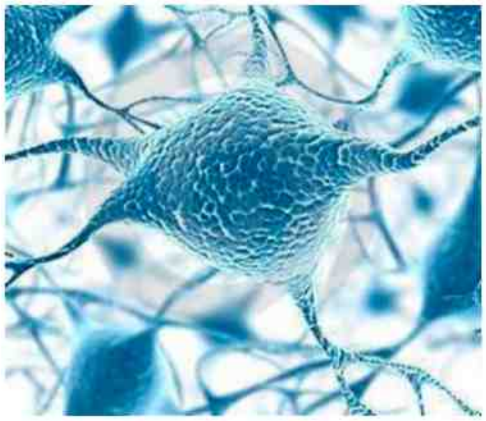
Brain Computer Interfaces Emotional State Detection (EEG Pattern Recognition)



A Thesis Presented to

The Faculty of the Computer Science Program

California State University Channel Islands

In (Partial) Fulfillment

of the Requirements for the Degree

Masters of Science in Computer Science

By

Atena Reyhani Shahrestani

April 2013

[illegible] 4/12/13

Advisor: Dr. Andrzej Bieszczad Date

[illegible] 4/12/13

Dr. Peter Smith Date

[illegible] 4/12/13

Dr. Richard Wasniowski Date

APPROVED FOR THE UNIVERSITY

[illegible] 4-12-13

Dr. Gary Berg Date

Emotional State Detection

(EEG Pattern Recognition)

By
Atena Reyhani Shahrestani

Computer Science Program
California State University Channel Islands

Abstract

Emotional State detection is a project for classifying brain waves into different emotional classes. In particular, EEG (Electroencephalogram) classification involves the categories of high arousal, low arousal, positive, and negative states. In this thesis, several classification methods are explored and tested on brain waves which are recorded under a supervised condition in order to present the best classification method for this particular problem. This project is not making any statement about Psychology but it is exploring methods for approaching such data from Computer Science perspective. Thus, the following document presents the process of data collection, data analysis through numerous solutions, and finally evaluates the results obtained from various classification strategies.

Acknowledgements

The author would like to thank the thesis advisor Dr. Andrzej Bieszczad for all the support and assistance during Master's program in Computer Science department at CSUCI and the thesis process.

TABLE OF CONTENT

TABLE OF FIGURES

Chapter 1: Introduction

1.1 Project Overview

This project originally started in the Computer Science department in collaboration with the Psychology department at California State University Channel Islands (CSUCI) to determine the brain's functionality in order to detect the traits of autism with the use of Emotiv EPOC Research Edition SDK. In the process of data analysis and due to the complexity of the problem and inconsistency in results, we decided to break it down, first, in order to better understand the nature of brain waves generated by emotional states, and second, to determine the features associated with EEG classification of the waves. In addition, due to consequence of problems dealing with the real-world data for Psychological purposes from Computer Science perspective, researching the possibility of dealing with data with such nature was critical.

Due to the complexity of wave pattern recognition, especially EEGs signals, determining the right classification method along with a feature vector suitable for this classification is a formidable task. Therefore, the research presented in this document allows for better understanding of this kind of problems. Furthermore, it provides guidelines to selecting right features and analytical methods suitable for tackling such problems. Identifying the right feature space for brain waves for the classification of complex concepts helps with implementing both applications and devices utilizing Brain Computer Interfaces.

In this project, the brain waves of a number of subjects are collected during watching videos which generate emotions such as calmness, happiness, sadness, and annoyance. The videos are selected from YouTube and they were assumed to prompt the emotions in the subjects. The selected videos showed satisfactory amount of correspondence to the emotions according to feedbacks from the subjects. Subsequently, the data are used by a machine learning algorithms to construct a pattern classifier that can recognize the emotions as four different categories; namely: positive low arousal, positive high arousal, negative low arousal, and negative high arousal emotions. A number of parameters and feature vectors are used to determine the best sets for this problem.

1.2 Hardware/ Software

For the purpose of data collection an Emotiv Epoc headset is used [1]. Emotiv is an EEG scanner with its accompanying software development kit that allows researchers to capture brain activity and analyze it in the context of associated cognitive behavior. More information about the device and its functionality is presented in chapter 2 and chapter 4.

The program written in C++ implements the communication between the headset and the Emotiv API in order to extract raw EEG data [Appendix C]. The process of data collection is illustrated more in chapter 4.

Data analysis and artifact removal are implemented in a MATLAB script [Appendix A]. In this process, a feature vector compatible with FANN (Fast Artificial Neural Network Library) [2] format is created from the resulting raw data file of the previous step. The result of the process is sent to another program in C in order to communicate with FANN.

In order to examine various neural network algorithms, the FANN library is used. The same software is adopted for developing, training and testing neural networks presented in this document. A portable graphical environment for FANN called fannExplorer [14] is also used for demonstrating the process of classification.

Finally, a python script is implemented to connect the parts of the project and enforce the order in the process [Appendix D].

1.3 Remaining Chapters

This document is going to present an overview of the field containing biological and computer neural networks in chapter two. When exploring neural networks, different types of pattern recognition are introduced in addition to some methods of data analysis used in this project. Furthermore, different methods of collecting brain waves are explored.

Chapter three contains technical details of the project and discloses the methods examined in this experiment.

In chapter four, the experiment is explained in detail, from the supervised test to the process of data collection, feature construction, and applying algorithms. The chapter continues with pointing out the process of finding the right parameters for classification.

Chapter five illustrates the results of the experience from a statistical point of view and presents comparisons of the results of examined methods in terms of graphs.

Then chapter six provides a summary of the work and focuses on the outcomes of the project.

Finally, potential improvements of the project are presented and future research which may benefit from this project is suggested.

1.4 Key Terms

- **ADHD** ------ Attention Deficit Hyperactivity Disorder.
- **AI** ----- Artificial Intelligence: A branch of computer science that works to create intelligence in machines, and software.
- **ANN** ----- Artificial Neural Networks: Mathematical model inspired by biological neural Networks.
- **API** ----- Application Programming Interface: A specification by a software product with the purpose of communication between software components.
- **Artifact** ----- Extra data resulting from the noise in the experiment that can result in error in classification.
- **BCI** ----- Brain Computer Interfaces: Software communicating directly using EEG scanners.
- **Cortex**----- The outermost sheet of neural tissue of the brain
- **EEG** ----- Electroencephaloghy: Voltage fluctuation of electrical activity on the surface of scalp.
- **Fast Artificial Neural Network (FANN)** ----- An open source neural network library written in C for multilayer artificial neural networks.
- **Firing Pattern**----- Sequence of activities associated with a certain action, or concept in the brain.
- **FMRI** ----- Functional Magnetic Resonance Imaging.
- **MSE** ----- Mean Square Error.
- NN ----- Neural Networks.
- **Pattern Recognition** ----- In machine learning, pattern recognition is classification of a given input data to groups labeled as different classes.
- **PCA** ----- Principal Component Analysis.
- **Tanh** ----- Hyperbolic Tangent.

Chapter 2: Field Overview

2.1 Brain Computer Interfaces

One of the mankind's largest and oldest fantasies has always been mind control and the ability to move and control objects at one's command or perform acts of magic with the power of mind. But mind control has been just fantasy in dreams, stories, or science-fiction movies.

Nowadays, new technologies are blurring the line between reality and imagination. Experiencing the content entirely in a new way, interacting with computers directly from the brain, and taking into account feelings and emotions are making the oldest mankind's dream real.

Human - machine communication being as simple as running simple devices, or as complex as programming complicated softwares, always has involved some sort of direct commands. Machines always needed to receive a direct command, or series of commands, in order to perform a specific function.

The data and input given to a machine always needed to be in a predefined structure and explicitly expressed. Unlike human communication, human-computer interaction never took into account implicit information such as body language, facial expression, etc. Thus, all the information that can be transferred without having to be explicitly expressed is lacking in our communications with machines.

With new technologies and direct communication between brain and computers, all those hidden data can be transferred and used. The interaction is much more informative and it is beyond what is explicitly expressed. Not only facial expression and body language are being transferred in these messages, the feelings and emotions are involved in dialogs with one another.

Thus, the next generation of human - computer interaction needs to evolve beyond direct and conscious inputs and involve both conscious and unconscious inputs. This chapter is going to introduce Neural Networks (NN) and Artificial Neural Networks (ANN). For more information on concepts of ANN, please refer to "Fundamentals of Artificial Neural Networks" [3] and more mathematical aspect of it can be found in "Introduction to The Theory of Neural Computing" [4]. A more physiological and psychological approach to NN and ANN is presented in "An Introduction to Neural Networks" [5].

2.2 Brain

In order to directly communicate with the brain, the first step is to understand how it works and learns from new input data. Understanding the brain helps us in presenting methods that these functionalities can be simulated by machines.

In order to simulate the functionality of the brain, lots of algorithms are implemented in computer science and more specifically in Artificial Neural Networks. Due to the complexity of the brain, Artificial Neural Networks which are computer simulation of neural networks are needed. Although some practical problems are solved by these learning algorithms inspired by the brain, it is not exactly how brain works. There is a basic understanding of operations driving the brain but there are lots of other mysteries about the brain to be discovered. In order to move forward with ANN, basic understanding of their biological counterparts is needed.

2.2.1 Brain's Structure

Brain consists of billions of neurons that interact with each other in neural networks.
A cortical neuron consists of: cell body, axon, and dendrite tree. It is through the branched axon of a cell that messages to other cells are being sent, and dendrite tree is where it receives messages from other neurons. The following figure (Figure 2.1) shows the structure of a neuron.

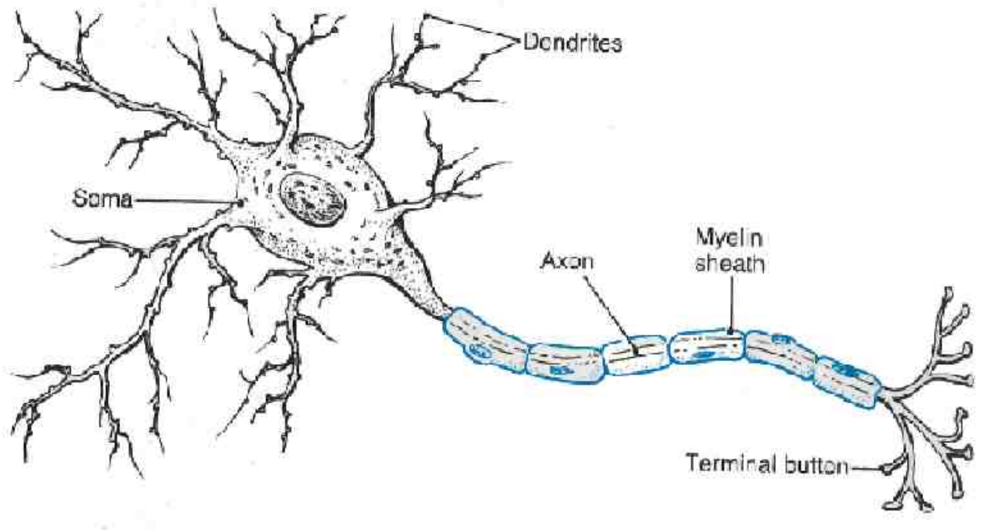


Figure 2.1: Structure of a Neuron [6]; Copied without the Consent of the Author s).

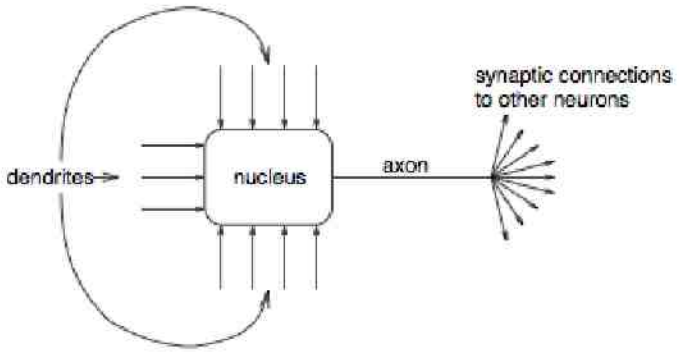


Figure 2.2: Simplified Neuron [7]; Copied without the Consent of the Author sı.

In neural network, positive weights are used to excite other neurons and negative weights are used to inhibit other neurons. When the dendrite tree of one neuron receives signal from axon of another neuron, a spike of activity in the axon causes charge injected to the post-synaptic neuron. A neuron generates spikes when it has enough charge in its dendrite tree to depolarize a part of axon body called axon hillock. This spike is the wave of depolarization that travels across the axon. When a spike of activity travels across the axon and arrives at the synapses, some vesicles of transmitter chemical are released and migrated to the surface. Some of these transmitter chemicals are implementing positive weights and some negative weights.

2.2.2 Brain's Learning

Brain has the capability of learning because synapses can adapt. Synapses get adapted by the number of vesicles that get released when spike arrives or by varying the number of receptor molecules that are sensitive to the released transmitter molecules. All the neurons receive inputs from other neurons and a few of them receive input from receptors. The neurons communicate with each other in the cortex with the spike of the activity. The effect of an input line is controlled by a synaptic weight which can be negative or positive. By adapting these weights, the network learns how to perform complicated computation.

There are about 10^11 neurons with high interconnectivity, each having about 10^15 connections [8].

Any task or computation in the brain can be affected by a large fraction of these neurons in a small fraction of time. The bandwidth used is much larger than a modern workstation.

Comparing synapses to Random Access Memory (RAM) of a computer, they are very small, with very low power requirements but they can adapt. They use locally available power to change their strength and that is how they learn to perform complicated computations. The way that they decide to change their strength and the rules of how they should adapt is still unknown.

As a result of each of the individual synapses adapting themselves, cortex learns to be modular. Thus, different bits of the cortex learn to perform different actions. These activities trigger larger blood flow in the specific region associated with the activity.

So, mapping of the brain regions to specific functions are genetically not predefined. Brain has a flexible learning algorithm and functions can be relocated to the other parts of the brain. Change in functionality of brain is referred to as neuroplasticity by neuroscientists [9]. The rapid parallel computations, flexibility, and adaptability of the brain introduce a fertile ground for research in a variety of disciplines such as biology, medicine, and computer science.

As a result, there is a need of identifying brain's functionality for various actions and concepts. In the next section (2.2.3), different technologies for collecting data from brain are presented.

2.2.3 Human Cognition and Cortical Mapping

As stated in the previous section, experiencing a specific thought or movement constantly activates the same region of the brain and blood flow increases in that particular region. So, identifying such regions and associating them with particular tasks, in other word mapping of the brain, is an important process. Cortical mapping is done through surgical/ invasive, or non-invasive techniques such as FMRI, connectome, or electroencephalography.

Methods such as FMRI and connectrome are used in order to map brain's activity based on blood flow. Strength of activities in different regions is used for constructing feature vector for activities and concepts classifications.

It has been also observed that ionic current flows within neurons of the brain results in voltage fluctuations. The waves recorded from these fluctuations show consistent pattern for a concept or an activity. EEG scanners record raw data of the signals transmitted from the brain. In this project, the latter method is used in combination with some data analysis and pattern recognition in order to find consistent firing pattern.

2.3 Emotiv Epoc, the EEG Scanner

One of EEG scanners commonly used for developing and researching purposes is Emotiv Epoc. Figure 2.3 shows the headset used for this project.

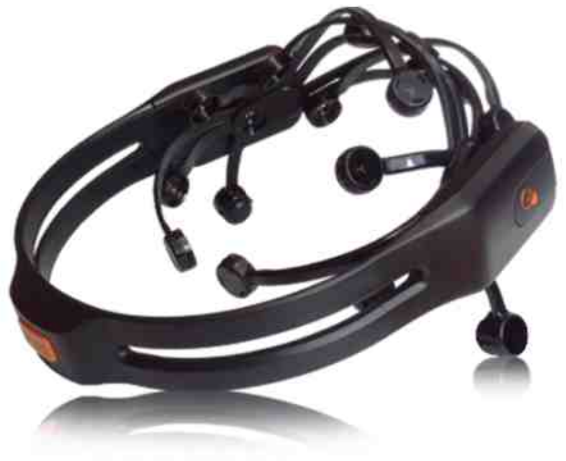


Figure 2 3: Emotiv Epoc Scanner [1]; Copied without the Consent of the Author(s).

Unlike the older methods and brain wave acquisition devices, this headset is a high fidelity EEG acquisition. There is no need for conductive gel or an expert to put it on. This device has been optimized for consumer settings and has three detecting applications called Expressive, Effective, and Cognitive.

Expressive application captures facial expression in real time such as blinking, smiling, or winking, etc.

Effective application detects emotions in real time. The levels of different emotions such as excitement, calmness, stress, frustration are shown in terms of digital values in real time.

Finally, in Cognitive application, the user has the ability to move an object by thinking about the concept. There are a number of actions that the software can be trained for such as push, pull, right, left, rotate, disappear, and so on. There is a period of 6 seconds for the device to record the base line of the brain waves. The reason for the base line training is the fact that individual's cortex is unique, and as the result, the base waves of individuals are different. The classification of actions is based on baseline of individual's brain activity. Afterwards, there is 6 seconds of training in order to record the way a person reacts to a certain action and the pattern of the signals that gets to the cortex for that person. The training data is used to train the system to classify the brain waves.

In addition to detecting applications in Emotiv Control Panel, there is software called Test Bench for showing and recording brain waves in terms of digital waves. In addition to Test Bench, there is an option to communicate with Emotiv APIs through an external program in order to collect raw brain signals.

In this project, a program written in C++ is making the API calls, requesting raw data and Emotiv API returns raw brain signals. The result is then saved in an external file for later training purposes.

2.4 Data Analysis

Data analysis is the process of discovering knowledge and modeling the data. In the process of data analysis, also called as data mining, the data is inspected, trimmed, and transformed to a form which solves a particular problem. Data analysis generally contains two main phases, initial analysis and main analysis. In initial analysis, the quality of the data is examined mostly using descriptive statistics characteristics of data, and main analysis contains more advance methods in addition to statistical methods in order to solve the problem. For Emotional state classification of EEG signals, due to the large amount of data points, descriptive statistics such as mean, standard deviation, number of peaks in time and frequency domain, in addition to observation of data in different plots are used for initial analysis. For knowledge extraction of the cleaned data, other advance data analysis methods are used. Principal Component Analysis and Fast Fourier Transform are two methods with positive results on this particular problem that are explored furthermore in section 2.4.1 and 2.4.2. The details of the transformation of data are presented in chapter 3.

2.4.1 Principal Component Analysis (PCA)

Classification of data is based on the analysis of variables of samples in data set. Inspecting and analyzing the variables individually does not necessarily help to classify the data. The correlation between the variables generates another new perspective to the data which solves various problems such as EEG emotional state classification.

Principal Component Analysis (PCA) [10] is an orthogonal projection of the data to a lower dimensional linear space. This mathematical procedure transforms a set of possibly correlated variables into a set of linearly uncorrelated variables. The result variables are called components and the number of them is less than or equal to original variables.

The new principal components are ordered by their variance from high to low, meaning that the first generated component has the most variability in the data and each subsequent component has the next highest orthogonal variance. Basically this linear transformation converts the data to a new coordinate system. The variance of any projection falls into a new coordinate called component in the descending order.

Performing PCA on data set minimizes the amount of noise and redundancy of variables, in addition to translating the data into lower dimension and presenting new information about the correlation of the variables.

2.4.2 Fast Fourier Transform

Fast Fourier Transform (FFT) [11] is a mathematical algorithm to convert a wave to its Discrete Fourier Transform (DFT) representations and its inverse. DFT is decomposition of waves into components which are coefficient of sine waves and converts the waves from time domain to Frequency domain.

If the original value of the experiment in the time domain is represented by function *s (t),* in the Fourier Transform process, a new function of frequency, *S (f),* will be generated referred to as frequency distribution. The frequency function is reversible, meaning given a function in frequency domain, the function in time domain can be generated as well. *s (t)* and *S(f)* are two functions representing the same events in time and frequency domain respectively. The transform of function *s (t)* at frequency *f* is given by the complex number:

$$S(f) = \int_{-\infty}^{\infty} s(t)\, e^{-i2\pi ft}\, dt$$

Such transformation can help in providing extra information about data for pattern recognition. Computing Fourier Transform takes *O(N2)* arithmetic operations while Fast Fourier Transform is the fast version that can perform the same process in *O(Nlog(N))* operations.

2.5 Artificial Neural Networks

Inspired by biological neural network, Artificial Neural Network (ANN) [12] is a mathematical model that consists of interconnected artificial neurons. The model is used to model complex relations between input and outputs of a classification.

ANN consists of different layers of artificial neurons which are interconnected. The input neurons are the only ones receiving data from outside of the network. The number of neurons in input layer depends on the number of features in the feature vector of sample data.

The output layer consists of output neurons, the value of which is calculated by the network using other neurons of ANN called hidden neurons. Each connection has a weight which is updated during the learning. The weights of ANN determine the values of neurons in hidden layers and output layer as a result.

The fundamental parameters in ANN construction are the topology of the interconnected neurons in different layers, the process used for changing the weights of those connections and the process for calculating the output of each neuron based on its inputs and their weights.

There are three major types of machine learning techniques namely supervised, unsupervised, and reinforcement learning [13]. The selection of machine learning paradigm depends on the problem and expected behavior of network for input data.

Supervised learning is the model to calculate relation between input and output data. In this paradigm, data labels are readily available. In other words, the input and target (expected outputs) are provided in training and sample data. ANN encodes a model of the relationship between inputs and outputs of the network. So, if X is an input to ANN and Y is an output, ANN generates function $f(x)$ where:

$$Y = f(X) + e$$

Where e is error that needs to be minimized during the training.

For each neuron, the output y is a function of the bias of the neuron (b) plus the sum of incoming connections of the activity of the input line times the weight of that line.

$$y = b + \Sigma_i x_i w_i$$

The learning starts with initial, randomized weights and the outputs are calculated based on the inputs. Afterwards, the errors of these predictions are calculated and error vector is used to update weight matrix.

This method is used for the cases in which ANN is expected to predict characteristics of data. There are two main categories for supervised learning called regression and classification. The purpose of regression is to predict a number or a vector of numbers as close to the real output, whereas in classification the label of the class that the input data belongs to is predicted.

In unsupervised learning, ANN is provided with unlabeled data. This learning tries to find structures or patterns in the data but as the data is not labeled, the learning cannot be evaluated by calculating the errors. Data processing and data mining techniques such as feature extraction, dimensionality reduction, Principal Component Analysis are considered methods of unsupervised learning to extract features of data.

Reinforcement learning is learning by interacting with environment and based on rewards. ANN learns from the consequence of its own action in new situation. The purpose of learning in this method is to maximize the amount of reward. The correct inputs and outputs are never presented to algorithms and the purpose is to select action or sequence of actions to maximize the rewards and rewards may occur occasionally.

2.6 Backward Propagation

Backward propagation [15], often referred to as back propagation, is one of the methods of ANN supervised learning. Each Feed forward back propagation algorithm has two phases namely propagation and weight update. First is propagation. This contains the forward propagation of inputs into the network in order to calculate the output and backward propagation of calculated outputs to calculate deltas on each hidden layer.

The second step is weight updates. The gradient of the weights are calculated by multiplying the nodes' activation and delta. Gradient of weights shows the area in the network with high value of error. The last step in weight updates is bringing the weights in opposite direction of its gradient to control and minimize errors. This is accomplished by subtracting gradient's ratio from the node's weight.

The weights can be updated after each pass which is called incremental (online) learning. Another method is to update weights every few passes. The later procedure is called batch learning and requires more memory capacity. Incremental learning has much more weight updates and processing involved.

Chapter 3: Technical Details

3.1 Fast Artificial Neural Network

For artificial neural network implementation, an open source library called Fast Artificial Neural Network (FANN) is used. Both fully connected and sparsely connected multilayer artificial networks can be implemented using this library.

3.2 Determining Features for Feature Vector

As mentioned in chapter 2, for any pattern recognition and machine learning, selection of the best neural network algorithms depends on the nature of data. Furthermore, another main component of classification is constructing the appropriate feature vector for inputs and outputs of classification algorithms.

Feature Vectors are n-dimensional vectors of features. Each feature is presented by one dimension. Such representation facilitates the classification process by presenting each sample data in the form of the feature vector.

Presenting ANN with raw data makes the classification time consuming, inefficient, complicated, and sometimes impossible.

In order to categorize data, features for feature vector should be selected carefully. Creating the feature vector requires analyzing the nature of the problem and sample data, and finding features and aspects of the data which distinguish various classes and can promote the classification. Basic candidate features for pattern recognition of EEG data can be statistical information about waves such as average, mean, minimum, maximum, standard deviation, number of peaks, and slopes, etc. Due to the complexity of brain waves, none of the combinations of these features were applicable for the classification.

As mentioned in the previous section, after filtering the data and removing artifacts for each of the 14 channels, there are 6,000 samples recorded in rate of 128 per second. In emotional states classification, raw data is in 15 dimensions (14 Channels and time). The process of the data analysis indicates that such a high dimensional space hides the characteristics of classes. According to a similar problem investigated in a thesis presented by Allison Marie Henderson at CSUCI (Autonomous Interoffice Delivery Robot (AIDeR) Environmental Cue Detection) [16], FANN is unable to accurately classify waves. The complexity of high dimensional data attests the need of dimensionality reduction. As a result, features are extracted from the new dimensional system and a feature vector is constructed for the classification.

More felicitous features to class attributes were found by extracting principal components of the raw data. By applying PCA procedure on raw data, binary classification of negative and positive feelings was achieved. Changing the domain from time to frequency and extracting principal component from frequency domain assisted composing applicable feature vector. More details of feature vector construction process are presented in chapter 4.

3.3 Classification Method Selection, and Neural Network Library

In order to find the best solution to any machine learning and classification problem, the performance of neural networks with various topologies and classification strategies should be examined.

In this document, numerous topologies, classification methods, error functions, and activation functions are performed, and the results are compared in order to compose the best strategy to solve EEG emotional classification problem. FANN Explorer [17], in combination with FANN Library, is used for executing training and testing.

In each experiment, the network with a certain topology is constructed. Then, the algorithm methods, error function, activation function for each neuron, in addition to other parameters are selected. Afterwards, the network is initialized, initial weights are randomized, and learning stop factor is set. The training data is shuffled most of the time for different runs of the same settings to compare the results.

After running the training, the test data is loaded and the output of the strategy is observed.

Chapter 4: Experiments

4.1 Experiment Overview

The purpose of this project is to classify human emotions into four classes namely: low arousal positive, low arousal negative, high arousal positive, and high arousal negative. Four emotions are selected, each one presenting one of the classes of this problem. In this experiment, calmness, sadness, happiness, and annoy are selected for low arousal positive, low arousal negative, high arousal positive, and high arousal negative, respectively. Videos are used for prompting associated emotional experience for data collection. Selection of videos and whether they are the best for these types of studies from Psychological perspective is not the subject of this project. This research focuses on strategies to analyze real-world data with Psychological nature from Computer Science perspective.

Figure 4.1 shows an overview of the project which includes data collection, data analysis, and neural network learning. Figure 4.2 shows the process in each section step by step. The details of each section are explained in details in the following sections.

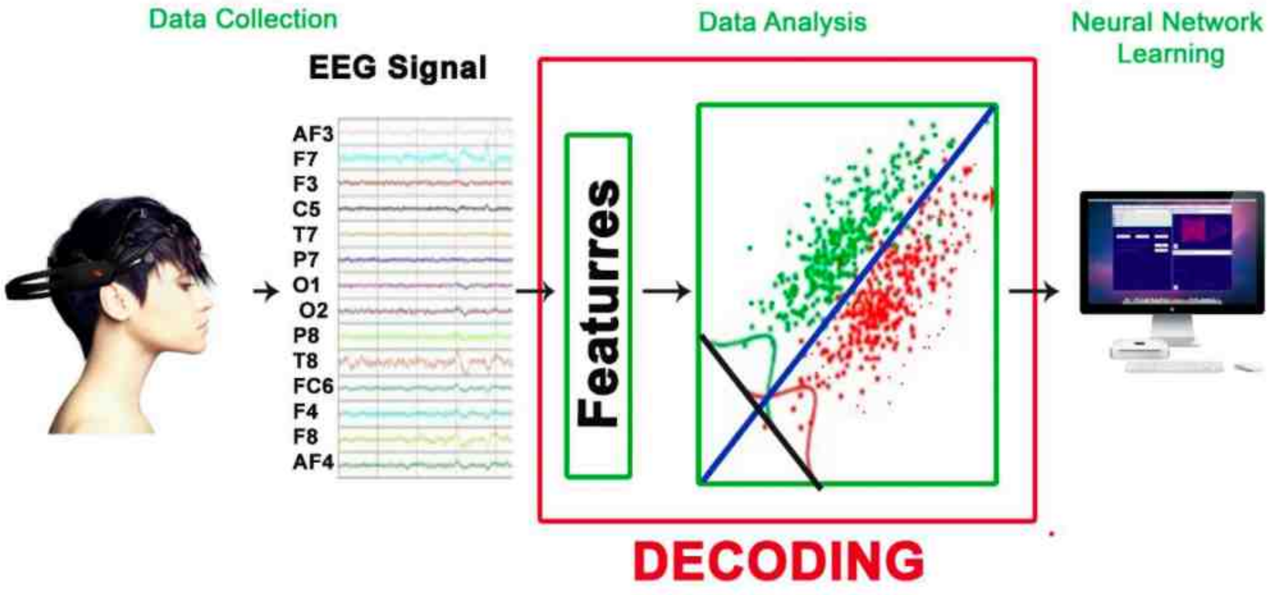


Figure 4.1: Experiment’s Overview.

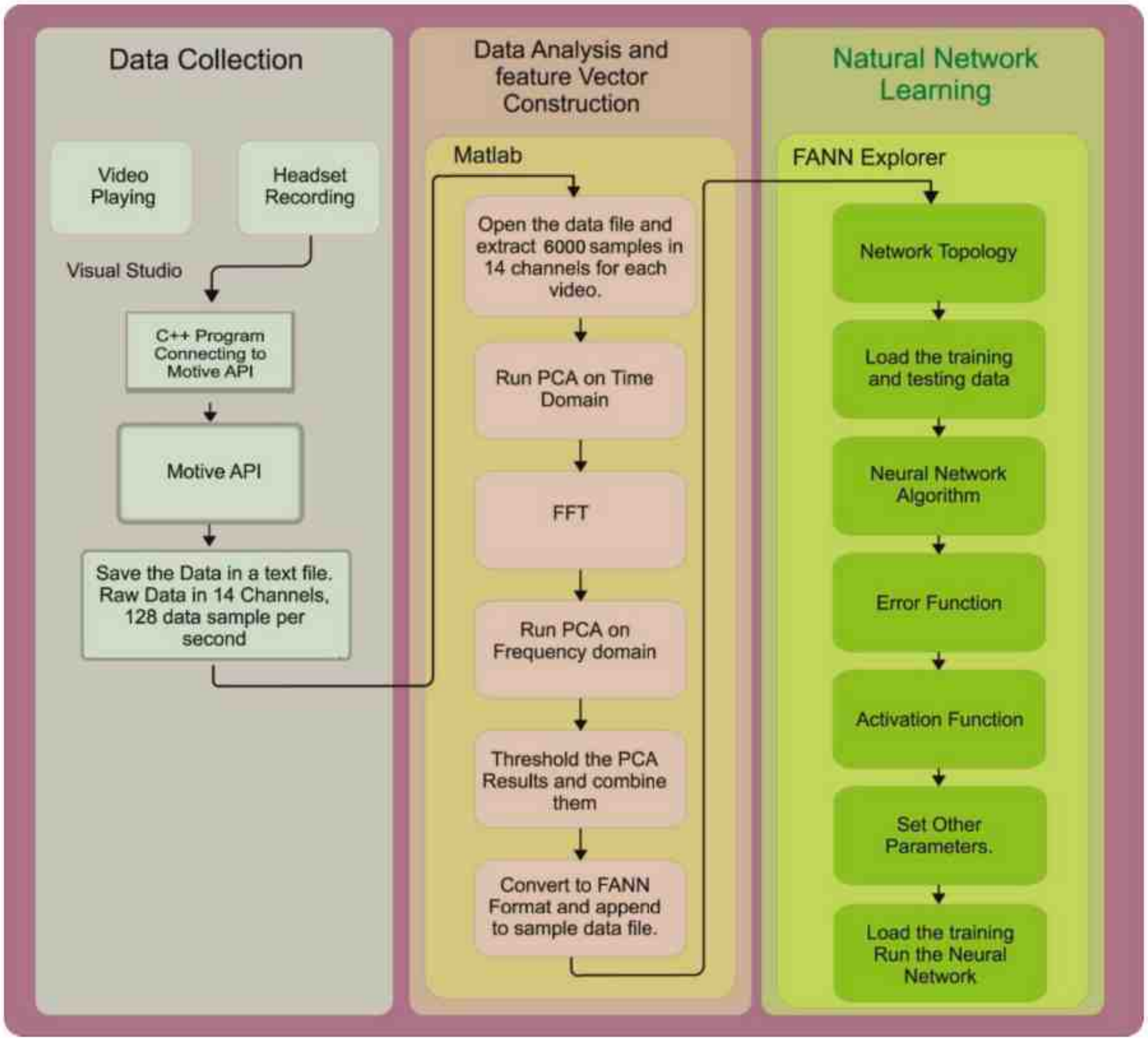

Data Collection
Video Playing
Headset Recording
Visual Studio
C++ Program Connecting to Motive API
Motive API
Save the Data in a text file. Raw Data in 14 Channels, 128 data sample per second
Data Analysis and feature Vector Construction
Matlab
Open the data file and extract 6000 samples in 14 channels for each video.
Run PCA on Time Domain
FFT
Run PCA on Frequency domain
Threshold the PCA Results and combine them
Convert to FANN Format and append to sample data file.
Natural Network Learning
FANN Explorer
Network Topology
Load the training and testing data
Neural Network Algorithm
Error Function
Activation Function
Set Other Parameters.
Load the training Run the Neural Network

Figure 4.2: Experiment's Flowchart.

4.2 Data Collection

The subjects of this experience are provided with four different video clips namely calm, sad, happy, and annoying. These clips are played in the order which they were introduced for all the subjects for the purpose of consistency in collected data. The videos for happy and sad emotions show people laughing and crying respectively. Watching happiness and sadness of other people triggers the same emotions in the subjects. For calmness movie a medication clip containing sounds in nature like sounds of birds and river is used. For annoying clip, an annoying high frequency sound is played. The feedbacks from subjects show that the selected videos were able to prompt the corresponding emotions successfully. Selected videos are available from links included in references [21].

The data used for analysis of the results chapter belongs to 10 subjects each watching 4 videos. Thus 40 sample data is generated in data collection process. For each sample data, 128 raw brain signals in micro Volt are collected for each second of the experiment. The data is saved in a separate file for each video clip and retrieved later by data analysis script to create feature vector. Finally, the data is recalled by classification program for training.

Recorded files are labeled as relax, sad, happy, and annoyed and saved in a folder labeled with the ID of the subject.

4.3 Feature Vector Construction

The raw data contains thousands of samples (from 7,000 to 27,000 data points) for each of the fourteen channels in each experiment. The number of data points varies depending on the videos. The length of videos for low arousal emotions is larger, due to the time required to prompt emotions such as calmness and sadness. Despite this, for the research, the number of samples for each of the experiments is a fixed number.

In the analysis section of feature vector construction, a script extracts the more appropriate 6,000 points of the data in each section depending on the video. The value of recorded raw EEG data is in micro Volt. An example of the structure of data is included in Appendix B. Then, the extreme peaks and slopes of the waves which are the result of noise in the experiment are removed. These values are replaced with the mean of all the points.

As the channels are correlated in each experiment, fourteen other linearly unrelated variables are extracted by applying Principal Component Analysis procedure on the raw data. First fourteen bits of the feature vector are threshold values of these principal components. As a consequence of constructing a fourteen-bit feature vector using Principle Component Analysis on raw data, binary classification of emotions into two classes of positive and negative emotional states is accomplished. In order to proceed with the classification into four classes, more analysis in frequency domain is needed.

As a result, the raw data is transferred to frequency domain by applying Fast Fourier Transform procedure. As a result, threshold values of principal components of frequency domain are calculated and fourteen bits are appended to the existing vector.

Two bits of output vector present four classes for training and test data. Furthermore, several representations of output neurons are explored in the topology section of this chapter and also in chapter 5. The next step is to convert the training and test data into FANN file format. The first line of the vector file contains three integers: number of data pairs, number of inputs and number of outputs.

Afterwards, the data pairs are presented so that each line of the input data values is followed by one line of the output data values. In other words, each input vector is followed by an output vector in the next line. All numbers are separated by blank space characters. The result vectors are then used as the input and output to neural networks. A sample of feature vector in FANN format is included in Appendix B.

4.4 Topology

Artificial neural networks are networks of neurons, similar to their biological prototypes. Network's topology is the way that these neurons are interconnected to form a network. In other words, neural network topology shows the architecture of the network and the relationship between its neurons.

Each neural network consists of an input layer and an output layer. Some additional hidden layers may be in the structure of the topology.

An input layer represents input values and consists of input neurons which get external inputs from outside of the network. The output layer contains output neurons whose values are calculated as the result of the values of other layers' neurons in the network.

Each neural network may contain one to several hidden layers. Hidden layers consist of hidden neurons which do not have any connection with the outside of the network. The input to each hidden layer can be either from input layer or other hidden layers, and the output of it is to output layer or other hidden layers.

Relationships between neurons are realized via connections between neurons in different layers, each of which is weighted. The behavior of neural network is controlled by these weights. In the process of learning, ANN's function adjusts the weights of these connections using the outputs and amount of errors in order to produce correct outputs.

ANN for emotional classification of brain waves consists of three layers: input, hidden, and output layer.

There are 28 input neurons, 28 hidden neurons, and output layer consists of two neurons in the selected topology for our experiments. Figure 4.3 and 4.4 show the topology and statistical information of normal topology respectively.

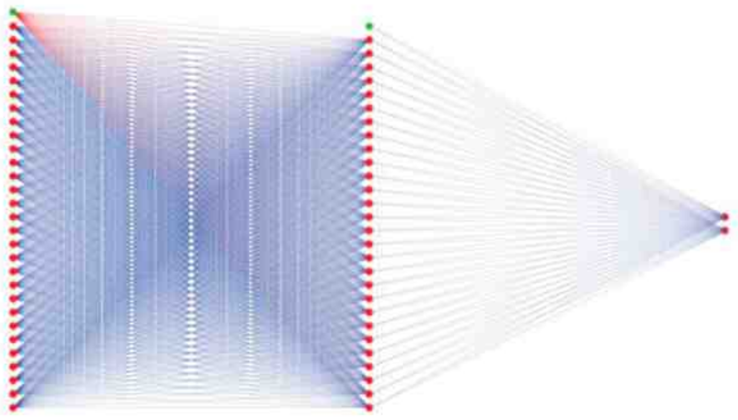
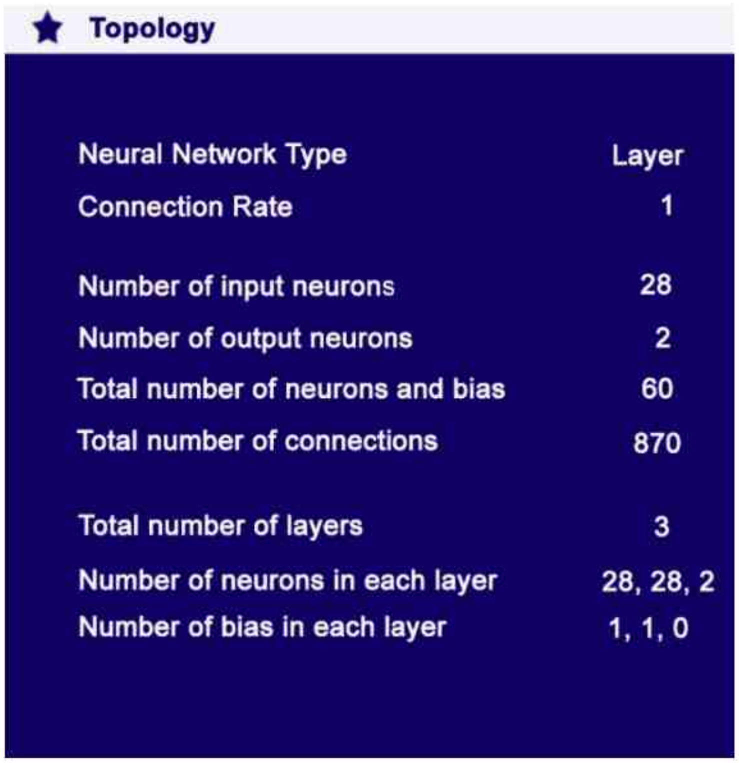


Figure 4.3: Fully Connected Network in Normal Topology. Circles represent neurons, and lines present weighted connections.



Topology

Neural Network Type	Layer
Connection Rate	1
Number of input neurons	28
Number of output neurons	2
Total number of neurons and bias	60
Total number of connections	870
Total number of layers	3
Number of neurons in each layer	28, 28, 2
Number of bias in each layer	1, 1, 0

Figure 4 4: Statistic of Normal Neural Network Topology.

The number of neurons in input layer depends on the number of features that are needed for the classification problem. As explained in the feature vector construction section, feature vector of EEG emotional state classification problem has 28 bits, and as a result, there are 28 input neurons.

Output neurons represent the resulting classes of the classification. Two bits are adequate to present four distinct classes of this problem. Networks with three output neurons and four output neurons are also examined in order to ensure that the learning performance is independent of output representation. More details of network's performances with various numbers of neurons in output and hidden layer are presented in chapter 5.

In order to avoid guessing the best topology and the number of hidden layers prior to training, cascade training can be used. Unlike normal training in which the full topology containing input, output, and hidden layer is established beforehand, cascade training starts with an empty neural network. Hidden neurons are added to the network one by one while training is in progress.

In this method, the most promising of some separately trained neurons are inserted into the neural network. In each step of training, output connections are trained and a new hidden layer with a candidate shortcut neuron is created. This means the final neural network will consist of a number of hidden layers with one shortcut connected neuron in each. The result of applying this method is a topology that consists of 28 input nodes, 1 hidden layer with the shortcut neuron, and output layer with 2 output nodes as shown in Figure 4.5. Figure 4.6 presents the statistics of cascade topology.

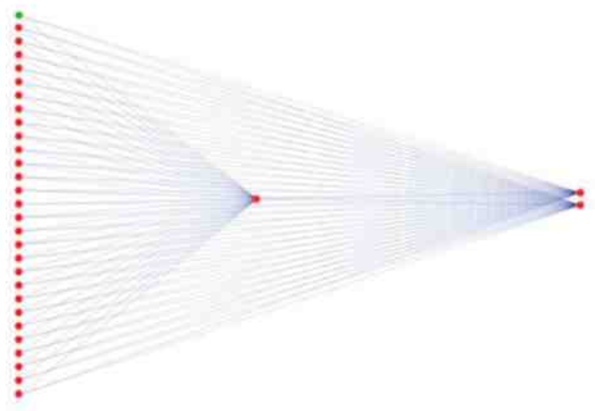
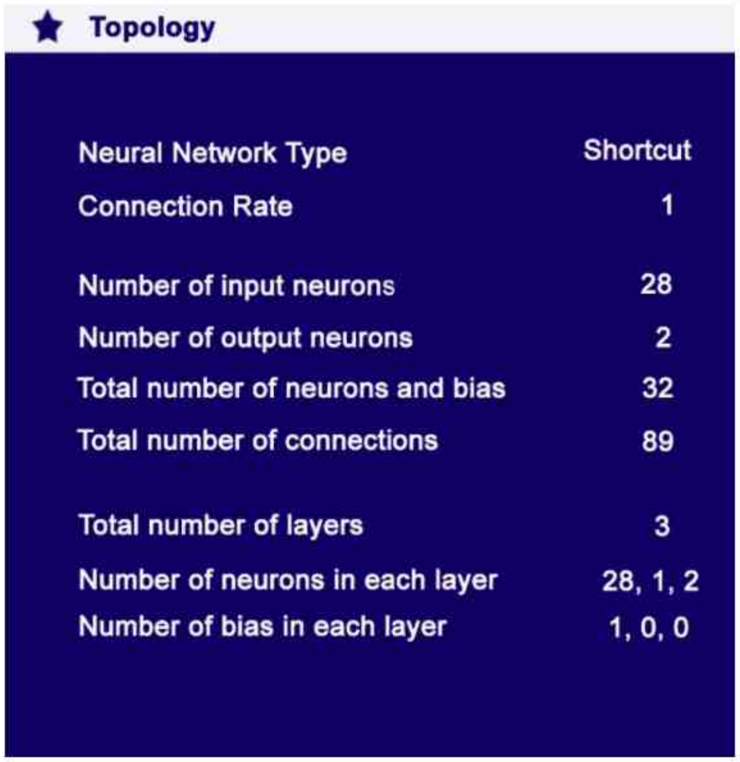


Figure 4 5: Cascade Training Topology: Fully connected network. Circles represent neurons, and lines present weighted connections. Blue lines are positive weights, and red lines are negative weights.


Topology

Neural Network Type	Shortcut
Connection Rate	1
Number of input neurons	28
Number of output neurons	2
Total number of neurons and bias	32
Total number of connections	89
Total number of layers	3
Number of neurons in each layer	28, 1, 2
Number of bias in each layer	1, 0, 0

Figure 4 6: Statistic of Cascade Neural Network Topology.

Suggested neural network of cascade training has only one shortcut neuron in the hidden layer which saves calculation time. Unfortunately, the topology is not efficient due to its slow learning. The number of epochs that the cascade network needs in order to drop the error rate below the threshold is much larger than the number of epochs needed in the suggested topology. More statistical discussions of these comparisons are presented in chapter 5.

As a result, the topology with 28 hidden neurons and 2 output neurons is used for the rest of the experiment.

4.5 Algorithms

During training of a neural network, the weights of its connections are adjusted using functions and according to the error that those functions yield in the outputs' predictions. This procedure is called training algorithm. There are several training algorithms, each of which accepts a different set of parameters such as learning rate, momentum, etc. In this experiment several algorithms are applied to the input data and the results are compared in chapter 5.

Training process utilized both incremental and batch method. Incremental method is a standard back propagation algorithm, in which the weight of each connection is continuously updated after each training pattern. Weights of connections are changed many times in each epoch. On the other hand, in batch learning, the weights are changed after calculating the mean square error for the whole training set. In other words, weights are updated once after each epoch.

Depending on the type of algorithms used, different error functions should be used. For incremental and cascade training algorithms, linear error function is used. Whereas for other algorithms, Hyperbolic Tangent (Tanh) error function is used. Tangent error function requires a lower learning rate and it aggressively targets outputs with large error values.

The learning rate is used to determine how aggressive training should be. Learning rate factor applies a portion of respective adjustment to the old weight. The greater the value of this factor is, the faster learning will be.

The next factor is algorithm's learning momentum. Momentum controls the magnitude of persistence in learning. Having momentum set to a value between 0 and 1, a fraction of the previous weight change is added to the current update. Using this factor helps the learning to prevent converging to local minima and maxima but high values of it may affect the learning in a negative way.

4.6 Activation Function

During the learning process, each neuron uses a function referred to as 'Activation Function' to produce its output. The entire training can use a single activation function but the activation function can also be set for each layer or each individual neuron.

In this experiment, linear, threshold, sigmoid, Gaussian, sine symmetric, and cosine symmetric activation functions are examined. Afterward, the results of these functions are compared to discover the best activation function for this particular feed forward, back propagation algorithm. Activation steepness - which determines the speed of activation going from minimum to maximum - is set to the middle; which is 0.5.

The other resilient propagation advanced parameters such as increase factor, decrease factor, and minimum and maximum delta is set depending on the algorithms and activation functions.

4.7 Training

After creating feature vector, deciding about neural network's topology, creating the network, and deciding about the activation and error functions and other parameters, the actual learning process starts. In the back propagation training process, all the connections have initial weights. These initial weights should be randomized before each training process. Default range of the weights is between -0.1 and 0.1, while these values can be changed manually. Before training starts, neural network needs to be provided with the sample data.

Resetting neural network randomizes the weights in a given interval and clamps the input values to input neurons. In multiple training attempts, reshuffling the training data also assists the training.

Training algorithms need stopping condition as well as other parameters. Mean Square Error (Erms) threshold is the parameter to stop training. In these experiments, the mean square error is reported after each epoch and training stops when mean square error value goes under this parameter. Extremely low values of stopping parameter may cause over-learning. In our experiments, the value of this parameter is set to 0.001.

4.8 Testing

In order to evaluate the performance of neural networks, testing procedure is performed. In the testing process, the network is presented with a new set of data which was not part of training data. The testing sets represent general cases. If the network performs well on the test data, it is expected to generally respond well to any real world data.

In the test procedure, the neural network is run with weight values which are results of training process and the outputs of the network are computed as during the training. The desired outputs are then compared with the actual outputs and real mean square of error is calculated.

4.9 Cross-validation

In order to avoid over-learning, the training and test data is switched and the performance of the networks is compared in each case.

Chapter 5: Analysis of Results

In this section, various topologies, learning methods, error functions, and activation functions are run over sample data and test data. Moreover, the statistical analysis and graphical presentations of all the observations are presented and the results are compared in order to find the best strategy for solving EEG emotional state classification problem.

5.1 Network's Topology

This section presents statistical discussions for topology selection. Various topologies with different numbers of neurons in their hidden and output layers are examined to ensure the best performance of the neural network.

5.1.1 Input Layer

Input layer receives its data from outside of the network. The input to each individual neuron in this layer is a bit (0 or 1) of sample data. The input layer depends on the feature vector of EEG emotional state classification. The feature vector of this problem has 28 bits. Therefore, at least 28 input neurons are needed.

As additional neurons in the fully interconnected neural network add a lot of calculation time and memory usage, adding unnecessary nodes is avoided.

5.1.2 Output Layer

Output layer represents EEG emotional state classes. The numbers of neurons in the output layer depends on the number of bits required for output representation. As stated before, two bits are sufficient to represent four classes. The presentation of classes can also be performed with three or four bits. Despite the additional calculation time for each data pattern, the performance of neural networks with those different topologies is evaluated in order to find out if the learning time (number of epochs) is independent of the output presentation.

In order to examine the role of output representation, neural networks with various numbers of outputs are constructed and run over the data samples in the context of the feature vector. Training and test data should have structures corresponding to the selected topology of neural networks. In order to run the training and testing process for each new network, the sample data need be rearranged to be compatible with the new topologies.

5.1.2.1 Two Output Nodes Presentation

For this experiment, a network with two output nodes was constructed. The output presentations of the sample data for individual patterns are as follow:

Low arousal positive:	00
High arousal positive:	01
Low arousal Negative:	10
High arousal negative:	11

The neural network topology with two outputs is shown in Figure 4.3. More details of outputs of the network with this topology using various parameters are presented in the following sections.

5.1.2.2 Three Output Nodes Presentation

Next, the format of training and test data were modified to be compatible with the new neural network which is consisting of three output nodes. The new output presentations are as follow:

Low arousal positive:	000
High arousal positive:	100
Low arousal Negative:	010
High arousal negative:	001

Figure 5.1 illustrates neural network Topology with three output nodes.

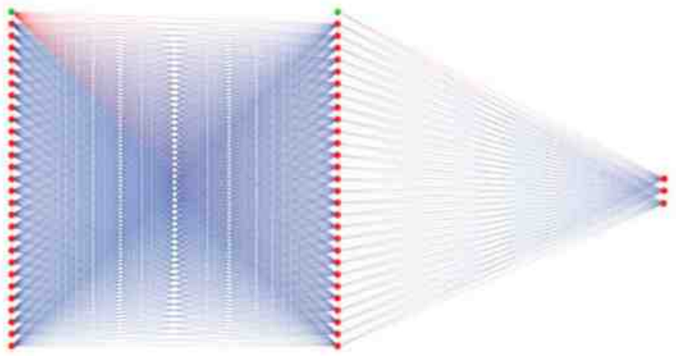


Figure 5.1: Fully Connected Network in Normal Topology with 3 Output Neurons. Circles represent neurons and lines present weighted connections. Blue lines are positive weights, and red lines are negative weights.

The performance of network with this topology was evaluated for both incremental and batch algorithms. Figure 5.2 shows the comparison between the output values of output neurons and the target values when the training and test processes were completed. These experiments were using incremental algorithm in combination with the linear error function and Gaussian activation function.

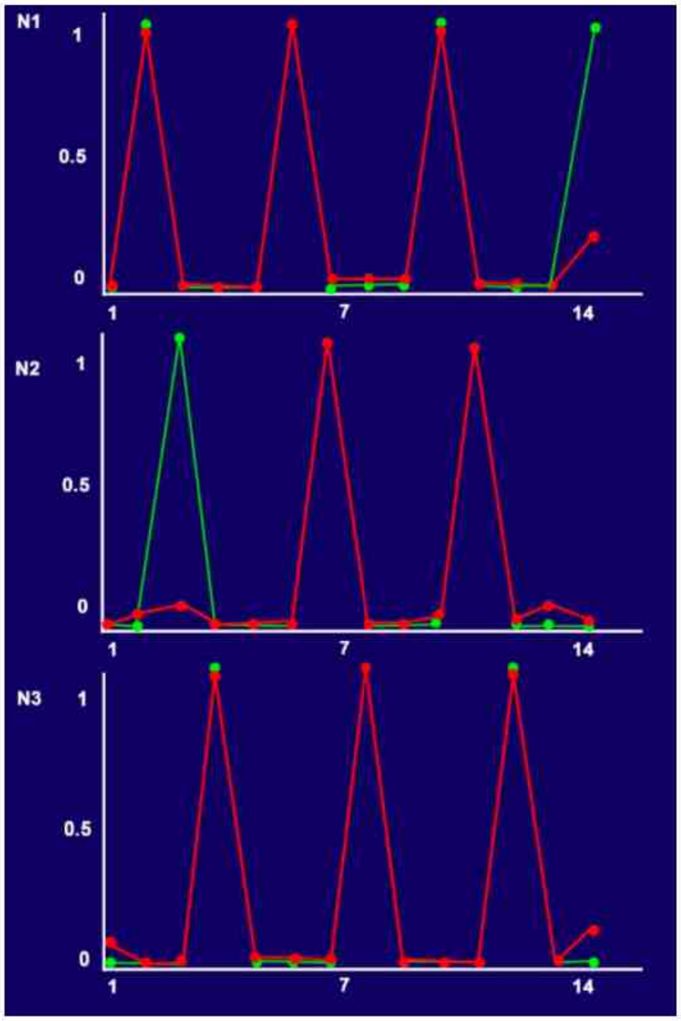


Figure 5.2: Calculated Output vs Desired Output after training in Incremental algorithm of a network with 3 output neurons. Horizontal bar shows different samples and vertical bars show value of outputs. Green dots demonstrate desired output, and red dots calculated outputs by network.

Figure 5.3 shows the comparison between output values of output neurons and their target values after the training and test process. Here, the batch algorithm in combination with tanh error function and Gaussian activation function were used.

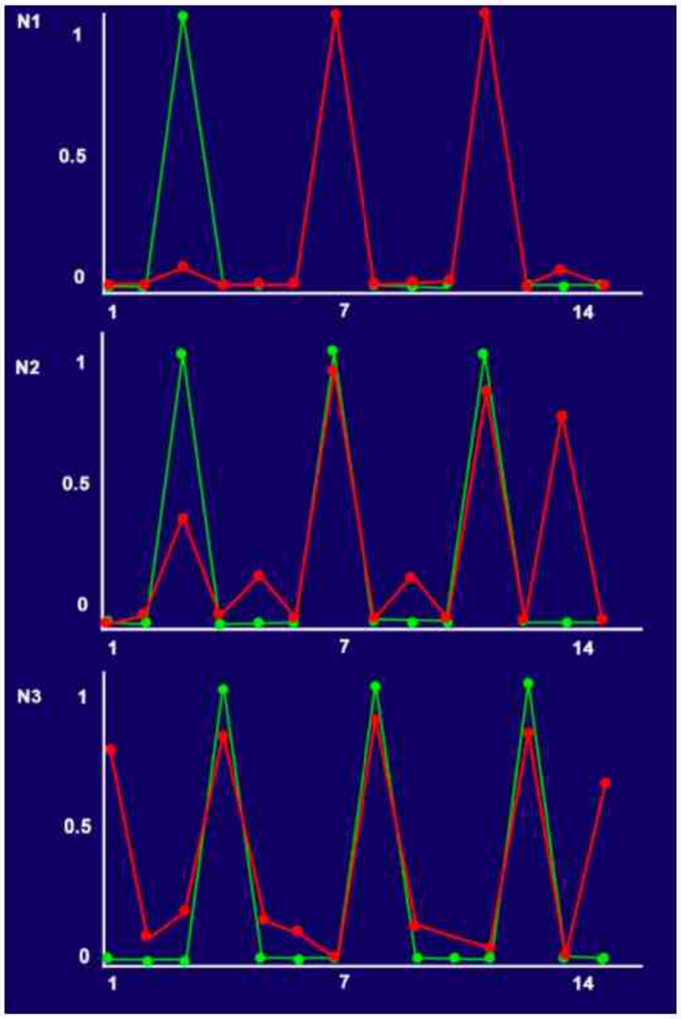


Figure 5.3: Calculated Output vs. Desired Output after training in Batch algorithm of a network with 3 output neurons. Horizontal bar shows different samples and vertical bars show value of outputs. Green dots demonstrate desired outputs, and red dots calculated outputs by network.

5.1.2.3 Four Output Nodes Presentation

Finally, the last examined format was four bits representation of four EEG emotional state classes. The following shows the output formats of data patterns which are compatible with a neural network consisting of four output neurons.

Low arousal positive: 1000

High arousal positive: 0100

Low arousal Negative: 0010

High arousal negative: 0001

Figure 5.4 illustrates the neural network topology with four output nodes.

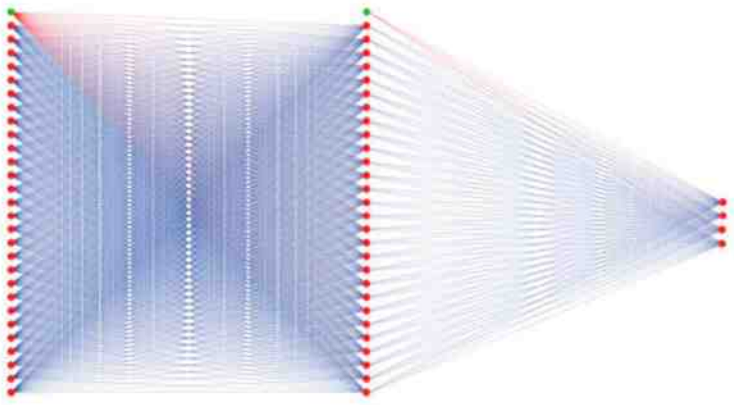


Figure 5.4: Fully Connected Network in Normal Topology with 4 Output Neurons. Circles represent neurons and lines present weighted connections. Blue lines are positive weights and red lines are negative weights.

The performances of learning with a topology which consists of four output nodes were also evaluated for both the incremental and batch algorithms. Figure 5.5 shows the comparison between output values of output neurons and their target values after the training and test process. The incremental algorithm in combination with the linear error function and Gaussian activation function was used for these experiments.

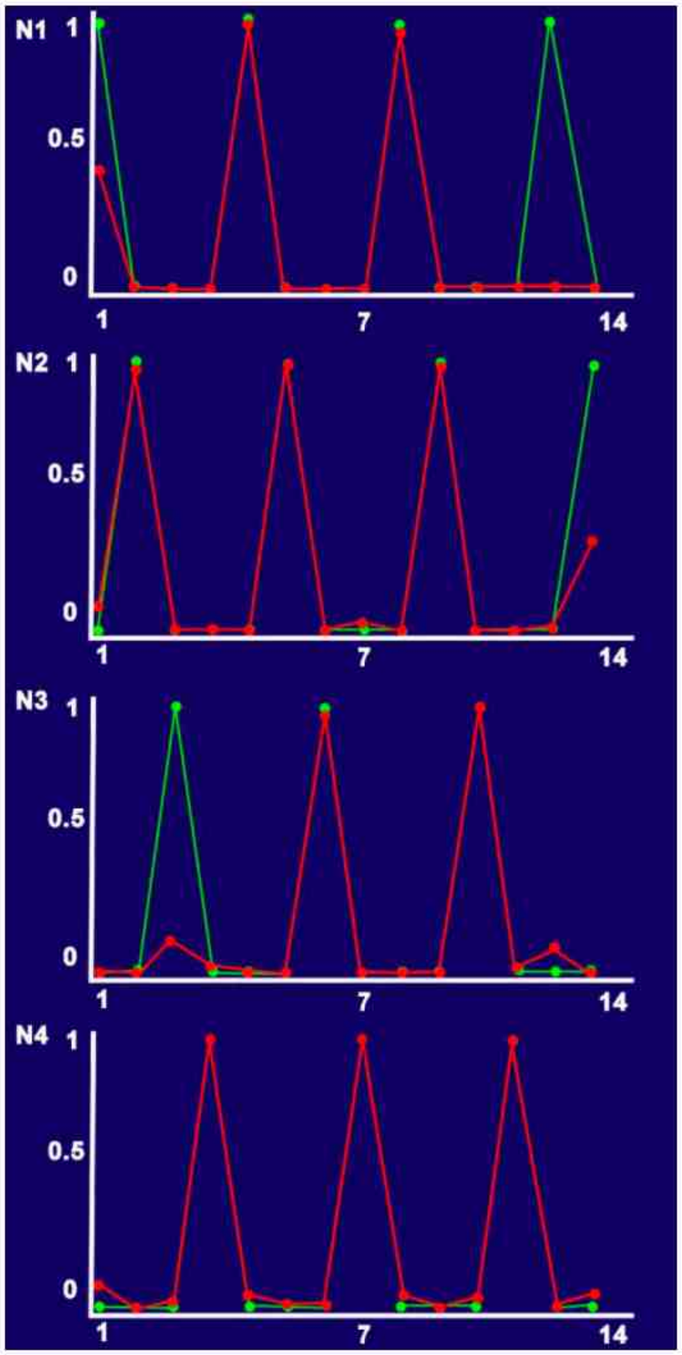


Figure 5.5: Calculated Output vs Desired Output after training in Incremental algorithm of a network with 4 output neurons. Horizontal bar shows different samples and vertical bars show value of outputs. Green dots demonstrate desired outputs, and red dots calculated outputs by network.

Figure 5.6 shows the comparison between output values of output neurons and their target values after the training and test process. The network used the batch algorithm in combination with tanh error function and Gaussian activation function.

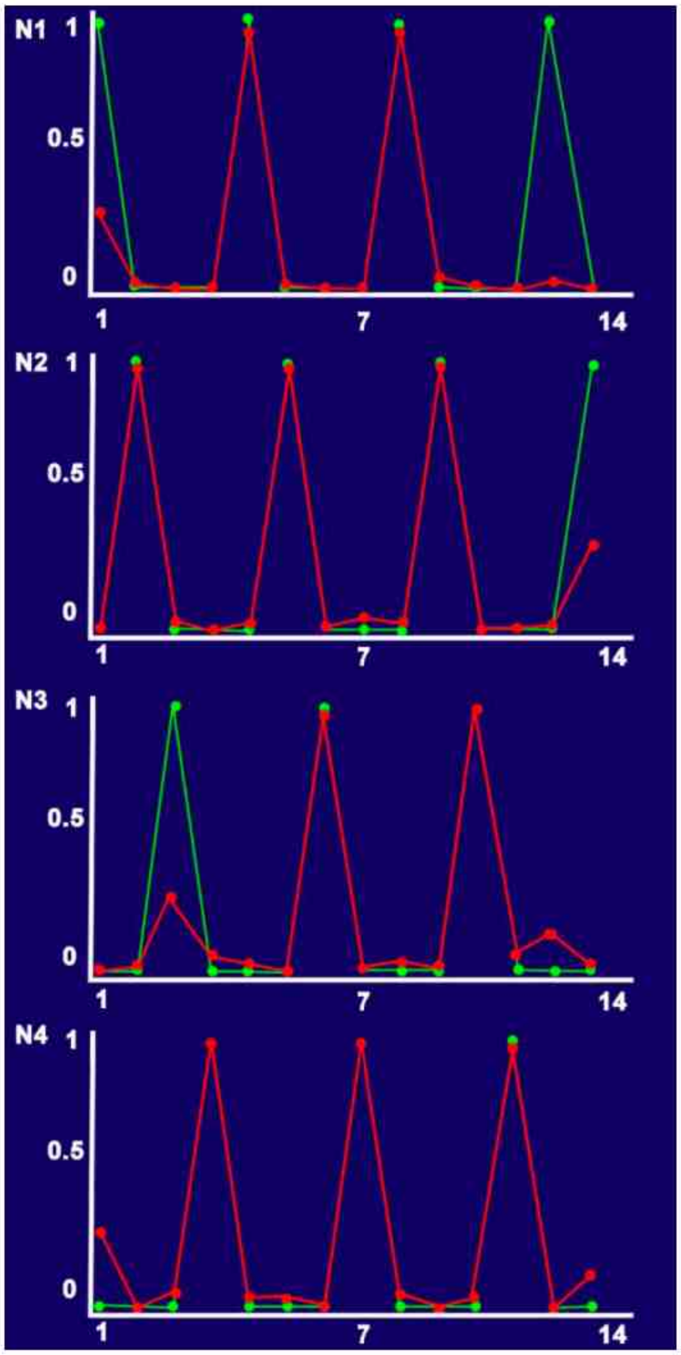


Figure 5.6: Calculated Output vs Desired Output after training in Batch Algorithm of a network with 4 output neurons. Horizontal bar shows different samples and vertical bars show value of outputs. Green dots demonstrate desired outputs, and red dots calculated outputs by network.

5.1.2.4 Conclusion

In the presented experiments, the learning performance of all three networks was evaluated for in both incremental and batch algorithms. Figure 5.7 presents Mean Square Error during the trainings of neural networks with two, three, and four output neurons using the incremental algorithm, linear error function, and Gaussian activation function. As it is illustrated in the figure, the performances of all the neural networks are similar.

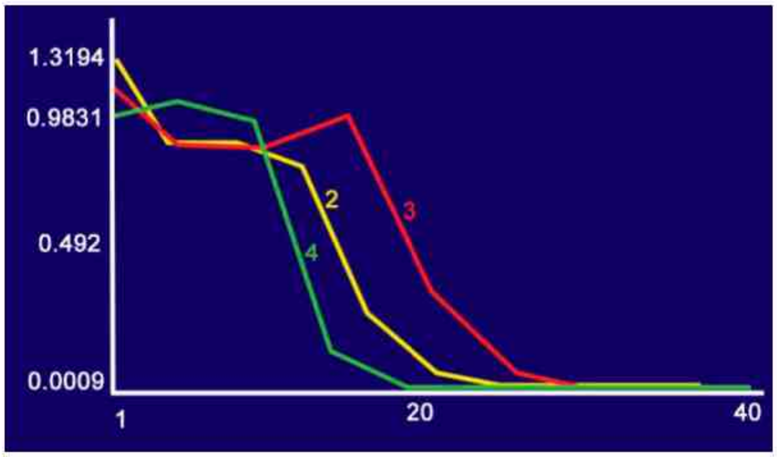


Figure 5.7: Output Layer. Comparison of Mean Square Errors during Incremental Algorithm, Linear Error Function, and Gaussian Activation Function. Horizontal bar presents epochs and vertical bar presents mean square of error. The number on each diagram shows the number of outputs of the network.

Figure 5.8 shows the learning performances of all three networks using the batch algorithm, tanh error function, and Gaussian activation function. As it is illustrated in the figure, the behaviors of all the topologies are similar. All the networks are unable to learn in 200 epochs but the mean square of error is decreasing.

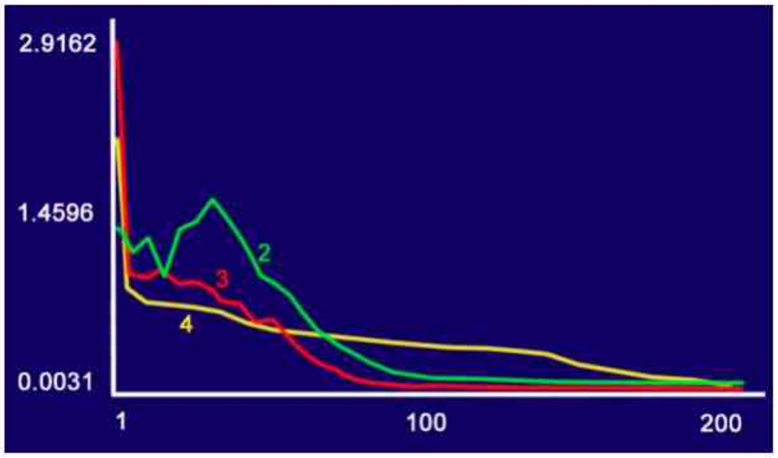


Figure 5 8: Output Layer. Comparison of Mean Square Errors during Batch Algorithm, Tanh Error Function, and Gaussian Activation Function. Horizontal bar presents epochs and vertical bar presents mean square of error. The number on each diagram shows the number of outputs of the network.

More algorithms and methods were explored in order to examine correlations between output representations and learning performances. Due to the similarity in the results, other methods are not included in this document.

As a result, the general behavior of the networks is similar in all the three cases. However, the calculation time of neural network's learning for each input pattern increases by adding extra nodes and connections. The calculation time increases because in the fully connected network, all the connections from hidden layer to output layer need value propagations, error calculations, and weight updates in each case. Thus, the network with 2 output neurons is selected to minimize the calculation time for extra nodes.

5.1.3 Hidden Layer

There are no predefined rules or methods for determining the numbers of hidden layers and the numbers of neurons in each layer. In order to determine the optimal numbers of hidden layers and hidden neurons, various topologies need to be examined to select an appropriate approach. In this section, performance of several hidden topologies is evaluated and the optimal solution is presented.

5.1.3.1 Single Hidden Layer Neural Network

In the experiments described in this section, neural networks with one hidden layer consisting of various numbers of hidden neurons are examined. Due to the similarity in the performance of the networks, which have similar numbers of hidden neurons, some hypothetical topologies were chosen to examine the behavior of networks with various topologies. In our experiments, networks with one hidden layer consisting of namely, one, seven, fourteen, twenty eight, and thirty five neurons are constructed. For each experiment, the training and testing process were performed in both incremental and batch algorithms. The performance and the error generated by these networks are compared here.

5.1.3.1.1 Single Neuron in the Hidden Layer

Figure 5.9 illustrates the topology of neural network with 28 input neurons, one hidden neuron, and two output neurons.

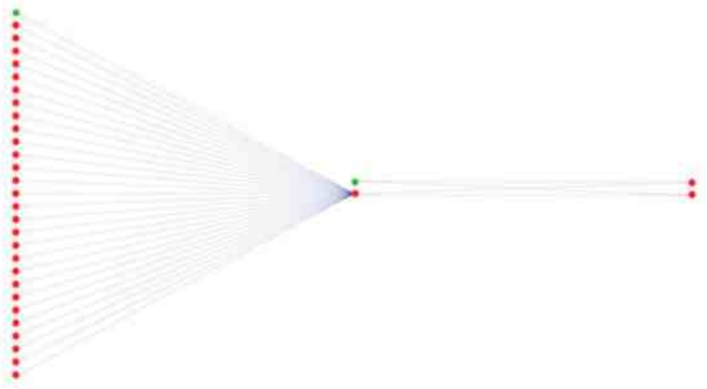


Figure 5 9: Fully Connected Network in Normal Topology with 1 Hidden Neuron and 2 Output Neurons.
Circles represent neurons and lines present weighted connections. Green neurons are the bias of the network. Blue lines are positive weights and red lines are negative weights.

The performance of the neural network is observed during both incremental and batch learning algorithms. Figure 5.10 shows the result of training with the incremental algorithm, linear error function, and Gaussian activation function. As illustrated in the figure, the network fails to learn with this method. The result of training with the batch algorithm in combination with tanh error function and Gaussian activation function is shown in Figure 5.11. As it can be seen in Figure 5.12, the resulting trained network is unable to predict any of the target values of test data correctly.

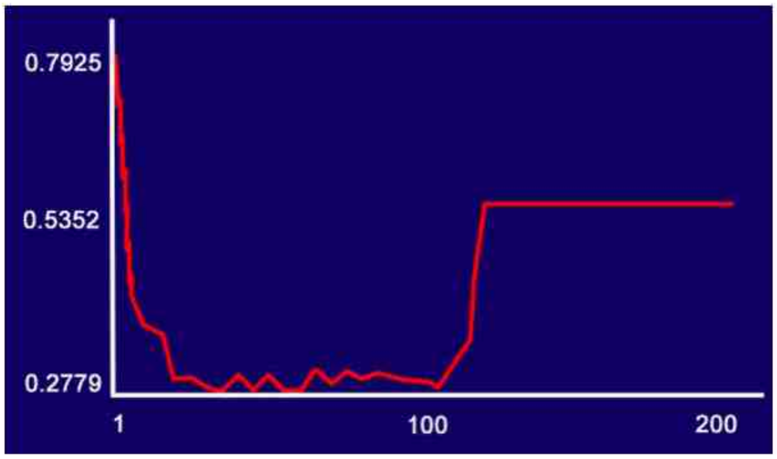


Figure 5.10: Mean Square Error during Training of Network with 1 hidden Neuron with Incremental Algorithm.
Horizontal bar presents epochs and vertical bar presents mean square of error.

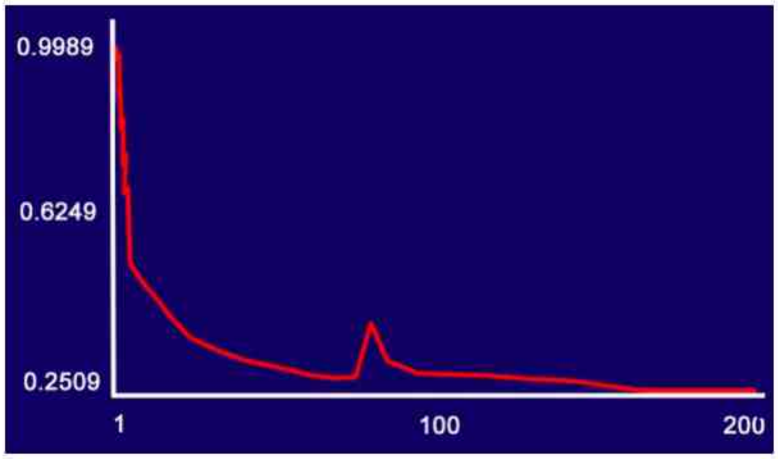


Figure 5.11: Mean Square Error during Training of Network with 1 hidden Neuron with Batch Algorithm.
Horizontal bar presents epochs and vertical bar presents mean square of error.

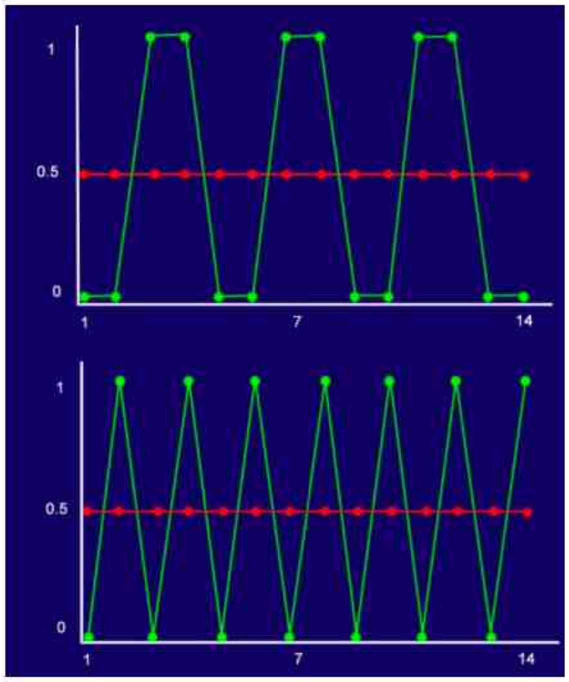


Figure 5.12: Calculated Output vs Desired Output after training in Incremental algorithm of a network with 1 hidden neuron. Horizontal bar shows different samples and vertical bars show value of outputs. Green dots demonstrate desired output, and red dots calculated outputs by network.

5.1.3.1.2 Seven Neurons in the Hidden Layer

As it was shown in the previous section, the topology with a single hidden neuron could not calculate the correct targets. So, in this section, the number of hidden neurons is increased in the single hidden layer and the performance of the neural network is evaluated.

Figure 5.13 illustrates the network with 7 neurons in the hidden layer.

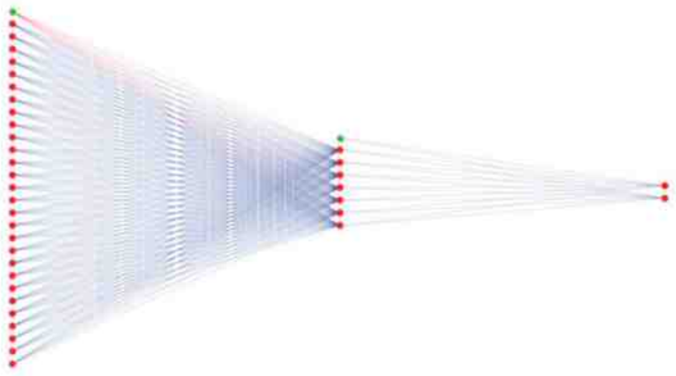


Figure 5.13: Fully Connected Network in Normal Topology with 1 Hidden Layer Consisting 7 Neurons and 2 Output Neurons. Circles represent neurons and lines present weighted connections. Green neurons are the bias of the network. Blue lines are positive weights and red lines are negative weights.

Figures 5.14 and 5.15 show the performance of the network over incremental and batch algorithms respectively. As it is demonstrated in the diagrams, the network learns after 39 epochs using the incremental method and after 469 epochs using the batch strategy.

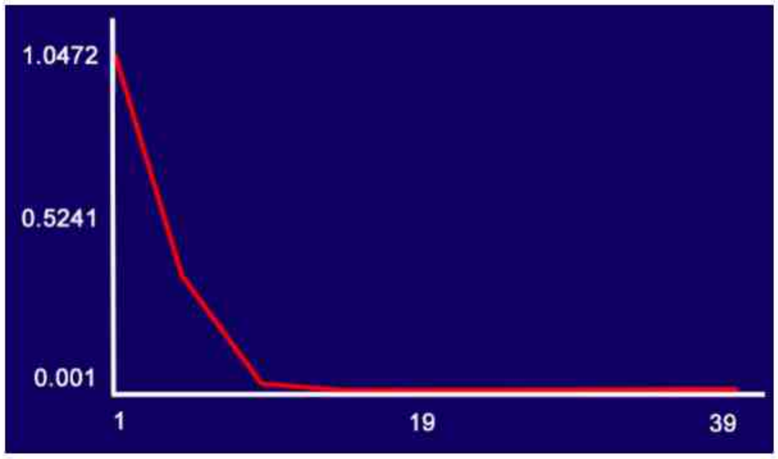


Figure 5.14: Mean Square Error during Training of Network with 1 hidden Layer Consisting 7 Neurons with Incremental Algorithm. Horizontal bar presents epochs and vertical bar presents mean square of error.

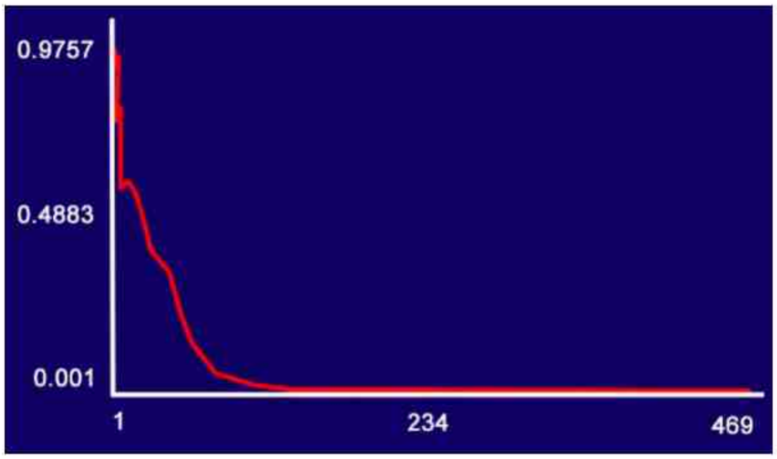


Figure 5.15: Mean Square Error during Training of Network with 1 hidden Layer Consisting 7 Neurons with Batch Algorithm. Horizontal bar presents epochs and vertical bar presents mean square of error.

5.1.3.1.3 Fourteen Neurons in the Hidden Layer

As it was shown in the previous section, increasing the number of neurons from one to seven boosts the learning performance of the neural network. In order to investigate the accuracy of the trend and to find optimal numbers of hidden neurons, a network with 14 hidden neurons was examined with both strategies. Figure 5.16 shows the topology of this network.

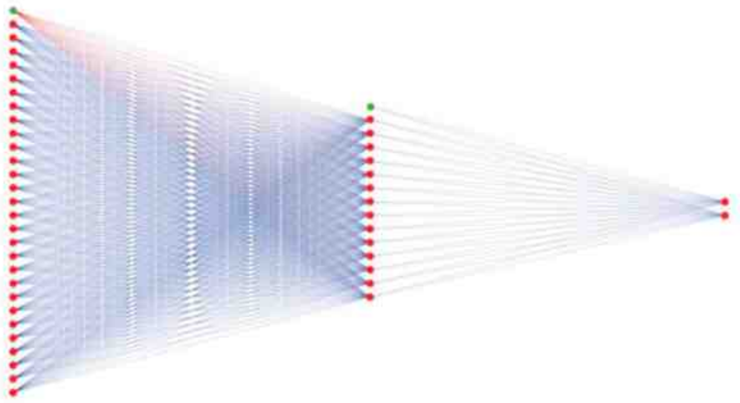


Figure 5.16: Fully Connected Network in Normal Topology with 1 Hidden Layer Consisting 14 Neurons and 2 Output Neurons. Circles represent neurons and lines present weighted connections. Green neurons are the bias of the network. Blue lines are positive weights and red lines are negative weights.

Figures 5.17 and 5.18 show the outcome of Mean Square Errors of learning during the incremental and batch algorithms, respectively. Figure 5.17 shows that the learning time (number of epochs) of the network is 52 epochs, which is larger in comparison with the previous experiment with seven hidden neurons.

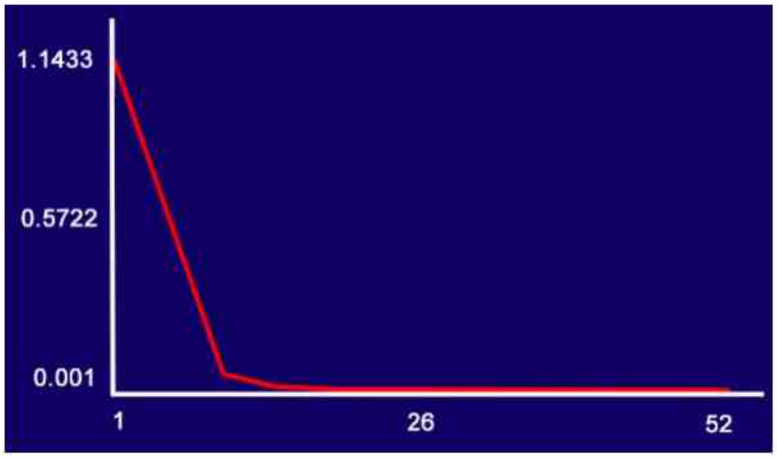


Figure 5.17: Mean Square Error during Training of Network with 1 hidden Layer Consisting 14 Neurons with Incremental Algorithm. Horizontal bar presents epochs and vertical bar presents mean square of error.

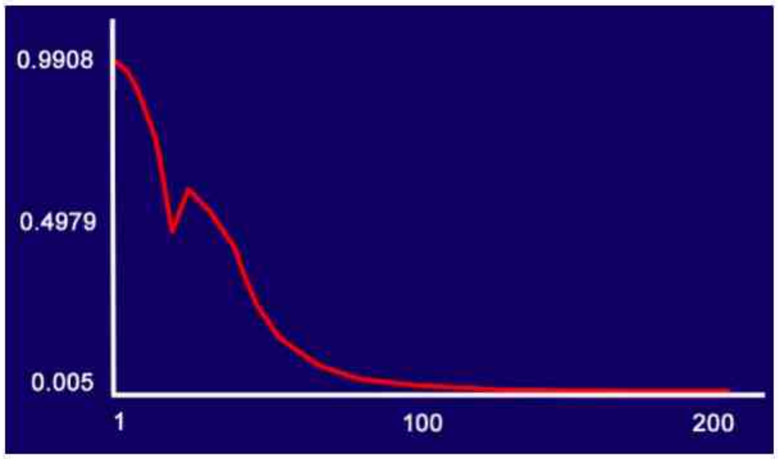


Figure 5.18: Mean Square Error during Training of Network with 1 hidden Layer Consisting 14 Neurons with Batch Algorithm. Horizontal bar presents epochs and vertical bar presents mean square of error.

5.1.3.1.4 Twenty Eight Neurons in the Hidden Layer, and Beyond

In this section, neural networks with 28 and 35 neurons in a single hidden layer are examined and the results are presented.

Figure 5.19 shows a topology with a single hidden layer with 28 neurons. Figure 5.20 which shows the performance of the neural network using the incremental algorithm is followed by Figure 5.21 illustrating the performance of the same network using batch algorithm. As it can be seen in these diagrams, the neural network with 28 hidden neurons has a better performance comparing to the previous experiment.

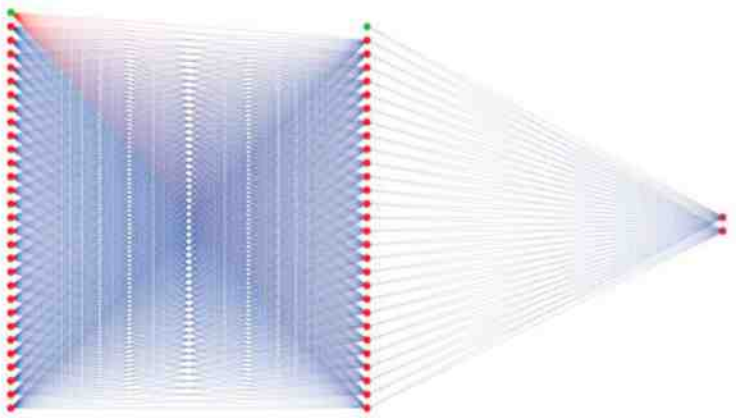


Figure 5.19: Fully Connected Network in Normal Topology with 1 Hidden Layer Consisting 28 Neurons and 2 Output Neurons. Circles represent neurons and lines present weighted connections. Green neurons are the bias of the network. Blue lines are positive weights and red lines are negative weights.

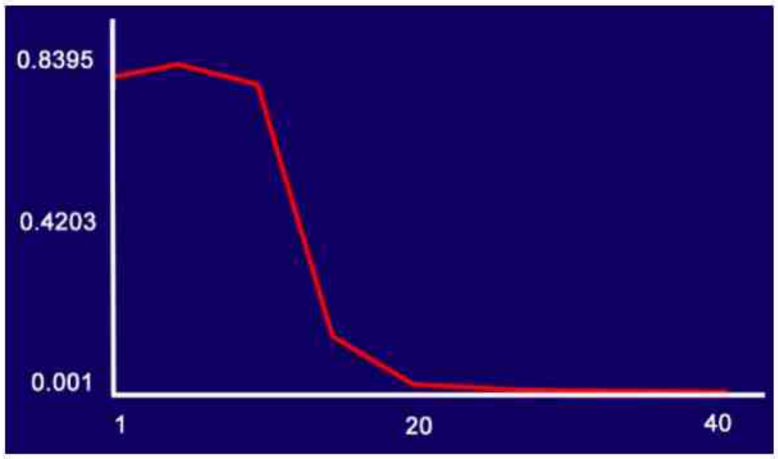


Figure 5 20: Mean Square Error during Training of Network with 1 hidden Layer Consisting 28 Neurons with Incremental Algorithm. Horizontal bar presents epochs and vertical bar presents mean square of error.

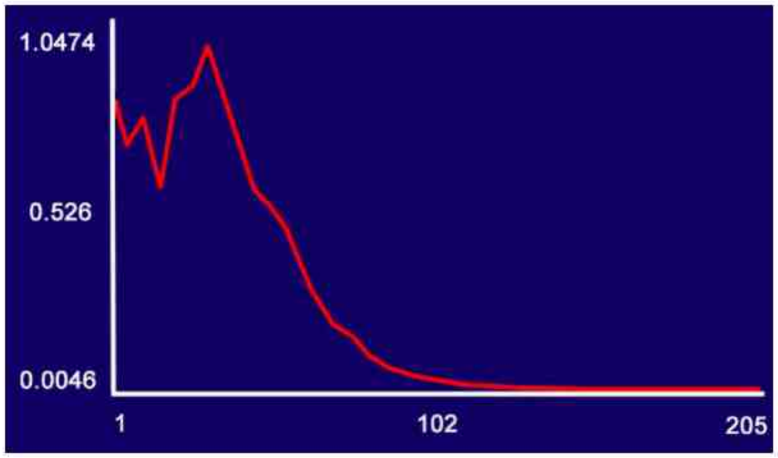


Figure 5 21: Mean Square Error during Training of Network with 1 hidden Layer Consisting 28 Neurons with Batch Algorithm. Horizontal bar presents epochs and vertical bar presents mean square of error

Finally, the experiment was repeated with 35 hidden neurons in single hidden layer. The topologies of the network, the performance using the incremental and batch algorithms are presented in Figures 5.22, 5.23, and 5.24, respectively.

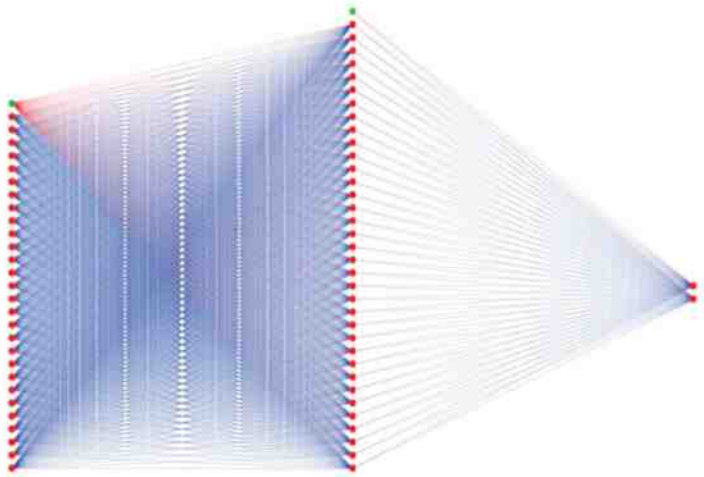


Figure 5.22: Fully Connected Network in Normal Topology with 1 Hidden Layer Consisting 35 Neurons and 2 Output Neurons. Circles represent neurons, and lines present weighted connections. Green neurons are the bias of the network. Blue lines are positive weights, and red lines are negative weights.

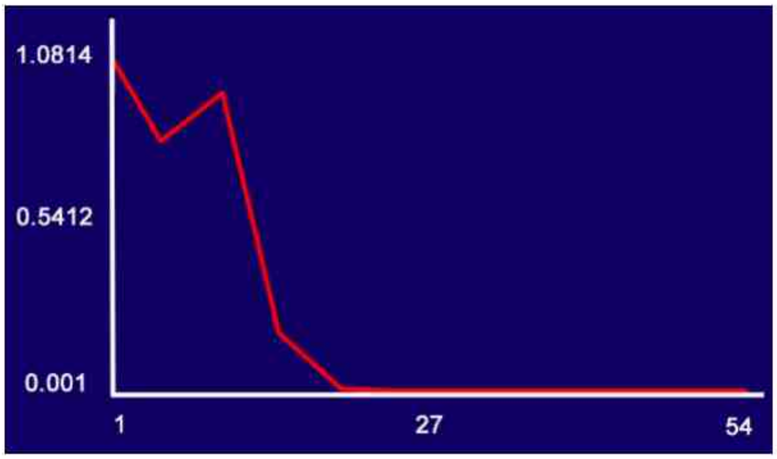


Figure 5 23: Mean Square Error during Training of Network with 1 hidden Layer Consisting 35 Neurons with Incremental Algorithm. Horizontal bar presents epochs, and vertical bar presents mean square of error.

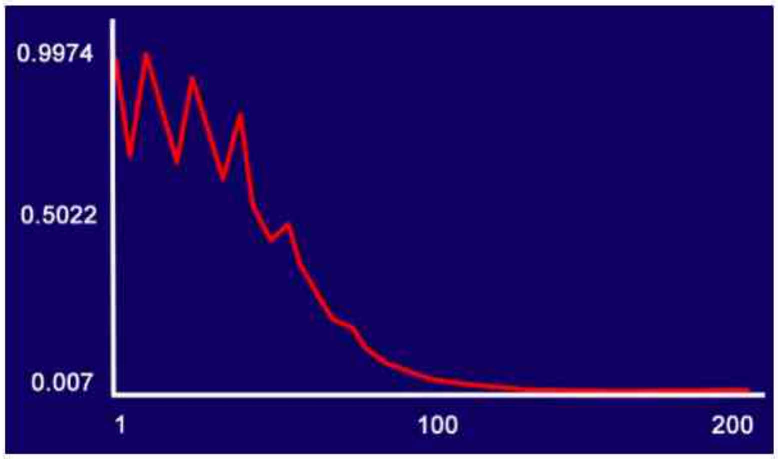


Figure 5 24: Mean Square Error during Training of Network with 1 hidden Layer Consisting 35 Neurons with Batch Algorithm. Horizontal bar presents epochs, and vertical bar presents mean square of error.

5.1.3.1.5 Conclusion

In this section, the performance of neural network with various numbers of hidden neurons was compared. The results of these comparisons with the incremental and batch algorithms can be individually seen in Figure 5.25 and Figure 5.26.

As the incremental comparison figure shows (Figure 5.25), there is no learning for a network with a single hidden neuron. The learning time and Mean Square Error of neural networks increase when adding hidden neurons, except for the network with 28 hidden neurons.

The performance and learning time of a network with a single hidden layer containing 7 and 28 neurons is similar. Both networks learn in about 40 epochs. Despite the fact that a smaller number of neurons in the network results in much shorter computation time and memory usage during learning, the performance of these two networks is compared for the batch algorithm. In large training sets, the extra computation time is compensated by much faster learning.

As a result, batch method performs much better with 28 hidden neurons neural network. Figure 5.26 shows that a network with 28 hidden neurons learns in about 205 epochs whereas it takes about 469 epochs for a network with 7 neurons to learn with batch algorithm.

Altogether, the overall performance of the topology with 28 hidden neurons in a single hidden layer is better than that of other topologies. In the next section, various combinations of multilayer topologies are evaluated and the results are compared with the conclusion of this section in order to determine the optimal solution.

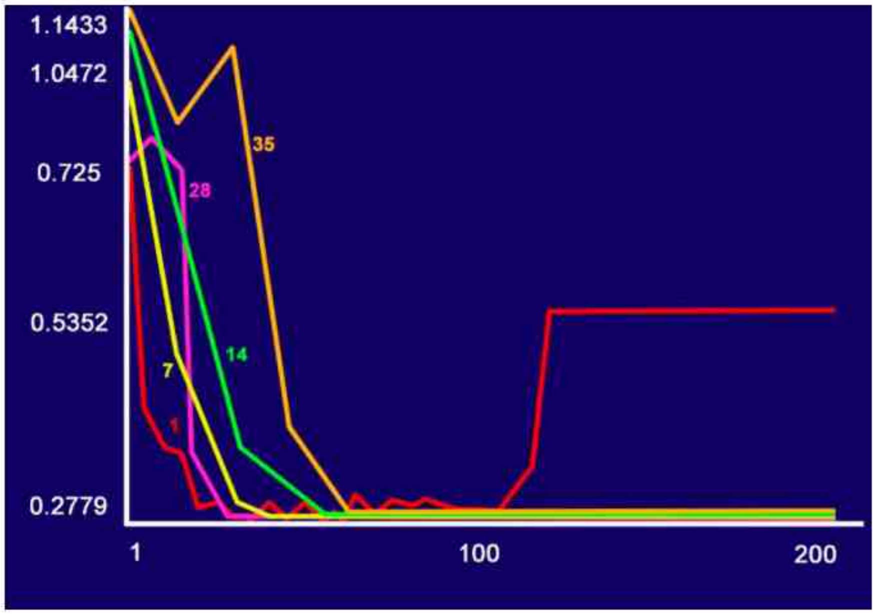


Figure 5.25: Hidden Layer. Comparison of Mean Square Errors of Networks with 1 Hidden Layer during Incremental Algorithm, Linear Error Function, and Gaussian Activation Function. Horizontal bar presents epochs and vertical bar presents mean square of error. The number on each diagram shows the number of hidden neurons.

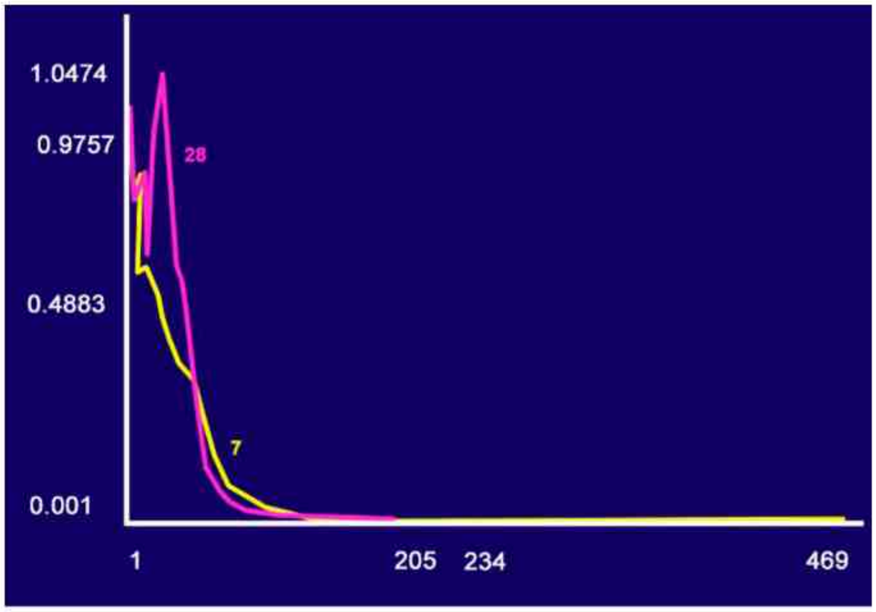


Figure 5 26: Hidden Layer. Comparison of Mean Square Errors of Networks with 1 Hidden Layer during Batch Algorithm. Horizontal bar presents epochs and vertical bar presents mean square of error. The number on each diagram shows the number of hidden neurons.

5.1.3.2 Multi Hidden Layer Neural Network

After finding the best topology for a neural network with a single layer, various combinations of neural networks with more than one hidden layer were explored. Due to the similarity in the performance of the networks, which have similar numbers of hidden neurons, some hypothetical topologies are chosen to examine the behavior of networks with various topologies. In the following sections, seven–seven hidden layer structure (seven neurons in first hidden layer and seven neurons in second hidden layer), fourteen–seven hidden layer structure, fourteen–fourteen hidden layer structure, and hidden layer structure with more neurons are explained.

5.1.3.2.1 Seven - Seven Hidden Layer Structure

In order to evaluate the effect of adding more hidden layers on the performance of neural networks, a network with two hidden layers, each containing seven neurons is constructed as shown in Figure 2.27. Figure 2.28 shows that the neural network learns after 88 epochs using incremental algorithm and Figure 2.29 shows the performance of the same network using batch algorithm.

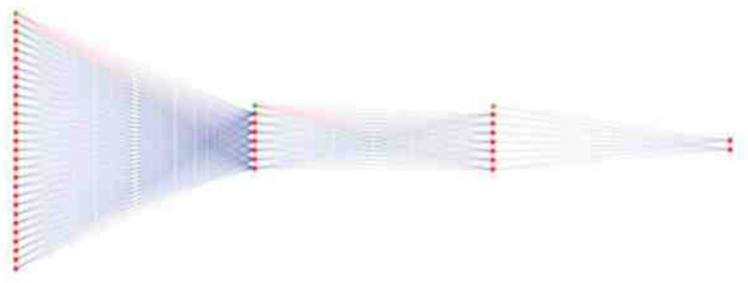


Figure 5 27: Fully Connected Network in Normal Topology with 2 Hidden Layers Consisting of 7 Neurons each and 2 Output Neurons. Circles represent neurons, and lines present weighted connections. Green neurons are the bias of the network. Blue lines are positive weights and red lines are negative weights.

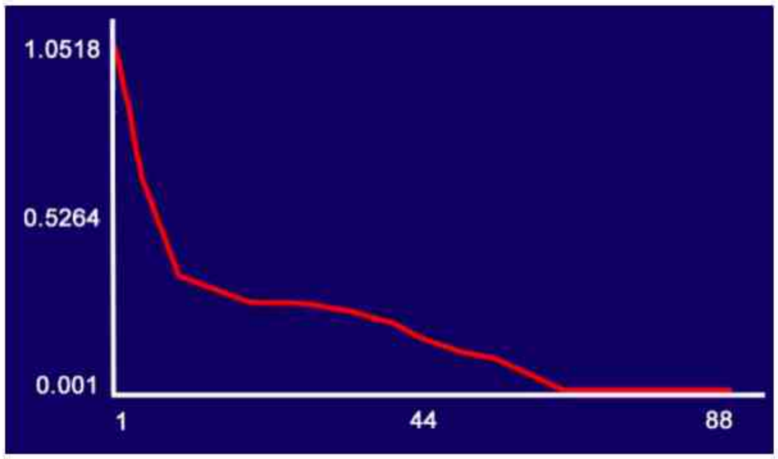


Figure 5.28: Mean Square Error during Training of Network 2 Hidden Layers Consisting 7 Neurons each, with Incremental Algorithm. Horizontal bar presents epochs and vertical bar presents mean square of error.

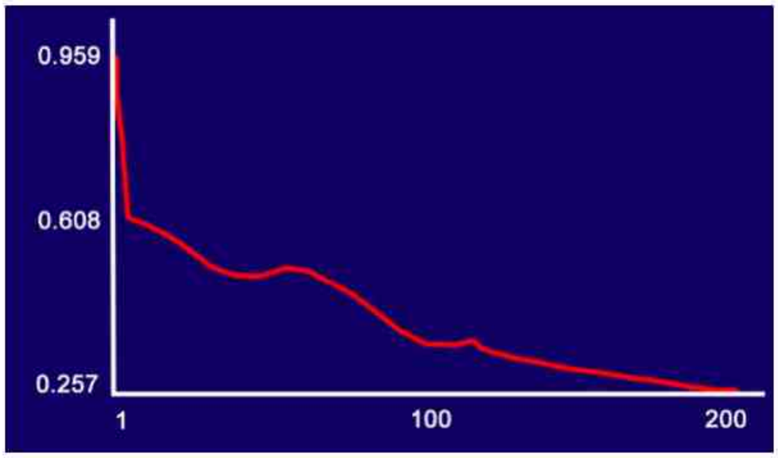


Figure 5.29: Mean Square Error during Training of Network 2 Hidden Layers Consisting 7 Neurons each, with Batch Algorithm. Horizontal bar presents epochs and vertical bar presents mean square of error

5.1.3.2.2 Fourteen - Seven Hidden Layer Structure

Figure 2.30 shows the topology of a network with first hidden layer containing fourteen neurons and second hidden layer containing seven neurons. As it can be seen in Figure 2.31, the network learns after 60 epochs which is faster learning than the seven–seven structure. Figure 2.32 shows the performance of the network in batch algorithm.

