**The Project entitled**

**Implementation of Back Propagation Algorithm**

Submitted in partial fulfillment of academic requirements for the award of the degree of

Bachelor of Engineering (Computer Science and Engineering)

**By**

**SUSMITHA.N 2451-13-733-102**

**G. LOUKYA 2451-13-733-113**

**ACHSAH JUNIA LEDALA 2451-13-733-116**

****

**Department of Computer Science and Engineering**

**M.V.S.R. ENGINEERING COLLEGE**

**(Affiliated to Osmania University & Recognized by AICTE)**

**Nadergul, Saroor Nagar Mandal, Hyderabad – 501 510**

**2015-16**

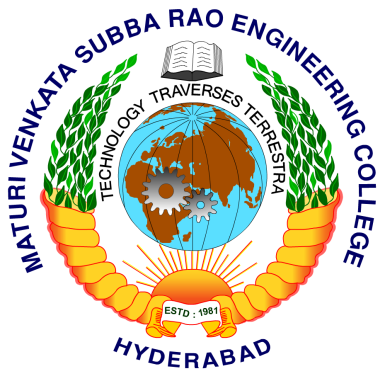
**Department of Computer Science and Engineering**

**M.V.S.R. ENGINEERING COLLEGE**

**(Sponsored by Matrusri Education Society, Estd. 1980)**

**(Affiliated to Osmania University & Recognized by AICTE)**

**Nadergul, Saroor Nagar Mandal, Hyderabad – 501 510**

****

**CERTIFICATE**

This is to certify that the project entitled **“Implementation of Back Propagation Algorithm”,** is being submitted by Ms **ACHSAH JUNIA LEDALA, SUSMITHA N, G.LOUKYA** bearing H.T No **2451-13-733-116, 2451-13-733-102, 2451-13-733-113** in partial fulfillment of academic requirements for the award of the degree of BACHELOR OF ENGINEERING in COMPUTER SCIENCE AND ENGINEERING from MVSR Engineering College, affiliated to OSMANIA UNIVERSITY, is a record of bonafide work carried out by us under the guidance and supervision of the faculty (CSED). The results embodied in this project report have not been submitted to any other University or Institute for the award of any degree or diploma to the best of our knowledge and belief.

|  |  |  |
| --- | --- | --- |
| Guide |  | Project Coordinator |
| **Mrs. B.SARITHA**, |  | **Mr. K.MURALI KRISHNA** |
| Associate Professor, |  | Assistant Professor, |
| Dept. of CSE. |  | Dept. of CSE. |
| MVSR Engineering College |  | MVSR Engineering College |

**ACKNOWLEDGMENT**

We take this opportunity to express our profound and sincere gratitude to all those who helped us in carrying out this project successfully.

At the very outset, we are thankful to our principal Sri **Dr. V Chandrasekhar** Garuand Sri **J.Prasanna Kumar**, Professor and Head, Department of Computer Science and Engineering, MVSR Engineering College, Hyderabad for their consent to do the project work as a part of our B.E Degree (CSE). We thank them for their valuable suggestions and advice throughout our stay at the college, during our project work.

We would like to thank our Internal Project Guide Mrs. **B. Saritha**, Associate Professor, Department of Computer Science and Engineering, MVSR Engineering College for her useful suggestions, guidance and encouragement.

We thank the teaching and non-teaching staff of CSE for extending their support.

Finally we are thankful to our parents for their cooperation and support throughout all endeavors in our life.

Susmitha N (2451-13-733-102)

Loukya G (2451-13-733-113)

Achsah ledala (2451-13-733-116)

**ABSTRACT**

Neural network software is used to [simulate](https://en.wikipedia.org/wiki/Simulation), [research](https://en.wikipedia.org/wiki/Research), [develop](https://en.wikipedia.org/wiki/Software_development), and apply [artificial neural networks](https://en.wikipedia.org/wiki/Artificial_neural_network), software concepts adapted from [biological neural networks](https://en.wikipedia.org/wiki/Biological_neural_network), and, in some cases, a wider array of [adaptive systems](https://en.wikipedia.org/wiki/Adaptive_system) such as [artificial intelligence](https://en.wikipedia.org/wiki/Artificial_intelligence) and [machine learning](https://en.wikipedia.org/wiki/Machine_learning).

A neural network for system is defined by a set of input neurons which may be activated by the components of an input data. After being weighted and transformed by a [function](https://en.wikipedia.org/wiki/Function_(mathematics)) (determined by the network's designer), the activations of these neurons are then passed on to other neurons. This process is repeated until finally, an output neuron is activated.

In the artificial neural network, each circular node represents an artificial neuron and an arrow represents a connection from the output of one neuron to the input of another neuron.

The artificial neural network consists of one layer of hidden neurons and one output neuron. When an input vector is propagated through the network, for the current set of weights there is an output Pred. The objective of supervised training is to adjust the weights so that the difference between the network output Pred and the required output Req is reduced. This requires an algorithm that reduces the absolute error, which is the same as reducing the squared error. This is done by back propagating the errors to the immediate previous layer and so on until the error is minimized. Thus we are implementing an algorithm that is used to minimize the squared error of the network output and the required output.

