A

Project Report on

Cryptography using Artificial Neural Networks

In partial fulfillment of the requirements of

Bachelor of Technology

In

Electronics & Instrumentation Engineering

Submitted By

Vikas Gujral
10507027

Satish Kumar Pradhan
10507033

Under the guidance of

Prof. G. S. Rath

Department of Electronics and Communication Engineering

National Institute of Technology

Rourkela-769008

Orissa

National Institute of Technology
Rourkela

CERTIFICATE

This is to certify that that the work in this thesis report entitled “***Cryptography using Artificial Neural Networks”*** submitted by Vikas Gujral and Satish kumar Pradhan in partial fulfillment of the requirements for the degree of Bachelor of Technology in Electronics & Instrumentation Engineering, Session 2005-2009, in the Department of Electronics and Communication Engineering, National Institute of Technology, Rourkela is an authentic work carried out by them under my supervision and guidance.

To the best of my knowledge the matter embodied in the thesis has not been submitted to any other University /Institute for the award of any degree.

Date:

Prof. G. S. Rath
Department of Electronics and
Communication Engineering
National Institute of Technology
Rourkela - 769008

ACKNOWLEDGEMENT

We owe a debt of deepest gratitude to our thesis supervisor, **Prof. G. S. Rath**, Department of Electronics and Communication Engineering, for his guidance, support, motivation and encouragement throughout the period this work was carried out. His readiness for consultation at all times, his educative comments, his concern and assistance even with practical things have been invaluable.

We are grateful to **Prof. S. K. Patra**, Head of the Department, Electronics and Communication Engineering for providing us the necessary opportunities for the completion of our project. We also thank the other faculty and staff members of our department for their invaluable help and guidance.

Vikas Gujral
10507027
E.I.E.

Satish kumar Pradhan
10507033
E.I.E.

CONTENTS

Chapter		**Page No.**
Certificate		2
Acknowledgement		3
List of figures		6
Abstract		7
Chapter 1	**Introduction**	**8**
1.1	Artificial Neural Networks	9
1.1.1	Introduction to ANN	9
1.1.2	Biological Model	9
1.1.3	ANN structure	11
1.1.4	Network Architectures	14
1.1.5	Learning Processes	16
1.1.6	Training of ANNs	16
1.1.7	Back-propagation	17
1.1.8	Multi-layer Feed-forward Network	17
1.1.9	Generalized delta rule	17
1.2	Cryptography	19
1.2.1	Introduction	19
1.2.2	Types of cryptographic algorithms	19
Chapter 2	**Application of Neural Network**	**21**
2.1	Sequential Machine	22
2.1.1	Serial Adder	22
2.1.2	Sequential Detector	23
2.2	Combinational Logic	24
2.3	Cryptography	25
2.3.1	Using Sequential machine	25
2.3.2	Using chaotic neural network	26
Chapter 3	**Implementation**	**28**
3.1	Sequential machine implementation	29

3.2	Combinational logic implementation	31
3.3	Cryptography using ANN bases sequential machine	34
3.4	Cryptography using chaotic neural network	35
Chapter 4	**Results and Conclusion**	**38**
4.1	A general n-state sequential machine	39
4.2	Combinational logic	42
4.3	Cryptography using sequential machine	44
4.4	Cryptography using chaotic neural network	44
4.5	Conclusion	47
APPENDIX 1: Matlab Code		48
APPENDIX 2: References		59

LIST OF FIGURES

Fig. No.	**Title**	**Page**
1.1	Block Diagram of a Human Nervous System	10
1.2	Schematic Diagram of a biological neuron	11
1.3	Basic components of an Artificial Neural Network	12
1.4	Various activation signals for a unit	14
1.5	Single layer feed-forward network	15
1.6	Multi layer feed-forward network	16
1.7	Recurrent network	16
1.8	A multi layer network with *l* neurons	19
1.9	Different encryption algorithms	21
2.1	State diagram of serial adder	23
2.2	State diagram of sequential detector	24
3.1	Jordan Network	30
3.2	Types of network architectures	33
3.3	Program flow chart	34
4.1	Entering input data in the sequential machine	39
4.2	Plot of error function after learning process	40
4.3	Output of the serial adder	41
4.4	MSE of fully connected neural network Type 1	42
4.5	MSE of a one output per network Type 2 (b_o)	42
4.6	MSE of a one output per network Type 2 (b_1)	43
4.7	Output of sequential machine based encryption	44
4.8	Output of chaotic neural network based encryption	45
4.9	Output of chaotic neural network based decryption	46

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 various kinds of digital circuits as well as in the field of Cryptography. During our project, we have studied different neural network architectures and training algorithms. A comparative study is done between different neural network architectures for an Adder and their merits/demerits are discussed.

Using a Jordan (Recurrent network), trained by back-propagation algorithm, a finite state sequential machine was successfully implemented. The sequential machine thus obtained was used for encryption with the starting key being the key for decryption process. Cryptography was also achieved by a chaotic neural network having its weights given by a chaotic sequence.

Chapter 1

INTRODUCTION

Artificial Neural Network
Cryptography

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 <u>Biological Model</u>

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

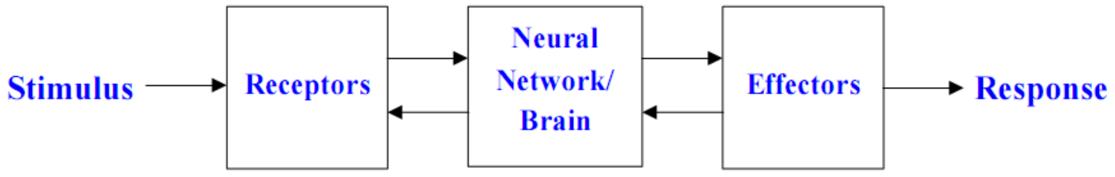


Fig 1.1: Block Diagram of a Human Nervous System.

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 neuron’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.

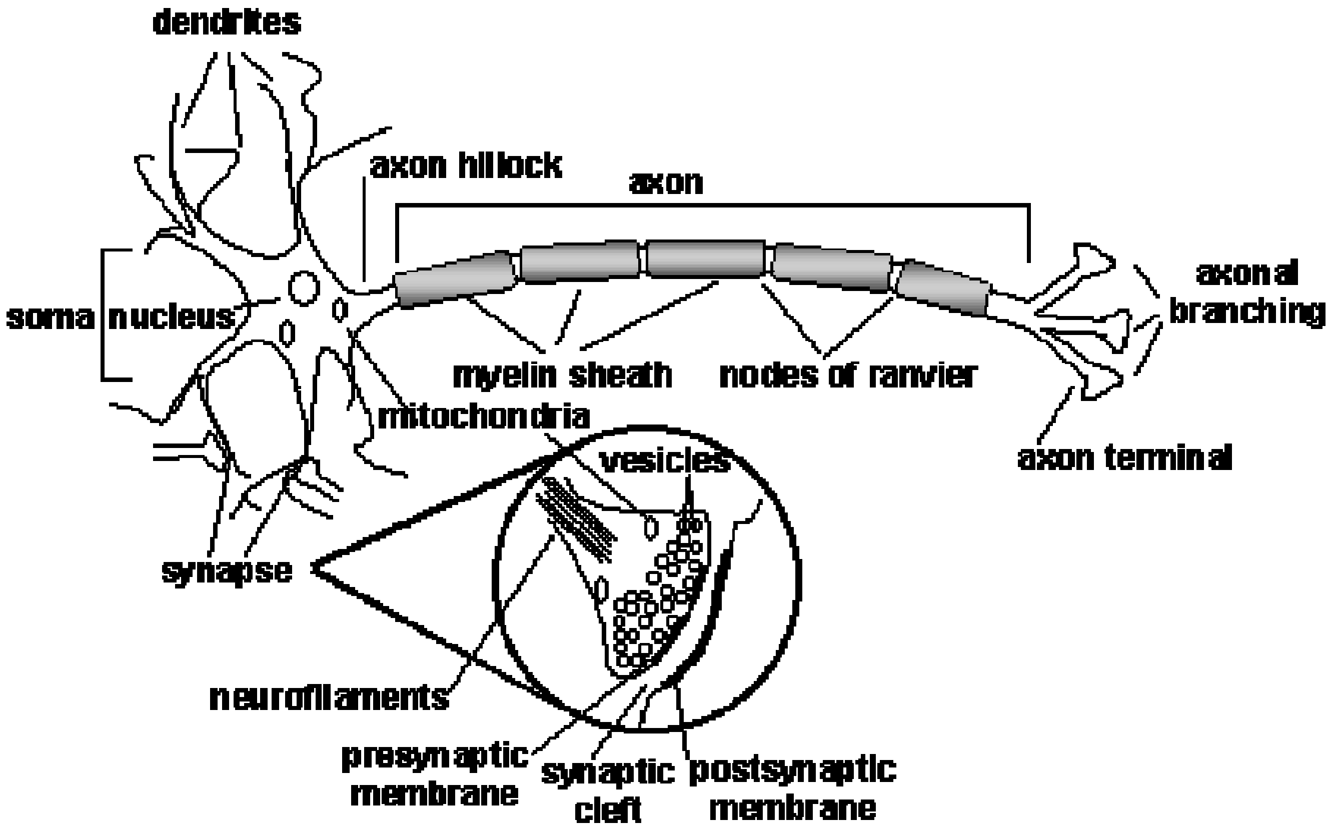


Fig 1.2: Schematic diagram of a Biological Neuron

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:

- A set of processing units ('neurons,' 'cells');
- A state of activation y_k for every unit, which equivalent to the output of the unit;
- Connections between the units. Generally each connection is defined by a weight w_{jk} which determines the effect which the signal of unit j has on unit k;
- A propagation rule, which determines the effective input s_k of a unit from its external inputs;
- An activation function F_k, which determines the new level of activation based on the effective input $s_k(t)$ and the current activation $y_k(t)$ (i.e., the update);
- An external input (aka bias, offset) θ_k for each unit;
- A method for information gathering (the learning rule);
- An environment within which the system must operate, providing input signals and if necessary error signals.

