A Project Report

On

Artificial Neural Networks

By

Nalluri Ravi Teja-17XJ1A0528

Surineela Sai Vijay Kumar-17XJ1A0344

Viswa Janith Paidisetty-17XJ1A0565

Sathvik Nimmaneni-17XJ1A0531

Under the supervision of

Prof.G.Rama Murthy

SUBMITTED IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS OF

PR 301: PROJECT TYPE COURSE

Acknowledgements

We would like to share our sincere gratitude to all those who helped us in the project. Firstly, we would like to thank Prof.G.Rama Murthy for letting us do this project and for his continuous guidance throughout the project. Because of his guidance and governance, our whole team was able to learn the minute aspects of the project work. Secondly, we team members would like to thank each other for being co-operative to each other and made the team work so well.

We would like to thank other teachers who indirectly helped us in the project by teaching us the programming languages we used in the project. Finally, we would like to thank the Mahindra Ecole Centrale for providing us with such an opportunity to learn from these experiences.

Thank You All.

Nalluri Ravi Teja-17XJ1A0528

Surineela Sai Vijay Kumar-17XJ1A0344

Viswa Janith Paidisetty-17XJ1A0565

Sathvik Nimmaneni-17XJ1A0531

Mahindra Ecole Centrale

Hyderabad

Certificate

This is to certify that the project report entitled "**Artificial Neural Networks**" submitted by Mr/Ms. _____ (HT No. _____) in partial fulfilment of the requirements of the course PR 301, Project Course, embodies the work done by him/her under my supervision and guidance.

Prof.G.Rama Murthy

Mahindra Ecole Centrale, Hyderabad.

Date:

Abstract

Neural networks are one of the most powerful and widely used algorithms in deep learning a subfield of Machine Learning, which evolved from the idea of simulating the human brain. At first, neural networks may seem a like black box; with an input layer getting the data into the "hidden layers" where something happens to the input data and finally, we get to see the result provided by the output layer. However, understanding what happens in the hidden layers is the key step to neural network implementation and optimization.

Neural Networks are currently being used in many industries such as: Aerospace, Robotics, Banking, Security and many more.

So, in this project we try to understand what a Neural network is, how they work and why do we use them and finally construct a Neural Network Called Hopfield's Neural network using McCulloch pits neuron the most fundamental unit of artificial Neuron.

Modern neural networks are just playing with matrices. So, in a few words, Hopfield recurrent artificial neural network is not an exception and is a customizable matrix of weights. The Hopfield model accounts for associative memory through the incorporation of memory vectors and is commonly used for pattern classification.

Hopfield nets serve as content-addressable memory systems with binary threshold nodes. They are guaranteed to converge to a local minimum, but convergence to a false pattern (wrong local minimum) rather than the stored pattern (expected local minimum) can occur. Hopfield networks also provide a model for understanding human memory.

This paper presents the results how it converges and some cases when it is not (According to the noise). It depends on noise of data given. As they are three modes (serial, parallel, partially parallel), we have shown the outputs and difference in three modes (computation time: number of iterations to converge). We have been working on different matrices (weight matrix), letting you know whether the neuron is in stable state or unstable state. Considering many cases, we have been implementing Hopfield neural network to hike the performance of the model.

Key Words: Neural Networks, Deep Learning, McCulloch pits neuron, Hopfield's Network, Feedforward, Backpropagation.

Contents

1.Introduction:

It is more often seen that people comparing the human brain and the electronic computer. A typical brain contains something like 100 billion miniscule cells called neurons. Each neuron is made up of a cell body (the central mass of the cell) with a number of connections coming off it: numerous dendrites (the cell's inputs—carrying information toward the cell body) and a single axon (the cell's output—carrying information away). Neurons are so tiny that you could pack about 100 of their cell bodies into a single millimetre. Inside a computer, the equivalent to a brain cell is a nanoscopically tiny switching device called a transistor.

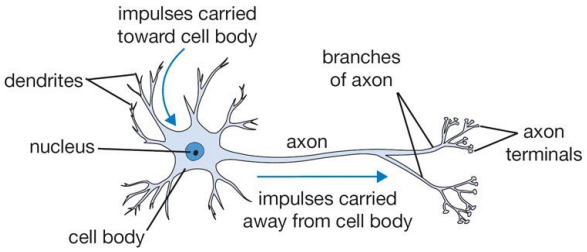

Fig1.1Theoretical model of a neuron

This essential structural difference between computers (with maybe a few hundred million transistors connected in a relatively simple way) and brains (perhaps 10–100 times more brain cells connected in richer and more complex ways) is what makes them "think" so very differently. Computers are perfectly designed for storing vast amounts of meaningless information and rearranging it in any number of ways according to precise instructions (programs) we feed into them in advance. Brains, on the other hand, learn slowly, by a more roundabout method, often taking months or years to make complete sense of something really complex. But, unlike computers, they can spontaneously put information together in astounding new ways—that's where the human creativity of a Beethoven or a Shakespeare comes from—recognizing original patterns, forging connections, and seeing the things they've learned in a completely different light.

