lab2

February 3, 2024

[1]:

```
# 2. Write a Program to implement XOR with backpropagation algorithm.

import numpy as np

class SimplePerceptron:
    def __init__(self, input_size, hidden_size, output_size):
    # Initialize weights and biases with random values
        self.weights_input_hidden = np.random.rand(input_size, hidden_size)
        self.bias_hidden = np.zeros((1, hidden_size))
        self.weights_hidden_output = np.random.rand(hidden_size, output_size)
        self.bias_output = np.zeros((1, output_size))

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

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

    def forward(self, inputs):
        # Forward pass through the network
        self.hidden_layer_activation = self.sigmoid(np.dot(inputs, self.
↪weights_input_hidden
    ) + self.bias_hidden)
        self.output_layer_activation = self.sigmoid(np.dot(self.
↪hidden_layer_activation,
    self.weights_hidden_output) + self.bias_output)

        return self.output_layer_activation

    def backward(self, inputs, targets, learning_rate):
        # Backward pass through the network

        # Calculate output layer error and delta
        output_layer_error = targets - self.output_layer_activation
        output_layer_delta = output_layer_error * self.sigmoid_derivative(self.
    output_layer_activation)
```

```
        # Calculate hidden layer error and delta
        hidden_layer_error = output_layer_delta.dot(self.weights_hidden_output.T)
        hidden_layer_delta = hidden_layer_error * self.sigmoid_derivative(self.hidden_layer_activation)

        # Update weights and biases
        self.weights_hidden_output += self.hidden_layer_activation.T.dot(output_layer_delta) * learning_rate
        self.bias_output += np.sum(output_layer_delta, axis=0, keepdims=True) * learning_rate

        self.weights_input_hidden += inputs.T.dot(hidden_layer_delta) * learning_rate
        self.bias_hidden += np.sum(hidden_layer_delta, axis=0, keepdims=True) * learning_rate

    def train(self, inputs, targets, epochs, learning_rate):
        for epoch in range(epochs):
            # Forward and backward pass for each training example
            for i in range(len(inputs)):
                input_data = np.array([inputs[i]])
                target_data = np.array([targets[i]])

            # Forward pass
            output = self.forward(input_data)

            # Backward pass
            self.backward(input_data, target_data, learning_rate)

        # Print the mean squared error for every 100 epochs
            if epoch % 100 == 0:
                mse = np.mean(np.square(targets - self.forward(inputs)))
                print(f"Epoch {epoch}, Mean Squared Error: {mse}")

# Example usage
if __name__ == "__main__":
    # Define the dataset (XOR problem)
    inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
    targets = np.array([[0], [1], [1], [0]])

    # Create a simple perceptron model
    model = SimplePerceptron(input_size=2, hidden_size=4, output_size=1)

    # Train the model
    model.train(inputs, targets, epochs=1000, learning_rate=0.1)
```

```
# Test the trained model
test_inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
predictions = model.forward(test_inputs)
print("\nPredictions:")
print(predictions)
```

```
Epoch 0, Mean Squared Error: 0.345016241179297
Epoch 100, Mean Squared Error: 0.31887436505541494
Epoch 200, Mean Squared Error: 0.37660234789907376
Epoch 300, Mean Squared Error: 0.40083811108114387
Epoch 400, Mean Squared Error: 0.41471959757155136
Epoch 500, Mean Squared Error: 0.4239653668774533
Epoch 600, Mean Squared Error: 0.4306831416578448
Epoch 700, Mean Squared Error: 0.4358475093690745
Epoch 800, Mean Squared Error: 0.43997782890683357
Epoch 900, Mean Squared Error: 0.443379164581669

Predictions:
[[0.07221412]
 [0.06200459]
 [0.05255291]
 [0.04651396]]
```

[ ]:

```

```