CLASS: B.E. E &TC	**SUBJECT: ML**
EXPT. NO.: 1	**DATE: 10/05/21**
ROLL NO.: 42428	

TITLE: Implement simple logic network using MP neuron model

CO 3: Apply the fundamentals of Artificial Neural Network (ANN) to design and implement back propagation algorithm for various applications like classification, Self-Organizing Feature Maps, Learning vector quantization and Radial Basis Function networks. Using a substantial training and test data set, validate the efficacy of designed Neural Networks in terms of Confusion Matrix, accuracy, recognition rate, sensitivity and specificity.

CO 5: Carry out experiments as an individual and in a team, comprehend and write a laboratory record and draw conclusions at a technical level.

AIM:

To implement:

1. To understand the concept of a MP neuron model.
2. To implement logic gates AND, OR, NAND and NOR using MP neuron model

SOFTWARES REQUIRED: MATLAB 7.0 or Python

THEORY:

The early model of an artificial neuron is introduced by Warren McCulloch and Walter Pitts in 1943. The McCulloch-Pitts neural model is also known as linear threshold gate. It is a neuron of a set of inputs and one output. The linear threshold gate simply classifies the set of inputs into two different classes. Thus, the output is binary. McCulloh-Pitts model was an extremely simple artificial neuron. The inputs could be either a zero or a one. The McCulloch-Pitts Model Architecture is shown below:

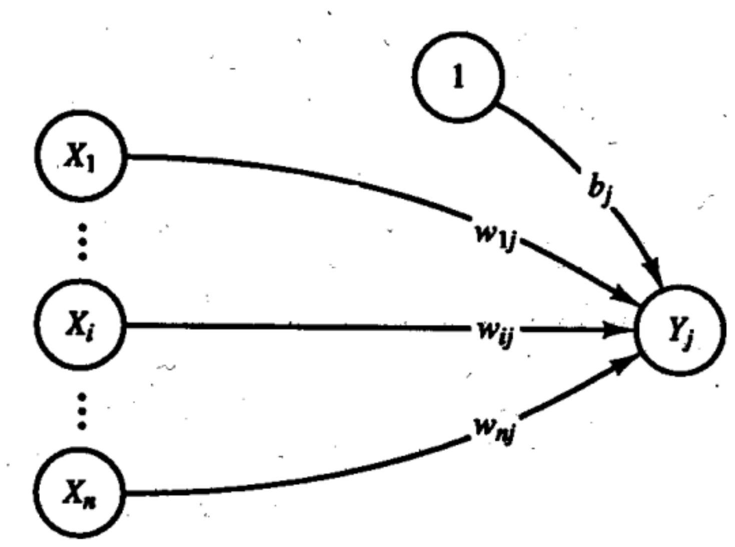
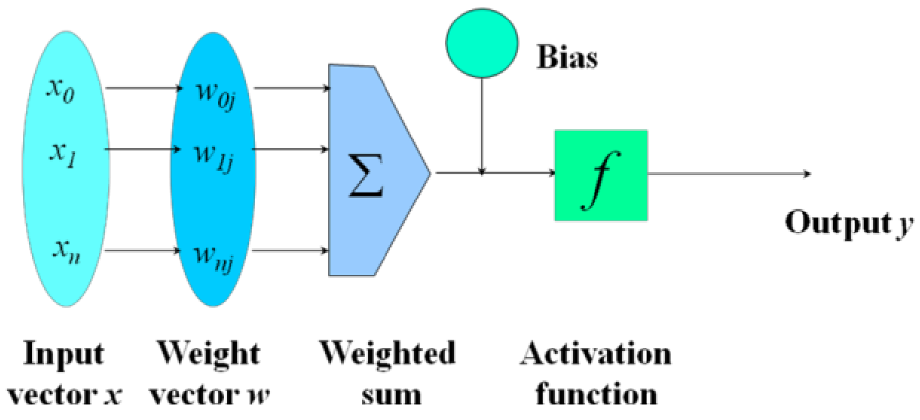


Figure: McCulloch-Pitts Model Architecture

In McCulloch -Pitts (MP) model is given by a weighted sum of its input values (xi), a bias term(b). The output signal is typically a non-linear activation function f (yin) which decides the threshold (Ɵ). The following describes the operation of a Neural Network:



Net input, yin is calculated as:

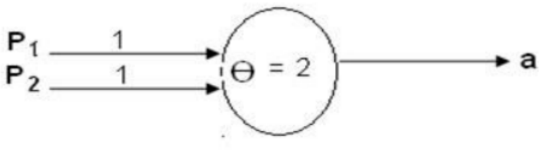
$$yin = \sum_{i=1}^{n} wixi - b$$

It should satisfy the condition for activation function (binary function) yin≥0 for which o/p is 1 else o/p is 0.