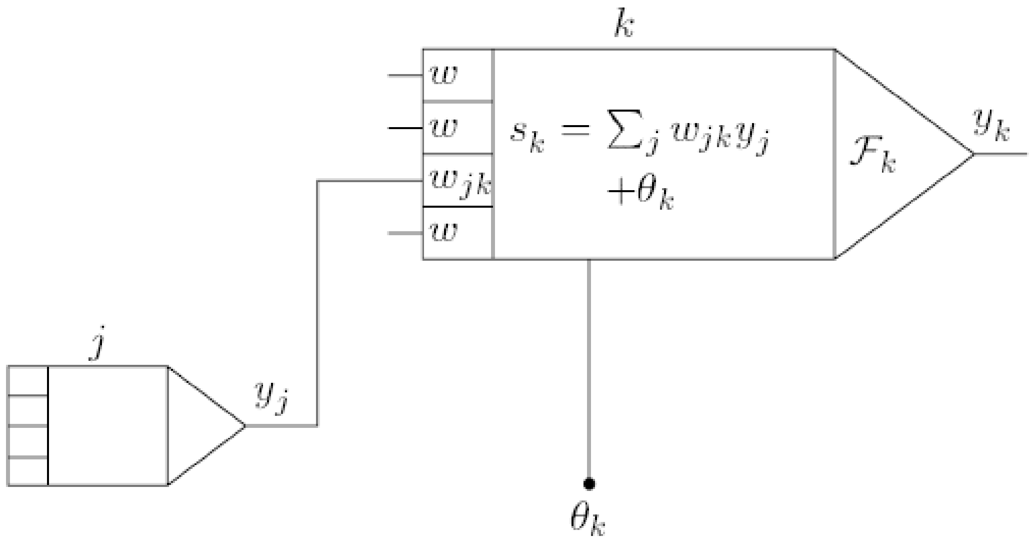


Fig 1.3: The basic components of an artificial neural network

a) Processing units

Each unit performs a relatively simple job: receive input from neighbors or external sources and use this to compute an output signal which is propagated to other units. Apart from this processing, a second task is the adjustment of the weights. The system is inherently parallel in the sense that many units can carry out their computations at the same time. Within neural systems it is useful to distinguish three types of units: input units (indicated by an index i) which receive data from outside the neural network, output units (indicated by an index o) which sends data out of the neural network, and hidden units (indicated by an index h) whose input and output signals remain within the neural network. During operation, units can be updated either synchronously or asynchronously. With synchronous updating, all units update their activation simultaneously; with asynchronous updating, each unit has a (usually fixed) probability of updating its activation at a time t, and usually only one unit will be able to do this at a time. In some cases the latter model has some advantages.

b) Connections between units

In most cases we assume that each unit provides an additive contribution to the input of the unit with which it is connected. The total input to unit k is simply the weighted sum of the separate outputs from each of the connected units plus a bias or offset term θ_k :

$$s_k(t) = \sum_j w_{jk}(t)\, y_j(t) + \theta_k(t). \tag{1.1}$$

The contribution for positive w_{jk} is considered as an excitation and for negative w_{jk} as inhibition. In some cases more complex rules for combining inputs are used, in which a distinction is made between excitatory and inhibitory inputs. We call units with propagation rule above as sigma units.

A different propagation rule, introduced by Feldman and Ballard, is known as the propagation rule for the sigma-pi unit:

$$s_k(t) = \sum_j w_{jk}(t) \prod_m y_{j_m}(t) + \theta_k(t). \tag{1.2}$$

c) Activation and the output rules

We also need a rule which gives the effect of the total input on the activation of the unit. We need a function F_k which takes the total input $s_k(t)$ and the current activation $y_k(t)$ and produces a new value of the activation of the unit k:

$$y_k(t+1) = \mathcal{F}_k(y_k(t), s_k(t)). \tag{1.3}$$

Often, the activation function is a non decreasing function of the total input of the unit:

$$y_k(t+1) = \mathcal{F}_k(s_k(t)) = \mathcal{F}_k\left(\sum_j w_{jk}(t)\, y_j(t) + \theta_k(t)\right), \tag{1.4}$$

although activation functions are not restricted to non decreasing functions. Generally, some sort of threshold function is used: a hard limiting threshold function (a sgn function), or a linear or semi-linear function, or a smoothly limiting threshold. For this smoothly limiting function often a sigmoid (S-shaped) function like

$$y_k = \mathcal{F}(s_k) = \frac{1}{1+e^{-s_k}} \tag{1.5}$$

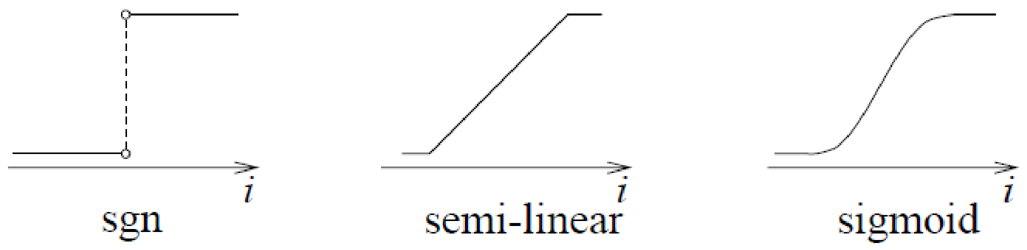


Fig 1.4: Various activation signals for a unit

In some cases, the output of a unit can be a stochastic function of the total input of the unit, the activation is not deterministically determined by the neuron input, but the neuron input determines the probability p that a neuron gets a high activation value:

$$p(y_k \leftarrow 1) = \frac{1}{1 + e^{-s_k/T}}, \tag{1.6}$$

in which T is a parameter which determines the slope of the probability function.

1.1.4 <u>Network Architectures</u>

There are three fundamental different classes of network architectures:

1) **Single-layer feed forward Networks**

   In a layered neural network the neurons are organized in the form of layers. In the simplest form of a layered network, we have an input layer of source nodes that projects onto an output layer of neurons, but not vice versa. This network is strictly a feed forward type. In single-layer network, there is only one input and one output layer. Input layer is not counted as a layer since no mathematical calculations take place at this layer.

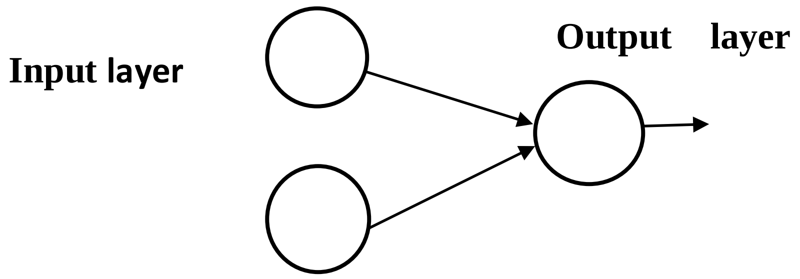


Fig 1.5: Single-layer Feed-forward Network

2) **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.

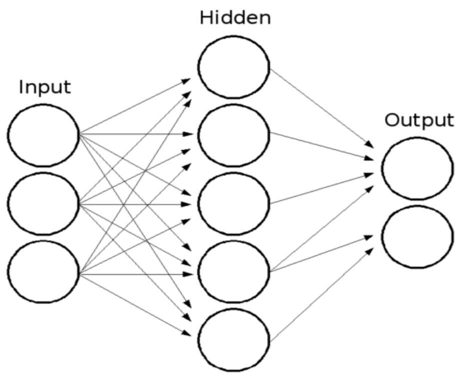


Fig 1.6: Multi-layer Feed-forward Network

3) **Recurrent networks**

A recurrent neural network has at least one feedback loop. A recurrent network may consist of a single layer of neurons with each neuron feeding its output signal back to the inputs of all the other neurons. Self-feedback refers to a situation where the output of a neuron is fed back into its own input. The presence of feedback loops has a profound impact on the learning capability of the network and on its performance.

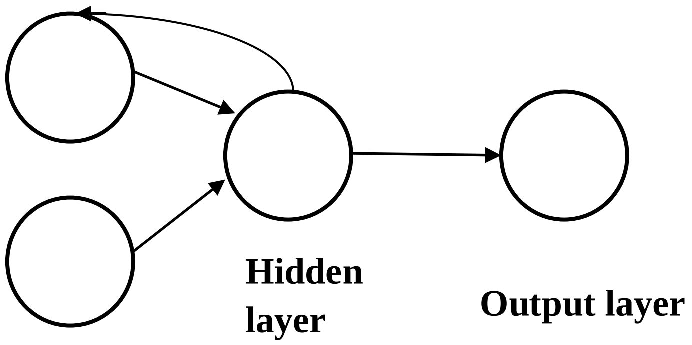


Fig 1.7: Recurrent Network

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 Multi-layer feed-forward networks

A feed-forward network has a layered structure. Each layer consists of units which receive their input from units from a layer directly below and send their output to units in a layer directly above the unit. There are no connections within a layer. The N_i inputs are fed into the first layer of $N_{h,1}$ hidden units. The input units are merely 'fan-out' units; no processing takes place in these units. The activation of a hidden unit is a function F_i of the weighted inputs plus a bias. The output of the hidden units is distributed over the next layer of $N_{h,2}$ hidden units, until the last layer of hidden units, of which the outputs are fed into a layer of N_o output units.

1.1.9 The Generalized delta rule

The following equation gives a recursive procedure for computing the δ's for all units in the network, which are then used to compute the weight changes accordingly

$$\delta_h^p = \mathcal{F}'(s_h^p) \sum_{o=1}^{N_o} \delta_o^p w_{ho}. \tag{1.7}$$

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 e_o 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.

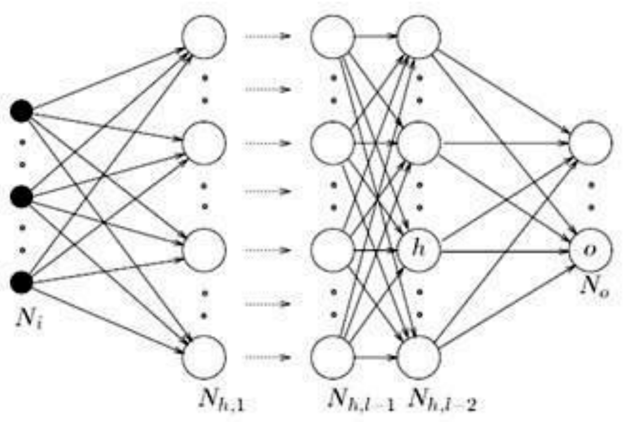


Fig 1.8: A multi-layer network with *l* layers of units

Although back-propagation can be applied to networks with any number of layers, just as for networks with binary units it has been shown that only one layer of hidden units suffices to approximate any function with finitely many discontinuities to arbitrary precision, provided the activation functions of the hidden units are non-linear. In most applications a feed-forward network with a single layer of hidden units is used with a sigmoid activation function for the 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 war-time 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*.

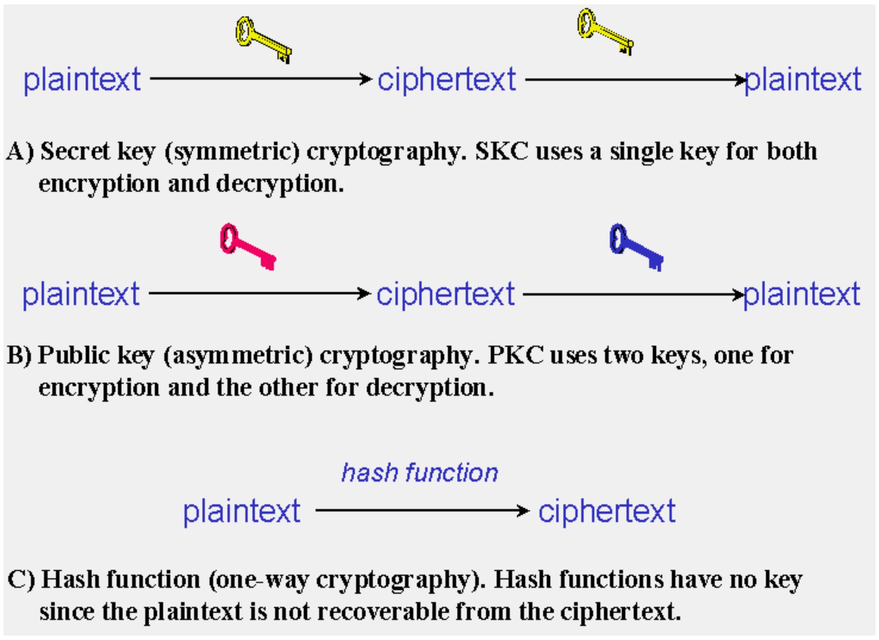

plaintext
ciphertext
plaintext
A) Secret key (symmetric) cryptography. SKC uses a single key for both encryption and decryption.
plaintext
ciphertext
plaintext
B) Public key (asymmetric) cryptography. PKC uses two keys, one for encryption and the other for decryption.
hash function
plaintext
ciphertext
C) Hash function (one-way cryptography). Hash functions have no key since the plaintext is not recoverable from the ciphertext.

