2014

Institute of Information Technology & Management

Lab Manual on Soft Computing [IT-802]

Ms. Neha Sexana

INSTITUTE OF INFORMATION TECHNOLOGY & MANAGEMENT , GWALIOR

01-Feb-2014

SOFTWARE REQUIREMENT :

1. Turbo C++ IDE (TurboC3)
2. Borland Turbo C++ (Version 4.5)
3. JAVA

REFERENCE BOOKS :

1. S.N. Shivnandam, “Principle of soft computing”, Wiley.

2. S. Rajshekaran and G.A.V. Pai, “Neural Network , Fuzzy logic And Genetic Algorithm”, PHI.

3. Jack M. Zurada, “Introduction to Artificial Neural Network System” JAico Publication.

4. Simon Haykins, “Neural Network- A Comprehensive Foudation”

LIST OF PROGRAMS

1. **Write A Program For Implementing Linear Saturating Function.**
2. **Study And Analysis Of Art Model.**
3. **Write A Program For Error Back Propagation Algorithm (Ebpa) Learning.**
4. **Study And Analysis Of CPN**
5. **Study And Analysis Of Genetic Algorithm Life Cycle.**
6. **Study And Analysis Of Fuzzy Vs Crisp Logic.**
7. **Write A Program Of Perceptron Training Algorithm.**
8. **Write A Program To Implement Hebb's Rule**
9. **Write A Program To Implement Of Delta Rule**
10. **Write A Program For Back Propagation Algorithm**
11. **Write A Program To Implement Logic Gates**

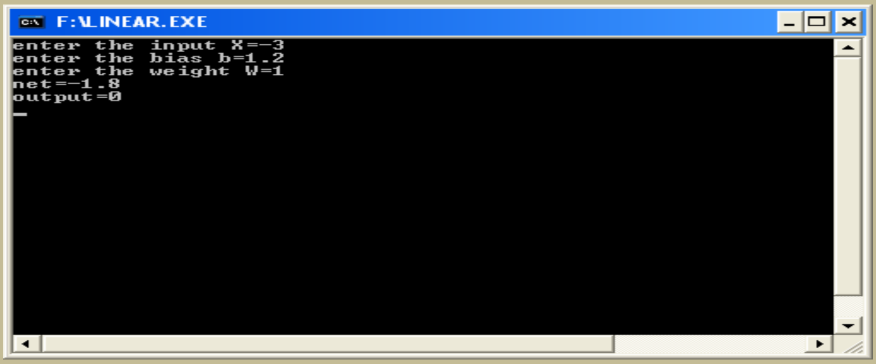
EXPERIMENT: 1

DESCRIPTION: WAP FOR IMPLEMENTING LINEAR SATURATING FUNCTION.

```
 #include<iostream.h>
#include<conio.h>
void main()
{
clrscr();
float x,b,w,net;
float out;
cout<<"enter the input X=";
cin>>x;
cout<<"enter the bias b=";
cin>>b;
cout<<"enter the weight W=";
cin>>w;
net=(w*x+b);
cout<<"net="<<net<<endl;
if(net<0)
{
out=0;
}
else
if((net>=0)&&(net<=1))
{
out=net;
}
else
out=1;
cout<<"output="<<out<<endl;
getch();
}
```

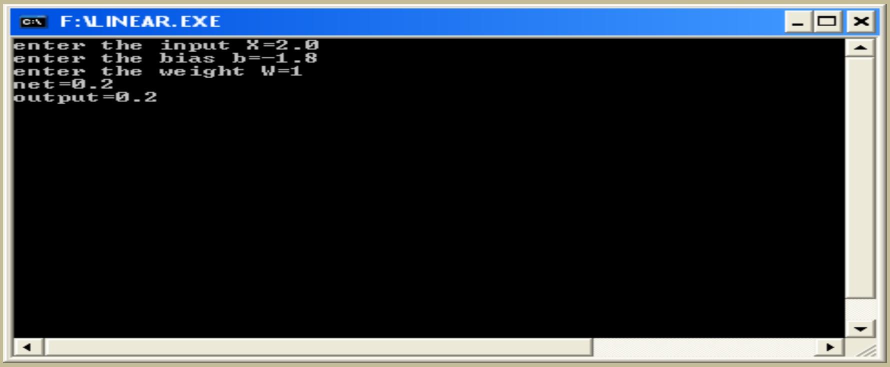
OUTPUT

When, net<0 :-



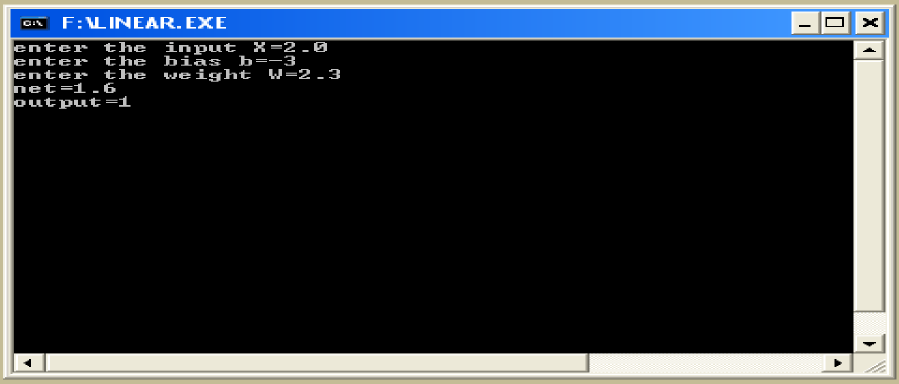
```
enter the input X=-3
enter the bias b=1.2
enter the weight W=1
net=-1.8
output=0
_
```

When, ((net>=0)&&(net<=1)) :-



```
enter the input X=2.0
enter the bias b=-1.8
enter the weight W=1
net=0.2
output=0.2
```

When, net>1 :-



```
enter the input X=2.0
enter the bias b=-3
enter the weight W=2.3
net=1.6
output=1
```

EXPERIMENT: 2

DESCRIPTION: STUDY AND ANALYSIS OF ART MODEL.

