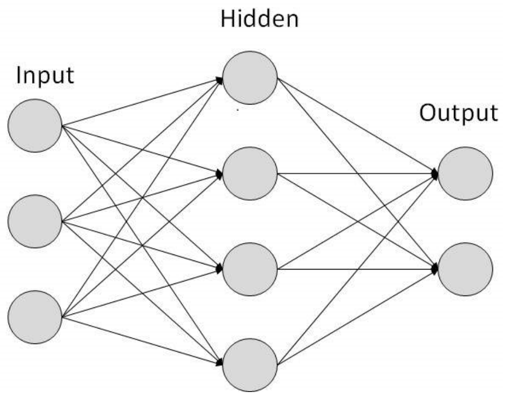
Artificial Neural Network:

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The idea of ANNs is based on the belief that working of human brain by making the right connections, can be imitated using silicon and wires as living neurons and dendrites.

The human brain is composed of 100 billion nerve cells called neurons. They are connected to other thousand cells by Axons. Stimuli from external environment or inputs from sensory organs are accepted by dendrites. These inputs create electric impulses, which quickly travel through the neural network. A neuron can then send the message to other neuron to handle the issue or does not send it forward. ANNs are composed of multiple nodes, which imitate biological neurons of human brain. The neurons are connected by links and they interact with each other. The nodes can take input data and perform simple operations on the data. The result of these operations is passed to other neurons. The output at each node is called its activation or node value.

Each link is associated with weight. ANNs are capable of learning, which takes place by altering weight values. The following illustration shows a simple ANN −



Machine Learning in ANNs

ANNs are capable of learning and they need to be trained. There are several learning strategies −

- **Supervised Learning** − It involves a teacher that is scholar than the ANN itself. For example, the teacher feeds some example data about which the teacher already knows the answers.

  For example, pattern recognizing. The ANN comes up with guesses while recognizing. Then the teacher provides the ANN with the answers. The network then compares it guesses with the teacher's "correct" answers and makes adjustments according to errors.

- **Unsupervised Learning** − It is required when there is no example data set with known answers. For example, searching for a hidden pattern. In this case, clustering i.e. dividing a set of elements into groups according to some unknown pattern is carried out based on the existing data sets present.

- **Reinforcement Learning** − This strategy built on observation. The ANN makes a decision by observing its environment. If the observation is negative, the network adjusts its weights to be able to make a different required decision the next time.

Implementations of Logic Functions

Program 1: Generate ANDNOT function using McCulloh-Pitts neural net by a MATLAB

X: column for inputs

Zi: column for true output

X1	X2	b(bias)	Y=X1 AND X2=t
0	0	1	0
0	1	1	0
1	0	1	1
1	1	1	0

Truth Table for AND function

MATLAB Program:

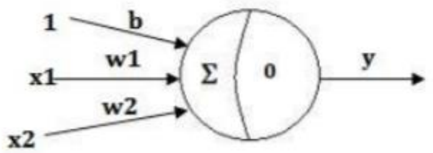
```
%ANDNOT function using Mcculloch-Pitts neuron
clear;
clc;
%Getting weights and threshold value
disp('Enter weights'); w1=input('Weight w1=');
w2=input('weight w2=');
disp('Enter Threshold Value');
theta=input('theta=');
y=[0 0 0 0];
x1=[0 0 1 1];
x2=[0 1 0 1];
z=[0 0 1 0];
con=1;
while con
zin=x1*w1+x2*w2;
for i=1:4
if zin(i)>=theta
y(i)=1;
else
y(i)=0;
```

```
end
end
disp('Output of Net');
disp(y);
if y==z
con=0;
else
disp('Net is not learning enter another set of weights and Threshold
value');
w1=input('weight w1=');
w2=input('weight w2=');
theta=input('theta=');
end
end
disp('Mcculloch-Pitts Net for ANDNOT function');
disp('Weights of Neuron');
disp(w1);
disp(w2);
disp('Threshold value');
disp(theta);
```

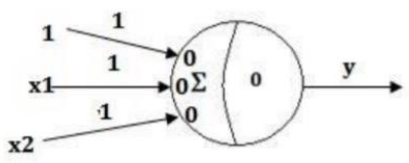
OUTPUT

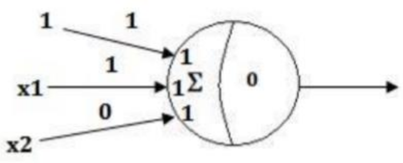
```
Enter weights
Weight w1=1
weight w2=1
Enter Threshold Value
theta=0.1
Output of Net
 0 1 1 1
Net is not learning enter another set of weights and Threshold value
Weight w1=1
weight w2=-1
theta=1
Output of Net
 0 0 1 0
Mcculloch-Pitts Net for ANDNOT function
Weights of Neuron
 1
 -1
Threshold value
 1
```

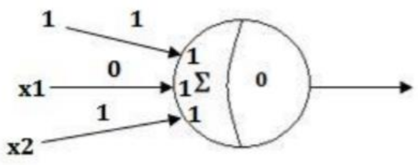
Assignment -1: Draw neural network weight adjustment at each step. Experiment with variation of initial weighting values. Initially, assume weight.

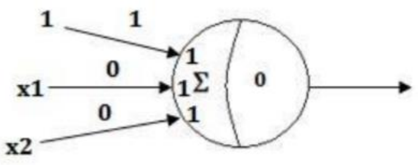


Initially, assume weight to be zero
i.e. w1 = w2 = b = 0
Formulae:
Wi(new) = Wi(old) + Xi*t
b(new) = b(old) + t









Assignment – 2: Similarly develop a McCulloch-Pitts neural net for OR, NAND and NOR gate and draw neural nets.

OR			**NAND**			**NOR**		
X	**Y**	**X OR Y**	**X**	**Y**	**X NAND Y**	**X**	**Y**	**X NOR Y**
0	0	0	0	0	1	0	0	1
0	1	1	0	1	1	0	1	0
1	0	1	1	0	1	1	0	0
1	1	1	1	1	0	1	1	0

Truth Table

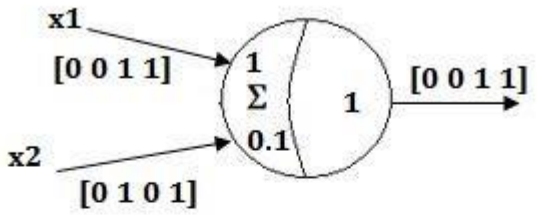
MATLAB Program:

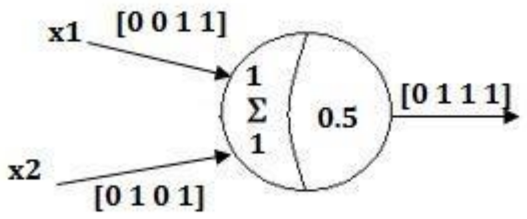
```
%OR NAND NOR function using McCulloch-Pitts neuron
function assign2()
function neural(name, ex1, ex2, th, w1, w2)
x1=ex1;
x2=ex2;
theta=th;
name=name;
y=[0 0 0 0];
switch name
case ‘NAND’
z=[1 1 1 0];
case ‘NOR’
z=[1 0 0 0];
case ‘OR’
z=[0 1 1 1];
end
con=1;
while con
zin=x1*w1+x2*w2;
for i=1:4
if zin(i)>=theta
y(i)=1;
else
y(i)=0;
end
end
disp('output of net');
disp(y);
if y==z
con=0;
else
disp('net is not learning enter another set of weights and
threshold value');
w1=input('weight w1=');
w2=input('weight w2=');
theta=input('theta=');
end
end
disp('McCulloch-Pitts net for function');
disp('weights of neuron');
disp(w1);
disp(w2);
disp('threshold value');
disp(theta);
clear;
clc;
%getting operation name
disp('type NAND for NAND gate; OR for OR gate and NOR for NOR
gate);
disp('enter operation in capital');
name=input('operation name=','s');
%getting weights and threshold value
disp=('enter weights');
w1=input('weight w1=');
w2=input('weight w2=');
theta=input('theta=');
neural(name,[0 0 1 1],[0 1 0 1],th,w1,w2)
end
```

OUTPUT:

Run – 1:

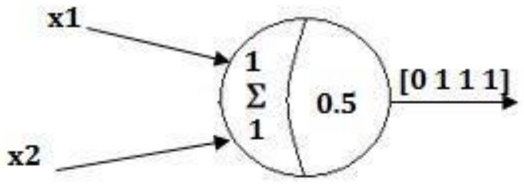
```
type NAND for NAND gate; OR for OR gate and NOR
for NOR gate enter operation
in capital       operation
name=OR  enter weights  weight
w1=1 weight w2=0.1 enter
threshold value theta =1

output of
net

0 0 1 1

net is not learning enter another set of weights and
threshold value

weight w1=1

weight w2=1 enter threshold
value theta =0.5 output of net

0 1 1 1

McCulloh-Pitts net for function
weights of neuron

1

1

threshold value

0.5000
```



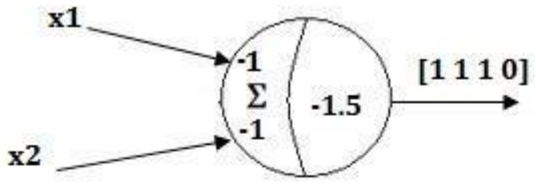


Run – 2:

```
type NAND for NAND gate; OR for OR gate and NOR for NOR
gate

enter operation in capital

operation name=NAND

enter weights  weight w1=1
weight w2=1 enter threshold
value theta =0.5 output of
net 0 1 1 1
```



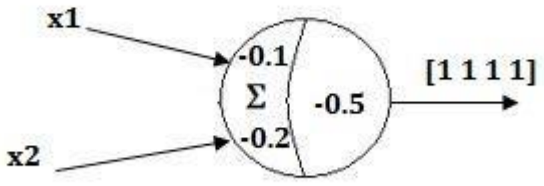
```
net is not learning enter
another set of weights and
threshold value

weight w1 = -1
weight w2 = -1 enter threshold
value theta = -1.5 output of
net

1 1 1 0
```



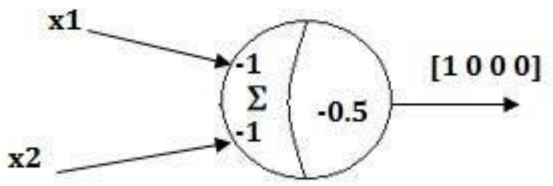
```
McCulloh-Pitts net for function
weights of neuron

-1  -1

threshold value

-1.5000
```

Run – 3: `type NAND for NAND gate; OR for OR gate and NOR for NOR gate enter operation in capital`

```
operation name=NOR  enter
weights  weight w1 = -0.1
weight w2 = -0.2 enter
threshold value theta = -
0.5 output of net 1 1 1 1

net is not learning enter
another set of weights
and threshold value
```



```
weight w1 = -1

weight w2 = -1 enter threshold
value theta = -0.5 output of
net

1 0 0 0

McCulloh-Pitts net for function
weights of neuron

-1  -1

threshold value

-0.5000
```



Assignment – 3:
Perform test for bipolar model as well.

The truth table for the AND function is given as

X1	X2	Y
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1

Truth Table for AND function

MATLAB PROGRAM:

```
%Perceptron for AND
funtion clear; clc;

x=[1 1 -1 -1;1 -
1 1 -1]; t=[1 -1
-1 -1]; w=[0 0];
b=0;
alpha=input('Ent
er Learning
rate=');
theta=input('Ent
er Threshold
value=');

con=1;
epoch=0;
while con
```

```
con=0;
for i=1:4

        yin=b+x(1,i)*w(1)+x(2,i)*w(2);

        if
yin>theta
y=1;

        end

        if yin <=theta &
yin>=-theta
y=0;         end
if yin<-theta
y=-1;         end
if y-t(i)
con=1;             for
j=1:2


w(j)=w(j)+alpha*t(i)*x(j,i);
end

            b=b+alpha*t(i);

        end

    end
epoch=epoch+1; end
disp('Perceptron for AND
funtion'); disp(' Final
Weight matrix'); disp(w);
disp('Final Bias');
disp(b);
```

OUTPUT:

```
  Enter Learning rate=1

  Enter Threshold value=0.5

  Perceptron for AND funtion

  Final Weight matrix

       1     1

  Final Bias

     -1
```

Assignment – 4:

Implement McCulloch-Pitts neural network model for XOR and give all the formula you used in the implementation. Draw the MLPs used for the implementation of above functions.

For the McCulloh-Pitts neural network model for XOR, previous single layer perceptron network cannot represent XOR gate. So, we need multiple network layer approach consisting of input layer, hidden layer and output layer. The multilayered perceptron implementation of XOR gate is as follow:

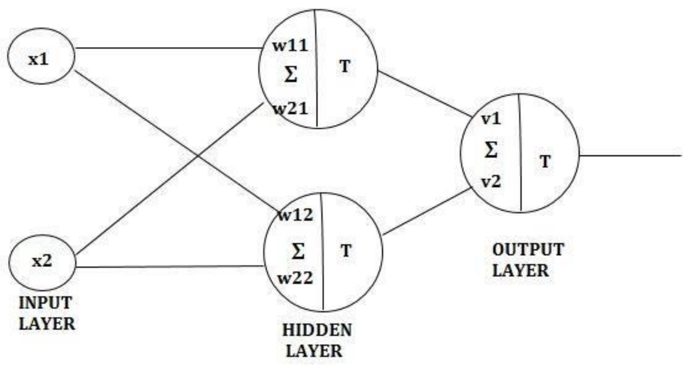


Figure 4 MCP-MLP implementation of XOR function

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

Truth Table for XOR

function **Formulas Used: Inputs to hidden layer:** $z_{in1} = x_1*w_{11}+x_2*w_{21}$ $z_{in2} = x_1*w_{12}+x_2*w_{22}$

```
for(i=1;i<=4;i++)
   if($z_{in1}$(i)>=theta


$y_1$(i)=1     else
     $y_1$(i)=0

   if($z_{in2}$(i)>=theta
          $y_2$(i)=1
else

          $y_2$(i)=0
```

Input to output layer:

```
y_in = y1*v1+y2*v2
for(i=1;i<=4;i++)

   if(y_in(i)>=theta


y(i)=1     else
     y(i)=0
```

MATLAB Program:

```
%XOR function using McCulloch-Pitts
neuron clear;

clc;

%Getting weights and
threshold value disp('Enter
weights');
w11=input('Weight w11=');
w12=input('weight w12=');
w21=input('Weight w21=');
w22=input('weight w22=');
v1=input('weight v1=');
v2=input('weight v2=');
disp('Enter Threshold
Value');
theta=input('theta=');
x1=[0 0 1 1]; x2=[0 1 0 1];
z=[0 1 1 0]; con=1; while
con   zin1=x1*w11+x2*w21;
zin2=x1*w21+x2*w22;

  for i=1:4
if
zin1(i)>=theta
y1(i)=1;
else
y1(i)=0;
end
if
zin2(i)>=theta
y2(i)=1;
else
y2(i)=0;
end

  end


yin=y1*v1+y2*v2;
```

```
for i=1:4
if
yin(i)>=theta;
y(i)=1;
else
y(i)=0;
end   end
disp('Output of
Net');
disp(y);   if
y==z
con=0;   else

      disp('Net is not learning enter another set of
weights and Threshold value');       w11=input('Weight
w11=');       w12=input('weight w12=');
w21=input('Weight w21=');       w22=input('weight w22=');
v1=input('weight v1=');       v2=input('weight v2=');
theta=input('theta=');   end end

disp('McCulloch-Pitts Net for XOR
function'); disp('Weights of
Neuron Z1'); disp(w11);
disp(w21);

disp('weights of
Neuron Z2');
disp(w12); disp(w22);

disp('weights of
Neuron Y');
disp(v1); disp(v2);
disp('Threshold
value');
disp(theta);
```

OUTPUT

```
  Enter weights
  Weight w11=1
  weight w12=-1
  Weight w21=-1
  weight w22=1
  weight v1=1
  weight v2=1
  Enter Threshold
  Value theta=1
  Output of Net
```

```
0  1
   1
   0
McCulloch-Pitts Net for XOR function

Weights of Neuron Z1

1
   -
   1
weights of Neuron Z2

    -1

     1

weights of Neuron Y

     1

     1

Threshold value
          1
```

Assignment – 5:

Implement MLP model for XOR by using back-propagation algorithm.

```
%xor using back propagation

function xor_back()

 function y=binsig(x) %sigmoid function

        y=1/(1+exp(-x));

     end

     function y=binsig1(x) %derivative purpose
y=binsig(x)*(1-binsig(x));

     end
clc;
clear;

 %initialize weight and bias and hidden layer

     v=[0.197 0.3191 -0.1448 0.3394;0.3099 0.1904 -
0.0347 -0.4861];      v1=zeros(2,4); %2*4 matrix with 0
as element

 b1=[-0.3378 0.2771 0.2859 -0.3329] %b(1) for input (0,0) and so on
 b2=-0.1401; %bias for output layer

 w=[0.4919; -0.2913; -0.3979; 0.3581];

     w1=zeros(4,1);   x=[1
1 0 0; 1 0 1 0];      t=[0
1 1 0];    alpha=0.02%;
%learning rate  mf=0.9;
%momentum factor
con=1;     epoch=0;
while con            e=0

  for i=1:4    %feed forward
for j=1:4
                          zin (j)=b1(j);

                           for i=1:2


zin(j)=zin(j)+x(i,i)*v(i,j);
end

                        z(j)=binsig(zin(j)); %activation function

                 end

                yin=b2+z*w;

              y(i)=binsig(yin);


               %back-propagation error

           delk=(t(i)-y(i))*binsig1(yin); %error for output layer
        network       delw=alpha*delk*z'+mf*(w-w1);%weight correction
        for o/p layer       delb2=alpha*delk; %bias weight correction
        for network

              delinj=delk*w; %for hidden layer

                for j=1:4

                      delj(j,1)=delinj(j,1)*binsig1(zin(j)); %delta
                 test m for hidden neuron

                 end

                for j=1:4

                     for i=1:2

                        delv(i,j)=alpha*delj(j,1)*x(i,i)+mf*(v(i,j)-
v1(i,j));

                      end
```

```
                  end

                delb1=alpha*delj;

           w1=w;
           v1=v;

           %weight update
           w=w+delw;
           b2=b2+delb;
           v=v+delv;

                b1=b1+delb1;

           e=e+(t(i)-y(i))^2;
           end

           if e<0.005

                con=0;

           end
           epoch=epoch+1;

     end
  disp('bpn for XOR function');
     disp('total epoch=');
disp(epoch);
     disp('error');
disp(e);
     disp('weight matrix
and bias');     v    b1
w    b2 end

%end of program
```

OUTPUT:

```
Bpn for XOR function

Total epoch=5385

Error=0.0050

Weight matrix and bias

V=

     4.4164    4.4836    2.6086    4.0386
     4.5230    -2.1693   -1.1147   -6.6716
B1=
     -0.9262   0.5910    0.6254    -1.0927
W=
      6.9573

     -5.5892

     -5.2180

      7.7782
B2=
     -0.3536
```

Discussion and Conclusion:

In this lab we came to know about the neural network, and logic function including AND,OR,NAND,NOR and XOR were implemented using matlab and several algorithm were used to implement those gates and neural network were also drawn.