Fig 1.9: Different Encryption Algorithms

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.

Chapter 2

APPLICATION OF NEURAL NETWORK

Sequential Machine & Combinational logic
Cryptography using Sequential machine
Cryptography Using Chaotic 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 whish 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.

2.1.1 The 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.

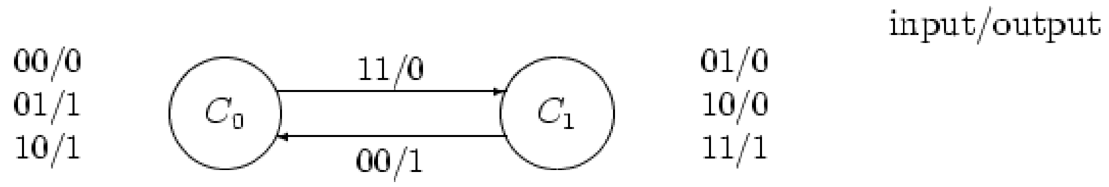


Fig 2.1: State diagram of Serial Adder

2.1.2 <u>A Sequential Detector</u>

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. The following is an example input sequence and output sequence:

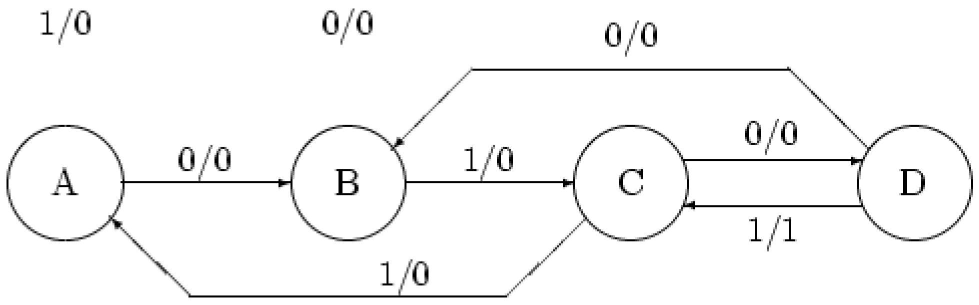
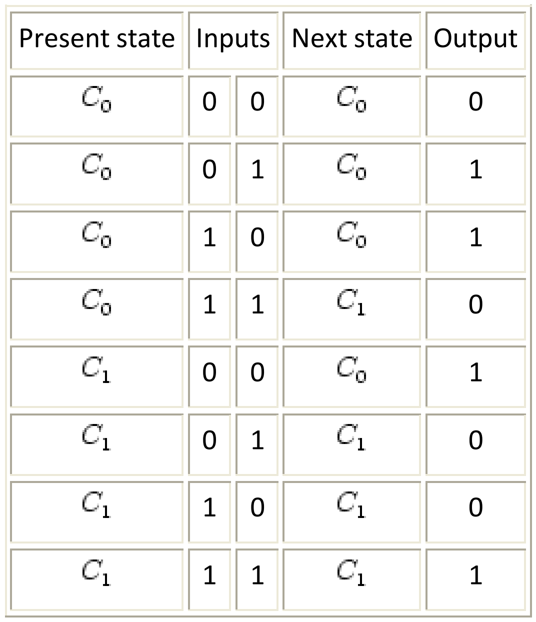


Fig 2.2: State diagram of Sequential Detector

The following is a state table corresponding to the state diagram. We can derive a state table from the state diagram as follows:



Present state	Inputs		Next state	Output
C_0	0	0	C_0	0
C_0	0	1	C_0	1
C_0	1	0	C_0	1
C_0	1	1	C_1	0
C_1	0	0	C_0	1
C_1	0	1	C_1	0
C_1	1	0	C_1	0
C_1	1	1	C_1	1

Table 2.1: State table of Sequential Detector

2.2 Combinational Logic

A combinational circuit is one for which the output value is determined solely by the values of the inputs. A combinational circuit consists of input variables, output variables, logic gates and interconnections. The interconnected logic gates accept signals from the inputs and generate signals at the output. The n input variables come from the environment of the circuit, and the m output variables are available for use by the environment. Each input and output variable exists physically as a binary signal that represents logic 1 or logic 0.

For n input variables, there are 2^n possible binary input combinations. For each binary combination of the input variables, there is one possible binary value on each output. Thus, a combinational circuit can be specified by a truth table that lists the output values for each combination of the input variables. A combinational circuit can also be described by m Boolean function, one for each output variable. Each such function is expressed as function of the n input variables.

In electronics, an **adder** is a digital circuit that performs addition of numbers. In modern computers adders reside in the arithmetic logic unit (ALU) where other operations are performed. Although adders can be constructed for many numerical representations, such as Binary-coded decimal or excess-3, the most common adders operate on binary numbers. In cases where twos complement or ones complement is being used to represent negative numbers, it is trivial to modify an adder into an adder-subtracter.

Many neural network designers are often curious about the capacity of a neural network. If they are able to know more about the capacity of neural networks, they would have an easier time deciding what neural network architecture to use as well as how many hidden neurons are needed for a neural network to perform a given function. Knowing the best architecture will save time with training, and allow for less expensive circuitry.

There are two neural network architectures considered:

(1) Multilayer, multiple outputs feed-forward.

(2) Multilayer, single output feed-forward.

We examined the advantages of both these networks and proved/disproved the fact that a single bit per output neural network uses less overall neurons to perform the same function as a multilayered network. The problem studied includes a multiple bit digital adder and a multiple bit digital multiplier.

2.3 Cryptography

In the project cryptography has been achieved by using neural network in the following manner:-

- **Using a neural network based n-state sequential machine**
- **Using a chaotic neural network**

2.3.1 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.

2.3.1 <u>Cryptography Using Chaotic Neural Network</u>

A new chaotic neural network for digital signal encryption and decryption was studied in this project. According to a binary sequence generated from a chaotic system, the biases and weights of neurons are set. The chaotic neural network can be used to encrypt digital signal. The network's features are as follows:

1) High security
2) No distortion
3) Suitable for system integration.

The MATLAB simulation results are also included for demonstration. It indicates that the integration of the proposed system and MPEG2 for TV distribution, communication, and storage is practicable.

Among the proposed encryption techniques, the basic ideas can be classified into three major types:

a. Position permutation - The position permutation algorithms scramble the positions of original data.
b. Value transformation - The value transformation algorithms transform the data value of the original signal.
c. The combining form - Finally, the *combining* form performs both operations.

The encryption scheme belongs to the category of value transformation. Based on a binary sequence generated from the 1-D logistic map, the biases and weights of neurons are set in each iteration.

Chaos

Chaos is statistically indistinguishable from randomness, and yet it is deterministic and not random at all. Chaotic system will produce the same results if given the same inputs, it is unpredictable in the sense that you can not predict in what way the system's behavior will change for any change in the input to that system. A random system will produce different results when given the same inputs.

Chaotic neural network

Chaotic neural networks offer greatly increase memory capacity. Each memory is encoded by an Unstable Periodic Orbit (UPO) on the chaotic attractor. A chaotic attractor is a set of states in a system's state space with very special property that the set is an attracting set. So the system starting with its initial condition in the appropriate basin, eventually ends up in the set. The most important, once the system is on the attractor nearby states diverge from each other exponentially fast, however small amounts of noise are amplified.

Chapter 3

IMPLEMENTATION

Sequential Machine Implementation
Combinational logic Implementation
Cryptography using Sequential machine
Cryptography Using Chaotic Neural Network

3.1 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.

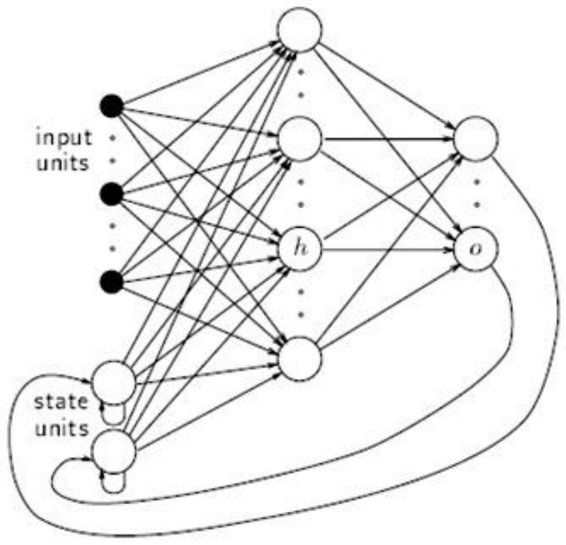


Fig 3.1: Jordan Network

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 $\mathbf{y_p^o}$ for each output unit. This output is compared with its desired value $\mathbf{d_o}$, resulting in an error signal $\boldsymbol{\delta_p^o}$ 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.

Weight adjustments with sigmoid activation function. The results from the previous section can be summarized in three equations:

a. The weight of a connection is adjusted by an amount proportional to the product of an error signal **δ**, on the unit **k** receiving the input and the output of the unit **j** sending this signal along the connection:

$$\Delta_p w_{jk} = \gamma \delta_k^p y_j^p. \tag{3.1}$$

b. If the unit is an output unit, the error signal is given by

$$\delta_o^p = (d_o^p - y_o^p)\,\mathcal{F}'(s_o^p) \tag{3.2}$$

Take as the activation function F the 'sigmoid' function as defined in chapter 1

$$y^p = \mathcal{F}(s^p) = \frac{1}{1+e^{-s^p}} \tag{3.3}$$

c. In this case the derivative is equal to:

$$\begin{aligned} \mathcal{F}'(s^p) &= \frac{\partial}{\partial s^p}\frac{1}{1+e^{-s^p}} \\ &= \frac{1}{(1+e^{-s^p})^2}(-e^{-s^p}) \\ &= \frac{1}{(1+e^{-s^p})}\frac{e^{-s^p}}{(1+e^{-s^p})} \\ &= y^p(1-y^p). \end{aligned} \tag{3.4}$$