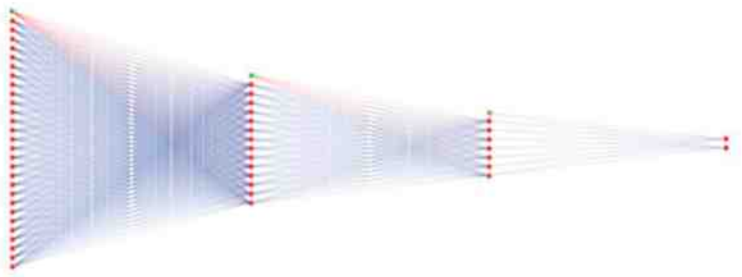


Figure 5 30: Fully Connected Network in Normal Topology with 2 Hidden Layers Consisting 14, and 7 Neurons and 2 Output Neurons. Circles represent neurons and lines present weighted connections. Green neurons are the bias of the network. Blue lines are positive weights and red lines are negative weights.

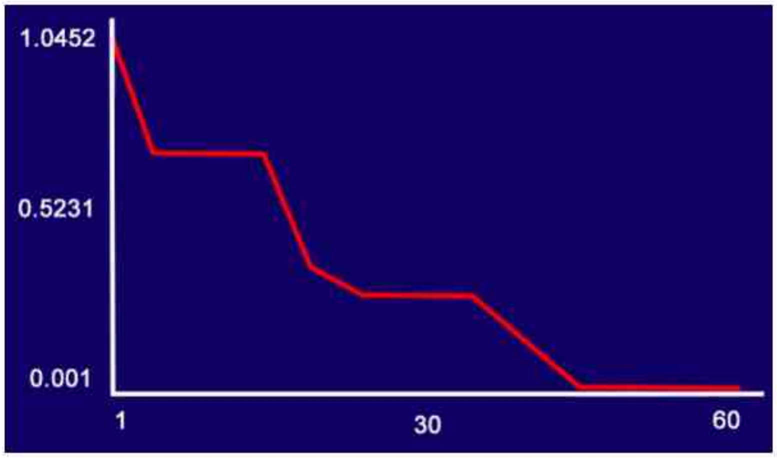


Figure 5 31: Mean Square Error during Training of Network 2 Hidden Layers Consisting 14, and 7 Neurons, with Incremental Algorithm. Horizontal bar presents epochs and vertical bar presents mean square of error

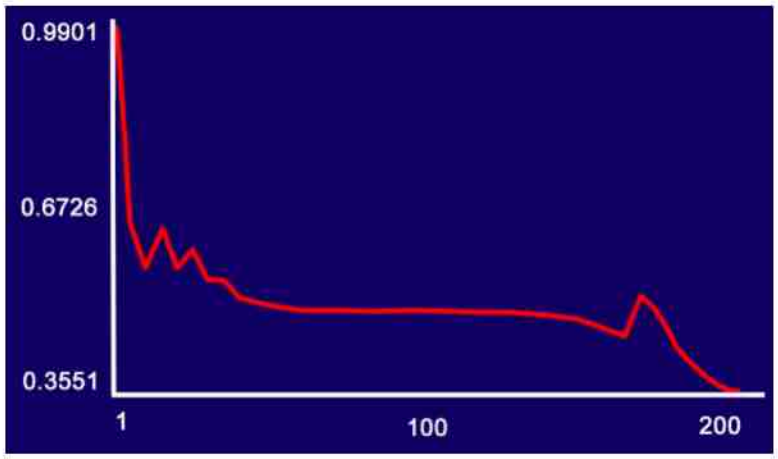


Figure 5 32: Mean Square Error during Training of Network 2 Hidden Layers Consisting 14, and 7 Neurons, with Batch Algorithm. Horizontal bar presents epochs and vertical bar presents mean square of error.

5.1.3.2.3 Fourteen - Fourteen Hidden Layer Structure

As it was presented in the previous sections, the network with 28 hidden neurons performs best for neural networks with a single hidden layer. Hence, the performance of a network with 28 neurons equally divided in two hidden layers was observed. Figure 5.33 shows the topology of this network. As it is shown in Figure 5.34, this structure learns at 83 epochs which is slower than the previous structure.

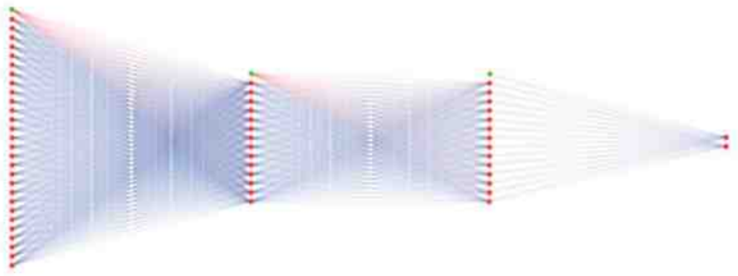


Figure 5 33: Fully Connected Network in Normal Topology with 2 Hidden Layers Consisting 14 Neurons each and 2 Output Neurons. Circles represent neurons and lines present weighted connections. Green neurons are the bias of the network. Blue lines are positive weights and red lines are negative weights.

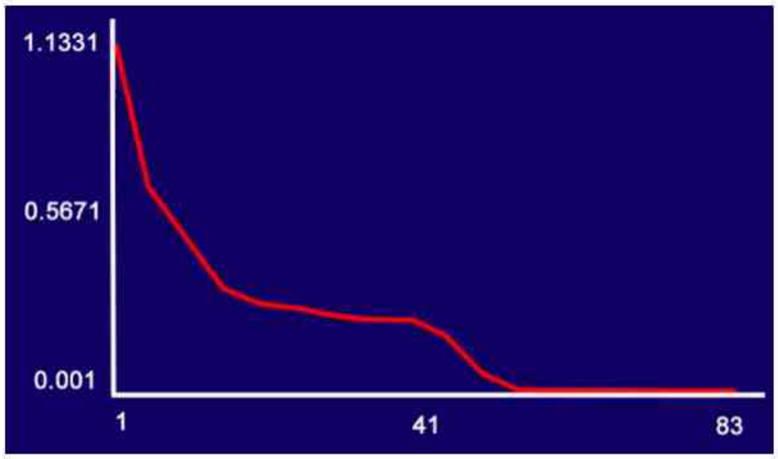


Figure 5 34: Mean Square Error during Training of Network with 2 Hidden Layers Consisting 14 Neurons each, with Incremental Algorithm. Horizontal bar presents epochs and vertical bar presents mean square of error.

Figure 5.35 illustrates the performance of a network with two hidden layers, each consisting of fourteen neurons during batch learning.

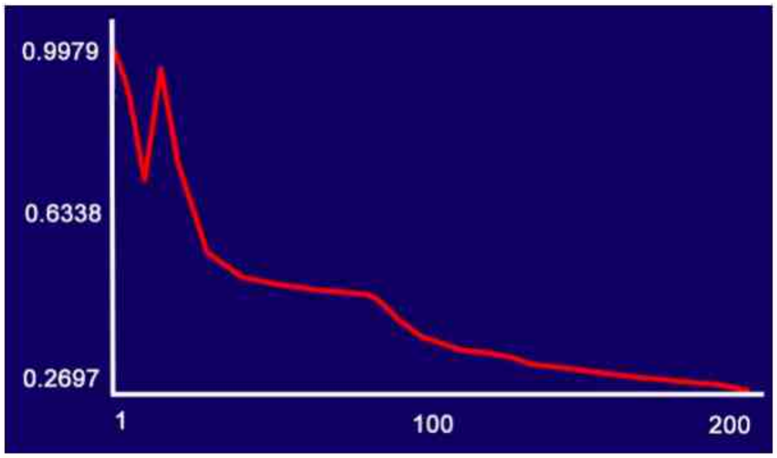


Figure 5 35: Mean Square Error during Training of Network with 2 Hidden Layers Consisting 14 Neurons each, with Batch Algorithm. Horizontal bar presents epochs and vertical bar presents mean square of error.

5.1.3.2.4 Hidden Layer Structure with More Neurons

In this section, other neural networks with two hidden layers are examined. Figure 5.36 illustrates the network with 48 neurons equally divided in two hidden layers in order to investigate the effect of having two layers of the optimal single layer structures. The performance of the network during incremental and batch algorithms is presented in Figures 5.37 and 5.38, respectively.

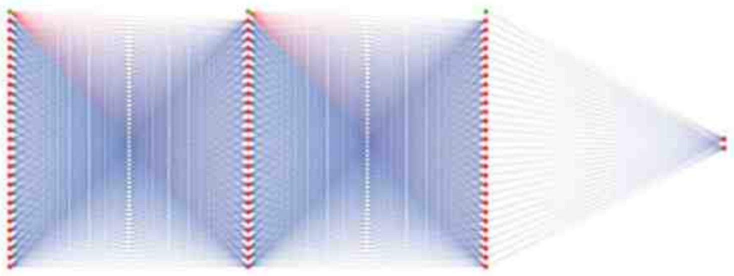


Figure 5.36: Fully Connected Network in Normal Topology with 2 Hidden Neuron Consisting 28 Neurons each and 2 Output Neurons. Circles represent neurons and lines present weighted connections. Green neurons are the bias of the network. Blue lines are positive weights and red lines are negative weights.

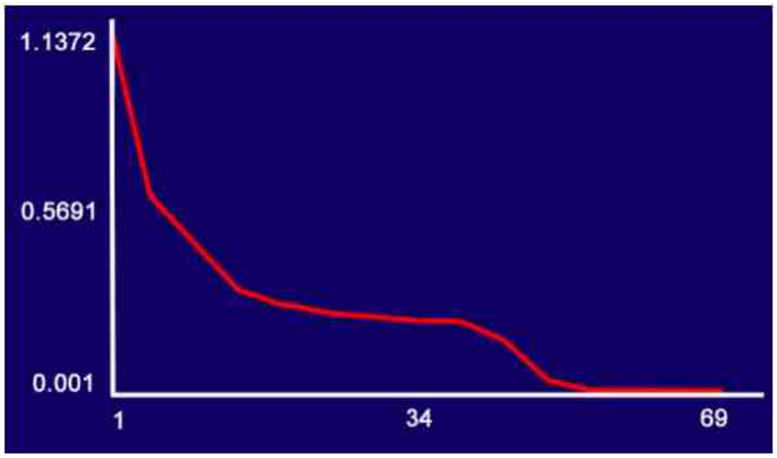


Figure 5 37: Mean Square Error during Training of Network with 2 Hidden Layers Consisting 28 Neurons each, with Incremental Algorithm. Horizontal bar presents epochs and vertical bar presents mean square of error.

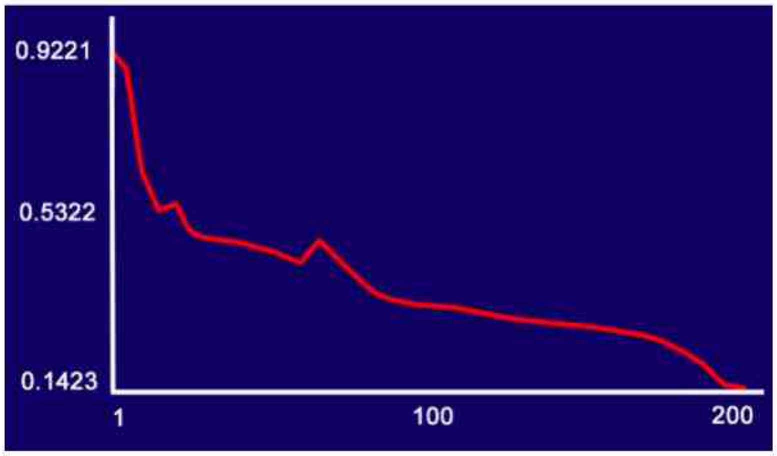


Figure 5 38: Mean Square Error during Training of Network with 2 Hidden Layers Consisting 28 Neurons each, with Batch Algorithm. Horizontal bar presents epochs and vertical bar presents mean square of error.

5.1.4 Conclusion

To conclude this section, Mean Square Errors of multi-hidden layer networks are compared in Figure 5.39. fourteen–seven structure shows faster learning among all multi-layer networks.

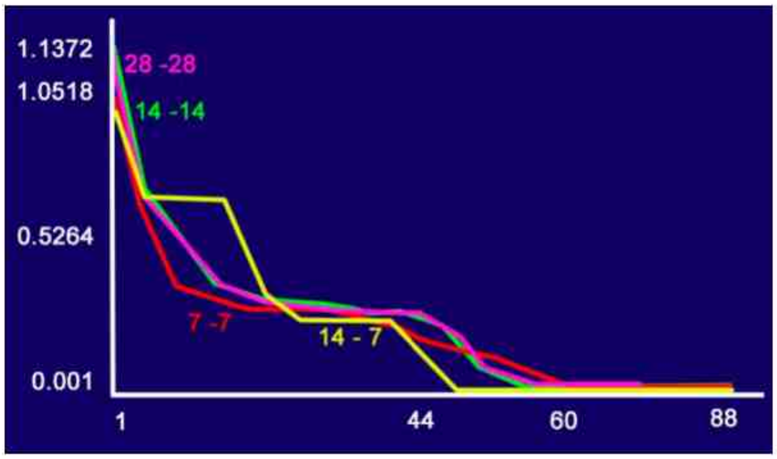


Figure 5.39: Hidden Layer. Comparison of Mean Square Errors of Networks with 2 Hidden Layers during Incremental Algorithm. Horizontal bar presents epochs and vertical bar presents mean square of error. The number on each diagram shows the number of Hidden Neurons in first and second hidden layer, respectively.

In order to make a final decision, the result of multi-layer network performance is compared with the optimal solution of networks with a single hidden layer in Figure 5.40. As seen in the figure, the network with 28 hidden neurons in a single hidden layer yields much smaller overall error and faster learning. Increasing the number of hidden layers continues to add learning time and overall learning error. As a result, the optimal topology of 28 neurons in a single layer is used for the rest of this project.

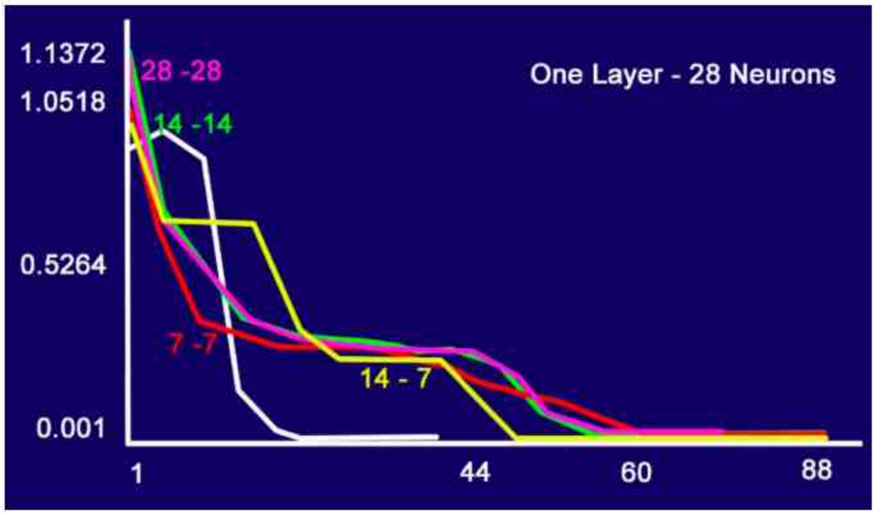


Figure 5.40: Hidden Layer. Comparison of Mean Square Errors of Networks with 1 and 2 Hidden Layers during Incremental Algorithm. Horizontal bar presents epochs and vertical bar presents mean square of error. The number on each diagram shows the number of hidden neurons in first and second hidden layer, respectively.

5.2 Normal vs. Cascade topology

As mentioned in chapter 4, one of the approaches to creating network topologies is cascade training in which training starts with empty neural network. Empty neural networks don't contain any hidden layers. Hidden layers - each containing one representative, shortcut neuron of all possible hidden neurons - are added to the network as training progresses.

In this section, the results of training and testing with normal and cascade methods are evaluated.

The network resulting from the cascade training approach to this particular problem contains one hidden neuron in one hidden layer and selected topology so far contains a hidden layer with 28 hidden neurons, as shown in Figures 4.3 and 4.5 respectively.

In order to examine these methods, networks with desired topologies, algorithms, and parameters were constructed and training was performed. In these experiments, Gaussian activation function and linear error function were used for incremental learning. Then, training continues and weights were adjusted to the point at which mean square of error drops under a certain threshold. In the testing process, the trained network was presented with a new set of sample data, and the outputs of trained network and desired outputs were compared.

For the purpose of comparing the two methods, speed of learning in training process and mean square of error in testing process are compared. Figure 5.41 illustrates the outputs of networks before any training happens. The upper and lower pictures show the results of two different output neurons. As seen in the figure, the mode of calculated output data is 0.5 with a very small standard deviation from the mean, which shows that the network is not able to predict any outputs.

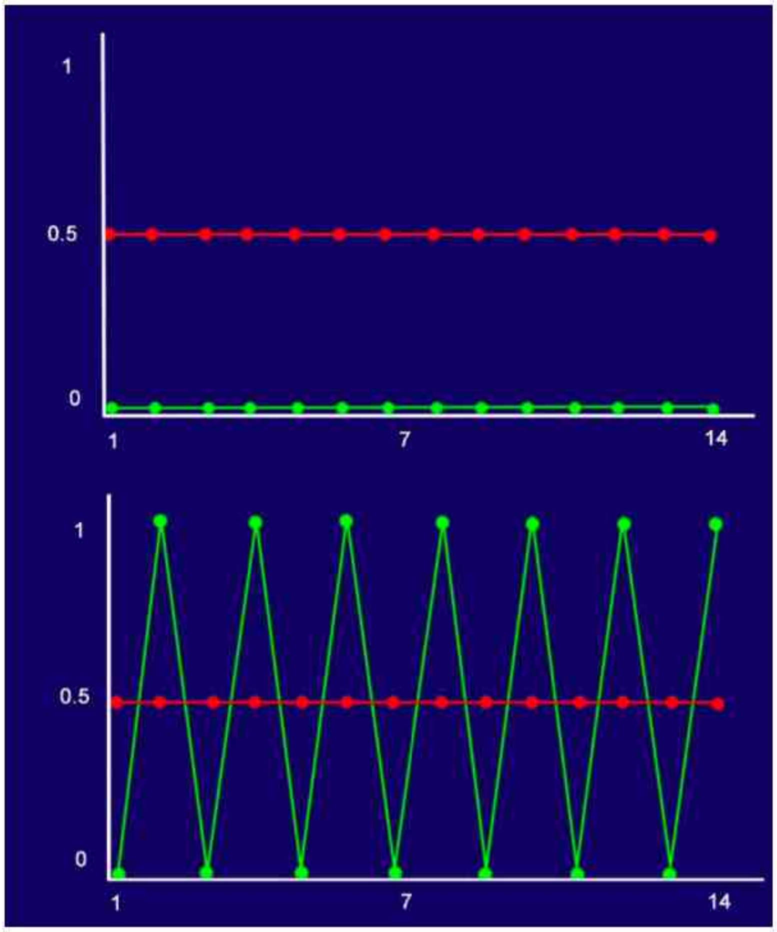


Figure 5.41: **Calculated Output vs Desired Output before training.** Horizontal bar shows different epochs and vertical bars show value of outputs. Green dots demonstrate desired outputs, and red dots calculated outputs by network.

5.2.1 Normal Network's Learning

As stated in chapter 4, the stopping factor of algorithms in this project is 0.001. This means that learning over input samples continues until the mean square of difference between the desired outputs and calculated outputs of the set is less than 0.001. As explained before, choosing a smaller value for this threshold may result in over-learning.

For the network with 28 hidden neurons, the mean square of error has a decreasing exponential function (Figure 5.42) and drops rapidly.

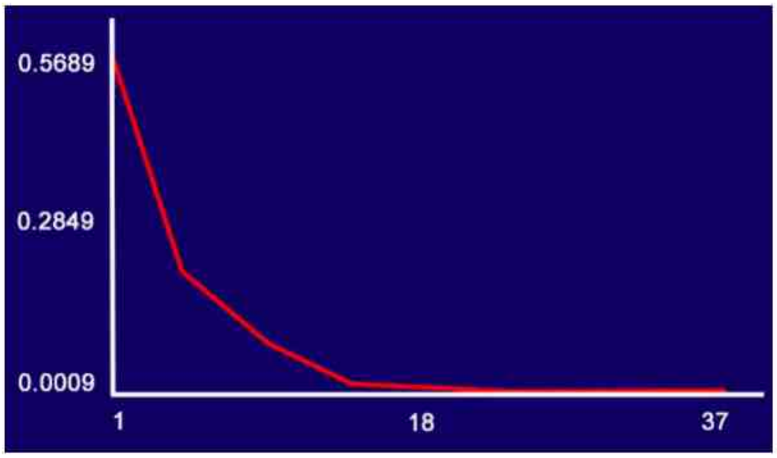


Figure 5.42: Mean Square Error during Training of Normal Topology.
Horizontal bar presents epochs and vertical bar presents mean square of error.

5.2.2 Cascade Network's Learning

Running cascade learning using the same network settings shows that the learning has a decreasing linear function and the error rate does not decline as quickly as expected. Furthermore, learning speed is much lower than that of the previous method. As Figure 5.43 illustrates, the mean square of errors does not reach 0.001 until about 180 epochs.

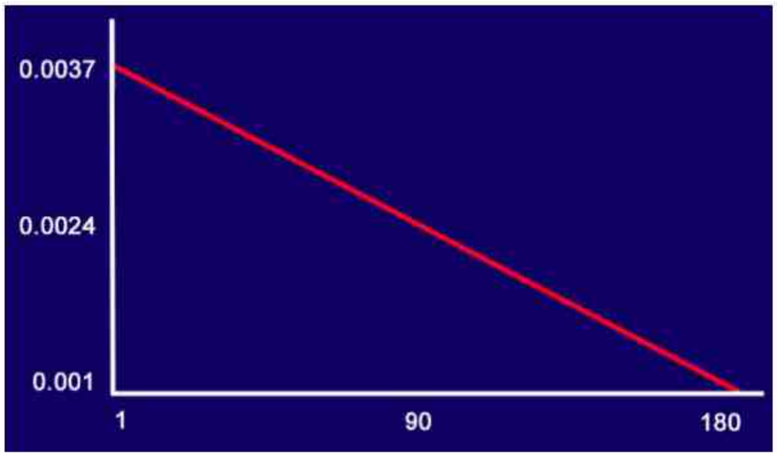


Figure 5.43: Mean Square Error during Training of Cascade Learning.
Horizontal bar presents epochs and vertical bar presents mean square of error.

5.2.3 Normal Network's Error Rate

After the training and test process, the comparison between individual calculated outputs and desired outputs is evaluated to determine the success of the training method. Figure 5.44 shows how well normal topology is performing. The performance of network is assessed individually for each of the two output neurons. The picture at the top of the figure shows the results of the first output neuron, and the picture at the bottom of the figure shows the results of the second output neuron. The overall Mean Square Error of this test is 0.00157 at epoch 37.

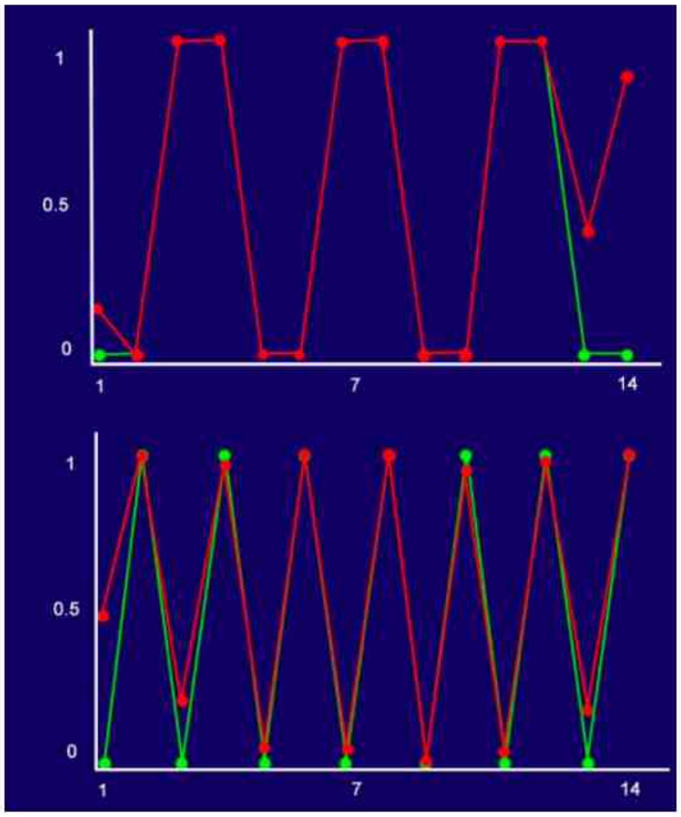


Figure 5 44: Calculated Output vs Desired Output after Training in Normal Topology. Horizontal bar shows different epochs and vertical bars show value of outputs. Green dots demonstrate desired outputs, and red dots calculated outputs by network.

5.2.4 Cascade Network's Error Rate

Similarly to what it was said in the previous section, the test process was executed using the cascade network after training. Figure 5.45 presents comparison between desired outputs and calculated outputs of the trained network. Furthermore, the outputs were evaluated for each output neuron individually.

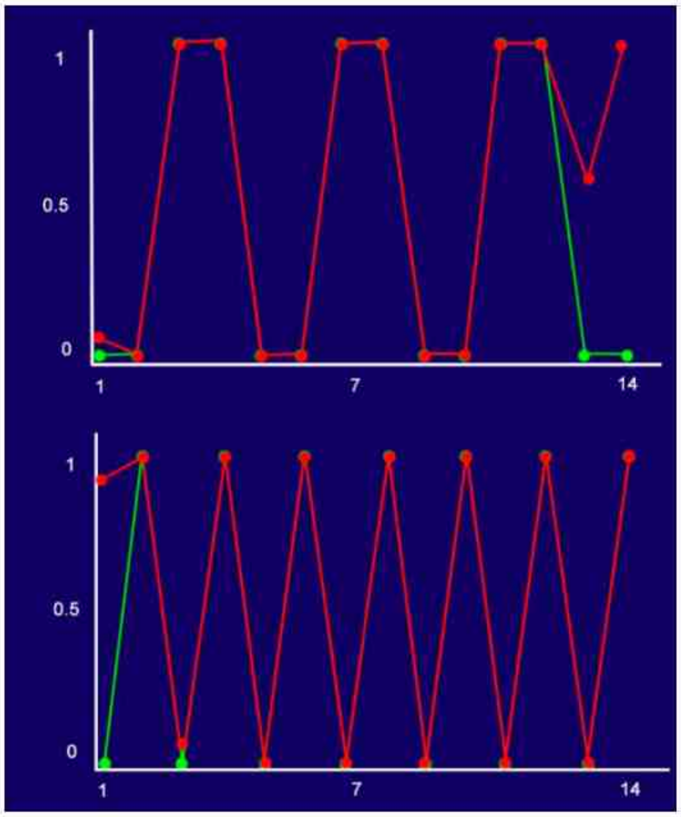


Figure 5.45: Calculated Output vs Desired Output after Training in Cascade Topology. Horizontal bar shows different samples and vertical bars show value of outputs. Green dots demonstrate desired outputs, and red dots calculated outputs by network.

