Raj Borad Batch : E2 Pract -8/9

practical -8

```
import numpy as np


X = np.array([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
])

Y = np.array([0, 0, 0, 1])


weights = np.random.rand(2)
bias = np.random.rand(1)
learning_rate = 0.1


def activation(x):
    return 1 if x >= 0 else 0

epochs = 10
for epoch in range(epochs):
    print(f"Epoch {epoch + 1}")
    for i in range(len(X)):
        linear_output = np.dot(X[i], weights) + bias
        y_pred = activation(linear_output)


        error = Y[i] - y_pred


        weights += learning_rate * error * X[i]
        bias += learning_rate * error

        print(f"Input: {X[i]}, Predicted: {y_pred}, True: {Y[i]}, Error: {error}")
    print(f"Weights: {weights}, Bias: {bias}\n")

# Testing the Perceptron
print("Testing AND Gate Perceptron")
for i in range(len(X)):
    linear_output = np.dot(X[i], weights) + bias
    y_pred = activation(linear_output)
    print(f"Input: {X[i]}, Predicted: {y_pred}, True: {Y[i]}")

```

```
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
```

```python
import numpy as np


X = np.array([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
])

Y = np.array([0, 1, 1, 1])


weights = np.random.rand(2)
bias = np.random.rand(1)
learning_rate = 0.1

def activation(x):
    return 1 if x >= 0 else 0

epochs = 10
for epoch in range(epochs):
    for i in range(len(X)):
        linear_output = np.dot(X[i], weights) + bias
        y_pred = activation(linear_output)
        error = Y[i] - y_pred

        weights += learning_rate * error * X[i]
        bias += learning_rate * error

# Testing the Perceptron
print("Testing OR Gate Perceptron")
for i in range(len(X)):
    linear_output = np.dot(X[i], weights) + bias
    y_pred = activation(linear_output)
    print(f"Input: {X[i]}, Predicted: {y_pred}, True: {Y[i]}")

```

```
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

```python
import numpy as np

X = np.array([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
])

Y = np.array([0, 1, 1, 0])
```

```

input_size = 2
hidden_size = 2
output_size = 1
learning_rate = 0.1
epochs = 10000

weights_input_hidden = np.random.rand(input_size, hidden_size)
bias_hidden = np.random.rand(hidden_size)
weights_hidden_output = np.random.rand(hidden_size, output_size)
bias_output = np.random.rand(output_size)
# print(weights_input_hidden,bias_hidden,weights_hidden_output,bias_output)

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

for epoch in range(epochs):
    for i in range(len(X)):
        hidden_input = np.dot(X[i], weights_input_hidden) + bias_hidden
        hidden_output = sigmoid(hidden_input)

        final_input = np.dot(hidden_output, weights_hidden_output) + bias_output
        final_output = sigmoid(final_input)


        error = Y[i] - final_output

        # Backpropagation
        d_output = error * sigmoid_derivative(final_output)
        error_hidden = d_output.dot(weights_hidden_output.T)
        d_hidden = error_hidden * sigmoid_derivative(hidden_output)

        weights_hidden_output += hidden_output.reshape(-1, 1) * d_output * learning_rate
        bias_output += d_output * learning_rate
        weights_input_hidden += X[i].reshape(-1, 1) * d_hidden * learning_rate
        bias_hidden += d_hidden * learning_rate


print("Testing XOR Gate")
for i in range(len(X)):
    hidden_input = np.dot(X[i], weights_input_hidden) + bias_hidden
    hidden_output = sigmoid(hidden_input)

    final_input = np.dot(hidden_output, weights_hidden_output) + bias_output
    final_output = sigmoid(final_input)

    print(f"Input: {X[i]}, Predicted: {round(final_output[0])}, True: {Y[i]}")

```

```
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
```

```
Start coding or generate with AI.
```