INDEX

List of Figures

PROJECT TEAM LEAD: VISHWA TEJ REDDY
TEAM MEMBERS: SAI SIDDARTHA, SAI SIDDARTHA, SHAIK AMER SOHAIL

List of Tables / Flow charts

ABSTRACT

A Neural Network is a machine that is designed to model the way in which the brain performs a task or function of interest. It has the ability to perform complex computations with ease. The objective of this project was to investigate the use of ANNs in the field of Cryptography. Cryptography is defined as the exchange of data into mix code. Cryptography is the technique of changing data or information into unreadable for unauthorized persons. In cryptography process information transfer from sender to receiver in a manner that prevents from unauthorized third person. There are so many cryptography methods are available which based on number theory. The cryptography based on number system has some disadvantages such as large computational power, complexity and time consumption. To overcome all the disadvantages of number theory based cryptography, Artificial neural network based cryptography is used.

CHAPTER 1

INTRODUCTION

1.1 ARTIFICIAL NEURAL NETWORK

1.1.1 Introduction

Work on artificial neural network has been motivated right from its inception by the recognition that the human brain computes in an entirely different way from the conventional digital computer. The brain is a highly complex, nonlinear and parallel information processing system. It has the capability to organize its structural constituents, known as neurons, so as to perform certain computations many times faster than the fastest digital computer in existence today. The brain routinely accomplishes perceptual recognition tasks, e.g. recognizing a familiar face embedded in an unfamiliar scene, in approximately 100-200 ms, whereas tasks of much lesser complexity may take days on a conventional computer.

A neural network is a machine that is designed to model the way in which the brain performs a particular task. The network is implemented by using electronic components or is simulated in software on a digital computer. A neural network is a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experimental knowledge and making it available for use. It resembles the brain in two respects:

1. Knowledge is acquired by the network from its environment through a learning process.

2. Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.

Neural networks, with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. Other advantages include:

1. Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.

2. Self-Organization: An ANN can create its own organization or representation of the information it receives during learning time.

3. Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.

1.1.2 Biological Model

The human nervous system can be broken down into three stages that may be represented as follows:

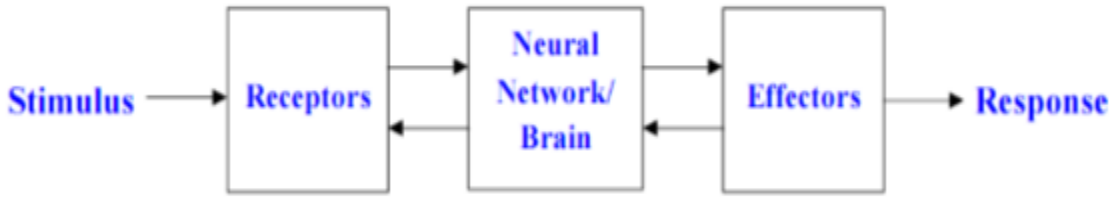


Fig 1.1: Block Diagram of a Human Nervous System

The receptors collect information from the environment. The effectors generate interactions with the environment e.g. activate muscles. The flow of information/activation is represented by arrows. There is a hierarchy of interwoven levels of organization:

1. Molecules and Ions
2. Synapses
3. Neuronal microcircuits
4. Dendritic trees
5. Neurons
6. Local circuits
7. Inter-regional circuits
8. Central nervous system

There are approximately 10 billion neurons in the human cortex. Each biological neuron is connected to several thousands of other neurons. The typical operating speed of biological neurons is measured in milliseconds. The majority of neurons encode their activations or outputs as a series of brief electrical pulses. The neuro's cell body processes the incoming activations and converts the into output activations. The neurons nucleus contains the genetic material in the form of DNA. This exists in most types of cells. Dendrites are fibers which emanate from the cell body and provide the receptive zones that receive activation from other neurons. Axons are fibers acting as transmission lines that send activation to other neurons. The junctions that allow signal transmission between axons and dendrites are called synapses.

1.1.3 ANN Structure

An artificial neural network consists of a pool of simple processing units which communicate by sending signals to each other over a large number of weighted connections. A set of major aspects of ANN are:

1) A set of processing units ('neurons,' 'cells');
2) A state of activation yk for every unit, which equivalent to the output of the unit;
3) Connections between the units. Generally each connection is defined by a weight wjk which determines the effect which the signal of unit j has on unit k;
4) A propagation rule, which determines the effective input sk of a unit from its external inputs;
5) An activation function Fk, which determines the new level of activation based on the effective input sk(t) and the current activation yk(t) (i.e., the update);
6) An external input (aka bias, offset) θk for each unit;
7) A method for information gathering (the learning rule);
8) An environment within which the system must operate, providing input signals and if necessary error signals.

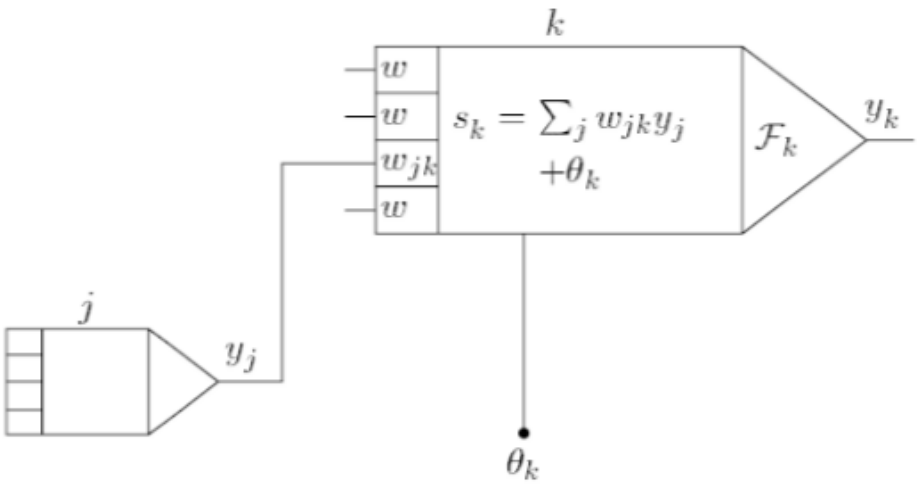


Fig 1.2: The basic components of an artificial neural network

1.1.4 Network Architectures

Multilayer feed forward Networks:

The second class of a feed forward neural network distinguishes itself by the presence of one or more hidden layers. The function of hidden neuron is to intervene between the external input and the network output in some useful manner. By adding more hidden layers, the network is enabled to extract higher order statistics. The input signal is applied to the neurons in the second layer. The output signal of second layer is used as inputs to the third layer, and so on for the rest of the network.

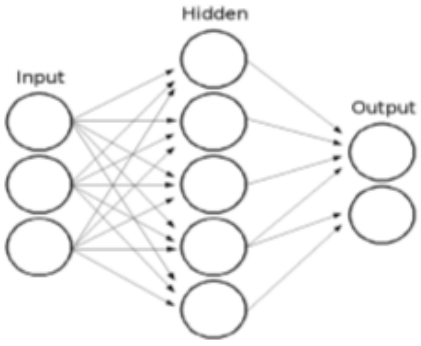


Fig 1.3: Multi-layer Feed-forward Network

1.1.5 Learning Processes

By learning rule we mean a procedure for modifying the weights and biases of a network. The purpose of learning rule is to train the network to perform some task. They fall into three broad categories:

1. **Supervised learning :** The learning rule is provided with a set of training data of proper network behavior. As the inputs are applied to the network, the network outputs are compared to the targets. The learning rule is then used to adjust the weights and biases of the network in order to move the network outputs closer to the targets.

2. **Reinforcement learning:** It is similar to supervised learning, except that, instead of being provided with the correct output for each network input, the algorithm is only given a grade. The grade is a measure of the network performance over some sequence of inputs.

3. **Unsupervised learning:** The weights and biases are modified in response to network inputs only. There are no target outputs available. Most of these algorithms perform some kind of clustering operation. They learn to categorize the input patterns into a finite number of classes.

1.1.6 Training of Artificial Neural Networks

A neural network has to be configured such that the application of a set of inputs produces (either 'direct' or via a relaxation process) the desired set of outputs. Various methods to set the strengths of the connections exist. One way is to set the weights explicitly, using a priori knowledge. Another way is to 'train' the neural network by feeding it teaching patterns and letting it change its weights according to some learning rule.

1.1.7 Back propagation

A single-layer network has severe restrictions: the class of tasks that can be accomplished is very limited. Minsky and Papert showed in 1969 that a two layer feed-forward network can overcome many restrictions, but did not present a solution to the problem of how to adjust the weights from input to hidden units. An answer to this question was presented by Rumelhart, Hinton and Williams in 1986 and similar

solutions appeared to have been published earlier. The central idea behind this solution is that the errors for the units of the hidden layer are determined by back-propagating the errors of the units of the output layer. For this reason the method is often called the back-propagation learning rule. Back-propagation can also be considered as a generalization of the delta rule for non-linear activation functions and multilayer networks.

1.1.8 The Generalized delta rule

This procedure constitutes the generalized delta rule for a feed-forward network of non-linear units. the whole back-propagation process is intuitively very clear. What happens in the above equations is the following. When a learning pattern is clamped, the activation values are propagated to the output units, and the actual network output is compared with the desired output values, we usually end up with an error in each of the output units. The simplest method to do this is the greedy method: we strive to change the connections in the neural network in such a way that, next time around, the error eo will be zero for this particular pattern.

That's step one. But it alone is not enough: when we only apply this rule, the weights from input to hidden units are never changed, and we do not have the full representational power of the feed-forward network as promised by the universal approximation theorem. In order to adapt the weights from input to hidden units, we again want to apply the delta rule. In this case, however, we do not have a value for δ for the hidden units. This is solved by the chain rule which does the following: distribute the error of an output unit o to all the hidden units that is it connected to, weighted by this connection. Differently put, a hidden unit h receives a delta from each output unit o equal to the delta of that output unit weighted with (= multiplied by) the weight of the connection between those units.

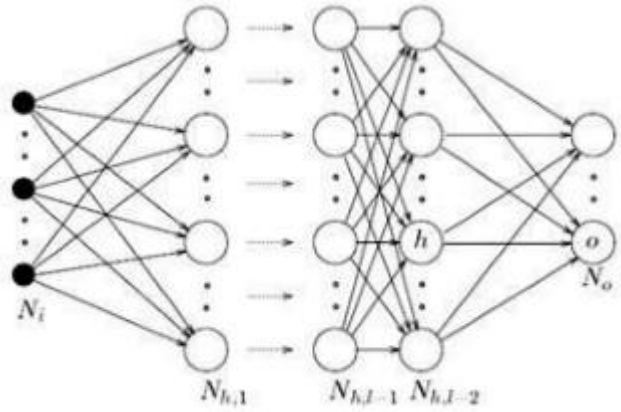


Fig 1.4: A multi-layer network with l layers of units

1.2 Cryptography

1.2.1 Introduction

There are many aspects to security and many applications, ranging from secure commerce and payments to private communications and protecting passwords. One essential aspect for secure communications is that of cryptography. Cryptography is the science of writing in secret code and is an ancient art; the first documented use of cryptography in writing dates back to circa 1900 B.C. when an Egyptian scribe used non-standard hieroglyphs in an inscription. Some experts argue that cryptography appeared spontaneously sometime after writing was invented, with applications ranging from diplomatic missives to wartime battle plans. It is no surprise, then, that new forms of cryptography came soon after the widespread development of computer communications. In data and telecommunications, cryptography is necessary when communicating over any untrusted medium, which includes just about any network, particularly the Internet. Cryptography, then, not only protects data from theft or alteration, but can also be used for user authentication. There are, in general, three types of cryptographic schemes typically used to accomplish these goals: secret key (or symmetric) cryptography, public-key (or asymmetric) cryptography, and hash functions, each of which is described below. In all cases, the initial unencrypted data is referred to as plaintext. It is encrypted into ciphertext, which will in turn (usually) be decrypted into usable plaintext.

1.2.2 Types of Cryptographic Algorithms

There are several ways of classifying cryptographic algorithms. Here they will be categorized based on the number of keys that are employed for encryption and decryption. The three types of algorithms are:

a) **Secret Key Cryptography -** With secret key cryptography, a single key is used for both encryption and decryption. As shown in the figure, the sender uses the key (or some set of rules) to encrypt the plaintext and sends the ciphertext to the receiver. The receiver applies the same key (or ruleset) to decrypt the message and recover the plaintext. Because a single key is used for both functions, secret key cryptography is also called symmetric encryption.

b) **Public Key Encryption** - Public-key cryptography has been said to be the most significant new development in cryptography in the last 300-400 years. Modern PKC was first described publicly by Stanford University professor Martin Hellman and graduate student Whitfield Diffie in 1976. Their paper described a two-key crypto system in which two parties could engage in a secure communication over a non-secure communications channel without having to share a secret key. In PKC, one of the keys is designated the public key and may be advertised as widely as the owner wants. The other key is designated the private key and is never revealed to another party. It is straight forward to send messages under this scheme.

c) **Hash Functions** - Hash functions, also called message digests and one-way encryption, are algorithms that, in some sense, use no key. Instead, a fixed-length hash value is computed based upon the plaintext that makes it impossible for either the contents or length of the plaintext to be recovered. Hash algorithms are typically used to provide a digital fingerprint of a file's contents, often used to ensure that the file has not been altered by an intruder or virus. Hash functions are also commonly employed by many operating systems to encrypt passwords. Hash functions, then, provide a measure of the integrity of a file.

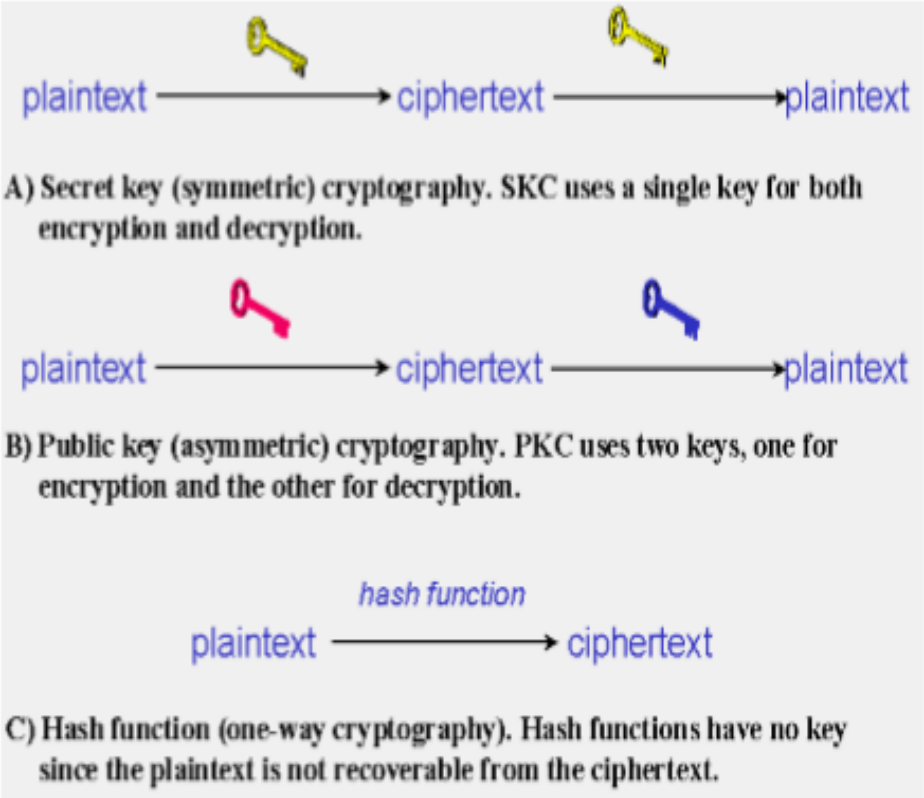


Fig 1.5: Different Encryption Algorithms

CHAPTER 2

Implementation of Neural Network

2.1 Sequential Machine

A ``sequential machine" is a device in which the output depends in some systematic way on variables other than the immediate inputs to the device. These ``other variables" are called the state variables for the machine, and depend on the history or state of the machine. For example, in a counter, the state variables are the values stored in the flip flops. The essence of a state table can be captured in a state diagram. A state diagram is a graph with labeled nodes and arcs; the nodes are the states, and the arcs are the possible transitions between states.

In the project we have used the fact that the output of the sequential machine depend on the state of the machine as well as the input given to the sequential machine. Therefore we have used a Jordan network in which a few outputs are used as inputs, these outputs denote the states. A multilayered neural network is designed on this basis which has a hard limiter in the output layer as a transfer function. The network has 3 layers an input layer, a hidden layer and an output layer. The size of the input layer depends on the number of inputs and the number of outputs being used to denote the states. The learning algorithm used for this network is back propagation algorithm and the transfer function in the hidden layer is a sigmoid function. For implementation of sequential machine a serial adder and a sequential decoder is used.

2.2 Cryptography Using Sequential Machine

For a sequential Machine, the output depends on the input as well as the state of the machine. Thus a sequential machine can be used in cryptography where the input data stream is the input to the sequential machine and the state determines the output input relationship. We can use the state of the sequential machine as the key and then use the data as an input to the sequential machine. The relationship between different output and states can be any random but unique sequence providing security to the encryption. As a sequential machine can be implemented by using a neural network, therefore a neural network can be used to encrypt data and another to decrypt data. In this case the

starting state of the sequential machine can act as a key. For this application the state diagram is drawn and the data is used to train the neural network as it provides the way the machine moves from one state to another.

2.2.1 Sequential Machine Implementation

A finite state sequential machine was implemented using a Jordan network is used. In the Jordan network, the activation values of the output units are fed back into the input layer through a set of extra input units called the state units. There are as many state units as there are output units in the network. The connections between the output and state units have a fixed weight of +1 and learning takes place only in the connections between input and hidden units as well as hidden and output units. Thus all the learning rules derived for the multi-layer perceptron can be used to train this network.

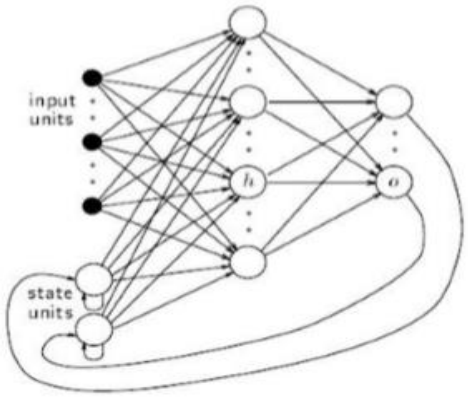


Fig 2.1 : Jordan Network

To train the Jordan network back propagation algorithm was used. The application of the generalized delta rule thus involves two phases: During the first phase the input x is presented and propagated forward through the network to compute the output values Ypo for each output unit. This output is compared with its desired value do, resulting in an error signal δpo for each output unit. The second phase involves a backward pass through the network during which the error signal is passed to each unit in the network and appropriate weight changes are calculated.

2.2.2 Weight adjustments with sigmoid activation function

Recall that in order for a neural networks to learn, weights associated with neuron connections must be updated after forward passes of data through the network. These weights are adjusted to help reconcile the differences between the actual and predicted outcomes for subsequent forward passes. But how, exactly, do the weights get adjusted?

First Phase: The input x is presented and propagated forward through the network to compute the output values yp for each output unit. This output is compared with its desired value do, resulting in an error signal δp for each output unit.

The Second Phase: This involves a backward pass through the network during which the error signal is passed to each unit in the network and appropriate weight changes are calculated.

For the implementation of the sequential machine the state table is used as input and the outputs as well as next states are used as the combined output for the Jordan network. Depending upon the size of the dataset the size of the hidden layer is changed as the complexity of the sequential machine increases. In sequential logic two implementations are done namely:-

a. Serial adder

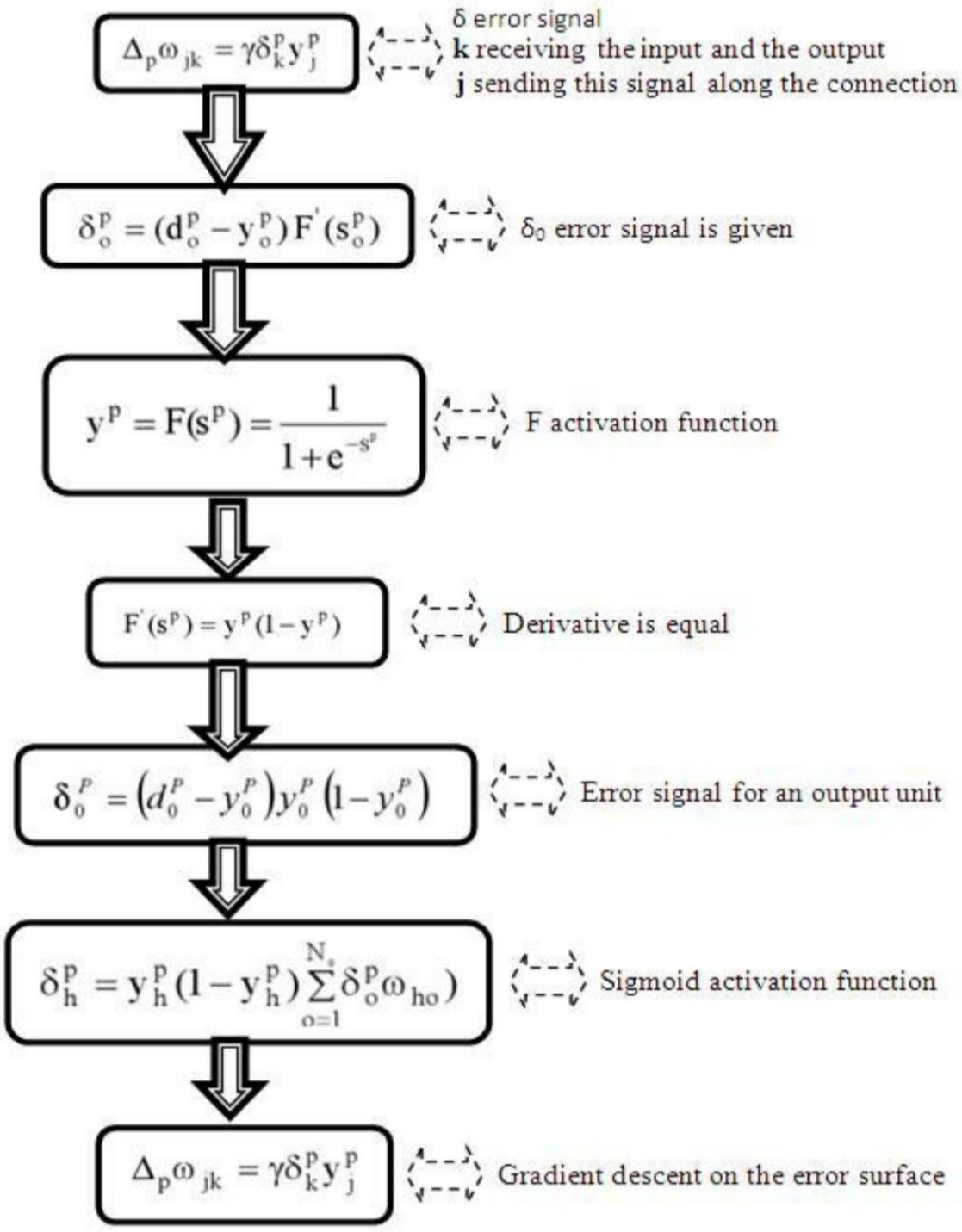
b. Sequence detector

Serial Adder

The serial adder accepts as input two serial strings of digits of arbitrary length, starting with the low order bits, and produces the sum of the two bit streams as its output. (The input bit streams could come from, say, two shift registers clocked simultaneously.) This device can be easily described as a state machine.

Sequential Detector

A state machine is required which outputs logic 1 whenever a particular sequence is detected in the input data stream, and which outputs a zero otherwise. The input is supplied serially, one bit at a time.



Flowchart 2.1: Weight adjustments with sigmoid activation function

2.3 Cryptography using ANN based sequential machine

The use of Sequential machine has been discussed in the previous section. A sequential machine using a Jordan network has been implemented using the back-propagation algorithm. For use of sequential machine for encryption and decryption, a state diagram is drawn and a state table is obtained. Using the state table, a training set is generated. The input set includes all the possible inputs and states possible whereas the output consists of the encrypted/decrypted output and the next state. The reason for using sequential machine for implementation is that the output and input can have any type of relationship and the output depends on the starting state. The starting state is used as a key for encryption and decryption.

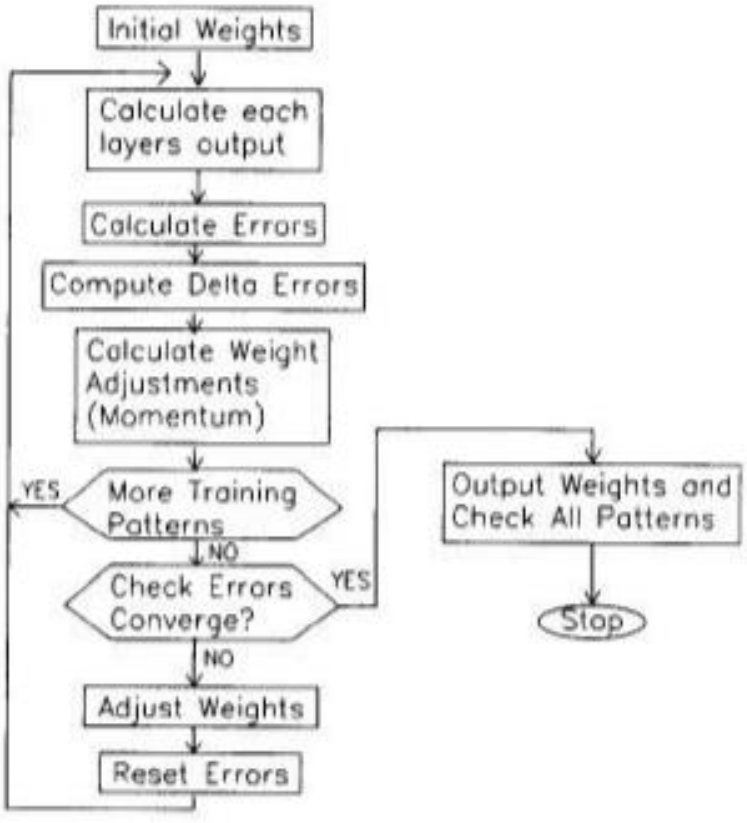


Table 2.2: Program Flow Chart

2.4 TCP/IP Implementation

Transmission Control Protocol (TCP) is a transport protocol layered on top of the Internet Protocol (IP) and is one of the most used networking protocols. The TCP/IP client support uses raw socket communication and lets you connect to remote hosts from MATLAB for reading and writing data. For example, you could use it to acquire data from a remote weather station, and plot the data.

Connection based protocol : The two ends of the communication link must be connected at all times during the communication.

Streaming protocol: TCP/IP has a long stream of data that is transmitted from one end of the connection to the other end, and another long stream of data flowing in the opposite direction. The TCP/IP stack at one end is responsible for breaking the stream of data into packets and sending those packets, while the stack at the other end is responsible for reassembling the packets into a data stream using information in the packet headers.

Reliable protocol : The packets sent by TCP/IP contain a unique sequence number. The starting sequence number is communicated to the other side at the beginning of communication. The receiver acknowledges each packet, and the acknowledgment contains the sequence number so that the sender knows which packet was acknowledged. This method implies that any packets lost on the way can be retransmitted because the sender would know that packets did not reach their destination because it had not received an acknowledgment. Also, packets that arrive out of sequence can be reassembled in the proper order by the receiver.

Timeouts can be established because the sender knows (from the first few packets) how long it takes on average for a packet to be sent and its acknowledgment received.

You can create a TCP/IP connection to a server or hardware and perform read/write operations. Use the **tcpclient** function to create the connection, and the `write` and `read` functions for synchronously reading and writing data.

CHAPTER 3

LITERATURE SURVEY

This literature review looks at the research that has been published in the area of cryptography as it relates to network data and global communications security. It compares and contrasts the research pointing out overall trends in what has already been published on this subject. It analyses the role that cryptography has played and will play in the future relative to security. This review addresses cryptography around the central theme of the security that it provides or should provide individuals, corporations, and others in the modern age of computing technology, networking, and Web-based ecommerce. By reviewing both scholarly and non-scholarly works, it is our objective to make a case that continuing research into the use of cryptography is paramount in preserving the future of electronic data security and privacy as well as the continuing development of Web-based applications that will permit the growth of ecommerce business worldwide to be conducted over the Internet.

Cryptography provides number of security goals to ensure of privacy of data, on-alteration of data and so on. The idea of encryption and encryption algorithm by which we can encode our data in secret code and not to be able readable by hackers or unauthorized person even it is hacked. The main reason for not using encryption in email communications is that current email encryption solutions and hard key management. Different encryption techniques for promoting the information security. In this paper we present a survey paper on cryptographic techniques based on some algorithm and which is suitable for many applications where security is main concern.

Purpose of Cryptography

1) Authentication: Authentication mechanisms help to establish proof of identities. This process ensures that the origin of the message is correctly identified.

2) Confidentiality: The principle of confidentiality specifies that only the sender and the intended recipient should be able to process the contents of a message.

3) Availability: The principle of availability states that resources should be available to authorized parties all the times.

4) Integrity: The integrity mechanism ensures that the contents of the message remain the same when it reaches the intended recipient as sent by the sender.

5) Access Control: Access Control specifies and controls who can access the process.

RELATED WORKS

3.1 DES

DES is a block cipher that uses shared secret key for encryption and decryption. DES algorithm as described by Davis R takes a fixed length of string in plaintext bits and transforms it through a series of operations into cipher text bit sting of the same length and its each block is 64 bits. There are 16 identical stages of processing, termed rounds. There is also an initial and final permutation which named as IP and FP

3.2 3DES

3DES is an enhancement of DES and it is 64 bit block size with 192 bits key size. In this standard the encryption of method is similar to the one in the original DES and increase the encryption level and the average safe time. In 3DES is slower than other block cipher methods. It uses either two or three 56 bit keys in the sequence order of Encrypt-Decrypt-Encrypt. TDES algorithm with three keys require 2168 chances of combinations and with two keys requires 2112 combinations; and the disadvantage of this algorithm is too time consuming problem.

3.3 AES

In AES is the almost identical of block cipher Rijndael cipher developed by two Belgian cryptographers, Joan and Vincent Rijmen. The algorithm explains about by AES is a secret-key algorithm which means of the same key is used for both encrypting and decrypting the data. AES on the other hand which encrypts all 128 bits in one iteration. This is one reason why it has a comparably small number of rounds. AES encryption is fast and flexible. It can be implemented on various platforms especially in small device.

3.4 Blowfish

Blowfish is one of the most common public domain encryption algorithm provided by Bruce Schneier one of the world's leading cryptologists, and the president of Counterpane Systems .

Blowfish encrypts 64-bits block cipher with variety length key, it contains two parts.

Data Encryption: Its involves the iteration of a simple function of 16 times. Each round contains a key dependent permutation and data dependent substitution. **Subkey Generation**: Its involves converts the key upto 448 bits long to 4168 bits.

3.5 RSA

RSA is a public key algorithm invented by Rivest, Shamir, Adleman [7]. RSA involves a public key and a private key. The public key can be known to everyone and is used for encrypting messages. Messages encrypted with the public key can only be decrypted using the private key. These keys for the RSA algorithm are generated in many ways.

Algorithm	Created by	Key Size (in bits)	Block Size (in bits)
DES	IBM in year 1975	56	64
3DES	IBM in year 1978	112 (or) 168	64
AES	Joan Daemen and Vincent Rijmen in year 1998	256	128
Blowfish	Bruce Schneier in year 1993	32 (or) 448	64

Table 3.1. Cryptography Algorithms – Comparison

This survey gives a detailed study of Cryptography Techniques like AES, DES, 3DES, Blowfish, RSA, CL-PKC. Among those algorithms and concepts the security for the data has become highly important since the selling and buying of products over the open network occur very frequently. In this paper it has been surveyed about the existing works on the encryption techniques. This paper presents the performance evaluation of selected symmetric algorithms. The selected algorithms are AES, 3DES, Blowfish and DES.

CHAPTER 4

SOFTWARE REQUIRMENTS

MATLAB

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

- Math and computation
- Algorithm development
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including Graphical User Interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non-interactive language such as C or Fortran.

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects, which together represent the state-of-the-art in software for matrix computation.

MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of application-specific solutions called toolboxes. Very important to most users of MATLAB, toolboxes allow to learn and apply specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in

which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

The MATLAB System

The MATLAB system consists of five main parts:

The MATLAB language:

This is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both "programming in the small" to rapidly create quick and dirty throw-away programs, and "programming in the large" to create complete large and complex application programs.

The MATLAB working environment:

This is the set of tools and facilities that you work with as the MATLAB user or programmer. It includes facilities for managing the variables in your workspace and importing and exporting data. It also includes tools for developing, managing, debugging, and profiling M-files, MATLAB's applications.

Handle Graphics:

This is the MATLAB graphics system. It includes high-level commands for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level commands that allow you to fully customize the appearance of graphics as well as to build complete Graphical User Interfaces on your MATLAB applications.

The MATLAB mathematical function library

This is a vast collection of computational algorithms ranging from elementary functions like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix eigenvalues, Bessel functions, and fast Fourier transforms.

The MATLAB Application Program Interface (API).

This is a library that allows you to write C and Fortran programs that interact with MATLAB. It include facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.

4.1 Initialization and functioning of TCP/IP

Create TCPIP object:

```
t = tcpip(RemoteHost)
t = tcpip(RemoteHost,RemotePort)
t = tcpip(__,Name,Value)
```

t = tcpip(RemoteHost) creates a TCPIP object, `t`, associated with remote host RemoteHost and the default remote port value of 80.

When the TCPIP object is created, its `Status` property value is `closed`. When the object is connected to the host with the fopen function, the `Status` property is configured to open.

The default local host in multihome hosts is the system default. The LocalPort property defaults to a value of `[]`, which allows any free local port to be used. LocalPort is assigned when fopen is issued.

t = tcpip(RemoteHost,RemotePort) creates a TCPIP object with the specified remote port value (RemotePort).

t = tcpip(___,Name,Value) creates a TCPIP object with the specified optional name-value pairs. If an invalid property name or property value is specified, the object is not created.

For example in our case:

t=tcpip('0.0.0.0',30000,'NetworkRole','Server','TimeOut',2);

Remote host is 0.0.0.0

Remote port is 30000

Network Role is one which specifies whether it is a client or server

Time out parameter that specifies the maximum amount of time that transmitted data may remain unacknowledged before **TCP** will forcefully close the corresponding connection.(our choice we can represent it or not).

fopen(tcpip object name) is used to open the connection.

fprintf(tcpip object name,string) is the command used to send the information which the sender sends to the receiver. The information should be a string.

fwrite(tcpip object, variable) is used to send information to receiver but information is an integer.

At receiver the TCPIP object should be created to receive the information sent by the sender and the tcp ip object at the receiver should contain the port number and Here in the code we specified Local Host as the exchange of the information takes place in same system but different Mat lab Sessions and we should also mention the Network Role and Timeout.

fscanf(tcp ip object, format specifier) : This command helps to get / read the string or the message transmitted by the sender.

fread(tcpip object,num): this command helps to get / read the integer values which were transmitted by the sender and num indicates how many integer values were send by the sender.

CHAPTER 5

SOURCE CODE

5.1 Encryption Training:

```
clc;
clear all;
close all;
inpu=7;                 %no. of bits of data
outp=7;                 %no of output bits
key=2;                  %no. of keys
states=1;               %key length
hl=24;                  % size of the hidden layer
w1=rand(hl,(inpu+states+1));   %generating random weights
w2=rand((outp+states),hl+1);   %generating random weights
neta=1;                 %learning rate
a=1;                    %indicate the row
sx=0;                   %iterations
m=[];                   %generating a matrix to display mean square error
training_setx=[];       %initialization of training matrix
training_setx=[0 0 0 0 0 0 0 0;0 0 0 0 0 0 1 0;0 0 0 0 0 1 0 0;0 0 0 0 0 1 1 0;0 0 0 0 1 0
0 0;0 0 0 0 1 0 1 0;0 0 0 0 1 1 0 0;0 0 0 0 1 1 1 0;0 0 0 1 0 0 0 0;0 0 0 1 0 0 1 0;0 0 0 1 0
1 0 0;0 0 0 1 0 1 1 0;0 0 0 1 1 0 0 0;0 0 0 1 1 0 1 0;0 0 0 1 1 1 0 0;0 0 0 1 1 1 1 0;0 0 1 0
0 0 0 0;0 0 1 0 0 0 1 0;0 0 1 0 0 1 0 0;0 0 1 0 0 1 1 0;0 0 1 0 1 0 0 0;0 0 1 0 1 0 1 0;0 0 1
0 1 1 0 0;0 0 1 0 1 1 1 0;0 0 1 1 0 0 0 0;0 0 1 1 0 0 1 0;0 0 1 1 0 1 0 0;0 0 1 1 0 1 1 0;0 0
1 1 1 0 0 0;0 0 1 1 1 0 1 0;0 0 1 1 1 1 0 0;0 0 1 1 1 1 1 0;0 1 0 0 0 0 0 0;0 1 0 0 0 0 1 0;0
1 0 0 0 1 0 0;0 1 0 0 0 1 1 0;0 1 0 0 1 0 0 0;0 1 0 0 1 0 1 0;0 1 0 0 1 1 0 0;0 1 0 0 1 1 1
0;0 1 0 1 0 0 0 0;0 1 0 1 0 0 1 0;0 1 0 1 0 1 0 0;0 1 0 1 0 1 1 0;0 1 0 1 1 0 0 0;0 1 0 1 1 0
1 0;0 1 0 1 1 1 0 0;0 1 0 1 1 1 1 0;0 1 1 0 0 0 0 0;0 1 1 0 0 0 1 0;0 1 1 0 0 1 0 0;0 1 1 0 0
1 1 0;0 1 1 0 1 0 0 0;0 1 1 0 1 0 1 0;0 1 1 0 1 1 0 0;0 1 1 0 1 1 1 0;0 1 1 1 0 0 0 0;0 1 1 1
0 0 1 0;0 1 1 1 0 1 0 0;0 1 1 1 0 1 1 0;0 1 1 1 1 0 0 0;0 1 1 1 1 0 1 0;0 1 1 1 1 1 0 0;0 1 1
1 1 1 1 0; 1 0 0 0 0 0 0 0;1 0 0 0 0 0 1 0;1 0 0 0 0 1 0 0;1 0 0 0 0 1 1 0;1 0 0 0 1 0 0 0;1 0
0 0 1 0 1 0;1 0 0 0 1 1 0 0;1 0 0 0 1 1 1 0;1 0 0 1 0 0 0 0;1 0 0 1 0 0 1 0;1 0 0 1 0 1 0 0;1
0 0 1 0 1 1 0;1 0 0 1 1 0 0 0;1 0 0 1 1 0 1 0;1 0 0 1 1 1 0 0;1 0 0 1 1 1 1 0;1 0 1 0 0 0 0
0;1 0 1 0 0 0 1 0;1 0 1 0 0 1 0 0;1 0 1 0 0 1 1 0;1 0 1 0 1 0 0 0;1 0 1 0 1 0 1 0;1 0 1 0 1 1
0 0;1 0 1 0 1 1 1 0;1 0 1 1 0 0 0 0;1 0 1 1 0 0 1 0;1 0 1 1 0 1 0 0;1 0 1 1 0 1 1 0;1 0 1 1 1
0 0 0;1 0 1 1 1 0 1 0;1 0 1 1 1 1 0 0;1 0 1 1 1 1 1 0;1 1 0 0 0 0 0 0;1 1 0 0 0 0 1 0;1 1 0 0
0 1 0 0;1 1 0 0 0 1 1 0;1 1 0 0 1 0 0 0;1 1 0 0 1 0 1 0;1 1 0 0 1 1 0 0;1 1 0 0 1 1 1 0;1 1 0
1 0 0 0 0;1 1 0 1 0 0 1 0;1 1 0 1 0 1 0 0;1 1 0 1 0 1 1 0;1 1 0 1 1 0 0 0;1 1 0 1 1 0 1 0;1 1
0 1 1 1 0 0;1 1 0 1 1 1 1 0;1 1 1 0 0 0 0 0;1 1 1 0 0 0 1 0;1 1 1 0 0 1 0 0;1 1 1 0 0 1 1 0;1
1 1 0 1 0 0 0;1 1 1 0 1 0 1 0;1 1 1 0 1 1 0 0;1 1 1 0 1 1 1 0;1 1 1 1 0 0 0 0;1 1 1 1 0 0 1
0;1 1 1 1 0 1 0 0;1 1 1 1 0 1 1 0;1 1 1 1 1 0 0 0;1 1 1 1 1 0 1 0;1 1 1 1 1 1 0 0;1 1 1 1 1 1
1 0;0 0 0 0 0 0 0 1;0 0 0 0 0 0 1 1;0 0 0 0 0 1 0 1;0 0 0 0 0 1 1 1;0 0 0 0 1 0 0 1;0 0 0 0 1
0 1 1;0 0 0 0 1 1 0 1;0 0 0 0 1 1 1 1;0 0 0 1 0 0 0 1;0 0 0 1 0 0 1 1;0 0 0 1 0 1 0 1;0 0 0 1
0 1 1 1;0 0 0 1 1 0 0 1;0 0 0 1 1 0 1 1;0 0 0 1 1 1 0 1;0 0 0 1 1 1 1 1;0 0 1 0 0 0 0 1;0 0 1
0 0 0 1 1;0 0 1 0 0 1 0 1;0 0 1 0 0 1 1 1;0 0 1 0 1 0 0 1;0 0 1 0 1 0 1 1;0 0 1 0 1 1 0 1;0 0
```

1 0 1 1 1 1;0 0 1 1 0 0 0 1;0 0 1 1 0 0 1 1;0 0 1 1 0 1 0 1;0 0 1 1 0 1 1 1;0 0 1 1 1 0 0 1;0 0 1 1 1 0 1 1;0 0 1 1 1 1 0 1;0 0 1 1 1 1 1 1;0 1 0 0 0 0 0 1;0 1 0 0 0 0 1 1;0 1 0 0 0 1 0 1;0 1 0 0 0 1 1 1;0 1 0 0 1 0 0 1;0 1 0 0 1 0 1 1;0 1 0 0 1 1 0 1;0 1 0 0 1 1 1 1;0 1 0 1 0 0 0 1;0 1 0 1 0 0 1 1;0 1 0 1 0 1 0 1;0 1 0 1 0 1 1 1;0 1 0 1 1 0 0 1;0 1 0 1 1 0 1 1;0 1 0 1 1 1 0 1;0 1 0 1 1 1 1 1;0 1 1 0 0 0 0 1;0 1 1 0 0 0 1 1;0 1 1 0 0 1 0 1;0 1 1 0 0 1 1 1;0 1 1 0 1 0 0 1;0 1 1 0 1 0 1 1;0 1 1 0 1 1 0 1;0 1 1 0 1 1 1 1;0 1 1 1 0 0 0 1;0 1 1 1 0 0 1 1;0 1 1 1 0 1 0 1;0 1 1 1 0 1 1 1;0 1 1 1 1 0 0 1;0 1 1 1 1 0 1 1;0 1 1 1 1 1 0 1;0 1 1 1 1 1 1 1; 1 0 0 0 0 0 0 1;1 0 0 0 0 0 1 1;1 0 0 0 0 1 0 1;1 0 0 0 0 1 1 1;1 0 0 0 1 0 0 1;1 0 0 0 1 0 1 1;1 0 0 0 1 1 0 1;1 0 0 0 1 1 1 1;1 0 0 1 0 0 0 1;1 0 0 1 0 0 1 1;1 0 0 1 0 1 0 1;1 0 0 1 0 1 1 1;1 0 0 1 1 0 0 1;1 0 0 1 1 0 1 1;1 0 0 1 1 1 0 1;1 0 0 1 1 1 1 1;1 0 1 0 0 0 0 1;1 0 1 0 0 0 1 1;1 0 1 0 0 1 0 1;1 0 1 0 0 1 1 1;1 0 1 0 1 0 0 1;1 0 1 0 1 0 1 1;1 0 1 0 1 1 0 1;1 0 1 0 1 1 1 1;1 0 1 1 0 0 0 1;1 0 1 1 0 0 1 1;1 0 1 1 0 1 0 1;1 0 1 1 0 1 1 1;1 0 1 1 1 0 0 1;1 0 1 1 1 0 1 1;1 0 1 1 1 1 0 1;1 0 1 1 1 1 1 1;1 1 0 0 0 0 0 1;1 1 0 0 0 0 1 1;1 1 0 0 0 1 0 1;1 1 0 0 0 1 1 1;1 1 0 0 1 0 0 1;1 1 0 0 1 0 1 1;1 1 0 0 1 1 0 1;1 1 0 0 1 1 1 1;1 1 0 1 0 0 0 1;1 1 0 1 0 0 1 1;1 1 0 1 0 1 0 1;1 1 0 1 0 1 1 1;1 1 0 1 1 0 0 1;1 1 0 1 1 0 1 1;1 1 0 1 1 1 0 1;1 1 0 1 1 1 1 1;1 1 1 0 0 0 0 1;1 1 1 0 0 0 1 1;1 1 1 0 0 1 0 1;1 1 1 0 0 1 1 1;1 1 1 0 1 0 0 1;1 1 1 0 1 0 1 1;1 1 1 0 1 1 0 1;1 1 1 0 1 1 1 1;1 1 1 1 0 0 0 1;1 1 1 1 0 0 1 1;1 1 1 1 0 1 0 1;1 1 1 1 0 1 1 1;1 1 1 1 1 0 0 1;1 1 1 1 1 0 1 1;1 1 1 1 1 1 0 1;1 1 1 1 1 1 1 1];

training_outx=[0 0 0 0 0 0 1 1;0 0 0 0 0 1 0 1;0 0 0 0 0 1 1 1;0 0 0 0 1 0 0 1;0 0 0 0 1 0 1 1;0 0 0 0 1 1 0 1;0 0 0 0 1 1 1 1;0 0 0 1 0 0 0 1;0 0 0 1 0 0 1 1;0 0 0 1 0 1 0 1;0 0 0 1 0 1 1 1;0 0 0 1 1 0 0 1;0 0 0 1 1 0 1 1;0 0 0 1 1 1 0 1;0 0 0 1 1 1 1 1;0 0 1 0 0 0 0 1;0 0 1 0 0 0 1 1;0 0 1 0 0 1 0 1;0 0 1 0 0 1 1 1;0 0 1 0 1 0 0 1;0 0 1 0 1 0 1 1;0 0 1 0 1 1 0 1;0 0 1 0 1 1 1 1;0 0 1 1 0 0 0 1;0 0 1 1 0 0 1 1;0 0 1 1 0 1 0 1;0 0 1 1 0 1 1 1;0 0 1 1 1 0 0 1;0 0 1 1 1 0 1 1;0 0 1 1 1 1 0 1;0 0 1 1 1 1 1 1;0 1 0 0 0 0 0 1;0 1 0 0 0 0 1 1;0 1 0 0 0 1 0 1;0 1 0 0 0 1 1 1;0 1 0 0 1 0 0 1;0 1 0 0 1 0 1 1;0 1 0 0 1 1 0 1;0 1 0 0 1 1 1 1;0 1 0 1 0 0 0 1;0 1 0 1 0 0 1 1;0 1 0 1 0 1 0 1;0 1 0 1 0 1 1 1;0 1 0 1 1 0 0 1;0 1 0 1 1 0 1 1;0 1 0 1 1 1 0 1;0 1 0 1 1 1 1 1;0 1 1 0 0 0 0 1;0 1 1 0 0 0 1 1;0 1 1 0 0 1 0 1;0 1 1 0 0 1 1 1;0 1 1 0 1 0 0 1;0 1 1 0 1 0 1 1;0 1 1 0 1 1 0 1;0 1 1 0 1 1 1 1;0 1 1 1 0 0 0 1;0 1 1 1 0 0 1 1;0 1 1 1 0 1 0 1;0 1 1 1 0 1 1 1;0 1 1 1 1 0 0 1;0 1 1 1 1 0 1 1;0 1 1 1 1 1 0 1;0 1 1 1 1 1 1 1; 1 0 0 0 0 0 0 1;1 0 0 0 0 0 1 1;1 0 0 0 0 1 0 1;1 0 0 0 0 1 1 1;1 0 0 0 1 0 0 1;1 0 0 0 1 0 1 1;1 0 0 0 1 1 0 1;1 0 0 0 1 1 1 1;1 0 0 1 0 0 0 1;1 0 0 1 0 0 1 1;1 0 0 1 0 1 0 1;1 0 0 1 0 1 1 1;1 0 0 1 1 0 0 1;1 0 0 1 1 0 1 1;1 0 0 1 1 1 0 1;1 0 0 1 1 1 1 1;1 0 1 0 0 0 0 1;1 0 1 0 0 0 1 1;1 0 1 0 0 1 0 1;1 0 1 0 0 1 1 1;1 0 1 0 1 0 0 1;1 0 1 0 1 0 1 1;1 0 1 0 1 1 0 1;1 0 1 0 1 1 1 1;1 0 1 1 0 0 0 1;1 0 1 1 0 0 1 1;1 0 1 1 0 1 0 1;1 0 1 1 0 1 1 1;1 0 1 1 1 0 0 1;1 0 1 1 1 0 1 1;1 0 1 1 1 1 0 1;1 0 1 1 1 1 1 1;1 1 0 0 0 0 0 1;1 1 0 0 0 0 1 1;1 1 0 0 0 1 0 1;1 1 0 0 0 1 1 1;1 1 0 0 1 0 0 1;1 1 0 0 1 0 1 1;1 1 0 0 1 1 0 1;1 1 0 0 1 1 1 1;1 1 0 1 0 0 0 1;1 1 0 1 0 0 1 1;1 1 0 1 0 1 0 1;1 1 0 1 0 1 1 1;1 1 0 1 1 0 0 1;1 1 0 1 1 0 1 1;1 1 0 1 1 1 0 1;1 1 0 1 1 1 1 1;1 1 1 0 0 0 0 1;1 1 1 0 0 0 1 1;1 1 1 0 0 1 0 1;1 1 1 0 0 1 1 1;1 1 1 0 1 0 0 1;1 1 1 0 1 0 1 1;1 1 1 0 1 1 0 1;1 1 1 0 1 1 1 1;1 1 1 1 0 0 0 1;1 1 1 1 0 0 1 1;1 1 1 1 0 1 0 1;1 1 1 1 0 1 1 1;1 1 1 1 1 0 0 1;1 1 1 1 1 0 1 1;1 1 1 1 1 1 0 1;1 1 1 1 1 1 1 1];

```
0 0;0 1 1 0 0 0 1 0;0 1 1 0 0 1 0 0;0 1 1 0 0 1 1 0;0 1 1 0 1 0 0 0;0 1 1 0 1 0 1 0;0 1 1 0 1
1 0 0;0 1 1 0 1 1 1 0;0 1 1 1 0 0 0 0;0 1 1 1 0 0 1 0;0 1 1 1 0 1 0 0;0 1 1 1 0 1 1 0;0 1 1 1
1 0 0 0;0 1 1 1 1 0 1 0;0 1 1 1 1 1 0 0;0 1 1 1 1 1 1 0; 1 0 0 0 0 0 0 0;1 0 0 0 0 0 1 0;1 0 0
0 0 1 0 0;1 0 0 0 0 1 1 0;1 0 0 0 1 0 0 0;1 0 0 0 1 0 1 0;1 0 0 0 1 1 0 0;1 0 0 0 1 1 1 0;1 0
0 1 0 0 0 0;1 0 0 1 0 0 1 0;1 0 0 1 0 1 0 0;1 0 0 1 0 1 1 0;1 0 0 1 1 0 0 0;1 0 0 1 1 0 1 0;1
0 0 1 1 1 0 0;1 0 0 1 1 1 1 0;1 0 1 0 0 0 0 0;1 0 1 0 0 0 1 0;1 0 1 0 0 1 0 0;1 0 1 0 0 1 1
0;1 0 1 0 1 0 0 0;1 0 1 0 1 0 1 0;1 0 1 0 1 1 0 0;1 0 1 0 1 1 1 0;1 0 1 1 0 0 0 0;1 0 1 1 0 0
1 0;1 0 1 1 0 1 0 0;1 0 1 1 0 1 1 0;1 0 1 1 1 0 0 0;1 0 1 1 1 0 1 0;1 0 1 1 1 1 0 0;1 0 1 1 1
1 1 0;1 1 0 0 0 0 0 0;1 1 0 0 0 0 1 0;1 1 0 0 0 1 0 0;1 1 0 0 0 1 1 0;1 1 0 0 1 0 0 0;1 1 0 0
1 0 1 0;1 1 0 0 1 1 0 0;1 1 0 0 1 1 1 0;1 1 0 1 0 0 0 0;1 1 0 1 0 0 1 0;1 1 0 1 0 1 0 0;1 1 0
1 0 1 1 0;1 1 0 1 1 0 0 0;1 1 0 1 1 0 1 0;1 1 0 1 1 1 0 0;1 1 0 1 1 1 1 0;1 1 1 0 0 0 0 0;1 1
1 0 0 0 1 0;1 1 1 0 0 1 0 0;1 1 1 0 0 1 1 0;1 1 1 0 1 0 0 0;1 1 1 0 1 0 1 0;1 1 1 0 1 1 0 0;1
1 1 0 1 1 1 0;1 1 1 1 0 0 0 0;1 1 1 1 0 0 1 0;1 1 1 1 0 1 0 0;1 1 1 1 0 1 1 0;1 1 1 1 1 0 0
0;1 1 1 1 1 0 1 0;1 1 1 1 1 1 0 0;1 1 1 1 1 1 1 0;0 0 0 0 0 0 0 0;0 0 0 0 0 0 1 0];
for x=1:100000
training_set=training_setx(a,:);       %assigning matrix
training_out=training_outx(a,:);       %assigning matrix
inputu=[1 training_set];                %generating a new matrix
sum_h=(w1*(inputu)')';                  %multiplying the randomly generated weights
with transpose of above generated matrix and
out_h=1./(1+exp(-sum_h));               %passing the above obtained values through a
Sigmoid Activation Function finally transposing it
% --------------------------- output layer
input_h=[1 out_h];                      %generating a matrix using the above obtained
values
sum_out=(w2*(input_h)')';               %multiplying the above obtained values transpose
matrix with weights and performing transpose again
out=1./(1+exp(-sum_out));               %applying Sigmoid activation Function to above
obtained values
% --------------------------- delta
delta_out=neta*(out.*(1-out)).*(training_out - out);%applying Delta rule at the output
layer considering the output defined in the training and the output which is generated
delta_h=(delta_out*w2).*input_h.*(1-input_h);   %applying delta rule considering the
input to output layer
% update of weight -------------------------
% output layer --------------------------------
for t=1:(outp+states)                                   %for loop
w2(t,:) = w2(t,:) + neta*delta_out(t)*input_h;      %output layer weights updation
end
% hidden layer ---------------------------------
for t=1:hl                                              %for loop
w1(t,:) = w1(t,:) + neta*delta_h(t+1)*inputu;       %hidden layer weights updation
end
for t=1:(outp+states)                                   %for loop iterating for 8 times
if out(t)>=0.7                                  %condition if the output is greater than 0.7 then
output is 1
out1(t)=1;
```

```
elseif out(t)<=0.2                    %else condition the output is less than 0.2 then
output is 0
out1(t)=0;
else out1(t)=out(t);                  %else output is same
end
end
m=[m sum(out-training_out)];          %generating a matrix for mean square error
if out1==training_out                 %if generated output is equals to trained one
a=a+1;                                %increment for the next input training data
sx=sx+1;                              %increment iteration number
end
if a>((2^inpu)*key)                   %row number is greater than 256
a=a-((2^inpu)*key);                   %decrement a by the function
end
end
plot(m.*m);                           %plot the mean square error%
```