Such that the error signal for an output unit can be written as:

$$\delta_o^p = (d_o^p - y_o^p)\,y_o^p(1-y_o^p) \tag{3.5}$$

d. The error signal for a hidden unit is determined recursively in terms of error signals of the units to which it directly connects and the weights of those connections. For the sigmoid activation function:

$$\delta_h^p = \mathcal{F}'(s_h^p)\sum_{o=1}^{N_o}\delta_o^p w_{ho} = y_h^p(1-y_h^p)\sum_{o=1}^{N_o}\delta_o^p w_{ho}. \tag{3.6}$$

e. We will get an update rule which is equivalent to the delta rule as described in the previous chapter, resulting in a gradient descent on the error surface if we make the weight changes according to:

$$\Delta_p w_{jk} = \gamma \delta_k^p y_j^p \tag{3.7}$$

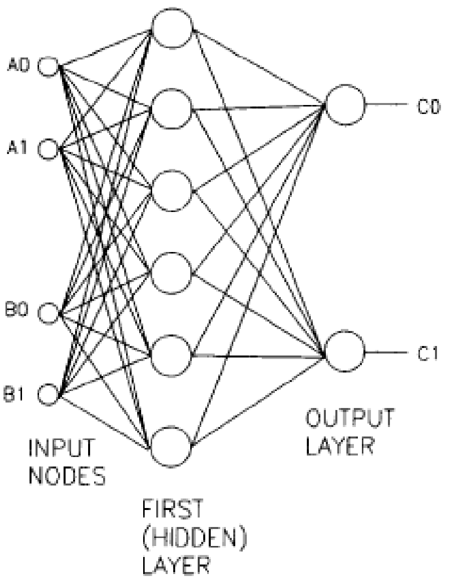
For the implementation of the sequential machine the state table is used as input and the outputs as well as next states are used as the combined output for the Jordan network. Depending upon the size of the dataset the size of the hidden layer is changed as the complexity of the sequential machine increases.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

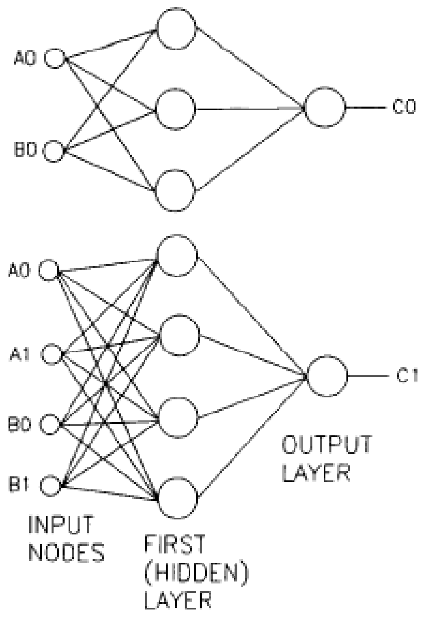
a. **Serial adder**
b. **Sequence detector**

Both of the examples can be represented by a simple state diagram given in chapter 2. The state table is made and the neural network is trained for the above stated examples of sequential logic. The weights after training obtained represents a network trained according to the sample data.

3.2 Combinational Logic

Two types of forward neural network architectures were used. The first type was a feed-forward neural network that was fully connected. The second type partitions the problem into smaller independent subtasks and a neural network is designed to implement each subtask. Both types of neural networks are shown below. *As* one can see in Figure **3.2.a** is a fully connected network that is dependent on every input to determine every output. On the other hand Figure **3.2.b** has two networks, a single network for each of the output bits, Since each neural network in Figure lb only considers one independent output, they have a smaller problem size, which allows for simpler neural networks and faster training. The fully connected neural network, type 1 has a goal to combine the training of every output bit and thus use fewer weights and fewer neurons. The goal of the type 2 neural network was that each neural network could look for a smaller number of patterns, thus reducing the training time as well as the number of neurons.





a) **b)**

Fig 3.2 Types of Networks Architectures: **a) Type 1 - Fully connected,**
b) Type 2 - One network per output.

In any optimization problem, the goal is to find the absolute minimum energy. It was decided that even though the back propagation algorithm does not always find the absolute minimum (in other words, the best solution for the problem), it would be reliable enough for this training. The algorithm was altered such that the weights only changed after every pattern had been checked in the current iteration. This was done for two purposes.

1. The first being the fact that when a number is multiplied by zero, the answer becomes zero. Thus many zero outputs in a row could cause the neural network to train so that every output would have a zero, thus impeding the proper training.
2. The second purpose was by evaluating every pattern without changing the weights until all patterns were finished, would make the program parallelize better.

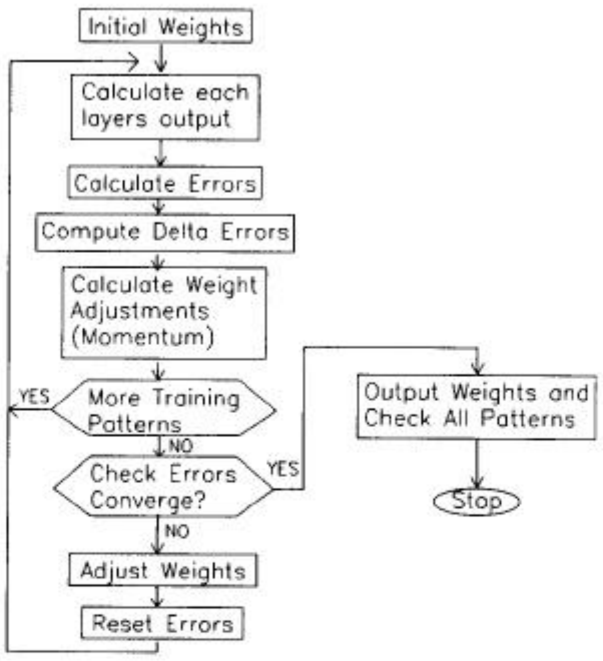


Fig 3.3: Program Flow Chart

The other features added to enhance the learning were:-

1. Random weights were used to help the network start at different places for the same network architecture. The weights were usually between 0.01 and .99**.**
2. A digital error was added such that if the output was between 0.0 and 0.2 it was considered a low and if it was between 0.7 and 1.0 it was a high. Thus a noise margin was added between 0.2 and 0.4, as with most digital circuits. This allowed the program to stop early, instead of finding a minimum error.

3.3 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. If the starting state is not known, it is not possible to retrieve the data by decryption even if the state table or the working of the sequential state is known. For training of the neural network, any type of sequential machine can be used with the key showing the complexity or the level of security obtained.

The working of a simple sequential machine for encryption is explained here. The sequential machine has *n* states and the input is an *m* bit input, as shown in the diagram. The output state can be anything according to the user. The encrypted data will depend upon the present state of the machine. Therefore, the starting state along with the input will generate an output and then the state will change according to the state table. In case of two states, if it not known whether the state is ‘0’ or ‘1’, the data cannot be decrypted and hence the starting state acts as a key.

3.4 Cryptography using a chaotic neural network

A network is called a chaotic neural network if its weights and biases are determined by a chaotic sequence. Let g denote a digital signal of length **M** and **g(n)**, $\mathbf{0 \leq M\text{-}1}$, be the one-byte value of the signal g at position n.

The Chaotic Neural Network (CNN) for Signal Encryption

Step 1: Set the value of the parameter M.

Step 2: Determine the parameter, U and the initial point x(0) of the 1-D logistic map [SI.

Step 3: Evolve the chaotic sequence x(l), x(2), ... , x(M) by **x(n+l) = µ(n)(l-x(n))**, and create b(O), b(l), ..., b(8M-1) from x(l), x(2), ..., x(M) by the generating scheme that 0.b(8m-8)b(8m-7) ….. b(8m-2)b(8m-l) … is the binary representation of x(m) for m = 1, 2,. . . ., M.

Step 4: FOR n: 0 TO (M - 1) DO

$$\boldsymbol{Let\ \ g(n)} = \sum_{i=0}^{7} \boldsymbol{d_i} \times \mathbf{2}^{\boldsymbol{i}} \tag{3.8}$$

```
    For i= 0 TO 7 DO 1
        $w_{ji} = \begin{cases} 1 & \text{if } j = i \text{ and } b(8 \times n + i) = 0, \\ -1 & \text{if } j = i \text{ and } b(8 \times n + i) = 1, \\ 0 & \text{if } j \neq i, \end{cases}$    (3.9)

    j ∈ (0,1,2,3,4,5,6,7)

        $\theta_i = \begin{cases} -1/2 & \text{if } b(8 \times n + i) = 0, \\ 1/2 & \text{if } b(8 \times n + i) = 1, \end{cases}$    (3.10)

    END

    For i= 0 TO 7 DO
        $\boldsymbol{d_i' = f(\sum_{i=0}^{7} w_{ji} \times d_i + \theta_i)}$    (3.11)
    Where f(x) is 1 if x 2 0 and 0 otherwise,

    END
        $\boldsymbol{Let\ \ g(n) = \sum_{i=0}^{7} d_i \times 2^i}$    (3.12)
    END
```

Step 5: The encrypted signal g' is obtained and the algorithm is terminated.

The decryption procedure is the same as the above one except that the input signal to the decryption CNN should be *g'(n)* and its output signal should be *g"(n).*

In case of a image, pixels are processed by neurons according to (**3.11)**. The desirable result of the encrypted image being completely disorder can be obtained. In the decryption phase of CNN, according to the same chaotic system and its initial state, i.e. the same chaotic binary sequence, the original image can be correctly obtained from decryption CNN.

Assume the encryption procedure is known except the chaotic binary sequence. When the CNN is applied to a signal of length ***M,*** it requires ***8M*** bits. The number of possible encryption results is $\mathbf{2^{8\times M}}$. Let us consider the raw data of size 65536 bytes. ***8M*** equals **524288** and all the possible results are $\mathbf{2^{52428}(\approx 10^{157810})}$**.** In the chaotic systems, it is well-known that

1) It has sensitive dependence on initial conditions
2) There exist trajectories that are dense, bounded, but neither periodic nor quasi-periodic in the state space.

Hence, the chaotic binary sequence is unpredictable. It is very difficult to decrypt an encrypted image correctly by making an exhaustive search without knowing ***x(0)*** and µ. Hence, CNN is one of guaranteed high security.

