


Figure 5.5: A two dimensional embedding of 500 digits of each class of MNIST data using a 784-500-250-125-2 autoencoder.

clear. Digit 7s are also clustered away from digit 0s. But the difference of rest of the digits in 2D plane is not clearly visible without the help of color annotation. If we go by the color annotation of different digits we can find structure in the clustering. For example digit 8 is close to digit 6; but digit 4 and digit 9 are completely overlapped to each other. Beside this there is no visual separation in the grouping of the digits 9, 6, 5, 4, etc.

If we compare the embedding of these algorithms, It is clear that centroid-encoder reveals more structured information than autoencoder. This is what one would expect. As the centroids of each class are different from each other in the ambient space, encoding the centroids would likely capture structured information in low dimensional space as well. This is exactly what we see in Figure 5.4. We can also observe that centroid-encoder groups samples of a class more compactly than autoencoder. We can also discover the fact that centroid-encoder puts the clusters of dissimilar objects far apart compared to autoencoder. The clusters of digits 1 and 0 clearly reveal this fact. This is also visible from the digits 7 and 0.

Now we discuss the 3 dimensional embedding of Macaque data using PCA, autoencoder and centroid-encoder. Figure 5.6 and Figure 5.7 show the 3 dimensional visualization of T Cell Receptor Signaling pathway from Macaque dataset using the methods PCA and autoencoder respectively.

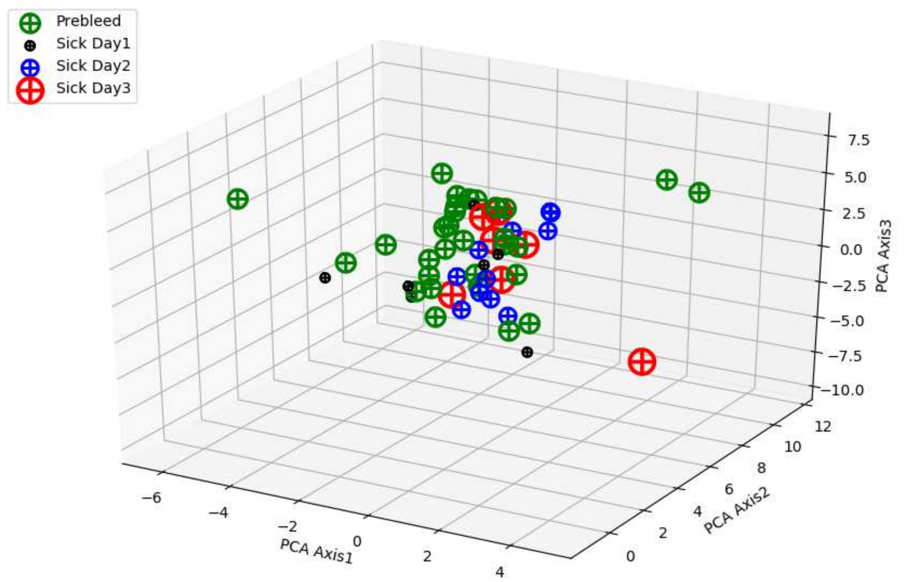


Figure 5.6: A three dimensional embedding of T Cell Receptor Signaling pathway using PCA.

Clearly these projections do not revealing any structured information about the data. The pre-bleed subjects (healthy ones) are mixed with the rest of the sick subjects. These plots do not show any temporal pattern among the sick subjects with respect to the pre-bleed subjects. In contrast, the 3 dimensional visualisation in Figure 5.8 using centroid-encoder reveals more structured information about the disease progression in Macaque over time. In most cases the test samples are close to training samples. Compared to day one, training and test samples on day two are further away from healthy. On day three the test subjects are clumped together completely with training subjects. This is because the infection gets worse as the day progresses which reveals a biological fact. This finding is consistent with the results of Rubins et al. [18] who found that the gene expression of this pathway changes (higher or lower) as the day progresses compared to the baseline values.

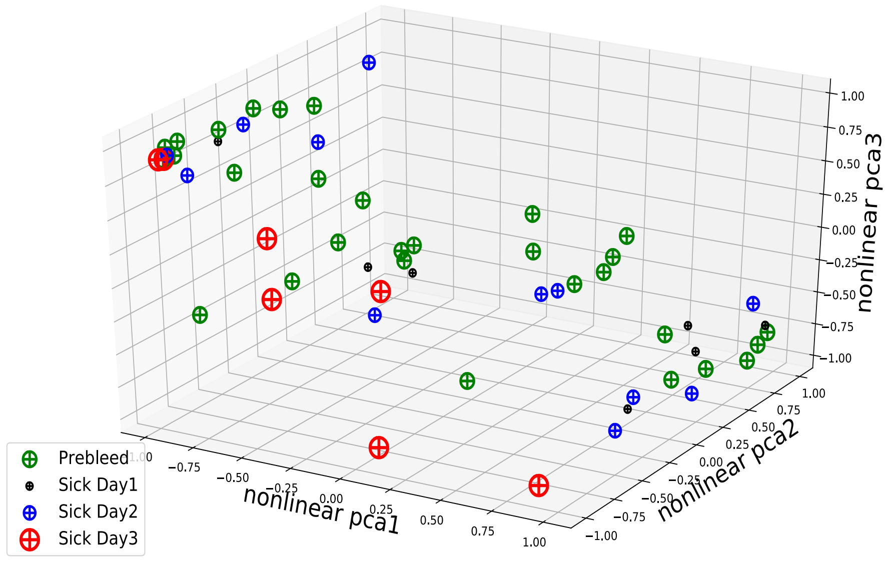


Figure 5.7: A three dimensional embedding of T cell receptor signaling pathway using autoencoder.

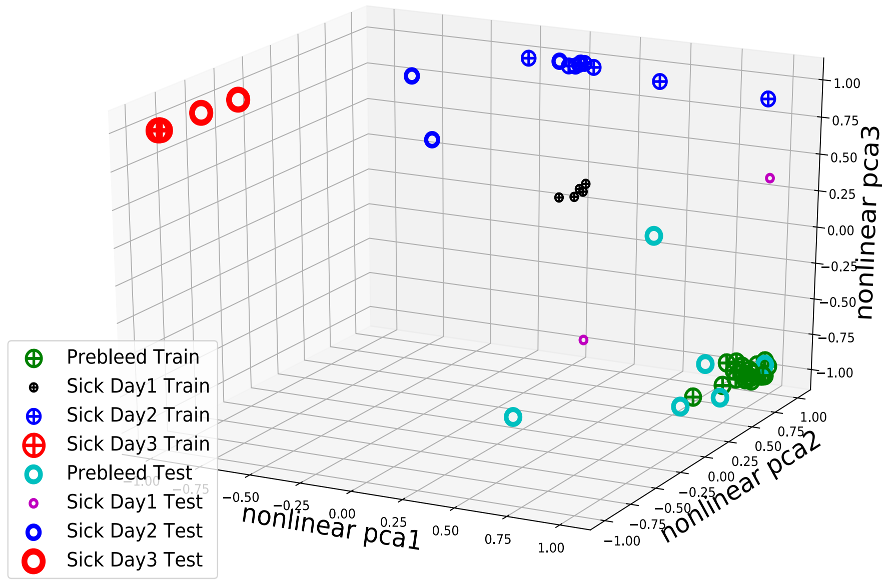


Figure 5.8: A three dimensional embedding of T Cell Receptor Signaling pathway using the supervised centroid-encoder.

In our case, the drifting of sick subjects from the healthy ones over time tells that the difference of gene expression between healthy macaque and sick macaque increases over time.

Now we discuss dimensionality reduction on the EEG data. Figure 5.9 and Figure 5.10 show the three dimensional embedding of EEG mental task data using standard autoencoder. We used two different hidden layer structures to capture the low dimensional relationship among the mental tasks.

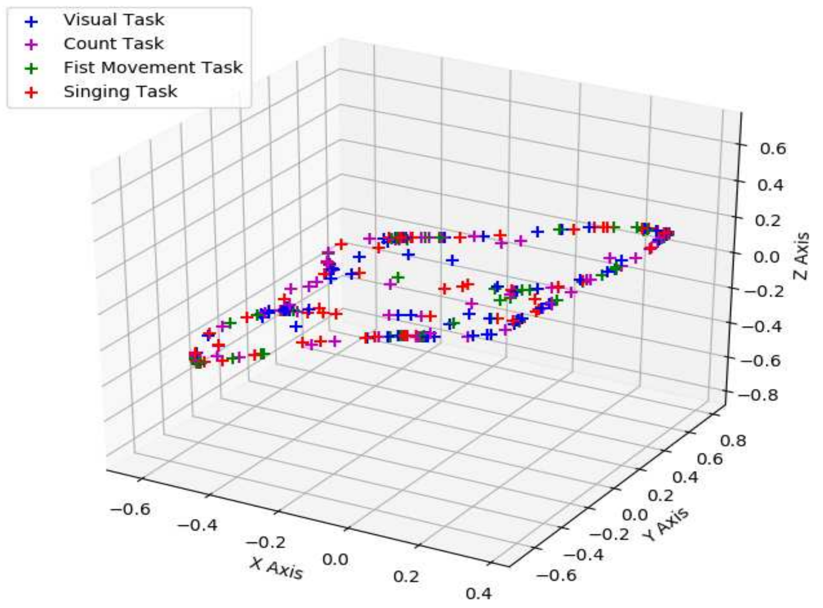


Figure 5.9: A 3 dimensional embedding of EEG mental task data using autoencoder. The data is passed through the network 512->[3,3,3,3,3]->512. Output of the 3rd hidden layer has been plotted in 3D space.

It is clear from these two plots that autoencoder is unable to show any correlation among the four different mental tasks in low dimension. But it is noteworthy that autoencoder does reveal a global pattern. Form these plots one can say that EEG mental task data may reside in a low dimensional manifold (circle/sphere).

In contrast the centroid-encoder does capture the correlation between the mental tasks in three dimensional space. In Figure 5.11 we used five hidden layers where each of them contains three hidden units. From this plot one can easily see that the singing metal task is different from the rest. The overlap between the visual and the count task suggests a strong correlation between them. We can’t say too much about the fist movement task as some of the points of this task are close to both

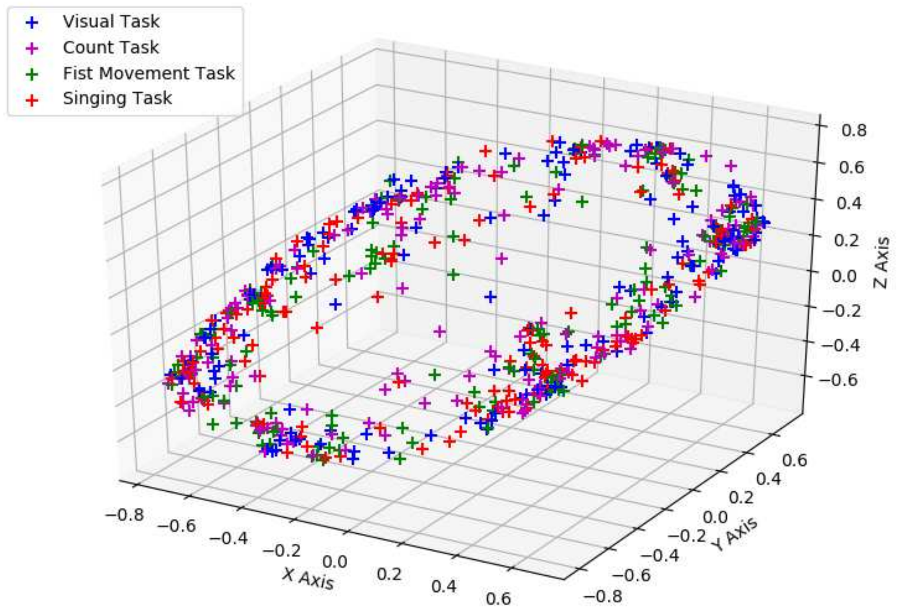


Figure 5.10: A 3 dimensional embedding of EEG mental task data using autoencoder. The data is passed through the network 512->[10,3,3,3,10]->512. Output of the 3rd hidden layer has been plotted in 3D space.

visual and count task, which are close to singing task. This pattern is consistent in train data as well as test data.

If we look at the Figure 5.12, we observe similar pattern on the singing task, visual task and count task that we have seen in Figure 5.11. In this case we use five hidden layers where the first and fifth ones have ten hidden units and rest of them have three hidden units each. This network configuration is able to separate the different mental tasks better compared to the previous one (Figure 5.11). One can see the clustering of fist movement task in training data as well as in test data.

In Figure 5.13 we have used five hidden layers where the first and fifth ones have five hidden units each and rest of them have three hidden units each. Qualitatively this figure is quite similar to Figure 5.11. We see that the singing task clustered separately from the rest; the visual and count task stay very close to each other.

We like to remark on the variability of these 2D / 3D plots on different initial conditions. As the optimization problem of neural network is non-convex, the solution will differ based on

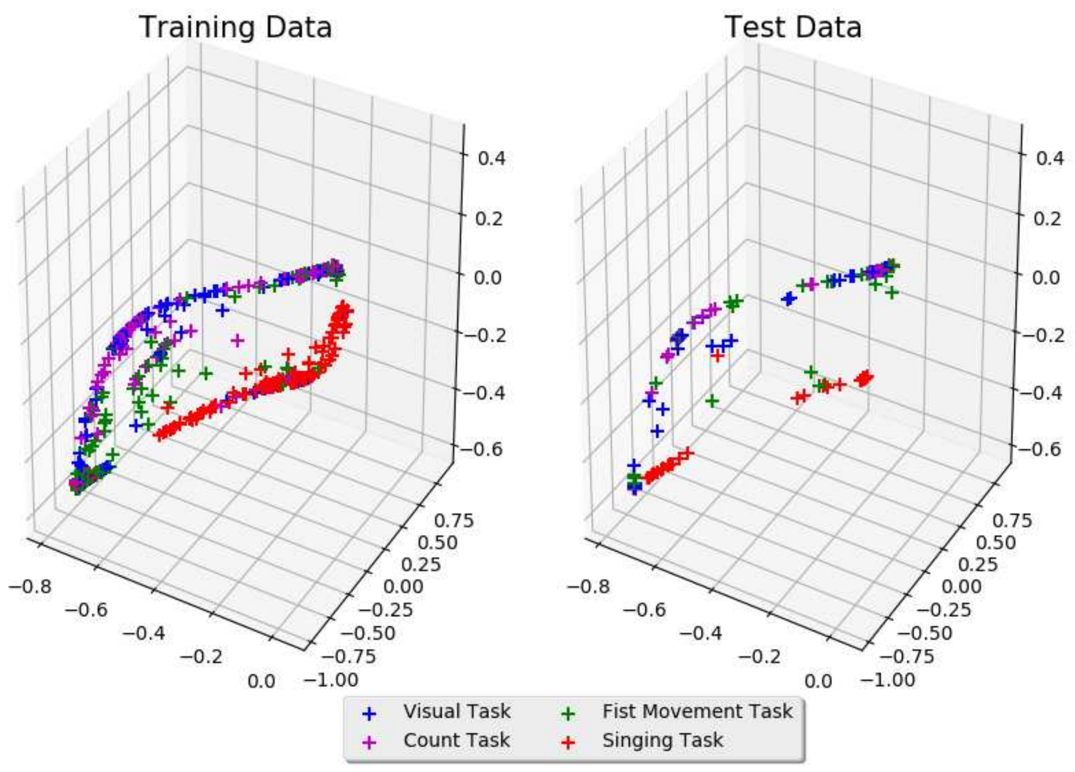


Figure 5.11: A 3 dimensional embedding of EEG mental task data using centroid-encoder. The data is passed through the network 512->[3,3,3,3,3]->512. Output of the 3rd hidden layer has been plotted in 3D space.

different initial conditions. To overcome this problem one can run the algorithms (autoencoder and centroid-encoder) multiple times and pick the solution which gave minimum training error.

