Vishwakarma Institute of Technology, Pune

Name: Tanishq Thuse

PRN- 12310237

Artificial Neural Networks Lab

Rollno. 52

Experiment Number: 08

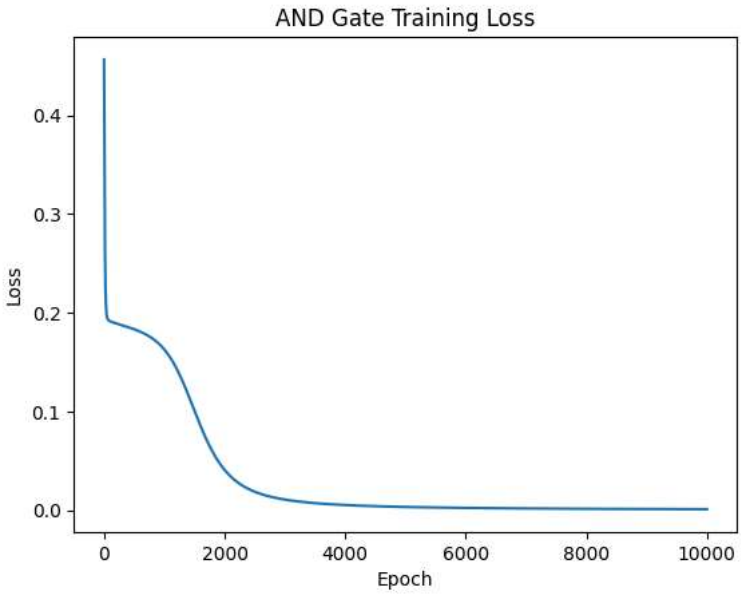
Title : Write a program to implement of AND/NAND gate using feed forward neural network.

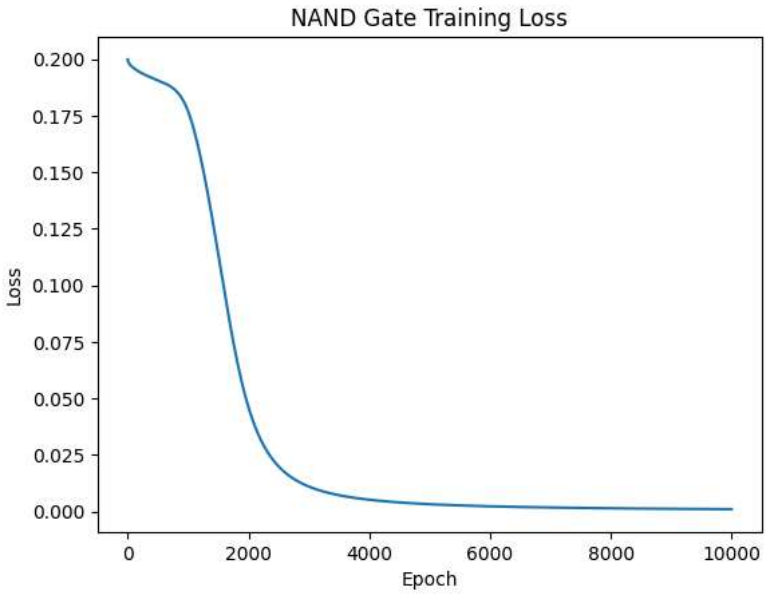
```
import numpy as np
import matplotlib.pyplot as plt

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
        self.weights_input_hidden = np.random.rand(self.input_size, self.hidden_size)
        self.bias_hidden = np.random.rand(1, self.hidden_size)
        self.weights_hidden_output = np.random.rand(self.hidden_size, self.output_size)
        self.bias_output = np.random.rand(1, self.output_size)
        self.loss_history = [] # To store loss for visualization

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

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

    def forward(self, X):
        # Input to hidden layer
        self.hidden_output = self.sigmoid(np.dot(X, self.weights_input_hidden) + self.bias_hidden)

        # Hidden to output layer
        self.predicted_output = self.sigmoid(np.dot(self.hidden_output, self.weights_hidden_output) + self.bias_output)

        return self.predicted_output

    def backward(self, X, y, learning_rate):
        # Calculate error
        error = y - self.predicted_output

        # Backpropagate error to output layer
        d_output = error * self.sigmoid_derivative(self.predicted_output)

        # Backpropagate error to hidden layer
        error_hidden = d_output.dot(self.weights_hidden_output.T)
        d_hidden = error_hidden * self.sigmoid_derivative(self.hidden_output)

        # Update weights and biases
        self.weights_hidden_output += self.hidden_output.T.dot(d_output) * learning_rate
        self.bias_output += np.sum(d_output, axis=0, keepdims=True) * learning_rate
        self.weights_input_hidden += X.T.dot(d_hidden) * learning_rate
        self.bias_hidden += np.sum(d_hidden, axis=0, keepdims=True) * learning_rate

    def train(self, X, y, epochs, learning_rate):
        self.loss_history = [] # Reset loss history for new training run
        for epoch in range(epochs):
            self.forward(X)
            self.backward(X, y, learning_rate)
            loss = np.mean(np.square(y - self.predicted_output))
            self.loss_history.append(loss)


            if epoch % 1000 == 0:
                print(f"Epoch {epoch}, Loss: {loss}")

# --- AND Gate Implementation ---
```

```
print("--- AND Gate ---")
# Training data for AND gate
X_and = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_and = np.array([[0], [0], [0], [1]])

# Create and train the neural network for AND gate
nn_and = NeuralNetwork(input_size=2, hidden_size=2, output_size=1)
nn_and.train(X_and, y_and, epochs=10000, learning_rate=0.1)

# Test the trained network for AND gate
print("Predictions for AND gate:")
for input_data in X_and:
    prediction = nn_and.forward(input_data)
    print(f"Input: {input_data}, Prediction: {prediction[0][0]:.4f}, Rounded: {round(prediction[0][0])}")

# --- NAND Gate Implementation ---
print("\n--- NAND Gate ---")
# Training data for NAND gate
X_nand = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_nand = np.array([[1], [1], [1], [0]])

# Create and train the neural network for NAND gate
nn_nand = NeuralNetwork(input_size=2, hidden_size=2, output_size=1)
nn_nand.train(X_nand, y_nand, epochs=10000, learning_rate=0.1)

# Test the trained network for NAND gate
print("Predictions for NAND gate:")
for input_data in X_nand:
    prediction = nn_nand.forward(input_data)
    print(f"Input: {input_data}, Prediction: {prediction[0][0]:.4f}, Rounded: {round(prediction[0][0])}")
```

```
--- AND Gate ---
Epoch 0, Loss: 0.4564834551744958
Epoch 1000, Loss: 0.1627804253099384
Epoch 2000, Loss: 0.0407939297855478
Epoch 3000, Loss: 0.010764347677258883
Epoch 4000, Loss: 0.005258302548563609
Epoch 5000, Loss: 0.0033115488701882935
Epoch 6000, Loss: 0.0023654656601612613
Epoch 7000, Loss: 0.001818811898044158
Epoch 8000, Loss: 0.0014671129591636559
Epoch 9000, Loss: 0.0012237309680312443
Predictions for AND gate:
Input: [0 0], Prediction: 0.0059, Rounded: 0
Input: [0 1], Prediction: 0.0350, Rounded: 0
Input: [1 0], Prediction: 0.0343, Rounded: 0
Input: [1 1], Prediction: 0.9581, Rounded: 1

--- NAND Gate ---
Epoch 0, Loss: 0.19986882126816374
Epoch 1000, Loss: 0.17701834141835715
Epoch 2000, Loss: 0.04614318519424584
Epoch 3000, Loss: 0.010983970098794345
Epoch 4000, Loss: 0.005153480421333889
Epoch 5000, Loss: 0.003183577367190878
Epoch 6000, Loss: 0.002247545793971455
Epoch 7000, Loss: 0.0017140045952554316
Epoch 8000, Loss: 0.001373924319924071
Epoch 9000, Loss: 0.0011402071180374308
Predictions for NAND gate:
Input: [0 0], Prediction: 0.9973, Rounded: 1
Input: [0 1], Prediction: 0.9706, Rounded: 1
Input: [1 0], Prediction: 0.9702, Rounded: 1
Input: [1 1], Prediction: 0.0460, Rounded: 0
```

```
# Visualize the loss for AND gate
plt.plot(nn_and.loss_history)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('AND Gate Training Loss')
plt.show()

# Visualize the loss for NAND gate
plt.plot(nn_nand.loss_history)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('NAND Gate Training Loss')
plt.show()
```


AND Gate Training Loss
Loss
Epoch
0.0
0.1
0.2
0.3
0.4
0
2000
4000
6000
8000
10000


NAND Gate Training Loss
Loss
Epoch
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
0
2000
4000
6000
8000
10000