Name : Azizullah
Roll NO: 2020MSCE104
Question: #4
Submitted : Dr. Faisal
Subject: Advance Machine learning assignment #3

==========

```
#!/usr/bin/python
import math

class NeturalNets:
	"""docstring for ClassName"""
	def __init__(self,input_layer,no_hindden_layer,target_layer,baised,wieghts):
		self.input_layer = input_layer
		self.no_hindden_layer = no_hindden_layer
		self.hindden_layer = []
		self.target_layer = target_layer
		self.wieghts = new_list = wieghts[:]
		self.update_wieghts = wieghts[:]
		self.baised = baised
		self.input_nets = []
		self.output_nets = []
		self.output = []
		self.error = []
		self.weight_index = 0
		self.final_wieghts = []

	def forward_propagation(self):
		# intialization of weights for new iteration
		self.weight_index = 0
		self.hindden_layer = []
		self.output = []

		# iterate over the input layer and find the values of hidden layers
		for node_h in range(self.no_hindden_layer):
			tmp =  0
			for node_i in range(len(self.input_layer)):
				# getting sum of weights with inputs and wieghts of input layer
```

```
                    tmp = self.input_layer[node_i]*self.wieghts[self.weight_index] + tmp
                    #moving to weight node
                    self.weight_index = self.weight_index + 1

            # got all the nets
            self.input_nets.append(tmp + self.baised[0])
            # applying the activation function
            self.hindden_layer.append(1/(1+(math.exp(-1*(tmp + self.baised[0])))))

        # iterate over the input layer and find the values of hidden layers
        for node_h in range(len(self.target_layer)):
            tmp =  0
            for node_i in range(len(self.hindden_layer)):
                # gett sum of weights with hidden layers inputs and weights
                tmp = self.wieghts[self.weight_index]*self.hindden_layer[node_i] + tmp
                #moving to next node
                self.weight_index = 1 + self.weight_index

            # appling activation function
            self.output.append(1/(1+(math.exp(-1*(tmp + self.baised[1])))))
            # gett all the nets of output
            self.output_nets.append(tmp + self.baised[1])

        sum_errors = 0
        # calculating sum of errors
        for out in range(len(self.output)):
            sum_errors = sum_errors + ((self.target_layer[out] - self.output[out])**2)/2

        # on last loop iteration increase by 1, but not used
        self.weight_index = self.weight_index-1

        return sum_errors

def back_propagation(self,n):
        """ updating the weights of output and hidden layer"""

        # updating the ouput layer weights

        # rate of change in error with respect ouput out
        error_out = self.output[0] - self.target_layer[0]
        # rate of change in output out with respect ouput net
        out_net = (self.output[0])*(1- self.output[0])

        for node_h in self.hindden_layer:
```

```
            # rate of change of error with respect to weight
            error_wieght = error_out * out_net * node_h
            # updating weights
            self.update_wieghts[self.weight_index] = self.wieghts[self.weight_index] - n * error_wieght
            # moving to next node
            self.weight_index = self.weight_index -1

    # updating the hidden layer weights
    index = self.weight_index+1
    netN = 0
    inputs_index = 0
    for weights_index in range(self.weight_index+1):
            # rate of change in error with respect to output net
            error_netF = error_out * out_net
            # rate of change in net output with respect node N out
            netF_outNode = self.wieghts[index]

            # check for next node of hidden layer
            if (weights_index+1) % len(self.input_layer)  == 0:
                    # move to next hidden node
                    index = index + 1
                    inputs_index = 0
                    # getting out node N value
                    outN = self.hindden_layer[int(weights_index / len(self.input_layer))]
            elif weights_index == 0:
                    # getting out node N value
                    outN = self.hindden_layer[int(weights_index / len(self.input_layer))]
            # rate of change of error with respect to node N out
            error_outN = error_netF * netF_outNode
            # rate of change of node N out with respect to node N net
            outN_netN = outN*(1-outN)

            if weights_index ==1:
                    netN_W = self.input_layer[inputs_index+1]
            else:
                    netN_W = self.input_layer[inputs_index]

            # calculating rate of change of error with respect to weight
            error_w = error_outN * outN_netN * netN_W

            # updating wieghts of hidden layer
            self.update_wieghts[self.weight_index] = self.wieghts[weights_index] - n * error_w

            inputs_index = inputs_index + 1
```

```
                # moving to next weight
                self.weight_index = self.weight_index -1

        # updating weights
        self.wieghts = self.update_wieghts[:]

    def train(self):
        from tqdm import tqdm
        minima = 10
        for i in tqdm(range(0, 70), total = 70, desc ="epoch"):
        # for x in range(100):
            error = self.forward_propagation()
            # passing learning rate of 1
            self.back_propagation(n=1)
            if error < minima:
                minima = error
                self.final_wieghts = self.update_wieghts[:]
        print ("====================================")
        print ("Minima: ",minima)
        print ("Final wieghts : ")
        print (self.final_wieghts)
        print ("====================================")


input_layer = [2,3]
no_hindden_layer = 3
target_layer = [0.1]
baised = [-0.2, -0.1]
wieghts = [0.1,-0.2,0.0,0.2,0.3,-0.4,0.3,0.3,-0.4]

NN_obj = NeturalNets(input_layer,no_hindden_layer,target_layer,baised, wieghts)
NN_obj.train()

"""
Output
==================================================
epoch: 100%|██████████| 70/70 [00:00<00:00, 29816.32it/s]
==================================================
minima:  0.00010299429490321281
final wieghts :
[0.3767835374474185, 0.2490766618347962, 0.11456060659372098, 0.28802264086476737,
0.8017450030841169, -0.07978661565773465, -0.5202505451794172, -0.6144478576333635,
-1.3351643771235187]
"""
```