Shubham warik
Roll No:-66
CSE(DS)
Exp 3

Back Propagation in Deep Learning

In simple terms, backpropagation is a supervised learning algorithm that allows a neuralnetwork to learn from its mistakes by adjusting its weights and biases. It enables the network to iteratively improve its performance on a given task, such as classification or regression.

Code:-

```
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)
```

<u>Output:-</u>

+ Code + Text

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

```
Epoch 4600, Loss: 0.0010
Epoch 4700, Loss: 0.0009
Epoch 4800, Loss: 0.0009
Epoch 4900, Loss: 0.0008
Epoch 5000, Loss: 0.0007
Epoch 5100, Loss: 0.0007
Epoch 5200, Loss: 0.0006
Epoch 5300, Loss: 0.0006
Epoch 5400, Loss: 0.0006
Epoch 5500, Loss: 0.0005
Epoch 5600, Loss: 0.0005
Epoch 5700, Loss: 0.0005
Epoch 5800, Loss: 0.0004
Epoch 5900, Loss: 0.0004
Epoch 6000, Loss: 0.0004
Epoch 6100, Loss: 0.0004
Epoch 6200, Loss: 0.0004
Epoch 6300, Loss: 0.0003
Epoch 6400, Loss: 0.0003
Epoch 6500, Loss: 0.0003
Epoch 6600, Loss: 0.0003
Epoch 6700, Loss: 0.0003
Epoch 6800, Loss: 0.0003
Epoch 6900, Loss: 0.0003
Epoch 7000, Loss: 0.0002
Epoch 7100, Loss: 0.0002
Epoch 7200, Loss: 0.0002
Epoch 7300, Loss: 0.0002
Epoch 7400, Loss: 0.0002
Epoch 7500, Loss: 0.0002
Epoch 7600, Loss: 0.0002
Epoch 7700, Loss: 0.0002
Epoch 7800, Loss: 0.0002
Epoch 7900, Loss: 0.0002
Epoch 8000, Loss: 0.0002
Epoch 8100, Loss: 0.0002
Epoch 8200, Loss: 0.0002
Epoch 8300, Loss: 0.0001
Epoch 8400, Loss: 0.0001
Epoch 8500, Loss: 0.0001
Epoch 8600, Loss: 0.0001
Epoch 8700, Loss: 0.0001
Epoch 8800, Loss: 0.0001
Epoch 8900, Loss: 0.0001
Epoch 9000, Loss: 0.0001
Epoch 9100, Loss: 0.0001
Epoch 9200, Loss: 0.0001
Epoch 9300, Loss: 0.0001
```

+ Code + Text

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

```
Epoch 9300, Loss: 0.0001
Epoch 9400, Loss: 0.0001
Epoch 9500, Loss: 0.0001
Epoch 9600, Loss: 0.0001
Epoch 9700, Loss: 0.0001
Epoch 9800, Loss: 0.0001
Epoch 9900, Loss: 0.0001
Epoch 10000, Loss: 0.0001
Epoch 10100, Loss: 0.0001
Epoch 10200, Loss: 0.0001
Epoch 10300, Loss: 0.0001
Epoch 10400, Loss: 0.0001
Epoch 10500, Loss: 0.0001
Epoch 10600, Loss: 0.0001
Epoch 10700, Loss: 0.0001
Epoch 10800, Loss: 0.0001
Epoch 10900, Loss: 0.0001
Epoch 11000, Loss: 0.0001
Epoch 11100, Loss: 0.0001
Epoch 11200, Loss: 0.0001
Epoch 11300, Loss: 0.0001
Epoch 11400, Loss: 0.0001
Epoch 11500, Loss: 0.0001
Epoch 11600, Loss: 0.0001
Epoch 11700, Loss: 0.0001
Epoch 11800, Loss: 0.0001
Epoch 11900, Loss: 0.0001
Epoch 12000, Loss: 0.0001
Epoch 12100, Loss: 0.0001
Epoch 12200, Loss: 0.0001
Epoch 12300, Loss: 0.0001
Epoch 12400, Loss: 0.0000
Epoch 12500, Loss: 0.0000
Epoch 12600, Loss: 0.0000
Epoch 12700, Loss: 0.0000
Epoch 12800, Loss: 0.0000
Epoch 12900, Loss: 0.0000
Epoch 13000, Loss: 0.0000
Epoch 13100, Loss: 0.0000
Epoch 13200, Loss: 0.0000
Epoch 13300, Loss: 0.0000
Epoch 13400, Loss: 0.0000
Epoch 13500, Loss: 0.0000
Epoch 13600, Loss: 0.0000
Epoch 13700, Loss: 0.0000
Epoch 13800, Loss: 0.0000
Epoch 13900, Loss: 0.0000
```