1. Implement the perceptron learning single layer algorithm by initializing the weights and threshold. Execute the code and check, how many iterations are needed, until the network coverage.
   USN-1NT18CS133

Explanation:
Perceptron consist of four parts-
a. Input values or one input layer: The input layer of a perceptron is made of artificial input neurons and brings the initial data into the system for further processing.
b. Weights: Weight represents the strength or dimension of the connection between units. If the weight from node 1 to node 2 has the greater quantity, then neuron 1 has greater influence over neuron 2. How much influence of the input will have on the output, is determined by weight.
c. Bias is similar to the intercept added in a linear equation. It is an additional parameter which task is to adjust the output along with the weighted sum of the inputs to the neuron.
d. Activation Function: A neuron should be activated or not, determined by an activation function. It calculates a weighted sum and further adds bias to the given result.

In [2]:

```python
import numpy as np
theta = 1
epoch = 3

class Perceptron(object):
    def __init__(self, input_size, learning_rate=0.2):
        self.learning_rate = learning_rate
        self.weights = np.zeros(input_size + 1) # zero init for weights and bias

    def predict(self, x):
         return (np.dot(x, self.weights[1:]) + self.weights[0]) # X.W + B

    def train(self, x, y, weights):
        for inputs, label in zip(x, y):
            net_in = self.predict(inputs)
            if net_in > theta:
                y_out = 1
            elif net_in < -theta:
                y_out = -1
            else:
                y_out = 0
            if y_out != label: # updating the net on incorrect prediction
                self.weights[1:] += self.learning_rate * label * inputs # W = alpha * Y * X
                self.weights[0] += self.learning_rate * label   # B = alpha * Y
            print(inputs, net_in, label, y_out, self.weights)

if __name__ == "__main__":
    x = []
    x.append(np.array([1, 1]))
    x.append(np.array([1, -1]))
    x.append(np.array([-1, 1]))
    x.append(np.array([-1, -1]))

    y = np.array([1, -1, -1, -1])

    perceptron = Perceptron(2)

    for i in range(epoch):
        print("Epoch",i)
        print("X1 X2 ", " Net ", " T ", " Y ", " B Weights")
        weights = perceptron.weights
        print("Initial Weights", weights)
        perceptron.train(x, y, weights)
```

```
Epoch 0
X1 X2   Net   T   Y   B Weights
Initial Weights [0. 0. 0.]
[1 1] 0.0 1 0 [0.2 0.2 0.2]
[ 1 -1] 0.2 -1 0 [0.  0.  0.4]
[-1  1] 0.4 -1 0 [-0.2  0.2  0.2]
[-1 -1] -0.6000000000000001 -1 0 [-0.4  0.4  0.4]
Epoch 1
X1 X2   Net   T   Y   B Weights
Initial Weights [-0.4  0.4  0.4]
[1 1] 0.4 1 0 [-0.2  0.6  0.6]
[ 1 -1] -0.2 -1 0 [-0.4  0.4  0.8]
[-1  1] -5.551115123125783e-17 -1 0 [-0.6  0.6  0.6]
[-1 -1] -1.8000000000000003 -1 -1 [-0.6  0.6  0.6]
Epoch 2
X1 X2   Net   T   Y   B Weights
Initial Weights [-0.6  0.6  0.6]
[1 1] 0.6000000000000001 1 0 [-0.4  0.8  0.8]
[ 1 -1] -0.4000000000000001 -1 0 [-0.6  0.6  1. ]
[-1  1] -0.20000000000000018 -1 0 [-0.8  0.8  0.8]
[-1 -1] -2.4000000000000004 -1 -1 [-0.8  0.8  0.8]
```

In [ ]: