```
        delta_hidden = np.dot(delta_output,
self.weights_hidden_output.T) * sigmoid_derivative(self.hidden_output)
        d_weights_input_hidden = np.dot(X.T, delta_hidden)

        # Update weights and biases
        self.weights_hidden_output += self.learning_rate *
d_weights_hidden_output
        self.bias_output += self.learning_rate * np.sum(delta_output,
axis=0)
        self.weights_input_hidden += self.learning_rate *
d_weights_input_hidden
        self.bias_hidden += self.learning_rate * np.sum(delta_hidden,
axis=0)
     def train(self, X, y, epochs):
        for epoch in range(epochs):
            # Forward and backward pass for each data point
            for i in range(len(X)):
                input_data = X[i].reshape(1, -1)
                target_output = y[i].reshape(1, -1)
                predicted_output = self.forward(input_data)
                self.backward(input_data, target_output)

            # Calculate and print the mean squared error for this epoch
            mse = np.mean(np.square(y - self.predict(X)))
            print(f"Epoch {epoch + 1}/{epochs}, Mean Squared Error:
{mse:.4f}")
# Example usage:
if __name__ == "__main__":
    # Generate synthetic data for binary classification
    np.random.seed(0)
    X = np.random.rand(100, 2)
    y = np.where(X[:, 0] + X[:, 1] > 1, 1, 0)

    # Define and train the neural network
    input_size = 2
    hidden_size = 4
    output_size = 1
    learning_rate = 0.1
    epochs = 1000

    nn = NeuralNetwork(input_size, hidden_size, output_size,
learning_rate)
    nn.train(X, y, epochs)
    # Make predictions
    X_test = np.array([[0.7, 0.3], [0.4, 0.6]])
    predictions = nn.forward(X_test)
    print("Predicted:", predictions)
```