Adaptive Resonance Theory (ART) :-

ART (Adaptive Resonance Theory) models are neural networks that performs clustering and can allow the no. of cluster to vary with problem size.

The difference between ART and other clustering model is that ART allows the user to control the degree of similarity between no. of same cluster by means of a user define constants called **vigilance parameter**.

The 1st version of ART was ART1, developed by Carpenter and Grossberg in 1988.

Block Diagram of ART ->

It consist of two layers of neurons labeled "Comparision" and "Recognition". Gain1, Gain2 and reset provide control functions needed for training and classification.

In a real world, the network will be exposed to constantly changing environment i.e. it may never see the same training vector twice, in such cases any of the conventional ANN seems till now will learn nothing.

ART network is a vector classifier. It accept the inputs and classifies it on the basis of already stored patterns to which it most resembles. The classification is done by the recognition layer.

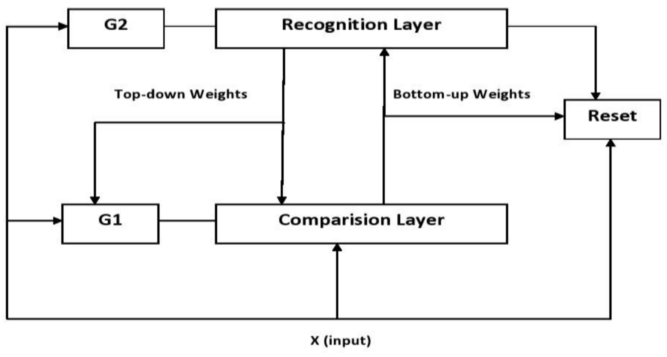
If the input vector does not matches any already stored pattern, a new pattern or category is created which matches the input vector.

No stored pattern is ever modified, if it does not match the current input within the vigilance.

1. **Comparision Layer** -> It works on 2/3 rule. This layer recieves the binary layer input vector represented by X and initially passes it trough unchanged to become the vector C.

   For one iteration C=X, in later case binary vector R is produce from Recognition layer. Each neuron in the comparision layer recieves three binary inputs.

   a. The component Xi from the input vector X.
   b. The feedback signal Pj i.e. the weighted sum of recognition layer output.
   c. The input from the gain signal i.e. Gain1 (G1).



BLOCK DIAGRAM OF ART

OR of X Component	OR of R Component	G2
0	0	0
1	0	1
1	1	0
0	1	0

2. **Recognition Layer** -> The recognition layer serves to classify the input vector. Each recognition layer neuron has an associated weight vector bj only the neuron with the weight vector best matching the input vectors, fire all other neurons are inhibited.

   The weights in the recognition layer make a stored pattern for a category of input output vector, these weights are real number.

   The binary version of same pattern is stored in the corresponding set of weights in the comparision layer.

   Operation perform in Recognition Layer -> Each recognition layer neuron computes a dot(.) product between its weights and incoming vector C. The neuron which has the highest dot product will win the competition and will fire, the response of recognition layer is in the fashion "winner takes all".

   a. **GAIN2 (G2)** -> It is set to 1 if one of the component of input vector X is 1. G2 is logically "OR" of the component of X.

   b. **GAIN1 (G1)** -> Like GAIN2, the output of GAIN1 is 1. If any component of binary input vector is 1.

      But if any component of R is 1 then, G1 is force to become 0.

c. **Reset Signal** -> The Reset modules measures the similarity between the vector X and C. Generally, this similarity is the ratio of 1's in the vector X.

If this ratio is below the vigilance parameter, a reset signal is issue and inhibits the neuron of recognition layer.

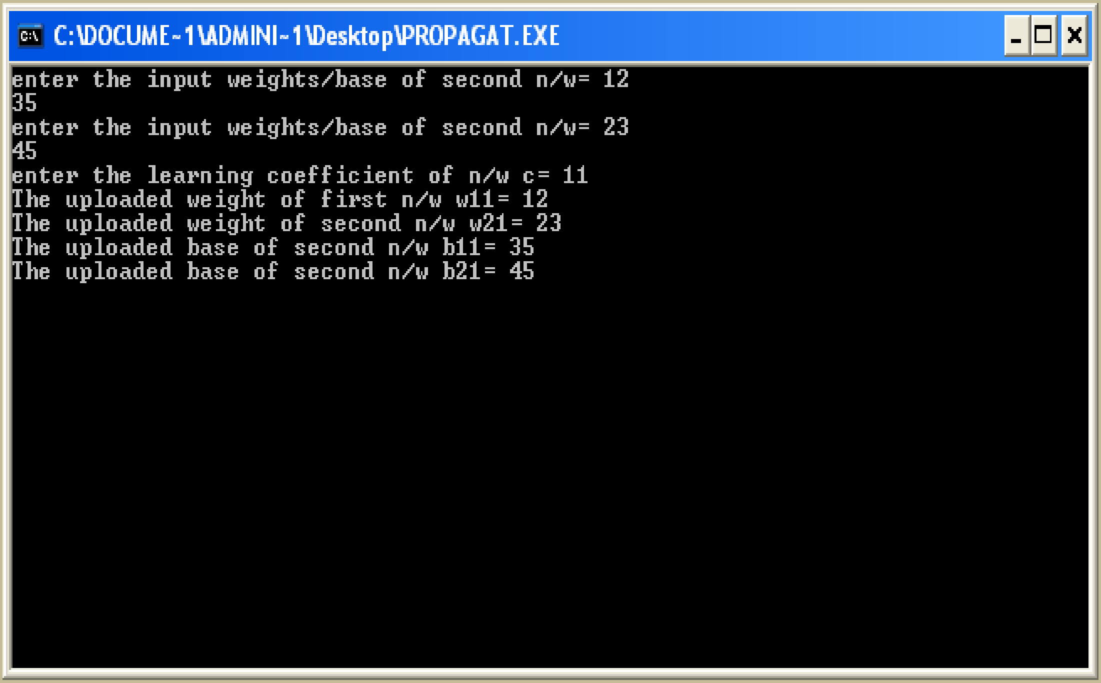
The vigilance parameter p is closer to 1 for accuracy & its value is 0.9

EXPERIMENT: 3

DESCRIPTION: WAP FOR ERROR BACK PROPAGATION ALGORITHM (EBPA) LEARNING.

```
#include<conio.h>
#include<iostream.h>
#include<math.h>
void main()
{
clrscr();
float l,c,s1,n1,n2,w10,b10,w20,b20,w11,b11,w21,b21,p,t,a0=-1,a1,a2,e,s2;
cout<<"enter the input weights/base of second n/w= ";
cin>>w10>>b10;
cout<<"enter the input weights/base of second n/w= ";
cin>>w20>>b20;
cout<<"enter the learning coefficient of n/w c= ";
cin>>c;
/* Step1:Propagation of signal through n/w */
n1=w10*p+b10;
a1=tanh(n1);
n2=w20*a1+b20;
a2=tanh(n2);
e=(t-a2);
/* Back Propagation of Sensitivities */
s2=-2*(1-a2*a2)*e;
s1=(1-a1*a1)*w20*s2;
/* Updation of weights and bases */
w21=w20-(c*s2*a1);
w11=w10-(c*s1*a0);
b21=b20-(c*s2);
b11=b10-(c*s1);
cout<<"The uploaded weight of first n/w w11= "<<w11;
cout<<"\n"<<"The uploaded weight of second n/w w21= "<<w21;
cout<<"\n"<<"The uploaded base of second n/w b11= "<<b11;
cout<<"\n"<<"The uploaded base of second n/w b21= "<<b21;
getch();
}
```

OUTPUT



C:\DOCUME~1\ADMINI~1\Desktop\PROPAGAT.EXE

```
enter the input weights/base of second n/w= 12
35
enter the input weights/base of second n/w= 23
45
enter the learning coefficient of n/w c= 11
The uploaded weight of first n/w w11= 12
The uploaded weight of second n/w w21= 23
The uploaded base of second n/w b11= 35
The uploaded base of second n/w b21= 45
```

EXPERIMENT: 4

<u>DESCRIPTION:</u> STUDY AND ANALYSIS OF CPN.

<u>Counter Propagation Network (CPN)</u> :-

Counter Propagation Network (CPN) is not as general as Back Propagation. But it provides a solution for those application that cannot tolerate long training time.

It is a combination of two well known algorithm i.e. Self Organizing Map of Kohonen and Grossberg outstar. The training process associates the input vector with the corresponding output vector.

These vectors may be binary or continuous. It has the generalization capability. The generalization capability of network allows it to produce a correct output even when it is given an input vector i.e. partially incomplete or partially incorrect.

CPN is used to compress the data before sending and decompress the data before receiving. With the help of these network we can compress audio data upto 100:1 and video data upto 10:1.

<u>Architecture Of CPN</u> ->

It consist of one input layer, one Kohonen layer and one Grossberg layer.

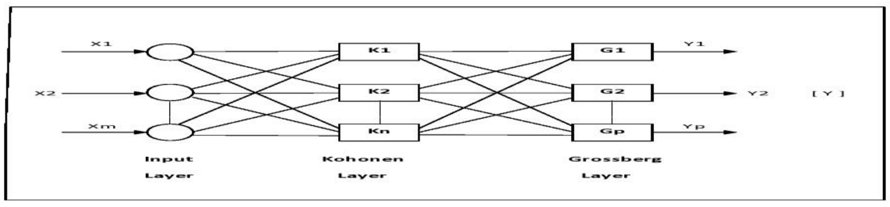
All the units of input layer are fully interconnected by weights to the units of Kohonen layer.

Similarly, all the units of Kohonen layer are fully interconnected by weights to the Grossberg layer.

It works in two mode.

1. **<u>Normal Mode</u> ->**
   When we apply the input vector and get the desired output.

2. **<u>Training Mode</u> ->**
   Apply input vector and modify the network weights so that, we get the desired output.



<u>Advantages of CPN</u> ->

1. It is simple and it forms a good statistical model of its input vector environment.

2. CPN trains rapidly. If appropriately applied, it can safe large amount of computer time.

3. It is also useful for rapid prototyping of system, where the greater accuracy of Back Propagation makes it the method of choice in the final version. But a quick approximation is important so, CPN is more useful then Back Propagation.

EXPERIMENT: 5

<u>DESCRIPTION:</u> STUDY AND ANALYSIS OF GENETIC ALGORITHM LIFE CYCLE.

<u>Genetic Algorithm (GA)</u> :-

Genetic Algorithm is a search heuristic (experience) that follows the process of natural evolution.

This heuristic is used to generate useful solutions to optimization and search problems.

Genetic Algorithm belong to the larger class of evolutionary algorithm (EA) which generate solutions to optimization problems and using techniques inspired by natural evolution like – inheritance, mutation, selection, crossover.

Genetic Algorithm need design space to be converted into genetic space. Genetic Algorithm works with coding variables.

Genetic Algorithm uses population of point at one time in contrast to the single point approach. It means that genetic algorithm processes a number of designs at the same time.

The advantage of coding of variable is that coding discretizes the search space even though the function may be continuous.

Traditional optimization methods use transition rules that are deterministic in nature.

While genetic algorithm uses randomize operators. Randomize operator improve the search space in an adaptive manner.

There are three important aspects of Genetic Algorithm are-

1. Definition of objective function.

2. Definition and implementation of genetic representation.

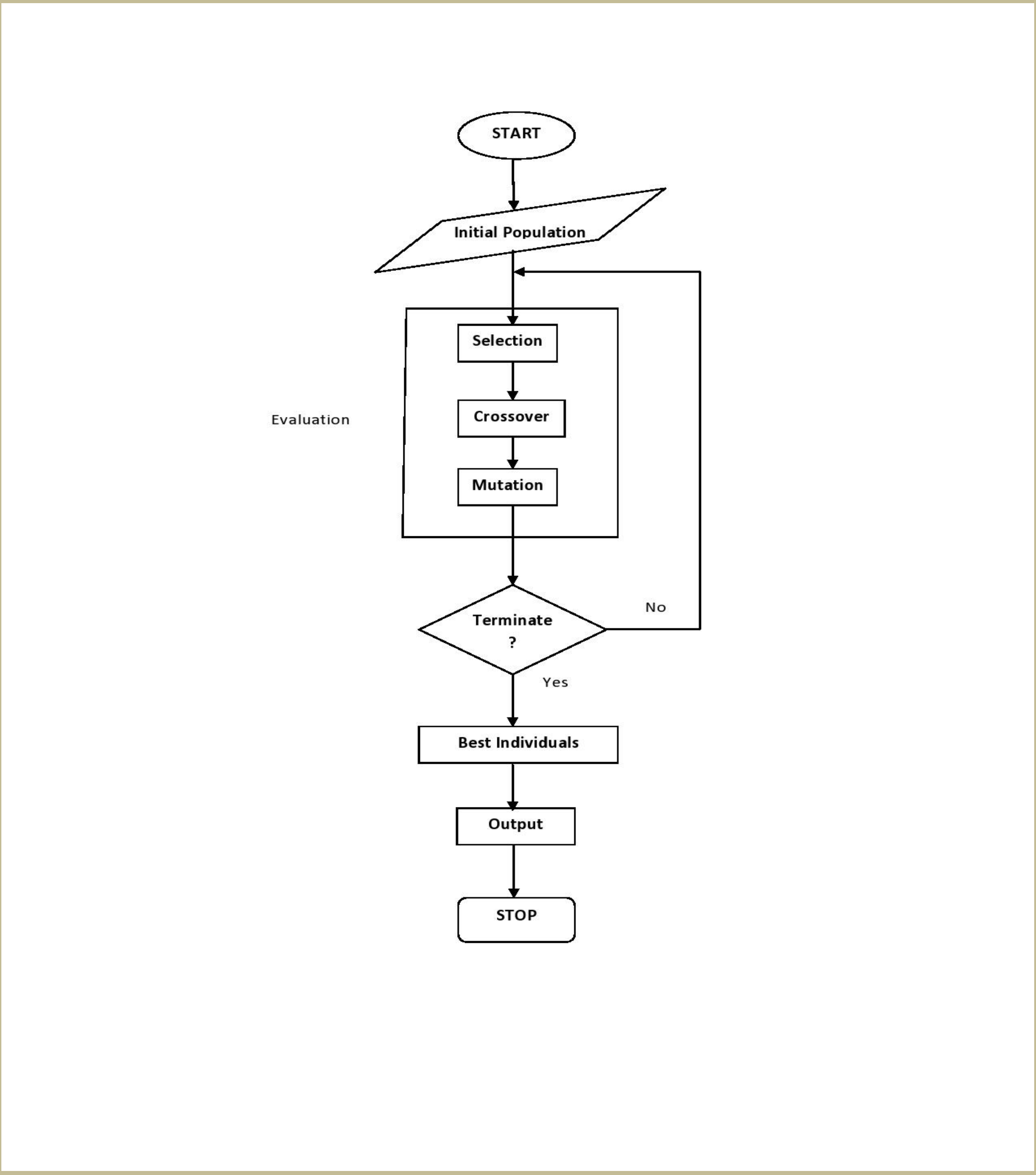
3. Definition and implementation of Genetic operators.

<u>Advantages of Genetic Algorithm (GA)</u> :-

1. It shows simplicity.
2. Ease of operation.
3. Minimal requirement.
4. Global perspective.

5. It does not guarantee to find global minimum solutions but acceptably good solutions to "acceptably quickly ".

<u>Flowchart of Genetic Algorithm (GA)</u> :-



Genetic Algorithm Steps :-

```
1.  BEGIN
2.  Create initial population ;
3.  Compute fitness of each individuals ;
4.  WHILE NOT finished DO Loop
5.  BEGIN
6.  Select individuals from old generation for mating ;
7.  Create offspring by applying crossover or mutation to the selected individuals ;
8.  Compute fitness of new individuals ;
9.   Kill old individuals to make a room for new chromosomes and insert offspring in
    the new
     generation ;
10. If population has converged
11. Then fitness=TRUE ;
12. END
13. END
```

EXPERIMENT: 6

DESCRIPTION: **STUDY AND ANALYSIS OF CPN.**

Counter Propagation Network (CPN) :-

Counter Propagation Network (CPN) is not as general as Back Propagation. But it provides a solution for those application that cannot tolerate long training time.

It is a combination of two well known algorithm i.e. Self Organizing Map of Kohonen and Grossberg outstar. The training process associates the input vector with the corresponding output vector.

These vectors may be binary or continuous. It has the generalization capability. The generalization capability of network allows it to produce a correct output even when it is given an input vector i.e. partially incomplete or partially incorrect.

CPN is used to compress the data before sending and decompress the data before receiving. With the help of these network we can compress audio data upto 100:1 and video data upto 10:1.

Architecture Of CPN ->

It consist of one input layer, one Kohonen layer and one Grossberg layer.

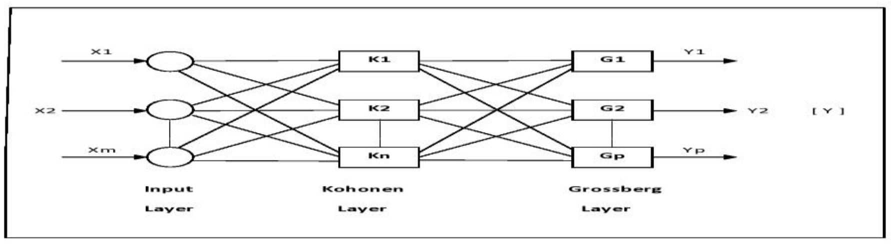
All the units of input layer are fully interconnected by weights to the units of Kohonen layer.

Similarly, all the units of Kohonen layer are fully interconnected by weights to the Grossberg layer.

It works in two mode.

1. **Normal Mode** ->
   When we apply the input vector and get the desired output.

3. **Training Mode** ->
   Apply input vector and modify the network weights so that, we get the desired output.



<u>Advantages of CPN</u> ->

4. It is simple and it forms a good statistical model of its input vector environment.

5. CPN trains rapidly. If appropriately applied, it can safe large amount of computer time.

6. It is also useful for rapid prototyping of system, where the greater accuracy of Back Propagation makes it the method of choice in the final version. But a quick approximation is important so, CPN is more useful then Back Propagation.

EXPERIMENT: 7

DESCRIPTION: **STUDY AND ANALYSIS OF FUZZY Vs CRISP LOGIC.**

FUZZY LOGIC :-

Fuzzy logic is a form of many valued Logic. It deals with reasoning i.e. approximate or inexact.

Fuzzy logic have been extended to handle the concept of partial truth where the truth value will be in the range between completely true and completely false.

Fuzzy logic began with the 1965 by Lotfi Zadeh. Fuzzy logic has been applied to many fields from control theory to artificial intelligence.

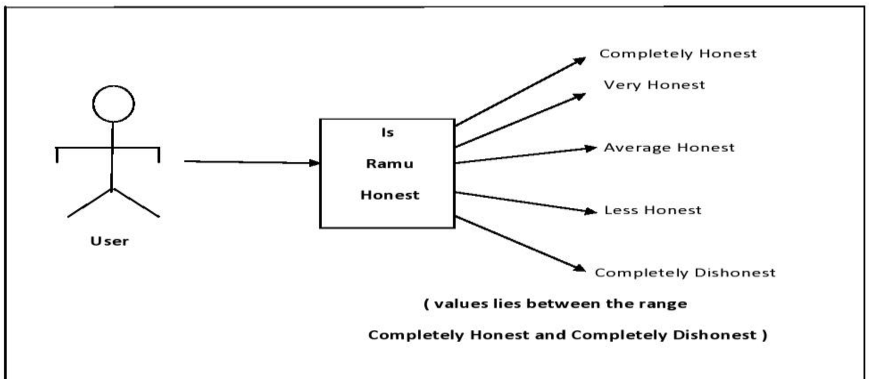


Fig: **FUZZY LOGIC**

<u>CRISP LOGIC</u> :-

In older days, Crisp logic were used to handle the problem of binary value ie. 0 and 1. Crisp logic is also known as traditional, conventional or binary logic.

Crisp logic have two valued logic first is true and other is false. Crisp logic is based on the reasoning which is exact and fixed. It is based on the logic of completely true and completely false.

We can defined completely true as one (1) and completely false as zero (0).

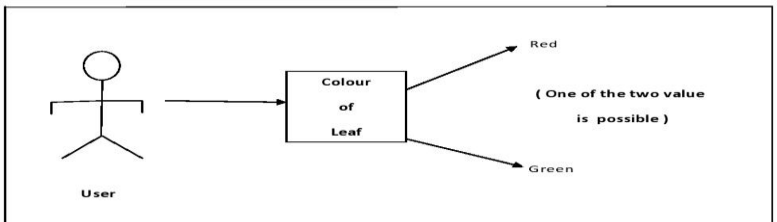


Fig: **<u>CRISP LOGIC</u>**

EXPERIMENT:7

DESCRIPTION: Write a program of Perceptron Training Algorithm.

Algorithm

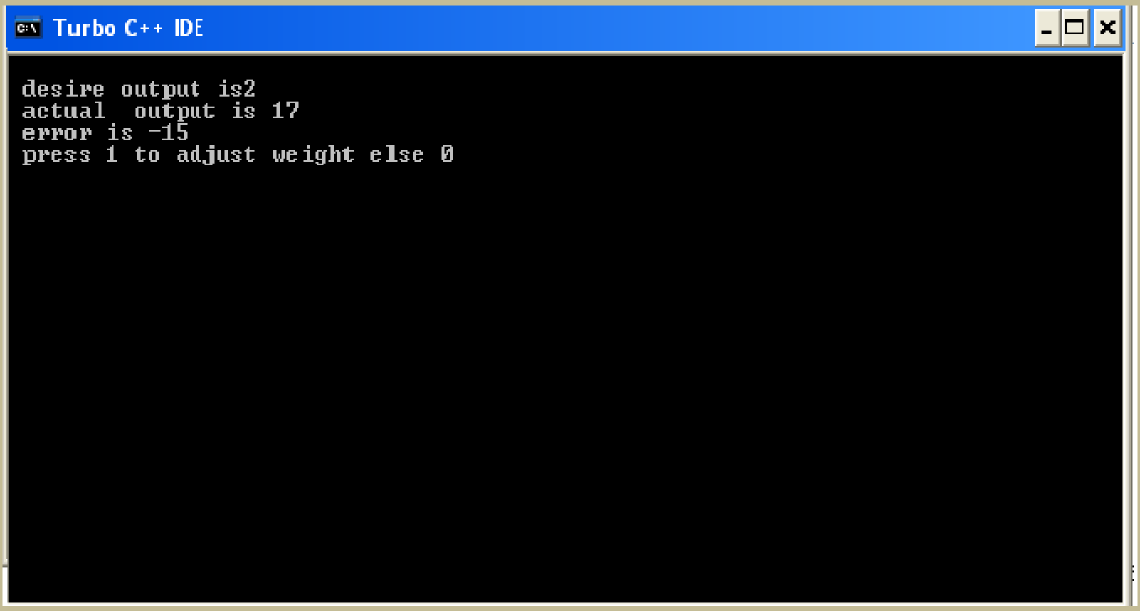
```
Start with a randomly chosen weight vector $w_0$;
Let k=1;
While these exists input vector that are misclassified by: $W_{k-1}$ do
Let i be a misclassified input vector
Let $X_k$=class(ij)ij, impling that $W_{k-1}.X_k<0$
Update the weight vector to $W_k= W_{k-1} + nX_k$;
increment k;
End while;
```

Program

```
#include<iostream.h>
#include<conio.h>

Void main( )
{
        clrscr( );
        int  in[3],d,w[3],a=0;
        for(int i=0;i<3,i++)
        {
                cout<<"\n initialize the weight vector w"<<i;
                cin>>w[i]
        }
        for(i=0;i<3:i++}
        {
                cout<<"\n enter the input vector i"<<i;
                cin>>in[i];
        }
        cout<<"\n enter the desined output";
        cin>>d;
        int ans=1;
        while(ans= = 1)
        {
        for (a= 0, i==0;i<3;i++)
        {
                        a = a + w[i] * in[i];
        }
        clrscr( );
```

```
          cout<<"\n desired output is"<<d;
          cout<<"\n actual  output is "<<a;
          int e;
          e=d-a;
          cout<<"\n error is "<<e;
          cout<<"\n press 1 to adjust weight else 0";
          cin>>ans;
          if (e<0)
          {
                    for(i=0;i<3;i++)
                    {
                              w[i]=w[i]-1;
                    }
          }
          else if (e>0)
          {
                    for(i=0;i<3:i++)
                    {
                              w[i]=w[i]+1;
                    }
          }
 getch( );
 }
```

OUTPUT:



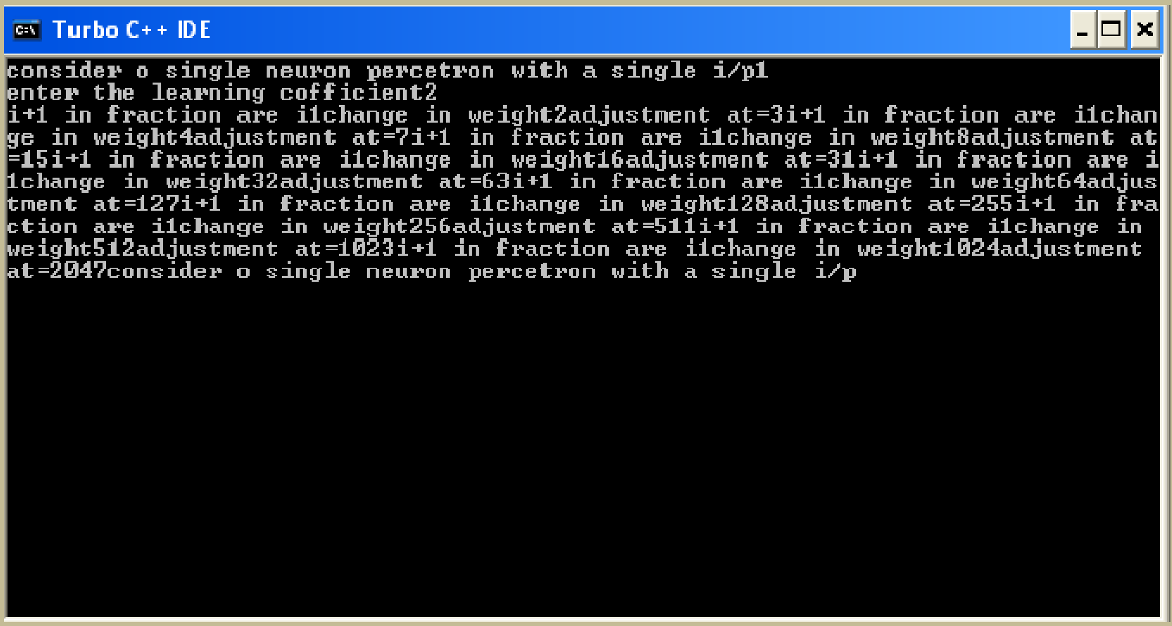
```
desire output is2
actual  output is 17
error is -15
press 1 to adjust weight else 0
```

EXPERIMENT:8

DESCRIPTION: Write a program to implement Hebb’s rule

```
#include<<iostream.h>>
#include<<conio.h>>
void main()
{
float n,w,t,net,div,a,al;
cout<<”consider o single neuron percetron with a single i/p”;
cin>>w;
cout<<”enter the learning cofficient”;
cin>>d;
for (i=0;i<10;i++)
{
net = x+w;
if(wt<0)
a=0;
else
a=1;
div=at+a+w;
w=w+div;
cout<<”i+1 in fraction are i”<<a<<”change in weight”<<dw<<”adjustment at=”<<w;
        }
}
```

OUTPUT:



Turbo C++ IDE

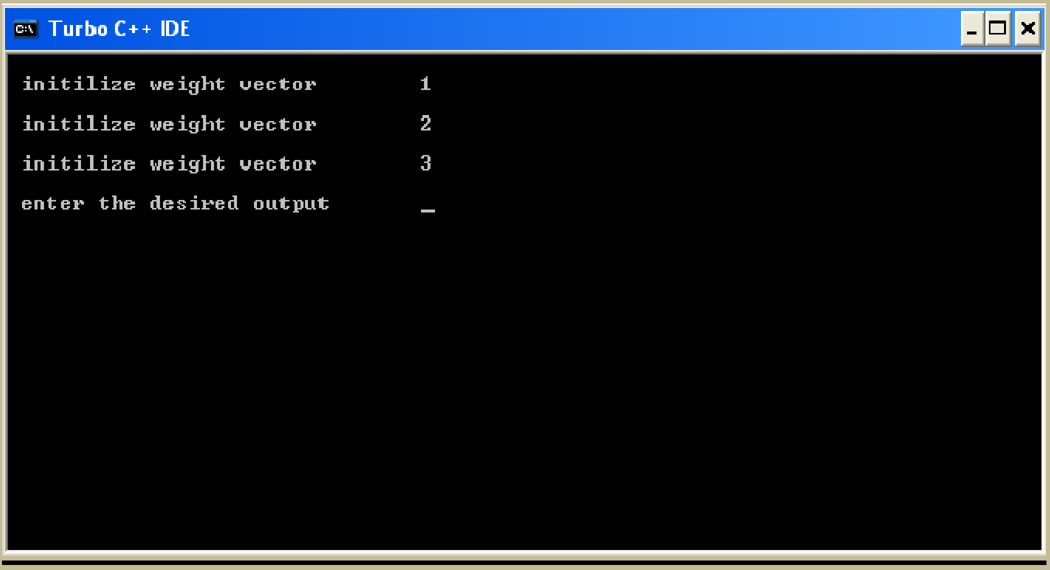
```
consider o single neuron percetron with a single i/p1
enter the learning cofficient2
i+1 in fraction are i1change in weight2adjustment at=3i+1 in fraction are i1chan
ge in weight4adjustment at=7i+1 in fraction are i1change in weight8adjustment at
=15i+1 in fraction are i1change in weight16adjustment at=31i+1 in fraction are i
1change in weight32adjustment at=63i+1 in fraction are i1change in weight64adjus
tment at=127i+1 in fraction are i1change in weight128adjustment at=255i+1 in fra
ction are i1change in weight256adjustment at=511i+1 in fraction are i1change in
weight512adjustment at=1023i+1 in fraction are i1change in weight1024adjustment
at=2047consider o single neuron percetron with a single i/p
```

EXPERIMENT :9

DESCRIPTION: Write a program to implement of delta rule.

```
#include<<iostream.h>>
#include<<conio.h>>
void main()
{
        clrscr( );
        float input[3],d,weight[3],delta;
        for(int i=0;i < 3 ; i++)
        {
                cout<<"\n initilize weight vector "<<i<<"\t";
                cin>>input[i];
        }
        cout<<""\n enter the desired output\t";
        cin>>d;
        do
        {
                del=d-a;
                if(del<0)
                        for(i=0 ;i<3 ;i++)
                        w[i]=w[i]-input[i];
                else if(del>0)
                        for(i=0;i<3;i++)
                        weight[i]=weight[i]+input[i];
                        for(i=0;i<3;i++)
                        {
                                val[i]=del*input[i];
                                weight[+1]=weight[i]+val[i];
                        }
                        cout<<"\value of delta is "<<del;
                        cout<<"\n weight have been adjusted";
        }while(del ≠ 0)
        if(del=0)
        cout<<"\n output is correct";
}
```

OUTPUT



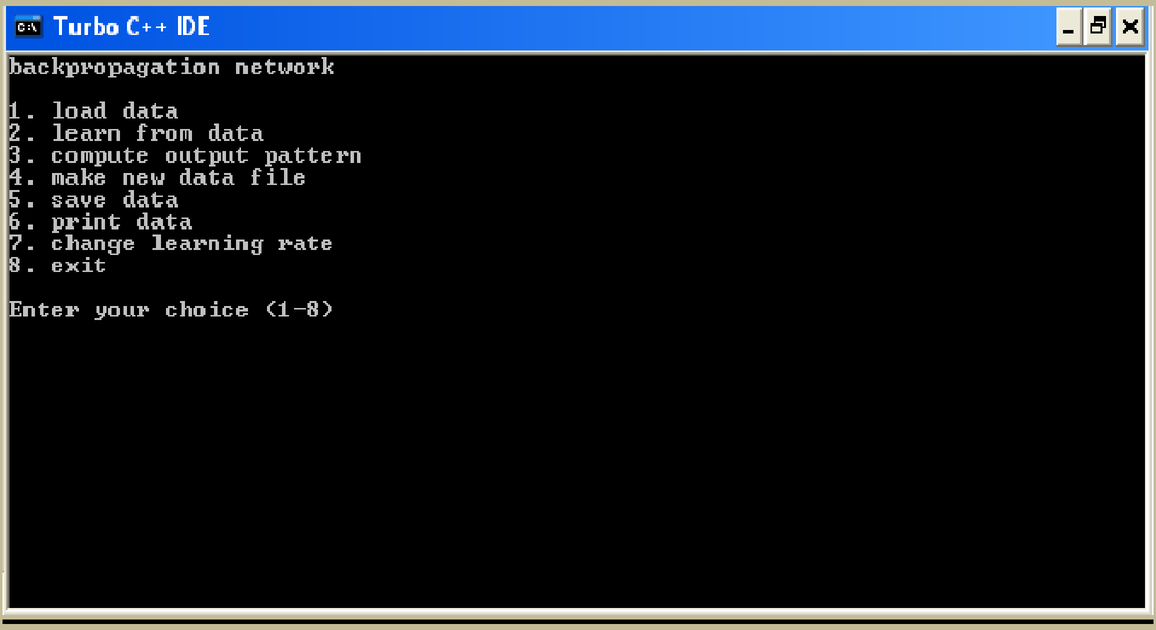
```
initilize weight vector         1
initilize weight vector         2
initilize weight vector         3
enter the desired output        _
```

EXPRIMENT :10

DESCRIPTION: Write a program for Back Propagation Algorithm

```
# include <iostream.h>
#include <conio.h>
void main ()
{
int i ;
float delta, com, coeff = 0.1;
struct input
{
                float val,out,wo, wi;
                int top;
}       s[3] ;

cout<< “\n Enter the i/p value to target o/p” << “\t”;
for (i=0; i<3 ; i++)
cin>> s [i], val>> s[i], top);
i = 0;
do
{
if (i =  = 0)
{
        W0 = -1.0;
        W1 = -0.3;
}
else
{
        W0 =  del [i - 1], W0 ;
        W1 =  del [i - 1] , Wi ;
}
del [i]. aop = w0 + (wi * del [i]. val);

del [i].out = del [i]. aop);

delta = (top – del [i]. out) * del [i].out * (1 – del [i].out);

corr = coeff * delta * del [i].[out];

del [i].w0 = w1 + corr;

del [i]. w1 = w1 + corr;

i++;

}While ( i ! = 3)

cout<< “VALUE”<<”Target”<<”Actual”<<”w0” <<”w1”<<’\n;

for (i=0; i=3; i++)
{
        cout<< s [i].val<< s[i].top<<s[i].out << s[i]. w0<< s[i]. w1;
        cout<< “\n”;
}
getch ();
}
```