It would be great if computers were more like brains, that's where Artificial Neural Networks come in!

1.1 What is Artificial Neural Network:

Artificial Neural Networks, also known as “Artificial neural nets”, “neural nets”, or ANN is an attempt to simulate the network of neurons that make up a human brain so that the computer will be able to learn things and make decisions like a human. ANN use different layers of mathematical processing to make sense of the information it’s fed. Typically, an ANN has anywhere from dozens to millions of artificial neurons—called units—arranged in a series of layers. The input layer receives various forms of information from the outside world. This is the data that the network aims to process or learn about. From the input unit, the data goes through one or more hidden units.

The hidden unit's job is to transform the input into something the output unit can use,

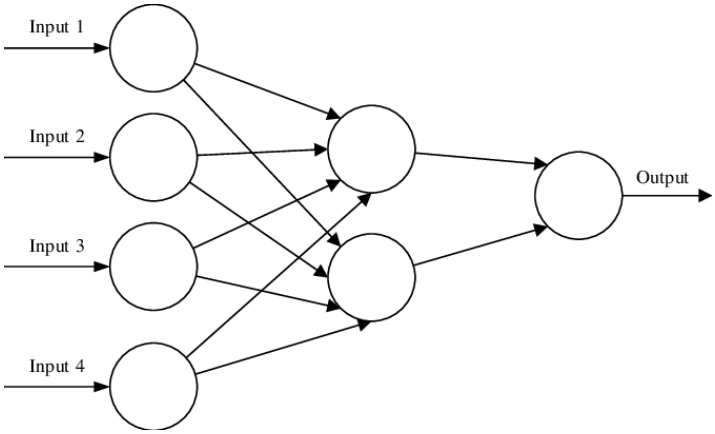

Fig 1.2 Schematic representation of ANN

1.2 Real life applications of ANN:

With the human-like ability to problem-solve — and apply that skill to huge datasets — Artificial neural networks possess the following powerful attributes:

- **Adaptive Learning:** Like humans, neural networks model non-linear and complex relationships and build on previous knowledge. For example, software uses adaptive learning to teach math and language arts.
- **Self-Organization:** The ability to cluster and classify vast amounts of data makes neural networks uniquely suited for organizing the complicated visual problems posed by medical image analysis.
- **Real-Time Operation:** ANN can (sometimes) provide real-time answers, as is the case with self-driving cars and drone navigation.
- **Prognosis:** ANN's ability to predict based on models has a wide range of applications, including for weather and traffic.
- **Fault Tolerance:** When significant parts of a network are lost or missing, ANN can fill in the blanks. This ability is especially useful in space exploration, where the failure of electronic devices is always a possibility.

Here's a list of few other Artificial neural network engineering applications currently in use in various industries:

- **Aerospace:** Aircraft component fault detectors and simulations, aircraft control systems, high-performance auto-piloting, and flight path simulations.
- **Automotive:** Improved guidance systems, development of power trains, virtual sensors, and warranty activity analysers.
- **Mechanics:** Condition monitoring, systems modelling, and control.

- **Electronics:** Chip failure analysis, circuit chip layouts, machine vision, non-linear modelling, prediction of the code sequence, process control, and voice synthesis.
- **Robotics:** Forklift robots, manipulator controllers, trajectory control, and vision systems.

Here are further current examples of ANN business applications:

- **Banking:** Credit card attrition, credit and loan application evaluation, fraud and risk evaluation, and loan delinquencies.
- **Business Analytics:** Customer behaviour modelling, customer segmentation, fraud propensity, market research, market mix, market structure, and models for attrition, default, purchase, and renewals
- **Défense:** Counterterrorism, facial recognition, feature extraction, noise suppression, object discrimination, sensors, sonar, radar and image signal processing, signal/image identification, target tracking, and weapon steering
- **Education:** Adaptive learning software, dynamic forecasting, education system analysis and forecasting, student performance modelling, and personality profiling
- **Financial:** Corporate bond ratings, corporate financial analysis, credit line use analysis, currency price prediction, loan advising, mortgage screening, real estate appraisal, and portfolio trading
- **Medical:** Cancer cell analysis, ECG and EEG analysis, emergency room test advisement, expense reduction and quality improvement for hospital systems, transplant process optimization, and prosthesis design
- **Securities:** Automatic bond rating, market analysis, and stock trading advisory systems
- **Transportation:** Routing systems, truck brake diagnosis systems, and vehicle scheduling

The use of neural networks seems unstoppable. With the advancement of computer and communication technologies, the whole process of doing business, engineering for the matter of fact almost everything has undergone a massive change.