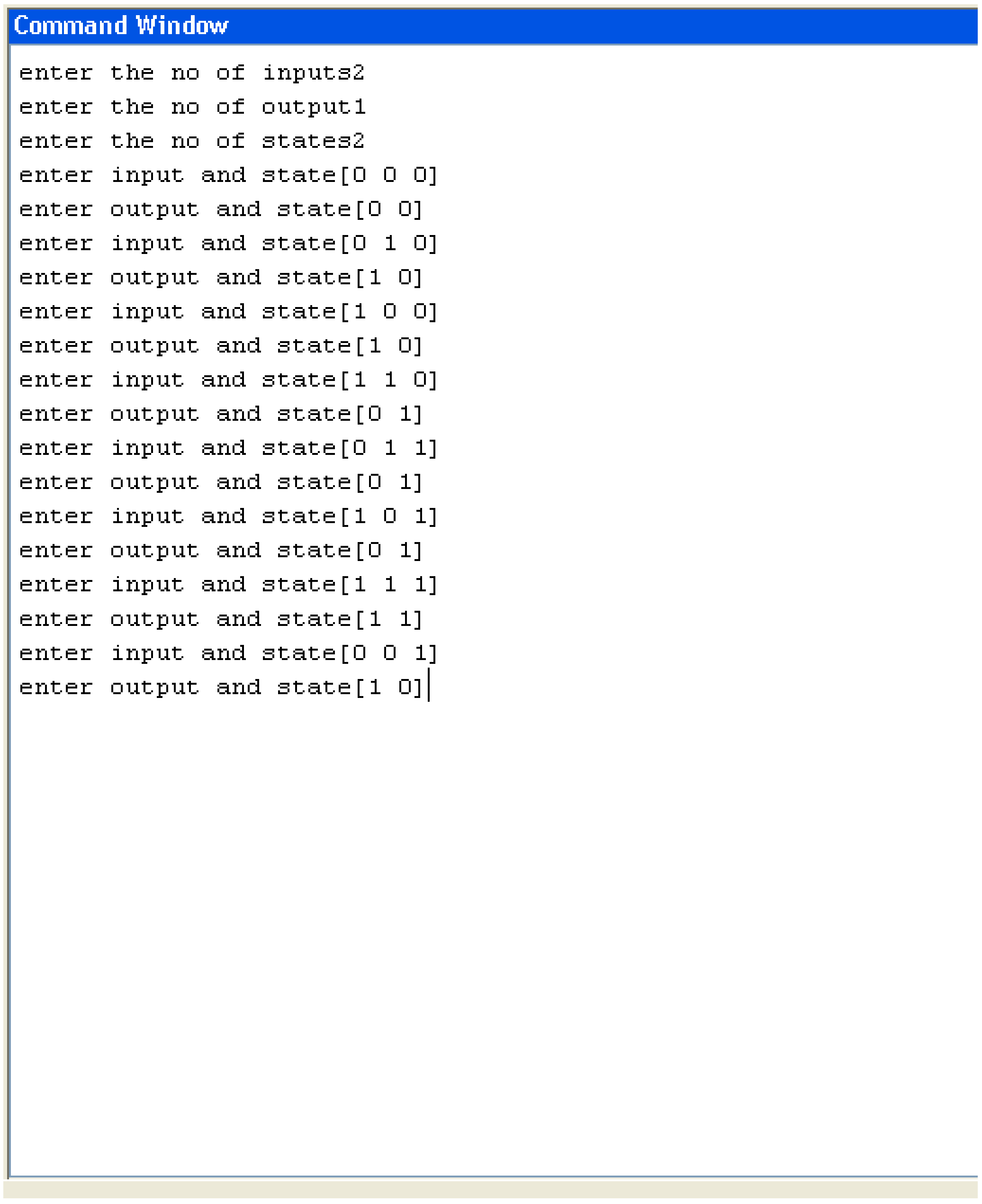
Chapter 4

RESULTS & CONCLUSION

Sequential Machine
Combinational logic Machine
Cryptography using ANN based Sequential machine
Cryptography Using Chaotic Neural Network

4.1 <u>A General n-state Sequential Machine</u>

A general n-state Sequential Machine was implemented as described in the last chapter. As an example, the serial adder was implemented using this machine. The following figures show different stages of the execution:



```
enter the no of inputs2
enter the no of output1
enter the no of states2
enter input and state[0 0 0]
enter output and state[0 0]
enter input and state[0 1 0]
enter output and state[1 0]
enter input and state[1 0 0]
enter output and state[1 0]
enter input and state[1 1 0]
enter output and state[0 1]
enter input and state[0 1 1]
enter output and state[0 1]
enter input and state[1 0 1]
enter output and state[0 1]
enter input and state[1 1 1]
enter output and state[1 1]
enter input and state[0 0 1]
enter output and state[1 0]
```

Fig 4.1: Entering the training data in the sequential machine

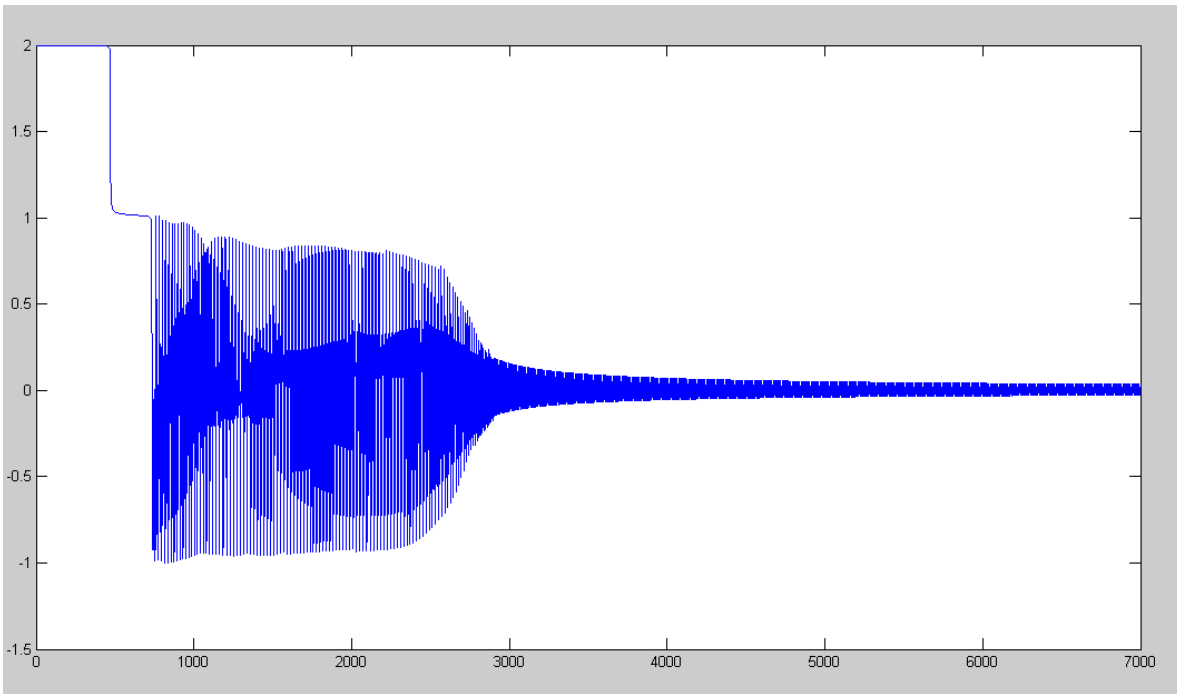
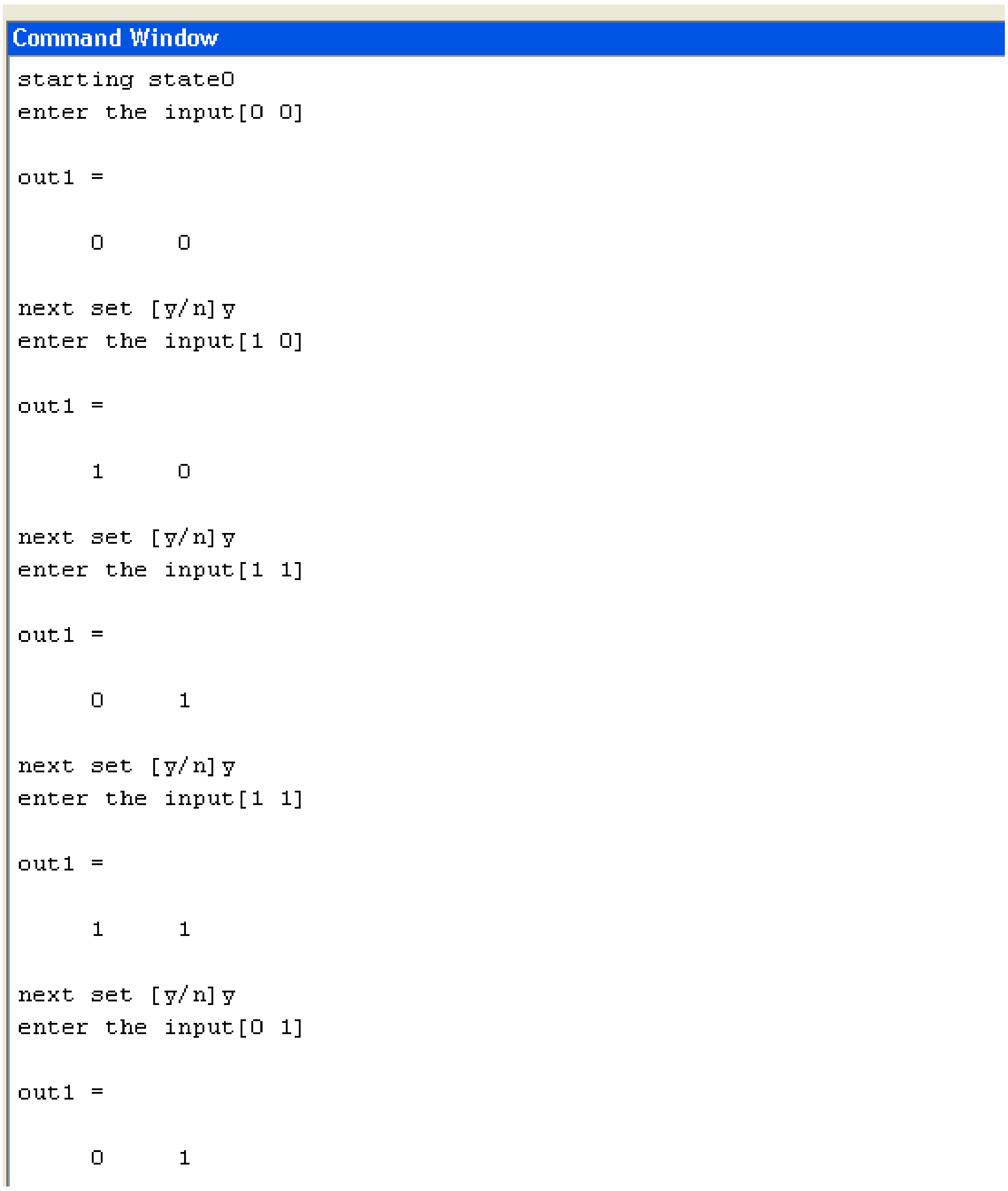


Fig 4.2: The plotted graph of the error function after the learning process

The data from the state table of the Serial Adder (Fig 4.3) is entered into the program as shown in figure 4.1. The current state represents any previous carry that might be present whereas the next state represents the output carry. Thus, this sequential machine consists of 2 input, 1 output and 2 states. After the training data has been entered into the program, the back-propagation algorithm, to minimize the error function, executes. Figure 4.2 shows the plot of the error function against the number of iterations.

Input 1	Input 2	Current state	Output	Next state
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 4.1: State table of the Serial Adder



```
starting state0
enter the input[0 0]

out1 =

     0     0

next set [y/n]y
enter the input[1 0]

out1 =

     1     0

next set [y/n]y
enter the input[1 1]

out1 =

     0     1

next set [y/n]y
enter the input[1 1]

out1 =

     1     1

next set [y/n]y
enter the input[0 1]

out1 =

     0     1
```

Fig 4.3: Output of the sequential machine implemented as a Serial Adder

Figure 4.4 shows the output of the program. An initial state is informed to the program. It asks the user for the input bits to be added. The output is the sum and the carry bit. The program automatically jumps to the new carry state. The user can keep on giving the input bits to be added and the program generates the output based on the previous carry state.

4.2 Combinational Logic Machine

As discussed in section **3.2** a comparative study is done between type 1 and type 2 architectures and the size of neural network required for the design of adder circuit.

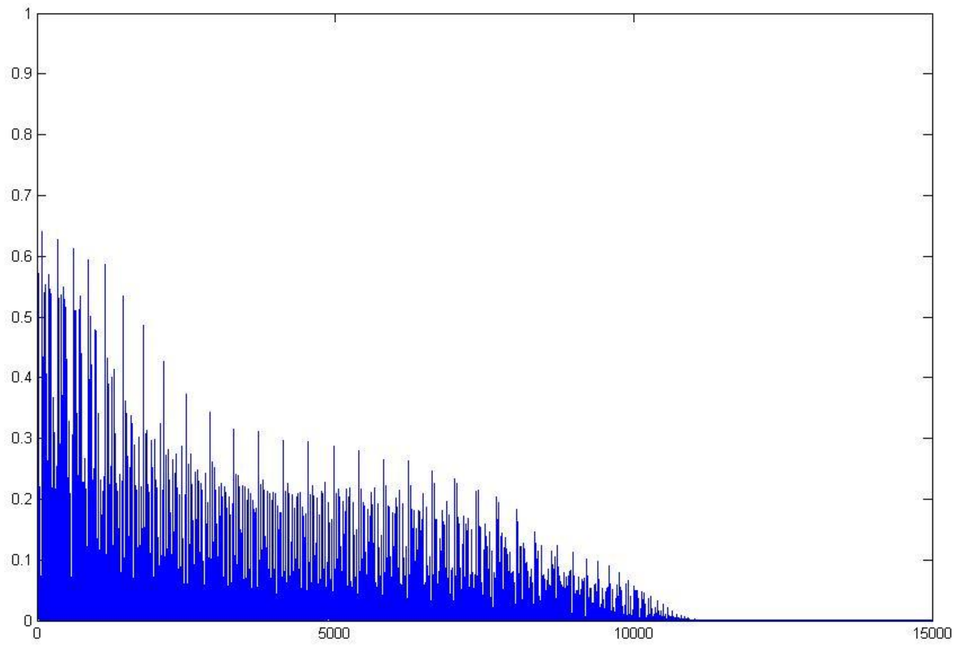


Fig 4.4: MSE of a fully connected neural network Type 1

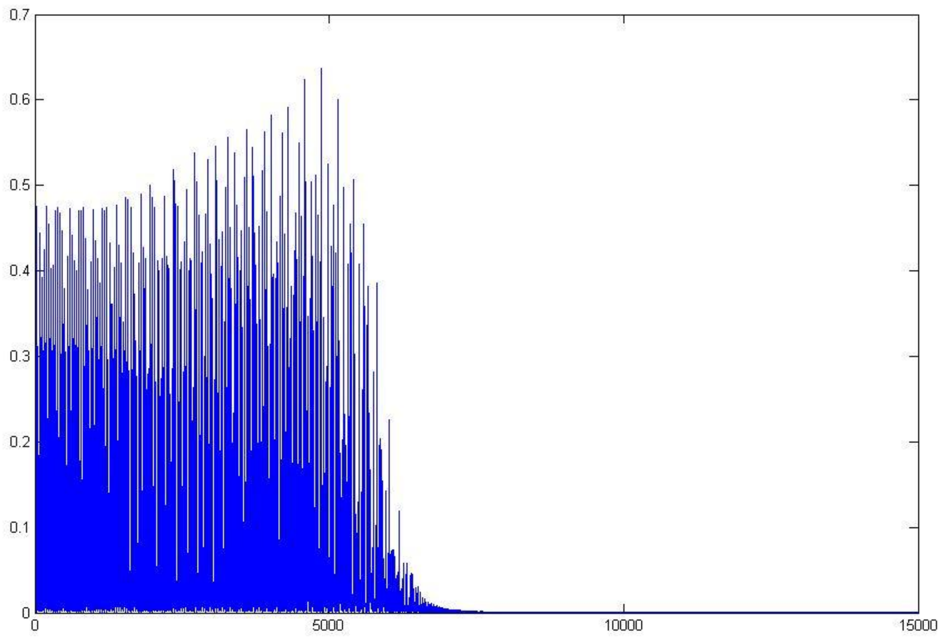


Fig 4.5: MSE of a one output per network Type 2(b_0)

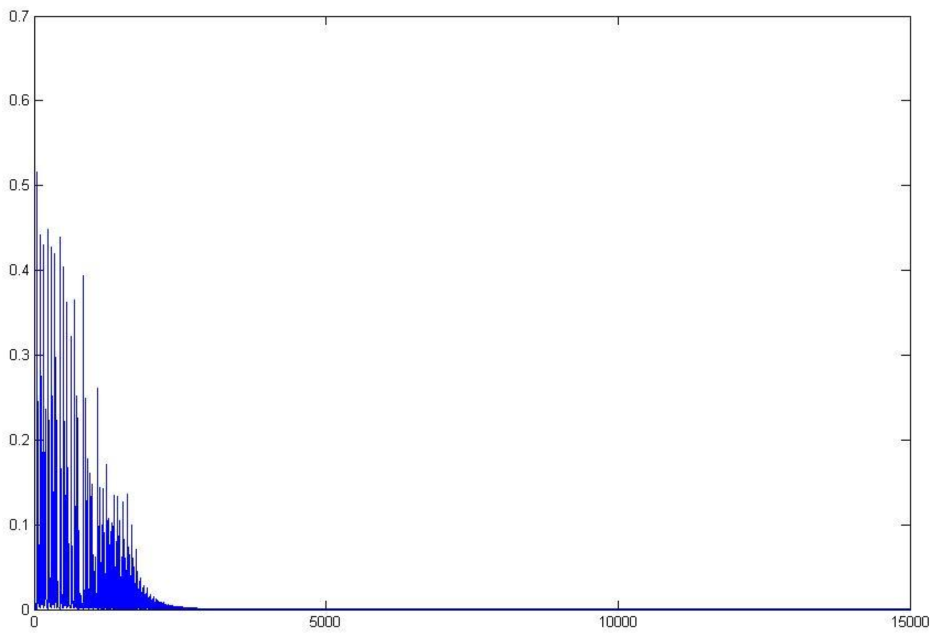


Fig 4.6: MSE of a one output per network Type 2(b_1)

It can clearly be that the type 2 neural network look for a smaller number of patterns and thus reduced the training time as well as the number of neurons.

No of bits	Adder	Adder
1×1	2	3
2×2	3	6
3×3	7	11

Table 4.2: Hidden layer neurons and total neurons for the Type 1 network

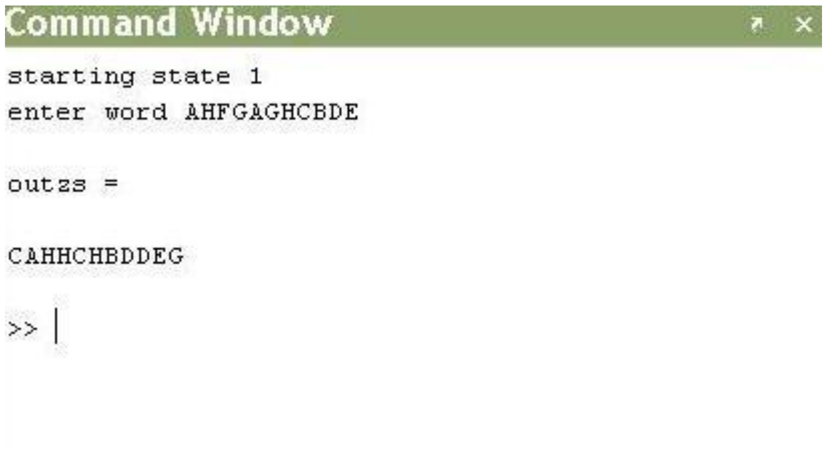
Adder	Bit1	Bit2	Bit3	Bit4	Hidden	Total
1×1	2	1	N/A	N/A	3	5
2×2	2	2	1	N/A	5	8
3×3	2	2	4	1	9	13

Table 4.3: Hidden neurons per bit for the individually connected network Type 2

The fully connected neural network, type 1 has a goal to combine the training of every output bit and thus use fewer weights and neurons.

4.3 Cryptography using ANN based Sequential Machine

A 3-bit encryption machine was successfully built using an ANN based sequential machine. The sequential machine used had 2 states (‘0’ and ‘1’) and the input is the 3-bit data to be encrypted. Letters ‘A’ to ‘H’ were used to represent all the possible 3-bit inputs. If the state is ‘0’, the input letter is shifted by one to generate the encrypted letter while if the state is ‘1’, the letter is shifted by 2. During this operation, the state is automatically switched. Thus, if the starting state is ‘0’ and the input is ‘A’, the output will be ‘B’ and the state switches to ‘1’. If the next input is again ‘A’, the output will be ‘C’ as the current state now is ‘1’. For ‘H’, state ‘0’ will flip the letter to ‘A’ while state ‘1’ will flip the output to ‘B’. This method can be used to encrypt a word containing only the letters ‘A’ to ‘H’.



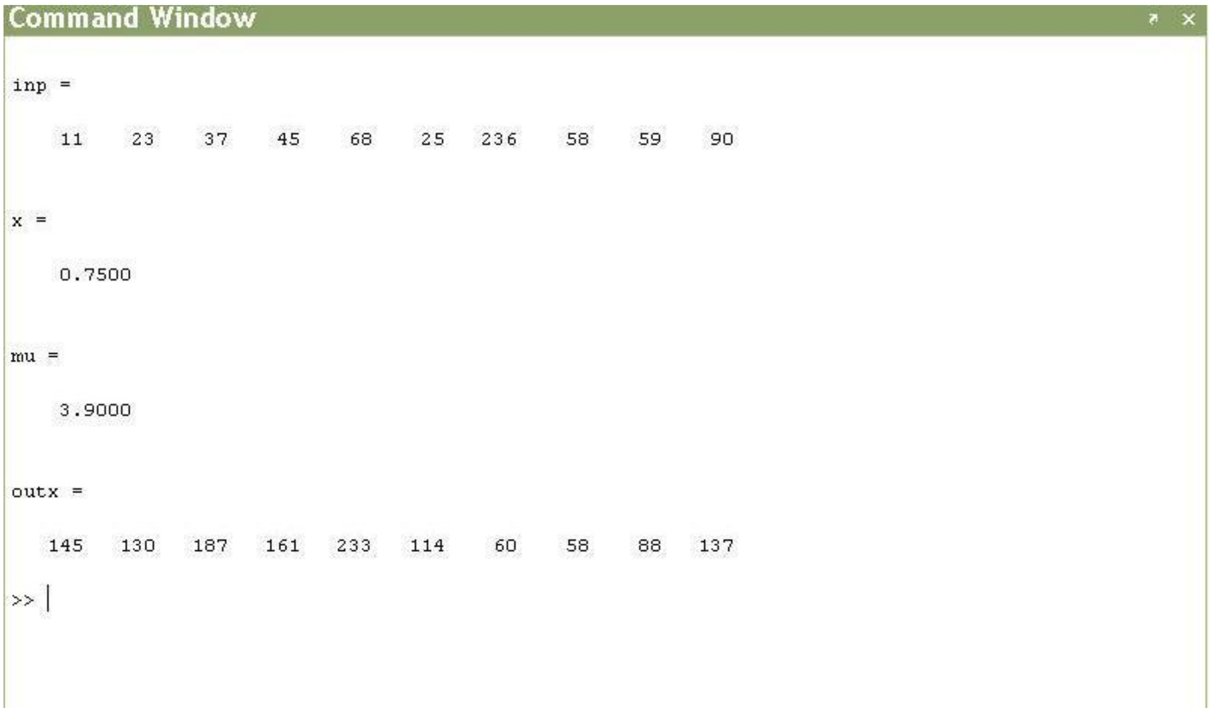
```
starting state 1
enter word AHFGAGHCBDE

outzs =

CAHHCHBDDEG

>> |
```

Fig 4.7: Output using sequential machine based Encryption

Figure 4.6 shows the implementation of the above method. The word “AHFGAGHCBDE” is to be encrypted using the starting state of ‘1’. The output in this case will be “CAHHCHBDDEG”.

