Experiment No : 4

29_Dinesh Madhav_CSE(DS)

Backpropagation Algorithm

```
import numpy as np
```

```
class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size

        # Initialize weights and biases for the hidden layer and output layer
        self.W1 = np.random.randn(hidden_size, input_size)
        self.b1 = np.zeros((hidden_size, 1))
        self.W2 = np.random.randn(output_size, hidden_size)
        self.b2 = np.zeros((output_size, 1))

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

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

    def forward(self, X):
        # Forward pass
        self.z1 = np.dot(self.W1, X) + self.b1
        self.a1 = self.sigmoid(self.z1)
        self.z2 = np.dot(self.W2, self.a1) + self.b2
        self.a2 = self.sigmoid(self.z2)
        return self.a2

    def backward(self, X, y, learning_rate):
        m = X.shape[1]

        # Compute the gradients
        dZ2 = self.a2 - y
        dW2 = (1 / m) * np.dot(dZ2, self.a1.T)
        db2 = (1 / m) * np.sum(dZ2, axis=1, keepdims=True)
        dZ1 = np.dot(self.W2.T, dZ2) * self.sigmoid_derivative(self.a1)
        dW1 = (1 / m) * np.dot(dZ1, X.T)
        db1 = (1 / m) * np.sum(dZ1, axis=1, keepdims=True)

        # Update weights and biases using gradients and learning rate
        self.W2 -= learning_rate * dW2
        self.b2 -= learning_rate * db2
        self.W1 -= learning_rate * dW1
        self.b1 -= learning_rate * db1

    def train(self, X, y, epochs, learning_rate):
        for epoch in range(epochs):
            # Forward pass
            predictions = self.forward(X)

            # Compute the mean squared error loss
            loss = np.mean((predictions - y) ** 2)

            # Backward pass to update weights and biases
            self.backward(X, y, learning_rate)

            if epoch % 100 == 0:
                print(f"Epoch {epoch}, Loss: {loss:.4f}")

    def predict(self, X):
        return self.forward(X)

# Example usage:
input_size = 2
hidden_size = 4
output_size = 1

learning_rate = 0.1
epochs = 10000

# Generate some sample data
```

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

# Create the neural network
nn = NeuralNetwork(input_size, hidden_size, output_size)

# Train the neural network
nn.train(X, y, epochs, learning_rate)

# Make predictions
predictions = nn.predict(X)
print("Predictions:", predictions)
```

```
Epoch 0, Loss: 0.4124
Epoch 100, Loss: 0.2538
Epoch 200, Loss: 0.2512
Epoch 300, Loss: 0.2492
Epoch 400, Loss: 0.2475
Epoch 500, Loss: 0.2459
Epoch 600, Loss: 0.2441
Epoch 700, Loss: 0.2420
Epoch 800, Loss: 0.2395
Epoch 900, Loss: 0.2364
Epoch 1000, Loss: 0.2326
Epoch 1100, Loss: 0.2280
Epoch 1200, Loss: 0.2225
Epoch 1300, Loss: 0.2160
Epoch 1400, Loss: 0.2085
Epoch 1500, Loss: 0.2002
Epoch 1600, Loss: 0.1912
Epoch 1700, Loss: 0.1814
Epoch 1800, Loss: 0.1710
Epoch 1900, Loss: 0.1598
Epoch 2000, Loss: 0.1476
Epoch 2100, Loss: 0.1342
Epoch 2200, Loss: 0.1198
Epoch 2300, Loss: 0.1047
Epoch 2400, Loss: 0.0897
Epoch 2500, Loss: 0.0754
Epoch 2600, Loss: 0.0625
Epoch 2700, Loss: 0.0512
Epoch 2800, Loss: 0.0417
Epoch 2900, Loss: 0.0339
Epoch 3000, Loss: 0.0276
Epoch 3100, Loss: 0.0226
Epoch 3200, Loss: 0.0186
Epoch 3300, Loss: 0.0154
Epoch 3400, Loss: 0.0129
Epoch 3500, Loss: 0.0109
Epoch 3600, Loss: 0.0092
Epoch 3700, Loss: 0.0079
Epoch 3800, Loss: 0.0068
Epoch 3900, Loss: 0.0059
Epoch 4000, Loss: 0.0051
Epoch 4100, Loss: 0.0045
Epoch 4200, Loss: 0.0040
Epoch 4300, Loss: 0.0035
Epoch 4400, Loss: 0.0031
Epoch 4500, Loss: 0.0028
Epoch 4600, Loss: 0.0025
Epoch 4700, Loss: 0.0023
Epoch 4800, Loss: 0.0021
Epoch 4900, Loss: 0.0019
Epoch 5000, Loss: 0.0017
Epoch 5100, Loss: 0.0016
Epoch 5200, Loss: 0.0014
Epoch 5300, Loss: 0.0013
Epoch 5400, Loss: 0.0012
Epoch 5500, Loss: 0.0011
Epoch 5600, Loss: 0.0010
Epoch 5700, Loss: 0.0010
```