Batch: A1 Roll No.: 16010120015

Experiment No. 02

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: To implement XOR LOGIC using perceptron network. .

Objective: To implement classifier using Multi-layer perceptron network for XOR logic of 2 inputs.

Expected Outcome of Experiment:

CO2 : Analyze various neural network architectures

Books/ Journals/ Websites referred:

- J.S.R.Jang, C.T.Sun and E.Mizutani, "Neuro-Fuzzy and Soft Computing", PHI, 2004, Pearson Education 2004.
- Davis E.Goldberg, "Genetic Algorithms: Search, Optimization and Machine Learning", Addison Wesley, N.Y., 1989.
- S. Rajasekaran and G.A.V.Pai, "Neural Networks, Fuzzy Logic and Genetic Algorithms", PHI, 2003.
- http://library.thinkquest.org/C007395/tqweb/history.html.

Pre Lab/ Prior Concepts:

Neural networks, sometimes referred to as connectionist models, are parallel-distributed models that have several distinguishing features-

1) A set of processing units;
2) An activation state for each unit, which is equivalent to the output of the unit;
3) Connections between the units. Generally, each connection is defined by a weight w_{jk} that determines the effect that the signal of unit *j* has on unit *k;*
4) A propagation rule, which determines the effective input of the unit from its external inputs;
5) An activation function, which determines the new level of activation based on the effective input and the current activation;

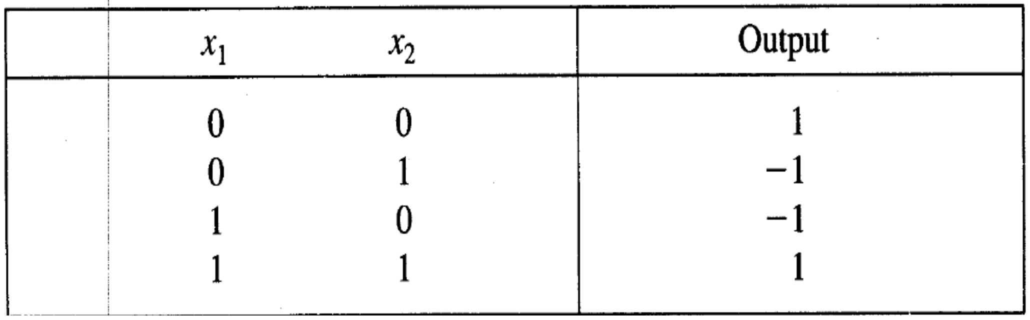
6) An external input (bias, offset) for each unit;
7) A method for information gathering (learning rule);
8) An environment within which the system can operate, provide input signals and, if necessary, error signals.

Perceptron Model

Linear separability

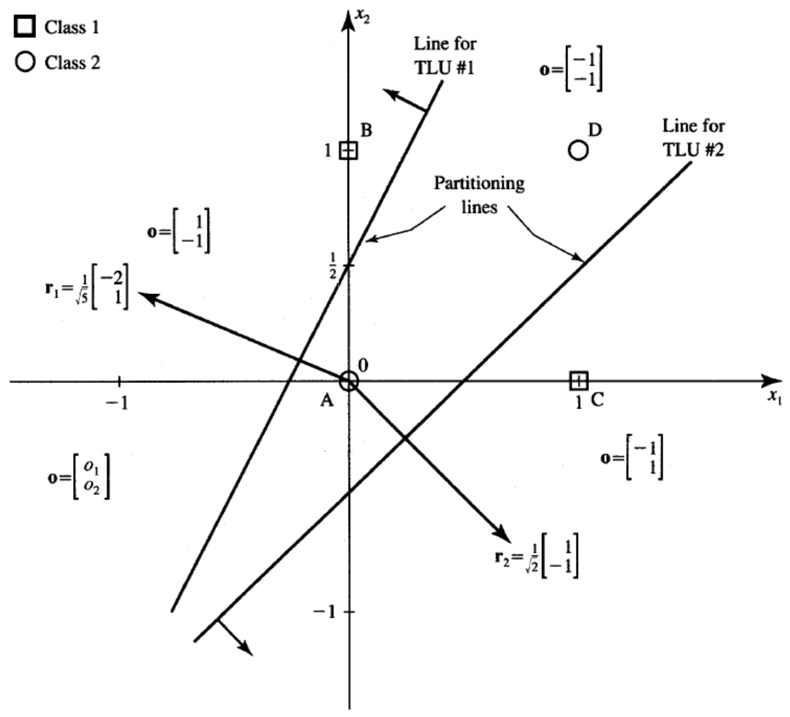
Design of Classifier using Multi-layer Perceptron model for XOR logic of 2 inputs (Refer Zurada 4.1 page no `68)