This project is about study and understanding McCulloch-Pitts Model and Hopfield neural network which will be introduced and will be discussed in more detail in the coming chapters of the report.

2.Problem Definition

This project deals with the understanding and implementing McCulloch-Pits model of neuron and subsequent concepts related to it, applying Hopfield neural network for McCulloch-Pits model of neuron, and stability and convergence of MP model neuron.

2.1 Overview of McCulloch-Pits model of neuron & Hopfield Network:

The McCulloch-Pitts model of a neuron or M-P neuron is simple yet has substantial computing potential. It also has a precise mathematical definition. However, this model is so simplistic that it only generates a binary output and also the weight and threshold values are fixed.

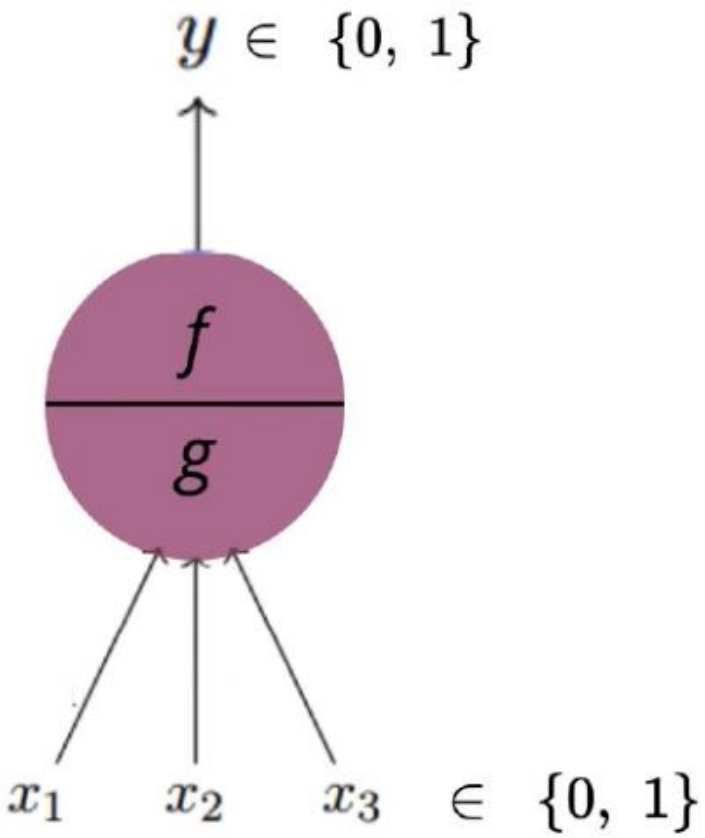

Fig 2.1 Basic model of McCulloch-Pits neuron

A Hopfield network which operates in a discrete line fashion or in other words, it can be said the input and output patterns are discrete vector, which is expressed in bipolar +1, −1 in nature. They

are fully interconnected neural networks. The network has symmetrical weights with no self-connections.

Common applications of Hopfield networks have been used for image detection and recognition, enhancement of X-Ray images, medical image restoration, etc.

Hopfield networks are somewhat older, and are the base of Restricted Boltzmann Machines (RBMs) and Deep belief network (DBNs), therefore much of the applications of the HNs can be made with these architectures.

The network has been run on different datasets and evolved results with different variation in weight matrix. we have some improvement in Hopfield network by training the noisy data. An approach to training a generalized Hopfield network is developed and evaluated.

2.2 The main problems which we were more likely to be focused were:

(1) Write a working program for implementing Hopfield neural network. Some variable conditions (ranges) were constrained that is given a [+1, -1} vector, run the network in serial mode as well as parallel mode.

Show with your program that convergence to a stable states e occurs or a cycle of length almost two is reached.

(2) Consider a weight matrix all of whose diagonal elements are negative. Determine the stable states of Hopfield network.

Now, change the weight matrix with strictly positive diagonal elements. Determine the stable states of this Hopfield neural network.

(3) Take simple examples of Hopfield network. For instance, (3 X 3) matrix, (4 X4) synaptic weight matrices. Manually run the network (without using your computer program) in serial and fully parallel modes. Check if the program gives same results.

(4) Run the neural network in partial parallel modes of operation (state updation is done at more than one neural node buy NOT all neuronal nodes) and verify what happens to convergence.

3.Background

3.1.Neural Networks and Related Work

Neural Network is a biologically inspired mathematical tool used widely for prediction and classification.

Neural Network consists of the following components:

1) Input Layer: The function of this layer is to take input
2) Hidden Layer: This layering process that input as per the given activation function
3) Output Layer: After processing by hidden layer output layer provides the output
4) A set of weights and biases between each layer, W and b
5) A choice of activation function for each hidden layer

Eg: Sigmoid Function is one of the well-known Threshold activation function

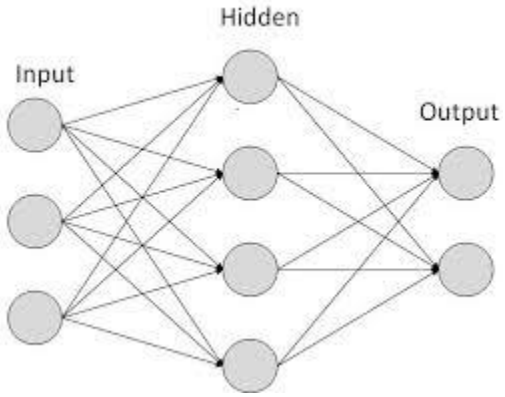

Figure 3.1 ANN

An artificial neuron is a processing unit which processes the input. The input is taken by an input vector x. Each edge connecting two neurons have some weight W(i,j) shows the connection strength between two neurons. The weights are given in a weight matrix W.

3.1.1. Training The Neural Network

Naturally, The right values for the weights and biases determine the strength of predictions. The process of fine-tuning the weights and biases from the input data is known as training a neural network.

The training process consists of the following steps:

1) Calculating the predicted output, known as feedforward.

2) Updating the weights and biases, known as backpropagation

3.1.1.1 Feedforward

Feedforward is the method of calculating the output using the activation function. However, we still need to know the goodness of our predictions. The Loss function allows us to do exactly that. There are many available loss functions, and the nature of our problem should dictate our choice of the loss function. Our goal in training is to find the best set of weights and biases that minimizes the loss function.

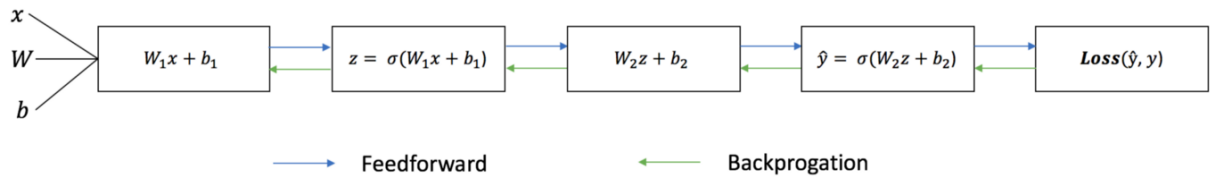

Fig 3.2 Two-layered NN demonstrating Feedforward and backpropagation

3.1.1.2 Backpropagation

Now that we've measured the error of our prediction (loss), we need to find a way to propagate the error back and to update our weights and biases. Backpropagation is about understanding how changing the weights and biases in a network changes the loss function. This means that we have to calculate the derivative of the loss function w.r.t the weights and biases. If we have the derivative, we can simply update the weights and biases by increasing/reducing with it. This is known as gradient descent. According to this, we update our weights and biases and calculate the output until the loss function is minimum.

3.2 McCulloch-Pitts Neuron

The first computational model of a neuron was proposed by Warren McCulloch and Walter Pitts.

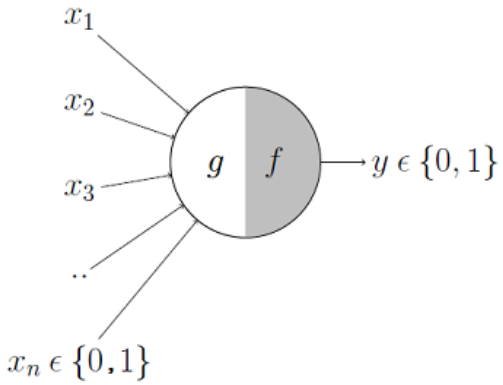

Fig 3.3: McCulloch Pitts Neuron

It may be divided into 2 parts. The first part, g takes an input, performs aggregation and based on the aggregated value the second part, f makes a decision. Each input can be either inhibitory or excitatory. Inhibitory inputs are those that have maximum effect on the decision making irrespective of other inputs. Excitatory inputs are not the ones that will make the neuron fire on their own but they might fire it when combined together. The output will be either 0 or 1 by using the below method. The input and output can be taken as -1 or +1 instead of 0 and 1 depending upon the problem. Here in our project, we took {-1,+1} implementation. We can see that g(x) is just doing a sum of the inputs — a simple aggregation. And theta here is called thresholding parameter.

$$g(x_1, x_2, x_3, \ldots, x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

$$\begin{aligned} y = f(g(\mathbf{x})) &= 1 \quad if \quad g(\mathbf{x}) \geq \theta \\ &= 0 \quad if \quad g(\mathbf{x}) < \theta \end{aligned}$$

Since the inputs are all Boolean the basic Boolean functions such as OR, AND, NAND, NOT, NOR could be implemented in this model.

3.3 Hopfield Neural Network

Hopfield neural networks represent a new neural computational paradigm by implementing an auto-associative memory. They are recurrent or fully interconnected neural networks. There are two versions of Hopfield neural networks: in the binary version all neurons are connected to each other but there is no connection from a neuron to itself, and in the continuous case all connections including self-connections are allowed.

The Hopfield network has a finite set of neurons x(i),$1 \leq i \leq N$, which serve as processing units. Each neuron has a value (or state) at the time t described by $x_t(i)$. A neuron in the Hopfield net has one of the two states, either -1or +1; that is, $x_t(i) \in \{-1,+1\}$.

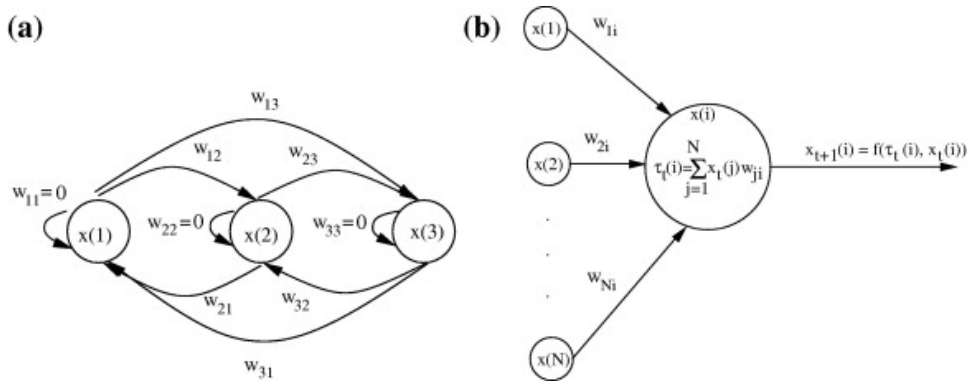

Fig 3.4(a) Connections in Hopfield Neural Network
Fig 3.4(b) Showing the processing in the Hopfield Neural Network

For each pair of neurons, x(i) and x(j), there is a connection wij called the synapse between x(i) and x(j). The design of the Hopfield net requires that wij=wji and wii=0.

3.4Activation Function

Activation functions are mathematical equations that determine the output of a neural network. The function is attached to each neuron in the network, and determines whether it should be activated ("fired") or not, based on whether each neuron's input is relevant for the model's prediction. Activation functions also help normalize the output of each neuron between -1 and 1.

The activation function f determines the next state of the neuron $x_{t+1}(i)$. The next state can be calculated by using the formula

$$x_{t+1}(i)=f(T_i,x_i)=\{1 \text{ if } T_t>\Theta$$
$$\{-1 \text{ if } T_t<\Theta$$

where $T_t(i)=\Sigma x_t(j)*w_{ij}$, Θ is Threshold Value. Here, f is signum function.

Binary Step Function, Linear Activation function, Non Linear Activation function are few examples of activation function

3.5 State Space

State-space of a neural network is the space in which all the possible bipolar (+1 or -1) values of a neural model can be visualized. For a 2- neuron system state space is {(+1, +1), (+1, -1), (-1,+1), (-1, -1)} which is a square. For a 3-neuron system state space is a cube. With N neurons state space is a unit hypercube.