5.2.5 Conclusion

The comparison of learning speeds and error rates were performed with other activation functions and parameters used in this project; however, due to the similarity of the resulting behavior to other experiments, they are not included in this document. Overall, the speed of learning in normal topology could increase up to ten times compared to that of the cascade training; and with much smaller Mean Square Error.

For the normal topology, a network with 28 neurons in a single layer has the best overall performance among all the topologies. Adding extra layers and neurons to the network increases the computing time and memory usage, in addition to slowing down the learning process with no apparent improvement in performance.

As the normal topology shows faster learning and smaller error rate, that topology is used for this project.

5.3 Incremental vs Batch learning

Both incremental learning and batch learning are feed forward, back propagation algorithms. The standard back propagation algorithm is incremental; the weights of connections are updated after each input sample pattern. This method involves many calculations in comparison to batch method. Batch method changes the weights after calculating the mean square error of each training epoch. For this section, both incremental and batch methods were performed on the sample data and their performance is compared. The results of incremental and batch algorithms can be seen in Figures 5.46 and 5.47 respectively.

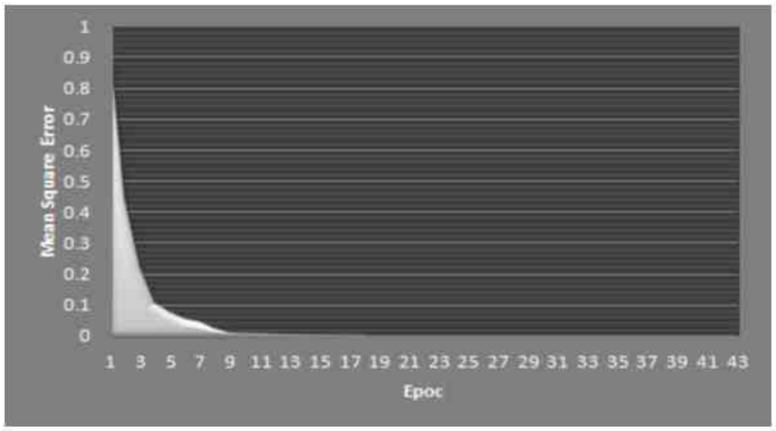


Figure 5.46: Incremental Learning Algorithm

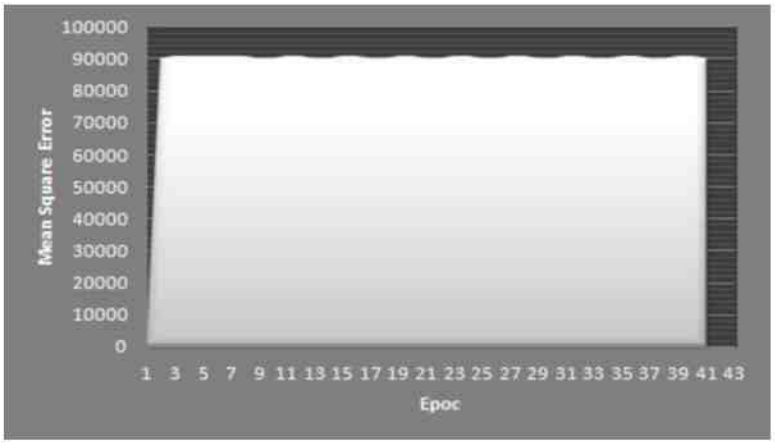


Figure 5.47: Batch Learning Algorithm

Running incremental algorithm shows that learning occurs quickly whereas default batch algorithm never learns and has a very high Mean Square Error value of 89667.719. Executing the test on a trained network with the default batch algorithm results in outputs which are shown in Figure 5.48. The two diagrams below show the results for two individual output neurons.

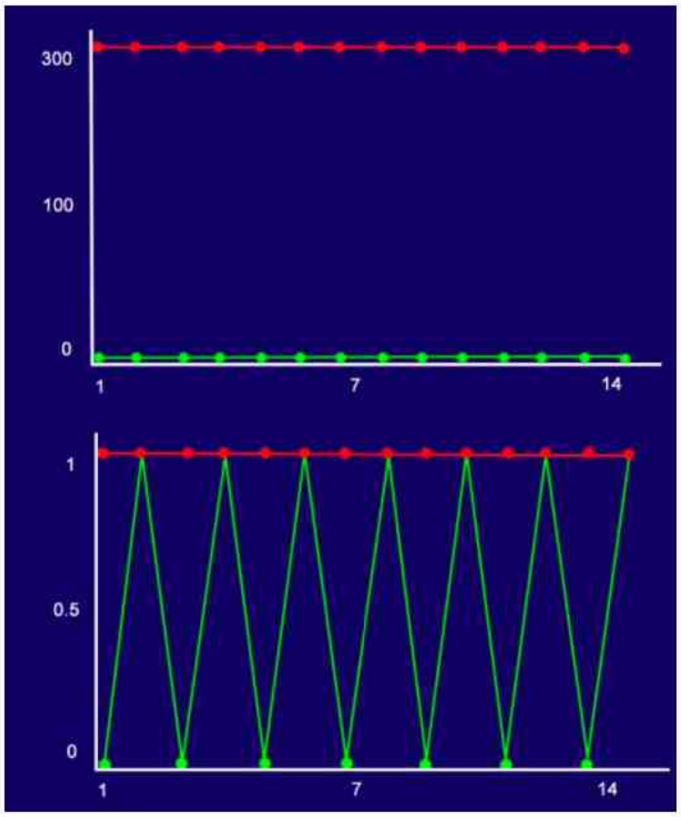


Figure 5.48: Calculated Output vs. Desired Output after Batch learning Algorithm. Horizontal bar shows different samples and vertical bars show value of outputs. Green dots demonstrate desired outputs, and red dots calculated outputs by network.

Changing the settings, error functions, or activation functions of individual neurons in hidden and output layers, etc., can result in eventual learning with batch algorithm as shown in Figure 5.49.

Although the network can learn with special settings of batch learning, the performance of incremental method is generally more efficient. The details of the comparison between batch learning and incremental learning are included in the following sections.

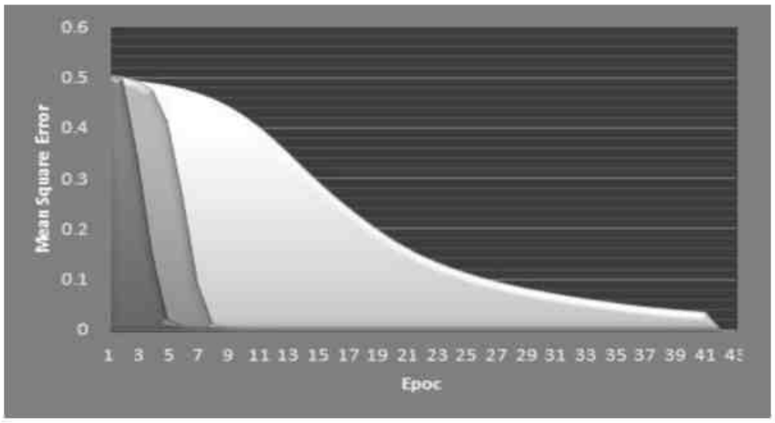


Figure 5.49: Comparison between different settings, and algorithm for batch learning.

Furthermore, the learning gets faster by reducing the step size between the intervals that the Mean Square Error is reporting. By decreasing the steps between these reports, the overall Mean Square Error decreases as well.

Decreasing steps in batch method makes the learning more similar to incremental method. The performance of batch algorithm with various step sizes is shown in Figure 5.50.

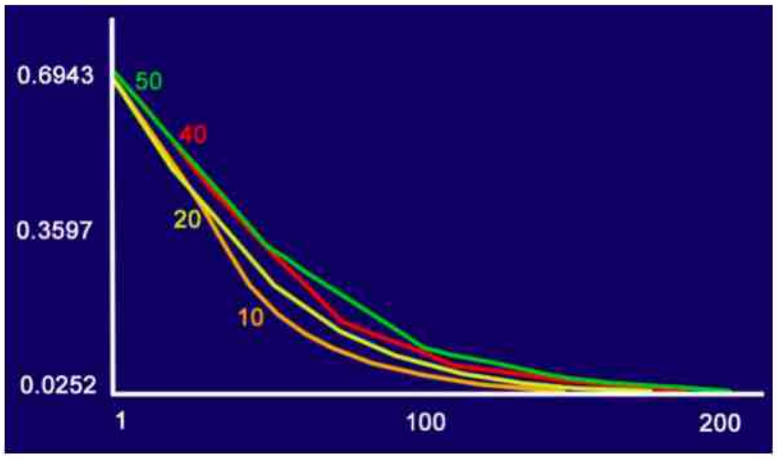


Figure 5.50: Comparison of Mean Square Errors during Batch learning with different step factor.
Horizontal bar presents epochs and vertical bar presents mean square of error.

The mean square of error is decreasing and the performance is increasing by decreasing the step size. Hence, batch algorithm was evaluated with the step value of one. As a result, even after 200 epochs, the value of Mean Square Error does not drop below the stopping factor; which was 0.001. The result of this experiment is shown in Figure 5.51.

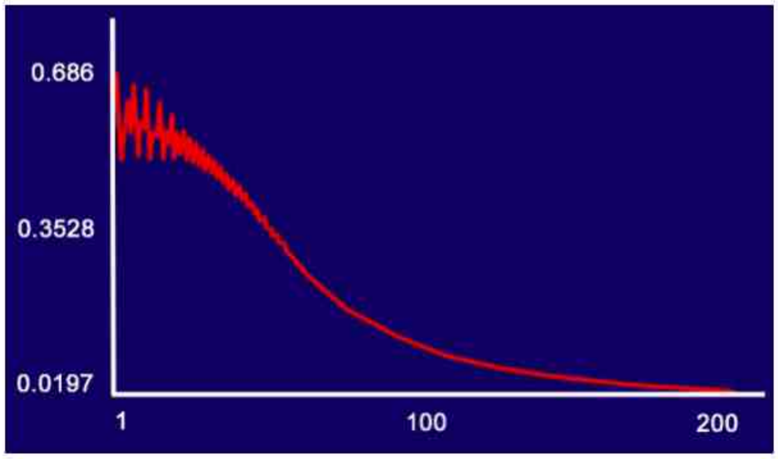


Figure 5.51: Mean Square Error during Training of Batch learning with step equal to one. Horizontal bar presents epochs and vertical bar presents mean square of error.

Counterintuitively, in spite of continuing to decrease the interval of reporting Mean Square Error, the performance of batch method is not similar to that of incremental method. The difference of the two strategies is illustrated in Figure 5.52.

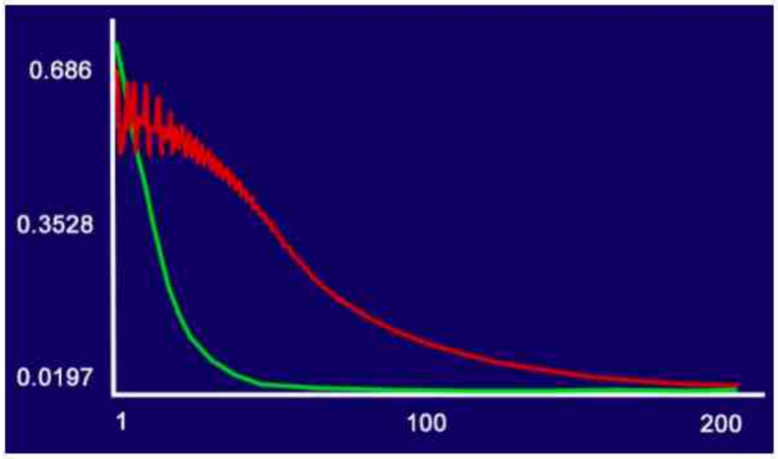


Figure 5.52: Comparison of Mean Square Errors during Training of Linear learning and Batch learning with step equal to one. Horizontal bar presents epochs and vertical bar presents mean square of error.

5.4 Algorithms' Performance

5.4.1 Error Function

During the classification, the weights were adjusted to calculate outputs closer to desired outputs for new input patterns. The network can use several error functions to measure the errors. In this section, linear and tangent error functions are evaluated for incremental and batch algorithms. Comparing the performance of the network for various error functions and algorithms allows to select best strategies for solving classification problems.

5.4.1.1 Linear Error Function

Executing training on the network with linear error function showed that linear error function in combination with incremental algorithm performed better comparing to linear error function with batch algorithm. Each algorithm was examined with various parameter settings and several activation functions, and the incremental algorithm generally demonstrated better performance. A sample comparison of the behavior of the network during training sessions with these algorithms is shown in Figure 5.53.

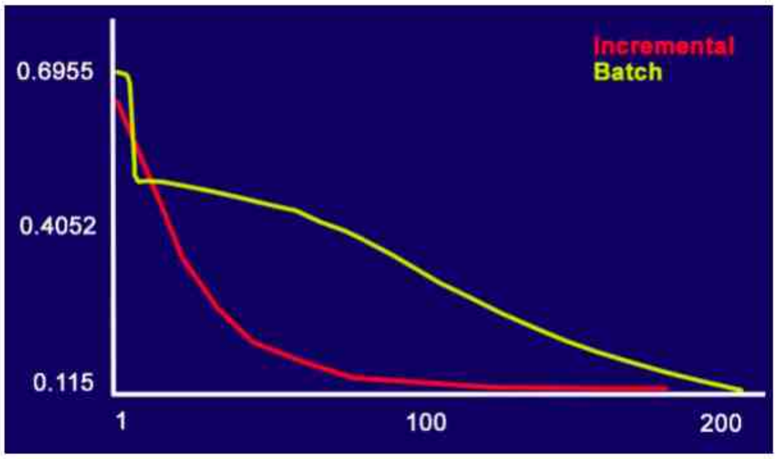


Figure 5.53: Linear Error Function. Comparison of Mean Square Errors during Incremental and Batch training Algorithm. Horizontal bar presents epochs and vertical bar presents mean square of error.

5.4.1.2 Tanh Error Function

Another option for calculating the errors during training is tanh function. In this section, incremental and batch algorithms are evaluated in combination with tanh error function with various parameter settings and activation functions. A sample comparison between the outcomes is presented in Figure 5.54. As it can be seen in the figure, tanh error function shows better performance with batch algorithms.

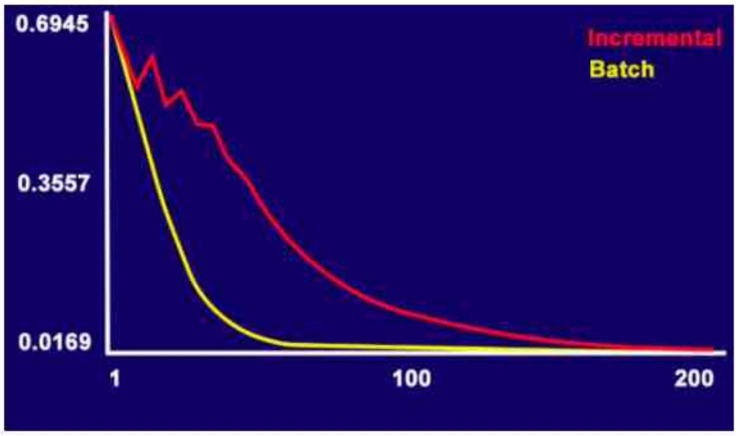


Figure 5.54: Tanh Error Function. Comparison of Mean Square Errors during Incremental and Batch training Algorithm. Horizontal bar presents epochs and vertical bar presents mean square of error.

5.4.1.3 Conclusion

Linear error function performs better with incremental algorithms and is selected to be used in incremental and cascade learning. On the other hand, tanh error function performs better when it is used with batch algorithm and it is not recommended for incremental learning.

5.4.2 Activation Functions

In the classification procedure, activation functions are used to transform the activation level of individual neurons from their input value to their output value. In this section, linear, threshold, sigmoid, Gaussian, sin, and cosine activation functions are evaluated using incremental and batch algorithms. The mean square of errors and learning times are then analyzed in order to compose the best strategy to solve the EEG emotional state classification.

As presented in section 5.4.1, the linear algorithm performs better with linear error function while tanh error function works better with the batch algorithm. Thus, all the trainings with individual activation functions were run once using the combination of tanh error function and batch algorithm, and another time using the combination of linear error function and incremental algorithm. The activation functions of all the neurons were set to the specific activation function under investigation.

5.4.2.1 Linear

This experiment showed that network was unable to learn with batch method and linear activation function. Figure 5.55 illustrates the performance of network training with batch algorithm, tanh error function, and linear activation function.

As it can be seen in the figure, the mean square of error arises to a very high value of 179367.72 on the second epoch and keeps that value with very small fluctuations. Continuing the learning to 1000 epochs shows that the error never drops and network is never trained. Thus, this combination is not suitable to solve the EEG emotional state classification problem.

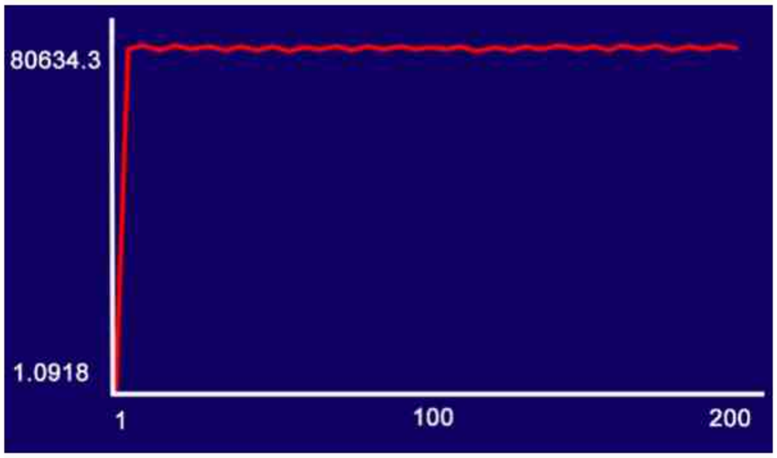


Figure 5.55: Linear Activation Function with Batch Algorithm. Horizontal bar presents epochs and vertical bar presents mean square of error.

The next step was to research the performance of the network with incremental algorithm and linear error function. As shown, Mean Square Error starts with a small value of 0.5689 and drops quickly. After 37 Epochs, the mean square of errors decreases to 0.0009 which is below the stopping point of learning. Figure 5.56 shows the performance of the network using linear activation function and incremental algorithm. The outputs of the trained network with this setup in testing process and their comparison with desired outputs (targets) for individual output neurons are shown in Figures 5.56 and 5.57.

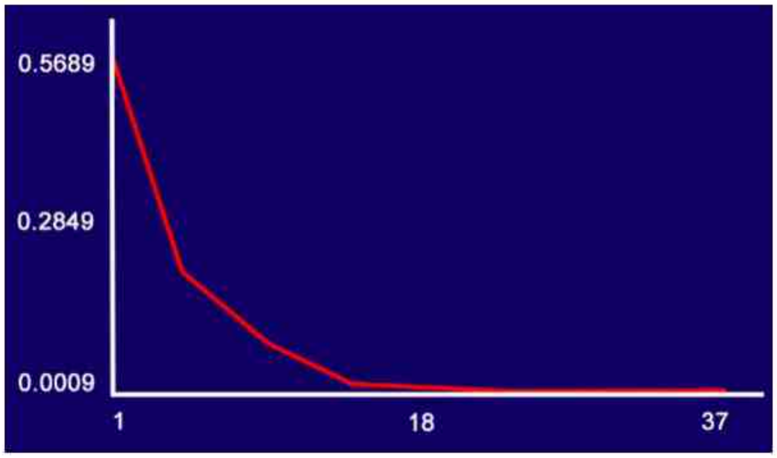


Figure 5.56: Linear Activation Function with Incremental Algorithm. Horizontal bar presents epochs and vertical bar presents mean square of error.

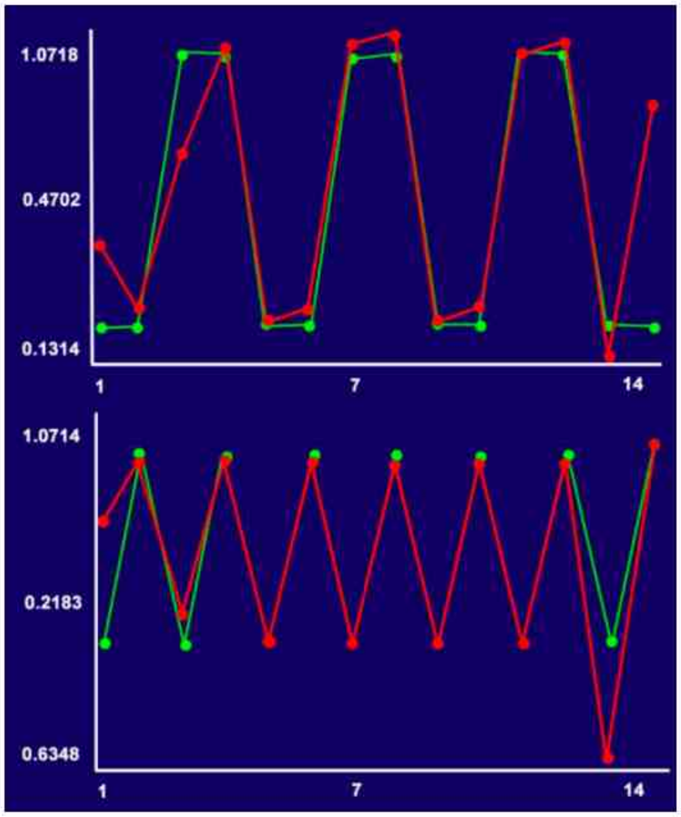


Figure 5.57: Calculated Output vs Desired Output after training in Normal Topology, Incremental Algorithm, Linear Error Function, and Linear Activation Function. Horizontal bar shows different samples and vertical bars show value of outputs. Green dots demonstrate desired outputs, and red dots calculated outputs by network.

5.4.2.2 Threshold

Training the network failed with threshold activation function both with method one (batch algorithm and tanh error function) and method two (incremental algorithm and linear error function). The learning process was performed for 200 epochs with both strategies. The Mean Square Error for batch method was a consistent value of 1.01388 and for incremental method was a consistent value of 0.73611.

5.4.2.3 Sigmoid

Training the network with sigmoid activation function led to the following results. In the batch method, learning starts with a small Mean Square Error value of 0.5003 and continues by decreasing gradually. The performance of network over presented samples has a decreasing exponential function. The value of Mean Square Error at epoch 200 is 0.03757 and network slowly continues to learn. Figure 5.58 shows the learning performance using this method.

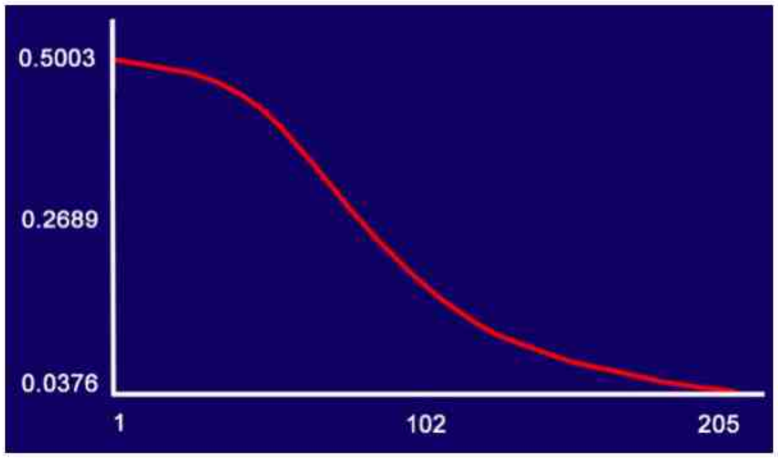


Figure 5 58: Sigmoid Activation Function with Batch Algorithm. Horizontal bar presents epochs and vertical bar presents mean square of error.

The Mean Square Error drops much faster using incremental algorithm and learning completes at epoch 190. The mean square of error at epoch 190 is 0.0009 which is less than the learning stopping parameter. The performance of learning is shown in Figure 5.59 and the comparison between the calculated and desired output for two output neurons in testing is shown in Figure 5.60.

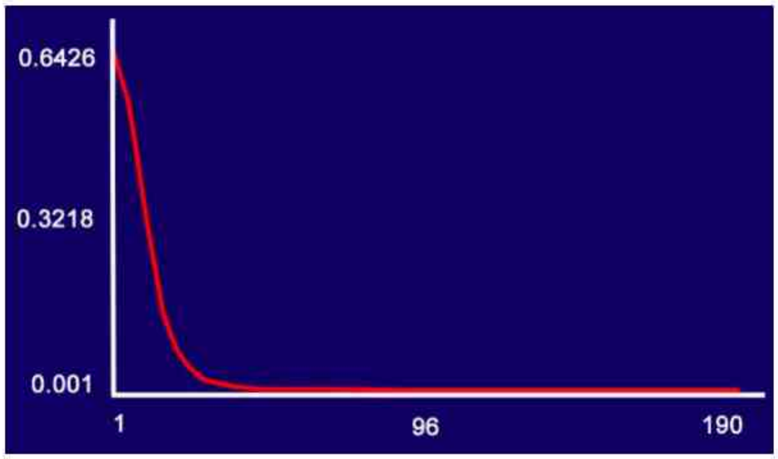


Figure 5.59: Sigmoid Activation Function with Incremental Algorithm. Horizontal bar presents epochs and vertical bar presents mean square of error.