(i)

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIG NO:** | **NAME OF THE FIGURE** | **PGNO:** |
| 1.1 | Simple neuron structure | 1 |
| 2.1 | Architechure of neural network | 3 |
| 2.2 | Representation of Back Propagation | 5 |
| 5.1 | Target Vs Observed output for two input values | 13 |
| 5.2 | Target Vs Observed output for three input values | 14 |
| 5.3 | Target Vs Observed output for four input values | 14 |
| 5.4 | Random values are generated for weights | 15 |
| 5.5 | Back propagating the errors for each epoch | 15 |
| 5.6 | Output for two input values | 16 |
| 5.7 | Output for three input values | 17 |
| 5.8 | Output for four input values | 18 |

**LIST OF TABLES**

|  |  |  |
| --- | --- | --- |
| **TABLE NO** | **NAME OF THE TABLE** | **PG NO** |
| 5.1 | Target Vs Output values for two inputs | 16 |
| 5.2 | Target Vs Output values for three inputs | 17 |
| 5.3 | Target Vs output values for four inputs | 18 |

(ii)

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **SL.NO** | **CHAPTER** | **PG NO:** |
| 1. | Introduction | **1** |
| 2. | Literature survey  2.1 Architectural neural networks  2.2 Learning | 2  2  4 |
| 3. | Software Requirement Specification  3.1 Functional Requirements  3.2 Non-Functional Requirements | 6  6  6 |
| 4. | Implementation  4.1 Pseudo code  4.2 Code for Backpropagation | 7  8  9 |
| 5. | Results  5.1 output screens  5.2 observation | 13  13  16 |
| 6. | Conclusion | 19 |

(iii)

1. **INTRODUCTION**

The simplest definition of a neural network, more properly referred to as an 'artificial' neural network (ANN), is provided by the inventor of one of the first neuro computers, Dr. Robert Nielsen. He defines a neural network as: *“*A computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs.”

The basic idea behind neural networks is the neural system in the human brains. A typical brain contains something like 100 billion miniscule cells called **neurons** (no-one knows exactly how many there are and estimates go from about 50 billion to as many as 500 billion). 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). The basic structure of a neuron is given in Fig.1.1

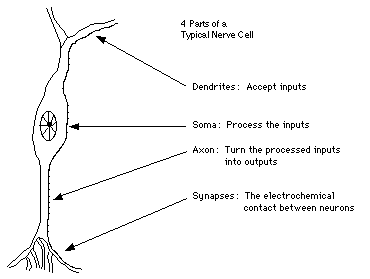


Fig 1.1: A SIMPLE NEURON STRUCTURE

**2. LITERATURE SURVEY**

**2.1 ARCHITECTURE OF NEURAL NETWORK**

**Input units**, are designed to receive various forms of information from the outside world that the network will attempt to learn about, recognize, or otherwise process.

**Output units**. Other units sit on the opposite side of the network and signal how it responds to the information it's learned.

In between the input units and output units are one or more layers of **hidden units**, which, together, form the majority of the artificial brain.

The connections between one unit and another are represented by a number called **weight**.

The higher the weight, the more influence one unit has on another.

The architecture of the neural network is shown in Fig. 2.1

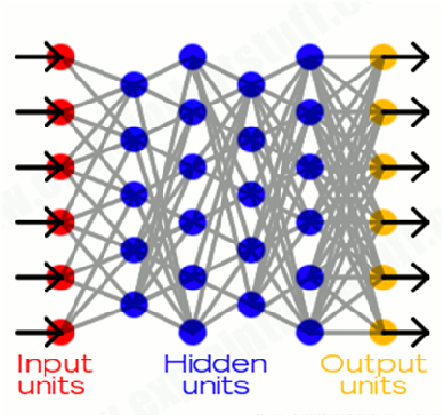


Fig 2.1: ARCHITECTURE OF NEURAL NETWORK

Information flows through a neural network in two ways:

* When it's learning (being trained)
* operating normally (after being trained)

Patterns of information are fed into the network via the input units, which trigger the layers of hidden units, and these in turn arrive at the output units.

**2.2 LEARNING**

* Neural networks learn things in exactly the same way as a human brain learns, typically by a feedback process called **back propagation** (sometimes abbreviated as "back prop").
* This involves comparing the output a network produces with the output it was meant to produce, and using the difference between them to modify the weights of the connections between the units in the network, working from the output units through the hidden units to the input units i.e going backward, in other words. Refer Fig.2.2
* In time, back propagation causes the network to learn, reducing the difference between actual and intended output to the point where the two exactly coincide, so the network figures things out exactly as it should.
* The idea of the back propagation algorithm is to reduce this error, until the ANN learns the training data. The training begins with random weights, and the goal is to adjust them so that the error will be minimal.

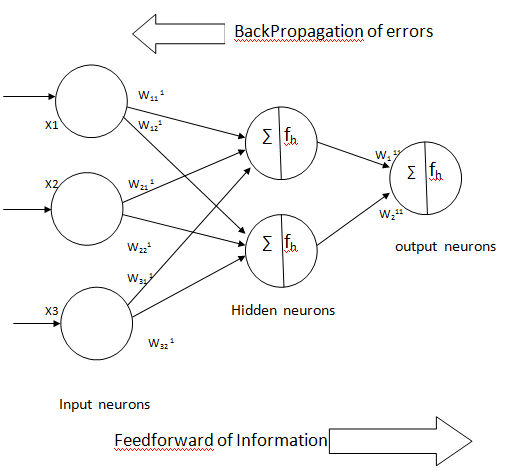


Fig 2.2 : REPRESENTATION OF BACK PROPAGATION

**3. SOFTWARE REQUIREMENTS SPECIFICATIONS**

**3.1 Functional Requirements:**

* The weights of each and every node are randomly assigned.
* Feedback should be available for the input node at every level of the algorithm for error minimization.
* The basic input data values must be provided.
* The target values also should be provided.
* The number of patterns that are obtained also should be specified.
* The activation function that is used to calculate the output at every stage should be chosen beforehand.