Three commonly used nonlinear functions are binary ramp & sigmoid, although only the binary function was used in original MP model. Network consisting of MP neurons with binary (on-off) output signals can be configured to perform several logical functions. Following figures shows some logical circuits using the MP model:

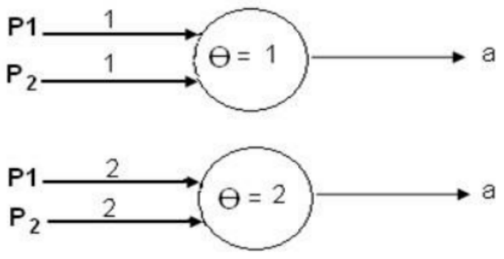
AND Gate:-

P1	P2	A
0	0	0
0	1	0
1	0	0
1	1	1



OR Gate:-

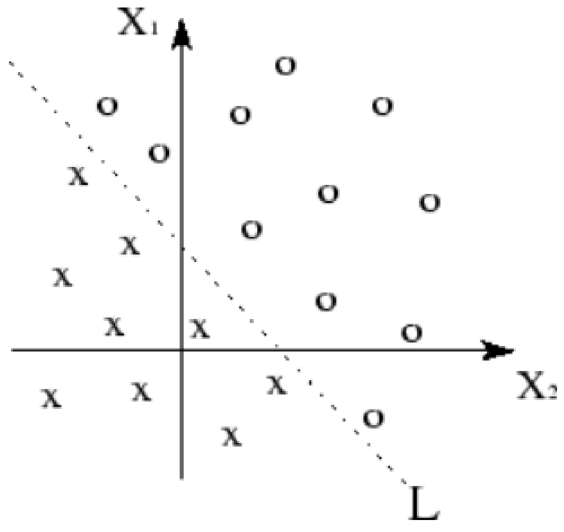
P1	P2	A
0	0	0
0	1	1
1	0	1
1	1	1



PROBLEM FACED WITH XOR GATE:

Consider two-input patterns (X_1, X_2) being classified into two classes as shown in figure. Each point with either symbol 'x' of 'o' or represents a pattern with a set of values (X_1, X_2). Each pattern is classified into one of two classes. Notice that these classes can be separated with a single line L. They are known as linearly separable patterns. Linear separability refers to the fact that classes of patterns with n-dimensional.

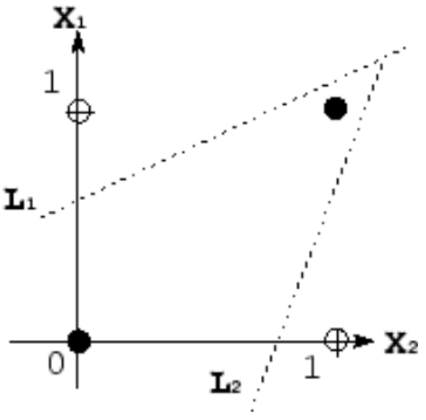
vector x = (x_1, x_2, ..., x_3) can be separated with a single decision surface. In the case above, the line represents the decision surface.