Truth table for XOR logic



	x_1	x_2	Output
	0	0	1
	0	1	−1
	1	0	−1
	1	1	1

Perceptron (TLU #1 and TLU +2) for first layer using bipolar activation function



Class 1
Class 2
x_2
Line for TLU #1
$\mathbf{o}=\begin{bmatrix}-1\\-1\end{bmatrix}$
B
D
1
Line for TLU #2
Partitioning lines
$\mathbf{o}=\begin{bmatrix}1\\-1\end{bmatrix}$
$\frac{1}{2}$
$\mathbf{r}_1=\frac{1}{\sqrt{5}}\begin{bmatrix}-2\\1\end{bmatrix}$
0
-1
A
1
C
x_1
$\mathbf{o}=\begin{bmatrix}-1\\1\end{bmatrix}$
$\mathbf{o}=\begin{bmatrix}o_1\\o_2\end{bmatrix}$
$\mathbf{r}_2=\frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\end{bmatrix}$
-1

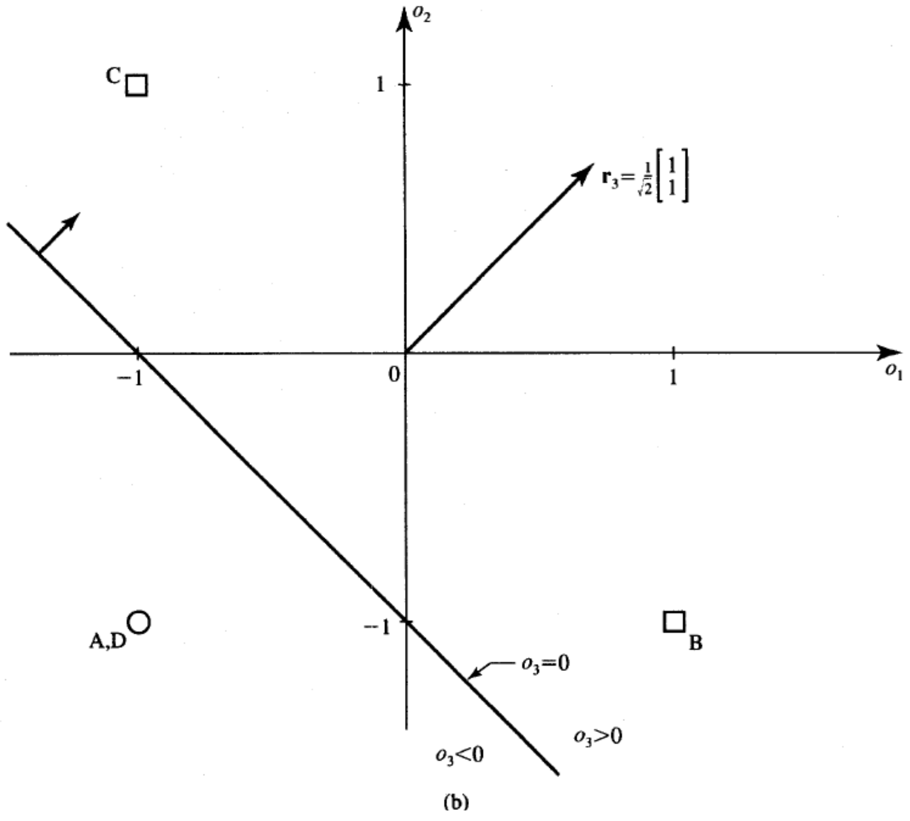
Two decision lines having equations using arbitrary selected partitioning as shown in above figure.