**3.2 Non Functional Requirements:**

* Hardware requirement: Windows 7, corei4processor.
* Software requirement: C, C++.

**4. IMPLEMENTATION**

The neural network consists of junctions which are connected with “LINKS”, also called processing units. For each junction a number is assigned, this number is called weight. The weights are the tools for the long distance information storing in the neural network, the learning process occurring with the appropriate modification of weights. These weights are modified so that the network input/output behaviour is in consonance with the environment,

which provide the input data.

The calculation algorithm consists of two basic steps:

1. Calculation of the output of the network, with inputs and weights.

2. Modification of weights with learning algorithm.

A single input neuron consists of a scalar input p multiplied by the scalar weight w to

form wp which is fed to the summer along with bias b multiplied by 1.

The net input is wp+b and the output a is: a=f (WP+b);

Where,

f- Transfer function

W & b can be adjusted by learning rule.

**4.1 PSEUDO-ALGORITHM**

1. Initialize all weights and biases in network;
2. **while** terminating condition is not satisfied {
3. **for** each training sample X in samples {
4. **//** Propagate the inputs forward:
5. **for** each hidden or output layer unit j {
6. Ij=∑i wijOi + Θj; //compute the net input of unit j with respect to the previous layer,i
7. Oj= 1/1+e-Ij ;} // compute the output of each unut j
8. // backpropagate the errors
9. **for** each unitj in the output layers
10. Errj= Oj(1-Oj)(T-O); //compute the error
11. for each unit j in the hidden layers,from the last to first hidden layer
12. Errj=Oj(1-Oj)∑j Errj wjk;//compute the error with respect to the next higher level , k
13. for each weight w in the network {
14. ∆wij=(l)Errj Oi; //weight increment
15. wij=wij+∆wij; } //weight update
16. for each bias in network{
17. ∆Θj=(l)Errj; //bias increment
18. Θj=Θj+∆Θj; } //bias update
19. }}

ALGORITHM FOR BACK PROPAGATION

**4.2 CODE FOR BACKPROPAGATION**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

#include <fcntl.h>

#define NUMPAT 4

#define NUMIN 2

#define NUMHID 2

#define NUMOUT 1

#define rando() ((double)rand()/((double)RAND\_MAX+1))

main() {

int i, j, k, p, np, op, ranpat[NUMPAT+1], epoch;

int NumPattern = NUMPAT, NumInput = NUMIN, NumHidden = NUMHID, NumOutput = NUMOUT;

double Input[NUMPAT+1][NUMIN+1] = { {0, 0, 0}, {0, 0, 0}, {0, 1, 0}, {0, 0, 1}, {0, 1, 1} };

double Target[NUMPAT+1][NUMOUT+1] = { {0, 0}, {0, 0}, {0, 1}, {0, 1}, {0, 0} };

double SumH[NUMPAT+1][NUMHID+1], WeightIH[NUMIN+1][NUMHID+1], Hidden[NUMPAT+1][NUMHID+1];

double SumO[NUMPAT+1][NUMOUT+1], WeightHO[NUMHID+1][NUMOUT+1], Output[NUMPAT+1][NUMOUT+1];

double DeltaO[NUMOUT+1], SumDOW[NUMHID+1], DeltaH[NUMHID+1];

double DeltaWeightIH[NUMIN+1][NUMHID+1], DeltaWeightHO[NUMHID+1][NUMOUT+1];

double Error, eta = 0.5, alpha = 0.9, smallwt = 0.5, r;

for( j = 1 ; j <= NumHidden ; j++ ) { /\* initialize WeightIH and DeltaWeightIH \*/

for( i = 0 ; i <= NumInput ; i++ ) {

DeltaWeightIH[i][j] = 0.0 ;

r=rando();

WeightIH[i][j] = 2.0 \* ( r - 0.5 ) \* smallwt ;

printf("the random value is %f \n",r);

}

}

for( k = 1 ; k <= NumOutput ; k ++ ) { /\* initialize WeightHO and DeltaWeightHO \*/

for( j = 0 ; j <= NumHidden ; j++ ) {

DeltaWeightHO[j][k] = 0.0 ;

//r=rando();

WeightHO[j][k] = 2.0 \* ( r - 0.5 ) \* smallwt ;

//printf("the random value is %f \n",r);

}

}

for( epoch = 0 ; epoch < 100000 ; epoch++) { /\* iterate weight updates \*/

for( p = 1 ; p <= NumPattern ; p++ ) { /\* randomize order of training patterns \*/

ranpat[p] = p ;

}

for( p = 1 ; p <= NumPattern ; p++) {

np = p + rando() \* ( NumPattern + 1 - p ) ;

op = ranpat[p] ; ranpat[p] = ranpat[np] ; ranpat[np] = op ;

