Deep Learning

Experiment 04

BACK PROPAGATION

Backpropagation is an essential algorithm for training Deep Neural Networks (DNNs) with multiple hidden layers. The algorithm involves iteratively adjusting the network's weights and biases to minimize the difference between the predicted output and the actual target output.
DNN with 2 hidden layers and a sigmoid activation function.
The architecture will be: Input Layer -> Hidden Layer 1 -> Hidden Layer 2 -> Output Layer.

Code:

```
import numpy as np

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

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

# Create the dataset (dummy data)
np.random.seed(0)
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])

# Initialize the neural network architecture
input_size = 2
hidden_size1 = 4
hidden_size2 = 3
output_size = 1
learning_rate = 0.1
epochs = 10000

# Initialize weights and biases with random values
weights_input_hidden1 = np.random.uniform(size=(input_size, hidden_size1))
bias_hidden1 = np.zeros((1, hidden_size1))
weights_hidden1_hidden2        =       np.random.uniform(size=(hidden_size1,
hidden_size2))
```

```
bias_hidden2 = np.zeros((1, hidden_size2))
weights_hidden2_output = np.random.uniform(size=(hidden_size2,
output_size))
bias_output = np.zeros((1, output_size))

# Training loop
for epoch in range(epochs):
    # Forward propagation
    hidden1_input = np.dot(X, weights_input_hidden1) + bias_hidden1
    hidden1_output = sigmoid(hidden1_input)
    hidden2_input = np.dot(hidden1_output, weights_hidden1_hidden2) +
bias_hidden2
    hidden2_output = sigmoid(hidden2_input)
    output_input = np.dot(hidden2_output, weights_hidden2_output) +
bias_output
    predicted_output = sigmoid(output_input)

    # Calculate the loss
    loss = np.mean(0.5 * (y - predicted_output) ** 2)

    # Backpropagation
    output_error = y - predicted_output
    output_delta = output_error * sigmoid_derivative(predicted_output)

    hidden2_error = output_delta.dot(weights_hidden2_output.T)
    hidden2_delta = hidden2_error * sigmoid_derivative(hidden2_output)

    hidden1_error = hidden2_delta.dot(weights_hidden1_hidden2.T)
    hidden1_delta = hidden1_error * sigmoid_derivative(hidden1_output)

    # Update weights and biases
    weights_hidden2_output += hidden2_output.T.dot(output_delta) *
learning_rate
    bias_output += np.sum(output_delta, axis=0, keepdims=True) *
learning_rate
    weights_hidden1_hidden2 += hidden1_output.T.dot(hidden2_delta) *
learning_rate
    bias_hidden2 += np.sum(hidden2_delta, axis=0, keepdims=True) *
learning_rate
    weights_input_hidden1 += X.T.dot(hidden1_delta) * learning_rate
```

```
        bias_hidden1  +=  np.sum(hidden1_delta,  axis=0,  keepdims=True)  *
learning_rate

    if epoch % 1000 == 0:
        print(f"Epoch {epoch}, Loss: {loss:.4f}")

print("Training complete!")
print("Final predicted output:")
print(predicted_output)
```

Output:

```
Epoch 0, Loss: 0.1884
Epoch 1000, Loss: 0.1243
Epoch 2000, Loss: 0.1235
Epoch 3000, Loss: 0.1215
Epoch 4000, Loss: 0.1151
Epoch 5000, Loss: 0.1007
Epoch 6000, Loss: 0.0914
Epoch 7000, Loss: 0.0879
Epoch 8000, Loss: 0.0863
Epoch 9000, Loss: 0.0855
Training complete!
Final predicted output:
[[0.06822035]
 [0.66145138]
 [0.6611    ]
 [0.66757633]]
```