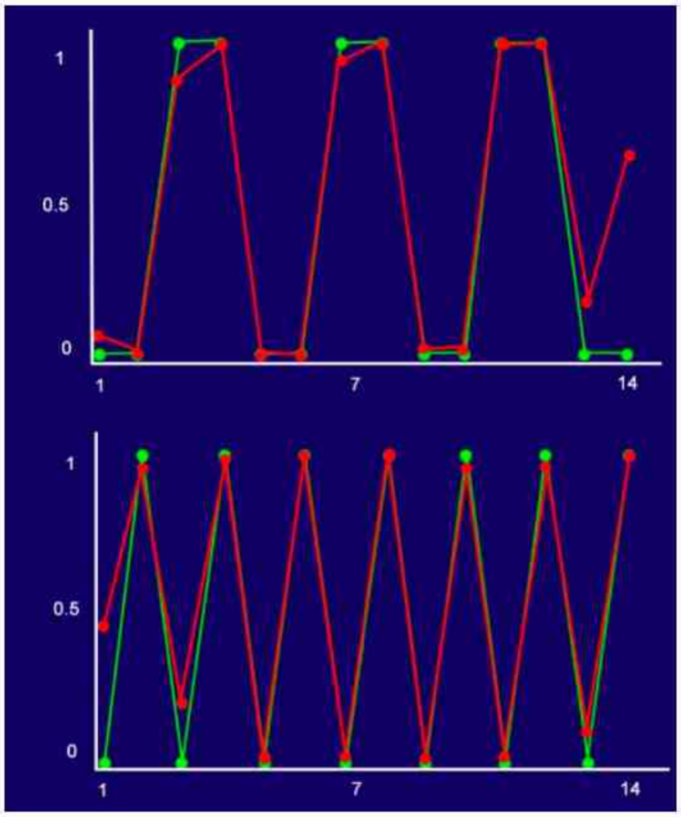


Figure 5.60: Calculated Output vs. Desired Output after training in Normal Topology, Incremental Algorithm, Linear Error Function, and Sigmoid Activation Function. Horizontal bar shows different samples and vertical bars show value of outputs. Green dots demonstrate desired outputs, and red dots calculated outputs by network.

5.4.2.4 Gaussian

Applying Gaussian activation function resulted in successful learning with both batch and incremental methods. Running the batch settings, the mean square of error drops to 0.0045 in 200 epochs as shown in Figure 5.61. The learning is much faster with incremental method and linear error function. As shown in Figure 5.62, the mean square function decreases to 0.00098 after 40 epochs. The comparison between outputs and targets of incremental method is shown in Figure 5.63.

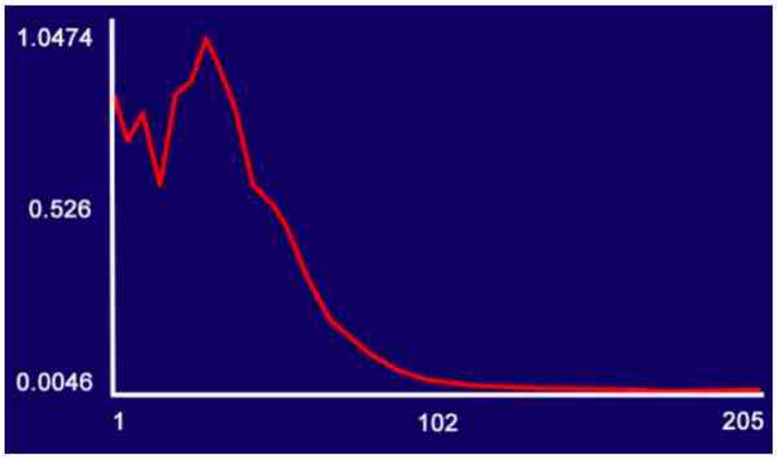


Figure 5.61: Gaussian Activation Function with Batch Algorithm. Horizontal bar presents epochs and vertical bar presents mean square of error.

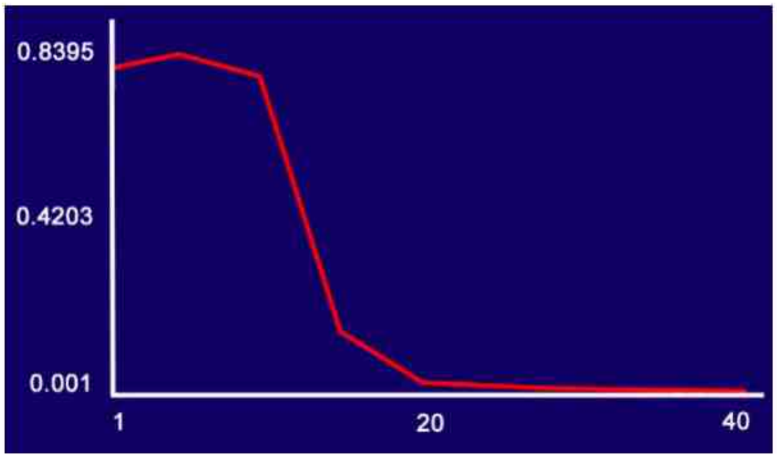


Figure 5.62: Gaussian Activation Function with Incremental Algorithm. Horizontal bar presents epochs and vertical bar presents mean square of error.

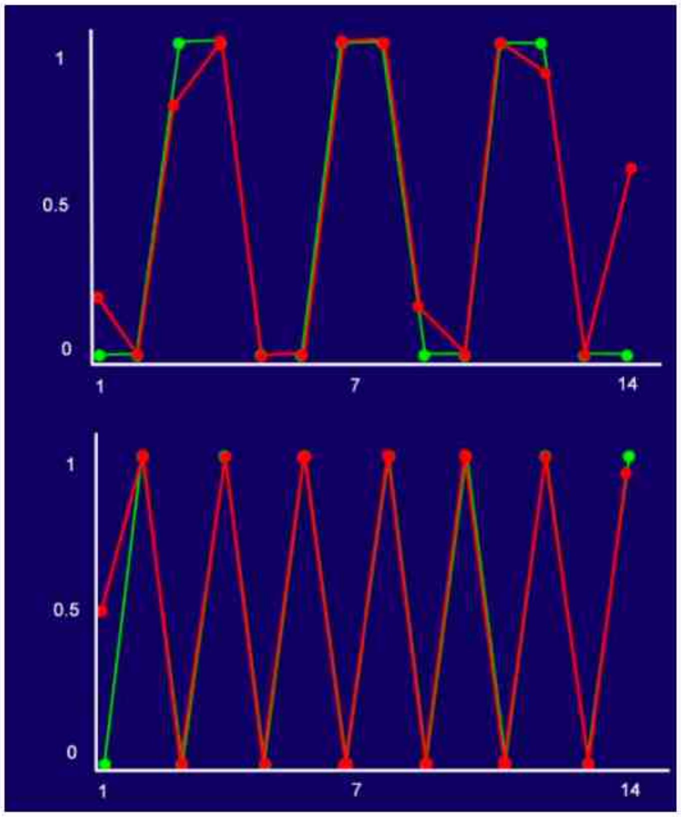


Figure 5.63: Calculated Output vs Desired Output after training in Normal Topology, Incremental Algorithm, Linear Error Function, and Gaussian Activation Function. Horizontal bar shows different samples and vertical bars show value of outputs. Green dots demonstrate desired outputs, and red dots calculated outputs by network.

5.4.2.5 Sine Symmetric

The same procedure presented in the previous sections was repeated for sine activation function. The network failed to learn with its neurons having sine activation function using both batch and incremental methods.

For batch learning, the mean square of error fell to the value of 0.0088 but it rose afterwards. The mean square of error fluctuates dramatically during the training. The value of Mean Square Error was 0.95516 after 200 epochs. Figure 5.64 shows the performance of sine activation function using batch method.

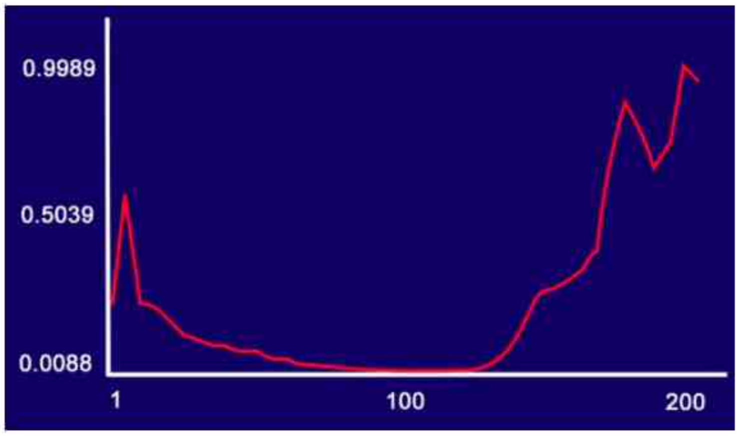


Figure 5 64: Sine Activation Function with Batch Algorithm in 200 Epochs. Horizontal bar presents epochs and vertical bar presents mean square of error.

As the mean square of error fluctuated drastically in the first 200 epochs, the learning was continued for 1000 epochs in order to observe the progress of the learning. The resulting diagram has local fluctuations but the overall mean square of error increases over time. Figure 5.65 illustrates the performance of sine activation function using batch method in 1000 epochs.

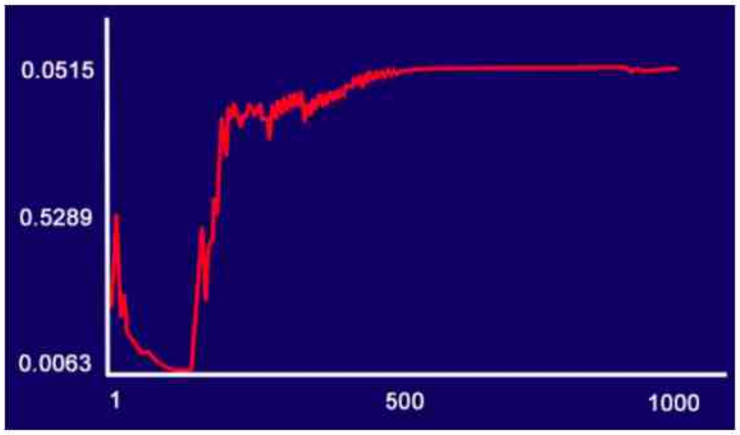


Figure 5.65: Sine Activation Function with Batch Algorithm in 1000 Epochs. Horizontal bar presents epochs and vertical bar presents mean square of error.

The network showed similar behavior for incremental method. The Mean Square Error drops to 0.02 and grows up again. The Mean Square Error at epoch 200 is 1.0565, as it is shown in Figure 5.66.

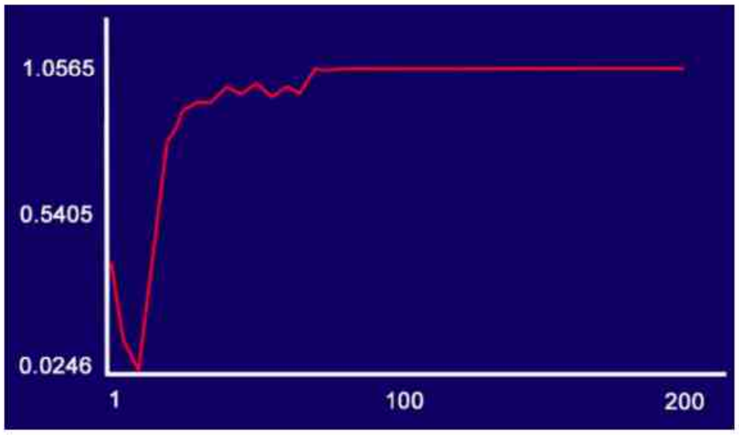


Figure 5.66: Sine Activation Function with Incremental Algorithm. Horizontal bar presents data points and vertical bar presents mean square of error.

As it can be seen in Figure 5.67, the network is unable to predict the targets correctly in the test process.

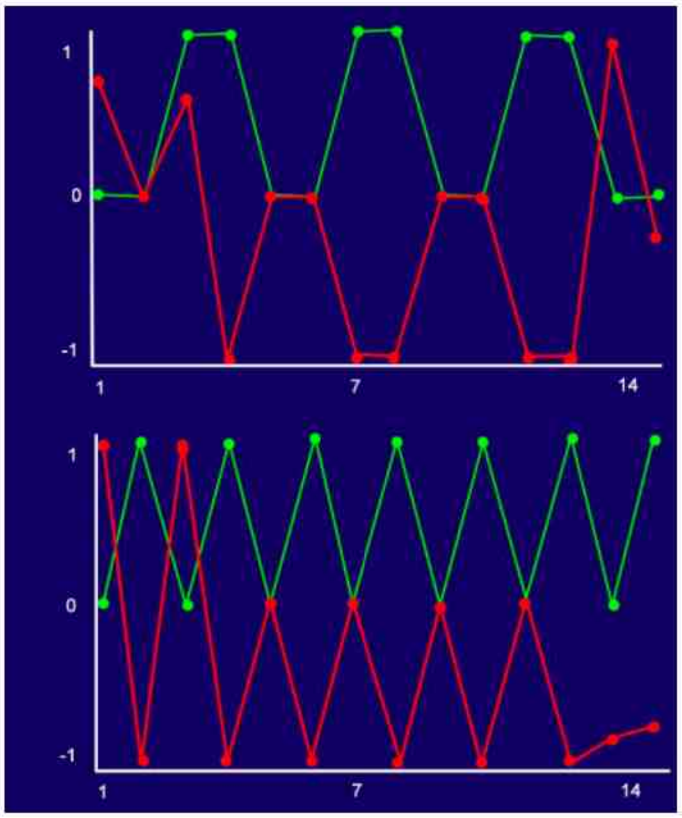


Figure 5.67: Calculated Output vs Desired Output after training in Normal Topology, Incremental Algorithm, Linear Error Function, and Sine Activation Function. Horizontal bar shows different samples and vertical bars show value of outputs. Green dots demonstrate desired outputs, and red dots calculated outputs by network.

5.4.2.6 Cosine Symmetric

In the experiment described in this section, cosine activation function was examined using batch and incremental algorithms. Although the network eventually learned in both cases, incremental method is much faster and much more efficient for cosine activation function.

For batch method, the mean square of error drops to 1.14 and fluctuates afterwards. Figure 5.68 illustrates the behavior of the network in 200 epochs. The value of Mean Square Error is 0.1302 at epoch 200. The overall mean square of error was gradually decreasing during the training. Thus, the performance of cosine activation function was examined in 1000 epochs. As it is shown in Figure 5.69, the mean square of error dropped below the learning stopping point and the learning stopped at epoch 870.

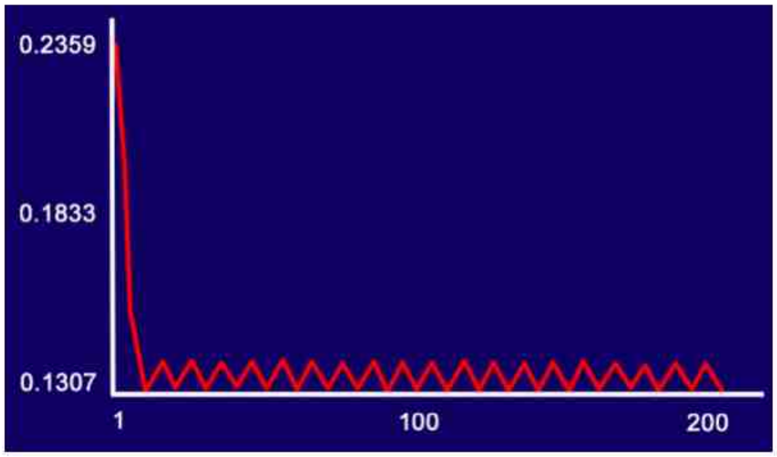


Figure 5.68: Cosine Activation Function with Batch Algorithm in 200 Epochs. Horizontal bar presents epochs and vertical bar presents mean square of error.

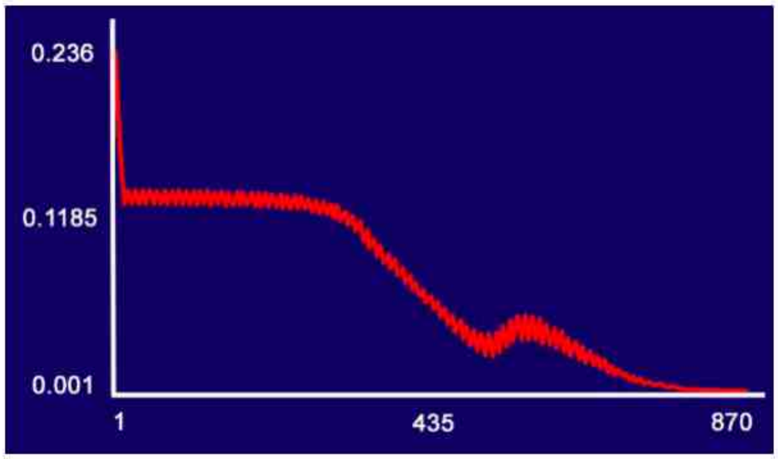


Figure 5.69: Cosine Activation Function with Batch Algorithm in 1000 Epochs. Horizontal bar presents epochs and vertical bar presents mean square of error.

Applying incremental method in combination with cosine activation function and linear error function resulted in a successfully trained network after 171 epochs, as it can be seen in Figure 5.70. The outcomes of testing are shown in Figure 5.71.

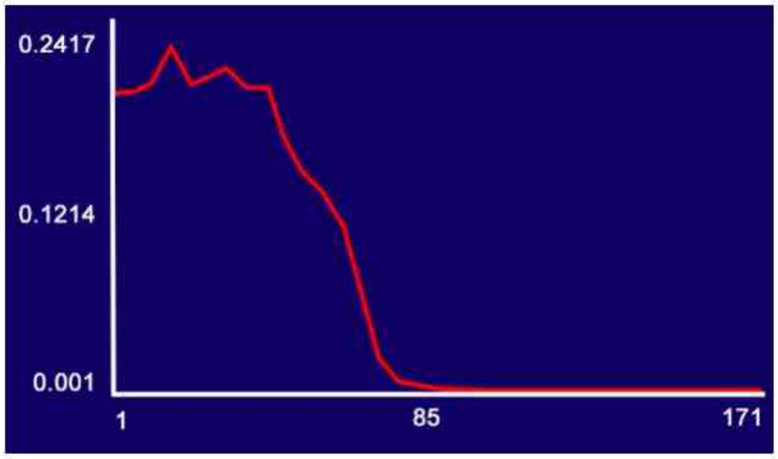


Figure 5.70: Cosine Activation Function with Incremental Algorithm. Horizontal bar presents epochs and vertical bar presents mean square of error.

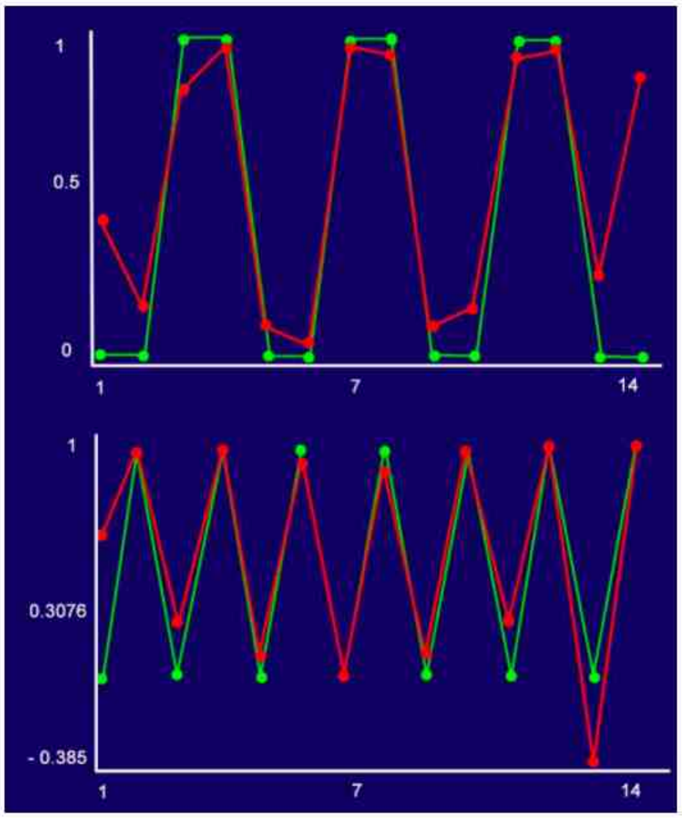


Figure 5.71: Calculated Output vs Desired Output after training in Normal Topology, Incremental Algorithm, Linear Error Function, and Cosine Activation Function. Horizontal bar shows different samples and vertical bars show value of outputs. Green dots demonstrate desired outputs, and red dots calculated outputs by network.

5.4.2.7 Conclusion

The overall Mean Square Error and the speed of learning were two main criteria for the selection of the best activation function.

As illustrated in this chapter, incremental learning generally shows better performance; especially with linear error function. More comparisons of the speed of batch and incremental algorithms are illustrated in Figure 5.73. The next step was comparing activation functions. Figure 5.72 shows the performance of all the various activation functions analyzed in the previous sections namely: linear, threshold, sigmoid, Gaussian, sin, cosine in combination with incremental algorithm and linear error function. As it can be seen in the diagram, networks with sin, and threshold activation function are unable to learn successfully. Networks with sigmoid and cosine function are able to learn and Mean Square Error of sigmoid activation function drops much faster. However, the problem with these two activation functions is the length of the training. It takes the network 171 and 190 epochs to learn with cosine and sigmoid activation functions, respectively. Furthermore, linear and Gaussian activation functions have lower Mean Square Errors and their training times are much higher. In the experiments presented in this document, network with linear activation function learned after 40 epochs and with Gaussian activation function learned after 37 epochs. Despite a few of epochs of difference in learning length and the difference in sum of Mean Square Errors, both activation methods are suitable for this classification.

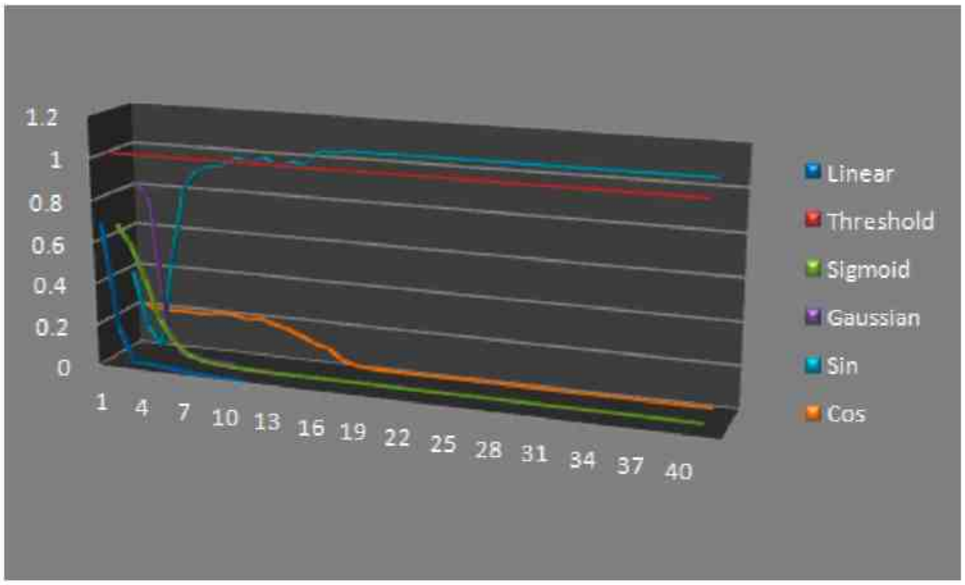


Figure 5.72: Comparison of Performance of Different Activation Functions. Horizontal bar presents steps each containing 5 epochs and vertical bar presents mean square of error.

Figure 5.73 illustrates the comparison between batch and incremental algorithms for the activation functions. As it can be seen in the diagram, the combination of incremental algorithm and linear error function generally yields better results than the combination of batch algorithm and tanh error function. The fastest strategy for this classification is incremental algorithm, linear error function, and Gaussian activation function.

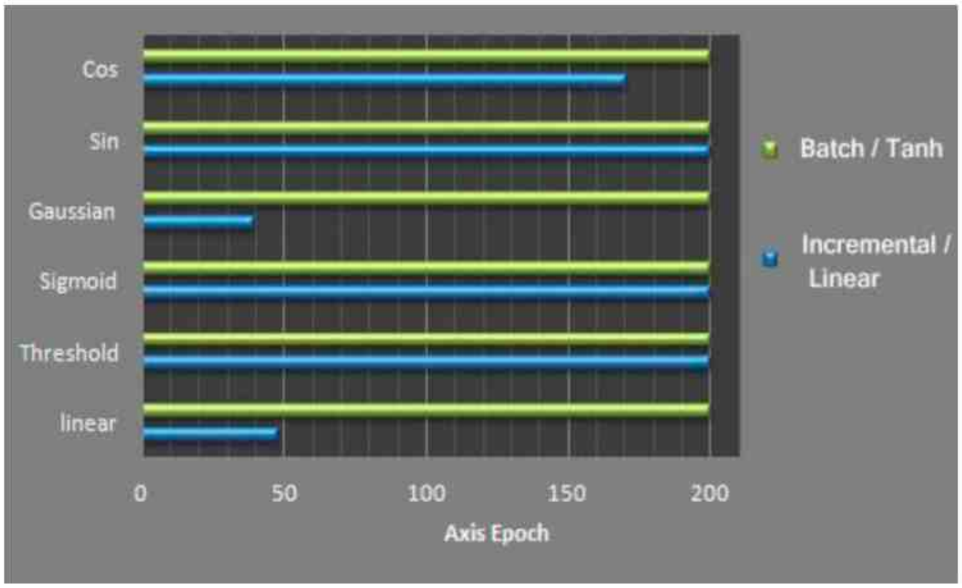


Figure 5.73: Comparison between Batch, and Incremental Algorithm of Different Activation Functions.
Horizontal bar presents epochs and vertical bar presents mean square of error.