}

Error = 0.0 ;

for( np = 1 ; np <= NumPattern ; np++ ) { /\* repeat for all the training patterns \*/

p = ranpat[np];

for( j = 1 ; j <= NumHidden ; j++ ) { /\* compute hidden unit activations \*/

SumH[p][j] = WeightIH[0][j] ;

for( i = 1 ; i <= NumInput ; i++ ) {

SumH[p][j] += Input[p][i] \* WeightIH[i][j] ;

}

Hidden[p][j] = 1.0/(1.0 + exp(-SumH[p][j])) ;

}

for( k = 1 ; k <= NumOutput ; k++ ) { /\* compute output unit activations and errors \*/

SumO[p][k] = WeightHO[0][k] ;

for( j = 1 ; j <= NumHidden ; j++ ) {

SumO[p][k] += Hidden[p][j] \* WeightHO[j][k] ;

}

Output[p][k] = 1.0/(1.0 + exp(-SumO[p][k])) ; /\* Sigmoidal Outputs \*/

/\* Output[p][k] = SumO[p][k]; Linear Outputs \*/

Error += 0.5 \* (Target[p][k] - Output[p][k]) \* (Target[p][k] - Output[p][k]) ; /\* SSE \*/

/\* Error -= ( Target[p][k] \* log( Output[p][k] ) + ( 1.0 - Target[p][k] ) \* log( 1.0 - Output[p][k] ) ) ; Cross-Entropy Error \*/

DeltaO[k] = (Target[p][k] - Output[p][k]) \* Output[p][k] \* (1.0 - Output[p][k]) ; /\* Sigmoidal Outputs, SSE \*/

/\* DeltaO[k] = Target[p][k] - Output[p][k]; Sigmoidal Outputs, Cross-Entropy Error \*/

/\* DeltaO[k] = Target[p][k] - Output[p][k]; Linear Outputs, SSE \*/

}

for( j = 1 ; j <= NumHidden ; j++ ) { /\* 'back-propagate' errors to hidden layer \*/

SumDOW[j] = 0.0 ;

for( k = 1 ; k <= NumOutput ; k++ ) {

SumDOW[j] += WeightHO[j][k] \* DeltaO[k] ;

}

DeltaH[j] = SumDOW[j] \* Hidden[p][j] \* (1.0 - Hidden[p][j]) ;

}

for( j = 1 ; j <= NumHidden ; j++ ) { /\* update weights WeightIH \*/

DeltaWeightIH[0][j] = eta \* DeltaH[j] + alpha \* DeltaWeightIH[0][j] ;

WeightIH[0][j] += DeltaWeightIH[0][j] ;

for( i = 1 ; i <= NumInput ; i++ ) {

DeltaWeightIH[i][j] = eta \* Input[p][i] \* DeltaH[j] + alpha \* DeltaWeightIH[i][j];

WeightIH[i][j] += DeltaWeightIH[i][j] ;

}

}

for( k = 1 ; k <= NumOutput ; k ++ ) { /\* update weights WeightHO \*/

DeltaWeightHO[0][k] = eta \* DeltaO[k] + alpha \* DeltaWeightHO[0][k] ;

WeightHO[0][k] += DeltaWeightHO[0][k] ;

for( j = 1 ; j <= NumHidden ; j++ ) {

DeltaWeightHO[j][k] = eta \* Hidden[p][j] \* DeltaO[k] + alpha \* DeltaWeightHO[j][k] ;

WeightHO[j][k] += DeltaWeightHO[j][k] ;

}

}

}

if( epoch%100 == 0 ) fprintf(stdout, "\nEpoch %-5d : Error = %f", epoch, Error) ;

if( Error < 0.0004 ) break ; /\* stop learning when 'near enough' \*/

}

fprintf(stdout, "\n\nNETWORK DATA - EPOCH %d\n\nPat\t", epoch) ; /\* print network outputs \*/

for( i = 1 ; i <= NumInput ; i++ ) {

fprintf(stdout, "Input%-4d\t", i) ;

}

for( k = 1 ; k <= NumOutput ; k++ ) {

fprintf(stdout, "Target%-4d\tOutput%-4d\t", k, k) ;

}

for( p = 1 ; p <= NumPattern ; p++ ) {

fprintf(stdout, "\n%d\t", p) ;

for( i = 1 ; i <= NumInput ; i++ ) {

fprintf(stdout, "%f\t", Input[p][i]) ;

}

for( k = 1 ; k <= NumOutput ; k++ ) {

fprintf(stdout, "%f\t%f\t", Target[p][k], Output[p][k]) ;

}

}

fprintf(stdout, "\n\nGoodbye!\n\n") ;

return 1 ;

}

**5. RESULTS**

**5.1 OUTPUT SCREENS**

The following are the output screens of the Back Propagation algorithm. Fig 5.1 shows that random values are assigned for the weights of each input. Fig 5.2 shows the error occurred for each epoch. Fig 5.3 shows target Vs observed output for two inputs values. Fig 5.4 shows target Vs observed output for three inputs values Fig 5.5 shows target Vs observed output for four inputs values.