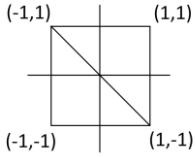

Fig 3.5 (a) State space of 2 neuron

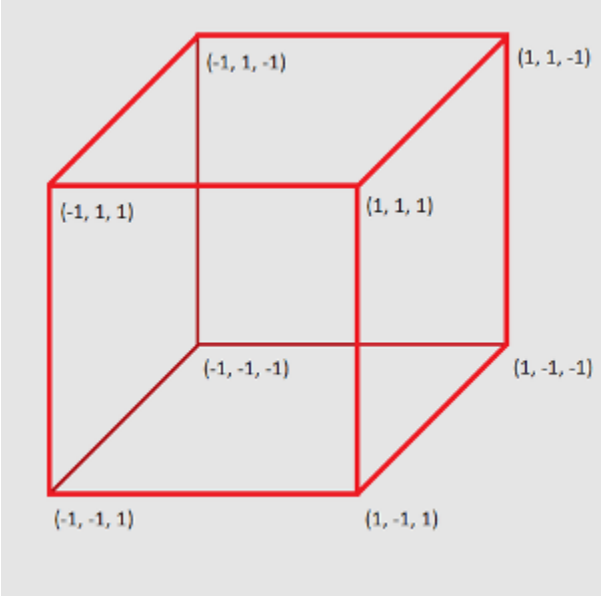

Fig 3.5(b) State space of 3 neurons

3.6 State Updation

$x_{t+1}(i)=f(T_i,x_i)=\{1 \text{ if } T_t>\Theta$

$\{-1 \text{ if } T_t<\Theta$

where $T_t(i)=\Sigma x_t(j)*w_{ij}$, $x_{t+1}(i)$ is state of neuron at time t+1, $x_t(i)$ is state at time t, f is a activation function

- **Serial Mode:** If the state updation takes places one neuron at a time instant at a time, it is called SERIAL MODE.
- **Fully Parallel Mode:** If the state updation of all the neurons takes place all a time, it is called the FULLY PARALLEL MODE.
- **Partial Parallel Mode:** If state updation of all the neurons does not take at a time then this mode is PARTIAL PARALLEL MODE.

3.7 Convergence Theorem:

The convergence properties are dependent on the structure of W and the method by which the nodes are updated.

The Hopfield network always converges to a stable state in serial mode i.e

(1) $z=f(T_i,z)$ where $T_t(i)=\Sigma x_t(j)*w_{ij}$ and f is the activation function, z is the current state of the neuron.

(2) In fully parallel mode we have convergence or cycle of length 2 i.e

$y_{l\_}>>$ y2 >>y1 >>y2.

4.Implementations

First, we were given a research paper in which there were concepts McCollouch-Pitts neuron and a basic 2-neuron Hopfield Neural Network. It also had the concepts of state space of the neural network and about the convergence of the neural network. We studied the paper and understood all the concepts. Then we made a presentation to project coordinator.

Hopfield Neural Networks are recurrent artificial neural networks. In this type of ANN, the processing elements are the neurons and every output of each neuron is connected to the input of all other neurons via synaptic weights. All weights are calculated using the Hebbian rule defined.

Let W be the synaptic weight matrix and V be the vector in which the state of each neuron in stored in order. The algorithm for finding whether is the network is stable or not is:

4.1 Algorithm Steps

1. Enter the number of neurons in the network
2. Enter the state of neurons either -/+ 1
3. If the state of the neurons isn't +/- 1 exit
4. Create square zero matrix with the dimensions same as the number neurons in the network
5. Now generate weight matrix by using np.random function in numpy library or we can use formulae W(i,j)=a(i)*a(j) which is known as Hebb's rule.
6. **Here we generated matrices in both ways and solved the problem separately**
7. Check that the matrix generated is symmetric and check whether all the diagonal elements in the matrix is zero.
8. Multiply the weight matrix element W(i,j) with the corresponding vector state element v(j)
9. Sum the result of the above multiplication along each row of to compute the weighted sum.
10. Apply activation function to the sum to get the state of neuron at t+1
11. Repeat steps 10,9 until all the rows in the weight matrix are done.
12. Now compare the new state vector to the old state vector matrix
13. If both the vectors are same return stable network or convergence theorem satisfied

    Else: return not stable or theorem not satisfied.

In the above method we generated the weight matrix using Hebb's rule.

4.1.1 Generating Weight Matrix Randomly

Next, we generated the weight matrix randomly. We generated it using the ***NUMPY library.*** Code snippet is attached.

```
import numpy as np
import random as rd


#defining weight matrix using HEBBS

r=int(input("Enter the no.of neurons you want in your network:"))
vector=[]
print("Enter the vectors of each neuron which should be +/-1:")
for i in range(1):
    for j in range(r):
        vector.append(int(input()))

print(vector)
given=vector
weigth=np.random.randint(-1,2,size=(len(vector),len(vector)))#generating random matrix
for i in range(len(vector)):
    for j in range(len(vector)):
        if(i==j):#for making diagonal elements zero
            weigth[i][j]=0
        elif(i!=j and weight[i][j]==0):
            weigth[i][j]=1
        else:#for making  the synnetric matrix
            weigth[j][i]=weight[i][j]

print('The weigth matrix is ....')
print(weigth)
```

Fig 4.1 Weight Matrix Generation

The above code ode snippet generates a random matrix with its elements ranging –1,1,0. The diagonal elements are made zero and the matrix is made symmetric because W(i,j)=W(j,i).

