In [1]:
```
# implement perception using AND gate
#Define the input vector for AND gate
import numpy as np
```

In [2]:
```
x=np.array([[1,1],[1,-1],[-1,1],[-1,-1]])
```

In [3]:
```
#define target
t=np.array([1,-1,-1,-1])
```

In [4]:
```
#Initialize the weight alpha,bias,theta
weights=np.array([0.0,0.0])
bias=0.0
alpha=1 #learning rate
theta=0
```

In [5]:
```
#define the activation function
def activation(yin):
    return 1 if yin>theta else -1 if yin<theta else 0
```

In [6]:
```
#training
epochs=10
for e in range (epochs):
    print("\n Epoch:",(e+1))
    #calculate net input
    for i in range(len(x)):
        x1=x[i]
    t1=t[i]
    yin=np.dot(weights,x1)+bias
    y=activation(yin)  #predicted y

    #compute the errror
    err=t1-y
    if(err!=0):
        #update the weight
        weights+=alpha*t1*x1
        bias+=alpha*t1
    print("Input:",x1,"Taget:",t1,"weight:",weights,"Bias:",bias)
```

```
 Epoch: 1
Input: [-1 -1] Taget: -1 weight: [1. 1.] Bias: -1.0

 Epoch: 2
Input: [-1 -1] Taget: -1 weight: [1. 1.] Bias: -1.0

 Epoch: 3
Input: [-1 -1] Taget: -1 weight: [1. 1.] Bias: -1.0

 Epoch: 4
Input: [-1 -1] Taget: -1 weight: [1. 1.] Bias: -1.0

 Epoch: 5
Input: [-1 -1] Taget: -1 weight: [1. 1.] Bias: -1.0

 Epoch: 6
Input: [-1 -1] Taget: -1 weight: [1. 1.] Bias: -1.0

 Epoch: 7
Input: [-1 -1] Taget: -1 weight: [1. 1.] Bias: -1.0

 Epoch: 8
Input: [-1 -1] Taget: -1 weight: [1. 1.] Bias: -1.0

 Epoch: 9
Input: [-1 -1] Taget: -1 weight: [1. 1.] Bias: -1.0

 Epoch: 10
Input: [-1 -1] Taget: -1 weight: [1. 1.] Bias: -1.0
```

In [ ]: