PRACTICAL 4

A. Aim: Write a python Program for back propagation algorithm.

```
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, input_size, hidden_size, output_size):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.learning_rate = 0.1
        self.weights_input_hidden = np.random.rand(self.input_size, self.hidden_size)
        self.weights_hidden_output = np.random.rand(self.hidden_size, self.output_size)
    def feedforward(self, X):
        self.hidden_layer_input = np.dot(X, self.weights_input_hidden)
        self.hidden_layer_output = sigmoid(self.hidden_layer_input)
        self.output_layer_input = np.dot(self.hidden_layer_output, self.weights_hidden_output)
        self.output_layer_output = sigmoid(self.output_layer_input)
    def backward(self, X, y):
        self.error = y - self.output_layer_output
        delta_output = self.error * sigmoid_derivative(self.output_layer_output)
        self.hidden_layer_error = delta_output.dot(self.weights_hidden_output.T)
        delta_hidden = self.hidden_layer_error * sigmoid_derivative(self.hidden_layer_output)
        self.weights_hidden_output += self.hidden_layer_output.T.dot(delta_output) *
self.learning_rate
        self.weights_input_hidden += X.T.dot(delta_hidden) * self.learning_rate
    def train(self, X, y, epochs):
        for _ in range(epochs):
            self.feedforward(X)
            self.backward(X, y)
    def predict(self, X):
        self.feedforward(X)
        return self.output_layer_output
if __name__ == "__main__":
    X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
    y = np.array([[0], [1], [1], [0]])
    neural_network = NeuralNetwork(2, 4, 1)
    neural_network.train(X, y, epochs=10000)
    predictions = neural_network.predict(X)
    print("Predictions:")
    print(predictions)
```

Output:

```
Predictions:
 [[0.14619378]
 [0.85005797]
 [0.83965413]
 [0.16800782]]
```

PRACTICAL 4

B. Aim: Write a python Program for error back propagation algorithm.

```
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, input_size, hidden_size, output_size, learning_rate):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.learning_rate = learning_rate
        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 forward(self, x):
        self.hidden_input = np.dot(x, self.weights_input_hidden) + self.bias_hidden
        self.hidden_output = sigmoid(self.hidden_input)
        self.output_input = np.dot(self.hidden_output, self.weights_hidden_output) +
self.bias_output
        self.output = sigmoid(self.output_input)
    def backward(self, x, y):
        loss = y - self.output
        delta_output = loss * sigmoid_derivative(self.output)
        hidden_error = delta_output.dot(self.weights_hidden_output.T)
        delta_hidden = hidden_error * sigmoid_derivative(self.hidden_output)
        self.weights_hidden_output += self.hidden_output.T.dot(delta_output) *
self.learning_rate
        self.bias_output += np.sum(delta_output, axis=0, keepdims=True) * self.learning_rate
        self.weights_input_hidden += x.T.dot(delta_hidden) * self.learning_rate
        self.bias_hidden += np.sum(delta_hidden, axis=0, keepdims=True) * self.learning_rate
    def train(self, x, y, epochs):
        for _ in range(epochs):
            self.forward(x)
            self.backward(x, y)
    def predict(self, x):
        self.forward(x)
        return self.output
if __name__ == "__main__":
    X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
    y = np.array([[0], [1], [1], [0]])
    neural_network = NeuralNetwork(input_size=2,hidden_size=4,
      output_size=1, learning_rate=0.1)
    neural_network.train(X, y, epochs=10000)
    predictions = neural_network.predict(X)
    print("Predictions:")
    print(predictions)
```

Output:

```
Predictions:
 [[0.05445362]
 [0.95325663]
 [0.95323372]
 [0.04766259]]
```