

Dikshant Buwa
RollNo:04
CSE(DS)
Exp2 Deep Learning

Back Propagation in Deep Learning

In simple terms, backpropagation is a supervised learning algorithm that allows a neural network 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)
```

Output:-

```
Epoch 0, Loss: 0.2672
Epoch 100, Loss: 0.2507
Epoch 200, Loss: 0.2505
Epoch 300, Loss: 0.2503
Epoch 400, Loss: 0.2500
Epoch 500, Loss: 0.2498
Epoch 600, Loss: 0.2494
Epoch 700, Loss: 0.2490
Epoch 800, Loss: 0.2484
Epoch 900, Loss: 0.2476
Epoch 1000, Loss: 0.2464
Epoch 1100, Loss: 0.2448
Epoch 1200, Loss: 0.2425
Epoch 1300, Loss: 0.2395
Epoch 1400, Loss: 0.2357
Epoch 1500, Loss: 0.2310
Epoch 1600, Loss: 0.2255
Epoch 1700, Loss: 0.2193
Epoch 1800, Loss: 0.2126
Epoch 1900, Loss: 0.2058
Epoch 2000, Loss: 0.1989
Epoch 2100, Loss: 0.1922
Epoch 2200, Loss: 0.1859
Epoch 2300, Loss: 0.1799
Epoch 2400, Loss: 0.1743
Epoch 2500, Loss: 0.1690
Epoch 2600, Loss: 0.1638
```

Epoch 2700,	Loss: 0.1500
Epoch 2800,	Loss: 0.1532
Epoch 2900,	Loss: 0.1473
Epoch 3000,	Loss: 0.1404
Epoch 3100,	Loss: 0.1323
Epoch 3200,	Loss: 0.1225
Epoch 3300,	Loss: 0.1113
Epoch 3400,	Loss: 0.0993
Epoch 3500,	Loss: 0.0871
Epoch 3600,	Loss: 0.0754
Epoch 3700,	Loss: 0.0646
Epoch 3800,	Loss: 0.0549
Epoch 3900,	Loss: 0.0465
Epoch 4000,	Loss: 0.0392
Epoch 4100,	Loss: 0.0330
Epoch 4200,	Loss: 0.0277
Epoch 4300,	Loss: 0.0234
Epoch 4400,	Loss: 0.0198
Epoch 4500,	Loss: 0.0168
Epoch 4600,	Loss: 0.0143
Epoch 4700,	Loss: 0.0122
Epoch 4800,	Loss: 0.0105
Epoch 4900,	Loss: 0.0091
Epoch 5000,	Loss: 0.0079
Epoch 5100,	Loss: 0.0069
Epoch 5200,	Loss: 0.0061
Epoch 5300,	Loss: 0.0054
Epoch 5400,	Loss: 0.0048
Epoch 5500,	Loss: 0.0042



Epoch 5600, Loss: 0.0038
Epoch 5700, Loss: 0.0034
Epoch 5800, Loss: 0.0031
Epoch 5900, Loss: 0.0028
Epoch 6000, Loss: 0.0025
Epoch 6100, Loss: 0.0023
Epoch 6200, Loss: 0.0021
Epoch 6300, Loss: 0.0019
Epoch 6400, Loss: 0.0018
Epoch 6500, Loss: 0.0016
Epoch 6600, Loss: 0.0015
Epoch 6700, Loss: 0.0014
Epoch 6800, Loss: 0.0013
Epoch 6900, Loss: 0.0012
Epoch 7000, Loss: 0.0011
Epoch 7100, Loss: 0.0010
Epoch 7200, Loss: 0.0010
Epoch 7300, Loss: 0.0009
Epoch 7400, Loss: 0.0009
Epoch 7500, Loss: 0.0008
Epoch 7600, Loss: 0.0008
Epoch 7700, Loss: 0.0007
Epoch 7800, Loss: 0.0007
Epoch 7900, Loss: 0.0006
Epoch 8000, Loss: 0.0006
Epoch 8100, Loss: 0.0006
Epoch 8200, Loss: 0.0006
Epoch 8300, Loss: 0.0005
Epoch 8400, Loss: 0.0005
Epoch 8500, Loss: 0.0005
Epoch 8600, Loss: 0.0005
Epoch 8700, Loss: 0.0004
Epoch 8800, Loss: 0.0004
Epoch 8900, Loss: 0.0004
Epoch 9000, Loss: 0.0004
Epoch 9100, Loss: 0.0004
Epoch 9200, Loss: 0.0003
Epoch 9300, Loss: 0.0003



```
Epoch 7900, Loss: 0.0006
Epoch 8000, Loss: 0.0006
Epoch 8100, Loss: 0.0006
Epoch 8200, Loss: 0.0006
Epoch 8300, Loss: 0.0005
Epoch 8400, Loss: 0.0005
Epoch 8500, Loss: 0.0005
Epoch 8600, Loss: 0.0005
Epoch 8700, Loss: 0.0004
Epoch 8800, Loss: 0.0004
Epoch 8900, Loss: 0.0004
Epoch 9000, Loss: 0.0004
Epoch 9100, Loss: 0.0004
Epoch 9200, Loss: 0.0003
Epoch 9300, Loss: 0.0003
Epoch 9400, Loss: 0.0003
Epoch 9500, Loss: 0.0003
Epoch 9600, Loss: 0.0003
Epoch 9700, Loss: 0.0003
Epoch 9800, Loss: 0.0003
Epoch 9900, Loss: 0.0003
Predictions: [[0.01142321 0.97920427 0.99242314 0.01997907]]
```