We have also analyzed the low dimensional visualization of different mental tasks by plotting the activity across different EEG channels. Figure 5.14 shows the topographic plot of the variance of the electrical activity across eight different EEG channels. First we picked a specific point for each task from the bottleneck layer (3D space) generated by centroid-encoder. For example we picked four points each from a task from the 3D space of Figure 5.12. Once the points are fixed in the 3D space, we capture the output of the centroid-encoder for each of those points. The high dimensional ($\mathbb{R}^{512}$) representation of each sample is then converted to signal across eight channels where each channel has 64 time samples. Now we calculate the variance of each channel to generate the topographic plot. For the counting task we see high variance in the channels 'F3', 'F4' and 'P4'. There is moderate variance in the channels 'C3', 'C4', 'O1' and 'O2'. The fist

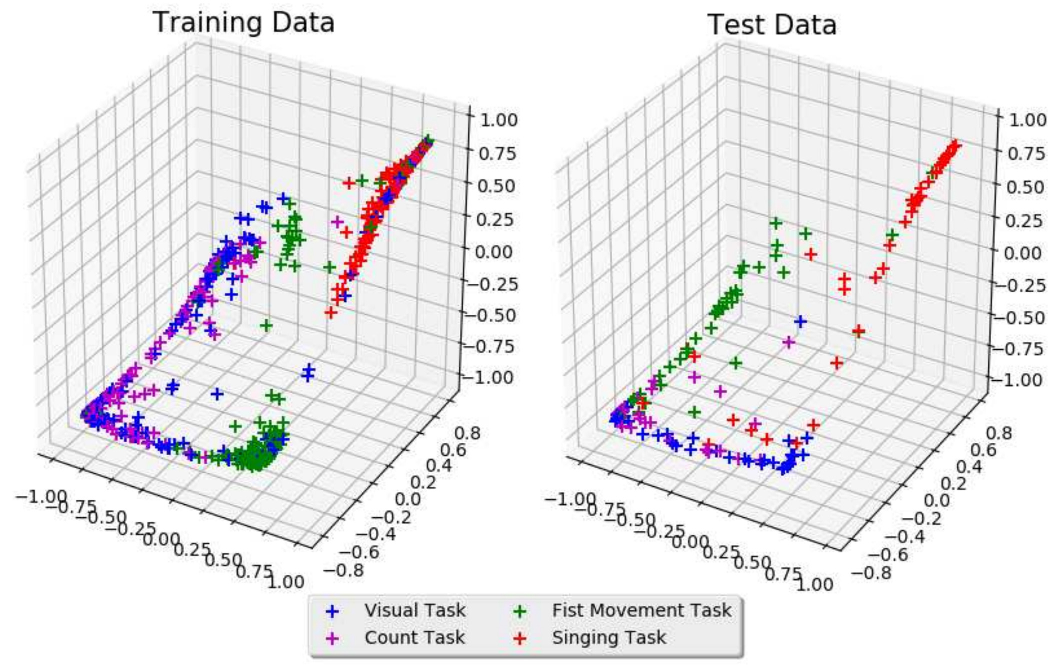


Figure 5.12: A 3 dimensional embedding of EEG mental task data using centroid-encoder. The data is passed through the network 512->[10,3,3,3,10]->512. Output of the 3rd hidden layer has been plotted in 3D space.

