

Artificial Intelligence (AI)

Lab No: 4

Amit Kumar Patel(071/BCT/502)

Introduction

Constraint 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 key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well.

Why Use Artificial Neural Networks?

Artificial Neural Networks, with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained Artificial Neural Network can be thought of as an "expert" in the category of information it has been given to analyze. This expert can then be used to provide projections given new situations of interest and answer "what if" questions.

Other advantages include:

1. Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.
2. Self-Organization: An ANN can create its own organization or representation of the information it receives during learning time.
3. Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.
4. Fault Tolerance via Redundant Information Coding: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage.

A Simple Neuron

An artificial neuron is a device with many inputs and one output. The neuron has two modes of operation; the training mode and the using mode. In the training mode, the neuron can be trained to fire (or not), for particular input patterns. In the using mode, when a taught input pattern is detected at the input, its associated output becomes the current output. If the input pattern does not belong in the taught list of input patterns, the firing rule is used to determine whether to fire or not.

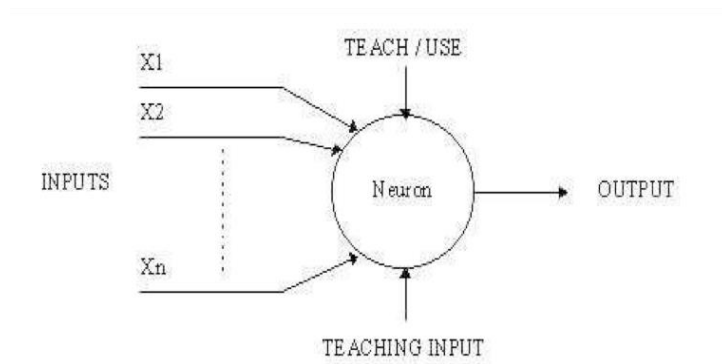


Figure 1.1: *A simple neuron*

Network layers

The commonest type of Artificial Neural Network consists of three groups, or layers, of units: a layer of "**input**" units is connected to a layer of "**hidden**" units, which is connected to a layer of "**output**" units. (See figure 1.2)

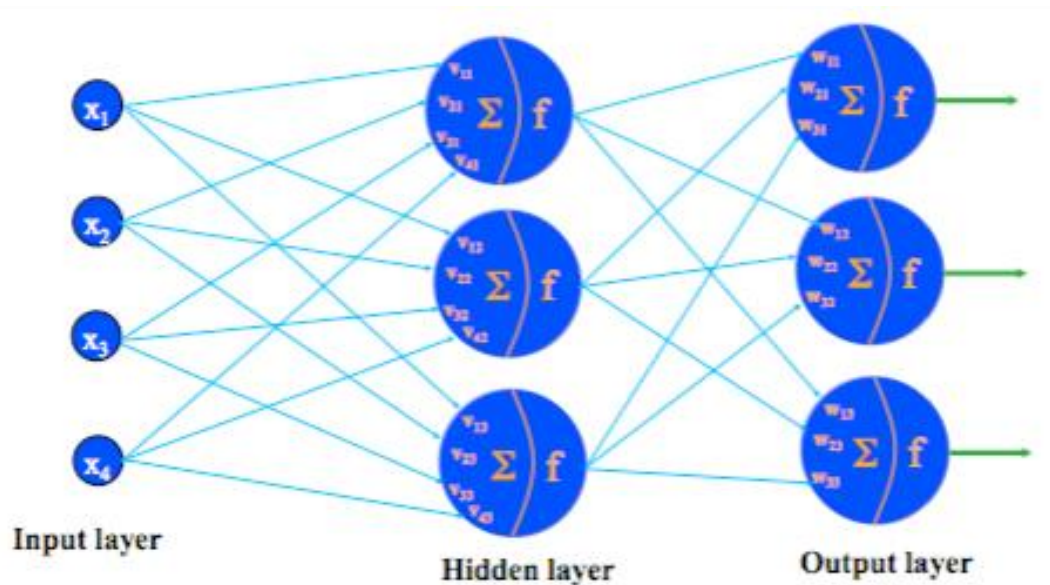


Figure 1.2: *A Simple Feed Forward Network*

The activity of the input units represents the raw information that is fed into the network. The activity of each hidden unit is determined by the activities of the input units and the weights on the connections between the input and the hidden units. The behavior of the output units depends on the activity of the hidden units and the weights between the hidden and output units.

This simple type of network is interesting because the hidden units are free to construct their own representations of the input. The weights between the input and hidden units determine when each hidden unit is active, and so by modifying these weights, a hidden unit can choose what it represents.

We also distinguish single-layer and multi-layer architectures. The single-layer organization, in which all units are connected to one another, constitutes the most general case and is of more potential computational

power than hierarchically structured multi-layer organizations. In multilayer networks, units are often numbered by layer, instead of following a global numbering.

Perceptrons

The most influential work on neural nets in the 60's went under the heading of 'perceptrons' a term coined by Frank Rosenblatt. The perceptron (See figure 1.3) turns out to be an MCP model (neuron with weighted inputs) with some additional, fixed, pre-processing. Units labeled A_1 , A_2 , A_j , A_p are called association units and their task is to extract specific, localized features from the input images. Perceptrons mimic the basic idea behind the mammalian visual system. They were mainly used in pattern recognition even though their capabilities extended a lot more.

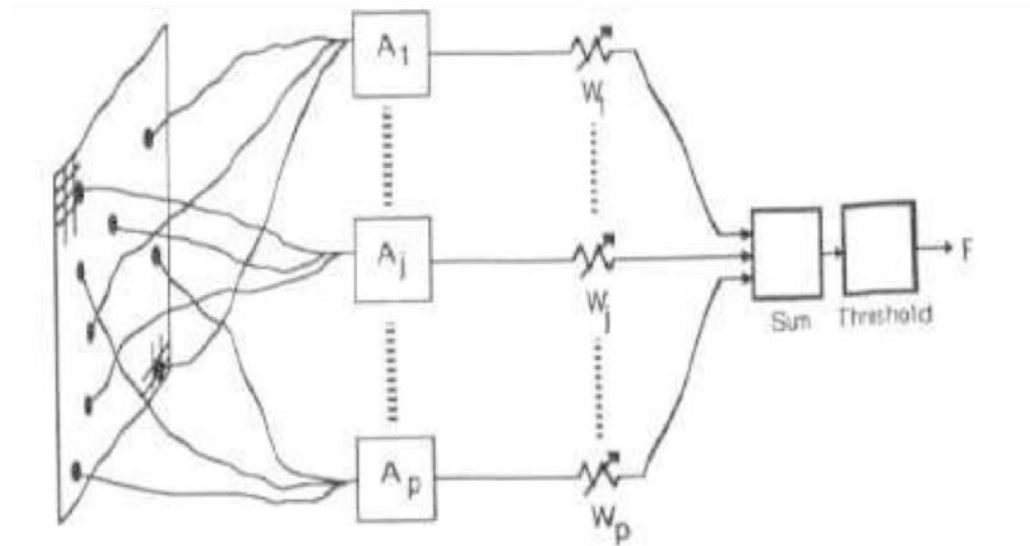


Figure 1.3: A Perceptron

Transfer Functions

The behavior of an ANN depends on both the weights and the input-output function (transfer function) that is specified for the units. This function typically falls into one of three categories:

- For **linear** (or ramp) the output activity is proportional to the total weighted output.
- For **threshold units**, the output is set at one of two levels, depending on whether the total input is greater than or less than some threshold value.
- For **sigmoid units**, the output varies continuously but not linearly as the input changes. Sigmoid units bear a greater resemblance to real neurons than do linear or threshold units, but all three must be considered rough approximations.

To make an Artificial Neural Network that performs some specific task, we must choose how the units are connected to one another and we must set the weights on the connections appropriately. The connections determine whether it is possible for one unit to influence another. The weights specify the strength of the influence.

We can teach a network to perform a particular task by using the following procedure:

1. We present the network with training examples, which consist of a pattern of activities for the input units together with the desired pattern of activities for the output units.
2. We determine how closely the actual output of the network matches the desired output.
3. We change the weight of each connection so that the network produces a better approximation of the desired output.

Implementation of Logic Functions

In this practical, we will learn how basic logic functions can be implemented and trained, using Matlab. The primitive procedure is explained by implementing a 2-input AND gate.

X : (column for inputs)

Zi : (column for true output)

PROGRAM: 1 Generate AND function using McCulloch-Pitts neural net by a MATLAB program.