5.2 Encryption Testing:

```
%testing the program
tx = datetime('now');                 %reading time and date
day=tx.Day;                           %reading day
sec=floor(tx.Second);                 %reading and rounding second value
min=tx.Minute;                        %reading minute value
sa=0;                                 %initializing some value
sb=0;
sc=0;
while(day~=0)                         %while loop for finding the sum of day
   r1=floor(mod(day,10));
   sa=sa+r1;
   day=floor(day/10);
end
while(sec~=0)                         %while loop for finding sum of second
   r2=floor((mod(sec,10)));
   sb=sb+r2;
   sec=floor(sec/10);
end
while(min~=0)                         %while loop for finding sum of minute
   r3=floor(mod(min,10));
   sc=sc+r3;
   min=floor(min/10);
end
da=sa+sb+sc;                          %adding all the sums
h=da;                                 %round means number of times message should be
encrypted
if(mod(da,2)==0)                      %condition for key
```

```
   statx=0;
else
   statx=1;
end
st=statx;                          %assigning key to some variable
ipo = input('enter the text:','s');  %enter the message
z=0;                               %assigning a variable to zero
while(z~=h)                        %iterating the loop till the round number is reached
finp=[];                           %initializing a matrix
for i=1:length(ipo)                %iterating the loop till the length of message is reached
b=ipo(i);                          %taking the message charcters according to the value of
i
switch b                           %using the switch condition to get the binary value of
the character
case('C')
set=[0 0 0 0 0 0 0];
case('f')
set=[0 0 0 0 0 0 1];
case('4')
set=[0 0 0 0 0 1 0];
case('@')
set=[0 0 0 0 0 1 1];
case(' ')
set=[0 0 0 0 1 0 0];
case('A')
set=[0 0 0 0 1 0 1];
case('h')
set=[0 0 0 0 1 1 0];
case('6')
set=[0 0 0 0 1 1 1];
case('#')
set=[0 0 0 1 0 0 0];
case('B')
set=[0 0 0 1 0 0 1];
case('b')
set=[0 0 0 1 0 1 0];
case('3')
set=[0 0 0 1 0 1 1];
case('(')
set=[0 0 0 1 1 0 0];
case('M')
set=[0 0 0 1 1 0 1];
case('n')
set=[0 0 0 1 1 1 0];
case('9')
set=[0 0 0 1 1 1 1];
case('D')
set=[0 0 1 0 0 0 0];
```

```
case('j')
set=[0 0 1 0 0 0 1];
case('p')
set=[0 0 1 0 0 1 0];
case('8')
set=[0 0 1 0 0 1 1];
case('$')
set=[0 0 1 0 1 0 0];
case('P')
set=[0 0 1 0 1 0 1];
case('r')
set=[0 0 1 0 1 1 0];
case('+')
set=[0 0 1 0 1 1 1];
case('7')
set=[0 0 1 1 0 0 0];
case('E')
set=[0 0 1 1 0 0 1];
case('d')
set=[0 0 1 1 0 1 0];
case('0')
set=[0 0 1 1 0 1 1];
case('!')
set=[0 0 1 1 1 0 0];
case('Z')
set=[0 0 1 1 1 0 1];
case('s')
set=[0 0 1 1 1 1 0];
case('<')
set=[0 0 1 1 1 1 1];
case('F')
set=[0 1 0 0 0 0 0];
case('a')
set=[0 1 0 0 0 0 1];
case('5')
set=[0 1 0 0 0 1 0];
case('%')
set=[0 1 0 0 0 1 1];
case('G')
set=[0 1 0 0 1 0 0];
case('c')
set=[0 1 0 0 1 0 1];
case('l')
set=[0 1 0 0 1 1 0];
case('=')
set=[0 1 0 0 1 1 1];
case('I')
set=[0 1 0 1 0 0 0];
```

```
case('g')
set=[0 1 0 1 0 0 1];
case('2')
set=[0 1 0 1 0 1 0];
case('|')
set=[0 1 0 1 0 1 1];
case('H')
set=[0 1 0 1 1 0 0];
case('t')
set=[0 1 0 1 1 0 1];
case('\')
set=[0 1 0 1 1 1 0];
case('N')
set=[0 1 0 1 1 1 1];
case('k')
set=[0 1 1 0 0 0 0];
case('{')
set=[0 1 1 0 0 0 1];
case('K')
set=[0 1 1 0 0 1 0];
case('u')
set=[0 1 1 0 0 1 1];
case('}')
set=[0 1 1 0 1 0 0];
case('e')
set=[0 1 1 0 1 0 1];
case('J')
set=[0 1 1 0 1 1 0];
case('[')
set=[0 1 1 0 1 1 1];
case('L')
set=[0 1 1 1 0 0 0];
case('v')
set=[0 1 1 1 0 0 1];
case(']')
set=[0 1 1 1 0 1 0];
case('O')
set=[0 1 1 1 0 1 1];
case('q')
set=[0 1 1 1 1 0 0];
case('?')
set=[0 1 1 1 1 0 1];
case('S')
set=[0 1 1 1 1 1 0];
case('z')
set=[0 1 1 1 1 1 1];
case(':')
set=[1 0 0 0 0 0 0];
```

```
case('Q')
set=[1 0 0 0 0 0 1];
case('i')
set=[1 0 0 0 0 1 0];
case(';')
set=[1 0 0 0 0 1 1];
case('R')
set=[1 0 0 0 1 0 0];
case('1')
set=[1 0 0 0 1 0 1];
case('m')
set=[1 0 0 0 1 1 0];
case('T')
set=[1 0 0 0 1 1 1];
case('y')
set=[1 0 0 1 0 0 0];
case('/')
set=[1 0 0 1 0 0 1];
case('"')
set=[1 0 0 1 0 1 0];
case('U')
set=[1 0 0 1 0 1 1];
case('o')
set=[1 0 0 1 1 0 0];
case('&')
set=[1 0 0 1 1 0 1];
case('V')
set=[1 0 0 1 1 1 0];
case('x')
set=[1 0 0 1 1 1 1];
case('*')
set=[1 0 1 0 0 0 0];
case('W')
set=[1 0 1 0 0 0 1];
case('.')
set=[1 0 1 0 0 1 0];
case('w')
set=[1 0 1 0 0 1 1];
case(',')
set=[1 0 1 0 1 0 0];
case('ß')
set=[1 0 1 0 1 0 1];
case('X')
set=[1 0 1 0 1 1 0];
case('^')
set=[1 0 1 0 1 1 1];
case('Y')
set=[1 0 1 1 0 0 0];
```

```
case('-')
set=[1 0 1 1 0 0 1];
case('_')
set=[1 0 1 1 0 1 0];
case(')')
set=[1 0 1 1 0 1 1];
case('~')
set=[1 0 1 1 1 0 0];
case('`')
set=[1 0 1 1 1 0 1];
case('>')
set=[1 0 1 1 1 1 0];
case('ô')
set=[1 0 1 1 1 1 1];
case('÷')
set=[1 1 0 0 0 0 0];
case('è')
set=[1 1 0 0 0 0 1];
case('Ø')
set=[1 1 0 0 0 1 0];
case('ï')
set=[1 1 0 0 0 1 1];
case('Ý')
set=[1 1 0 0 1 0 0];
case('Í')
set=[1 1 0 0 1 0 1];
case('ù')
set=[1 1 0 0 1 1 0];
case('1')
set=[1 1 0 0 1 1 1];
case('«')
set=[1 1 0 1 0 0 0];
case('»')
set=[1 1 0 1 0 0 1];
case('ç')
set=[1 1 0 1 0 1 0];
case('ã')
set=[1 1 0 1 0 1 1];
case('å')
set=[1 1 0 1 1 0 0];
case('°')
set=[1 1 0 1 1 0 1];
case('½')
set=[1 1 0 1 1 1 0];
case('Ñ')
set=[1 1 0 1 1 1 1];
case('Ä')
set=[1 1 1 0 0 0 0];
```

```
case('µ')
set=[1 1 1 0 0 0 1];
case('¾')
set=[1 1 1 0 0 1 0];
case('©')
set=[1 1 1 0 0 1 1];
case('ð')
set=[1 1 1 0 1 0 0];
case('Ô')
set=[1 1 1 0 1 0 1];
case('á')
set=[1 1 1 0 1 1 0];
case('õ')
set=[1 1 1 0 1 1 1];
case('ó')
set=[1 1 1 1 0 0 0];
case('ê')
set=[1 1 1 1 0 0 1];
case('Ê')
set=[1 1 1 1 0 1 0];
case('ý')
set=[1 1 1 1 0 1 1];
case('£')
set=[1 1 1 1 1 0 0];
case('ñ')
set=[1 1 1 1 1 0 1];
case('ë')
set=[1 1 1 1 1 1 0];
case('Æ')
set=[1 1 1 1 1 1 1];
end
ipox=[set statx];                    %generating a matrix using the generated binary value
and the key
inputp=[1 ipox];                     %generating another matrix using above matrix
sum_h=(w1*(inputp)')';                  %multiplying the weights with transpose of above
matrix and again transpose it
o_h=1./(1+exp(-sum_h));                 %passing the above matrix values into a sigmoid
activation function
%output layer ----------------------------
input_h=[1 o_h];                     %generating another matrix using the above  obtained
values
sum_out=(w2*(input_h)')';               %multiplying with the transpose of above matrix
and find transpose again
out=1./(1+exp(-sum_out));               %passing the above obtained values into a sigmoid
for t=1:(states+outp)                %iterating the loop for 8 times
if out(t)>=0.8                       %if output is greater than or equal to 0.8 then output is 1
out1(t)=1;
elseif out(t)<=0.4                   %output is less than or equal to 0.4 then output is 0
```

```
out1(t)=0;
else out1(t)=out(t);
end
end
finp=[finp;out1];                    %generating a matrix with the outputs for the given
input binary bits using the training
tempch=[];                           %declaring an empty matrix
statx=out1( (outp + 1):(outp+states));  %defining the out1(8:8) last bit indicates the key
end
outzs=[''];                          %declaring a matrix
for f=1:length(ipo)                  %iterating the loop length(message) times
tempch=finp(f,:);                    %using the every row of above obtained matrix
tempch=tempch(1:7);                  %taking only 7 bits into consideration as the last bit
is the key
%here the generated output bits for the given input bits is taken into
%consideration and respective character to the output bit is printed
if tempch==[0 0 0 0 0 0 0]
outzs=[outzs 'C'];
end
if tempch==[0 0 0 0 0 0 1]
outzs=[outzs 'f'];
end
if tempch==[0 0 0 0 0 1 0]
outzs=[outzs '4'];
end
if tempch==[0 0 0 0 0 1 1]
outzs=[outzs '@'];
end
if tempch==[0 0 0 0 1 0 0]
outzs=[outzs ' '];
end
if tempch==[0 0 0 0 1 0 1]
outzs=[outzs 'A'];
end
if tempch==[0 0 0 0 1 1 0]
outzs=[outzs 'h'];
end
if tempch==[0 0 0 0 1 1 1]
outzs=[outzs '6'];
end
if tempch==[0 0 0 1 0 0 0]
outzs=[outzs '#'];
end
if tempch==[0 0 0 1 0 0 1]
outzs=[outzs 'B'];
end
if tempch==[0 0 0 1 0 1 0]
outzs=[outzs 'b'];
```

```
end
if tempch==[0 0 0 1 0 1 1]
outzs=[outzs '3'];
end
if tempch==[0 0 0 1 1 0 0]
outzs=[outzs '('];
end
if tempch==[0 0 0 1 1 0 1]
outzs=[outzs 'M'];
end
if tempch==[0 0 0 1 1 1 0]
outzs=[outzs 'n'];
end
if tempch==[0 0 0 1 1 1 1]
outzs=[outzs '9'];
end
if tempch==[0 0 1 0 0 0 0]
outzs=[outzs 'D'];
end
if tempch==[0 0 1 0 0 0 1]
outzs=[outzs 'j'];
end
if tempch==[0 0 1 0 0 1 0]
outzs=[outzs 'p'];
end
if tempch==[0 0 1 0 0 1 1]
outzs=[outzs '8'];
end
if tempch==[0 0 1 0 1 0 0]
outzs=[outzs '$'];
end
if tempch==[0 0 1 0 1 0 1]
outzs=[outzs 'P'];
end
if tempch==[0 0 1 0 1 1 0]
outzs=[outzs 'r'];
end
if tempch==[0 0 1 0 1 1 1]
outzs=[outzs '+'];
end
if tempch==[0 0 1 1 0 0 0]
outzs=[outzs '7'];
end
if tempch==[0 0 1 1 0 0 1]
outzs=[outzs 'E'];
end
if tempch==[0 0 1 1 0 1 0]
outzs=[outzs 'd'];
```

```
end
if tempch==[0 0 1 1 0 1 1]
outzs=[outzs '0'];
end
if tempch==[0 0 1 1 1 0 0]
outzs=[outzs '!'];
end
if tempch==[0 0 1 1 1 0 1]
outzs=[outzs 'Z'];
end
if tempch==[0 0 1 1 1 1 0]
outzs=[outzs 's'];
end
if tempch==[0 0 1 1 1 1 1]
outzs=[outzs '<'];
end
if tempch==[0 1 0 0 0 0 0]
outzs=[outzs 'F'];
end
if tempch==[0 1 0 0 0 0 1]
outzs=[outzs 'a'];
end
if tempch==[0 1 0 0 0 1 0]
outzs=[outzs '5'];
end
if tempch==[0 1 0 0 0 1 1]
outzs=[outzs '%'];
end
if tempch==[0 1 0 0 1 0 0]
outzs=[outzs 'G'];
end
if tempch==[0 1 0 0 1 0 1]
outzs=[outzs 'c'];
end
if tempch==[0 1 0 0 1 1 0]
outzs=[outzs 'l'];
end
if tempch==[0 1 0 0 1 1 1]
outzs=[outzs '='];
end
if tempch==[0 1 0 1 0 0 0]
outzs=[outzs 'I'];
end
if tempch==[0 1 0 1 0 0 1]
outzs=[outzs 'g'];
end
if tempch==[0 1 0 1 0 1 0]
outzs=[outzs '2'];
```

```
end
if tempch==[0 1 0 1 0 1 1]
outzs=[outzs '|'];
end
if tempch==[0 1 0 1 1 0 0]
outzs=[outzs 'H'];
end
if tempch==[0 1 0 1 1 0 1]
outzs=[outzs 't'];
end
if tempch==[0 1 0 1 1 1 0]
outzs=[outzs '\'];
end
if tempch==[0 1 0 1 1 1 1]
outzs=[outzs 'N'];
end
if tempch==[0 1 1 0 0 0 0]
outzs=[outzs 'k'];
end
if tempch==[0 1 1 0 0 0 1]
outzs=[outzs '{'];
end
if tempch==[0 1 1 0 0 1 0]
outzs=[outzs 'K'];
end
if tempch==[0 1 1 0 0 1 1]
outzs=[outzs 'u'];
end
if tempch==[0 1 1 0 1 0 0]
outzs=[outzs '}'];
end
if tempch==[0 1 1 0 1 0 1]
outzs=[outzs 'e'];
end
if tempch==[0 1 1 0 1 1 0]
outzs=[outzs 'J'];
end
if tempch==[0 1 1 0 1 1 1]
outzs=[outzs '['];
end
if tempch==[0 1 1 1 0 0 0]
outzs=[outzs 'L'];
end
if tempch==[0 1 1 1 0 0 1]
outzs=[outzs 'v'];
end
if tempch==[0 1 1 1 0 1 0]
outzs=[outzs ']'];
```

```
end
if tempch==[0 1 1 1 0 1 1]
outzs=[outzs 'O'];
end
if tempch==[0 1 1 1 1 0 0]
outzs=[outzs 'q'];
end
if tempch==[0 1 1 1 1 0 1]
outzs=[outzs '?'];
end
if tempch==[0 1 1 1 1 1 0]
outzs=[outzs 'S'];
end
if tempch==[0 1 1 1 1 1 1]
outzs=[outzs 'z'];
end
if tempch==[1 0 0 0 0 0 0]
outzs=[outzs ':'];
end
if tempch==[1 0 0 0 0 0 1]
outzs=[outzs 'Q'];
end
if tempch==[1 0 0 0 0 1 0]
outzs=[outzs 'i'];
end
if tempch==[1 0 0 0 0 1 1]
outzs=[outzs ';'];
end
if tempch==[1 0 0 0 1 0 0]
outzs=[outzs 'R'];
end
if tempch==[1 0 0 0 1 0 1]
outzs=[outzs '1'];
end
if tempch==[1 0 0 0 1 1 0]
outzs=[outzs 'm'];
end
if tempch==[1 0 0 0 1 1 1]
outzs=[outzs 'T'];
end
if tempch==[1 0 0 1 0 0 0]
outzs=[outzs 'y'];
end
if tempch==[1 0 0 1 0 0 1]
outzs=[outzs '/'];
end
if tempch==[1 0 0 1 0 1 0]
outzs=[outzs ''''];
```

```
end
if tempch==[1 0 0 1 0 1 1]
outzs=[outzs 'U'];
end
if tempch==[1 0 0 1 1 0 0]
outzs=[outzs 'o'];
end
if tempch==[1 0 0 1 1 0 1]
outzs=[outzs '&'];
end
if tempch==[1 0 0 1 1 1 0]
outzs=[outzs 'V'];
end
if tempch==[1 0 0 1 1 1 1]
outzs=[outzs 'x'];
end
if tempch==[1 0 1 0 0 0 0]
outzs=[outzs '*'];
end
if tempch==[1 0 1 0 0 0 1]
outzs=[outzs 'W'];
end
if tempch==[1 0 1 0 0 1 0]
outzs=[outzs '.'];
end
if tempch==[1 0 1 0 0 1 1]
outzs=[outzs 'w'];
end
if tempch==[1 0 1 0 1 0 0]
outzs=[outzs ','];
end
if tempch==[1 0 1 0 1 0 1]
outzs=[outzs 'ß'];
end
if tempch==[1 0 1 0 1 1 0]
outzs=[outzs 'X'];
end
if tempch==[1 0 1 0 1 1 1]
outzs=[outzs '^'];
end
if tempch==[1 0 1 1 0 0 0]
outzs=[outzs 'Y'];
end
if tempch==[1 0 1 1 0 0 1]
outzs=[outzs '-'];
end
if tempch==[1 0 1 1 0 1 0]
outzs=[outzs '_'];
```

```
end
if tempch==[1 0 1 1 0 1 1]
outzs=[outzs ')'];
end
if tempch==[1 0 1 1 1 0 0]
outzs=[outzs '~'];
end
if tempch==[1 0 1 1 1 0 1]
outzs=[outzs '`'];
end
if tempch==[1 0 1 1 1 1 0]
outzs=[outzs '>'];
end
if tempch==[1 0 1 1 1 1 1]
outzs=[outzs 'ô'];
end
if tempch==[1 1 0 0 0 0 0]
outzs=[outzs '÷'];
end
if tempch==[1 1 0 0 0 0 1]
outzs=[outzs 'è'];
end
if tempch==[1 1 0 0 0 1 0]
outzs=[outzs 'Ø'];
end
if tempch==[1 1 0 0 0 1 1]
outzs=[outzs 'ï'];
end
if tempch==[1 1 0 0 1 0 0]
outzs=[outzs 'Ý'];
end
if tempch==[1 1 0 0 1 0 1]
outzs=[outzs 'Í'];
end
if tempch==[1 1 0 0 1 1 0]
outzs=[outzs 'ù'];
end
if tempch==[1 1 0 0 1 1 1]
outzs=[outzs '¹'];
end
if tempch==[1 1 0 1 0 0 0]
outzs=[outzs '«'];
end
if tempch==[1 1 0 1 0 0 1]
outzs=[outzs '»'];
end
if tempch==[1 1 0 1 0 1 0]
outzs=[outzs 'ç'];
```

```
end
if tempch==[1 1 0 1 0 1 1]
outzs=[outzs 'ã'];
end
if tempch==[1 1 0 1 1 0 0]
outzs=[outzs 'å'];
end
if tempch==[1 1 0 1 1 0 1]
outzs=[outzs '°'];
end
if tempch==[1 1 0 1 1 1 0]
outzs=[outzs '½'];
end
if tempch==[1 1 0 1 1 1 1]
outzs=[outzs 'Ñ'];
end
if tempch==[1 1 1 0 0 0 0]
outzs=[outzs 'Ä'];
end
if tempch==[1 1 1 0 0 0 1]
outzs=[outzs 'µ'];
end
if tempch==[1 1 1 0 0 1 0]
outzs=[outzs '¾'];
end
if tempch==[1 1 1 0 0 1 1]
outzs=[outzs '©'];
end
if tempch==[1 1 1 0 1 0 0]
outzs=[outzs 'ð'];
end
if tempch==[1 1 1 0 1 0 1]
outzs=[outzs 'Ô'];
end
if tempch==[1 1 1 0 1 1 0]
outzs=[outzs 'á'];
end
if tempch==[1 1 1 0 1 1 1]
outzs=[outzs 'õ'];
end
if tempch==[1 1 1 1 0 0 0]
outzs=[outzs 'ó'];
end
if tempch==[1 1 1 1 0 0 1]
outzs=[outzs 'ê'];
end
if tempch==[1 1 1 1 0 1 0]
outzs=[outzs 'Ê'];
```

```
end
if tempch==[1 1 1 1 0 1 1]
outzs=[outzs 'ý'];
end
if tempch==[1 1 1 1 1 0 0]
outzs=[outzs '£'];
end
if tempch==[1 1 1 1 1 0 1]
outzs=[outzs 'ñ'];
end
if tempch==[1 1 1 1 1 1 0]
outzs=[outzs 'ë'];
end
if tempch==[1 1 1 1 1 1 1]
outzs=[outzs 'Æ'];
end
end
ipo=outzs                          %assigning the output generated to input as
mentioned above round encryption
z=z+1;                             %incrementing the value of z
end
outzs
out3=outzs;                        %assigning the output to another variable
t=tcpip('0.0.0.0',30000,'NetworkRole','Server','TimeOut',2);%creating a tcpip object for
communicating with another matlab
fopen(t);                          %open the tcpip connection
fprintf(t,'%s\n',out3);            %giving the output generated at the sender as
an input so that it can be transmitted to receiver
fwrite(t,h);                       %sending the number of rounds
fwrite(t,st);                      %sending the key
```

5.3 Decryption Training:

```
clc;
clear all;
close all;
inpu=7;                            %no. of bits of data
outp=7;                            %no of output bits
key=2;                             %no. of keys
states=1;                          %key length
hl=24;                             % size of the hidden layer
w1=rand(hl,(inpu+states+1));       %generating random weights
w2=rand((outp+states),hl+1);       %generating random weights
neta=1;                            %learning rate
a=1;                               %indicate the row
sx=0;                              %iterations
m=[];                              %generating a matrix to display mean square error
training_setx=[];                  %initialization of training matrix
```

training_setx=[0 0 0 0 0 0 0 0;0 0 0 0 0 0 1 0;0 0 0 0 0 1 0 0;0 0 0 0 0 1 1 0;0 0 0 0 1 0 0 0;0 0 0 0 1 0 1 0;0 0 0 0 1 1 0 0;0 0 0 0 1 1 1 0;0 0 0 1 0 0 0 0;0 0 0 1 0 0 1 0;0 0 0 1 0 1 0 0;0 0 0 1 0 1 1 0;0 0 0 1 1 0 0 0;0 0 0 1 1 0 1 0;0 0 0 1 1 1 0 0;0 0 0 1 1 1 1 0;0 0 1 0 0 0 0 0;0 0 1 0 0 0 1 0;0 0 1 0 0 1 0 0;0 0 1 0 0 1 1 0;0 0 1 0 1 0 0 0;0 0 1 0 1 0 1 0;0 0 1 0 1 1 0 0;0 0 1 0 1 1 1 0;0 0 1 1 0 0 0 0;0 0 1 1 0 0 1 0;0 0 1 1 0 1 0 0;0 0 1 1 0 1 1 0;0 0 1 1 1 0 0 0;0 0 1 1 1 0 1 0;0 0 1 1 1 1 0 0;0 0 1 1 1 1 1 0;0 1 0 0 0 0 0 0;0 1 0 0 0 0 1 0;0 1 0 0 0 1 0 0;0 1 0 0 0 1 1 0;0 1 0 0 1 0 0 0;0 1 0 0 1 0 1 0;0 1 0 0 1 1 0 0;0 1 0 0 1 1 1 0;0 1 0 1 0 0 0 0;0 1 0 1 0 0 1 0;0 1 0 1 0 1 0 0;0 1 0 1 0 1 1 0;0 1 0 1 1 0 0 0;0 1 0 1 1 0 1 0;0 1 0 1 1 1 0 0;0 1 0 1 1 1 1 0;0 1 1 0 0 0 0 0;0 1 1 0 0 0 1 0;0 1 1 0 0 1 0 0;0 1 1 0 0 1 1 0;0 1 1 0 1 0 0 0;0 1 1 0 1 0 1 0;0 1 1 0 1 1 0 0;0 1 1 0 1 1 1 0;0 1 1 1 0 0 0 0;0 1 1 1 0 0 1 0;0 1 1 1 0 1 0 0;0 1 1 1 0 1 1 0;0 1 1 1 1 0 0 0;0 1 1 1 1 0 1 0;0 1 1 1 1 1 0 0;0 1 1 1 1 1 1 0; 1 0 0 0 0 0 0 0;1 0 0 0 0 0 1 0;1 0 0 0 0 1 0 0;1 0 0 0 0 1 1 0;1 0 0 0 1 0 0 0;1 0 0 0 1 0 1 0;1 0 0 0 1 1 0 0;1 0 0 0 1 1 1 0;1 0 0 1 0 0 0 0;1 0 0 1 0 0 1 0;1 0 0 1 0 1 0 0;1 0 0 1 0 1 1 0;1 0 0 1 1 0 0 0;1 0 0 1 1 0 1 0;1 0 0 1 1 1 0 0;1 0 0 1 1 1 1 0;1 0 1 0 0 0 0 0;1 0 1 0 0 0 1 0;1 0 1 0 0 1 0 0;1 0 1 0 0 1 1 0;1 0 1 0 1 0 0 0;1 0 1 0 1 0 1 0;1 0 1 0 1 1 0 0;1 0 1 0 1 1 1 0;1 0 1 1 0 0 0 0;1 0 1 1 0 0 1 0;1 0 1 1 0 1 0 0;1 0 1 1 0 1 1 0;1 0 1 1 1 0 0 0;1 0 1 1 1 0 1 0;1 0 1 1 1 1 0 0;1 0 1 1 1 1 1 0;1 1 0 0 0 0 0 0;1 1 0 0 0 0 1 0;1 1 0 0 0 1 0 0;1 1 0 0 0 1 1 0;1 1 0 0 1 0 0 0;1 1 0 0 1 0 1 0;1 1 0 0 1 1 0 0;1 1 0 0 1 1 1 0;1 1 0 1 0 0 0 0;1 1 0 1 0 0 1 0;1 1 0 1 0 1 0 0;1 1 0 1 0 1 1 0;1 1 0 1 1 0 0 0;1 1 0 1 1 0 1 0;1 1 0 1 1 1 0 0;1 1 0 1 1 1 1 0;1 1 1 0 0 0 0 0;1 1 1 0 0 0 1 0;1 1 1 0 0 1 0 0;1 1 1 0 0 1 1 0;1 1 1 0 1 0 0 0;1 1 1 0 1 0 1 0;1 1 1 0 1 1 0 0;1 1 1 0 1 1 1 0;1 1 1 1 0 0 0 0;1 1 1 1 0 0 1 0;1 1 1 1 0 1 0 0;1 1 1 1 0 1 1 0;1 1 1 1 1 0 0 0;1 1 1 1 1 0 1 0;1 1 1 1 1 1 0 0;1 1 1 1 1 1 1 0;0 0 0 0 0 0 0 1;0 0 0 0 0 0 1 1;0 0 0 0 0 1 0 1;0 0 0 0 0 1 1 1;0 0 0 0 1 0 0 1;0 0 0 0 1 0 1 1;0 0 0 0 1 1 0 1;0 0 0 0 1 1 1 1;0 0 0 1 0 0 0 1;0 0 0 1 0 0 1 1;0 0 0 1 0 1 0 1;0 0 0 1 0 1 1 1;0 0 0 1 1 0 0 1;0 0 0 1 1 0 1 1;0 0 0 1 1 1 0 1;0 0 0 1 1 1 1 1;0 0 1 0 0 0 0 1;0 0 1 0 0 0 1 1;0 0 1 0 0 1 0 1;0 0 1 0 0 1 1 1;0 0 1 0 1 0 0 1;0 0 1 0 1 0 1 1;0 0 1 0 1 1 0 1;0 0 1 0 1 1 1 1;0 0 1 1 0 0 0 1;0 0 1 1 0 0 1 1;0 0 1 1 0 1 0 1;0 0 1 1 0 1 1 1;0 0 1 1 1 0 0 1;0 0 1 1 1 0 1 1;0 0 1 1 1 1 0 1;0 0 1 1 1 1 1 1;0 1 0 0 0 0 0 1;0 1 0 0 0 0 1 1;0 1 0 0 0 1 0 1;0 1 0 0 0 1 1 1;0 1 0 0 1 0 0 1;0 1 0 0 1 0 1 1;0 1 0 0 1 1 0 1;0 1 0 0 1 1 1 1;0 1 0 1 0 0 0 1;0 1 0 1 0 0 1 1;0 1 0 1 0 1 0 1;0 1 0 1 0 1 1 1;0 1 0 1 1 0 0 1;0 1 0 1 1 0 1 1;0 1 0 1 1 1 0 1;0 1 0 1 1 1 1 1;0 1 1 0 0 0 0 1;0 1 1 0 0 0 1 1;0 1 1 0 0 1 0 1;0 1 1 0 0 1 1 1;0 1 1 0 1 0 0 1;0 1 1 0 1 0 1 1;0 1 1 0 1 1 0 1;0 1 1 0 1 1 1 1;0 1 1 1 0 0 0 1;0 1 1 1 0 0 1 1;0 1 1 1 0 1 0 1;0 1 1 1 0 1 1 1;0 1 1 1 1 0 0 1;0 1 1 1 1 0 1 1;0 1 1 1 1 1 0 1;0 1 1 1 1 1 1 1; 1 0 0 0 0 0 0 1;1 0 0 0 0 0 1 1;1 0 0 0 0 1 0 1;1 0 0 0 0 1 1 1;1 0 0 0 1 0 0 1;1 0 0 0 1 0 1 1;1 0 0 0 1 1 0 1;1 0 0 0 1 1 1 1;1 0 0 1 0 0 0 1;1 0 0 1 0 0 1 1;1 0 0 1 0 1 0 1;1 0 0 1 0 1 1 1;1 0 0 1 1 0 0 1;1 0 0 1 1 0 1 1;1 0 0 1 1 1 0 1;1 0 0 1 1 1 1 1;1 0 1 0 0 0 0 1;1 0 1 0 0 0 1 1;1 0 1 0 0 1 0 1;1 0 1 0 0 1 1 1;1 0 1 0 1 0 0 1;1 0 1 0 1 0 1 1;1 0 1 0 1 1 0 1;1 0 1 0 1 1 1 1;1 0 1 1 0 0 0 1;1 0 1 1 0 0 1 1;1 0 1 1 0 1 0 1;1 0 1 1 0 1 1 1;1 0 1 1 1 0 0 1;1 0 1 1 1 0 1 1;1 0 1 1 1 1 0 1;1 0 1 1 1 1 1 1;1 1 0 0 0 0 0 1;1 1 0 0 0 0 1 1;1 1 0 0 0 1 0 1;1 1 0 0 0 1 1 1;1 1 0 0 1 0 0 1;1 1 0 0 1 0 1 1;1 1 0 0 1 1 0 1;1 1 0 0 1 1 1 1;1 1 0 1 0 0 0 1;1 1 0 1 0 0 1 1;1 1 0 1 0 1 0 1;1 1 0 1 0 1 1 1;1 1 0 1 1 0 0 1;1 1 0 1 1 0 1 1;1 1 0 1 1 1 0 1;1 1 0 1 1 1 1 1;1 1 1 0 0 0 0 1;1 1 1 0 0 0 1 1;1 1 1 0 0 1 0 1;1 1 1 0 0 1 1 1;1 1 1 0 1 0 0 1;1 1 1 0 1 0 1 1;1 1 1 0 1 1 0 1;1 1 1 0 1 1 1 1;1 1 1 1 0 0 0 1;1 1 1 1 0 0 1 1;1 1 1 1 0 1 0 1;1 1 1 1 0 1 1 1;1 1 1 1 1 0 0 1;1 1 1 1 1 0 1 1;1 1 1 1 1 1 0 1;1 1 1 1 1 1 1 1];
training_outx=[1 1 1 1 1 1 1 1;0 0 0 0 0 0 0 1;0 0 0 0 0 0 1 1;0 0 0 0 0 1 0 1;0 0 0 0 0 1 1 1;0 0 0 0 1 0 0 1;0 0 0 0 1 0 1 1;0 0 0 0 1 1 0 1;0 0 0 0 1 1 1 1;0 0 0 1 0 0 0 1;0 0 0 1 0 0 1 1;0 0 0 1 0 1 0 1;0 0 0 1 0 1 1 1;0 0 0 1 1 0 0 1;0 0 0 1 1 0 1 1;0 0 0 1 1 1 0 1;0 0 0 1 1 1 1 1;0 0 1 0 0 0 0 1;0 0 1 0 0 0 1 1;0 0 1 0 0 1 0 1;0 0 1 0 0 1 1 1;0 0 1 0 1 0 0 1;0 0 1 0 1 0 1 1;0 0 1 0 1 1 0 1;0 0 1 0 1 1 1 1;0 0 1 1 0 0 0 1;0 0 1 1 0 0 1 1;0 0 1 1 0 1 0 1;0 0 1 1 0 1 1 1;0 0 1 1 1 0 0 1;0 0 1 1 1 0 1 1;0 0 1 1 1 1 0 1;0 0 1 1 1 1 1 1;0 1 0 0 0 0 0 1;0 1 0 0 0 0 1 1;0 1 0 0 0 1 0 1;0 1 0 0 0 1 1 1;0 1 0 0 1 0 0 1;0 1 0 0 1 0 1 1;0 1 0 0 1 1 0 1;0 1 0 0 1 1 1 1;0 1 0 1 0 0 0 1;0 1 0 1 0 0 1 1;0 1 0 1 0 1 0 1;0 1 0 1 0 1 1 1;0 1 0 1 1 0 0 1;0 1 0 1 1 0 1 1;0 1 0 1 1 1 0 1;0 1 0 1 1 1 1 1;0 1 1 0 0 0 0 1;0 1 1 0 0 0 1 1;0 1 1 0 0 1 0 1;0 1 1 0 0 1 1 1;0 1 1 0 1 0 0 1;0 1 1 0 1 0 1 1;0 1 1 0 1 1 0 1;0 1 1 0 1 1 1 1;0 1 1 1 0 0 0 1;0 1 1 1 0 0 1 1;0 1 1 1 0 1 0 1;0 1 1 1 0 1 1 1;0 1 1 1 1 0 0 1;0 1 1 1 1 0 1 1;0 1 1 1 1 1 0 1;0 1 1 1 1 1 1 1;1 0 0 0 0 0 0 1;1 0 0 0 0 0 1 1;1 0 0 0 0 1 0 1;1 0 0 0 0 1 1 1;1 0 0 0 1 0 0 1;1 0 0 0 1 0 1 1;1 0 0 0 1 1 0 1;1 0 0 0 1 1 1 1;1 0 0 1 0 0 0 1;1 0 0 1 0 0 1 1;1 0 0 1 0 1 0 1;1 0 0 1 0 1 1 1;1 0 0 1 1 0 0 1;1 0 0 1 1 0 1 1;1 0 0 1 1 1 0 1;1 0 0 1 1 1 1 1;1 0 1 0 0 0 0 1;1 0 1 0 0 0 1 1;1 0 1 0 0 1 0 1;1 0 1 0 0 1 1 1;1 0 1 0 1 0 0 1;1 0 1 0 1 0 1 1;1 0 1 0 1 1 0 1;1 0 1 0 1 1 1 1;1 0 1 1 0 0 0 1;1 0 1 1 0 0 1 1;1 0 1 1 0 1 0 1;1 0 1 1 0 1 1 1;1 0 1 1 1 0 0 1;1 0 1 1 1 0 1 1;1 0 1 1 1 1 0 1;1 0 1 1 1 1 1 1;1 1 0 0 0 0 0 1;1 1 0 0 0 0 1 1;1 1 0 0 0 1 0 1;1 1 0 0 0 1 1 1;1 1 0 0 1 0 0 1;1 1 0 0 1 0 1 1;1 1 0 0 1 1 0 1;1 1 0 0 1 1 1 1;1 1 0 1 0 0 0 1;1 1 0 1 0 0 1 1;1 1 0 1 0 1 0 1;1 1 0 1 0 1 1 1;1 1 0 1 1 0 0 1;1 1 0 1 1 0 1 1;1 1 0 1 1 1 0 1;1 1 0 1 1 1 1 1;1 1 1 0 0 0 0 1;1 1 1 0 0 0 1 1;1 1 1 0 0 1 0 1;1 1 1 0 0 1 1 1;1 1 1 0 1 0 0 1;1 1 1 0 1 0 1 1;1 1 1 0 1 1 0 1;1 1 1 0 1 1 1 1;1 1 1 1 0 0 0 1;1 1 1 1 0 0 1 1;1 1 1 1 0 1 0 1;1 1 1 1 0 1 1 1;1 1 1 1 1 0 0 1;1 1 1 1 1 0 1 1;1 1 1 1 1 1 0 1;1 1 1 1 1 1 1 1];

```
1 0 1 0 1 1;0 0 1 0 1 1 0 1;0 0 1 0 1 1 1 1;0 0 1 1 0 0 0 1;0 0 1 1 0 0 1 1;0 0 1 1 0 1 0 1;0
0 1 1 0 1 1 1;0 0 1 1 1 0 0 1;0 0 1 1 1 0 1 1;0 0 1 1 1 1 0 1;0 0 1 1 1 1 1 1;0 1 0 0 0 0 0
1;0 1 0 0 0 0 1 1;0 1 0 0 0 1 0 1;0 1 0 0 0 1 1 1;0 1 0 0 1 0 0 1;0 1 0 0 1 0 1 1;0 1 0 0 1 1
0 1;0 1 0 0 1 1 1 1;0 1 0 1 0 0 0 1;0 1 0 1 0 0 1 1;0 1 0 1 0 1 0 1;0 1 0 1 0 1 1 1;0 1 0 1 1
0 0 1;0 1 0 1 1 0 1 1;0 1 0 1 1 1 0 1;0 1 0 1 1 1 1 1;0 1 1 0 0 0 0 1;0 1 1 0 0 0 1 1;0 1 1 0
0 1 0 1;0 1 1 0 0 1 1 1;0 1 1 0 1 0 0 1;0 1 1 0 1 0 1 1;0 1 1 0 1 1 0 1;0 1 1 0 1 1 1 1;0 1 1
1 0 0 0 1;0 1 1 1 0 0 1 1;0 1 1 1 0 1 0 1;0 1 1 1 0 1 1 1;0 1 1 1 1 0 0 1;0 1 1 1 1 0 1 1;0 1
1 1 1 1 0 1;0 1 1 1 1 1 1 1; 1 0 0 0 0 0 0 1;1 0 0 0 0 0 1 1;1 0 0 0 0 1 0 1;1 0 0 0 0 1 1 1;1
0 0 0 1 0 0 1;1 0 0 0 1 0 1 1;1 0 0 0 1 1 0 1;1 0 0 0 1 1 1 1;1 0 0 1 0 0 0 1;1 0 0 1 0 0 1
1;1 0 0 1 0 1 0 1;1 0 0 1 0 1 1 1;1 0 0 1 1 0 0 1;1 0 0 1 1 0 1 1;1 0 0 1 1 1 0 1;1 0 0 1 1 1
1 1;1 0 1 0 0 0 0 1;1 0 1 0 0 0 1 1;1 0 1 0 0 1 0 1;1 0 1 0 0 1 1 1;1 0 1 0 1 0 0 1;1 0 1 0 1
0 1 1;1 0 1 0 1 1 0 1;1 0 1 0 1 1 1 1;1 0 1 1 0 0 0 1;1 0 1 1 0 0 1 1;1 0 1 1 0 1 0 1;1 0 1 1
0 1 1 1;1 0 1 1 1 0 0 1;1 0 1 1 1 0 1 1;1 0 1 1 1 1 0 1;1 0 1 1 1 1 1 1;1 1 0 0 0 0 0 1;1 1 0
0 0 0 1 1;1 1 0 0 0 1 0 1;1 1 0 0 0 1 1 1;1 1 0 0 1 0 0 1;1 1 0 0 1 0 1 1;1 1 0 0 1 1 0 1;1 1
0 0 1 1 1 1;1 1 0 1 0 0 0 1;1 1 0 1 0 0 1 1;1 1 0 1 0 1 0 1;1 1 0 1 0 1 1 1;1 1 0 1 1 0 0 1;1
1 0 1 1 0 1 1;1 1 0 1 1 1 0 1;1 1 0 1 1 1 1 1;1 1 1 0 0 0 0 1;1 1 1 0 0 0 1 1;1 1 1 0 0 1 0
1;1 1 1 0 0 1 1 1;1 1 1 0 1 0 0 1;1 1 1 0 1 0 1 1;1 1 1 0 1 1 0 1;1 1 1 0 1 1 1 1;1 1 1 1 0 0
0 1;1 1 1 1 0 0 1 1;1 1 1 1 0 1 0 1;1 1 1 1 0 1 1 1;1 1 1 1 1 0 0 1;1 1 1 1 1 0 1 1;1 1 1 1 1
1 0 1;1 1 1 1 1 1 1 0;1 1 1 1 1 1 1 0; 0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 1 0;0 0 0 0 0 0 1 0 0;0 0 0 0
0 1 1 0;0 0 0 0 1 0 0 0;0 0 0 0 1 0 1 0;0 0 0 0 1 1 0 0;0 0 0 0 1 1 1 0;0 0 0 1 0 0 0 0;0 0 0
1 0 0 1 0;0 0 0 1 0 1 0 0;0 0 0 1 0 1 1 0;0 0 0 1 1 0 0 0;0 0 0 1 1 0 1 0;0 0 0 1 1 1 0 0;0 0
0 1 1 1 1 0;0 0 1 0 0 0 0 0;0 0 1 0 0 0 1 0;0 0 1 0 0 1 0 0;0 0 1 0 0 1 1 0;0 0 1 0 1 0 0 0;0
0 1 0 1 0 1 0;0 0 1 0 1 1 0 0;0 0 1 0 1 1 1 0;0 0 1 1 0 0 0 0;0 0 1 1 0 0 1 0;0 0 1 1 0 1 0
0;0 0 1 1 0 1 1 0;0 0 1 1 1 0 0 0;0 0 1 1 1 0 1 0;0 0 1 1 1 1 0 0;0 0 1 1 1 1 1 0;0 1 0 0 0 0
0 0;0 1 0 0 0 0 1 0;0 1 0 0 0 1 0 0;0 1 0 0 0 1 1 0;0 1 0 0 1 0 0 0;0 1 0 0 1 0 1 0;0 1 0 0 1
1 0 0;0 1 0 0 1 1 1 0;0 1 0 1 0 0 0 0;0 1 0 1 0 0 1 0;0 1 0 1 0 1 0 0;0 1 0 1 0 1 1 0;0 1 0 1
1 0 0 0;0 1 0 1 1 0 1 0;0 1 0 1 1 1 0 0;0 1 0 1 1 1 1 0;0 1 1 0 0 0 0 0;0 1 1 0 0 0 1 0;0 1 1
0 0 1 0 0;0 1 1 0 0 1 1 0;0 1 1 0 1 0 0 0;0 1 1 0 1 0 1 0;0 1 1 0 1 1 0 0;0 1 1 0 1 1 1 0;0 1
1 1 0 0 0 0;0 1 1 1 0 0 1 0;0 1 1 1 0 1 0 0;0 1 1 1 0 1 1 0;0 1 1 1 1 0 0 0;0 1 1 1 1 0 1 0;0
1 1 1 1 1 0 0;0 1 1 1 1 1 1 0; 1 0 0 0 0 0 0 0;1 0 0 0 0 0 0 1 0;1 0 0 0 0 0 1 0 0;1 0 0 0 0 0 1 1
0;1 0 0 0 1 0 0 0;1 0 0 0 1 0 1 0;1 0 0 0 1 1 0 0;1 0 0 0 1 1 1 0;1 0 0 1 0 0 0 0;1 0 0 1 0 0
1 0;1 0 0 1 0 1 0 0;1 0 0 1 0 1 1 0;1 0 0 1 1 0 0 0;1 0 0 1 1 0 1 0;1 0 0 1 1 1 0 0;1 0 0 1 1
1 1 0;1 0 1 0 0 0 0 0;1 0 1 0 0 0 1 0;1 0 1 0 0 1 0 0;1 0 1 0 0 1 1 0;1 0 1 0 1 0 0 0;1 0 1 0
1 0 1 0;1 0 1 0 1 1 0 0;1 0 1 0 1 1 1 0;1 0 1 1 0 0 0 0;1 0 1 1 0 0 1 0;1 0 1 1 0 1 0 0;1 0 1
1 0 1 1 0;1 0 1 1 1 0 0 0;1 0 1 1 1 0 1 0;1 0 1 1 1 1 0 0;1 0 1 1 1 1 1 0;1 1 0 0 0 0 0 0;1 1
0 0 0 0 1 0;1 1 0 0 0 1 0 0;1 1 0 0 0 1 1 0;1 1 0 0 1 0 0 0;1 1 0 0 1 0 1 0;1 1 0 0 1 1 0 0;1
1 0 0 1 1 1 0;1 1 0 1 0 0 0 0;1 1 0 1 0 0 1 0;1 1 0 1 0 1 0 0;1 1 0 1 0 1 1 0;1 1 0 1 1 0 0
0;1 1 0 1 1 0 1 0;1 1 0 1 1 1 0 0;1 1 0 1 1 1 1 0;1 1 1 0 0 0 0 0;1 1 1 0 0 0 1 0;1 1 1 0 0 1
0 0;1 1 1 0 0 1 1 0;1 1 1 0 1 0 0 0;1 1 1 0 1 0 1 0;1 1 1 0 1 1 0 0;1 1 1 0 1 1 1 0;1 1 1 1 0
0 0 0;1 1 1 1 0 0 1 0;1 1 1 1 0 1 0 0;1 1 1 1 0 1 1 0;1 1 1 1 1 0 0 0;1 1 1 1 1 0 1 0];
for x=1:100000
training_set=training_setx(a,:);          %assigning matrix
training_out=training_outx(a,:);          %assigning matrix
inputu=[1 training_set];                  %generating a new matrix
sum_h=(w1*(inputu)')';                    %multiplying the randomly generated weights
with transpose of above generated matrix and finally transposing it
out_h=1./(1+exp(-sum_h));                 %passing the above obtained values through a
Sigmoid Activation Function
```

```
% -------------------------- output layer
input_h=[1 out_h];                    %generating a matrix using the above obtained
values
sum_out=(w2*(input_h)')';             %multiplying the above obtained values transpose
matrix with weights and performing transpose again
out=1./(1+exp(-sum_out));             %applying Sigmoid activation Function to above
obtained values
% --------------------------- delta
delta_out=neta*(out.*(1-out)).*(training_out - out);%applying Delta rule at the output
layer considering the output defined in the training and the output which is generated
delta_h=(delta_out*w2).*input_h.*(1-input_h);       %applying delta rule considering
the  input to output layer
% update of weight -------------------------
% output layer ------------------------------
for t=1:(outp+states)                                %for loop
w2(t,:) = w2(t,:) + neta*delta_out(t)*input_h;      %output layer weights updation
end
% hidden layer ----------------------------------
for t=1:hl                                           %for loop
w1(t,:) = w1(t,:) + neta*delta_h(t+1)*inputu;       %hidden layer weights updation
end
for t=1:(outp+states)                                %for loop iterating for 8 times
if out(t)>=0.7                                %condition if the output is greater than 0.7 then
output is 1
out1(t)=1;
elseif out(t)<=0.2                            %else condition the output is less than 0.2 then
output is 0
out1(t)=0;
else out1(t)=out(t);                          %else output is same
end
end
m=[m sum(out-training_out)];                   %generating a matrix for mean square error
if out1==training_out                         %if generated output is equals to trained one
a=a+1;                                        %increment for the next input training data
sx=sx+1;                                      %increment iteration number
end
if a>((2^inpu)*key)                           %row number is greater than 256
a=a-((2^inpu)*key);                           %decrement a by the function
end
end
plot(m.*m);                                   %plot the mean square error
```