$$-2x_1 + x_2 - \frac{1}{2} = 0$$
$$x_1 - x_2 - \frac{1}{2} = 0$$

Mapping performed by first layer perceptron's TLU#1 and TLU#2

$$o_1 = \text{sgn}\left(-2x_1 + x_2 - \frac{1}{2}\right)$$
$$o_2 = \text{sgn}\left(x_1 - x_2 - \frac{1}{2}\right)$$

Final Output layer perceptron TLU#3



o_2
1
C
$\mathbf{r}_3 = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix}$
−1
0
1
o_1
A,D
−1
B
$o_3=0$
$o_3>0$
$o_3<0$
(b)

The decision line represents the equation

$$o_1 + o_2 + 1 = 0$$

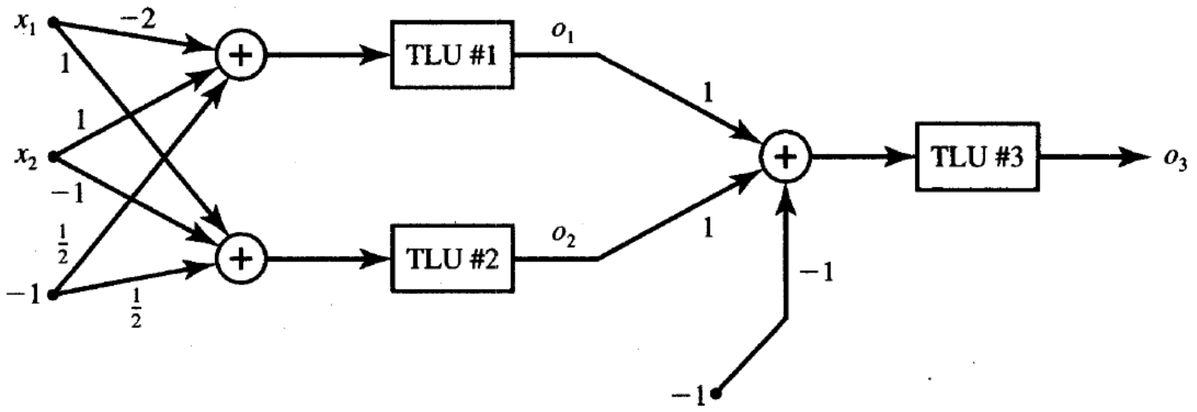
Mapping performed by output perceptron

The TLU #3 implements the decision o_3 as

$$o_3 = \operatorname{sgn}(o_1 + o_2 + 1)$$

Classification summary table

Symbol	Pattern Space		Image Space		TLU #3 Input	Output Space	Class Number
	x_1	x_2	o_1	o_2	$o_1 + o_2 + 1$	o_3	
A	0	0	−1	−1	−	−1	2
B	0	1	1	−1	+	+1	1
C	1	0	−1	1	+	+1	1
D	1	1	−1	−1	−	−1	2

