```
#Perceptron learning Algorithm for Implementing AND Logic Gate

import numpy as np
# for AND gate (Input/Output)
x_list = np.array([[0,0],[0,1],[1,0],[1,1]])
y_list = np.array([0,0,0,1])
epochs = 4
w,b = np.zeros(2),0

for _ in range(epochs):
  for x,y in zip(x_list,y_list):
    z = np.dot(x,w) + b
    ### condition area for y_pred
    y_pred = 1 if z >= 0 else 0
    E = y - y_pred
    print("error : ",E,"<- at epoch",_)
    w += E*x
    b += E
    print(w,b)
```

```
error :  -1 <- at epoch 0
[0. 0.] -1
error :  0 <- at epoch 0
[0. 0.] -1
error :  0 <- at epoch 0
[0. 0.] -1
error :  1 <- at epoch 0
[1. 1.] 0
error :  -1 <- at epoch 1
[1. 1.] -1
error :  -1 <- at epoch 1
[1. 0.] -2
error :  0 <- at epoch 1
[1. 0.] -2
error :  1 <- at epoch 1
[2. 1.] -1
error :  0 <- at epoch 2
[2. 1.] -1
error :  -1 <- at epoch 2
[2. 0.] -2
error :  -1 <- at epoch 2
[1. 0.] -3
error :  1 <- at epoch 2
[2. 1.] -2
error :  0 <- at epoch 3
[2. 1.] -2
error :  0 <- at epoch 3
[2. 1.] -2
error :  -1 <- at epoch 3
[1. 1.] -3
error :  1 <- at epoch 3
[2. 2.] -2
```

• Change the epochs and observe the output

```
#Perceptron learning Algorithm for Implementing AND Logic Gate

import numpy as np
# for AND gate (Input/Output)
x_list = np.array([[0,0],[0,1],[1,0],[1,1]])
y_list = np.array([0,0,0,1])
epochs = 6
w,b = np.zeros(2),0

for _ in range(epochs):
  for x,y in zip(x_list,y_list):
    z = np.dot(x,w) + b
    ### condition area for y_pred
    y_pred = 1 if z >= 0 else 0
    E = y - y_pred
    print("error : ",E,"<- at epoch",_)
    w += E*x
    b += E
    print(w,b)
```

```
error :  -1 <- at epoch 0
[0. 0.] -1
error :  0 <- at epoch 0
[0. 0.] -1
error :  0 <- at epoch 0
```

```
[0. 0.] -1
error :  1 <- at epoch 0
[1. 1.] 0
error :  -1 <- at epoch 1
[1. 1.] -1
error :  -1 <- at epoch 1
[1. 0.] -2
error :  0 <- at epoch 1
[1. 0.] -2
error :  1 <- at epoch 1
[2. 1.] -1
error :  0 <- at epoch 2
[2. 1.] -1
error :  -1 <- at epoch 2
[2. 0.] -2
error :  -1 <- at epoch 2
[1. 0.] -3
error :  1 <- at epoch 2
[2. 1.] -2
error :  0 <- at epoch 3
[2. 1.] -2
error :  0 <- at epoch 3
[2. 1.] -2
error :  -1 <- at epoch 3
[1. 1.] -3
error :  1 <- at epoch 3
[2. 2.] -2
error :  0 <- at epoch 4
[2. 2.] -2
error :  -1 <- at epoch 4
[2. 1.] -3
error :  0 <- at epoch 4
[2. 1.] -3
error :  0 <- at epoch 4
[2. 1.] -3
error :  0 <- at epoch 5
[2. 1.] -3
error :  0 <- at epoch 5
[2. 1.] -3
error :  0 <- at epoch 5
[2. 1.] -3
error :  0 <- at epoch 5
[2. 1.] -3
```

change the random weight and biad and observe the convergence of the network.

```
#Perceptron learning Algorithm for Implementing AND Logic Gate

import numpy as np
# for AND gate (Input/Output)
x_list = np.array([[0,0],[0,1],[1,0],[1,1]])
y_list = np.array([0,0,0,1])
epochs = 6
w = np.array([0.2,-0.8])
b = 0.2
for _ in range(epochs):
  for x,y in zip(x_list,y_list):
    z = np.dot(x,w) + b
    ### condition area for y_pred
    y_pred = 1 if z >= 0 else 0
    E = y - y_pred
    print("error : ",E,"<- at epoch",_)
    w += E*x
    b += E
    print(w,b)
```

```
error :  -1 <- at epoch 0
[ 0.2 -0.8] -0.8
error :  0 <- at epoch 0
[ 0.2 -0.8] -0.8
error :  0 <- at epoch 0
[ 0.2 -0.8] -0.8
error :  1 <- at epoch 0
[1.2 0.2] 0.19999999999999996
error :  -1 <- at epoch 1
[1.2 0.2] -0.8
error :  0 <- at epoch 1
[1.2 0.2] -0.8
error :  -1 <- at epoch 1
[0.2 0.2] -1.8
error :  1 <- at epoch 1
[1.2 1.2] -0.8
error :  0 <- at epoch 2
[1.2 1.2] -0.8
error :  -1 <- at epoch 2
```

```
[1.2 0.2] -1.8
error :  0 <- at epoch 2
[1.2 0.2] -1.8
error :  1 <- at epoch 2
[2.2 1.2] -0.8
error :  0 <- at epoch 3
[2.2 1.2] -0.8
error :  -1 <- at epoch 3
[2.2 0.2] -1.8
error :  -1 <- at epoch 3
[1.2 0.2] -2.8
error :  1 <- at epoch 3
[2.2 1.2] -1.7999999999999998
error :  0 <- at epoch 4
[2.2 1.2] -1.7999999999999998
error :  0 <- at epoch 4
[2.2 1.2] -1.7999999999999998
error :  -1 <- at epoch 4
[1.2 1.2] -2.8
error :  1 <- at epoch 4
[2.2 2.2] -1.7999999999999998
error :  0 <- at epoch 5
[2.2 2.2] -1.7999999999999998
error :  -1 <- at epoch 5
[2.2 1.2] -2.8
error :  0 <- at epoch 5
[2.2 1.2] -2.8
error :  0 <- at epoch 5
[2.2 1.2] -2.8
```

• Add logic to stop the training when the error is ‘zero’ for all the inputs.

```
# Perceptron learning Algorithm for Implementing AND Logic Gate

import numpy as np

# AND gate data
x_list = np.array([[0,0],[0,1],[1,0],[1,1]])
y_list = np.array([0,0,0,1])

epochs = 6
w, b = np.zeros(2), 0

for epoch in range(epochs):
    print(f"\nEpoch {epoch+1}")
    error_found = False   # NEW

    for x, y in zip(x_list, y_list):
        z = np.dot(x, w) + b
        y_pred = 1 if z >= 0 else 0
        E = y - y_pred
        print("error:", E)

        if E != 0:
            error_found = True   # NEW

        w += E * x
        b += E
        print("weights:", w, "bias:", b)

    # STOP condition
    if not error_found:
        print("\nTraining stopped: error is zero for all inputs ✅")
        break
```

```
Epoch 1
error: -1
weights: [0. 0.] bias: -1
error: 0
weights: [0. 0.] bias: -1
error: 0
weights: [0. 0.] bias: -1
error: 1
weights: [1. 1.] bias: 0

Epoch 2
error: -1
weights: [1. 1.] bias: -1
error: -1
weights: [1. 0.] bias: -2
error: 0
weights: [1. 0.] bias: -2
error: 1
weights: [2. 1.] bias: -1
```

```
Epoch 3
error: 0
weights: [2. 1.] bias: -1
error: -1
weights: [2. 0.] bias: -2
error: -1
weights: [1. 0.] bias: -3
error: 1
weights: [2. 1.] bias: -2

Epoch 4
error: 0
weights: [2. 1.] bias: -2
error: 0
weights: [2. 1.] bias: -2
error: -1
weights: [1. 1.] bias: -3
error: 1
weights: [2. 2.] bias: -2

Epoch 5
error: 0
weights: [2. 2.] bias: -2
error: -1
weights: [2. 1.] bias: -3
error: 0
weights: [2. 1.] bias: -3
error: 0
weights: [2. 1.] bias: -3

Epoch 6
error: 0
weights: [2. 1.] bias: -3
error: 0
weights: [2. 1.] bias: -3
error: 0
weights: [2. 1.] bias: -3
```

or gate

```
import numpy as np
# for AND gate (Input/Output)
x_list = np.array([[0,0],[0,1],[1,0],[1,1]])
y_list = np.array([0,1,1,1])
epochs = 7
w,b = np.zeros(2),0

for _ in range(epochs):
  for x,y in zip(x_list,y_list):
    z = np.dot(x,w) + b
    ### condition area for y_pred
    y_pred = 1 if z >= 0 else 0
    E = y - y_pred
    print("error : ",E,"<- at epoch",_)
    w += E*x
    b += E
    print(w,b)
```

```
error :  -1 <- at epoch 0
[0. 0.] -1
error :  1 <- at epoch 0
[0. 1.] 0
error :  0 <- at epoch 0
[0. 1.] 0
error :  0 <- at epoch 0
[0. 1.] 0
error :  -1 <- at epoch 1
[0. 1.] -1
error :  0 <- at epoch 1
[0. 1.] -1
error :  1 <- at epoch 1
[1. 1.] 0
error :  0 <- at epoch 1
[1. 1.] 0
error :  -1 <- at epoch 2
[1. 1.] -1
error :  0 <- at epoch 2
[1. 1.] -1
error :  0 <- at epoch 2
[1. 1.] -1
error :  0 <- at epoch 2
[1. 1.] -1
error :  0 <- at epoch 3
[1. 1.] -1
```

```
error :  0 <- at epoch 3
[1. 1.] -1
error :  0 <- at epoch 3
[1. 1.] -1
error :  0 <- at epoch 3
[1. 1.] -1
error :  0 <- at epoch 4
[1. 1.] -1
error :  0 <- at epoch 4
[1. 1.] -1
error :  0 <- at epoch 4
[1. 1.] -1
error :  0 <- at epoch 4
[1. 1.] -1
error :  0 <- at epoch 5
[1. 1.] -1
error :  0 <- at epoch 5
[1. 1.] -1
error :  0 <- at epoch 5
[1. 1.] -1
error :  0 <- at epoch 5
[1. 1.] -1
error :  0 <- at epoch 6
[1. 1.] -1
error :  0 <- at epoch 6
[1. 1.] -1
error :  0 <- at epoch 6
[1. 1.] -1
error :  0 <- at epoch 6
[1. 1.] -1
```

xor gate

```
import numpy as np
# for AND gate (Input/Output)
x_list = np.array([[0,0],[0,1],[1,0],[1,1]])
y_list = np.array([0,1,1,0])
epochs = 7
w,b = np.zeros(2),0

for _ in range(epochs):
  for x,y in zip(x_list,y_list):
    z = np.dot(x,w) + b
    ### condition area for y_pred
    y_pred = 1 if z >= 0 else 0
    E = y - y_pred
    print("error : ",E,"<- at epoch",_)
    w += E*x
    b += E
    print(w,b)
```

```
error :  -1 <- at epoch 0
[0. 0.] -1
error :  1 <- at epoch 0
[0. 1.] 0
error :  0 <- at epoch 0
[0. 1.] 0
error :  -1 <- at epoch 0
[-1.  0.] -1
error :  0 <- at epoch 1
[-1.  0.] -1
error :  1 <- at epoch 1
[-1.  1.] 0
error :  1 <- at epoch 1
[0. 1.] 1
error :  -1 <- at epoch 1
[-1.  0.] 0
error :  -1 <- at epoch 2
[-1.  0.] -1
error :  1 <- at epoch 2
[-1.  1.] 0
error :  1 <- at epoch 2
[0. 1.] 1
error :  -1 <- at epoch 2
[-1.  0.] 0
error :  -1 <- at epoch 3
[-1.  0.] -1
error :  1 <- at epoch 3
[-1.  1.] 0
error :  1 <- at epoch 3
[0. 1.] 1
error :  -1 <- at epoch 3
[-1.  0.] 0
error :  -1 <- at epoch 4
```

```
[-1.  0.] -1
error :  1 <- at epoch 4
[-1.  1.] 0
error :  1 <- at epoch 4
[0. 1.] 1
error :  -1 <- at epoch 4
[-1.  0.] 0
error :  -1 <- at epoch 5
[-1.  0.] -1
error :  1 <- at epoch 5
[-1.  1.] 0
error :  1 <- at epoch 5
[0. 1.] 1
error :  -1 <- at epoch 5
[-1.  0.] 0
error :  -1 <- at epoch 6
[-1.  0.] -1
error :  1 <- at epoch 6
[-1.  1.] 0
error :  1 <- at epoch 6
[0. 1.] 1
error :  -1 <- at epoch 6
[-1.  0.] 0
```