Solution The truth table for the AND function is as follows:

<u>X</u>	<u>Y</u>	<u>Z= X AND Y</u>
1	1	0
0	1	0
1	0	0
0	0	1

The MATLAB program is given by,

Program

```
%AND 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);

```

Assignment (1): Draw neural network weight adjustment at each step. Experiment with variation of initial weighting values.

```

clear;
clc;
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:
```

```

weight w1=> 1
weight w2=> .1
theta=> 1
Output of Net
    0    0    1    1
Net is not learning enter another set of weights and Threshold
weight w1=> .1
weight w2=> -.1
theta=> .1
Output of Net
    0    0    1    0
McCulloch-Pitts Net for ANDNOT function
Weights of Neuron
    0.10000
   -0.10000
Threshold value
    0.10000

```

Discussion:

In the above program, we train the neural network for the AND NOT gate logic. We provide various weights and thresh holds and check if the output corresponds to the actual truth table and then repeat asking for new weight adjustments until and the actual result matches the truth table and hence we find the actual weights and the threshold.

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

NAND

```

clear;

clc;

w1 = input('Weight w1 = ');

% for nand gate use wt as: -0.1 -0.1 -0.1 i.e w1,w2 and theta works on bipolar as well

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 = [1 1 1 0]; % this is the output so change accordingly for and,or,nand and nor

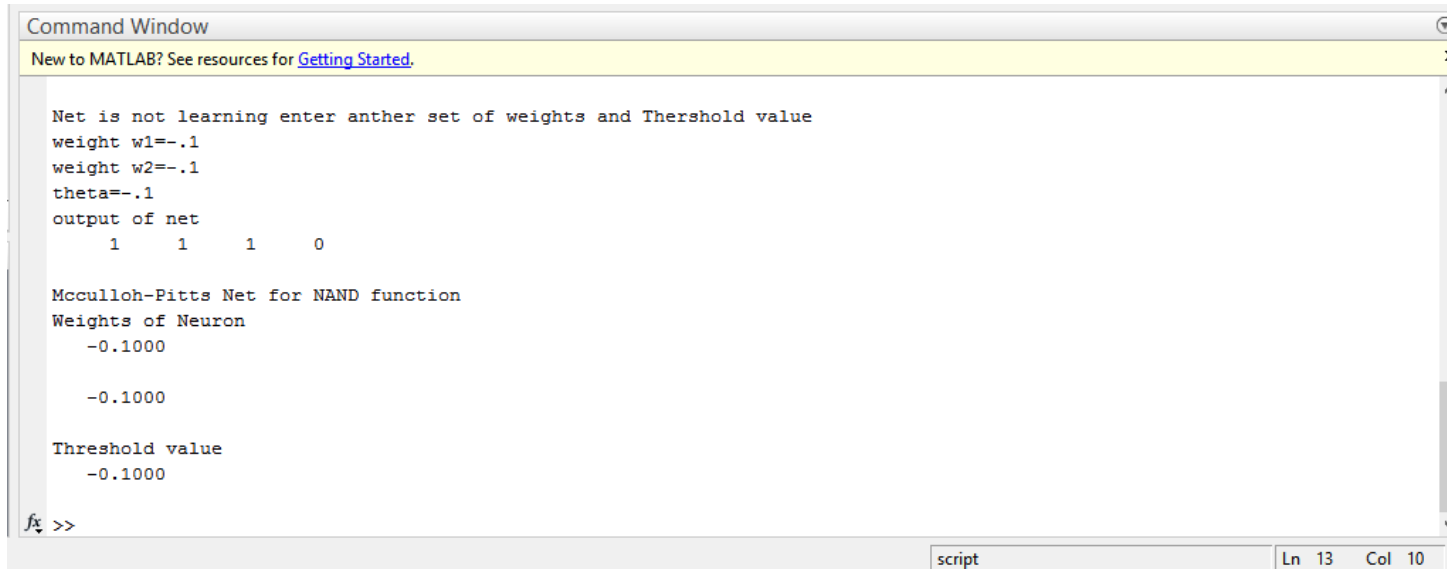
```

```

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 Thershold value');
        w1 = input('weight w1=');
        w2 = input('weight w2=');
        theta = input('theta=');
    end
end
disp('Mcculloh-Pitts Net for NAND function');
disp('Weights of Neuron');
disp(w1);
disp(w2);
disp('Threshold value');
disp(theta);

```


Output:



```
Command Window
New to MATLAB? See resources for Getting Started.

Net is not learning enter another set of weights and Thershold value
weight w1=-.1
weight w2=-.1
theta=-.1
output of net
    1    1    1    0

McCulloch-Pitts Net for NAND function
Weights of Neuron
    -0.1000

    -0.1000

Threshold value
    -0.1000

fx >>
```

script Ln 13 Col 10

NOR

```
clear;
clc;
w1 = input('Weight w1 = ');
% for nor gate use wt as : -0.1 -0.1 0 i.e w1,w2 and theta works on bipolar as well
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 = [1 0 0 0]; % this is the output so change accordingly for and,or,nand and nor
con = 1;
while con
    zin = x1*w1 + x2*w2;
    for i=1:4
        if zin(i) >= theta
            y(i) = 1;
        end
    end
    con = 0;
end
```

```

        else
            y(i) = 0;
        end
    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 Thershold value');
    w1 = input('weight w1=');
    w2 = input('weight w2=');
    theta = input('theta=');
end
end
disp('Mcculloh-Pitts Net for NOR function');
disp('Weights of Neuron');
disp(w1);
disp(w2);
disp('Threshold value');
disp(theta);

```

Output:

```
Command Window
New to MATLAB? See resources for Getting Started.

Weight w1 = -.1
Weight w2 = -.1
Enter Threshold Value
theta=0
output of net
    1    0    0    0

McCulloch-Pitts Net for NOR function
Weights of Neuron
    -0.1000

    -0.1000

Threshold value
    0

fx >> |
```

OR

```
clear;
```

```
clc;
```

```
w1 = input('Weight w1 = ');
```

```
% for or gate use wt as: 0.1 0.1 0.1 i.e w1,w2 and theta works on bipolar as well
```

```
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 1 1 1];
```

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

```

Output:

```
Command Window
New to MATLAB? See resources for Getting Started.

Weight w1 = .1
Weight w2 = .1
Enter Threshold Value
theta=.1
output of net
    0    1    1    1

McCulloch-Pitts Net for OR function
Weights of Neuron
    0.1000

    0.1000

Threshold value
    0.1000

fx >> |
```

Discussion:

For the nand, nor and or gates we again follow the same principal as per question 1 where we keep assigning and adjusting the weights until and unless we match with the actual truth table provided in Z variable in the above code where x1 and x2 are the inputs. Then as we get the result that matches the actual the truth table, the neural network is said to have learnt and have been assigned weights and threshold for that gate.

Assignment (3): Perform test for bipolar model as well.

BIPOLAR NAND

```
clear;
```

```
clc;
```

```
w1 = input('Weight w1 = ');
```

```
% for nand gate use wt as: -0.1 -0.1 -0.1 i.e w1,w2 and theta works on bipolar as well
```

```
w2 = input('Weight w2 = ');
```

```
disp('Enter Threshold Value');
```

```
theta = input('theta=');
```

```
y = [0 0 0 0];
```

```
x1 = [-1 -1 1 1];
```

```
x2 = [-1 1 -1 1];
```

```

z = [1 1 1 -1]; % this is the output so change accordingly for and,or,nand and nor
con = 1;
while con
    zin = x1*w1 + x2*w2;
    for i=1:4
        if zin(i) >= theta
            y(i) = 1;
        else
            y(i) = -1;
        end
    end
    disp('output of net');
    disp(y);
    if y == z
        con = 0;
    else
        disp('Net is not learning enter anther set of weights and Thershold value');
        w1 = input('weight w1=');
        w2 = input('weight w2=');
        theta = input('theta=');
    end
end
disp('Mcculloh-Pitts Net for NAND function');
disp('Weights of Neuron');
disp(w1);
disp(w2);
disp('Threshold value');
disp(theta);

```

Output:

```
Command Window
New to MATLAB? See resources for Getting Started.

Weight w1 = -.1
Weight w2 = -.1
Enter Threshold Value
theta=-.1
output of net
    1    1    1   -1

Mcculloh-Pitts Net for NAND function
Weights of Neuron
    -0.1000

    -0.1000

Threshold value
    -0.1000

fx >> |
```

BIPOLAR NOR

```
clear;
clc;
w1 = input('Weight w1 = ');
% for nor gate use wt as : -0.1 -0.1 0 i.e w1,w2 and theta works on bipolar as well
w2 = input('Weight w2 = ');
disp('Enter Threshold Value');
theta = input('theta=');
y = [0 0 0 0];
x1 = [-1 -1 1 1];
x2 = [-1 1 -1 1];
z = [1 -1 -1 -1]; % this is the output so change accordingly for and,or,nand and nor
con = 1;
while con
    zin = x1*w1 + x2*w2;
```

```

for i=1:4
    if zin(i) >= theta
        y(i) = 1;
    else
        y(i) = -1;
    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 Thershold value');
    w1 = input('weight w1=');
    w2 = input('weight w2=');
    theta = input('theta=');
end
end
disp('Mcculloh-Pitts Net for NOR function');
disp('Weights of Neuron');
disp(w1);
disp(w2);
disp('Threshold value');
disp(theta);

```


Output:

```
Command Window
New to MATLAB? See resources for Getting Started.

Net is not learning enter another set of weights and Thershold value
weight w1=.1
weight w2=.1
theta=-.1
output of net
    -1     1     1     1

Net is not learning enter another set of weights and Thershold value
weight w1=-.1
weight w2=-.1
theta=.1
output of net
     1    -1    -1    -1

McCulloch-Pitts Net for NOR function
Weights of Neuron
    -0.1000

    -0.1000

Threshold value
     0.1000
```

Activate Windows
Go to Settings to activate Windows.

BIPOLAR OR

```
clear;
```

```
clc;
```

```
w1 = input('Weight w1 = ');
```

```
% for or gate use wt as: 0.1 0.1 0.1 i.e w1,w2 and theta works on bipolar as well
```

```
w2 = input('Weight w2 = ');
```

```
disp('Enter Threshold Value');
```

```
theta = input('theta=');
```

```
y = [0 0 0 0];
```

```
x1 = [-1 -1 1 1];
```

```
x2 = [-1 1 -1 1];
```

```
z = [-1 1 1 1]; % this is the output so change accordingly for and,or,nand and nor
```

```
con = 1;
```

```
while con
```

```
    zin = x1*w1 + x2*w2;
```

```
    for i=1:4
```

```

        if zin(i) >= theta
            y(i) = 1;
        else
            y(i) = -1;
        end
    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 Thershold value');
    w1 = input('weight w1=');
    w2 = input('weight w2=');
    theta = input('theta=');
end
end
disp('Mcculloh-Pitts Net for OR function');
disp('Weights of Neuron');
disp(w1);
disp(w2);
disp('Threshold value');
disp(theta);

```

Output:

```
Command window
New to MATLAB? See resources for Getting Started.

weight w1=.1
weight w2=.1
theta=-.1
output of net
    -1     1     1     1

Mcculloch-Pitts Net for OR function
Weights of Neuron
    0.1000

    0.1000

Threshold value
   -0.1000

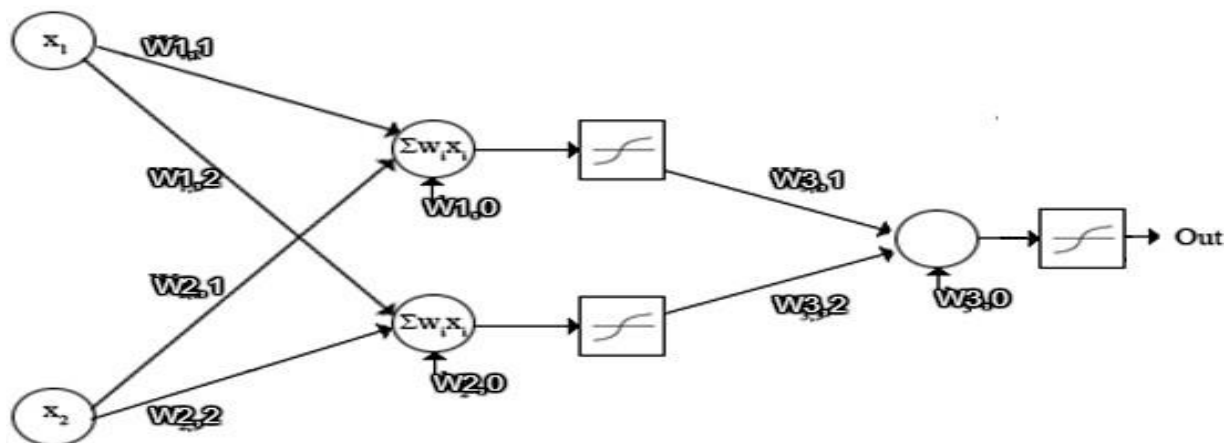
fx >>
```

Discussion:

In the bipolar implementation, we provide inputs as 1 for high and -1 for low unlike the previous ones where we provide 0 for low. Then we perform the weight adjustment again and as the result matches we find the weights and the threshold values.

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.

The MLP used for the implementation of XOR gate is:



MLP for XOR

The formulae used in the implementation are:

- $z1_in = x1*w11 + x2*w12 + w10$
- $z2_in = x1*w21 + x2*w22 + w20$
- $z3_in = y1*w31 + y2*w32 + w30$;