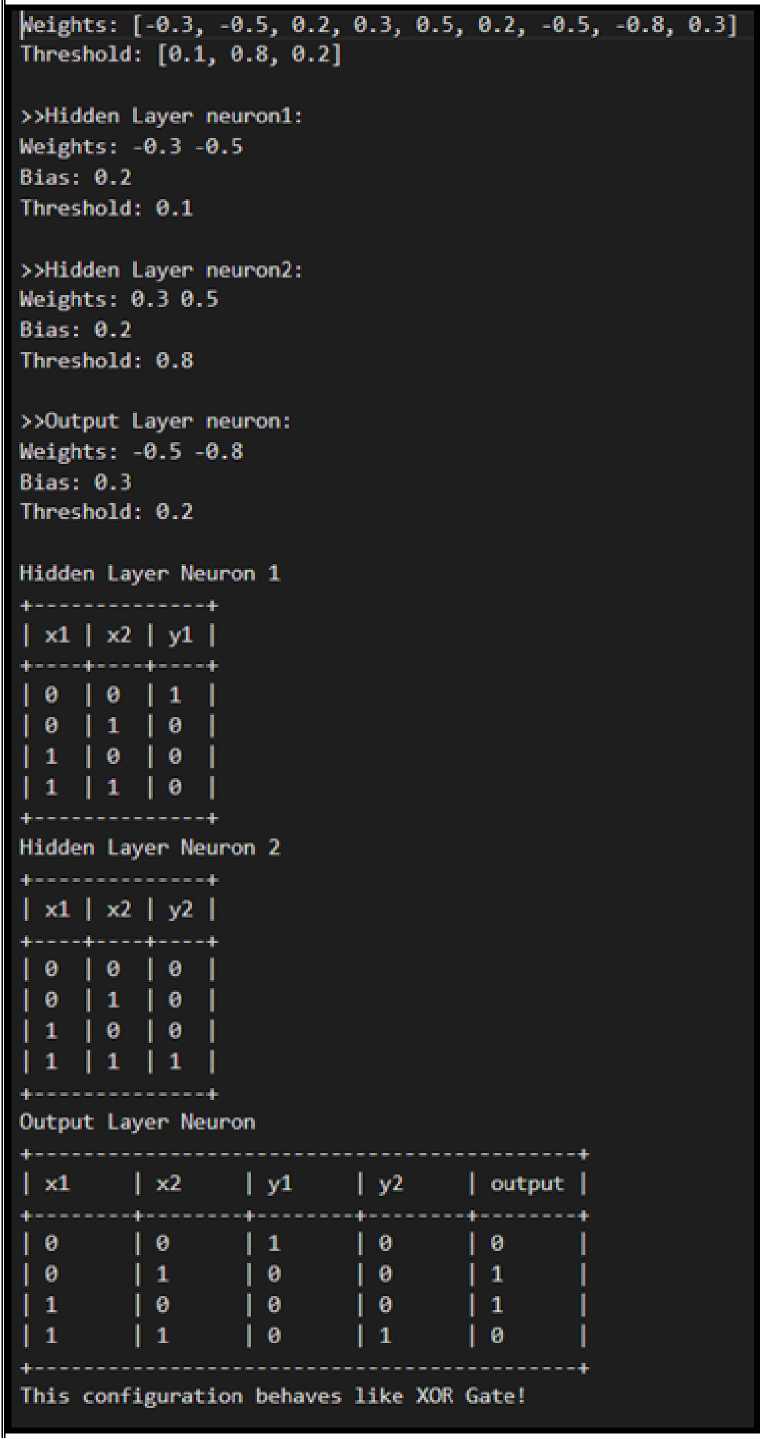


Classifier using Multi-layer Perception Network for implementing XOR logic for 2 inputs.

Code:

```
import numpy as np
import TableIt

def mlnn(x,w,t):
   bias1 = w[2]
   bias2 = w[5]
   bias3 = w[8]
   norHiddenLayerWeights = [w[0],w[1]]
   andHiddenLayerWeights = [w[3],w[4]]
   norOutputLayerWeights = [w[6],w[7]]
   threshold1 = t[0]
   threshold2 = t[1]
   threshold3 = t[2]
   print("\n>>Hidden Layer neuron1:")
   print(f"Weights: {w[0]} {w[1]}")
   print(f"Bias: {w[2]}")
   print(f"Threshold: {t[0]}\n")

   print(">>Hidden Layer neuron2:")
   print(f"Weights: {w[3]} {w[4]}")
   print(f"Bias: {w[5]}")
   print(f"Threshold: {t[1]}\n")

   print(">>Output Layer neuron:")
   print(f"Weights: {w[6]} {w[7]}")
   print(f"Bias: {w[8]}")
   print(f"Threshold: {t[2]}\n")

   output1 = []
   output2 = []
   output3 = []
   i=0
   while i < 4:
      v = np.dot(norHiddenLayerWeights,x[i]) + bias1
      if v >= threshold1:
         output1.append(1)
      else:
         output1.append(0)
      i+=1
   print("Hidden Layer Neuron 1")
   table = [
    ["x1","x2","y1"],
    ["0","0",output1[0]],
    ["0","1",output1[1]],
    ["1","0",output1[2]],
    ["1","1",output1[3]],
]
TableIt.printTable(table, useFieldNames=True)

i=0
while i<4:
    v = np.dot(andHiddenLayerWeights,x[i]) + bias2
    if v >= threshold2:
        output2.append(1)
    else:
        output2.append(0)
    i+=1
print("Hidden Layer Neuron 2")
table1 = [
    ["x1","x2","y2"],
    ["0","0",output2[0]],
    ["0","1",output2[1]],
    ["1","0",output2[2]],
    ["1","1",output2[3]],
]
TableIt.printTable(table1, useFieldNames=True)

norOutputLayerInputs = []
i=0
while i<4:
    norOutputLayerInputs.append([output1[i],output2[i]])
    i+=1
i=0
while i<4:
    v = np.dot(norOutputLayerWeights,norOutputLayerInputs[i]) + bias3
    if v >= threshold3:
        output3.append(1)
    else:
        output3.append(0)
    i+=1

print("Output Layer Neuron")
table2 = [
        ["x1","x2","y1","y2","output"],
        ["0","0",output1[0],output2[0],output3[0]],
        ["0","1",output1[1],output2[1],output3[1]],
        ["1","0",output1[2],output2[2],output3[2]],
        ["1","1",output1[3],output2[3],output3[3]],
    ]
    TableIt.printTable(table2, useFieldNames=True)

    xorOutput = [0,1,1,0]
    if output3==xorOutput:
        print("This configuration behaves like XOR Gate!")
    else:
        print("This configuration doesn't behave like XOR Gate!")


inputs = [[0,0],[0,1],[1,0],[1,1]]
weights = []
threshold = []
i = 0
while i<9:
    val = float(input(f'Enter weight{i+1}: '))
    weights.append(val)
    i += 1
print("Weights:",weights)
k=0
while k<3:
    val = float(input(f"Enter threshold for neuron{k+1}: "))
    threshold.append(val)
    k += 1
print(f"Threshold: {threshold}")

mlnn(inputs,weights,threshold)
```

