**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 -