```
clear;
```

```
clc;
```

```
w10 = input('Bias w10= '); %follow the input sequence as
```

```
%-1 2 2 3 -2 -2 -3 2 2 as w10,w11,w12, w20, w21, w22, w30, w31,w32
```

```
w11 = input('Weight w11 = ');
```

```
w12 = input('Weight w12 = ');
```

```
w20 = input('Bias w20= ');
```

```
w21 = input('Weight w21 = ');
```

```
w22 = input('Weight w22 = ');
```

```
w30 = input('Bias w30= ');
```

```
w31 = input('Weight w31= ');
```

```
w32 = input('Weight w32= ');
```

```
disp('Enter Threshold Value');
```

```
% thershold is 0 for all
```

```
theta1 = input('theta1=');
```

```
theta2 = input('theta2=');
```

```
theta3 = input('theta3=');
```

```
y1 = [0 0 0 0];
```

```
y2= [0 0 0 0];
```

```
y3 = [0 0 0 0];
```

```
x1 = [0 0 1 1];
```

```
x2 = [0 1 0 1];
```

```
z = [0 1 1 0];
```

```

con = 1;
while con
    z1_in = x1*w11 + x2*w12 + w10;
    z2_in = x1*w21 + x2*w22 + w20;
    for i=1:4
        if z1_in(i) >= theta1
            y1(i) = 1;
        else
            y1(i) = 0;
        end
    end
    for i=1:4
        if z2_in(i) >= theta2
            y2(i) = 1;
        else
            y2(i) = 0;
        end
    end
    z3_in = y1*w31+y2*w32+w30;
    for i=1:4
        if z3_in(i) >= theta3
            y3(i) = 1;
        else
            y3(i) = 0;
        end
    end
    disp('output of net');
    disp(y3);
    if y3 == z

```

```

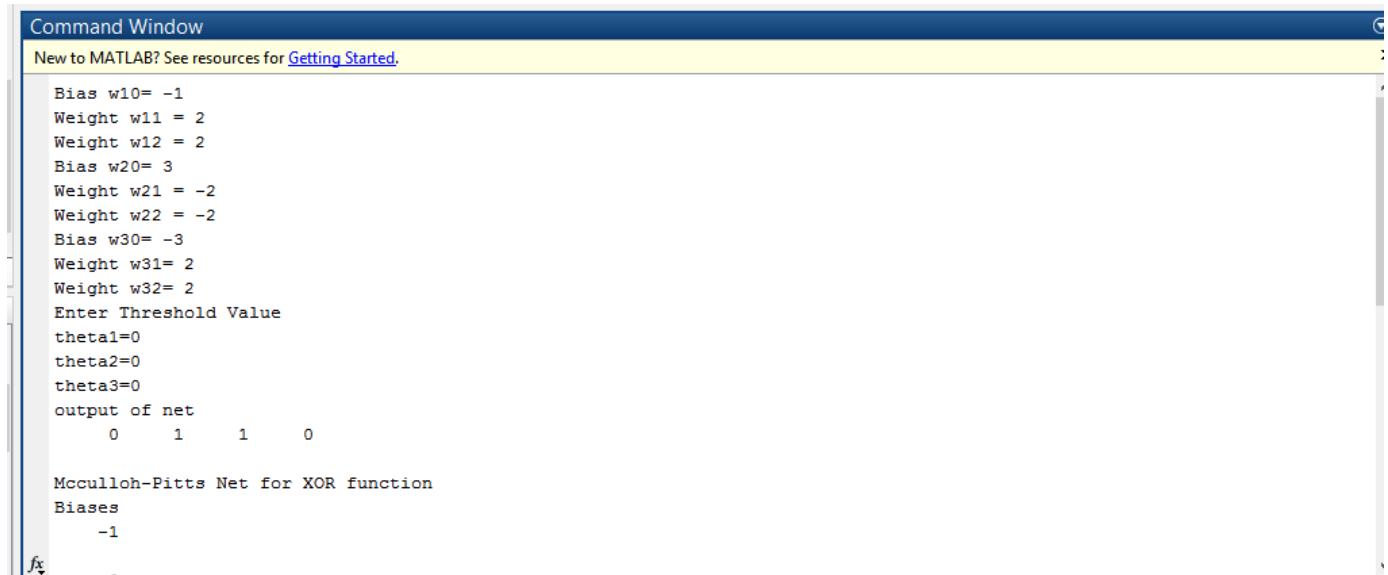
        con = 0;
    else
        disp('Net is not learning enter another set of weights and Threshold value');
        w11 = input('Weight w11 = ');
        w12 = input('Weight w12 = ');
        w21 = input('Weight w21 = ');
        w22 = input('Weight w22 = ');
        w31 = input('Weight w31 = ');
        w32 = input('Weight w32 = ');
        disp('Enter Threshold Value');
        theta1 = input('theta1 = ');
        theta2 = input('theta2 = ');
        theta3 = input('theta3 = ');
    end
end