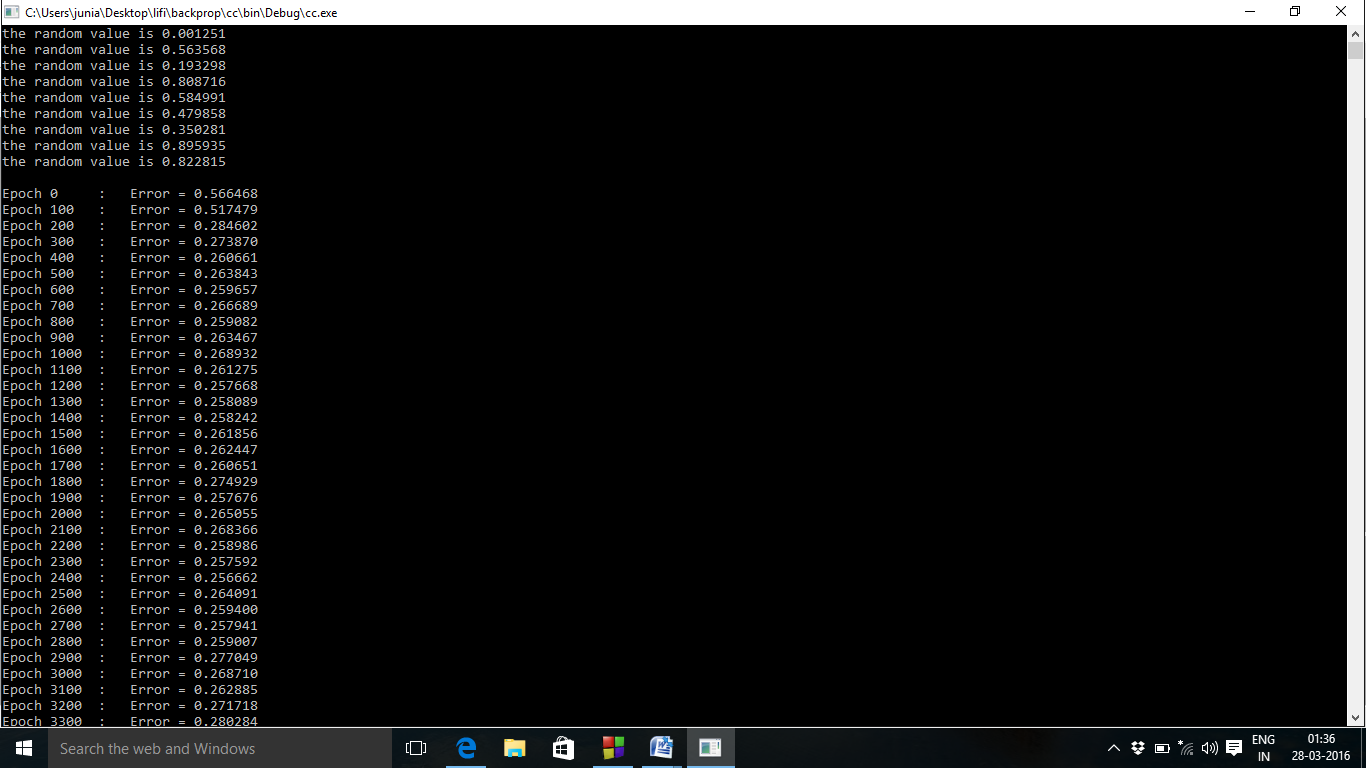


Fig 5.1: RANDOM VALUES ARE GENERATED FOR WEIGHTS

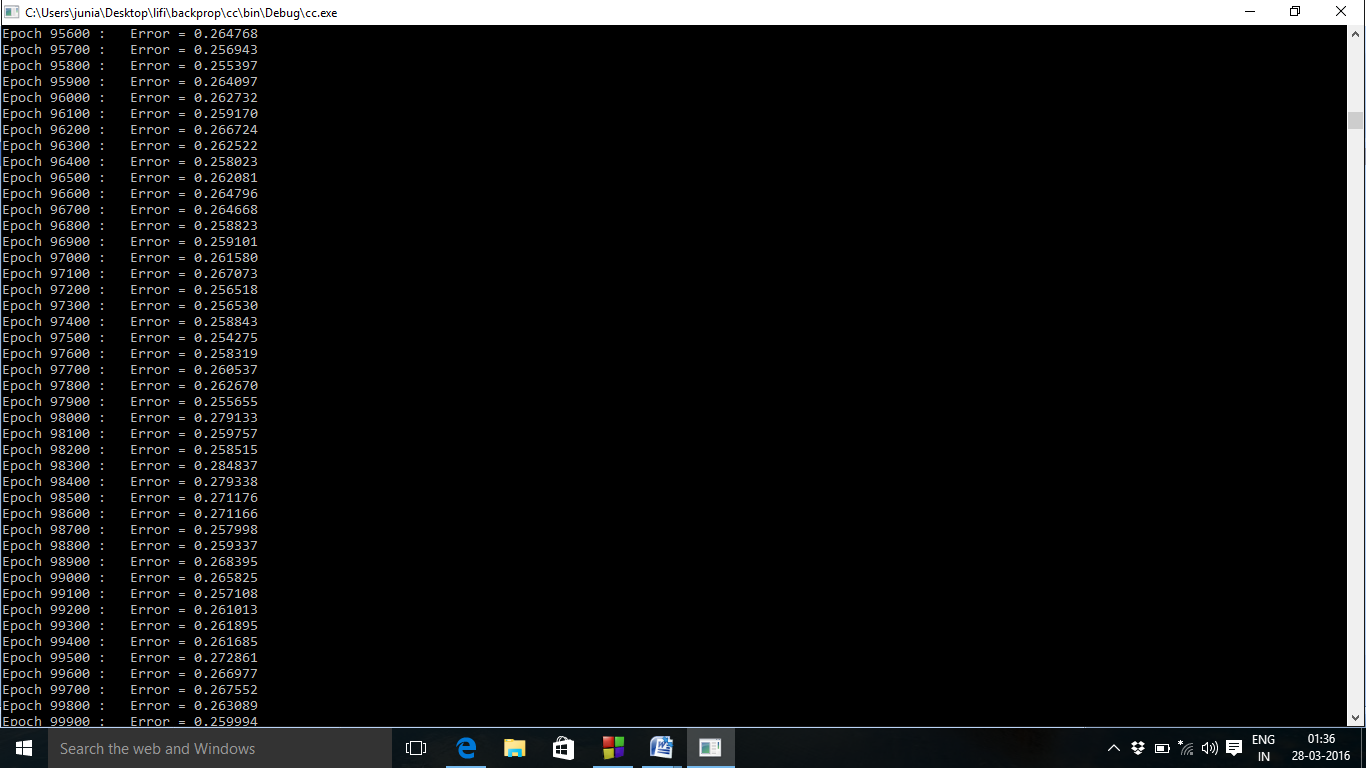


Fig 5.2: BACK PROPAGATING THE ERRORS FOR EACH EPOCH.

