Practical 07

Aim- To implement a Simple Neural Network using backpropogation.

Theory-

Artificial Neural Networks (ANNs) are machine learning models inspired by the structure and functioning of the human brain. They are powerful tools for solving tasks such as classification, regression, and pattern recognition, as they can approximate highly complex relationships. An ANN is organized into layers of interconnected processing units called neurons. The first layer takes the input data, the final layer produces the output, and the intermediate hidden layers transform the information.

During computation, data moves forward through the network in what is called **forward propagation**. Each neuron receives inputs, multiplies them by connection weights, sums them, and applies an activation function (such as ReLU, sigmoid, or tanh) before passing the result to the next layer.

The learning process in ANNs revolves around adjusting these weights so that the model's predictions become increasingly accurate. This is achieved using the **backpropagation algorithm**, which relies on the chain rule of calculus to compute how the loss function changes with respect to each weight. The general training workflow is:

1. **Forward Propagation** – Inputs flow through the network to generate predictions.
2. **Loss Evaluation** – A loss function (e.g., Mean Squared Error or Cross-Entropy) measures the gap between predicted and actual outputs.
3. **Backpropagation** – Errors are propagated backward, and gradients are calculated for each weight.

The gradients guide weight updates in the direction that reduces the error. Repeating this process across many training iterations allows the network to refine its parameters and improve performance, even in deep multi-layered architectures.

Neural networks excel at modeling nonlinear patterns and extracting meaningful representations from data, but they also come with challenges such as high computational requirements, sensitivity to hyperparameter settings, and the need for large amounts of training data.

Despite these challenges, ANNs are widely applied today—in image classification, speech and text processing, fraud detection, and even blockchain security. For instance, in blockchain systems, a neural network can be trained to spot fraudulent transactions by recognizing subtle irregularities in transaction patterns.

Code-

```
import numpy as np

# Activation function (Sigmoid) and its derivative
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

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

# Training dataset (XOR problem for simplicity)
X = np.array([[0,0],
              [0,1],
              [1,0],
              [1,1]])

# Expected output
y = np.array([[0],
              [1],
              [1],
              [0]])

# Seed for reproducibility
np.random.seed(42)

# Initialize weights and biases
input_neurons = 2
hidden_neurons = 2
output_neurons = 1

W1 = np.random.uniform(size=(input_neurons, hidden_neurons))
b1 = np.random.uniform(size=(1, hidden_neurons))
W2 = np.random.uniform(size=(hidden_neurons, output_neurons))
b2 = np.random.uniform(size=(1, output_neurons))

# Training parameters
epochs = 10000
lr = 0.1   # learning rate

# Training loop
for epoch in range(epochs):
    # ---- Forward Propagation ----
    z1 = np.dot(X, W1) + b1
    a1 = sigmoid(z1)

    z2 = np.dot(a1, W2) + b2
    a2 = sigmoid(z2)
```

```
    # ---- Loss (Mean Squared Error) ----
    error = y - a2

    # ---- Backpropagation ----
    d_a2 = error * sigmoid_derivative(a2)
    d_a1 = d_a2.dot(W2.T) * sigmoid_derivative(a1)

    # ---- Update Weights & Biases ----
    W2 += a1.T.dot(d_a2) * lr
    b2 += np.sum(d_a2, axis=0, keepdims=True) * lr
    W1 += X.T.dot(d_a1) * lr
    b1 += np.sum(d_a1, axis=0, keepdims=True) * lr

# Final predictions
print("Final outputs after training:")
print(a2)
```

Output-

```
Final outputs after training:
[[0.05322146]
 [0.95171535]
 [0.95160449]
 [0.05175396]]
```

Conclusion- In this practical, we implemented a simple neural network and trained it using the backpropagation algorithm. The experiment demonstrated how forward propagation computes predictions, backpropagation adjusts weights using gradient descent, and the network gradually learns to minimize error. This provided a foundational understanding of how neural networks learn complex relationships, forming the basis for advanced deep learning applications in domains like image recognition, natural language processing, and blockchain fraud detection.