

Name : Dipanshu Vartak

Roll no : 62

Exp : 3

CSE DS

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:-


```
+ Code + Text
```

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

```
Epoch 9200, Loss: 0.0001
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
Epoch 14000, Loss: 0.0000
```

```
Epoch 18600, Loss: 0.0000
Epoch 18700, Loss: 0.0000
Epoch 18800, Loss: 0.0000
Epoch 18900, Loss: 0.0000
Epoch 19000, Loss: 0.0000
Epoch 19100, Loss: 0.0000
Epoch 19200, Loss: 0.0000
Epoch 19300, Loss: 0.0000
Epoch 19400, Loss: 0.0000
Epoch 19500, Loss: 0.0000
Epoch 19600, Loss: 0.0000
Epoch 19700, Loss: 0.0000
Epoch 19800, Loss: 0.0000
Epoch 19900, Loss: 0.0000
Predictions: [[0.00424371 0.99688135 0.99530621 0.00330101]]
```