OUTPUT



```
backpropagation network

1. load data
2. learn from data
3. compute output pattern
4. make new data file
5. save data
6. print data
7. change learning rate
8. exit

Enter your choice (1-8)
```

EXPERIMENT :11

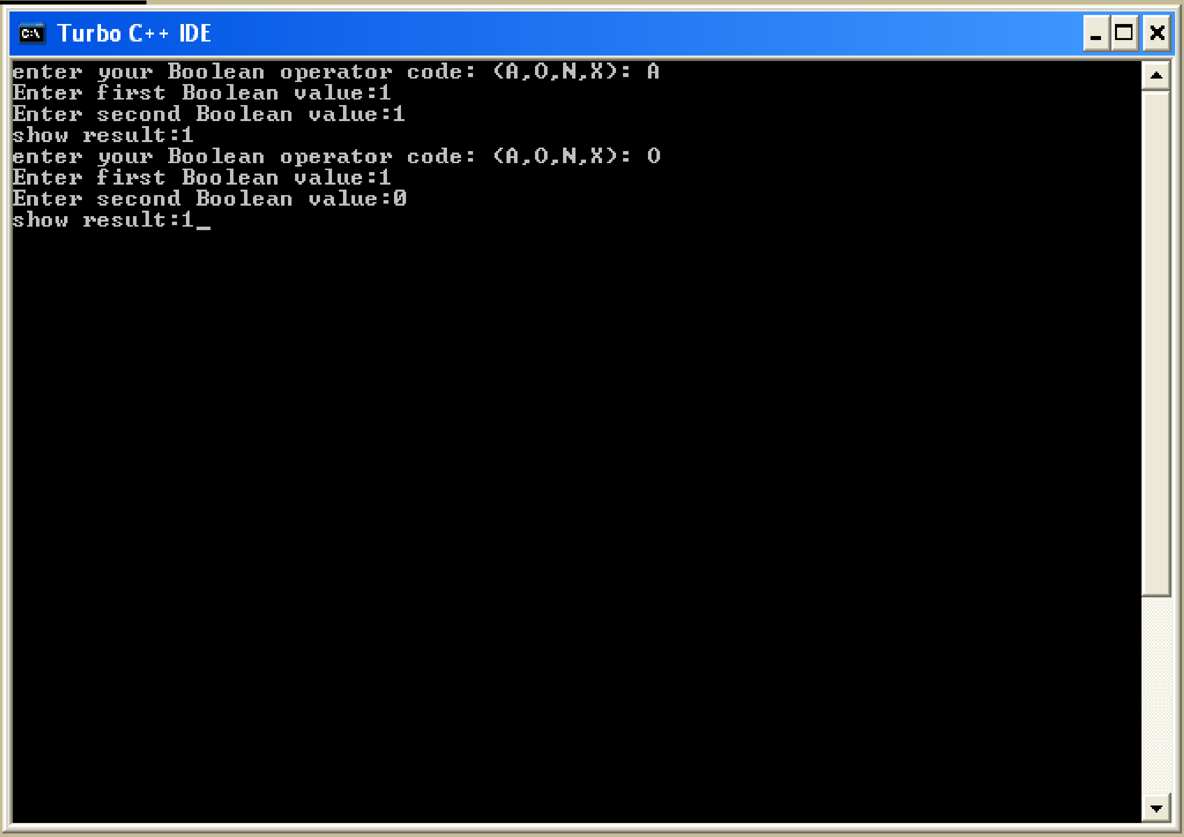
DESCRIPTION: Write a program to implement logic gates.

```
#include <iostream>
int main()
{
	char menu;		//Menu control variable
	int result;		//final output variable
	int dataValue1;
	int dataValue2;
	cout << "enter your Boolean operator code: (A,O,N,X): ";
	cin >> menu;
	switch (menu) //Menu control variable
	{
		case 'A':
			cout << "Enter first Boolean value:";
			cin >> dataValue1;
			cout << "Enter second Boolean value:";
			cin >> dataValue2;
			if(dataValue1 == 1 && dataValue2 == 1)
```

```
                {
                        result = 1;
                }
                else
                {
                        result = 0;
                 }
                cout << "show result:" << result;
                break;
        case 'O':
                cout << "Enter first Boolean value:";
                cin >> dataValue1;
                cout << "Enter second Boolean value:";
                cin >> dataValue2;
                if(dataValue1 == 1 || dataValue2 == 1)
                {
                         result = 1;
                }else
                {
                        result = 0;
                 }
                cout << "show result:" << result;
                break;
        case 'N':
                cout << "Enter first Boolean value:";
                cin >> dataValue1;
                result = !dataValue1;
                cout << "show result:" << result;
                break;
        case 'X':
                cout << "Enter first Boolean value:";
                cin >> dataValue1;
                cout << "Enter second Boolean value:";
                cin >> dataValue2;
                if(dataValue1 = !dataValue1)
                {
                         result = 1;
                }else
                {
                        result = 0;
                }
                cout << "show result:" << result;
                break;
        default:
                 result = 0;
                 break;
}//end switch

cin.ignore(2);
return 0;
```

```
}//end main
```

OUTPUT



```
enter your Boolean operator code: (A,O,N,X): A
Enter first Boolean value:1
Enter second Boolean value:1
show result:1
enter your Boolean operator code: (A,O,N,X): O
Enter first Boolean value:1
Enter second Boolean value:0
show result:1_
```