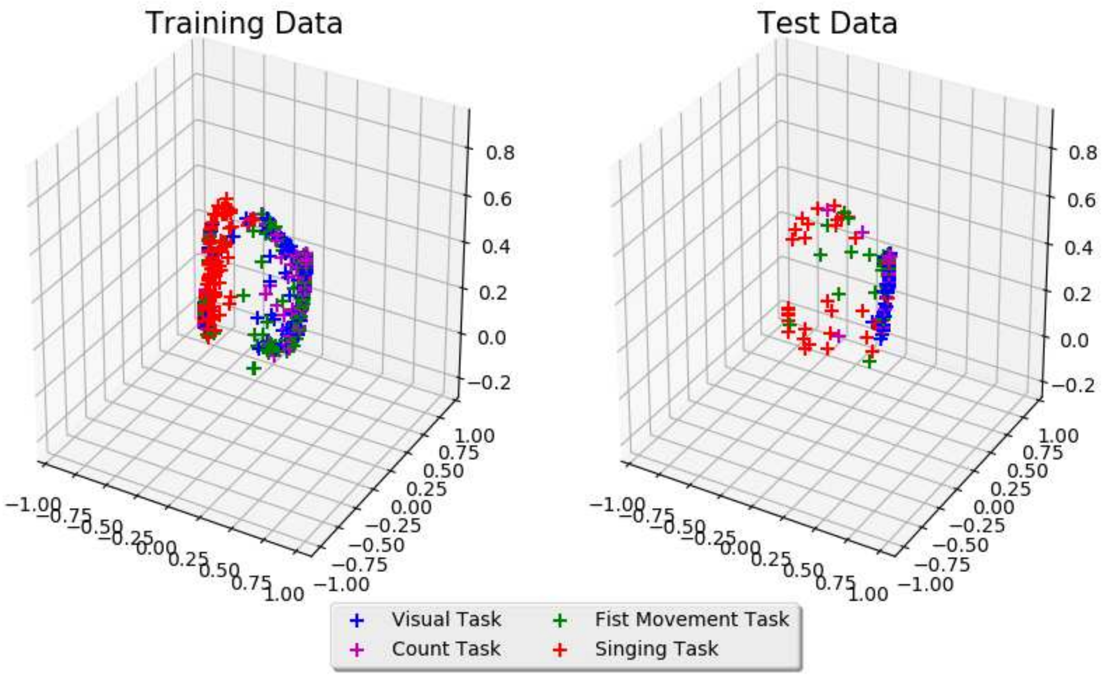
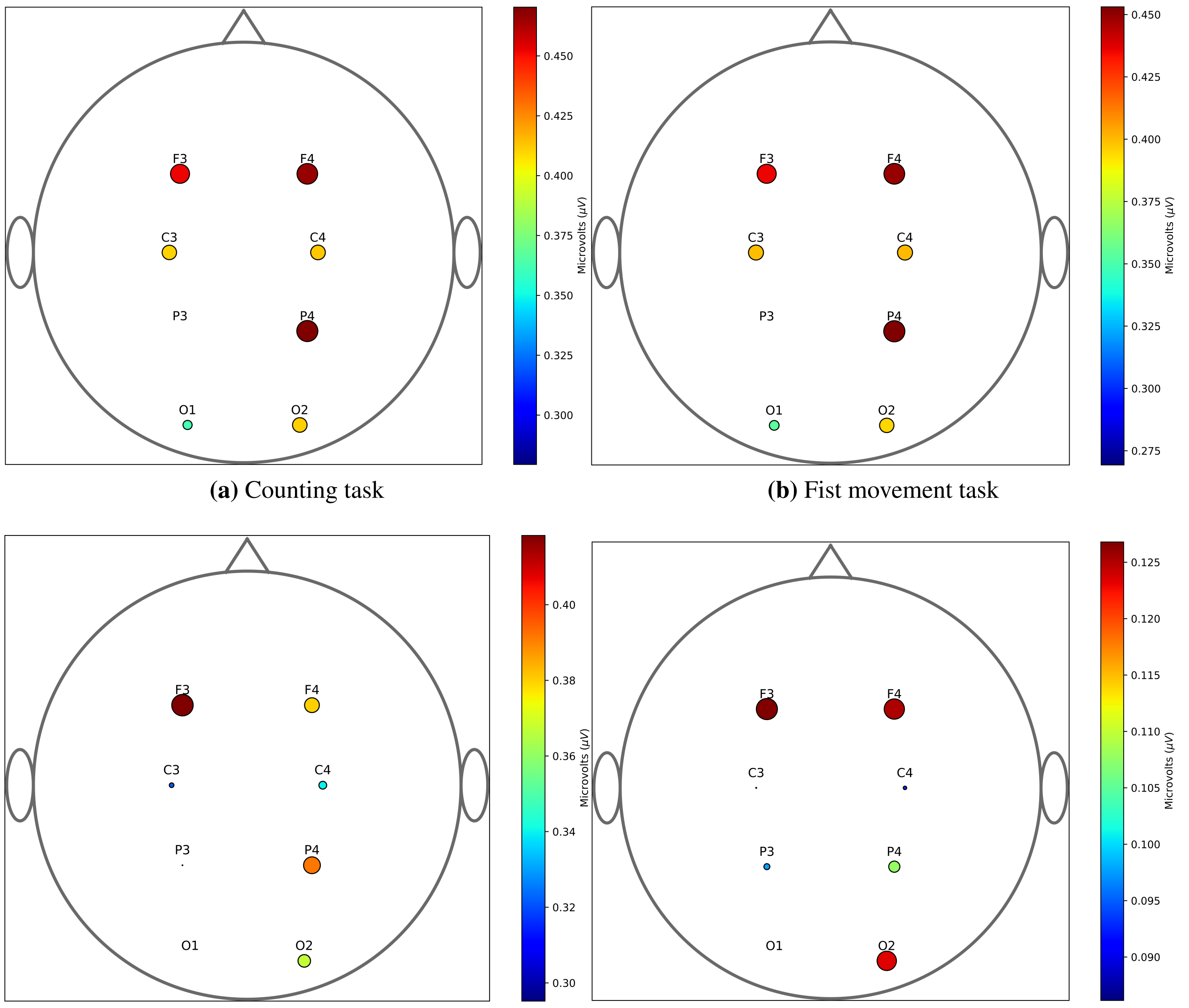


Figure 5.13: A 3 dimensional embedding of EEG mental task data using centroid-encoder. The data is passed through the network 512->[5,3,3,3,5]->512. Output of the 3rd hidden layer has been plotted in 3D space.



