Dikshita Kambri

118A2044

BE EXTC A2

EXPERIMENT 2A

MCCULLOCH-PITTS NEURAL NETWORK – AND, OR FUNCTION

EXPERIMENT 2A

MCCULLOCH-PITTS NEURAL NETWORK – AND, OR FUNCTION

AIM: Write a program to implement the a) AND and b) OR functionality using McCulloch-Pitts neural network from given weight and threshold values.

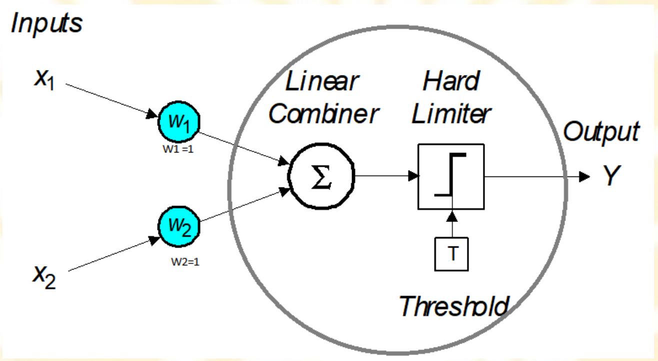
AND function: Inputs are [x1,x2] = [(0,0) ,(0,1) ,(1,0),(1,1)]

Target output is [0 0 0 1]

OR function: Inputs are [x1,x2] = [(0,0) ,(0,1) ,(1,0),(1,1)]

Target output is [0 1 1 1]

Also, study about its noise tolerance of the function.



Hard limiter is defined as:

Y = 1 if $\sum W.X \geq T$ Where T is a Threshold (Assume T =1.5 for AND and 0.5 for OR)

Y = 0 if $\sum W.X < T$

APPARATUS: PC with Python software.

THEORY: McCulloch-Pitts neurons (Fig. could be used to implement logic gates. The neuron uses the hard-limit transfer function hardlimit (threshold) [2]. The proper values of connection weights and neuron thresholds are determined to get the correct outputs for each set of inputs. Each external input is weighted with an appropriate weight *w*, and the sum of the weighted inputs is sent to the hard-limit transfer function, which also has an input of 1 transmitted to it through the bias. The hard-limit transfer function, which returns a 0 or a. The hard-limit transfer function gives a perceptron the ability to classify input vectors by dividing the input space into two regions. The hard-limit transfer function limits the output of the neuron to either 0, if the net input is less than 0; or 1, if *net input* is greater than or equal to 0.

AND function can work for linearly separable data (this is valid for OR also). They can be implemented simply using single neuron.

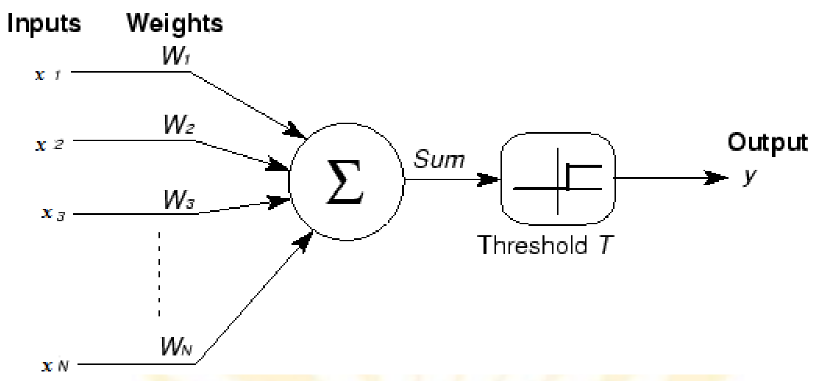


Fig. 1. MP Neuron

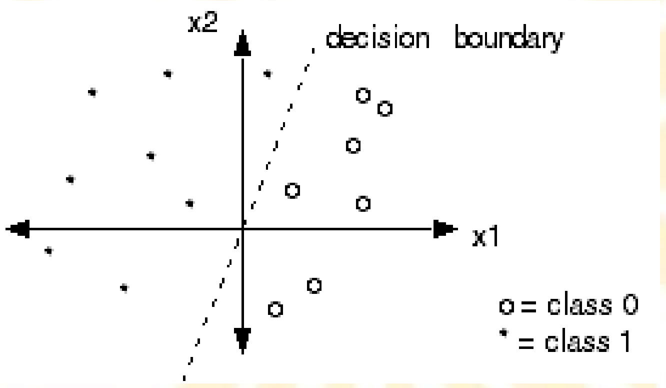


Fig. 2. Linear Classification

In this experiment we assume that we are taking already trained AND function network.

Inputs		Output
x_1	x_2	Y
0	0	0
0	1	0
1	0	0
1	1	1

Case1: x_1= 0; x2 = 0;

When both the inputs are zero when these inputs are multiplied with weights value then the result is less than the threshold value (T=1.5) so the final output (Y) is zero.

Case 2: x_1= 0; x2 = 1;

Case 3: x_1= 1; x2 = 0;

In the above cases the value of result is less than threshold value (T=1.5) and the final output (Y) is zero.

Case 4: x_1= 1; x2 = 1;

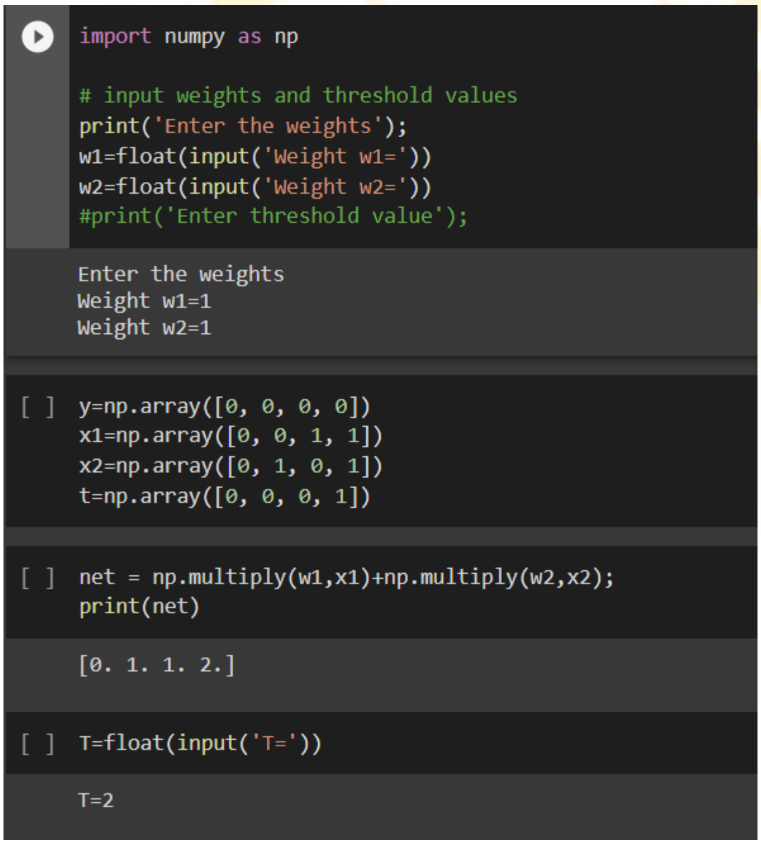
In this case when the both inputs are one and multiply with weight values and the result is more than the threshold value and the final output (Y) is one.

So the network behave like a Logical AND gate.

Similar is the case with OR function but here we change the threshold to 0.5.

OUTPUT:

1) **AND**



```
import numpy as np

# input weights and threshold values
print('Enter the weights');
w1=float(input('Weight w1='))
w2=float(input('Weight w2='))
#print('Enter threshold value');
```

```
Enter the weights
Weight w1=1
Weight w2=1
```