disp('Mcculloh-Pitts Net for XOR function');
disp('Biases');
disp(w10);
disp(w20);
disp(w30);
disp('Weights of Neuron');
disp(w11);
disp(w12);
disp(w21);
disp(w22);
disp(w31);
disp(w32);
disp('Threshold values');
disp(theta1);
disp(theta2);

```

```
disp(theta3);
```

Output:



```
Command Window
New to MATLAB? See resources for Getting Started.

Bias w10= -1
Weight w11 = 2
Weight w12 = 2
Bias w20= 3
Weight w21 = -2
Weight w22 = -2
Bias w30= -3
Weight w31= 2
Weight w32= 2
Enter Threshold Value
theta1=0
theta2=0
theta3=0
output of net
      0      1      1      0

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

Discussion:

In this implementation of XOR gate, we use multiple layer perceptron, where there is a hidden layer as well. Here single level perceptron cannot yield the result so we use MLP as shown in figure above. The inputs are provided to the first layer whose output is provided to the second layer which in turn produces the final output. The weights are adjusted unless it matches the actual output and here the threshold is also adjusted which is 0 for this case.

Assignment (5): Implement MLP model for XOR by using backpropagation algorithm

```
clear;
```

```
clc;
```

```
w10 = -0.3;
```

```
w11 = 0.21;
```

```
w12 = 0.15;
```

```
w20 = 0.25;
```

```
w21 = -0.4;
```

```
w22 = 0.1;
```

```
w30 = -0.4;
```

```
w31 = -0.2;
```

```
w32 = 0.3;
```

```
r = 1 ;
```

```

y1 = 0;
y2 = 0;
y3 = 0;
x1 = [0 0 1 1];
x2 = [0 1 0 1];
z = [0 1 1 0];
d3 = 0;
d1 = 0;
d2 = 0;

con2 = 1;
while con2 <= 100000
    for i = 1:4
        z1_in = x1(i) * w11 + x2(i) * w12 + w10;
        y1 = activation_function(z1_in);
        z2_in = x1(i) * w21 + x2(i) * w22 + w20;
        y2 = activation_function(z2_in);
        z3_in = y1 * w31 + y2 * w32 + w30;
        y3 = activation_function(z3_in);
        d3 = (z(i)-y3) * y3 * (1-y3);

        d_in1 = d3 * w31;
        d1 = d_in1 * y1 * (1-y1);

        d_in2 = d3 * w32;
        d2 = d_in2 * y2 * (1-y2);

        w30 = w30 + (r * d3 * 1);
        w31 = w31 + (r * d3 * y1);
        w32 = w32 + (r * d3 * y2);
    end
end