Output:



```
Weights: [-0.3, -0.5, 0.2, 0.3, 0.5, 0.2, -0.5, -0.8, 0.3]
Threshold: [0.1, 0.8, 0.2]

>>Hidden Layer neuron1:
Weights: -0.3 -0.5
Bias: 0.2
Threshold: 0.1

>>Hidden Layer neuron2:
Weights: 0.3 0.5
Bias: 0.2
Threshold: 0.8

>>Output Layer neuron:
Weights: -0.5 -0.8
Bias: 0.3
Threshold: 0.2

Hidden Layer Neuron 1
+--------------+
| x1 | x2 | y1 |
+----+----+----+
| 0  | 0  | 1  |
| 0  | 1  | 0  |
| 1  | 0  | 0  |
| 1  | 1  | 0  |
+--------------+
Hidden Layer Neuron 2
+--------------+
| x1 | x2 | y2 |
+----+----+----+
| 0  | 0  | 0  |
| 0  | 1  | 0  |
| 1  | 0  | 0  |
| 1  | 1  | 1  |
+--------------+
Output Layer Neuron
+--------------------------------------------+
| x1     | x2     | y1     | y2     | output |
+--------+--------+--------+--------+--------+
| 0      | 0      | 1      | 0      | 0      |
| 0      | 1      | 0      | 0      | 1      |
| 1      | 0      | 0      | 0      | 1      |
| 1      | 1      | 0      | 1      | 0      |
+--------------------------------------------+
This configuration behaves like XOR Gate!
```

Algorithm:

Step 1: To initialize the weights and biases with random values to get output as per truth table of XOR.
Step 2: Compute net and apply activation function on it
Y = 1 if net >= Threshold
Y = 0 net < Threshold
Step 3: Repeat Step2 For hidden layer neuron2.
Step 4: Compute output for output layer neuron using output of hidden layer neurons as input vector.
Step 5: Check output from output layer neuron to the desired output to check if configuration used by user behaves like XOR gate or not.