4.1.2 Serial Mode

```
#finding STABLE or NOT for SERIES
print('Series......')
for i in range(0,len(vector)):
    summ=0
    for j in range(0,len(vector)):
        summ+=vector[j]*weigth[i][j]
    if summ>=0:
        vector[i]=1
        summ=0
    else:
        vector[i]=-1
        summ=0

for i in range(0,len(vector)):
    if vector[i]==given[i]:
        print('neuron'+str(i+1)+' is STABLE')
    else:
        print('neuron'+str(i)+' is UNSTABLE')
```

Fig 4.2 Serial Mode

The above code snippet will check the convergence of the neural network in the series mode. We multiply the weight matrix element W(i,j) with the corresponding vector state element v(j). Then we sum the result of the multiplication along each row of to compute the weighted sum. Next, we applied the activation function to the weighted sum which gives us a certain value. The activation function we used is called **Threshold Activation Function or Sigmoid Function.**

The threshold function or sigmoid function is defined as follows:

Sign{x}=1 if x>=0

=-1 if x<0

4.1.3 Parallel mode

```
#for parallel
print('parallel......')

par=np.zeros((1,len(vector)),dtype='int')
print(par)
for i in range(0,len(vector)):
    for j in range(0,len(vector)):
        par[0][i]+=given[j]*weigth[i][j]
    if par[0][i]>=0:
        par[0][i]=1
    else:
        par[0][i]=-1

for i in range(0,len(vector)):
    if par[0][i]==given[i]:
        print('neuron'+str(i+1)+' is STABLE')
    else:
        print('neuron'+str(i+1)+' is UNSTABLE')
```

Fig 4.3 Parallel Mode

The above code snippet is for checking the convergence in fully parallel mode. Unlike serial mode, in fully parallel mode we update all the neurons at the same time. Here we took separate matrix and multiplied all weight elements W(I,j) with the vector state along the row all at a time. And we applied activation function I.e is sigmoid function to the output.

After implementing the above code, we concluded that in series mode the network converges to a stable mode. In fully parallel mode the network converges to a stable state or a cycle of length 2.

4.1.4 Partial Parallel mode

```
#for partial parallel
print('partial paralle......')

diff=rd.randint(0,len(vector)-1)
print('diff is '+str(diff))
par=np.zeros((1,len(vector)),dtype='int')
print(par)
for i in range(0,len(vector)):
    if i!=diff:
        for j in range(0,len(vector)):
            par[0][i]+=given[j]*weigth[i][j]
        if par[0][i]>=0:
            par[0][i]=1
        else:
            par[0][i]=-1
for j in range(0,len(vector)):
    if j!=diff:
        par[0][diff]+=par[0][j]*weigth[diff][j]
    else:
        par[0][diff]+=given[j]*weigth[diff][j]
if par[0][diff]>=0:
    par[0][diff]=1
else:
    par[0][diff]=-1
for i in range(0,len(vector)):
    if par[0][i]==given[i]:
        print('neuron'+str(i+1)+' is STABLE')
    else:
        print('neuron'+str(i+1)+' is UNSTABLE')
```

Fig 4.4 Partial Parallel mode

The above code snippet is for checking the convergence in fully parallel mode. Unlike serial mode, in fully parallel mode we update all the neurons at the same time. Here we took separate matrix and multiplied all weight elements W(i,j) with the vector state along the row all at a time. And we applied activation function I.e is sigmoid function to the output.

5.Results

Let N= (W, T) be a Hopfield neural network where W is an n X n symmetric matrix, where w_{ij} is equal to the weight attached to the neuron i & j. T is a vector of dimension n.

1) if N is operating in serial mode and W is a symmetric matrix with zero diagonal, then the network will always converge to a stable state
2) if N is operating in serial mode and W is a symmetric matrix with non-negative elements on the diagonal, then the network will always converge to a stable state.
3) if N is operating in a fully parallel mode, then for an arbitrary symmetric matrix W the network will always converge to a stable state or a cycle of length 2; that is, the 2 i.e cycles in the state space are of length 2

Now if the weight matrix doesn't contain zero diagonal elements, instead have negative elements in the diagonals we observed that network dynamics doesn't necessarily lead to stable state.

For example, we considered the weight below weight matrix:

$$W = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

When using the above weight matrix, it transforms the state vector (1, 1, 1) into the state vector (−1, −1, −1) and vice-versa. In the case of asynchronous updating or fully parallel mode, the network chooses randomly among the eight possible network states. So, by considering above example we can say that the network need not converge to a stable state when there are negative elements in the diagonals.

Next, we took simple examples of 3*3 and 4*4 weight matrices and manully ran the network in both serial and parallel mode. Then we took the same synaptic weight matrices and ran them using the program. We got the same resu;ts for both of them.