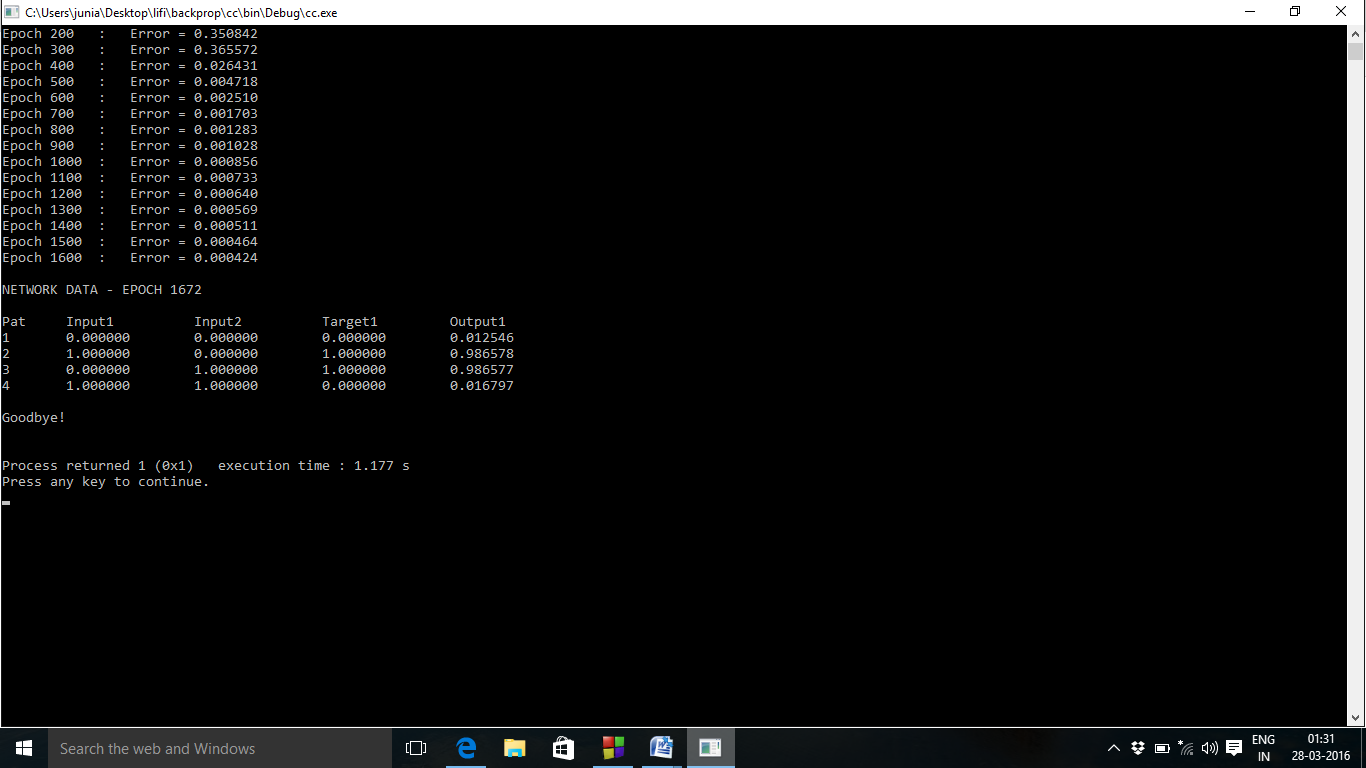


Fig 5.3: TARGET VS OBSERVED OUTPUT FOR TWO INPUTS VALUES

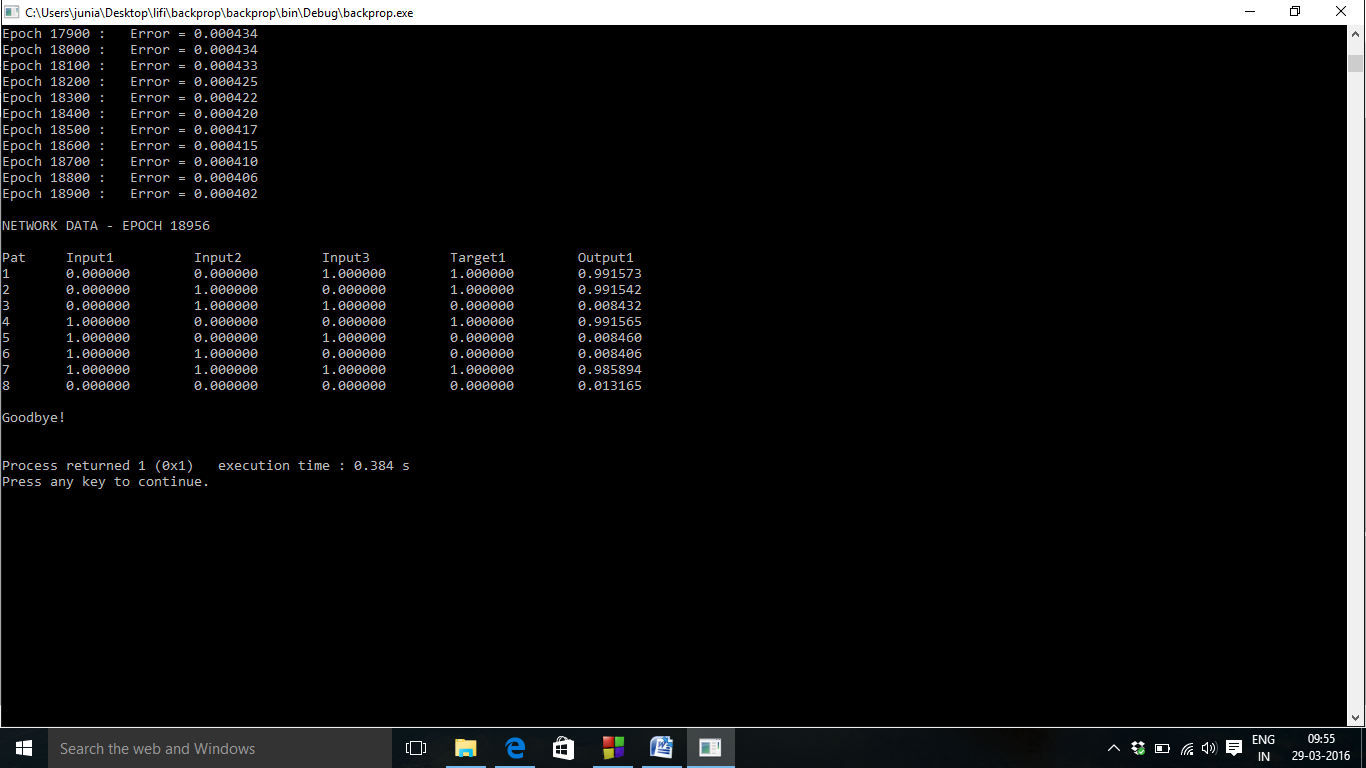


Fig 5.4: TARGET VS OBSERVED OUTPUT FOR THREE INPUTS VALUES

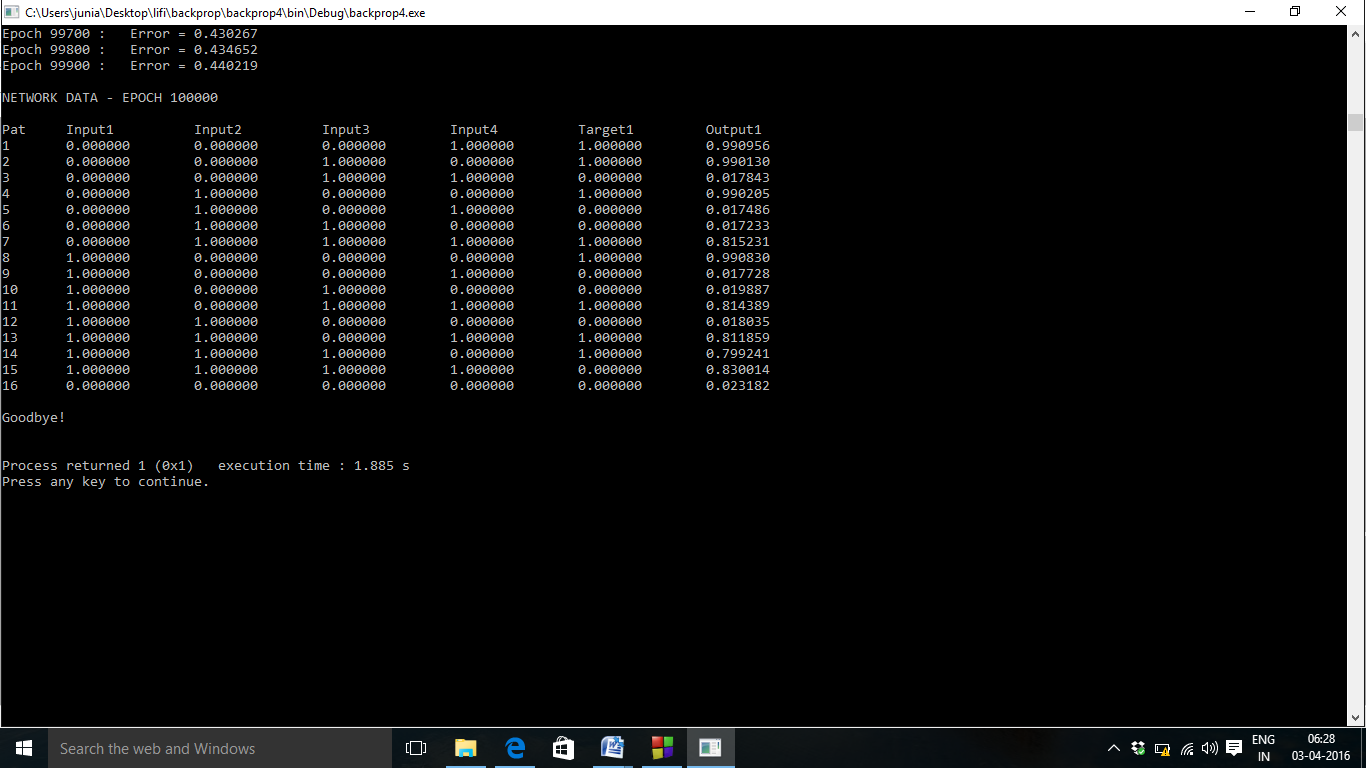


Fig 5.5: TARGET VS OBSERVED OUTPUT FOR FOUR INPUTS VALUES

**5.2 OBSERVATIONS**

Graphs are drawn representing Target Vs Observed values for two, three, four input values shown the figures 5.6, 5.7, 5.8 respectively. The X-axis represents the pattern number and the Y-axis represents the range Target Vs Observed values.

TABLE:

|  |  |  |  |
| --- | --- | --- | --- |
| Input 1 | Input 2 | Target | Output |
| 0.000000 | 0.000000 | 0.000000 | 0.012546 |
| 1.000000 | 0.000000 | 1.000000 | 0.986578 |
| 0.000000 | 1.000000 | 1.000000 | 0.986577 |
| 1.000000 | 1.000000 | 0.000000 | 0.016797 |

Table no: 5.1

GRAPH:

Fig.5.6: TARGET VS OBSERVED OUTPUT FOR TWO INPUT VALUES

TABLE:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input 1 | Input 2 | Input 3 | Target | Output |
| 0.000000 | 0.000000 | 1.000000 | 1.000000 | 0.991573 |
| 0.000000 | 1.000000 | 0.000000 | 1.000000 | 0.991542 |
| 0.000000 | 1.000000 | 1.000000 | 0.000000 | 0.008432 |
| 1.000000 | 0.000000 | 0.000000 | 1.000000 | 0.991565 |
| 1.000000 | 0.000000 | 1.000000 | 0.000000 | 0.008460 |
| 1.000000 | 1.000000 | 0.000000 | 0.000000 | 0.008406 |
| 1.000000 | 1.000000 | 1.000000 | 1.000000 | 0.985894 |
| 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.013165 |

Table no: 5.2

GRAPH:

Fig 5.7: TARGET VS OBSERVED OUTPUT FOR THREE INPUTS VALUES

TABLE:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input 1 | Input 2 | Input 3 | Input 4 | Target | Output |
| 0.000000 | 0.000000 | 0.000000 | 1.000000 | 1.000000 | 0.990956 |
| 0.000000 | 0.000000 | 1.000000 | 0.000000 | 1.000000 | 0.990130 |
| 0.000000 | 0.000000 | 1.000000 | 1.000000 | 0.000000 | 0.017843 |
| 0.000000 | 1.000000 | 0.000000 | 0.000000 | 1.000000 | 0.990205 |
| 0.000000 | 1.000000 | 0.000000 | 1.000000 | 0.000000 | 0.017486 |
| 0.000000 | 1.000000 | 1.000000 | 0.000000 | 0.000000 | 0.017233 |
| 0.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 0.815231 |
| 1.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 0.990830 |
| 1.000000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 0.017728 |
| 1.000000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 0.019887 |
| 1.000000 | 0.000000 | 1.000000 | 1.000000 | 1.000000 | 0.814389 |
| 1.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.018035 |
| 1.000000 | 1.000000 | 0.000000 | 1.000000 | 1.000000 | 0.811859 |