4.4 Cryptography Using Chaotic Neural Network

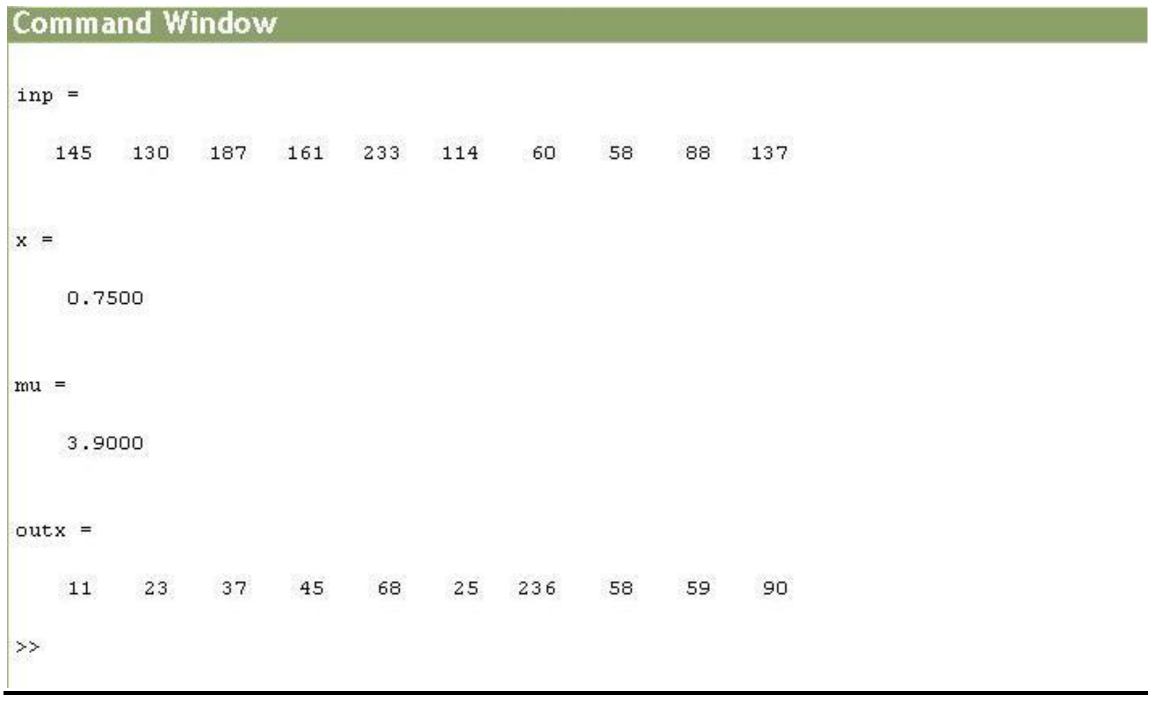
A chaotic network is a neural network whose weights depend on a chaotic sequence. The chaotic sequence highly depends upon the initial conditions and the parameters, **x(0)** = 0.75, and ,µ = 3.9 are set. It is very difficult to decrypt an encrypted data correctly by making an exhaustive search without knowing **x(0)** and µ.



```
Command Window

inp =

    11    23    37    45    68    25   236    58    59    90


x =

    0.7500


mu =

    3.9000


outx =

   145   130   187   161   233   114    60    58    88   137

>> |
```

Fig 4.8: Output using chaotic neural network based Encryption

Here a sequence of ten numbers is used for encryption and the initial parameters for the chaotic network are used as mentioned. The output or the encrypted data is then used for decryption. It can easily be seen that the output is in a chaotic state.



```
inp =

   145   130   187   161   233   114    60    58    88   137


x =

    0.7500


mu =

    3.9000


outx =

    11    23    37    45    68    25   236    58    59    90

>>
```

Fig 4.8: Output sequence using chaotic neural network based Decryption

Depending upon the chaotic sequence a weight matrix and a bias matrix is obtained and the net input is obtained. Then a hard limiter is applied as a transfer function in order to obtain the digital encrypted data. For decryption the same network is used and the same initial value is used to generate the chaotic sequence and for decrypting the data successfully. The CNN here used can also be used to encrypt and decrypt data such as images by converting it into a single vector. Hence, **CNN** is one of guaranteed high security.

4.5 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 ha 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 two methods. A sequential machine based method for encryption of data is designed. Also, a chaotic neural network for digital signal cryptography is analyzed. 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.

APPENDIX-1

Matlab Code

a. Sequential machine

```
clc;
clear all;
close all;

inputs_x=input('enter the no of inputs');
output_x=input('enter the no of output');
statx=input('enter the no of states');

% no of bits for states
for tem=1:100
if 2^tem >=statx
states=tem;
break
end
end

hls=5;%2^(inputs_x+output_x+2); % size of the hidden layer
% weight initialization----------------------
w1=rand(hls,(inputs_x+states+1));
w2=rand((output_x+states),hls+1);
neta=1;
a=1;
sx=0;
p=[];

for tex=1:(2^(inputs_x)*statx);
   training_setx(tex,:)=input('enter input and state');
   training_outx(tex,:)=input('enter output and state');
end

for x=1:7000

   training_set=training_setx(a,:);
   training_out=training_outx(a,:);
% output hidden layer---------------------
   inputu=[1 training_set];
   sum_h=(w1*(inputu)')';
   o_h=1./(1+exp(-sum_h));

% output layer ---------------------------
   input_h=[1 o_h];
   sum_out=(w2*(input_h)')';
   out=1./(1+exp(-sum_out));


% delta -----------------------------------
   delta_out=(out.*(1-out)).*(training_out - out);

   delta_h=(delta_out*w2).*input_h.*(1-input_h);


 % update of weight --------------------------
 % output layer --------------------------------

    for t=1:(output_x+states)
        w2(t,:) = w2(t,:) + neta*delta_out(t)*input_h;
    end

 % hidden layer ----------------------------------

     for t=1:hls
         w1(t,:) = w1(t,:) + neta*delta_h(t+1)*inputu;
     end


    for t=1:(output_x+states)

        if out(t)>=0.7
             out1(t)=1;
        elseif out(t)<=0.2
            out1(t)=0;
        else out1(t)=out(t);
        end

    end
   p=[p sum(out-training_out)];

    if out1==training_out
        a=a+1;
        sx=sx+1;
    end

    if a > ((2^inputs_x)*statx)
        a=a-((2^inputs_x)*statx);
    end
end
plot(p);


% testing the program

statx=zeros(1,states);

zaz='y';

for xz=1:10

    ipo = input('enter the input');
    ipox=[ipo statx];

    inputp=[1 ipox];
    sum_h=(w1*(inputp)')';
    o_h=1./(1+exp(-sum_h));

    % output layer ---------------------------
    input_h=[1 o_h];
    sum_out=(w2*(input_h)')';
    out=1./(1+exp(-sum_out));

    for t=1:(states+output_x)
```

```
        if out(t)>=0.7
             out1(t)=1;
        elseif out(t)<=0.2
            out1(t)=0;
        else out1(t)=out(t);
        end

    end

  out1

statx=out1( (output_x + 1):(output_x+states));
end
```

b. Multilayer single output feed-forward Adder

```
clc;
clear all;
close all;

inputs_x=input('enter the no of inputs');

training_setxm=[];
training_outxm=[];
for c1=0:1
    for inx1=0:2^(inputs_x)-1
        for inx2=0:2^(inputs_x)-1
            temp=inx1+inx2+c1;
            training_outxm=[training_outxm ; bitget(temp,inputs_x+1:-1:1)];
            training_setxm=[training_setxm ; bitget(inx1,inputs_x:-1:1) bitget(inx2,inputs_x:-1:1) c1];
        end
    end
end


training_setx=[];
training_outx=[];
hls=[4 3 1];% size of the hidden layer

% weight initialization---------------------

for v1=1:inputs_x+1

    if v1 == inputs_x+1
        w1=rand(hls(v1),2*(v1-1)+2);
    else
        w1=rand(hls(v1),2*v1+2);
    end
    w2=rand(1,hls(v1)+1);
    neta=0.5;
    a=1;
    sx=0;
    p=[];

    if v1 == inputs_x+1
        training_setx=training_setxm;
    else
        training_setx=[training_setxm(:,inputs_x-v1+1:inputs_x) training_setxm(:,2*inputs_x-
v1+1:2*inputs_x+1)];
    end
```

```
training_outx=[training_outxm(:,(inputs_x+1)-v1+1)];


for x=1:15000

    training_set=training_setx(a,:);
    training_out=training_outx(a,:);
% output hidden layer---------------------
    inputu=[1 training_set];
    sum_h=(w1*(inputu)')';
    o_h=1./(1+exp(-sum_h));

% output layer ---------------------------
    input_h=[1 o_h];
    sum_out=(w2*(input_h)')';
    out=1./(1+exp(-sum_out));


% delta -----------------------------------
    delta_out=(out.*(1-out)).*(training_out - out);

    delta_h=(delta_out*w2).*input_h.*(1-input_h);


 % update of weight -------------------------

 % output layer -------------------------------


        w2 = w2 + neta*delta_out*input_h;
 % hidden layer ---------------------------------

     for t=1:hls(v1)
         w1(t,:) = w1(t,:) + neta*delta_h(t+1)*inputu;
     end


    for t=1:1

        if out(t)>=0.7
             out1(t)=1;
        elseif out(t)<=0.2
            out1(t)=0;
        else out1(t)=out(t);
        end

    end
   p=[p mean((out-training_out).^2)];

    if out1==training_out
        a=a+1;
        sx=sx+1;
    end

    if a > ((2^(inputs_x))^2)*2
        a=a-((2^(inputs_x))^2)*2;
    end
    end
    figure;
    plot(p.*p);
end
```