```



```

w20 = w20 + (r * d2 * 1);
w21 = w21 + (r * d2 * x1(i));
w22 = w22 + (r * d2 * x2(i));

w10 = w10 + (r * d1 * 1);
w11 = w11 + (r * d1 * x1(i));
w12 = w12 + (r * d1 * x2(i));
con2 = con2+1;
end
end

disp('BackPropagation Net for XOR function');
disp('Weights of Neuron');
disp(w10);
disp(w11);
disp(w12);
disp(w20);
disp(w21);
disp(w22);
disp(w30);
disp(w31);
disp(w32);
disp('Check');
intr = 1;
while intr
    disp('Enter check value');
    i1 = input('i1=');
    i2 = input('i2=');

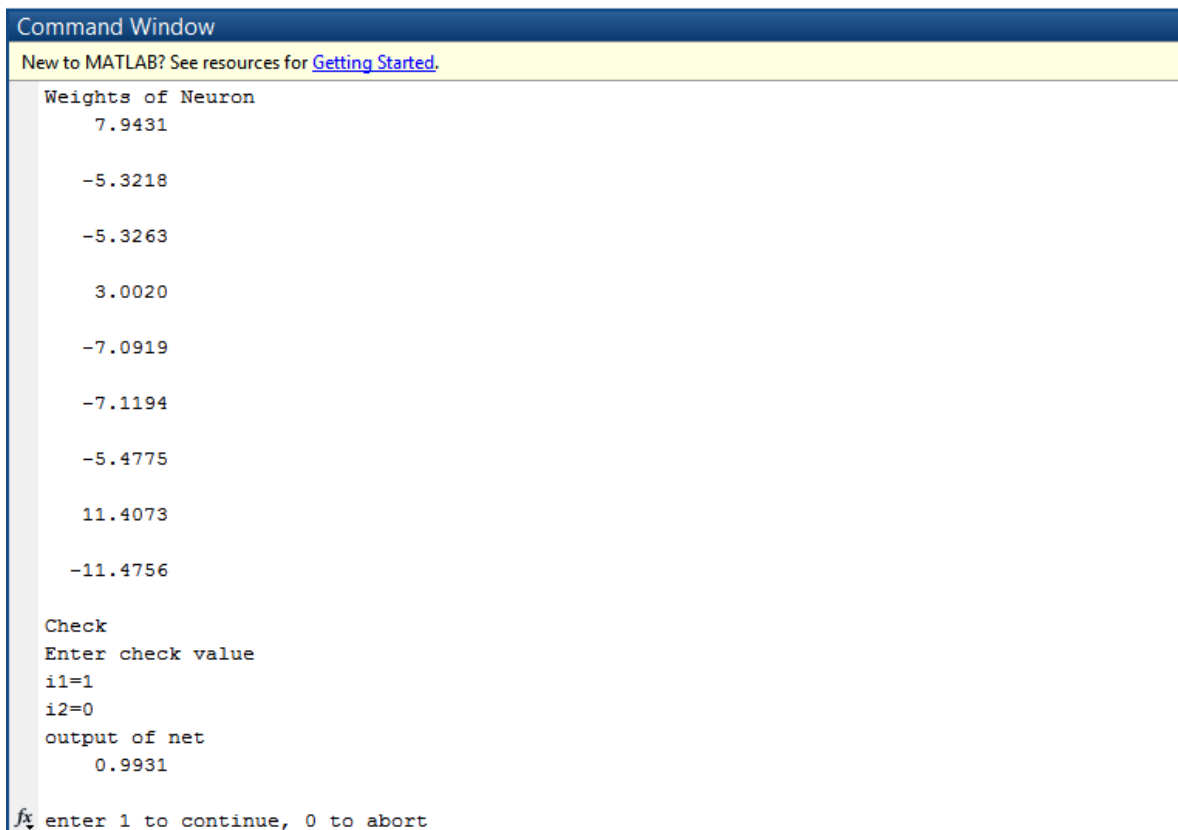
```

```

k1 = (i1 * w11) + (i2 * w12) + w10;
k1 = activation_function(k1);
k2 = (i1 * w21) + (i2 * w22) + w20;
k2 = activation_function(k2);
k3 = (k1 * w31) + (k2 * w32) + w30;
k3 = activation_function(k3);
disp('output of net');
disp(k3);
disp('enter 1 to continue, 0 to abort');
intr = input('continue=');
end

```

Output:



```

Command Window
New to MATLAB? See resources for Getting Started.

Weights of Neuron
    7.9431

   -5.3218

   -5.3263

    3.0020

   -7.0919

   -7.1194

   -5.4775

   11.4073

  -11.4756

Check
Enter check value
i1=1
i2=0
output of net
    0.9931

fx enter 1 to continue, 0 to abort

```

Discussion:

In the backpropagation method, we initially assign some values to the weights and then we repeat to adjust the weights without any external intervention or interaction for a large number of times(100000 times here).

IN each loop for every pair of inputs we calculate all the outputs at each layer and instead of a threshold we use activation function and then we calculate the adjusted weights and enter the next loop. After the network gets trained we can check for an input for example we provided 1 and 0 and got 0.9931 as the output which is close to 1.

Conclusion:

Hence, we can develop neural networks for various gates like NAND,NOR, OR using McCulloh-pits approach and also the bipolar approach. We can also develop XOR gate using various approaches which include McCulloh-pits method as well as backpropagation method.