5.4 Decryption Testing:

```
%testing the program
t=tcpip('LocalHost',30000,'NetworkRole','Client','TimeOut',10);%creating a tcpip object
at the receiver
fopen(t);                                     %opening the tcpip connection
```

```
nam=fscanf(t,'%s\n')                    %reading the message which was sent
by the sender
b1=fread(t,2);                          %reading the round value and the key
c=b1(1);                                %assigning the round number to a variable
d=b1(2);                                %assigning the key to a variable
ipo=nam;                                %assigning the message which was sent by
the sender as an input to decryption
statx=d;                                %assigning key to a variable
z=0;                                    %assigning a variable to zero
while(z~=c)                             %iterating the loop till the round number is
reached
finp=[];                                %initializing a matrix
for i=1:length(ipo)                     %iterating the loop till the length of
message is reached
b=ipo(i);                               %taking the message charcters according to
the value of i
switch b                                %using the switch condition to get the
binary value of the character
case('C')
set=[0 0 0 0 0 0 0];
case('f')
set=[0 0 0 0 0 0 1];
case('4')
set=[0 0 0 0 0 1 0];
case('@')
set=[0 0 0 0 0 1 1];
case(' ')
set=[0 0 0 0 1 0 0];
case('A')
set=[0 0 0 0 1 0 1];
case('h')
set=[0 0 0 0 1 1 0];
case('6')
set=[0 0 0 0 1 1 1];
case('#')
set=[0 0 0 1 0 0 0];
case('B')
set=[0 0 0 1 0 0 1];
case('b')
set=[0 0 0 1 0 1 0];
case('3')
set=[0 0 0 1 0 1 1];
case('(')
set=[0 0 0 1 1 0 0];
case('M')
set=[0 0 0 1 1 0 1];
case('n')
set=[0 0 0 1 1 1 0];
```

```
case('9')
set=[0 0 0 1 1 1 1];
case('D')
set=[0 0 1 0 0 0 0];
case('j')
set=[0 0 1 0 0 0 1];
case('p')
set=[0 0 1 0 0 1 0];
case('8')
set=[0 0 1 0 0 1 1];
case('$')
set=[0 0 1 0 1 0 0];
case('P')
set=[0 0 1 0 1 0 1];
case('r')
set=[0 0 1 0 1 1 0];
case('+')
set=[0 0 1 0 1 1 1];
case('7')
set=[0 0 1 1 0 0 0];
case('E')
set=[0 0 1 1 0 0 1];
case('d')
set=[0 0 1 1 0 1 0];
case('0')
set=[0 0 1 1 0 1 1];
case('!')
set=[0 0 1 1 1 0 0];
case('Z')
set=[0 0 1 1 1 0 1];
case('s')
set=[0 0 1 1 1 1 0];
case('<')
set=[0 0 1 1 1 1 1];
case('F')
set=[0 1 0 0 0 0 0];
case('a')
set=[0 1 0 0 0 0 1];
case('5')
set=[0 1 0 0 0 1 0];
case('%')
set=[0 1 0 0 0 1 1];
case('G')
set=[0 1 0 0 1 0 0];
case('c')
set=[0 1 0 0 1 0 1];
case('l')
set=[0 1 0 0 1 1 0];
```

```
case('=')
set=[0 1 0 0 1 1 1];
case('I')
set=[0 1 0 1 0 0 0];
case('g')
set=[0 1 0 1 0 0 1];
case('2')
set=[0 1 0 1 0 1 0];
case('|')
set=[0 1 0 1 0 1 1];
case('H')
set=[0 1 0 1 1 0 0];
case('t')
set=[0 1 0 1 1 0 1];
case('\')
set=[0 1 0 1 1 1 0];
case('N')
set=[0 1 0 1 1 1 1];
case('k')
set=[0 1 1 0 0 0 0];
case('{')
set=[0 1 1 0 0 0 1];
case('K')
set=[0 1 1 0 0 1 0];
case('u')
set=[0 1 1 0 0 1 1];
case('}')
set=[0 1 1 0 1 0 0];
case('e')
set=[0 1 1 0 1 0 1];
case('J')
set=[0 1 1 0 1 1 0];
case('[')
set=[0 1 1 0 1 1 1];
case('L')
set=[0 1 1 1 0 0 0];
case('v')
set=[0 1 1 1 0 0 1];
case(']')
set=[0 1 1 1 0 1 0];
case('O')
set=[0 1 1 1 0 1 1];
case('q')
set=[0 1 1 1 1 0 0];
case('?')
set=[0 1 1 1 1 0 1];
case('S')
set=[0 1 1 1 1 1 0];
```

```
case('z')
set=[0 1 1 1 1 1 1];
case(':')
set=[1 0 0 0 0 0 0];
case('Q')
set=[1 0 0 0 0 0 1];
case('i')
set=[1 0 0 0 0 1 0];
case(';')
set=[1 0 0 0 0 1 1];
case('R')
set=[1 0 0 0 1 0 0];
case('1')
set=[1 0 0 0 1 0 1];
case('m')
set=[1 0 0 0 1 1 0];
case('T')
set=[1 0 0 0 1 1 1];
case('y')
set=[1 0 0 1 0 0 0];
case('/')
set=[1 0 0 1 0 0 1];
case('"')
set=[1 0 0 1 0 1 0];
case('U')
set=[1 0 0 1 0 1 1];
case('o')
set=[1 0 0 1 1 0 0];
case('&')
set=[1 0 0 1 1 0 1];
case('V')
set=[1 0 0 1 1 1 0];
case('x')
set=[1 0 0 1 1 1 1];
case('*')
set=[1 0 1 0 0 0 0];
case('W')
set=[1 0 1 0 0 0 1];
case('.')
set=[1 0 1 0 0 1 0];
case('w')
set=[1 0 1 0 0 1 1];
case(',')
set=[1 0 1 0 1 0 0];
case('ß')
set=[1 0 1 0 1 0 1];
case('X')
set=[1 0 1 0 1 1 0];
```

```
case('^')
set=[1 0 1 0 1 1 1];
case('Y')
set=[1 0 1 1 0 0 0];
case('-')
set=[1 0 1 1 0 0 1];
case('_')
set=[1 0 1 1 0 1 0];
case(')')
set=[1 0 1 1 0 1 1];
case('~')
set=[1 0 1 1 1 0 0];
case('`')
set=[1 0 1 1 1 0 1];
case('>')
set=[1 0 1 1 1 1 0];
case('ô')
set=[1 0 1 1 1 1 1];
case('÷')
set=[1 1 0 0 0 0 0];
case('è')
set=[1 1 0 0 0 0 1];
case('Ø')
set=[1 1 0 0 0 1 0];
case('ï')
set=[1 1 0 0 0 1 1];
case('Ý')
set=[1 1 0 0 1 0 0];
case('Í')
set=[1 1 0 0 1 0 1];
case('ù')
set=[1 1 0 0 1 1 0];
case('1')
set=[1 1 0 0 1 1 1];
case('«')
set=[1 1 0 1 0 0 0];
case('»')
set=[1 1 0 1 0 0 1];
case('ç')
set=[1 1 0 1 0 1 0];
case('ã')
set=[1 1 0 1 0 1 1];
case('å')
set=[1 1 0 1 1 0 0];
case('°')
set=[1 1 0 1 1 0 1];
case('½')
set=[1 1 0 1 1 1 0];
```

```
case('Ñ')
set=[1 1 0 1 1 1 1];
case('Ä')
set=[1 1 1 0 0 0 0];
case('µ')
set=[1 1 1 0 0 0 1];
case('¾')
set=[1 1 1 0 0 1 0];
case('©')
set=[1 1 1 0 0 1 1];
case('ð')
set=[1 1 1 0 1 0 0];
case('Ô')
set=[1 1 1 0 1 0 1];
case('á')
set=[1 1 1 0 1 1 0];
case('õ')
set=[1 1 1 0 1 1 1];
case('ó')
set=[1 1 1 1 0 0 0];
case('ê')
set=[1 1 1 1 0 0 1];
case('Ê')
set=[1 1 1 1 0 1 0];
case('ý')
set=[1 1 1 1 0 1 1];
case('£')
set=[1 1 1 1 1 0 0];
case('ñ')
set=[1 1 1 1 1 0 1];
case('ë')
set=[1 1 1 1 1 1 0];
case('Æ')
set=[1 1 1 1 1 1 1];
end
ipox=[set statx];                    %generating a matrix using the generated binary value
and the key
inputp=[1 ipox];                     %generating another matrix using above matrix
sum_h=(w1*(inputp)')';                  %multiplying the weights with transpose of above
matrix and again transpose it
o_h=1./(1+exp(-sum_h));                 %passing the above matrix values into a sigmoid
activation function
%output layer ----------------------------
input_h=[1 o_h];                     %generating another matrix using the above  obtained
values
sum_out=(w2*(input_h)')';               %multiplying with the transpose of above matrix
and find transpose again
out=1./(1+exp(-sum_out));               %passing the above obtained values into a sigmoid
```

```
for t=1:(states+outp)                %iterating the loop for 8 times
if out(t)>=0.8                       %if output is greater than or equal to 0.8 then output is 1
out1(t)=1;
elseif out(t)<=0.4                   %output is less than or equal to 0.4 then output is 0
out1(t)=0;
else out1(t)=out(t);
end
end
finp=[finp;out1];                    %generating a matrix with the outputs for the given
input binary bits using the training
tempch=[];                           %declaring an empty matrix
statx=out1( (outp + 1):(outp+states));  %defining the out1(8:8) last bit indicates the key
end
outzs=[''];                          %declaring a matrix
for f=1:length(ipo)                  %iterating the loop length(message) times
tempch=finp(f,:);                    %using the every row of above obtained matrix
tempch=tempch(1:7);                  %taking only 7 bits into consideration as the last bit
is the key
%here the generated output bits for the given input bits is taken into
%consideration and respective character to the output bit is printed
if tempch==[0 0 0 0 0 0 0]
outzs=[outzs 'C'];
end
if tempch==[0 0 0 0 0 0 1]
outzs=[outzs 'f'];
end
if tempch==[0 0 0 0 0 1 0]
outzs=[outzs '4'];
end
if tempch==[0 0 0 0 0 1 1]
outzs=[outzs '@'];
end
if tempch==[0 0 0 0 1 0 0]
outzs=[outzs ' '];
end
if tempch==[0 0 0 0 1 0 1]
outzs=[outzs 'A'];
end
if tempch==[0 0 0 0 1 1 0]
outzs=[outzs 'h'];
end
if tempch==[0 0 0 0 1 1 1]
outzs=[outzs '6'];
end
if tempch==[0 0 0 1 0 0 0]
outzs=[outzs '#'];
end
if tempch==[0 0 0 1 0 0 1]
```

```
outzs=[outzs 'B'];
end
if tempch==[0 0 0 1 0 1 0]
outzs=[outzs 'b'];
end
if tempch==[0 0 0 1 0 1 1]
outzs=[outzs '3'];
end
if tempch==[0 0 0 1 1 0 0]
outzs=[outzs '('];
end
if tempch==[0 0 0 1 1 0 1]
outzs=[outzs 'M'];
end
if tempch==[0 0 0 1 1 1 0]
outzs=[outzs 'n'];
end
if tempch==[0 0 0 1 1 1 1]
outzs=[outzs '9'];
end
if tempch==[0 0 1 0 0 0 0]
outzs=[outzs 'D'];
end
if tempch==[0 0 1 0 0 0 1]
outzs=[outzs 'j'];
end
if tempch==[0 0 1 0 0 1 0]
outzs=[outzs 'p'];
end
if tempch==[0 0 1 0 0 1 1]
outzs=[outzs '8'];
end
if tempch==[0 0 1 0 1 0 0]
outzs=[outzs '$'];
end
if tempch==[0 0 1 0 1 0 1]
outzs=[outzs 'P'];
end
if tempch==[0 0 1 0 1 1 0]
outzs=[outzs 'r'];
end
if tempch==[0 0 1 0 1 1 1]
outzs=[outzs '+'];
end
if tempch==[0 0 1 1 0 0 0]
outzs=[outzs '7'];
end
if tempch==[0 0 1 1 0 0 1]
```

```
outzs=[outzs 'E'];
end
if tempch==[0 0 1 1 0 1 0]
outzs=[outzs 'd'];
end
if tempch==[0 0 1 1 0 1 1]
outzs=[outzs '0'];
end
if tempch==[0 0 1 1 1 0 0]
outzs=[outzs '!'];
end
if tempch==[0 0 1 1 1 0 1]
outzs=[outzs 'Z'];
end
if tempch==[0 0 1 1 1 1 0]
outzs=[outzs 's'];
end
if tempch==[0 0 1 1 1 1 1]
outzs=[outzs '<'];
end
if tempch==[0 1 0 0 0 0 0]
outzs=[outzs 'F'];
end
if tempch==[0 1 0 0 0 0 1]
outzs=[outzs 'a'];
end
if tempch==[0 1 0 0 0 1 0]
outzs=[outzs '5'];
end
if tempch==[0 1 0 0 0 1 1]
outzs=[outzs '%'];
end
if tempch==[0 1 0 0 1 0 0]
outzs=[outzs 'G'];
end
if tempch==[0 1 0 0 1 0 1]
outzs=[outzs 'c'];
end
if tempch==[0 1 0 0 1 1 0]
outzs=[outzs 'l'];
end
if tempch==[0 1 0 0 1 1 1]
outzs=[outzs '='];
end
if tempch==[0 1 0 1 0 0 0]
outzs=[outzs 'I'];
end
if tempch==[0 1 0 1 0 0 1]
```

```
outzs=[outzs 'g'];
end
if tempch==[0 1 0 1 0 1 0]
outzs=[outzs '2'];
end
if tempch==[0 1 0 1 0 1 1]
outzs=[outzs '|'];
end
if tempch==[0 1 0 1 1 0 0]
outzs=[outzs 'H'];
end
if tempch==[0 1 0 1 1 0 1]
outzs=[outzs 't'];
end
if tempch==[0 1 0 1 1 1 0]
outzs=[outzs '\'];
end
if tempch==[0 1 0 1 1 1 1]
outzs=[outzs 'N'];
end
if tempch==[0 1 1 0 0 0 0]
outzs=[outzs 'k'];
end
if tempch==[0 1 1 0 0 0 1]
outzs=[outzs '{'];
end
if tempch==[0 1 1 0 0 1 0]
outzs=[outzs 'K'];
end
if tempch==[0 1 1 0 0 1 1]
outzs=[outzs 'u'];
end
if tempch==[0 1 1 0 1 0 0]
outzs=[outzs '}'];
end
if tempch==[0 1 1 0 1 0 1]
outzs=[outzs 'e'];
end
if tempch==[0 1 1 0 1 1 0]
outzs=[outzs 'J'];
end
if tempch==[0 1 1 0 1 1 1]
outzs=[outzs '['];
end
if tempch==[0 1 1 1 0 0 0]
outzs=[outzs 'L'];
end
if tempch==[0 1 1 1 0 0 1]
```

```
outzs=[outzs 'v'];
end
if tempch==[0 1 1 1 0 1 0]
outzs=[outzs ']'];
end
if tempch==[0 1 1 1 0 1 1]
outzs=[outzs 'O'];
end
if tempch==[0 1 1 1 1 0 0]
outzs=[outzs 'q'];
end
if tempch==[0 1 1 1 1 0 1]
outzs=[outzs '?'];
end
if tempch==[0 1 1 1 1 1 0]
outzs=[outzs 'S'];
end
if tempch==[0 1 1 1 1 1 1]
outzs=[outzs 'z'];
end
if tempch==[1 0 0 0 0 0 0]
outzs=[outzs ':'];
end
if tempch==[1 0 0 0 0 0 1]
outzs=[outzs 'Q'];
end
if tempch==[1 0 0 0 0 1 0]
outzs=[outzs 'i'];
end
if tempch==[1 0 0 0 0 1 1]
outzs=[outzs ';'];
end
if tempch==[1 0 0 0 1 0 0]
outzs=[outzs 'R'];
end
if tempch==[1 0 0 0 1 0 1]
outzs=[outzs '1'];
end
if tempch==[1 0 0 0 1 1 0]
outzs=[outzs 'm'];
end
if tempch==[1 0 0 0 1 1 1]
outzs=[outzs 'T'];
end
if tempch==[1 0 0 1 0 0 0]
outzs=[outzs 'y'];
end
if tempch==[1 0 0 1 0 0 1]
```

```
outzs=[outzs '/'];
end
if tempch==[1 0 0 1 0 1 0]
outzs=[outzs ''''];
end
if tempch==[1 0 0 1 0 1 1]
outzs=[outzs 'U'];
end
if tempch==[1 0 0 1 1 0 0]
outzs=[outzs 'o'];
end
if tempch==[1 0 0 1 1 0 1]
outzs=[outzs '&'];
end
if tempch==[1 0 0 1 1 1 0]
outzs=[outzs 'V'];
end
if tempch==[1 0 0 1 1 1 1]
outzs=[outzs 'x'];
end
if tempch==[1 0 1 0 0 0 0]
outzs=[outzs '*'];
end
if tempch==[1 0 1 0 0 0 1]
outzs=[outzs 'W'];
end
if tempch==[1 0 1 0 0 1 0]
outzs=[outzs '.'];
end
if tempch==[1 0 1 0 0 1 1]
outzs=[outzs 'w'];
end
if tempch==[1 0 1 0 1 0 0]
outzs=[outzs ','];
end
if tempch==[1 0 1 0 1 0 1]
outzs=[outzs 'ß'];
end
if tempch==[1 0 1 0 1 1 0]
outzs=[outzs 'X'];
end
if tempch==[1 0 1 0 1 1 1]
outzs=[outzs '^'];
end
if tempch==[1 0 1 1 0 0 0]
outzs=[outzs 'Y'];
end
if tempch==[1 0 1 1 0 0 1]
```

```
outzs=[outzs '-'];
end
if tempch==[1 0 1 1 0 1 0]
outzs=[outzs '_'];
end
if tempch==[1 0 1 1 0 1 1]
outzs=[outzs ')'];
end
if tempch==[1 0 1 1 1 0 0]
outzs=[outzs '~'];
end
if tempch==[1 0 1 1 1 0 1]
outzs=[outzs '`'];
end
if tempch==[1 0 1 1 1 1 0]
outzs=[outzs '>'];
end
if tempch==[1 0 1 1 1 1 1]
outzs=[outzs 'ô'];
end
if tempch==[1 1 0 0 0 0 0]
outzs=[outzs '÷'];
end
if tempch==[1 1 0 0 0 0 1]
outzs=[outzs 'è'];
end
if tempch==[1 1 0 0 0 1 0]
outzs=[outzs 'Ø'];
end
if tempch==[1 1 0 0 0 1 1]
outzs=[outzs 'ï'];
end
if tempch==[1 1 0 0 1 0 0]
outzs=[outzs 'Ý'];
end
if tempch==[1 1 0 0 1 0 1]
outzs=[outzs 'Í'];
end
if tempch==[1 1 0 0 1 1 0]
outzs=[outzs 'ù'];
end
if tempch==[1 1 0 0 1 1 1]
outzs=[outzs '¹'];
end
if tempch==[1 1 0 1 0 0 0]
outzs=[outzs '«'];
end
if tempch==[1 1 0 1 0 0 1]
```

```
outzs=[outzs '»'];
end
if tempch==[1 1 0 1 0 1 0]
outzs=[outzs 'ç'];
end
if tempch==[1 1 0 1 0 1 1]
outzs=[outzs 'ã'];
end
if tempch==[1 1 0 1 1 0 0]
outzs=[outzs 'å'];
end
if tempch==[1 1 0 1 1 0 1]
outzs=[outzs 'º'];
end
if tempch==[1 1 0 1 1 1 0]
outzs=[outzs '½'];
end
if tempch==[1 1 0 1 1 1 1]
outzs=[outzs 'Ñ'];
end
if tempch==[1 1 1 0 0 0 0]
outzs=[outzs 'Ä'];
end
if tempch==[1 1 1 0 0 0 1]
outzs=[outzs 'µ'];
end
if tempch==[1 1 1 0 0 1 0]
outzs=[outzs '¾'];
end
if tempch==[1 1 1 0 0 1 1]
outzs=[outzs '©'];
end
if tempch==[1 1 1 0 1 0 0]
outzs=[outzs 'ð'];
end
if tempch==[1 1 1 0 1 0 1]
outzs=[outzs 'Ô'];
end
if tempch==[1 1 1 0 1 1 0]
outzs=[outzs 'á'];
end
if tempch==[1 1 1 0 1 1 1]
outzs=[outzs 'õ'];
end
if tempch==[1 1 1 1 0 0 0]
outzs=[outzs 'ó'];
end
if tempch==[1 1 1 1 0 0 1]
```

```
outzs=[outzs 'ê'];
end
if tempch==[1 1 1 1 0 1 0]
outzs=[outzs 'Ê'];
end
if tempch==[1 1 1 1 0 1 1]
outzs=[outzs 'ý'];
end
if tempch==[1 1 1 1 1 0 0]
outzs=[outzs '£'];
end
if tempch==[1 1 1 1 1 0 1]
outzs=[outzs 'ñ'];
end
if tempch==[1 1 1 1 1 1 0]
outzs=[outzs 'ë'];
end
if tempch==[1 1 1 1 1 1 1]
outzs=[outzs 'Æ'];
end
end
ipo=outzs                                   %assigning the output generated to input as
mentioned above round decryption is performed
z=z+1;                                      %incrementing the value of z
end
outzs
```

