

Raj Borad Batch : E2 Pract -8/9

practical -8

```

1 import numpy as np
2
3
4 X = np.array([
5     [0, 0],
6     [0, 1],
7     [1, 0],
8     [1, 1]
9 ])
10
11 Y = np.array([0, 0, 0, 1])
12
13
14 weights = np.random.rand(2)
15 bias = np.random.rand(1)
16 learning_rate = 0.1
17
18
19 def activation(x):
20     return 1 if x >= 0 else 0
21
22 epochs = 10
23 for epoch in range(epochs):
24     print(f"Epoch {epoch + 1}")
25     for i in range(len(X)):
26         linear_output = np.dot(X[i], weights) + bias
27         y_pred = activation(linear_output)
28
29
30         error = Y[i] - y_pred
31
32
33         weights += learning_rate * error * X[i]
34         bias += learning_rate * error
35
36         print(f"Input: {X[i]}, Predicted: {y_pred}, True: {Y[i]}, Error: {error}")
37     print(f"Weights: {weights}, Bias: {bias}\n")
38
39 # Testing the Perceptron
40 print("Testing AND Gate Perceptron")
41 for i in range(len(X)):
42     linear_output = np.dot(X[i], weights) + bias
43     y_pred = activation(linear_output)
44     print(f"Input: {X[i]}, Predicted: {y_pred}, True: {Y[i]}")
45

```

↩ Epoch 1

```

Input: [0 0], Predicted: 1, True: 0, Error: -1
Input: [0 1], Predicted: 1, True: 0, Error: -1
Input: [1 0], Predicted: 1, True: 0, Error: -1
Input: [1 1], Predicted: 1, True: 1, Error: 0
Weights: [ 0.82208323 -0.06583318], Bias: [0.68884889]

```

Epoch 2

```

Input: [0 0], Predicted: 1, True: 0, Error: -1
Input: [0 1], Predicted: 1, True: 0, Error: -1
Input: [1 0], Predicted: 1, True: 0, Error: -1
Input: [1 1], Predicted: 1, True: 1, Error: 0
Weights: [ 0.72208323 -0.16583318], Bias: [0.38884889]

```

Epoch 3

```

Input: [0 0], Predicted: 1, True: 0, Error: -1
Input: [0 1], Predicted: 1, True: 0, Error: -1
Input: [1 0], Predicted: 1, True: 0, Error: -1
Input: [1 1], Predicted: 1, True: 1, Error: 0
Weights: [ 0.62208323 -0.26583318], Bias: [0.08884889]

```

Epoch 4

```

Input: [0 0], Predicted: 1, True: 0, Error: -1
Input: [0 1], Predicted: 0, True: 0, Error: 0
Input: [1 0], Predicted: 1, True: 0, Error: -1
Input: [1 1], Predicted: 1, True: 1, Error: 0
Weights: [ 0.52208323 -0.26583318], Bias: [-0.11115111]

```

Epoch 5

```

Input: [0 0], Predicted: 0, True: 0, Error: 0
Input: [0 1], Predicted: 0, True: 0, Error: 0
Input: [1 0], Predicted: 1, True: 0, Error: -1

```

```
Input: [1 1], Predicted: 0, True: 1, Error: 1
Weights: [ 0.52208323 -0.16583318], Bias: [-0.11115111]
```

Epoch 6

```
Input: [0 0], Predicted: 0, True: 0, Error: 0
Input: [0 1], Predicted: 0, True: 0, Error: 0
Input: [1 0], Predicted: 1, True: 0, Error: -1
Input: [1 1], Predicted: 1, True: 1, Error: 0
Weights: [ 0.42208323 -0.16583318], Bias: [-0.21115111]
```

Epoch 7

```
Input: [0 0], Predicted: 0, True: 0, Error: 0
Input: [0 1], Predicted: 0, True: 0, Error: 0
Input: [1 0], Predicted: 1, True: 0, Error: -1
Input: [1 1], Predicted: 0, True: 1, Error: 1
Weights: [ 0.42208323 -0.06583318], Bias: [-0.21115111]
```

