Soham Dey
Roll no - **10**
CSE(DS)

Apply learning algorithm to learn the parameters of the supervised single layer feed forward neural network using **Stochastic Gradient Descent .**

Code -

```
import numpy as np

class SingleLayerNeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size

        # Initialize weights and biases
        self.weights_hidden = np.random.rand(self.input_size, self.hidden_size)
        self.bias_hidden = np.zeros((1, self.hidden_size))

        self.weights_output = np.random.rand(self.hidden_size, self.output_size)
        self.bias_output = np.zeros((1, self.output_size))

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

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

    def forward(self, x):
        self.hidden_activation = self.sigmoid(np.dot(x, self.weights_hidden) + self.bias_hidden)
        self.output = self.sigmoid(np.dot(self.hidden_activation, self.weights_output) +
self.bias_output)
        return self.output

    def backward(self, x, y, learning_rate):
        output_error = y - self.output
        output_delta = output_error * self.sigmoid_derivative(self.output)

        hidden_error = output_delta.dot(self.weights_output.T)
        hidden_delta = hidden_error * self.sigmoid_derivative(self.hidden_activation)

        self.weights_output += self.hidden_activation.T.dot(output_delta) * learning_rate
```

```
        self.bias_output += np.sum(output_delta) * learning_rate

        self.weights_hidden += x.reshape(-1, 1).dot(hidden_delta.reshape(1, -1)) * learning_rate
        self.bias_hidden += np.sum(hidden_delta) * learning_rate

    def train(self, X, y, epochs, learning_rate):
        for epoch in range(epochs):
            for i in range(len(X)):
                x = X[i]
                target = y[i]

                self.forward(x)
                self.backward(x, target, learning_rate)

                if (i+1) % 100 == 0:
                    loss = np.mean(np.square(target - self.output))
                    print(f'Epoch {epoch+1}, Sample {i+1}, Loss: {loss:.4f}')

# Example usage
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])

input_size = 2
hidden_size = 4
output_size = 1

nn = SingleLayerNeuralNetwork(input_size, hidden_size, output_size)
nn.train(X, y, epochs=10000, learning_rate=0.1)

# Test the trained network
test_input = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
for x in test_input:
    prediction = nn.forward(x)
    print(f'Input: {x}, Prediction: {prediction}')
```

Output -

```
[4]

      Input: [0 0], Prediction: [[0.50004413]]
      Input: [0 1], Prediction: [[0.50006781]]
      Input: [1 0], Prediction: [[0.50089602]]
      Input: [1 1], Prediction: [[0.49959904]]
```