CHAPTER 6

RESULTS, APPLICATIONS AND CONCLUSION

6.1 RESULTS

Encryption Training Graph:

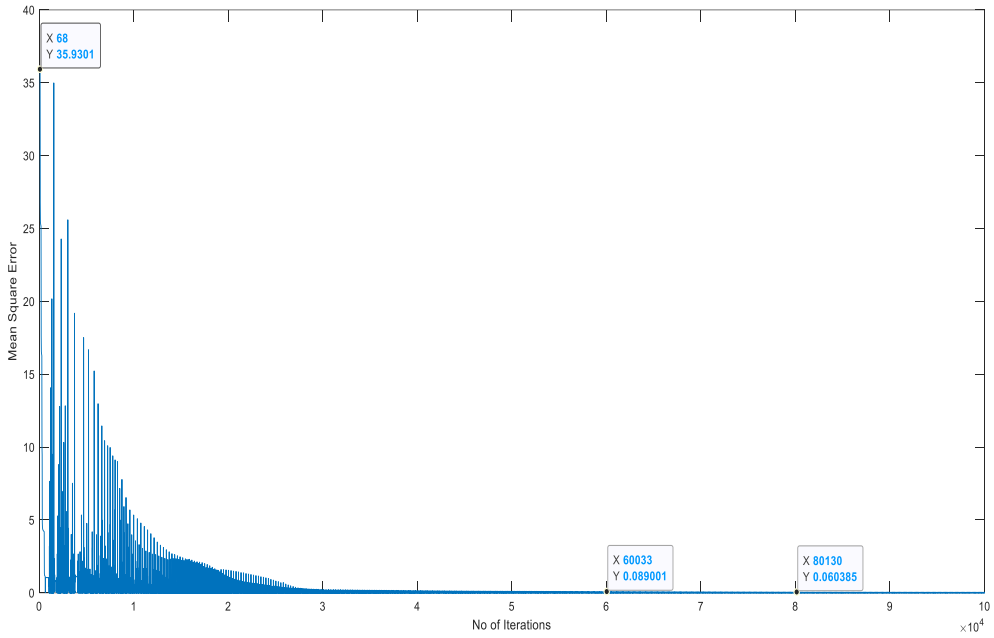


Fig 6.1 Mean Square Error figure at Encryption Training

OUTPUT ON COMMAND WINDOW FOR ENCRYPTION TESTING

Message Encryption , Key and Rounds:

enter the text : Hai How are You? Pandemic CoronaVirus is spreading Rapidly.

ipo =

't%;htV,h57Jh-V}zA+5D0[TRlhfV+V9%xR+e<h;FAF87J%0R9|Am5$;!="w'

ipo =

'NG16NxX6GEL6)xJ:67GjZL/1I6@xExjGW1EJa61a6aPELGZ1jH6TGP1ZgUß'

ipo =

'klmBkW^Bc0vB~W[i#dc8s]"TgB WdWpl.TdL5Bm%#%r0vlsTp\#/c+m<2&X'

ipo =

'K=ybK.-b=!Ob>.v;b0=$FOoy|bh.!.$=,y!vGbyGbG7!O=Fy$Nb"=7yFHVY'

ipo =

'ug/(u,_(Isq(ô,]13ZIra?&"H(6,Z,Pgß"ZOc(/l3lEsqga"P{3oId/5t*-'

ipo =

'e2UMeß~M2<SMèßqmMs2+%SxU\MBß<ß+2^U<q=MU=M=0<S2%U+KM&20U%NW)'

outzs =

'e2UMeß~M2<SMèßqmMs2+%SxU\MBß<ß+2^U<q=MU=M=0<S2%U+KM&20U%NW)'

>> tx

tx =

datetime

11-Apr-2020 16:22:00

>> day

day =

0

>> da

da =

6

>> sa

sa =

2

>> sb

sb =

0

>> sc

sc =

4

>> st

st =

0

NOTE: Here we are not entering any key, number of rounds instead we are using the date and time, using the date, minutes, seconds to calculate the rounds and key.

Explanation:

For example:11-Apr-2020 16:22:00

Here we are taking the date which is 11 and adding the 1+1=2 which is sa

Now the seconds at which we clicked on Run button is 00 that is sb=0+0=0

Minutes is 22 and sc=2+2=4

Now we need to add sa+sb+sc

That is da which represents number of rounds:da=sa+sb+sc=2+0+4=6

That is number of times message gets encrypted is 6 times

If the Round number is even then key is 0 else it is 1.

Decryption Training Graph:

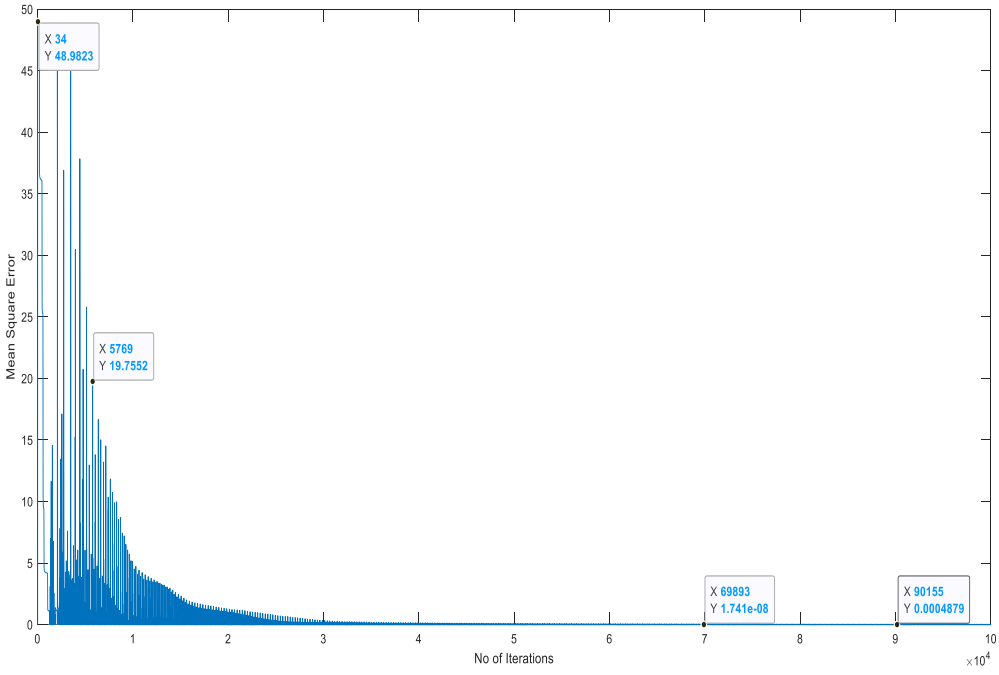


Fig 6.2 Mean Square Error figure at Decryption Training

OUTPUT ON COMMAND WINDOW FOR DECRYPTION TESTING

Message Decryption:

```
>> decry7

nam =

'e2UMeß~M2<SMèßqmMs2+%SxU\MBß<ß+2^U<q=MU=M=0<S2%U+KM&20U%
NW)'

ipo =

   '}I"3}w)3gZ?3÷wOR(!gP5qV/t3#wswrIX/s]l3"c(cdZ?I5/rk(UgE"a\x_'

ipo =

   'K=ybK.-b=!Ob>.v;b0=$FOoy|bh.!.$=,y!vGbyGbG7!O=Fy$Nb"=7yFHVY'

ipo =

   '{cT#{*Y#ld]#`*LQBElp<vUm2#A*0*8cwm0[%#T5B5+d]c<m8tBylrTs|o^

ipo =

   'NG16NxX6GEL6)xJ:67GjZL/1I6@xExjGW1EJa61a6aPELGZ1jH6TGP1ZgUß'

ipo =

   '\5RA\&ßA%+[A_&eShr%9!Jy;=A4&7&D5*;7}FAR<h<$+[5!;D2h1%8R0I/,'

ipo =

   'Hai How are You? Pandemic CoronaVirus is spreading Rapidly.'

outzs =

   'Hai How are You? Pandemic CoronaVirus is spreading Rapidly.'

>> c

c =

    6

>> d

d =

    0
```

NOTE: As you can see that the last Encrypted message is same as the received message at the input of Decryption algorithm. In the above you can see the rounds and key are transmitted to the receiver using tcpip and we can see those values in ‘c’ and ‘d’

variables. ‘c’ represents rounds and ‘d’ represents key, with the help of the received encrypted message, key and rounds we can decrypt the message. As You can see that the message which is encrypted is same as the decrypted one.

6.2 Applications of cryptography

Conventionally, cryptography was in implementation only for securing purposes. Wax seals, hand signatures and few other kinds of security methods were generally utilized to make sure of reliability and accuracy of the transmitter. And with the arrival of digital transmissions, security becomes more essential and then cryptography mechanisms began to outstrip its utilization for maintaining utmost secrecy. A few of the applications of cryptography are discussed below.

To Maintain Secrecy in Storage

Cryptography allows storing the encrypted data permitting users to stay back from the major hole of circumvention by hackers.

Reliability in Transmission

A conventional approach that allows reliability is to carry out a checksum of the communicated information and then communicate the corresponding checksum in an encrypted format. When both the checksum and encrypted data is received, the data is again checksummed and compared to the communicated checksum after the process of decryption. Thus, effective cryptographic mechanisms are more crucial to assure reliability in message transmission.

Authentication of Identity

Cryptography is strongly linked to the approach of using passwords, and innovative systems probably make use of strong cryptographic methods together with the physical methods of individuals and collective secrets offering highly reliable verification of identity.

Examples

The **examples of cryptograph**y include the following.

One of the prominent examples of cryptography encryption these days is end-to-end encryption in WhatsApp. This feature is included in WhatsApp through the asymmetry

model or via public key methods. Here only the destined member knows about the actual message. Once after the installation of WhatsApp is finished, public keys are registered with the server and then messages are transmitted.

The next real-time application of cryptography is digital signatures. In the situation that when two clients are necessary to sign documents for a business transaction. But when two clients never come across each other they might not believe each other. Then encryption in the digital signatures ensures enhanced authentication and security.

As cyber-attacks are constantly progressing, security needs to be more necessary and thus cryptography methodologies also become more prominent. These cryptographic algorithms not only let down the hacking activities but shows no scope for these activities to emerge.

Secure communications

The most obvious use of cryptography, and the one that all of us use frequently, is encrypting communications between us and another system. This is most commonly used for communicating between a client program and a server. Examples are a web browser and web server, or email client and email server. When the internet was developed it was a small academic and government community, and misuse was rare. Most systems communicated in the clear (without encryption), so anyone who intercepted network traffic could capture communications and passwords. Modern switched networks make interception harder, but some cases – for example, public wifi – still allow it. To make the internet more secure, most communication protocols have adopted encryption. Many older protocols have been dropped in favour of newer, encrypted replacements.

6.3 CONCLUSION

Artificial Neural Networks is a simple yet powerful technique which has the ability to emulate highly complex computational machines. In this project, we have used this technique to built simple combinational logic and sequential machine using back-propagation algorithm. A comparative study has been done between two different neural network architectures and their merits/demerits are mentioned. ANNs can be used to implement much complex combinational as well as sequential circuits.

Data security is a prime concern in data communication systems. The use of ANN in the field of Cryptography is investigated using “A sequential machine” based method for encryption of data is designed. Better results can be achieved by improvement of code or by use of better training algorithms. Thus, Artificial Neural Network can be used as a new method of encryption and decryption of data.

REFERENCES

1) International Journal of Innovative Research in Computer and Communication Engineering (An ISO 3297: 2007 Certified Organization) Vol. 4, Special Issue 4, August 2016

IJIRCCE www.ijircce.com

A Novel Approach for Cryptography Using Artificial Neural Networks by

Neeru Rathee, Rajat Sachdeva,Vijul Dalel, Yatin Jaie

Assistant Professor, Dept. of ECE, Maharaja Surajmal Institute of Technology, New Delhi, India, Dept. of ECE, Maharaja Surajmal Institute of Technology, New Delhi, India

2) International Journal of Advanced Research in Electronics and Communication Engineering (IJARECE) Volume 4, Issue 11, November 2015 2785 All Rights Reserved © 2015 IJARECE

Cryptography Based on Artificial Neural Network by Shital Daulat Jagtap, Mr. P. BalaRamudu, Mr. Manoj Kumar Singh

3) SunitaBhati, Anita Bhati and S. K. Sharma, "A New Approach towards Encryption Schemes: Byte - Rotation Encryption Algorithm," in Proceedings of the World Congress on Engineering and Computer Science 2012 Vol II WCECS 2012, October 24- 26,2012, San Francisco,USA.

4) Data Communications and Networking by Ben forouzan for information transmission(TCP IP)