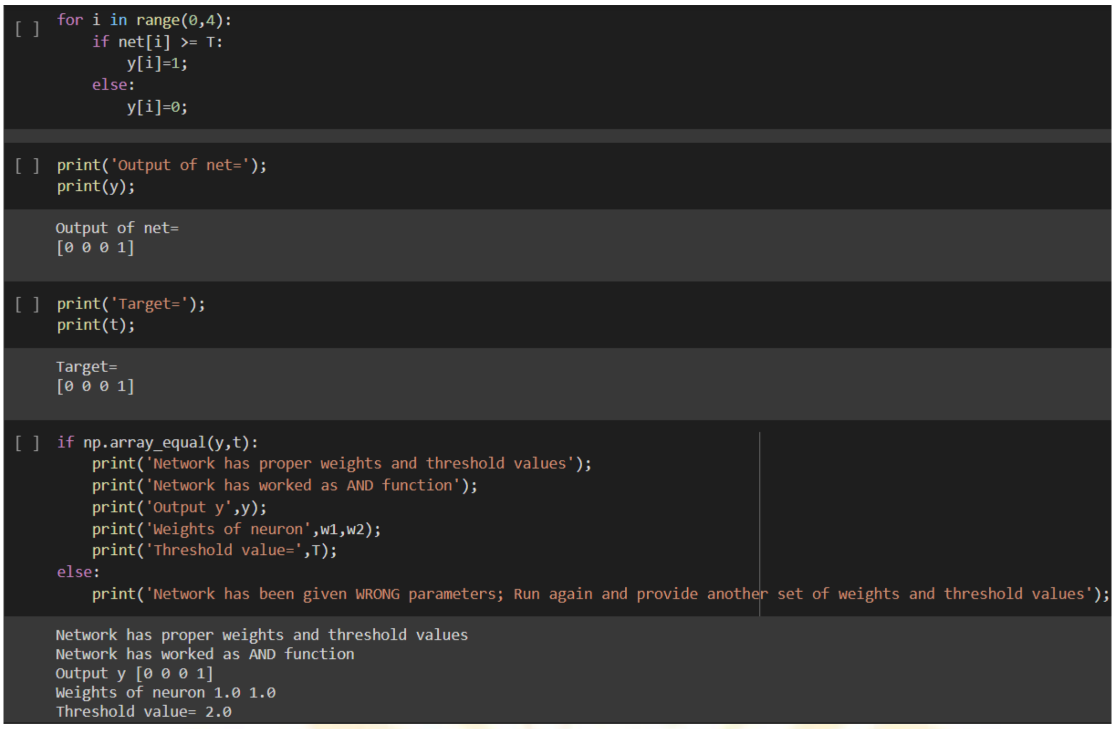
```
[ ]  y=np.array([0, 0, 0, 0])
     x1=np.array([0, 0, 1, 1])
     x2=np.array([0, 1, 0, 1])
     t=np.array([0, 0, 0, 1])
```

```
[ ]  net = np.multiply(w1,x1)+np.multiply(w2,x2);
     print(net)
```

```
[0. 1. 1. 2.]
```

```
[ ]  T=float(input('T='))
```

```
T=2
```



```
for i in range(0,4):
    if net[i] >= T:
        y[i]=1;
    else:
        y[i]=0;
```

```
print('Output of net=');
print(y);
```

```
Output of net=
[0 0 0 1]
```

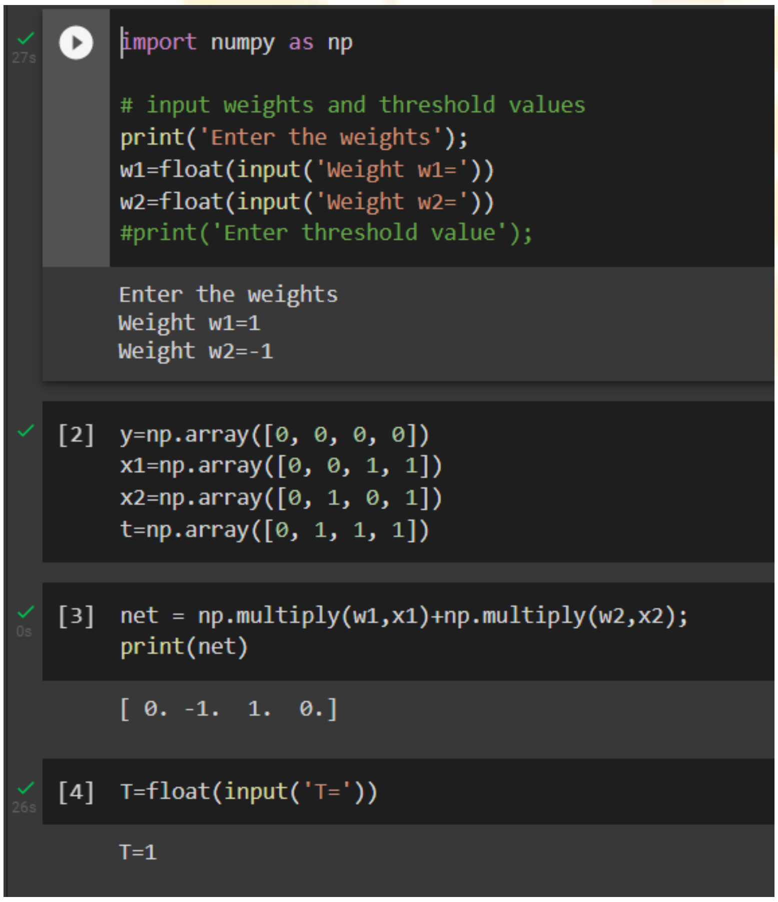
```
print('Target=');
print(t);
```

```
Target=
[0 0 0 1]
```

```
if np.array_equal(y,t):
    print('Network has proper weights and threshold values');
    print('Network has worked as AND function');
    print('Output y',y);
    print('Weights of neuron',w1,w2);
    print('Threshold value=',T);
else:
    print('Network has been given WRONG parameters; Run again and provide another set of weights and threshold values');
```

```
Network has proper weights and threshold values
Network has worked as AND function
Output y [0 0 0 1]
Weights of neuron 1.0 1.0
Threshold value= 2.0
```

2)OR



```
import numpy as np

# input weights and threshold values
print('Enter the weights');
w1=float(input('Weight w1='))
w2=float(input('Weight w2='))
#print('Enter threshold value');
```

```
Enter the weights
Weight w1=1
Weight w2=-1
```

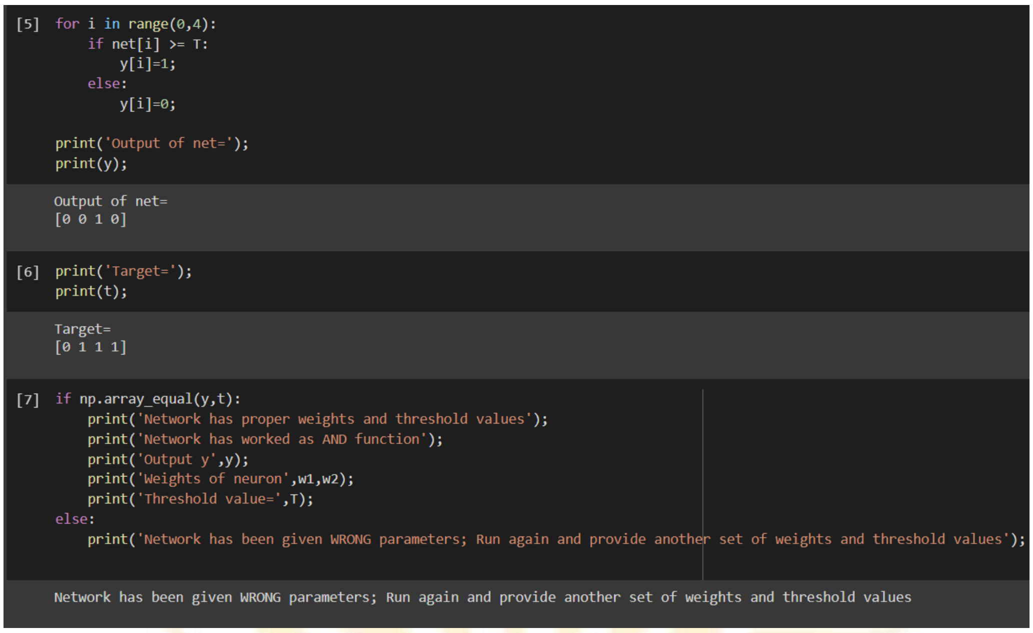
```
[2] y=np.array([0, 0, 0, 0])
    x1=np.array([0, 0, 1, 1])
    x2=np.array([0, 1, 0, 1])
    t=np.array([0, 1, 1, 1])
```

```
[3] net = np.multiply(w1,x1)+np.multiply(w2,x2);
    print(net)
```

```
[ 0. -1.  1.  0.]
```

```
[4] T=float(input('T='))
```

```
T=1
```



```
[5] for i in range(0,4):
        if net[i] >= T:
            y[i]=1;
        else:
            y[i]=0;

    print('Output of net=');
    print(y);

    Output of net=
    [0 0 1 0]

[6] print('Target=');
    print(t);

    Target=
    [0 1 1 1]

[7] if np.array_equal(y,t):
        print('Network has proper weights and threshold values');
        print('Network has worked as AND function');
        print('Output y',y);
        print('Weights of neuron',w1,w2);
        print('Threshold value=',T);
    else:
        print('Network has been given WRONG parameters; Run again and provide another set of weights and threshold values');

    Network has been given WRONG parameters; Run again and provide another set of weights and threshold values
```