Chapter 6: Conclusions

6.1 Summary

In this project, we accomplished classification of emotional states into four categories of low arousal negative, high arousal negative, low arousal positive, and high arousal positive from Computer Science perspective. The project encompassed three stages namely: data collection, feature extraction, and classification.

In data collection, after setting up the experiment by playing a number of videos representing various classes of emotional states, EEG brain wave raw data was recorded. The raw data was collected using Emotiv Epoc headset, Research Edition. The interface to Emotiv API was via a C++ program.

Feature extraction is the critical part of the classification, and feature recognition needs domain knowledge and a great deal of data analysis. For this particular problem, the basic statistical analysis of data did not show any trend of separation of data into desired categories. Thus, there was a need to change the dimensionality of the data. Therefore, principal components of time and frequency domains were extracted. The combination of a threshold of the values of these components constituted suitable feature vector for classification.

Finally, the last step was constructing a proper neural network and exploring the best parameters for supervised classification. For the performance evaluation of each strategy, the neural network was constructed, the parameters were set, and the neural network was trained and tested. The statistical details of performance and analysis of numerous topologies, normal and cascade learning, incremental and batch algorithms were presented. Furthermore, various error functions such as linear and tanh in addition to numerous activation functions such as linear, threshold, Gaussian, sigmoid, sin, and cosine were explored. In conclusion, normal topology, incremental learning, and linear error function were found to perform best in combination with Gaussian or linear activation function.

6.2 Challenges

Despite the fact that the localized source of EEG data is very consistent across the population, the signal is random by the time it gets to the cortex because the cortex is individualized similarly to fingerprints. Although the method presented in this project is able to classify the EEG raw data into four general emotional states, stepping further and distinguishing individual emotions needs much more accurate data.

In spite of using one of the best non-invasive EEG acquisition devices (Emotiv headset) at the time of this project, the process of collecting accurate data may require repeating each experience multiple times. There are some fluctuations in the signal's quality. These interruptions may due to dried sensor pads, movement of the headset, etc.

Using the same method for clinical and more advanced problems, and dealing with patients, need more research to ensure the reliability of non-invasive EEG acquisition devices.

Furthermore, feature vector construction for pattern recognition requires deep domain knowledge. Due to the high dimensionality of the data, extracting right features for classifiers is a challenge. Data should be explored from different perspectives to identify features. The nature of such data goes beyond the statistical characteristics of the data and requires studying the data in various domains and in numerous dimensions in addition to exploring the correlation between various variables.

Chapter 7: Future Work

The project could be extended by recognizing patterns for individual feelings in each of the emotional state classes. The emotional state recognition can be used in clinical studies for curing disorders by monitoring the emotional responses of patients in a particular situation.

Emotional detection is used in psychological research such as studying anxiety disorders [18], ADHD, and other disorders like clinically isolated syndrome and multiple sclerosis [19], etc. At the time of this project, another research is in progress at CSUCI to research the psychological perspective of this project.

Furthermore, the result of this project can be utilized in monitoring customers' satisfaction, in addition to usability and user satisfaction test for new products.

The scope of this project could also be expanded to finding patterns for other brain activities and actions. Detecting patterns for different movements, colors, objects, and thoughts would result in a range of applications in human computer interaction.

References

[1] Emotiv Epoc < http://www.emotiv.com>.

[2] FANN Library < http://leenissen.dk/fann/html/files/fann_cascade-h.html >.

[3] Fundamentals of Artificial Neural Networks. The MIT Press, Hassoun, M. H., 1995.

[4] Hertz, J., Krogh, A., and Palmer, R. G., "Introduction to the Theory of Neural Computing." Addison-Wesley Publishing Company, 1991.

[5] Anderson, J. A., "An Introduction to Neural Networks.", the MIT Press, 1995.

[6] Dementia SOS: Colorado's Dementia News and Resource Center.
< http://coloradodementia.org >

[7] Neural Network Theory <fann.sourceforge.net >.

[8] Tettamanzi and Tomassini, "Soft Computing: Integrating Evolutionary, Neural, and Fuzzy Systems", 2001

[9] Brain Plasticity < http://www.positscience.com/brain-resources/brain-plasticity/what-is-brain-plasticity>.

[10] Principal Component < http://en.wikipedia.org/wiki/Principal_component_analysis>.

[11] Fast Fourier Transform < http://en.wikipedia.org/wiki/Fast_Fourier_transform>.

[12] Artificial Neural Networks
< http://en.wikibooks.org/wiki/Artificial_Neural_Networks/Neural_Network_Basics >.

[13] Neural Networks < http://en.wikipedia.org/wiki/Artificial_neural_network>.

[14] Reinforcement Learning < http://www.scholarpedia.org/article/Reinforcement_learning >.

[15] Back Propagation < http://en.wikipedia.org/wiki/Backpropagation>.

[16] Allison Marie Henderson, "Autonomous Interoffice Delivery Robot (AIDeR) Environmental Cue Detection", California State University Channel Islands, 2012.

[17] FANN Explorer < http://leenissen.dk/fann/gui.php >.

[18] Easter J, McClure EB, Monk CS, Dhanani M, Hodgdon H, Leibenluft E, Charney DS, Pine DS, Ernst M.," Emotion recognition deficits in pediatric anxiety disorders: implications for amygdala research. ", National Institute of Mental Health, Bethesda, Maryland, USA, Aug 2005.

[19]Jehna M, Neuper C, Petrovic K, Wallner-Blazek M, Schmidt R, Fuchs S, Fazekas F, Enzinger C," An exploratory study on emotion recognition in patients with a clinically isolated syndrome and multiple sclerosis", Institute for Psychology, Karl Franzens University Graz, Austria, July 2010.

[20] Ali S. AlMejrad," Human Emotions Detection using Brain Wave Signals: A Challenging Biomedical Technology Department, College of Applied Medical Sciences", King Saud University, P.O.Box 10219, Riyadh 11433, Kingdom Saudi Arabia

[21] Links to videos:
http://dl.dropbox.com/u/117743635/mp4/annoying.mp4
http://dl.dropbox.com/u/117743635/mp4/happy.mp4
http://dl.dropbox.com/u/117743635/mp4/relax.mp4
http://dl.dropbox.com/u/117743635/mp4/sad.mp4

Appendix A

Feature Vector Construction Matlab Script.

```
% Ask the user which Subject/ sample directory to analyze.
folder = input('Which Sample Directory would you like to analyze?', 's');

% Get the vector file to save the result vector
vectorFile='C:\Users\Atena\Desktop\vector.txt'

% Open the relax.csv in the directory that the user chose in the previous step.
file= ['C:\Users\Atena\Desktop\EEGLogger\',folder,'\relax.csv'];

% Set the variable headerlinesIn to 1 for starting to read at line 1.
headerlinesIn =1;

% Set the variable delimiterIn to ',' for using ',' delimiter to separate data in the data file.
delimiterIn = ',';

% Import the data with the parameters set in file, deltimiterIn, and headerlineIn variables and save it in
relaxdata variable.
relaxdata = importdata(file,delimiterIn,headerlinesIn);

% Get the data from row 1501 column 2 to row 7500 column 15 (6000 samples in 14 channels)
relaxdata = relaxdata.data(1501:7500,2:15);


% Open the happy.csv in the directory that the user chose in the previous step.
file= ['C:\Users\Atena\Desktop\EEGLogger\',folder,'\happy.csv'];

% Import the data with the parameters set in file, deltimiterIn, and headerlineIn variables and save it in
happydata variable.
happydata = importdata(file,delimiterIn,headerlinesIn);

% Get the data from row 501 column 2 to row 6500 column 15 (6000 samples in 14 channels)
happydata = happydata.data(501:6500,2:15);

% Open the sad.csv in the directory that the user chose in the previous step.
file= ['C:\Users\Atena\Desktop\EEGLogger\',folder,'\sad.csv'];

% Import the data with the parameters set in file, deltimiterIn, and headerlineIn variables and save it in saddata
variable.
saddata = importdata(file,delimiterIn,headerlinesIn);

% Get the data from row 4501 column 2 to row 10500 column 15 (6000 samples in 14 channels)
saddata = saddata.data(4501:10500,2:15);

% Open the annoying.csv in the directory that the user chose in the previous step.
file= ['C:\Users\Atena\Desktop\EEGLogger\',folder,'\annoying.csv'];
```

```
% Import the data with the parameters set in file, deltimiter n, and headerlineIn variables and save it in annoyingdata variable.
annoyingdata = importdata(file,delimiterIn,headerlinesIn);

% Get the data from row 1 column 2 to row 6000 column 15 ( 6000 samples in 14 channels)
annoyingdata = annoyingdata.data(1:6000,2:15);

% Perfome Principal Component Analysis(PCA) on relaxdata and save the result in relaxpca variable.
relaxpca= princomp(relaxdata);

% Perfome Fast Fourier Transform(FFT) on relaxdata and save the result in relaxfft variable.
relaxfft= fft(relaxdata);

% Get the magnitude of the complex numbers resulting from fft and replace the values.
relaxfft=abs(relaxfft);

% Perfome Principal Component Analysis on relaxfft and save the result in relaxfftpca variable.
relaxfftpca= princomp(relaxfft);

% Construct the first part of relax vector by using threshold with the value of 0.5 on relaxpca. Any value in the array that has value larger than 0.5 is set to 1 and the rest is set to 0. Save the result in relaxpcaV variable.
relaxpcaV= any(relaxpca>0.5);

% Construct the second part of relax vector by using threshold with the value of 0.5 on relaxfftpca. Any value in the array that has value larger than 0.5 is set to 1 and the rest is set to 0. Save the result in relaxfftpcaV variable.
relaxfftpcaV= any(relaxfftpca>0.5);

% Construct relax vector  relaxV variable) by appending results of previous steps to relaxV array.
relaxV(1,1:14)= relaxpcaV;
relaxV(1,15:28  = relaxfftpcaV;

% Set the output vector for relax data to  0 0  (first class representation) and save it in relaxoutV variable.
relaxoutV =  0 0';

% Perform PCA on happy data. Save the result in happypca variable.
happypca= princomp(happydata);

% Perform FFT on happy data. Save the result in happyfft variable.
happyfft= fft(happydata);

% Get the magnitude of the complex numbers resulting from fft and replace the values.
happyfft=abs(happyfft);

% Perform PCA on happyfft. Save the result in happyfftpca.
happyfftpca= princomp(happyfft);

% Perform threshold on happypca and happyfftpca to convert values to binary.
happypcaV= any(happypca>0 5);
happyfftpcaV= any(happyfftpca>0.5);

% Add the result binary vectors to happy vector to construct the happy input vector (happyV).
happyV(1,1:14)= happypcaV;
happyV(1,15:28  = happyfftpcaV;
% Set the happy output vector to [0 1] (The second class representation).
happyoutV = [0 1]';

% Perform PCA on sad data. Save the result in sadpca variable.
sadpca= princomp(saddata);

% Perform FFT on sad data. Save the result in sadfft variable.
sadfft= fft(saddata);

% Get the magnitude of the complex numbers resulting from fft and replace the values.
sadfft=abs(sadfft);

% Perform PCA on sadfft. Save the result in sadfftpca.
sadfftpca= princomp(sadfft);


% Perform threshold on sadpca and sadfftpca to convert values to binary.
sadpcaV= any(sadpca>0.5);
sadfftpcaV= any(sadfftpca>0.5);

% Add the result binary vectors to sad vector to construct the sad input vector (sadV).
sadV(1,1:14)= sadpcaV;
sadV(1,15:28)= sadfftpcaV;

% Set the sad output vector to [1 0] ( The third class representation).
sadoutV = [1 0]'

% Perform PCA on annoying data. Save the result in annoypca variable.
annoypca= princomp(annoyingdata);

% Perform FFT on annoying data. Save the result in annoyfft variable.
annoyfft=fft(annoyingdata);

% Get the magnitude of the complex numbers resulting from fft and replace the values.
annoyfft=abs(annoyfft);

% Perform PCA on annoyfft. Save the result in annoyfftpca.
annoyfftpca=princomp(annoyfft);

% Perform threshold on annoypca and annoyfftpca to convert values to binary.
annoypcaV=any(annoypca>0.5);
annoyfftpcaV = any(annoyfftpca>0.5);

% Add the result binary vectors to annoy vector to construct the annoy input vector (annoyV).
annoyV(1,1:14)=annoypcaV;
annoyV(1,15:28)=annoyfftpcaV;

% Set the annoy output vector to [1 1] (The fourth class representation).
annoyoutV = [1 1]';

 % Write the results of input and output vectors in the vector file so that it matches the FANN Format. – append
parameter makes sure that new vectors are appended to the file and the file is not overwritten.' Delimiter
```

parameter is used to separate matrix elements pc parameter sets the terminator to the vector terminator (implies carriage return/line feed (CR/LF)).

```
dlmwrite(vectorFile  relaxV,'-append','delimiter',   ,'newline','pc');
dlmwrite(vectorFile  relaxoutV, '-append','delimiter',   ,'newline','pc');
dlmwrite(vectorFile  happyV,'-append','delimiter',   ,'newline','pc');
dlmwrite(vectorFile  happyoutV,'-append','delimiter',   ,'newline','pc');
dlmwrite(vectorFile  sadV,'-append','delimiter',  ,'newline','pc');
dlmwrite(vectorFile  sadoutV,'-append','delimiter',   ,'newline','pc');
dlmwrite(vectorFile  annoyV,'-append','delimiter',   ,'newline','pc');
dlmwrite(vectorFile  annoyoutV,'-append','delimiter',' ','newline','pc');
```

Appendix B

An Example of recorded EEG raw data. Each experiment contains 6000 samples of data in 14 channels named based on international brain region namely: Counter, AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, AF4. Values of each channel are in µv.

Counter	F3	FC5	T7	.....	F4	F8	AF4
1	4399.49	4477.44	4278.46	.....	4501.03	4115.38	4399.49
2	4208.21	4404.62	4508.72	.....	4505.13	4135.9	4440
3	4206.67	4414.36	4526.67	.....	4512.82	4136.41	4394.36
.....	.....	.....	.....	.....	.....	.....	.....
6000	4340.51	4472.82	4075.9	.....	4348.72	4001.03	4321.03

A sample of feature vector in FANN format using 2 output neuron presentation after running Matlab script included in Appendix A. Values are binary.

Input Vector: 0 1 1 0 1 1 0 0 1 1 0 1 1 1 0 0 1 0 0 0 0 1 1 1 0 1 1 0
Output Vector: 0 0

Appendix C

The following program makes connection to Emotiv Epoc headset's API in order to extract EEG raw data. The program only works with Emotiv headsets that allow raw EEG access (Research Versions). The process and function calls of this connection is included in Emotiv Epoc's manual along with some examples. The same functions are used for Emotiv API calls here. Parameters and variables are adjusted to work with Emotional State Recognition project.

```
//bci project
// Import libraries and headers
#include <iostream>
#include <fstream>
#include <conio.h>
#include <sstream>
#include <windows.h>
#include <map>

#include "EmoStateDLL.h"
#include "edk.h"
#include "edkErrorCode.h"

#pragma comment(lib, "../lib/edk.lib")

// Create an array to store the content of EEG raw data.
EE_DataChannel_t targetChannelList[] = {
                    ED_COUNTER,
                    ED_AF3, ED_F7, ED_F3, ED_FC5, ED_T7,
                    ED_P7, ED_O1, ED_O2, ED_P8, ED_T8,
                    ED_FC6, ED_F4, ED_F8, ED_AF4, ED_GYROX, ED_GYROY, ED_TIMESTAMP,
                    ED_FUNC_ID, ED_FUNC_VALUE, ED_MARKER, ED_SYNC_SIGNAL
          };

// Create an array to label the content of EEG raw data in the result file.
const char header[] = "COUNTER,AF3,F7,F3, FC5, T7, P7, O1, O2,P8"
              ", T8, FC6, F4,F8, AF4,GYROX, GYROY, TIMESTAMP, "
              "FUNC_ID, FUNC_VALUE, MARKER, SYNC_SIGNAL,";

int main(int argc, char** argv) {

// Create an Even Handler Obj and buffer and initialize parameters.
          EmoEngineEventHandle eEvent        = EE_EmoEngineEventCreate();
          EmoStateHandle eState= EE_EmoStateCreate();
          unsigned int userID= 0;
          const unsigned short composerPort= 1726;
          float secs= 128;
          unsigned int datarate= 0;
          bool readytocollect= false;
          int option= 0;
          int state= 0;
          std::string input;
```

```
//Initialize connection with Emotiv EmoEngine and verify successful connection. The engine will not register any
user at this point.

        try {
                if (EE_EngineConnect() != EDK_OK) {
                        throw std::exception("Emotiv Engine start up failed.");
                }

                std::cout << "Start receiving EEG Data! Press any key to stop logging...\n" << std::endl;


        std::ofstream ofs("log.csv",std::ios::trunc);
                ofs << header << std::endl;

// Access to EEG Data

// Create data handle to provide access to underlying data and initialize it with a call to DataCreate.
                DataHandle hData = EE_DataCreate();

// During the measurement EmotivEngine maintains a data buffer of sample data. At the beginning this buffer
should be initialized by the following call.
                EE_DataSetBufferSizeInSec(secs);

                std::cout << "Buffer size in secs:" << secs << std::endl;

// Start Acquiring Data

                while (!_kbhit()) {
// Get the current EmotivEngine State.

                        state = EE_EngineGetNextEvent(eEvent);

                        if (state == EDK_OK) {
// Get the current EmotivEngine event type to check if there is any update.
                                EE_Event_t eventType = EE_EmoEngineEventGetType(eEvent);

// Get the update for the particular.
                                EE_EmoEngineEventGetUserId(eEvent, &userID);

// Log the EmoState if it has been updated. User add event will trigger user registeration in the connection.
                                if (eventType == EE_UserAdded) {
                                        std::cout << "User added";
// After user registration, the data acquisition must be enabled for that user in the connection.
                                        EE_DataAcquisitionEnable(userID,true);
                                        readytocollect = true;
                                }
                        }

// Acquiring Data

                        if (readytocollect) {

// Initiate retrieval of the latest EEG buffered data.
```

```
                                        EE_DataUpdateHandle(0, hData);
                                        unsigned int nSamplesTaken=0;

// Get the amount of buffer available in the buffer  hData contains the latest buffer data.
                                        EE_DataGetNumberOfSample(hData,&nSamplesTaken);

                                        std::cout << "Updated " << nSamplesTaken << std::endl;

// Transfer data into a buffer in the application and write into the file.
        if (nSamplesTaken != 0) {

          double* data = new double[nSamplesTaken];
      for (int sampleIdx=0 ; sampleIdx<(int)nSamplesTaken ; ++ sampleIdx)      {
           for (int i = 0 ; i<sizeof(targetChannelList)/sizeof(EE_DataChannel_t) ; i++)                    {
                 EE_DataGet(hData, targetChannelList[i], data, nSamplesTaken);
                 ofs << data[sampleIdx] << ",";
                      }
               ofs << std::endl;
               }
         delete[] data;
        }

}

Sleep(100);
}

               ofs.close();
               EE_DataFree(hData);

        }
        catch (const std::exception& e) {
               std::cerr << e.what() << std::endl;
               std::cout << "Press any key to exit..." << std::endl;
               getchar();
        }
// Terminate the connection with EmoEngine and free the memory allocated to buffer and event handler.
        EE_EngineDisconnect();
        EE_EmoStateFree(eState);
        EE_EmoEngineEventFree(eEvent);

        return 0;
}
```

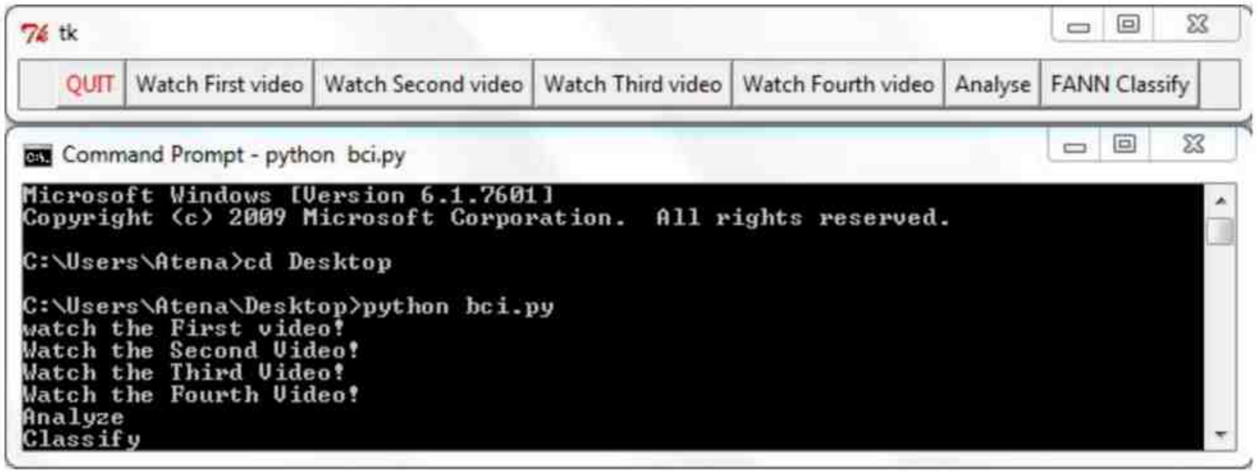
Appendix D

Following python script is coded in order to speed the process of watching videos, analyzing data and sending the result to FANN library. The following script runs the required programs and sends the proper files to those programs in each step.

```
# import the libraries
import tkinter
import subprocess
import os
from tkinter import *

class Application(Frame):
# Watch the first video function
  def watch_first_video_function (self):
    print ("watch the First video!")
# Create and initialize variables for video and sample file path location.
    video = "C:\\Users\Atena\Desktop\\video\mp4\\relax.mp4"
    samplefilePath = "C:\\Users\Atena\Desktop\EEGLogger\sample\\relax.csv"
# Create global vaiables
    global emotivAPI
    global excel
    global visualStudio
    global mediaPlayer
    global realPlayer
    global matlab
# Create and initialize variables for the path various applications.
    matlab = "C:\Program Files\MATLAB\R2009a\bin\matlab.exe"
    emotivAPI = "C:\\Users\Atena\Desktop\EEGLogger\EEGLogger.vcxproj"
    excel = "C:\Program Files\Microsoft Office\Office12\excel.exe"
     visualStudio = "C:\Program Files\Microsoft Visual Studio 10.0\Common7\IDE\devenv.exe"

  # Open emotvAPI program (C++ API Connection in Appendix C) in Visual Studio so that the program can collect
EEG raw data while subjects are watching the videos.
    subprocess.Popen("%s %s" % (visualStudio, emotivAPI))
#Start the video
    os.startfile(video)
# In each step emotivAPI program writes the result in a file.

# Watch the second Video Function
  def watch_second_video_function (self):
    print("Watch the Second Video!")
    video = "C:\\Users\Atena\Desktop\\video\mp4\sad.mp4"
    os.startfile(video)

# Watch the second Video Function
  def watch_third_video_function (self):
        print("Watch the Third Video!")
        video = "C:\\Users\Atena\Desktop\\video\mp4\happy.mp4"
        os.startfile(video)

# Watch the second Video Function
    def watch_fourth_video_function (self):
        print("Watch the Fourth Video!")
        video = "C:\\Users\Atena\Desktop\\video\mp4\\annoying.mp4"
        os.startfile(video)

# Analyse function
    def analyse_function (self):
        matlab = "C:\Program Files\\MATLAB\\R2009a\\bin\\matlab.exe"
        samplefilePath = "C:\\Users\Atena\Desktop\\analyze.m"
        print("Analyze")
# Open Matlab script for creating feature vectors from raw data (Appendix A) in matlab
        subprocess.Popen("%s %s" % (matlab, samplefilePath))
#FANN Function
    def fann_function (self):
        print("Classify")
 # Create and initialize variables for FANN Kernel and Explorer
fannKernel="C:\\Users\Atena\Desktop\\fannExplorer21\\fann\\fannSoap\\fannKernel\\Release\\fannKernel.exe"
fannExplorer="C:\\Users\Atena\Desktop\\fannExplorer21\\fann\\fannSoap\\fannKernel\\Release\\fannExplorer\\
fannExplorer.html"

  # Execute FANN Kernel
        os.startfile(fannKernel)
# Start FANN Explorer
        os.startfile(fannExplorer)

# Create the control window and call the functions when buttons are clicked.
    def createWidgets(self):
# Quite Button
        self.QUIT = Button(self)
        self.QUIT["text"] = "QUIT"
        self.QUIT["fg"] = "red"
        self.QUIT["command"] = self.quit
        self.QUIT.pack({"side": "left"})
# Watch First Video Button
        self.watch_first_video = Button(self)
        self.watch_first_video["text"] = "Watch First video"
        self.watch_first_video["command"] = self.watch_first_video_function
        self.watch_first_video.pack({"side": "left"})
```

```
# Watch Second Video Button
    self watch_second_video = Button(self)
    self watch_second_video["text"] =  Watch Second video"
    self watch_second_video["command"  = self watch_second_video_function
    self watch_second_video.pack({"side"  'left"})

# Watch Third Video Button
    self watch_third_video = Button(self)
    self watch_third_video[' text" = 'Watch Third video"
    self watch_third_video["command"  = self watch_third_video_function
    self watch_third_video.pack({"side"  'left"})

# Watch Fourth Video Button
    self watch_fourth_video = Button(self)
    self watch_fourth_video[' text"  =  Watch Fourth video"
    self watch_fourth_video["command" = self watch_fourth_video_function
    self watch_fourth_video.pack({' side"  'left"})

# Analyse Button
    self analyse = Button(self)
    self analyse[' text"  =  Analyse"
    self analyse["command" = self analyze_function
    self analyse ({"side"  'left"})

# FANN Button
    self fann = Button(self)
    self fann[' text"  = 'FANN Classify"
    self fann["command" = self fann_function
    self fann({"side"  'left"})

  def __init__(self master=None):
    Frame.__init__(self master)
    self pack()
    self createWidgets()
root = Tk()
app = Application(master=root)
app.mainloop()
root destroy()
```

Screen Shot of the control window:



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Atena>cd Desktop

C:\Users\Atena\Desktop>python bci.py
watch the First video!
Watch the Second Video!
Watch the Third Video!
Watch the Fourth Video!
Analyze
Classify
```

Figure Appendix D: Control window.