c. Multilayer multiple output feed-forward Adder

```
clc;
clear all;
close all;

inputs_x=input('enter the no of inputs');

training_setx=[];
training_outx=[];
for c1=0:1
    for inx1=0:2^(inputs_x)-1
        for inx2=0:2^(inputs_x)-1
            temp=inx1+inx2+c1;
            training_outx=[training_outx ; bitget(temp,inputs_x+1:-1:1)];
            training_setx=[training_setx ; bitget(inx1,inputs_x:-1:1) bitget(inx2,inputs_x:-1:1) c1];
        end
    end
end



hls=12;% size of the hidden layer

% weight initialization----------------------
w1=rand(hls,2*inputs_x+2);
w2=rand(inputs_x+1,hls+1);
neta=0.5;
a=1;
sx=0;
p=[];

for x=1:15000

    training_set=training_setx(a,:);
    training_out=training_outx(a,:);
% output hidden layer---------------------
    inputu=[1 training_set];
    sum_h=(w1*(inputu)')';
    o_h=1./(1+exp(-sum_h));

% output layer ---------------------------
    input_h=[1 o_h];
    sum_out=(w2*(input_h)')';
    out=1./(1+exp(-sum_out));


% delta -----------------------------------
    delta_out=(out.*(1-out)).*(training_out - out);

    delta_h=(delta_out*w2).*input_h.*(1-input_h);
```

```
 % update of weight --------------------------

 % output layer --------------------------------

   for t=1:(inputs_x+1)
      w2(t,:) = w2(t,:) + neta*delta_out(t)*input_h;
   end

 % hidden layer ----------------------------------

    for t=1:hls
       w1(t,:) = w1(t,:) + neta*delta_h(t+1)*inputu;
    end


   for t=1:inputs_x+1

      if out(t)>=0.7
          out1(t)=1;
      elseif out(t)<=0.2
         out1(t)=0;
      else out1(t)=out(t);
      end

   end
  p=[p mean((out-training_out).^2)];

   if out1==training_out
      a=a+1;
      sx=sx+1;
   end

   if a > ((2^(inputs_x))^2)*2
      a=a-((2^(inputs_x))^2)*2;
   end

end
plot(p.*p);
```

d. Encryption using ANN based sequential machine

```
clc;
clear all;
close all;

inputs_x=3;     %------------------------no. of bits of data
output_x=3;
statx=2; %------------------no. of states

% ------------------------  no of bits for states

for tem=1:100
if 2^tem >=statx
states=tem;
break
end
end
```

```
hls=6;        % size of the hidden layer

% --------------------------  weight initialization
w1=rand(hls,(inputs_x+states+1));
w2=rand((output_x+states),hls+1);
neta=1;
a=1;
sx=0;
p=[];

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

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

for x=1:10000

    training_set=training_setx(a,:);
    training_out=training_outx(a,:);
% ---------------------------  output hidden layer
    inputu=[1 training_set];
    sum_h=(w1*(inputu)')';
    o_h=1./(1+exp(-sum_h));

% ---------------------------  output layer
    input_h=[1 o_h];
    sum_out=(w2*(input_h)')';
    out=1./(1+exp(-sum_out));

% ----------------------------  delta
    delta_out=(out.*(1-out)).*(training_out - out);

    delta_h=(delta_out*w2).*input_h.*(1-input_h);

     % update of weight --------------------------

 % output layer --------------------------------

    for t=1:(output_x+states)
        w2(t,:) = w2(t,:) + neta*delta_out(t)*input_h;
    end

 % hidden layer ---------------------------------

     for t=1:hls
         w1(t,:) = w1(t,:) + neta*delta_h(t+1)*inputu;
     end


    for t=1:(output_x+states)

       if out(t)>=0.7
            out1(t)=1;
       elseif out(t)<=0.2
           out1(t)=0;
       else out1(t)=out(t);
       end
```

```
   end
   p=[p sum(out-training_out)];

    if out1==training_out
        a=a+1;
        sx=sx+1;
    end

    if a > ((2^inputs_x)*statx)
        a=a-((2^inputs_x)*statx);
    end

end
plot(p.*p);

% testing the program

statx=input('starting state ');

    ipo = input('enter word ','s');
    finp=[];
    for i=1:length(ipo)
    b=ipo(i);
        switch b
        case('A')
            set=[0 0 0];
        case('B')
            set=[0 0 1];
        case('C')
            set=[0 1 0];
        case('D')
            set=[0 1 1];
        case('E')
            set=[1 0 0];
        case('F')
            set=[1 0 1];
        case('G')
            set=[1 1 0];
        case('H')
            set=[1 1 1];
        end
    ipox=[set statx];
    inputp=[1 ipox];
    sum_h=(w1*(inputp)')';
    o_h=1./(1+exp(-sum_h));

    % output layer ---------------------------
    input_h=[1 o_h];
    sum_out=(w2*(input_h)')';
    out=1./(1+exp(-sum_out));

    for t=1:(states+output_x)

        if out(t)>=0.7
             out1(t)=1;
        elseif out(t)<=0.2
            out1(t)=0;
        else out1(t)=out(t);
        end
```

```
      end
     finp=[finp;out1];
     tempch=[];

     statx=out1( (output_x + 1):(output_x+states));

   end
   outzs=[''];
   for f=1:length(ipo)
      tempch=finp(f,:);
      tempch=tempch(1:3);

      if tempch==[0 0 0]
         outzs=[outzs 'A'];
      end
      if tempch==[0 0 1]
         outzs=[outzs 'B'];
      end
      if tempch==[0 1 0]
         outzs=[outzs 'C'];
      end
      if tempch==[0 1 1]
         outzs=[outzs 'D'];
      end
      if tempch==[1 0 0]
         outzs=[outzs 'E'];
      end
      if tempch==[1 0 1]
         outzs=[outzs 'F'];
      end
      if tempch==[1 1 0]
         outzs=[outzs 'G'];
      end
      if tempch==[1 1 1]
         outzs=[outzs 'H'];
      end
   end
   outzs
```

e. Encryption using chaos neural network

```
clear all;
close all;
clc;
% input -----------------------------------------------------------------
inp=[11 23 37 45 68 25 236 58 59 90]

for i=1:length(inp)
   inpx(i,:)=bitget(inp(i),8:-1:1);
end

% generating a chaotic sequence ------------------------------------------

l=length(inp);
x(1)=0.75
mu=3.9

for  i=2:l
    x(i)=mu*x(i-1)*(1-x(i-1));
```

```
end

x=uint8(((x-min(x))/max(x))*255);
b=[];

for i=1:l
    b(i,:)= bitget(x(i),8:-1:1);
end

temp=0;
outx=zeros(1,length(inp));
% network -----------------------------------------------------------------
for c=1:length(inp)

    for i=1:8

        for j=1:8

            if (b(c,i)==0)&(i==j)
                weight(i,j)=1;
            elseif (b(c,i)==1)&(i==j)
                weight(i,j)=-1;
            elseif i~=j
                weight(i,j)=0;
            end
        end

        if (b(c,i)==0)
            theta(i)=-1/2;
        else theta(i)=1/2;
        end
    end
    for i=1:8

        dx(c,i) = hardlim(sum(weight(i,:).*inpx(c,:))+theta(i));

    end
    for i=1:8

        outx(c)=outx(c)+uint8(dx(c,i))*(2^(8-i));
    end
end

    outx
```

f. Decryption using chaos neural network

```
clear all;
close all;
clc;
% input -------------------------------------------------------------------
inp=[145   130   187   161   233   114    60    58    88   137]

for i=1:length(inp)
    inpx(i,:)=bitget(inp(i),8:-1:1);
end

% generating a chaotic sequence ------------------------------------------
```

```
l=length(inp);
x(1)=0.75
mu=3.9

for  i=2:l
    x(i)=mu*x(i-1)*(1-x(i-1));
end

x=uint8(((x-min(x))/max(x))*255);
b=[];

for i=1:l
    b(i,:)= bitget(x(i),8:-1:1);
end
temp=0;
outx=zeros(1,length(inp));
% network ------------------------------------------------------------------
for c=1:length(inp)

    for i=1:8

        for j=1:8

            if (b(c,i)==0)&(i==j)
                weight(i,j)=1;
            elseif (b(c,i)==1)&(i==j)
                weight(i,j)=-1;
            elseif i~=j
                weight(i,j)=0;
            end
        end

        if (b(c,i)==0)
            theta(i)=-1/2;
        else theta(i)=1/2;
        end

    end
    for i=1:8

        dx(c,i) = hardlim(sum(weight(i,:).*inpx(c,:))+theta(i));

    end
    for i=1:8

        outx(c)=outx(c)+uint8(dx(c,i))*(2^(8-i));
    end

end

    outx
```

APPENDIX-2

References

[I] M. E. Smid and D. K. Branstad, "The Data Encryption Standard: Past and Future," Proceedings of The IEEE, vol. 76, no. 5, pp. 550-559, 1988.

[2] C. Boyd, "Modem Data Encryption," Electronics & Communication Journal, pp. 271-278, Oct. 1993. 131 N. Bourbakis and C. Alexopoulos, "Picture Data Encryption Using SC4N Pattern," Pattern Recognition, vol. 25, no. 6, pp. 567-581, 1992.

[4] J. C. Yen and J. I. GUO, "A New Image Encryption Algorithm and Its VLSI Architecture," 1999 IEEE Workshop on Signal Procs. Systems, Grand Hotel, Taipei, Taiwan, Oct. 18-22, pp. 430-437, 1999.

[5] C. J. Kuo and M. S. Chen, "A New Signal Encryption Technique and Its Attack Study," IEEE International Conference on Security Technology, Taipei, Taiwan,

[6] C. W. Wu and N. F. Rulkov, "Studying chaos via 1-D maps - A tutorial," IEEE Trans. on Circuits and Systems I-Fundamental Theory and Applications, vol. 40, no. 10, pp. 707-721, 1993.

[7] T. S. Parker and L. 0. Chua, "Chaos - A tutorial for engineers," IEEE Proc., vol. 75, pp. 982-1008, 1987.

[8]Haykin, Simon. Neural Networks, A Comprehensive Foundation. MacMillin College Publishing CO, New York. 1994.

[9]Lansner, Anders and Ekeberg, Orjan. An Associative Network Solving the 4-Bit ADDER Problem". UCNN International Joint Conference on Neural Networks, Vol2, 1987.

[10]Rumelhart, D.E, Hinton, G. E., and Williams, RJ. Learning internal representations by error propagation. In Parallel Distributed Processing, Vol. 1, MlT Press, Cambridge.

[11]Hebb, D. 0. Organization ofBehavior. New York : Science Editions.

[12]"An Introduction to Neural network" by Ben Krose and Patrick van der Smagt Eighth editionNovember 1996

[13]Zurada, Jacek M. Introduction to Artificial Systems. West Publishing CO, St. Paul. 1992.

[14]Wasserman, Philip D. Neural Computing, Theory and Practice. Van Nordstrand Reinhold, New York. 1989.

[15]"Machine Learning, Neural and Statistical Classification" by D. Michie, D.J. Spiegelhalter, C.C. Taylor February 17, 1994

[16] “Design and Realization of A New Chaotic Neural Encryption/Decryption Network” by Scott Su, Alvin Lin, and Jui-Cheng Yen.

[17] “Capacity of Several Neural Networks With Respect to Digital Adder and Multiplier” by Daniel C. Biederman and Esther Ososanya

[18]"Artificial Intelligence A Modern Approach" by Stuart J. Russell and Peter Norvig.

[19] http://www.garykessler.net/library/crypto.html

[20] Digital design, fourth edition by M. Morris Mano and Michael D. Ciletti

[21] http://www.wikipedia.org