| 1.000000 | 1.000000 | 1.000000 | 0.000000 | 1.000000 | 0.799241 |
| 1.000000 | 1.000000 | 1.000000 | 1.000000 | 0.000000 | 0.830014 |
| 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.023182 |

Table no: 5.3

GRAPH:

Fig 5.8: TARGET VS OBSERVED OUTPUT FOR FOUR INPUTS VALUES

**6. CONCLUSION**

Using the Back Propagation Algorithm, we have successfully made the machine i.e the computer to learn & predict values for an XoR function for any number of inputs. Errors were back propagated from each layer to the previous layer and the weights were changed so as to minimise the errors. When the output of two epochs matches, the back propagation is stopped. Also, we have gathered from our calculation that, as the number of inputs increases, the error value is greatly reduced and the observed value is very close to the target values.

**REFERENCES**

1. http://www.cs.bham.ac.uk/~jxb/INC/nn.html

2. http://www.philbrierley.com/

3. http://neuroph.sourceforge.net/tutorials/TeachingAssistant/TeachingAssistantEvaluation/

4. <http://natureofcode.com/book/chapter-10-neural-networks/>

5. IEEE Paper - http://ethesis.nitrkl.ac.in/245/1/10502014.pdf

6. Data Mining: Concepts and Techniques by Han J & KamberM

7. Artificial Intelligence: A New Synthesis by Elsevier Stuart Russell, Peter Norvig.