CONCLUSION:

We performed the AND gate and OR gate using MP neuron and found that,

1) W1=1, w2=1 and threshold= 3 are proper weights and threshold values for AND gate but not for OR gate.
2) So network worked as a AND function.

EXPERIMENT 2B

MCCULLOCH-PITTS NEURAL NETWORK – EX-OR FUNCTION

EXPERIMENT 2B

MCCULLOCH-PITTS NEURAL NETWORK – EX-OR FUNCTION

AIM: Implementation of two input EX-OR function using neural networks in Python from given weight and threshold values.

APPARATUS: PC with Python

THEORY:

The classification problem can be better explained by plotting the output in n-dimensional feature space as shown below.

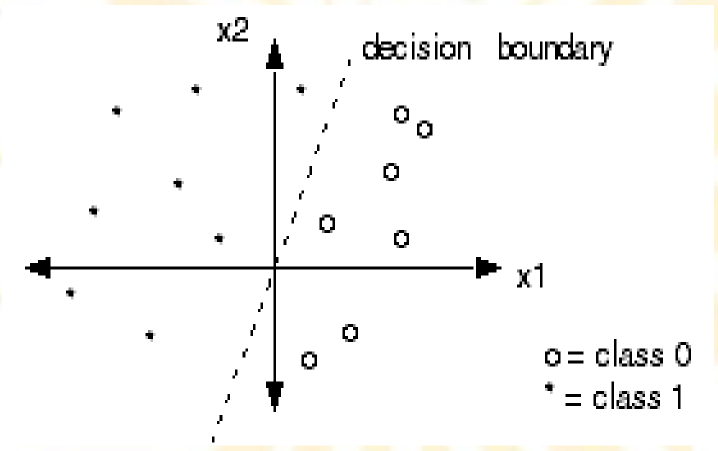


Fig. 1. Linear classification

Classification problems for which there is a line that exactly separates the data classes (decision boundary) are called linearly separable. AND and OR functions with single neuron are able to solve linearly separable problems only. However, most of the real world data classes are not linearly separable [2][3].
A multilayer perceptron (MLP NN) is a feedforward neural network with one or more hidden layers. The network consists of an **input layer** of source neurons, at least one middle or **hidden layer** of computational neurons, and an **output layer** of computational neurons. The input signals are propagated in a forward direction on a layer-by-layer basis. The multilayer perceptron neural network helps in classification of non-linear data.

The Exclusive–Or function is the most commonly known non-linearly separable classification problem. Here two boundaries are required to solve the classification problem as shown in Fig. 2.

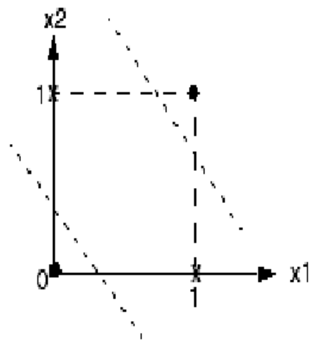


Fig. 2 Non-linear classification in EX-OR

Table for EX-OR function

x1	x2	y
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

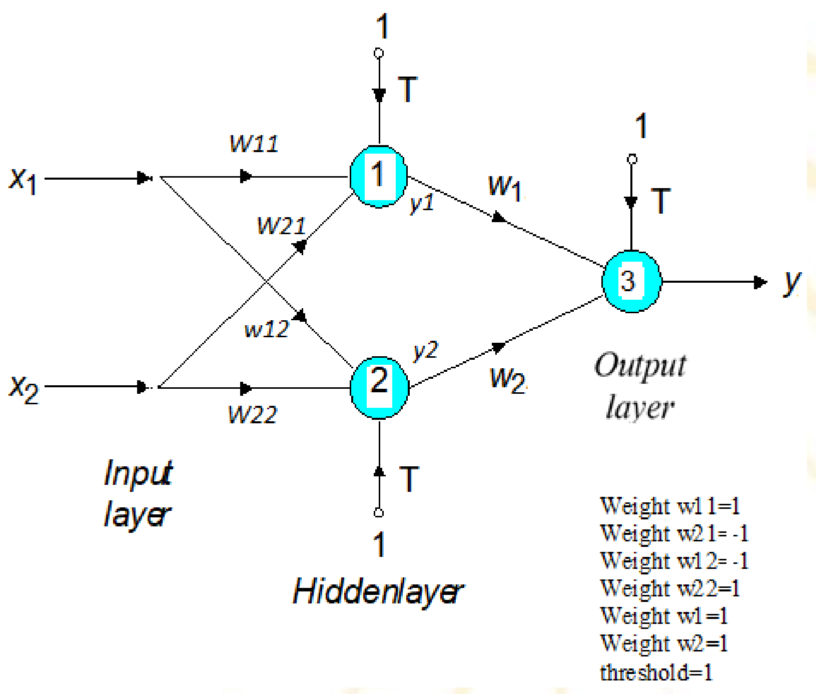
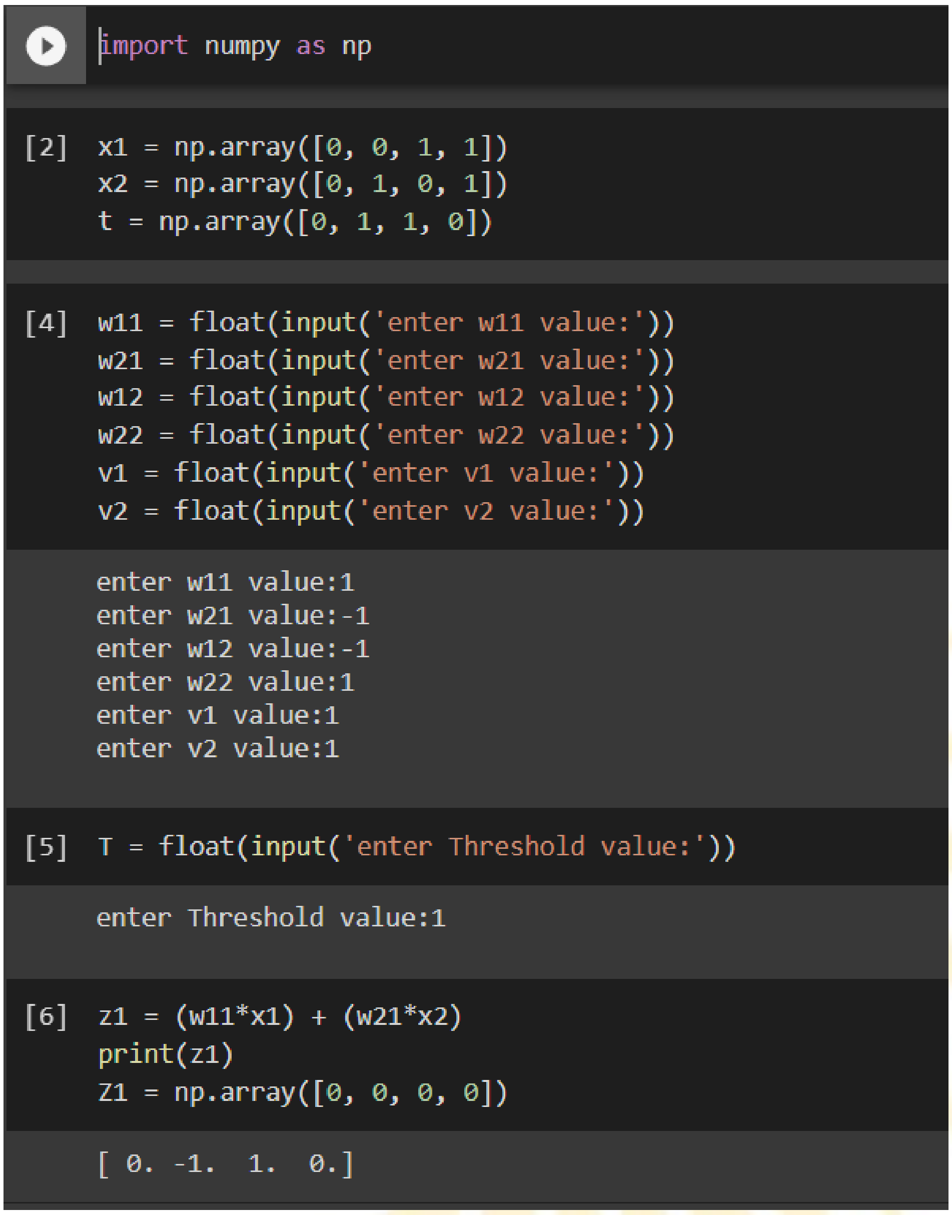


Fig. 3 Network for Exclusive-OR operation

OUTPUT:



```
import numpy as np
```

```
[2] x1 = np.array([0, 0, 1, 1])
    x2 = np.array([0, 1, 0, 1])
    t = np.array([0, 1, 1, 0])
```

```
[4] w11 = float(input('enter w11 value:'))
    w21 = float(input('enter w21 value:'))
    w12 = float(input('enter w12 value:'))
    w22 = float(input('enter w22 value:'))
    v1 = float(input('enter v1 value:'))
    v2 = float(input('enter v2 value:'))
```

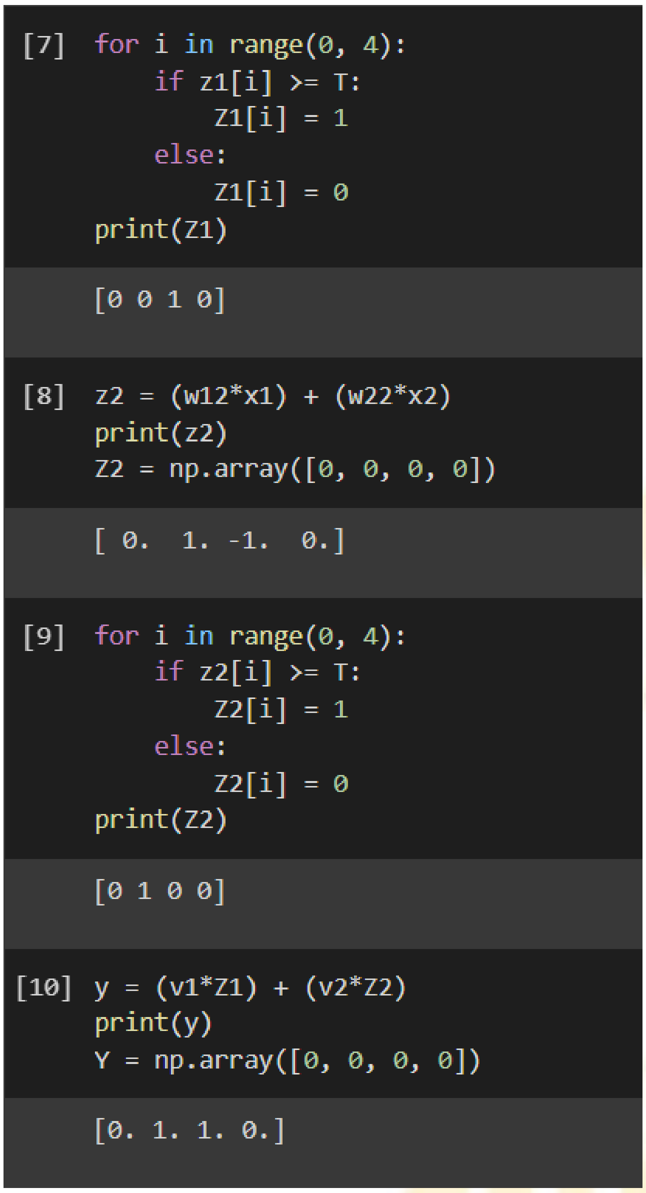
```
    enter w11 value:1
    enter w21 value:-1
    enter w12 value:-1
    enter w22 value:1
    enter v1 value:1
    enter v2 value:1
```

```
[5] T = float(input('enter Threshold value:'))
```

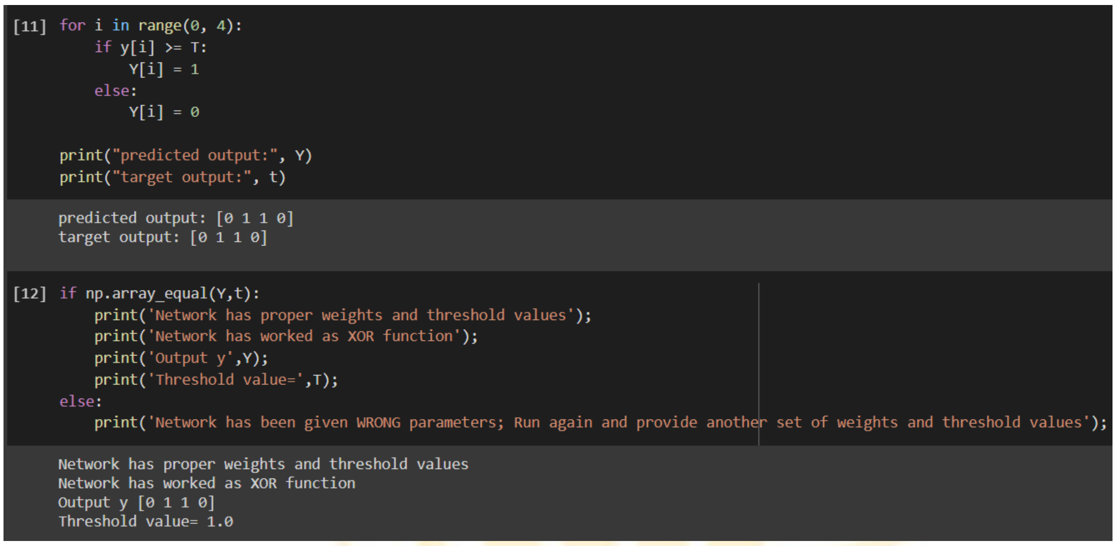
```
    enter Threshold value:1
```

```
[6] z1 = (w11*x1) + (w21*x2)
    print(z1)
    Z1 = np.array([0, 0, 0, 0])
```

```
    [ 0. -1.  1.  0.]
```



```
[7]  for i in range(0, 4):
         if z1[i] >= T:
             Z1[i] = 1
         else:
             Z1[i] = 0
     print(Z1)

     [0 0 1 0]

[8]  z2 = (w12*x1) + (w22*x2)
     print(z2)
     Z2 = np.array([0, 0, 0, 0])

     [ 0.  1. -1.  0.]

[9]  for i in range(0, 4):
         if z2[i] >= T:
             Z2[i] = 1
         else:
             Z2[i] = 0
     print(Z2)

     [0 1 0 0]

[10] y = (v1*Z1) + (v2*Z2)
     print(y)
     Y = np.array([0, 0, 0, 0])

     [0. 1. 1. 0.]
```



```
[11] for i in range(0, 4):
         if y[i] >= T:
             Y[i] = 1
         else:
             Y[i] = 0

     print("predicted output:", Y)
     print("target output:", t)

     predicted output: [0 1 1 0]
     target output: [0 1 1 0]

[12] if np.array_equal(Y,t):
         print('Network has proper weights and threshold values');
         print('Network has worked as XOR function');
         print('Output y',Y);
         print('Threshold value=',T);
     else:
         print('Network has been given WRONG parameters; Run again and provide another set of weights and threshold values');

     Network has proper weights and threshold values
     Network has worked as XOR function
     Output y [0 1 1 0]
     Threshold value= 1.0
```

CONCLUSION:

We performed the XOR gate using MP neuron and found that,

1) W11=1, W21=-1, W12=-1, W22=1, V1=1, V2=1 and threshold= 1 are proper weights and threshold values XOR gate.
2) So, network worked as a XOR function.