6.Conclusion

In this project we studied, analysing the convergence and stability of Hopfield Neural Networks operating in serial mode, fully parallel mode. We also learnt about the convergence and stability of Hopfield Networks in partially parallel mode. This analysis is interesting because these neural networks are easily implementable and take advantage of the original parallelism of Hopfield Neural Networks. We also checked the network for series, parallel, partial parallel manually and made sure the code is correct.

Moreover, with our study we proved that Hopfield Networks operating in fully parallel mode always converge to a stable state or to a cycle of length two. We also learnt that the Hopfield Neural Networks operating in serial mode always converge. The convergence of the Hopfield Neural Network depends upon the weight matrix.

7.References

(1) J.J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", *Proc. of the National Academy of Sciences USA*, vol. 79, pp. 2554-2558, 1982.
(2) G. Rama Murthy and Moncef Gabbouj,"Existence and Synthesis of Complex Hopfield Type Associative Memories," Proceedings of International Work Conference on Artificial Neural Networks, June 2015, Lecture Notes in Computer Science, Springer
(3) Bruck, J.: On the Convergence Properties of the Hopfield Model. Proceeding of the IEEE **78**(10), October 1990
(4) J. Bruck, J. W. Goodman, "A generalized convergence theorem for neural networks", *IEEE Trans. Inform. Theory*, vol. 34, pp. 1089-1092, Sept. 1988.
(5) Taiwei Lu, Xin Xu, Sudong Wu, and Francis T. S. Yu, "Neural network model using interpattern association
(6) D Burshtein - IEEE Transactions on Information Theory, 1994 - ieeexplore.ieee.org
(7) J. Komlós, R. Paturi, "Convergence results in an associative memory model", *Neural Networks*, vol. 1, pp. 239-250, 1988.
(8) Enhanced Hopfield Network for Pattern Satisfiability Optimization
(9) Mohd. Asyraf Mansor, Mohd Shareduwan Bin Mohd Kasihmuddin, Saratha Sathasivam , Published 2016
(10) International Journal of Intelligent Systems and Applications
(11) Somesh Kumar Ramesh Chandra Sahoo and Puneet Goswami. Implementation of Hopfield Neural Network for its Capacity with Finger Print Images. *International Journal of Computer Applications* 141(5):44-49, May 2016
(12) Ramasamy Ramachandran, Natarajan Gunusekharan, "Optimal Implementation of two-Dimensional Bipolar Hopfield Model Neural Network", Proc. Natl. Sci. Counc. ROC(A), 2000
(13) Hopfield Neural Networks—A Survey Humayun Karim Sulehria, Ye ZhangSchool of Electronics and Information EngineeringHarbin Institute of Technology, HarbinPR China\
(14) A Generalized Convergence Theorem for Neural Networks JEHOSHUA BRUCK AND JOSEPH W. GOODMAN, FELLOW, IEEE
(15) Convergence and Stability of Quantized Hopfield Networks Operating in a Fully Parallel Mode Daniel Calabuig ,Jose F. Monserrat, Narcís Cardona,Universidad Politécnica de Valencia, Institute of Telecommunications and Multimedia Applications, Valencia 46022, Spain
(16) https://www.sciencedirect.com/topics/earth-and-planetary-sciences/artificial-neural-network
(17) https://towardsdatascience.com/introduction-to-neural-networks-advantages-and-applications-96851bd1a207
(18) https://data-flair.training/blogs/artificial-neural-network/
(19) https://mindmajix.com/artificial-neural-network-and-how-it-works
(20) http://web.cs.ucla.edu/~rosen/161/notes/hopfield.html
(21) https://www.sciencedirect.com/topics/computer-science/hopfield-network
(22) https://towardsdatascience.com/hopfield-networks-are-useless-heres-why-you-should-learn-them-f0930ebeadcd
(23) https://www.skedsoft.com/books/neural-network-fuzzy-systems/limitations-to-using-the-hopfield-network
(24) http://ecee.colorado.edu/~ecen4831/hoplecs/hoplec1.html
(25) https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1
(26) https://aishack.in/tutorials/artificial-neurons-mccullochpitts-model/
(27) https://www.smartsheet.com/neural-network-applications
(28) https://aishack.in/tutorials/artificial-neurons-mccullochpitts-model/
(29) http://wwwold.ece.utep.edu/research/webfuzzy/docs/kk-thesis/kk-thesis-html/node12.html
(30) https://www.explainthatstuff.com/introduction-to-neural-networks.html
(31) https://en.wikibooks.org/wiki/Artificial_Neural_Networks/Neural_Network_Basics
(32) https://medium.com/@jamesdacombe/an-introduction-to-artificial-neural-networks-with-example-ad459bb6941b
(33) https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/