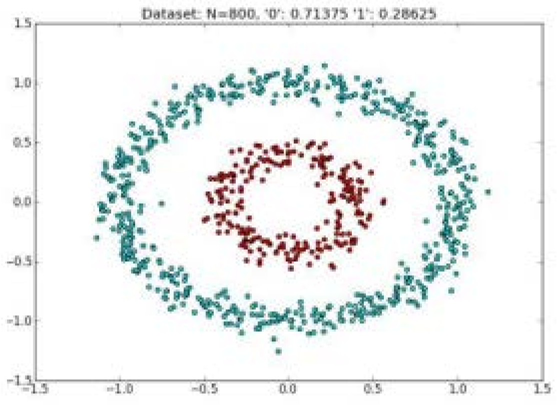
The processing unit of a single-layer perceptron network is able to categorize a set of patterns into two classes as the linear threshold function defines their linear separability. Conversely, the two classes must be linearly separable in order for the perceptron network to function correctly. Indeed, this is the main limitation of a single-layer perceptron network. The most classic example of linearly inseparable pattern is a logical exclusive-OR (XOR) function. Shown in figure is the illustration of XOR function that two classes, 0 for black dot and 1 for white dot, cannot be separated with a single line. The solution seems that patterns (X_1, X_2) of can be logically classified with two lines L1 and L2.

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

$$Y = X_1 \oplus X_2$$



Other example of linearly non-separable data is radially separated data.



Problem to be solved: Implement XOR function using McCulloch-Pitts neuron. Consider Binary Data. Consider two cases:

1) Both excitatory weights
2) One excitatory and one inhibitory

ALGORITHM:

1)Decide the no. of inputs to a McCulloch-Pitts model.
2)Accept the inputs from the user

3) Check whether the inputs are binary or not
4) If inputs are not in proper form (binary) ask the user to enter correct inputs.
5) Ask the user to enter choice of gate
    1) AND 2) OR 3) XOR
6) Depending upon no. of inputs & gate function required calculate the weights & bias.
7) Calculate the output of neuron.

Conclusion:

In this practical, we briefly looked at biological neurons. We then established the concept of MuCulloch-Pitts neuron, the first ever mathematical model of a biological neuron. We represented a bunch of boolean functions using the M-P neuron. We also tried to get a geometric intuition of what is going on with the model, using 3D plots. In the end, we also established a motivation for a more generalized model, the one and only artificial neuron/perceptron model.

References:

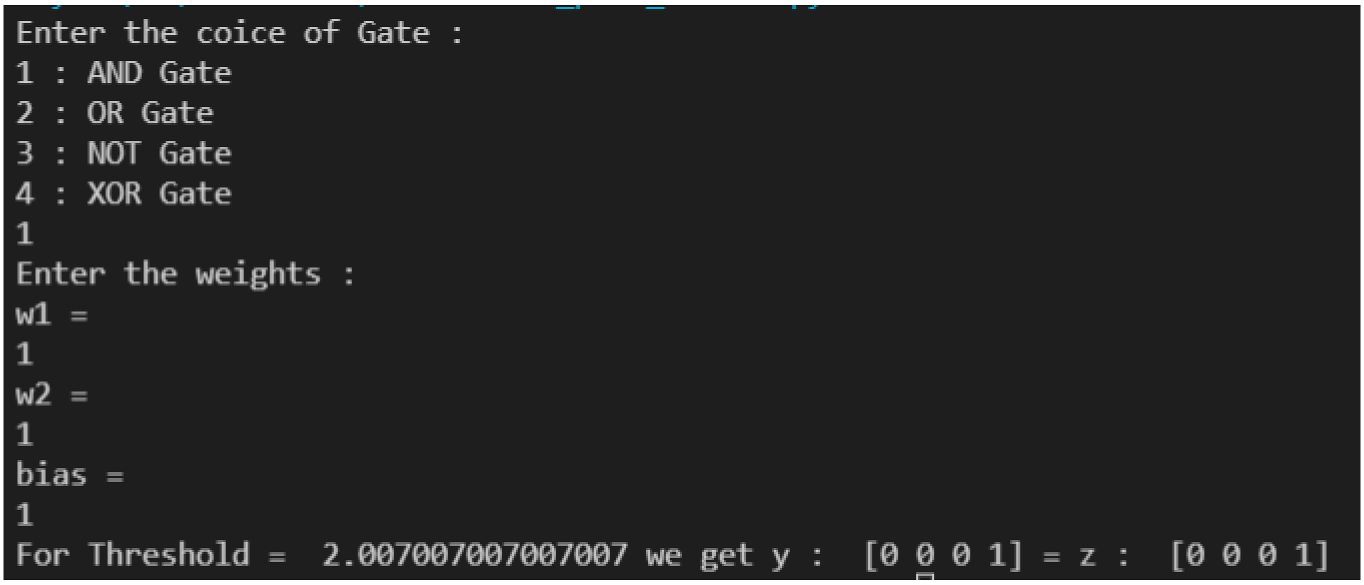
1. Laurene Fausett,"Fundamentals of Neural Networks: Architectures, Algorithms and Applications", Pearson Education, Inc, 2008.
2. S. N. Sivanandam, S.Sumathi, S. N. Deepa, "Introduction to Neural Networks using MATLAB", McGraw Hill, 2006.
3. S. N. Sivanandam, S. N. Deepa, "Principles of Soft Computing", John Wiley & Sons, 2007
4. Phil Kim, "MATLAB Deep Learning: With Machine Learning, Neural Networks and Artificial Intelligence", a Press 2017.

(Course Teacher)

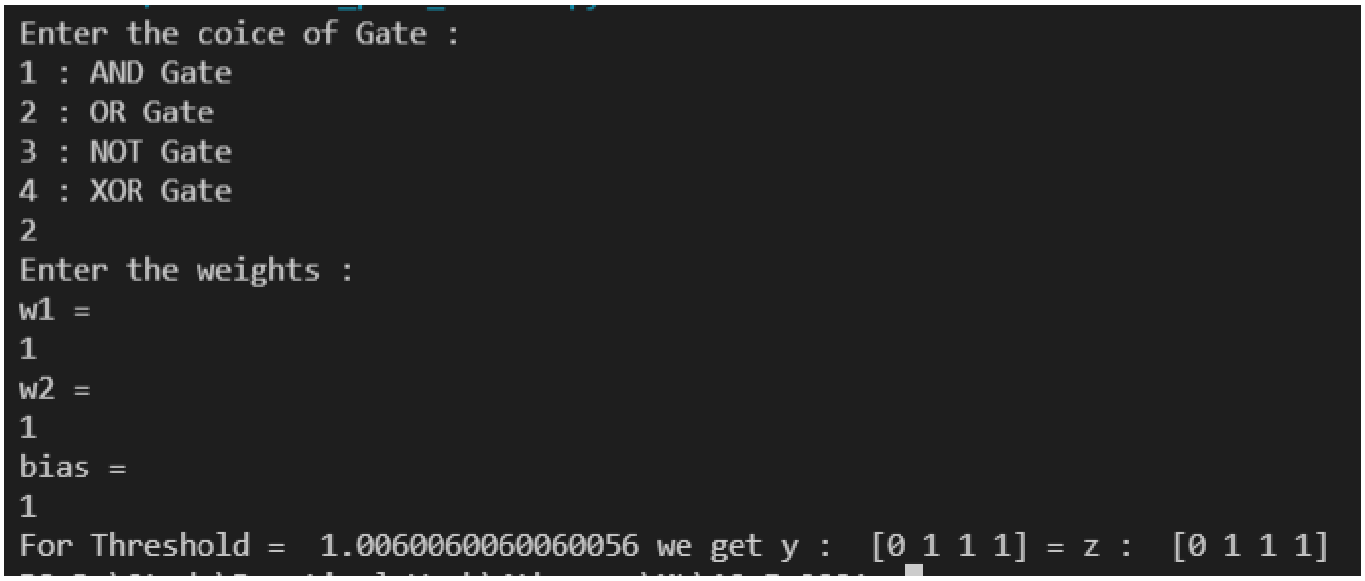
Code :

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

print("Enter the coice of Gate :\n1 : AND Gate\n2 : OR Gate\n3 : NOT Gate\n4 : XOR Gate"
)
c = int(input())
threshold = np.linspace(-5,5,1000)
if c==1:
    print("Enter the weights :\nw1 = ")
    w1 = int(input())
    print("w2 = ")
    w2 = int(input())
    print("bias = ")
    bias = int(input())

    x1 = np.array([0, 0, 1, 1])
    #print("x1 = ",x1)

    x2 = np.array([0, 1, 0, 1])
    #print("x2 = ",x2)

    z = np.array([0, 0, 0, 1])
    #print("z = ",z)

    yin = w1*x1 + w2*x2 + bias
    #print("yin = ",yin)

    for i in threshold:
        y = np.where(yin > i, 1, 0)
        #print("y = ",y)
        if(np.array_equal(y, z)):
            print("For Threshold = ",i,"we get y : ",y,"= z : ",z)
            break
elif c==2:
    print("Enter the weights :\nw1 = ")
    w1 = int(input())
    print("w2 = ")
    w2 = int(input())
    print("bias = ")
    bias = int(input())

    x1 = np.array([0, 0, 1, 1])
    #print("x1 = ",x1)

    x2 = np.array([0, 1, 0, 1])
    #print("x2 = ",x2)

    z = np.array([0, 1, 1, 1])
    #print("z = ",z)

    yin = w1*x1 + w2*x2 + bias
    #print("yin = ",yin)

    for i in threshold:
        y = np.where(yin > i, 1, 0)
        #print("y = ",y)
        if(np.array_equal(y, z)):
            print("For Threshold = ",i,"we get y : ",y,"= z : ",z)
            break
elif c==3:
    print("Enter the weights :\nw = ")
    #In case of NOT gate we will have to use inhibitory weight(which contribute to negay
ive values of inputs) to get proper threshold.
    w = int(input())
    print("bias = ")
    bias = int(input())

    x = np.array([0, 1])
    #print("x1 = ",x1)

    z = np.array([1, 0])
    #print("z = ",z)

    yin = w*x + bias
    #print("yin = ",yin)

    for i in threshold:
        y = np.where(yin > i, 1, 0)
        #print("y = ",y)
        if(np.array_equal(y, z)):
            print("For Threshold = ",i,"we get y : ",y,"= z : ",z)
            break
elif c==4:
    print("Enter the weights :\nw1 = ")
    w1 = int(input())
    print("w2 = ")
    w2 = int(input())
    print("bias = ")
    bias = int(input())
    threshold = np.linspace(-5,5,1000)
    x1 = np.array([0, 0, 1, 1])
    #print("x1 = ",x1)

    x2 = np.array([0, 1, 0, 1])
    #print("x2 = ",x2)

    z = np.array([0, 1, 1, 0])
    #print("z = ",z)

    yin = w1*x1 + w2*x2 + bias
    #print("yin = ",yin)

    for i in threshold:
        y = np.where((yin < i) & (yin > (i-max(w1,w2))), 1, 0)
        #print("y = ",y)
        if(np.array_equal(y, z)):
            print("For Threshold = ",i,"we get y : ",y,"= z : ",z)
            break
else:
    print("\nPlease enter Proper Option")
```

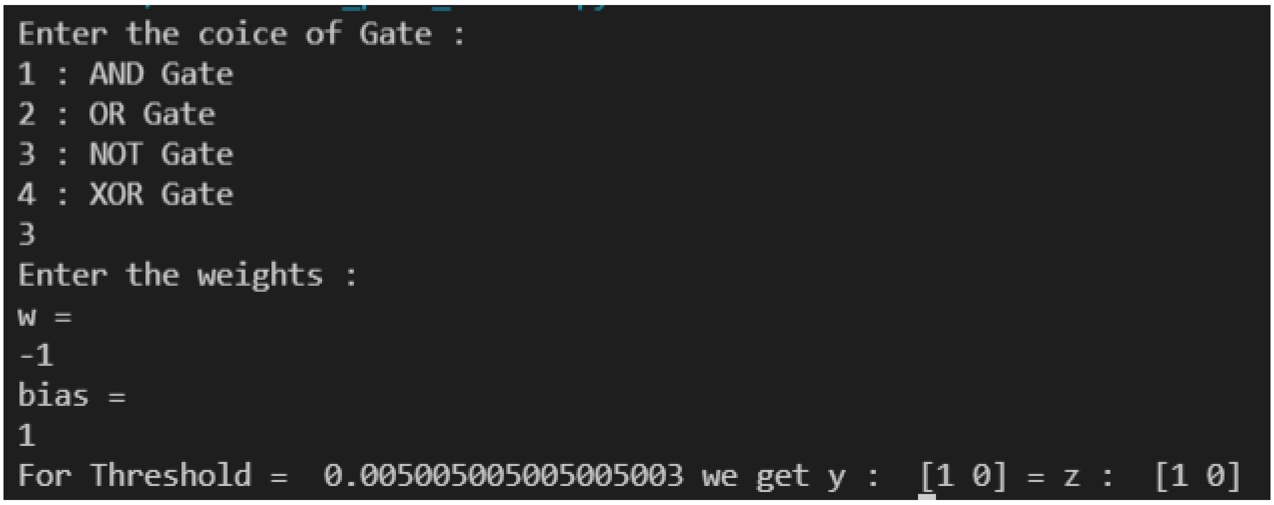
Output :



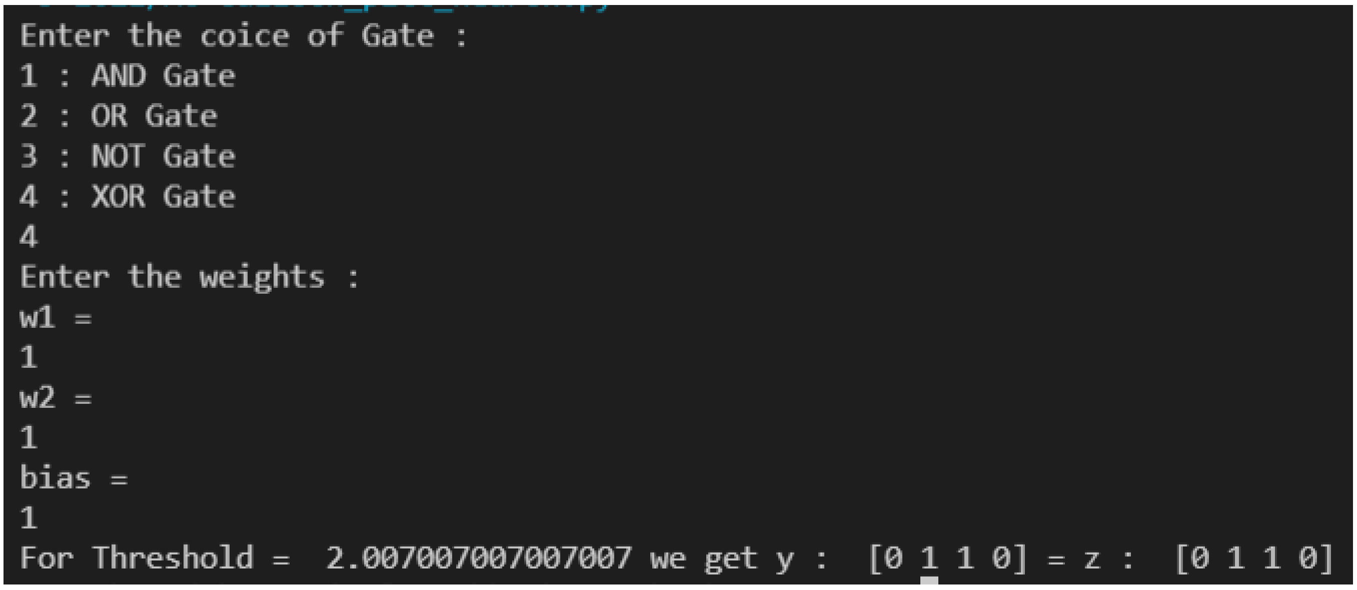
```
Enter the coice of Gate :
1 : AND Gate
2 : OR Gate
3 : NOT Gate
4 : XOR Gate
1
Enter the weights :
w1 =
1
w2 =
1
bias =
1
For Threshold =  2.007007007007007 we get y :  [0 0 0 1] = z :  [0 0 0 1]
```



```
Enter the coice of Gate :
1 : AND Gate
2 : OR Gate
3 : NOT Gate
4 : XOR Gate
2
Enter the weights :
w1 =
1
w2 =
1
bias =
1
For Threshold =  1.0060060060060056 we get y :  [0 1 1 1] = z :  [0 1 1 1]
```



```
Enter the coice of Gate :
1 : AND Gate
2 : OR Gate
3 : NOT Gate
4 : XOR Gate
3
Enter the weights :
w =
-1
bias =
1
For Threshold =  0.005005005005005003 we get y :  [1 0] = z :  [1 0]
```



```
Enter the coice of Gate :
1 : AND Gate
2 : OR Gate
3 : NOT Gate
4 : XOR Gate
4
Enter the weights :
w1 =
1
w2 =
1
bias =
1
For Threshold =  2.007007007007007 we get y :  [0 1 1 0] = z :  [0 1 1 0]
```