Aim: Develop an algorithm using Hebbian Learning rule to demonstrate NAND and NOR logic gates' weights and bias.

Theory: Hebbian Learning Rule, also known as Hebb Learning Rule, was proposed by Donald O Hebb.It is a single layer neural network, valid for only one input layer and one output layer. Hebbian rule works by updating the weights between neurons in the neural network for each training sample.

Hebbian Learning Rule Algorithm:

1. Set all weights to zero, wi = 0 for i=1 to n, and bias to zero.

2. For each input vector, S(input vector) : t(target output pair).

3. Set activations for input units with the input vector Xi = Si for i = 1 to n.

4. Set the corresponding output value to the output neuron, i.e. y = t.

5. Update weight and bias by applying Hebb rule for all i = 1 to n.

Codes and Results:

```
#hebb algorithm for weights and bias learning

import numpy as np

import matplotlib.pyplot as plt

import random as rd



def hebb_weight(x):

    w1,w2,b = 0,0,0

    print(f'Initially w1 = {w1}, w2 = {w2}, b = {b}')

    print(f'(x1, x2)  y  (x1*y, x2*y, y) (w1, w2, b)')

    for x1,x2,y in x:

        w1 += x1*y

        w2 += x2*y

        b += y

        print(f'({x1:2}, {x2:2}) {y:2}  ({x1*y:2}, {x2*y:2}, {y:2})    ({w1:2},
{w2:2}, {b:2})')
```

```
    print(f'Finally w1 = {w1}, w2 = {w2}, b = {b} \n')


NAND_input =  [
        [ 1, 1, -1],
        [ 1, -1, 1],
        [-1, 1, 1],
        [-1, -1, 1]
    ]


NOR_input = [
        [ 1, 1, -1],
        [ 1, -1, -1],
        [-1, 1, -1],
        [-1, -1, 1]
    ]
print(f'for NAND Gate weight and bias are:')
hebb_weight(NAND_input)
print(f'for NOR Gate weight and bias are:')
hebb_weight(NOR_input)
```

```
for NAND Gate weight and bias are:
Initially w1 = 0, w2 = 0, b = 0
(x1, x2)  y  (x1*y, x2*y, y) (w1, w2, b)
( 1,  1) -1  (-1, -1, -1)    (-1, -1, -1)
( 1, -1)  1  ( 1, -1,  1)    ( 0, -2,  0)
(-1,  1)  1  (-1,  1,  1)    (-1, -1,  1)
(-1, -1)  1  (-1, -1,  1)    (-2, -2,  2)
Finally w1 = -2, w2 = -2, b = 2

for NOR Gate weight and bias are:
Initially w1 = 0, w2 = 0, b = 0
(x1, x2)  y  (x1*y, x2*y, y) (w1, w2, b)
( 1,  1) -1  (-1, -1, -1)    (-1, -1, -1)
( 1, -1) -1  (-1,  1, -1)    (-2,  0, -2)
(-1,  1) -1  ( 1, -1, -1)    (-1, -1, -3)
(-1, -1)  1  (-1, -1,  1)    (-2, -2, -2)
Finally w1 = -2, w2 = -2, b = -2
```

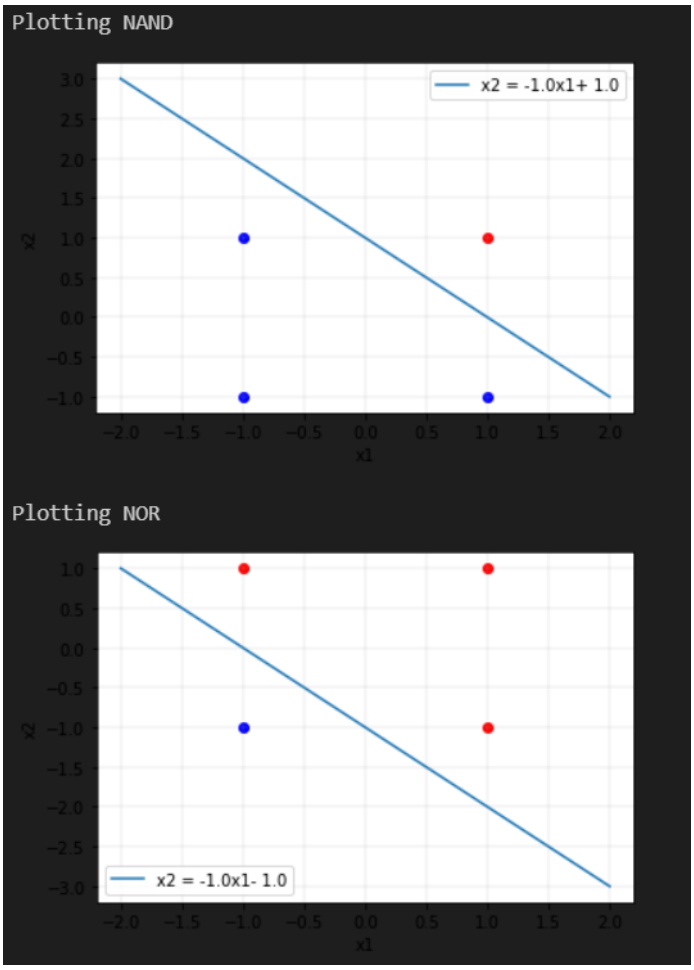
```python
def Plotting_hebbian_rule(X,y):
    n = len(X)
    w = np.array([0 for i in range(len(X[0]))])
    for i in range(n):
        x = np.array(X[i])
        w = w + x*y[i]
    b = X[0][-1]
    x1 = np.linspace(-2,2,10)
    x2 = (-w[0]*x1-w[2]*b)/w[1]
    label = f'x2 = {-w[0]/w[1]}x1'
```

```
    if -w[2]/w[1]>0:
        label += f'+ {-w[2]/w[1]}'
    else:
        label += f'- {w[2]/w[1]}'
    plt.plot(x1, x2, label=label)
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.legend()
    plt.grid(linewidth=0.2)

    for i,j in zip(X,y):
        if j==1:
            plt.scatter(i[0],i[1],c='b',marker="o")
        else:
            plt.scatter(i[0],i[1],c='r',marker="o")
    plt.show()

print("Plotting NAND")
X = [[-1,-1,1],[-1,1,1],[1,-1,1],[1,1,1]]
y = [1,1,1,-1]
Plotting_hebbian_rule(X,y)

print("Plotting NOR")
X = [[-1,-1,1],[-1,1,1],[1,-1,1],[1,1,1]]
y = [1,-1,-1,-1]
Plotting_hebbian_rule(X,y)
```



Conclusion: In this experiment we have learnt about the Hebbian Postulates and resolve the bias and weights issues faced earlier, we also plotted the decision boundaries for NAND and NOR gates.

By

Heramba Panda

118EI0372