(a) Counting task

(b) Fist movement task

(c) Singing task

(d) Visual task

Figure 5.14: Variation of EEG scalp signal across different mental tasks

movement task shows very similar characteristics to the counting task. This is quite evident from the topographic plot of these two tasks. The similarity among these tasks suggests that for this subject brain activity was similar for these two tasks. In contrast to these tasks, singing and visual tasks have considerable differences among them. The singing task shows high variance in the channels 'F3' and 'P4' and moderate variance in channels 'O2' and 'F4'. In contrast to this, the visual task shows very high variance in the channels 'F3', 'F4' and 'O2'. We also observe that the 'F3' channel has high variance across all the tasks. At the same time we would like to mention the variability in the visual task. We noticed that sometimes the channel 'P4' has moderate variance and at the same time channel 'O2' has low variance. This could be a signature of the visual task but more analysis needs to be done to understand this better.

5.3 Classification using Centroid-encoding

In this section we illustrate the classification task using centroid-encoding. We focus on how centroid-encoder can be used as a pre-training step. We compare our results with autoencoder and standard neural network on multiple architecture.

5.3.1 Experiments

For classification task centroid-encoding can be viewed as a pre-training step. In this step a sample from a class is mapped to its corresponding centroid through the hidden layers. As each class of a dataset is different from each other so the corresponding centroids will be different too. When the model maps each sample to its centroid it automatically learns the difference among the classes. Hence conceptually, pre-training a neural network model with centroid-encoding may be beneficial. The steps of classification are mentioned in Algorithm 2

```
Algorithm 2: Classification with Centroid-encoder.
  Step1: Calculate the centroid of each training class.
  Step2: Pick a network architecture(input layer,hidden layers, output layer).
  Step3: Train the centroid-encoder as mentioned in the algorithm 1.
  Step4: Replace the output layer of centroid-encoder by a softmax classification layer.
  Step5: Train the classification layer while keeping the weights of the rest of the network
         fixed.
  Step6: Fine tune the network.
```

In this experiment our goal is to train centroid-encoder classifier using a single hidden layer and measure the classification accuracy on test data. From the original MNIST dataset 50,000 samples are drawn randomly to train our models. The rest 10,000 samples are used as validation set. The validation set is used to find the optimal network architecture, optimal number of training iteration etc. We have used the original MNIST test set which has 10,000 samples to test our models. We also standardize the data. We used five different bottleneck architectures as mentioned below: 28 x 28 ->[500 -> x -> 500]-> 28 x 28 where $x \in \{3, 10, 20, 40, 80\}$. The bottleneck architectures are used to pre-train autoencoder and centroid-encoder. After the pre-training a softmax layer is added for classification. The softmax layer takes the values from the bottleneck layer and maps them to the corresponding class values as shown: $[x]$ -> 10 where $x \in \{3, 10, 20, 40, 80\}$ and 10 is the number of classes. This process is followed by a fine tuning/post-training after combining the pre-trained network with softmax layer. The structure of the fine tuning network is: 28 x 28 ->[500 -> x]-> 10 where $x \in \{3, 10, 20, 40, 80\}$. In contrast we do not use any pre-training step for standard neural network. We directly train the fine tuning network to do classification. We have used Scaled Conjugate Gradient method in error backpropagation. We have also employed early stopping as a regularization technique to prevent our models from over fitting. We repeat the experiment three times for each method.

5.3.2 Results

Result of this experiment are in Tables 5.7, 5.8, 5.9, 5.10 and 5.11. While analyzing these results we want to focus on the following aspects:

- Whether centroid-encoder is a better pre-training step than autoencoder.
- How does the classification accuracy change over the change of hidden units in bottleneck layer.

To analyze performance of pre-training of centroid-encoder and autoencoder we need to look into the classification accuracy of softmax layer in autoencoder and centroid-encoder. We used five different bottleneck architectures. Table 5.7 shows the classification accuracy when the bottleneck layer has only 3 hidden nodes. The classification accuracy on test data after pre-training for autoencoder is 59.92$\%$ which is quite low compared to centroid-encoder. Test accuracy of centroid-encoder is 95.68$\%$. We also see that centroid-encoder outperforms autoencoder in training and validation set as well.

Now we discuss the results, which is in Table 5.8, for the second architecture where we have 10 hidden units in bottleneck layer. We see that this architecture helps to improve the performance of autoencoder compared to the previous architecture. The accuracies on three different datasets (training, validation and test) reaches around 81% which is a big jump from the previous results. At the same time we see that centroid-encoder outperforms autoencoder in all the three datasets (training, validation and test) for this architecture too; the accuracies reach around 97%.

As the number of hidden units increases in the bottleneck layer the accuracy after pre-training keeps increasing for autoencoder. This is evident from Table 5.9 to Table 5.11. On the other hand, centroid-encoder performs better than autoencoder in all of these five bottleneck configurations.

Now we consider the classification accuracies after the fine tuning; we observe that there is no considerable difference between autoencoder and centroid-encoder. In fact when the bottleneck layer has 80 hidden units classification accuracy is exactly the same which is 97.96%. We also note that the classification accuracy for autoencoder improves a lot after fine tuning whereas for centroid-encoder we do not see too much of an improvement after fine tuning. Therefore fine tuning plays a significant role in autoencoder classification network as opposed to centroid-encoder. From this observation we can say that pre-training with centroid-encoding puts the solution much closer to an optimal one.

Now we will discuss the change of classification accuracy over the different size of bottleneck layer. To emphasize this aspect we will consider accuracies of all the three models: autoencoder, centroid-encoder and standard ANN. For autoencoder and centroid-encoder we will only consider the accuracy after fine tuning. Recall that we have used five different architectures for classifications: 28 x 28 ->[500 -> x]-> 10 where $x \in \{3, 10, 20, 40, 80\}$. Table 5.7 shows the classification accuracies for the first architecture which is 28 x 28 ->[500 -> 3]-> 10. We see that, compared to autoencoder and centroid-encoder, standard ANN has done very poorly. We think that standard ANN has stuck in a local minima which has caused the results to be poor. Due to the narrow bottleneck, the optimization function is unable to find a direction to come out of the local minima. This type of behaviour is common in standard neural network, and this is why now a days people are using very large deep architecture which is more likely to overcome the problem with local minima since the optimization function can search from a large parameter space. In autoencoder and centroid-encoder we have used a pre-training step with the architecture: 28 x 28 ->[500 -> 3 -> 500] -> 28 x 28. We think that this pre-training has helped to overcome the local minima problem because the optimization algorithm can search from a larger parameter space compared to standard ANN. As the number of hidden units increases in the bottleneck layer the performance of standard ANN starts to increase which is evident from Table 5.8 to Table 5.11. We observe that the accuracy is slightly higher for standard ANN compared to autoencoder and centroid-encoder but this difference is not that significant.

Table 5.7: Comparison of classification accuracy among three models where the bottleneck layer has 3 hidden nodes.

	Autoencoder		Centroid-encoder		Standard ANN
	Softmax(%)	Fine tuning(%)	Softmax(%)	Fine tuning(%)	Accuracy(%)
Training	59.45	99.05	98.72	99.91	69.97
Validation	59.30	95.69	95.16	95.74	67.13
Test	59.92	95.95	95.68	95.68	67.35

Table 5.8: Comparison of classification accuracy among three models where the bottleneck layer has 10 hidden nodes.

	Autoencoder		Centroid-encoder		Standard ANN
	Softmax(%)	Fine tuning(%)	Softmax(%)	Fine tuning(%)	Accuracy(%)
Training	81.92	99.99	98.53	100	100
Validation	81.13	97.08	97.00	97.56	97.97
Test	82.40	97.17	97.18	97.72	97.87

Table 5.9: Comparison of classification accuracy among three models where the bottleneck layer has 20 hidden nodes.

	Autoencoder		Centroid-encoder		Standard ANN
	Softmax(%)	Fine tuning(%)	Softmax(%)	Fine tuning(%)	Accuracy(%)
Training	87.78	100	99.34	100	100
Validation	87.47	97.48	97.46	97.79	98
Test	88.16	97.52	97.52	97.78	98.13

Table 5.10: Comparison of classification accuracy among three models where the bottleneck layer has 40 hidden nodes.

	Autoencoder		Centroid-encoder		Standard ANN
	Softmax(%)	Fine tuning(%)	Softmax(%)	Fine tuning(%)	Accuracy(%)
Training	90.38	100	99.31	100	100
Validation	89.62	97.42	97.25	97.79	98.23
Test	90.72	97.38	97.58	97.98	98.25

Table 5.11: Comparison of classification accuracy among three models where the bottleneck layer has 80 hidden nodes.

	Autoencoder		Centroid-encoder		Standard ANN
	Softmax(%)	Fine tuning(%)	Softmax(%)	Fine tuning(%)	Accuracy(%)
Training	92.07	100	99.24	100	100
Validation	91.18	97.90	96.97	97.80	98.26
Test	91.95	97.96	97.25	97.96	98.24

Chapter 6

Conclusions and Future Work

In this chapter we will present our conclusion on our proposed methods based on the results of the experiments we have done. We will also propose some of the future direction on our work.

6.1 Pre-training

In this research we have proposed two variations in layer wise pre-training. The results of our experiments show that pre-training with layer-freeze has performed better compared to pre-training without layer-freeze and standard stacked autoencoder. The performance of standard stacked autoencoder and pre-training without layer freeze is very similar. In Section 3.1 we have explained how standard stacked autoencoder adds noise in the model. We have hypothesized that if we reconstruct the original input while adding new layers we might be able to reduce the effect of noise addition in the model. The experimental results support our hypothesis. Pre-training without layer freeze has not done well compared to pre-training with layer freeze. We think that this is due to the unlearning of previously trained layers when we add a new hidden layer. This is explained in detail below.

In our experiments we have used the network architecture: 28x28 ->[1000, 500, 250, 30, 250, 500, 1000] -> 28x28. First we have trained the autoencoder: 28x28 ->[1000] -> 28x28. From this training we obtain weight matrices $w1$=[785x1000], which connects input to hidden units, and $w2$=[1001x784] which connects hidden units to output. We have then added the new hidden layer with 500 hidden units. But while training the autoencoder 28x28 ->[1000,500,1000] -> 28x28, we have allowed the pre-trained weights $w1$ and $w2$ to change. This change has caused a loss of learning in $w1$ and $w2$, whereas in layer-freeze approach we have not allowed the pre-trained weight to change. So in layer-freeze approach the local learning associated in each hidden layer is intact. We also observe that if we use many iterations in post training, the error-over-iteration curve of layer-freeze and without layer-freeze converge to each other which can be seen in Figure

5.3. This is expected. Here we also observe that the error curve in post training for standard autoencoder is consistently high compared to the other two curves. Both of our proposed pre-training approaches minimize the addition of noise in the model. Thus we think that the weight initialization using our pre-training methods puts solution close to the optimal one.

6.2 Centroid-encoding

We have proposed a new supervised learning algorithm that can be used in dimensionality reduction as well as in classification. Our main motivation is to come up with an algorithm that can produce a good low dimensional structure of high dimensional data. We are particularly interested to have a two/three dimensional visualization of MNIST data, EEG data and gene expression data as the ambient space of both of these dataset is high dimensional. We tried both linear and nonlinear PCA but the methods were unable to give any interesting structure of the data. Although we note that on MNIST data autoencoder (nonlinear PCA) has performed relatively well compared to other two datasets.

For a given dataset, which is labeled, one can expect to have a variance between the classes / categories. At the same time we can also expect the intra-class variance to be less than the inter-class variance. This assumption may not hold true for all datasets, but in practice this assumption may work in most of the cases. Keeping this assumption in mind we can say that the centroid is a good representative of the corresponding class. We have hypothesized that a nonlinear mapping from input data to its corresponding centroid through low dimensional space might reveal some hidden structure in low dimensional space. The three dimensional visualization of crab eating macaque and EEG data do support our hypothesis. The two dimensional embedding of MNIST data also supports this claim. The three dimensional embedding of T cell receptor signaling pathway using centroid-encoding (Figure 5.8) is a remarkable improvement over autoencoder (Figure 5.7). We can make similar comments on EEG data and MNIST data, too. Although the autoencoder was able to capture a global pattern of EEG data in low dimensional space, the centroid-encoder captures both the global pattern as well as local pattern in the data. We also want to note that the global structure in three dimensional space captured by both the methods suggests that EEG data may reside in a low dimensional manifold (circle/sphere).

The comparison of classification results among the three methods reveals a lot of information. First of all the pre-training using centroid-encoding outperforms the pre-training using autoencoding. This is clear from the classification accuracy of the softmax layer. This observation is consistent among the five different bottleneck architectures. Another interesting fact is that fine tuning does not improve the result as much for centroid-encoder as compared to autoencoder. This means the pre-training using centroid encoding found the solution very close the optimal one. We would also like to comments on the result of the standard ANN for the architecture: 28 x 28 -> [500 -> 3] -> 10. This is the worst result among all of them. We think that this particular architecture is not suitable for a classification network for MNIST data. Probably the small number of hidden units in the second hidden layer does not give enough space (dimension) to find an optimal solution. This particular classification architecture found local minimum very often as compared to other classification architecture. This is also evident from the results of another classification network which has more hidden nodes in the second layer. On the other hand autoencoder and centroid-encoder do not suffer from this problem. This is because the pre-training have helped to overcome the local minima. In real world problems, we often work with datasets which have low number of samples but high number of dimensions/features. For this type of dataset a big ANN architecture is not preferable because the model will most likely to overfit the data. We believe our centroid-encoder will be a good modeling tool for small datasets.

6.3 Future Work

In this context we would like to mention one potential scope of improvement of our proposed methods. In the future we would like to mix our layer-wise pre-training approach with RBM pre-training. In centoid-encoding our main goal was to capture the inter-class variance through a non-linear mapping. Now this method can be further extended to capture the intra-class variance. In other words we want to capture the finer local relationships in a class. To do this we can order points of a class based on a distance metric. Then we can define a neighborhood N_ϵ of a point $x_i \in C_k$,where C_k is a class/category, such that $x_j \in N_\epsilon$ if $d(x_i, x_j) <= \epsilon$ and $x_j \in C_k$ where $d(x_i, x_j)$ is the distance between x_i and x_j. In the non-linear mapping we will map each point x_i to the centroid of the neighborhood N_ϵ.

Overall, we can say that our proposed methods have the potential to produce good results. Further investigation needs to be carried out to fully exploit the capabilities of our proposed methods.

Bibliography

[1] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[2] Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, Inc., New York, NY, USA, 1995.

[3] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[4] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[6] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[7] David E Rumelhart, James L McClelland, PDP Research Group, et al. *Parallel distributed processing*, volume 1. MIT press Cambridge, MA, USA:, 1987.

[8] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91, 1991. The thesis is written in German, but it can be translated into English using Google translator.

[9] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[10] Geoffrey Hinton. A practical guide to training restricted boltzmann machines. *Momentum*, 9(1):926, 2010.

[11] I.T. Jolliffe. *Principal Component Analysis*. Springer, New York, 1986.

[12] Mark A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243, 1991.

[13] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[14] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[15] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[16] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, December 2010.

[17] M. Wang, Y. Chen, and X. Wang. Recognition of handwritten characters in chinese legal amounts by stacked autoencoders. In *2014 22nd International Conference on Pattern Recognition*, pages 3002–3007, Aug 2014.

[18] Kathleen H Rubins, Lisa E Hensley, Victoria Wahl-Jensen, Kathleen M Daddario DiCaprio, Howard A Young, Douglas S Reed, Peter B Jahrling, Patrick O Brown, David A Relman, and Thomas W Geisbert. The temporal program of peripheral blood gene expression in the response of nonhuman primates to ebola hemorrhagic fever. *Genome biology*, 8(8):R174, 2007.

[19] Martin Fodslette Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural networks*, 6(4):525–533, 1993.