Conclusion: Thus, we have successfully implemented classifier using MLP for linearly non-separable XOR functions of 2 inputs

Post Lab Descriptive Questions :

1. Why is XOR-logic function being Linearly notseparable?

ANSWER:
Let us suppose, if possible, that the XORXOR function, given by following table, is linearly separable.
x0011y0101x xor y0110xyx xor y000011101110

This ensures the existence of a line L:ax+by+c=0L:ax+by+c=0 such that the points (0,0)(0,0) && (1,1)(1,1) both lie on the same side of LL and the points (0,1)(0,1) && (1,0)(1,0) also lie on same side of LL but opposite to that of (0,0)(0,0) && (1,1).(1,1).
Also from basic coordinate geometry, we know that if the points (x1,y1)(x1,y1) && (x2,y2)(x2,y2) lie on same side of a line given by px+qy+r=0px+qy+r=0 then,
(px1+qy1+r)·(px2+qy2+r)>0(px1+qy1+r)·(px2+qy2+r)>0

and if they are on opposite sides of the line then,
(px1+qy1+r)·(px2+qy2+r)<0(px1+qy1+r)·(px2+qy2+r)<0

Since (0,0)(0,0) and (1,1)(1,1) lie on same side of LL, so
c·(a+b+c)>0(1)(1)c·(a+b+c)>0

And, as (1,0)(1,0) and (1,1)(1,1) lie on different sides of LL, so (a+c)⋅(a+b+c)<0.(2)(2)(a+c)⋅(a+b+c)<0.

Similarly as (0,1)(0,1) and (1,1)(1,1) lie on different sides of LL,

(b+c)⋅(a+b+c)<0(3)(3)(b+c)⋅(a+b+c)<0

On adding equations (2)(2) and (3)(3) we get,

(a+b+2c)⋅(a+b+c)<0(a+b+2c)⋅(a+b+c)<0

⟹(a+b+c)⋅(a+b+c)+c⋅(a+b+c)<0⟹(a+b+c)⋅(a+b+c)+c⋅(a+b+c)<0

⟹(a+b+c)2+c⋅(a+b+c)<0⟹(a+b+c)2+c⋅(a+b+c)<0

Since (a+b+c)2≥0(a+b+c)2≥0 for any choice of numbers a,b,c,a,b,c, so

c⋅(a+b+c)<−(a+b+c)2≤0c⋅(a+b+c)<−(a+b+c)2≤0

c⋅(a+b+c)<0c⋅(a+b+c)<0

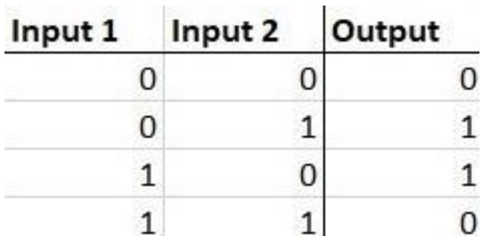
which is a contradiction of equation (1)(1), proving the non-existence any such line L.

2. Design a classifier using MLP perceptron model for implementing NOT XOR logic

ANSWER:

XOr logical Operator

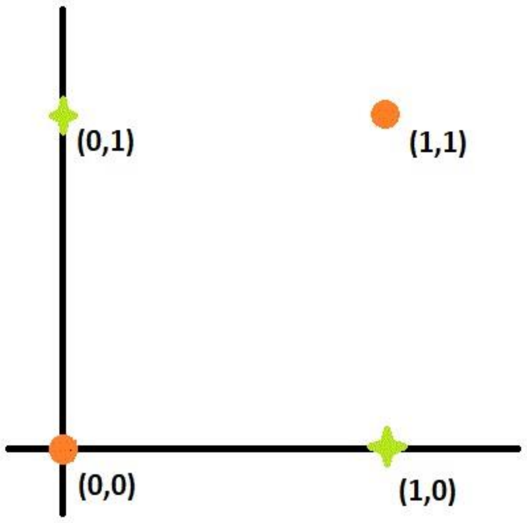
XOr, or Exclusive Or, is a binary logical operator that takes in Boolean inputs and gives out True if and only if the two inputs are different. This logical operator is especially useful when we want to check two conditions that can't be simultaneously true. The following is the Truth table for XOr function



Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

The XOr problem

The XOr problem is that we need to build a Neural Network (a perceptron in our case) to produce the truth table related to the XOr logical operator. This is a binary classification problem. Hence, **supervised learning** is a better way to solve it. In this case, we will be using perceptrons. Uni layered perceptrons can only work with linearly separable data. But in the following diagram drawn in accordance with the truth table of the XOr loical operator, we can see that the data is **NOT** linearly separable.

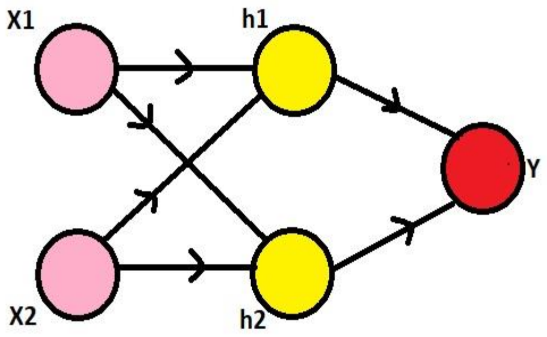


The Solution

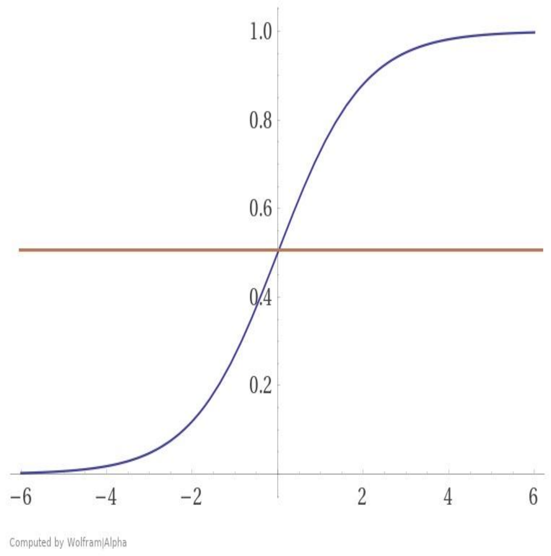
To solve this problem, we add an extra layer to our vanilla perceptron, i.e., we create a **Multi Layered Perceptron** (or **MLP**). We call this extra layer as the **Hidden layer**. To build a perceptron, we first need to understand that the XOr gate can be written as a combination of AND gates, NOT gates and OR gates in the following way:

a **XOr** b = (a **AND NOT** b)**OR**(b**AND NOT**a)

The following is a plan for the perceptron.



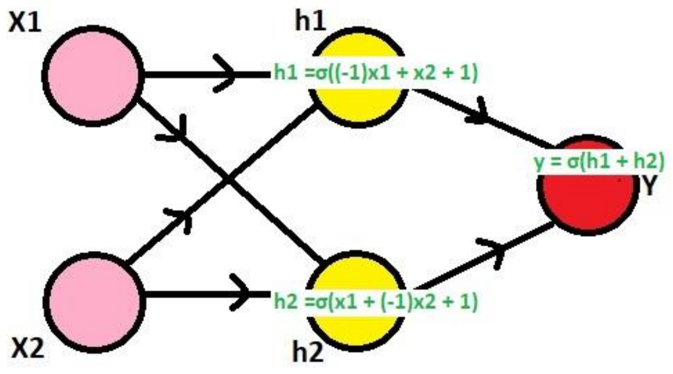
Here, we need to observe that our inputs are 0s and 1s. To make it a XOr gate, we will make the h1 node to perform the (x2 **AND NOT** x1) operation, the h2 node to perform (x1 **AND NOT** x2) operation and the y node to perform (h1 **OR** h2) operation. The NOT gate can be produced for an input a by writing (1-a), the AND gate can be produced for inputs a and b by writing (a.b) and the OR gate can be produced for inputs a and b by writing (a+b). Also, we'll use the sigmoid function as our activation function σ, i.e., σ(x) = 1/(1+e^(-x)) and the threshold for classification would be 0.5, i.e., any x with σ(x)>0.5 will be classified as 1 and others will be classified as 0.



Now, since we have all the information, we can go on to define h1, h2 and y. Using the formulae for AND, NOT and OR gates, we get:

1. h1 = σ((1-x1) + x2) = σ((-1)x1 + x2 + 1)
2. h2 = σ(x1 + (1-x2)) = σ(x1 + (-1)x2 + 1)
3. y = σ(h1 + h2) = σ(h1 + h2 + 0)

Hence, we have built a multi layered perceptron with the following weights and it predicts the output of a XOr logical operator.



Date: ____10/10/2022_ **Signature of faculty in-charge**