Vishwakarma Institute of Technology, Pune

Name: Tanishq Thuse

PRN- 12310237

Artificial Neural Networks Lab

Rollno. 52

Experiment Number: 09

Title : Write a program to implement of Ex-OR gate using feed forward neural network.

Understanding the XOR Problem and Neural Networks

The Exclusive OR (XOR) gate is a fundamental concept in digital logic. It outputs true (1) only when the inputs are different, and false (0) when they are the same. The truth table for XOR is:

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

The XOR problem is a classic challenge in the history of neural networks because it is not linearly separable. This means you cannot draw a single straight line to separate the input combinations that result in an output of 0 from those that result in an output of 1. Early single-layer perceptrons could only solve linearly separable problems, which led to a period of reduced interest in neural networks.

The ability of multi-layer neural networks (like the one implemented here with a hidden layer) to solve the XOR problem demonstrated their power and capability to learn complex, non-linear relationships. The hidden layer allows the network to create a non-linear decision boundary, effectively solving the XOR problem.

This experiment demonstrates how a simple feedforward neural network with a hidden layer can successfully learn to implement the XOR logic through the process of training with backpropagation.

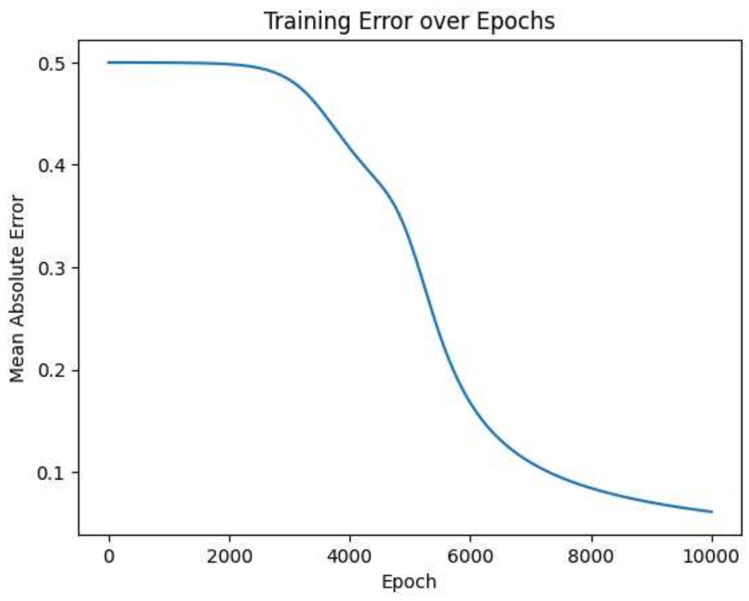
```
import numpy as np

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

# Define the derivative of the sigmoid function
def sigmoid_derivative(x):
    return x * (1 - x)

# Define the input data for the Ex-OR gate
X = np.array([[0, 0],
              [0, 1],
              [1, 0],
              [1, 1]])

# Define the output data for the Ex-OR gate
y = np.array([[0],
              [1],
              [1],
              [0]])

# Set the random seed for reproducibility
np.random.seed(1)

# Initialize the weights and biases for the neural network
input_layer_neurons = 2
hidden_layer_neurons = 2
output_layer_neurons = 1

# Weights and biases for the hidden layer
weights_hidden = np.random.uniform(size=(input_layer_neurons, hidden_layer_neurons))
bias_hidden = np.random.uniform(size=(1, hidden_layer_neurons))

# Weights and biases for the output layer
```

```
weights_output = np.random.uniform(size=(hidden_layer_neurons, output_layer_neurons))
bias_output = np.random.uniform(size=(1, output_layer_neurons))

# Set the learning rate and number of epochs
learning_rate = 0.1
epochs = 10000

# Training the neural network
for epoch in range(epochs):
    # Forward propagation
    hidden_layer_input = np.dot(X, weights_hidden) + bias_hidden
    hidden_layer_output = sigmoid(hidden_layer_input)

    output_layer_input = np.dot(hidden_layer_output, weights_output) + bias_output
    predicted_output = sigmoid(output_layer_input)

    # Backpropagation
    error = y - predicted_output
    d_predicted_output = error * sigmoid_derivative(predicted_output)

    error_hidden_layer = d_predicted_output.dot(weights_output.T)
    d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_layer_output)

    # Update weights and biases
    weights_output += hidden_layer_output.T.dot(d_predicted_output) * learning_rate
    bias_output += np.sum(d_predicted_output, axis=0, keepdims=True) * learning_rate
    weights_hidden += X.T.dot(d_hidden_layer) * learning_rate
    bias_hidden += np.sum(d_hidden_layer, axis=0, keepdims=True) * learning_rate

# Print the final predicted output after training
print("Predicted output after training:")
print(predicted_output)

# Test the trained network with input data
print("\nTesting the trained network:")
hidden_layer_input_test = np.dot(X, weights_hidden) + bias_hidden
hidden_layer_output_test = sigmoid(hidden_layer_input_test)
output_layer_input_test = np.dot(hidden_layer_output_test, weights_output) + bias_output
predicted_output_test = sigmoid(output_layer_input_test)

print("Predicted output after testing:")
print(predicted_output_test)
```

```
Predicted output after training:
[[0.06368082]
 [0.94085536]
 [0.94108726]
 [0.06402009]]

Testing the trained network:
Predicted output after testing:
[[0.06367371]
 [0.94086271]
 [0.94109457]
 [0.06401166]]
```

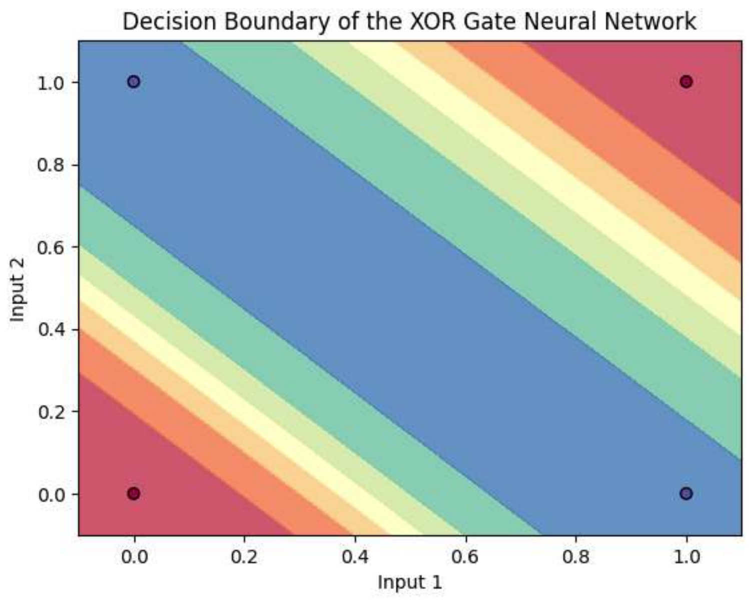
```
import matplotlib.pyplot as plt

# Store the error at each epoch
error_history = []

# Re-initialize weights and biases for a fresh start for visualization
np.random.seed(1)
weights_hidden = np.random.uniform(size=(input_layer_neurons, hidden_layer_neurons))
bias_hidden = np.random.uniform(size=(1, hidden_layer_neurons))
weights_output = np.random.uniform(size=(hidden_layer_neurons, output_layer_neurons))
bias_output = np.random.uniform(size=(1, output_layer_neurons))

# Re-train the network and store error
for epoch in range(epochs):
    # Forward propagation
    hidden_layer_input = np.dot(X, weights_hidden) + bias_hidden
    hidden_layer_output = sigmoid(hidden_layer_input)
    output_layer_input = np.dot(hidden_layer_output, weights_output) + bias_output
    predicted_output = sigmoid(output_layer_input)

    # Backpropagation
```

```
        error = y - predicted_output
        error_history.append(np.mean(np.abs(error))) # Store mean absolute error
        d_predicted_output = error * sigmoid_derivative(predicted_output)
        error_hidden_layer = d_predicted_output.dot(weights_output.T)
        d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_layer_output)

        # Update weights and biases
        weights_output += hidden_layer_output.T.dot(d_predicted_output) * learning_rate
        bias_output += np.sum(d_predicted_output, axis=0, keepdims=True) * learning_rate
        weights_hidden += X.T.dot(d_hidden_layer) * learning_rate
        bias_hidden += np.sum(d_hidden_layer, axis=0, keepdims=True) * learning_rate

    # Plot the error history
    plt.plot(error_history)
    plt.xlabel("Epoch")
    plt.ylabel("Mean Absolute Error")
    plt.title("Training Error over Epochs")
    plt.show()

    print("Predicted output after training (with error visualization):")
    print(predicted_output)

    print("\nTesting the trained network (with error visualization):")
    hidden_layer_input_test = np.dot(X, weights_hidden) + bias_hidden
    hidden_layer_output_test = sigmoid(hidden_layer_input_test)
    output_layer_input_test = np.dot(hidden_layer_output_test, weights_output) + bias_output
    predicted_output_test = sigmoid(output_layer_input_test)

    print("Predicted output after testing (with error visualization):")
    print(predicted_output_test)
```



```
Predicted output after training (with error visualization):
[[0.06368082]
 [0.94085536]
 [0.94108726]
 [0.06402009]]

Testing the trained network (with error visualization):
Predicted output after testing (with error visualization):
[[0.06367371]
 [0.94086271]
 [0.94109457]
 [0.06401166]]
```

```
    # Visualize the decision boundary
    def plot_decision_boundary(X, y, weights_hidden, bias_hidden, weights_output, bias_output):
        # Set the bounds for the plot
        x_min, x_max = X[:, 0].min() - 0.1, X[:, 0].max() + 0.1
        y_min, y_max = X[:, 1].min() - 0.1, X[:, 1].max() + 0.1
        h = 0.01  # Step size in the mesh

        # Create a meshgrid
        xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
        # Predict the output for each point in the meshgrid
        Z = sigmoid(np.dot(sigmoid(np.dot(np.c_[xx.ravel(), yy.ravel()], weights_hidden) + bias_hidden), weights_output) + bias_outp
        Z = Z.reshape(xx.shape)

        # Plot the contour and the data points
        plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral, alpha=0.8)
        plt.scatter(X[:, 0], X[:, 1], c=y.flatten(), cmap=plt.cm.Spectral, edgecolors='k')
        plt.title("Decision Boundary of the XOR Gate Neural Network")
        plt.xlabel("Input 1")
        plt.ylabel("Input 2")
        plt.show()

    # Plot the decision boundary using the trained weights and biases
    plot_decision_boundary(X, y, weights_hidden, bias_hidden, weights_output, bias_output)
```


Decision Boundary of the XOR Gate Neural Network
Input 1
Input 2