Lesson 9: Backpropagation and Gradient Descent

Overview: Backpropagation is a fundamental concept in training neural networks. It is an optimization algorithm used to minimize the error by updating the weights in the network. Gradient Descent is used alongside backpropagation to find the minimum of the loss function by iteratively updating the model parameters based on the gradient of the loss.

Key Concepts:

Gradient Descent: A method for optimizing the loss function by iterating and adjusting weights in the direction that reduces the error.
Learning Rate: The step size at each iteration during the optimization process.
Backpropagation: The process of computing the gradient of the loss function with respect to each weight by the chain rule and updating the weights.
Mathematical Formulation:

Forward pass: Calculate the predicted output.
Loss function: Compute the error between the predicted and true outputs.
Backward pass: Compute the gradient of the loss with respect to the weights.
Weight update: Adjust the weights using the gradient descent algorithm.
Code Example (Gradient Descent with Backpropagation):

```
python
import numpy as np

# Define sigmoid activation function and its derivative
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

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

# Training data
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  # Input
y = np.array([[0], [1], [1], [0]])  # Expected output (XOR problem)

# Initialize weights
weights = np.random.rand(2, 1)
bias = np.random.rand(1)

# Set hyperparameters
learning_rate = 0.1
epochs = 10000

# Training loop
for epoch in range(epochs):
    # Forward pass
    z = np.dot(X, weights) + bias
    output = sigmoid(z)
    # Compute the error
    error = y - output

    # Backpropagation (derivatives)
    d_output = error * sigmoid_derivative(output)

    # Update weights and bias
    weights += learning_rate * np.dot(X.T, d_output)
    bias += learning_rate * np.sum(d_output)

    if epoch % 1000 == 0:
        print(f"Epoch {epoch}, Error: {np.mean(np.abs(error))}")

print(f"Trained weights: {weights}, Trained bias: {bias}")
```