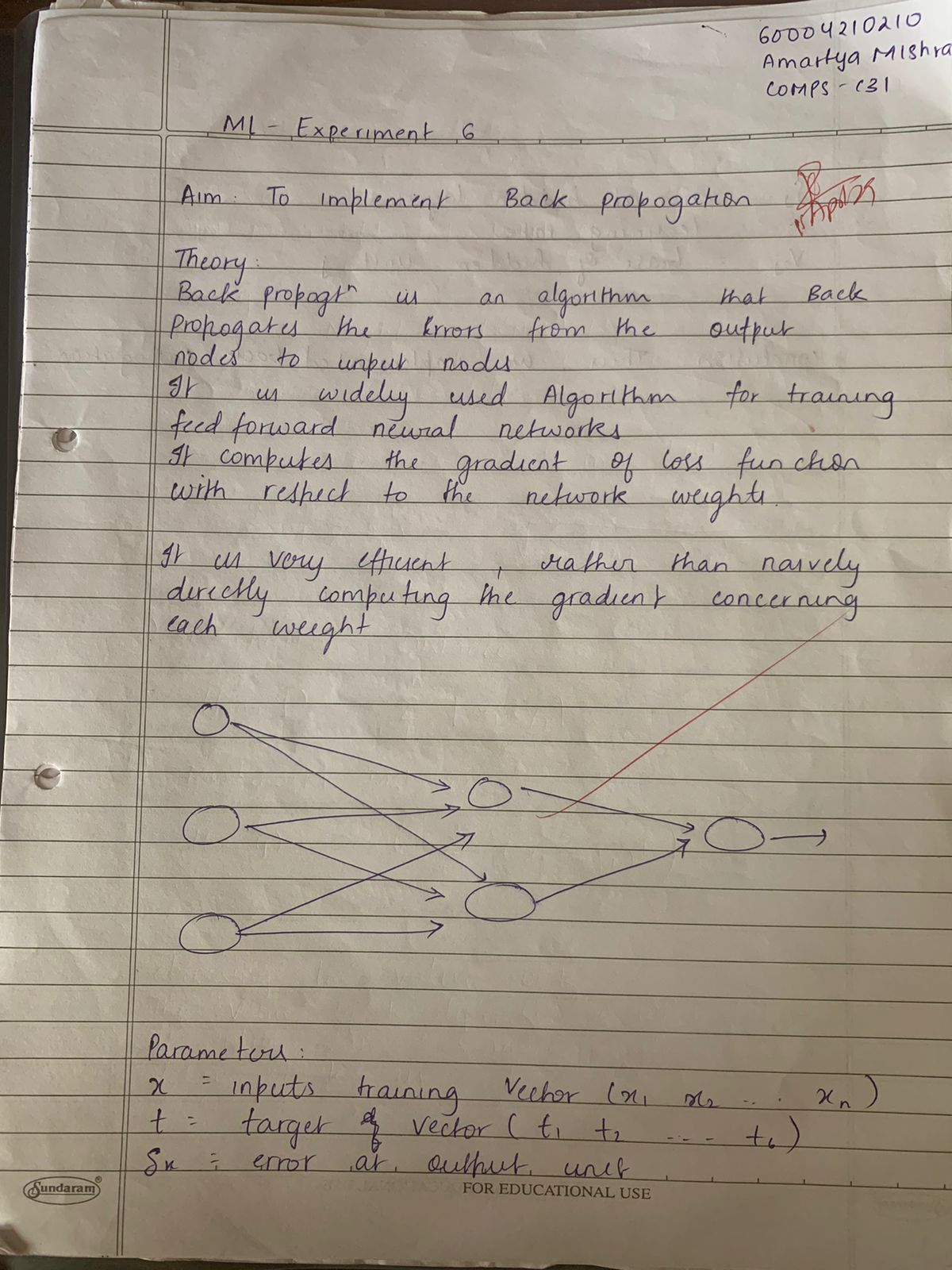
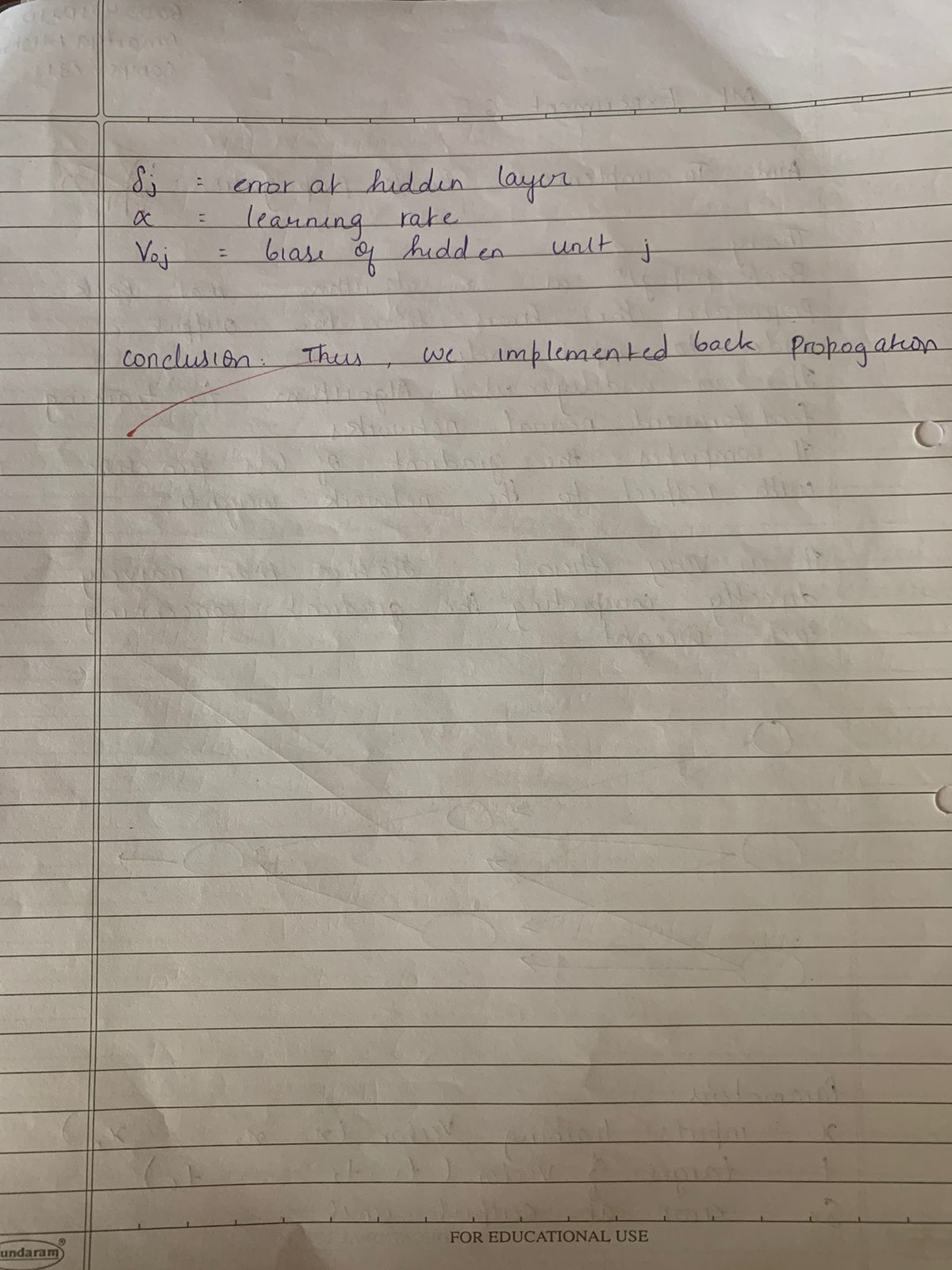
60004210210

Amartya Mishra

COMPS – C31

ML Experiment 6

****

****

**Implementation:**

import numpy as np

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

# Initialize weights and biases

self.weights\_input\_hidden = np.random.randn(self.input\_size,

self.hidden\_size)

self.bias\_input\_hidden = np.random.randn(1, self.hidden\_size)

self.weights\_hidden\_output = np.random.randn(self.hidden\_size,

self.output\_size)

self.bias\_hidden\_output = np.random.randn(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, inputs):

self.hidden\_input = np.dot(inputs, self.weights\_input\_hidden) +

self.bias\_input\_hidden

self.hidden\_output = self.sigmoid(self.hidden\_input)

self.final\_input = np.dot(self.hidden\_output,

self.weights\_hidden\_output) + self.bias\_hidden\_output

self.final\_output = self.sigmoid(self.final\_input)

return self.hidden\_output, self.final\_output

def backward(self, inputs, targets, learning\_rate):

error = targets - self.final\_output

delta\_output = error \* self.sigmoid\_derivative(self.final\_output)

delta\_hidden = np.dot(delta\_output, self.weights\_hidden\_output.T) \*

self.sigmoid\_derivative(self.hidden\_output)

self.weights\_hidden\_output += np.dot(self.hidden\_output.T,

delta\_output) \* learning\_rate

self.bias\_hidden\_output += np.sum(delta\_output, axis=0,

keepdims=True) \* learning\_rate

self.weights\_input\_hidden += np.dot(inputs.T, delta\_hidden) \*

learning\_rate

self.bias\_input\_hidden += np.sum(delta\_hidden, axis=0,

keepdims=True) \* learning\_rate

return error

def train(self, inputs, targets, learning\_rate):

hidden\_output, final\_output = self.forward(inputs)

error = self.backward(inputs, targets, learning\_rate)

print("Output of hidden layer:")

print(hidden\_output)

print("Output of output layer:")

print(final\_output)

print("Error found:")

print(error)

print("Updated weights after 1 iteration:")

print("Weights from input to hidden layer:")

print(self.weights\_input\_hidden)

print("Weights from hidden to output layer:")

print(self.weights\_hidden\_output)

dataset = pd.read\_csv('reduced\_digits\_dataset.csv')

inputs = dataset.drop(columns=['target']).values

targets = dataset['target'].values.reshape(-1, 1)

input\_size = inputs.shape[1]

output\_size = len(np.unique(targets))

hidden\_size = 3

nn = NeuralNetwork(input\_size, hidden\_size, output\_size)

nn.train(inputs, targets, learning\_rate=0.1)