PRACTICAL – 5

AIM: Aim: Write a python program to create a simple 3-layer neural network for implementation of binary function.

Code:

```
import numpy as np

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

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

class NeuralNetwork:
    def __init__(self):
        np.random.seed(0)

        self.input_size = 2
        self.hidden_size = 3
        self.output_size = 1

        self.weights_input_hidden = np.random.uniform(-1, 1, (self.input_size, self.hidden_size))
        self.bias_hidden = np.random.uniform(-1, 1, (1, self.hidden_size))

        self.weights_hidden_output = np.random.uniform(-1, 1, (self.hidden_size, self.output_size))
        self.bias_output = np.random.uniform(-1, 1, (1, self.output_size))

    def forward(self, X):
        self.hidden_input = np.dot(X, self.weights_input_hidden) + self.bias_hidden
        self.hidden_output = sigmoid(self.hidden_input)

        self.final_input = np.dot(self.hidden_output, self.weights_hidden_output) + self.bias_output
        self.final_output = sigmoid(self.final_input)

        return self.final_output

    def backward(self, X, y, learning_rate):
        output_error = y - self.final_output  # Loss function: Mean Squared Error (MSE)
        output_delta = output_error * sigmoid_derivative(self.final_output)

        hidden_error = np.dot(output_delta, self.weights_hidden_output.T)
        hidden_delta = hidden_error * sigmoid_derivative(self.hidden_output)
```

```
        self.weights_hidden_output += np.dot(self.hidden_output.T, output_delta) * learning_rate
        self.bias_output += np.sum(output_delta, axis=0, keepdims=True) * learning_rate

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

    def train(self, X, y, epochs=10000, learning_rate=0.1):
        for epoch in range(epochs):
            self.forward(X)
            self.backward(X, y, learning_rate)

    def print_parameters(self):
        print("Weights from Input to Hidden Layer:\n", self.weights_input_hidden)
        print("Bias of Hidden Layer:\n", self.bias_hidden)
        print("Weights from Hidden to Output Layer:\n", self.weights_hidden_output)
        print("Bias of Output Layer:\n", self.bias_output)

    def predict(self, X):
        return self.forward(X)

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])
nn = NeuralNetwork()
nn.train(X, y)
nn.print_parameters()
print("\nPredictions after training:")
for i in range(len(X)):
    print(f"Input: {X[i]} -> Output: {nn.predict(X[i].reshape(1, -1))[0][0]:.4f}")
```

Output:

```
Weights from Input to Hidden Layer:
 [[3.50680746 5.48851224 0.19968016]
 [3.26509818 5.58862662 0.94635634]]
Bias of Hidden Layer:
 [[-5.16519319 -2.18017667  0.29848228]]
Weights from Hidden to Output Layer:
 [[-7.1798058 ]
 [ 7.33883282]
 [-2.28684986]]
Bias of Output Layer:
 [[-1.9243128]]

Predictions after training:
Input: [0 0] -> Output: 0.0736
Input: [0 1] -> Output: 0.9220
Input: [1 0] -> Output: 0.9299
Input: [1 1] -> Output: 0.0819
```