Epoch 8

```
Input: [0 0], Predicted: 0, True: 0, Error: 0
Input: [0 1], Predicted: 0, True: 0, Error: 0
Input: [1 0], Predicted: 1, True: 0, Error: -1
Input: [1 1], Predicted: 0, True: 1, Error: 1
Weights: [0.42208323 0.03416682], Bias: [-0.21115111]
```

Epoch 9

```
Input: [0 0], Predicted: 0, True: 0, Error: 0
```

```
1 import numpy as np
2
3
4 X = np.array([
5     [0, 0],
6     [0, 1],
7     [1, 0],
8     [1, 1]
9 ])
10
11 Y = np.array([0, 1, 1, 1])
12
13
14 weights = np.random.rand(2)
15 bias = np.random.rand(1)
16 learning_rate = 0.1
17
18 def activation(x):
19     return 1 if x >= 0 else 0
20
21 epochs = 10
22 for epoch in range(epochs):
23     for i in range(len(X)):
24         linear_output = np.dot(X[i], weights) + bias
25         y_pred = activation(linear_output)
26         error = Y[i] - y_pred
27
28         weights += learning_rate * error * X[i]
29         bias += learning_rate * error
30
31 # Testing the Perceptron
32 print("Testing OR Gate Perceptron")
33 for i in range(len(X)):
34     linear_output = np.dot(X[i], weights) + bias
35     y_pred = activation(linear_output)
36     print(f"Input: {X[i]}, Predicted: {y_pred}, True: {Y[i]}")
37
```



Testing OR Gate Perceptron

```
Input: [0 0], Predicted: 0, True: 0
Input: [0 1], Predicted: 1, True: 1
Input: [1 0], Predicted: 1, True: 1
Input: [1 1], Predicted: 1, True: 1
```

Practical -9

```
1 import numpy as np
2
3 X = np.array([
4     [0, 0],
5     [0, 1],
6     [1, 0],
7     [1, 1]
8 ])
9
10 Y = np.array([0, 1, 1, 0])
```

```

11
12 input_size = 2
13 hidden_size = 2
14 output_size = 1
15 learning_rate = 0.1
16 epochs = 10000
17
18 weights_input_hidden = np.random.rand(input_size, hidden_size)
19 bias_hidden = np.random.rand(hidden_size)
20 weights_hidden_output = np.random.rand(hidden_size, output_size)
21 bias_output = np.random.rand(output_size)
22 # print(weights_input_hidden,bias_hidden,weights_hidden_output,bias_output)
23
24 def sigmoid(x):
25     return 1 / (1 + np.exp(-x))
26
27 def sigmoid_derivative(x):
28     return x * (1 - x)
29
30 for epoch in range(epochs):
31     for i in range(len(X)):
32         hidden_input = np.dot(X[i], weights_input_hidden) + bias_hidden
33         hidden_output = sigmoid(hidden_input)
34
35         final_input = np.dot(hidden_output, weights_hidden_output) + bias_output
36         final_output = sigmoid(final_input)
37
38         error = Y[i] - final_output
39
40
41         # Backpropagation
42         d_output = error * sigmoid_derivative(final_output)
43         error_hidden = d_output.dot(weights_hidden_output.T)
44         d_hidden = error_hidden * sigmoid_derivative(hidden_output)
45
46         weights_hidden_output += hidden_output.reshape(-1, 1) * d_output * learning_rate
47         bias_output += d_output * learning_rate
48         weights_input_hidden += X[i].reshape(-1, 1) * d_hidden * learning_rate
49         bias_hidden += d_hidden * learning_rate
50
51
52 print("Testing XOR Gate")
53 for i in range(len(X)):
54     hidden_input = np.dot(X[i], weights_input_hidden) + bias_hidden
55     hidden_output = sigmoid(hidden_input)
56
57     final_input = np.dot(hidden_output, weights_hidden_output) + bias_output
58     final_output = sigmoid(final_input)
59
60     print(f"Input: {X[i]}, Predicted: {round(final_output[0])}, True: {Y[i]}")
61

```

Testing XOR Gate

```

Input: [0 0], Predicted: 0, True: 0
Input: [0 1], Predicted: 1, True: 1
Input: [1 0], Predicted: 1, True: 1
Input: [1 1], Predicted: 0, True: 0

```

1 Start coding or generate with AI.