Name:Shreya Kakade

Roll No:TEAD22541

Div:A

Practical No :6

1]:
```python
import numpy as np

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

# Initialize dataset (XOR problem)
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  # Input
y = np.array([[0], [1], [1], [0]])  # Expected Output

# Set network parameters
input_neurons = 2
hidden_neurons = 4
output_neurons = 1

# Initialize weights and biases
np.random.seed(42)
weights_input_hidden = np.random.uniform(-1, 1, (input_neurons, hidden_neurons))
weights_hidden_output = np.random.uniform(-1, 1, (hidden_neurons, output_neurons))
bias_hidden = np.random.uniform(-1, 1, (1, hidden_neurons))
bias_output = np.random.uniform(-1, 1, (1, output_neurons))

# Forward Propagation
hidden_layer_input = np.dot(X, weights_input_hidden) + bias_hidden
hidden_layer_output = sigmoid(hidden_layer_input)
```

```python
bias_hidden = np.random.uniform(-1, 1, (1, hidden_neurons))
bias_output = np.random.uniform(-1, 1, (1, output_neurons))

# Forward Propagation
hidden_layer_input = np.dot(X, weights_input_hidden) + bias_hidden
hidden_layer_output = sigmoid(hidden_layer_input)

output_layer_input = np.dot(hidden_layer_output, weights_hidden_output) + bias_output
output = sigmoid(output_layer_input)

print("Output after Forward Propagation:")
print(output)
```

```
Output after Forward Propagation:
[[0.47151531]
 [0.53028619]
 [0.47572561]
 [0.53949477]]
```

[ ]:

[1]:

```python
import numpy as np

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

# Sigmoid derivative function for backpropagation
def sigmoid_derivative(x):
    return x * (1 - x)

# XOR Input and Output
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  # Input data
y = np.array([[0], [1], [1], [0]])  # Expected XOR output

# Neural Network structure
input_neurons = 2
hidden_neurons = 4
output_neurons = 1

# Initialize weights and biases
np.random.seed(42)  # For reproducibility
weights_input_hidden = np.random.uniform(-1, 1, (input_neurons, hidden_neurons))
weights_hidden_output = np.random.uniform(-1, 1, (hidden_neurons, output_neurons))

bias_hidden = np.random.uniform(-1, 1, (1, hidden_neurons))
bias_output = np.random.uniform(-1, 1, (1, output_neurons))
```

```python
epochs = 10000

# Training process
for epoch in range(epochs):
    # Forward Propagation
    hidden_layer_input = np.dot(X, weights_input_hidden) + bias_hidden
    hidden_layer_output = sigmoid(hidden_layer_input)

    output_layer_input = np.dot(hidden_layer_output, weights_hidden_output) + bias_output
    output = sigmoid(output_layer_input)

    # Compute error
    error = y - output

    # Backward Propagation
    output_gradient = error * sigmoid_derivative(output)
    hidden_gradient = output_gradient.dot(weights_hidden_output.T) * sigmoid_derivative(hidden_layer_output)

    # Update weights and biases
    weights_hidden_output += hidden_layer_output.T.dot(output_gradient) * learning_rate
    weights_input_hidden += X.T.dot(hidden_gradient) * learning_rate

    bias_output += np.sum(output_gradient, axis=0, keepdims=True) * learning_rate
    bias_hidden += np.sum(hidden_gradient, axis=0, keepdims=True) * learning_rate

    # Print loss every 1000 epochs
    if epoch % 1000 == 0:
        loss = np.mean(np.abs(error))
        print(f"Epoch {epoch}, Loss: {loss}")
```

```python
        loss = np.mean(np.abs(error))
        print(f"Epoch {epoch}, Loss: {loss}")

# Final Output after Training
print("\nFinal Output after Training:")
print(output)
```

```
Epoch 0, Loss: 0.5012495691702666
Epoch 1000, Loss: 0.07892380962319563
Epoch 2000, Loss: 0.04200729440540749
Epoch 3000, Loss: 0.03137807616202634
Epoch 4000, Loss: 0.025949337857365067
Epoch 5000, Loss: 0.022544096456915925
Epoch 6000, Loss: 0.020163626571278296
Epoch 7000, Loss: 0.01838364481836235
Epoch 8000, Loss: 0.0169900605067304
Epoch 9000, Loss: 0.01586192435894144

Final Output after Training:
[[0.01572886]
 [0.98627449]
 [0.98425486]
 [0.01450469]]
```

]: