**Artificial Intelligence Course**

**Assignment 08**



**Dr.Hashim Yasin**

Student Name: Abdul Salam & Taha Raja

Roll No: 18F-0326 & 18F-0137 Section: CS(6B)

Date performed: 02/07/2021

# Question no 01:

import numpy as np  
import pandas  
from pandas import read\_csv  
import random  
  
  
class nural\_network:  
 def \_\_init\_\_(self):  
 self.inputs = 0  
 self.layers = 1  
 self.neurons = [1]  
 self.w = []  
 self.total\_neuron = 1  
  
 def set\_layer(self, layers):  
 self.layers = layers  
 self.neurons = []  
 for n in range(0, layers):  
 self.neurons.append(1)  
  
 def set\_inputs(self, input):  
 self.inputs = input  
  
 def add\_neurons(self, neurons, layer\_number):  
 self.neurons[layer\_number - 1] = neurons  
 self.set\_weight()  
  
 def set\_weight(self):  
 self.total\_neuron = self.neurons[0]  
 total\_weight = self.inputs \* self.neurons[0]  
 for n in range(1, len(self.neurons)):  
 self.total\_neuron += self.neurons[n]  
 total\_weight += self.neurons[n-1] \* self.neurons[n]  
  
 self.w = []  
 total\_weight += self.total\_neuron  
 for n in range(0, total\_weight):  
 self.w.append(0.05)  
  
 def sigmoid(self, y):  
 sigma\_s = 1.0 / (1.0 + np.exp(-y))  
 return sigma\_s  
  
 def solve(self, x , y):  
  
 epoch = 10000  
 #self.w = [0.05, 0.05, 0.05, 0.03, 0.04, 0.02, 0.01, -0.02, 0.02]  
 #x = [[0,0],[0,1],[1,0],[1,1]]  
 #solution = [0,1,1,0]  
 #solution = [[0,0],[0,1],[1,0],[1,1]]  
 #dw = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  
 dw = []  
 for i in range(0, len(self.w)):  
 dw.append(0)  
  
 eta = 0.05  
 thresh = 2  
  
  
 for e in range(epoch):  
  
 o1 = 1  
 o2 = 1  
  
 for sol in range(0, len(X)):  
 count1 = 0  
 count2 = 0  
  
 actual = y[sol][0]  
  
 h1 = self.sigmoid((-self.w[0] \* thresh) + self.w[1] \* x[sol][0] + self.w[2] \* x[sol][1] + self.w[3] \* x[sol][2] + self.w[4] \* x[sol][3] + self.w[5] \* x[sol][4] + self.w[6] \* x[sol][5] + self.w[7] \* x[sol][6] + self.w[8] \* x[sol][7] + self.w[9] \* x[sol][8])  
 h2 = self.sigmoid((-self.w[10] \* thresh) + self.w[11] \* x[sol][0] + self.w[12] \* x[sol][1] + self.w[13] \* x[sol][2] + self.w[14] \* x[sol][3] + self.w[15] \* x[sol][4] + self.w[16] \* x[sol][5] + self.w[17] \* x[sol][6] + self.w[18] \* x[sol][7] + self.w[19] \* x[sol][8])  
  
 output1 = self.sigmoid(self.w[20] \* thresh + self.w[21] \* h1 + self.w[22] \* h2)  
 #output2 = self.sigmoid(self.w[9] \* thresh + self.w[10] \* h1 + self.w[11] \* h2)  
  
  
 if e %500 == 0:  
  
 if actual == 1 and o1 == 1:  
 print('epoch: ', e)  
 print(actual , ' --> ', output1)  
 o1 = 0  
 if actual == 0 and o2 == 1:  
 print(actual , ' --> ', output1)  
 o2 = 0  
 print('--------------------')  
  
 sigma\_h1 =0  
 sigma\_h2 =0  
 sigma\_output1 = 0  
 if output1 == actual:  
 nulll = 0  
 #pcount += 1  
 #break  
 else:  
 sigma\_output1 = output1 \* (1 - output1) \* (actual - output1)  
 #sigma\_output2 = output2 \* (1 - output2) \* (actual[1] - output2)  
 sigma\_h1 = h1 \* (1 - h1) \* ((sigma\_output1 \* self.w[21]))  
 sigma\_h2 = h2 \* (1 - h2) \* ((sigma\_output1 \* self.w[22]))  
  
  
 for k in range(0, len(dw)-3):  
 if k == 0:  
 dw[k] = eta \* (sigma\_h1 \* thresh)  
 elif k == 10:  
 dw[k] = eta \* (sigma\_h2 \* thresh)  
 elif k < 10:  
 dw[k] = eta \* (sigma\_h1 \* x[sol][k-1])  
 else:  
 dw[k] = eta \* (sigma\_h2 \* x[sol][k-11])  
  
 self.w[k] += dw[k]  
  
 dw[20] = eta \* (sigma\_output1) \* thresh  
 self.w[20] += dw[20]  
 dw[21] = eta \* (sigma\_output1) \* h1  
 self.w[21] += dw[21]  
 dw[22] = eta \* (sigma\_output1) \* h2  
 self.w[22] += dw[22]  
  
  
  
  
 def test(self, x, y):  
  
 thresh = 1  
  
 r = random.randint(0, len(Y))  
  
 print('Test: ')  
  
 actual = y[r][0]  
  
 h1 = self.sigmoid(  
 self.w[0] \* thresh + self.w[1] \* x[r][0] + self.w[2] \* x[r][1] + self.w[3] \* x[r][2] + self.w[  
 4] \* x[r][3] + self.w[5] \* x[r][4] + self.w[6] \* x[r][5] + self.w[7] \* x[r][6] + self.w[  
 8] \* x[r][7] + self.w[9] \* x[r][8])  
 h2 = self.sigmoid(  
 self.w[10] \* thresh + self.w[11] \* x[r][0] + self.w[12] \* x[r][1] + self.w[13] \* x[r][2] +  
 self.w[14] \* x[r][3] + self.w[15] \* x[r][4] + self.w[16] \* x[r][5] + self.w[17] \* x[r][6] +  
 self.w[18] \* x[r][7] + self.w[19] \* x[r][8])  
  
 output1 = self.sigmoid(self.w[20] \* thresh + self.w[21] \* h1 + self.w[22] \* h2)  
 # output2 = self.sigmoid(self.w[9] \* thresh + self.w[10] \* h1 + self.w[11] \* h2)  
  
 print(actual, ' --> ', output1)  
  
 print('--------------------')  
  
  
Class ={  
 'no-recurrence-events': 1,  
 'recurrence-events': 0  
}  
age ={ '10-19' : 1,  
 '20-29' : 2,  
 '30-39' : 3,  
 '40-49' : 4,  
 '50-59' : 5,  
 '60-69' : 6,  
 '70-79' : 7,  
 '80-89' : 8,  
 '90-99' : 9,  
 '?' : 0  
}  
menopause ={ 'lt40' :1,  
 'ge40' :2,  
 'premeno' :3,  
 '?' : 0  
}  
tumor\_size = { '0-4' : 1,  
 '5-9' : 2,  
 '10-14' : 3,  
 '15-19' : 4,  
 '20-24' : 5,  
 '25-29' : 6,  
 '30-34' : 7,  
 '35-39' : 8,  
 '40-44' : 9,  
 '45-49' : 10,  
 '50-54' : 11,  
 '55-59' : 12,  
 '?' : 0  
}  
inv\_nodes ={ '0-2' : 1,  
 '3-5' : 2,  
 '6-8' : 3,  
 '9-11' : 4,  
 '12-14' : 5,  
 '15-17' : 6,  
 '18-20' : 7,  
 '21-23' : 8,  
 '24-26' : 9,  
 '27-29' : 10,  
 '30-32' : 11,  
 '33-35' : 12,  
 '36-39' : 13,  
 '?' : 0  
}  
node\_caps ={ 'yes' : 1,  
 'no' : 0,  
 '?' : 2  
}  
deg\_malig ={ '1' : 1,  
 '2' : 2,  
 '3' : 3,  
 '?' : 0  
}  
breast ={ 'left' : 1,  
 'right' : 2,  
 '?' : 0  
 }  
breast\_quad ={ 'left\_up' : 1,  
 'left\_low' : 2,  
 'right\_up' : 3,  
 'right\_low' : 4,  
 'central' : 5,  
 '?' : 0  
 }  
irradiat ={ 'yes' : 1,  
 'no' : 0,  
 '?' : 2  
}  
  
  
url = 'breast-cancer.csv'  
names = ['Class','age','menopause','tumor-size','inv-nodes','node-caps','deg-malig','breast','breast-quad','irradiat']  
dataset = read\_csv(url, names=names)  
array = dataset.values  
X = array[1:,1:10]  
Y = array[1:,0:1]  
for i in range(0,len(X)):  
 X[i][0] = age[X[i][0]]  
 X[i][1] = menopause[X[i][1]]  
 X[i][2] = tumor\_size[X[i][2]]  
 X[i][3] = inv\_nodes[X[i][3]]  
 X[i][4] = node\_caps[X[i][4]]  
 X[i][5] = deg\_malig[X[i][5]]  
 X[i][6] = breast[X[i][6]]  
 X[i][7] = breast\_quad[X[i][7]]  
 X[i][8] = irradiat[X[i][8]]  
 Y[i][0] = Class[Y[i][0]]  
  
  
ANN = nural\_network()  
ANN.set\_inputs(9)  
ANN.set\_layer(2)  
ANN.add\_neurons(2,1)  
ANN.add\_neurons(1,2)  
ANN.solve(X, Y)  
  
ANN.test(X,Y)  
  
  
print("O K ")  
  
  
  
"""  
1. Class: no-recurrence-events, recurrence-events  
2. age: 10-19, 20-29, 30-39, 40-49, 50-59, 60-69, 70-79, 80-89, 90-99.  
3. menopause: lt40, ge40, premeno.  
4. tumor-size: 0-4, 5-9, 10-14, 15-19, 20-24, 25-29, 30-34, 35-39, 40-44, 45-49, 50-54, 55-59.  
5. inv-nodes: 0-2, 3-5, 6-8, 9-11, 12-14, 15-17, 18-20, 21-23, 24-26, 27-29, 30-32, 33-35, 36-39.  
6. node-caps: yes, no.  
7. deg-malig: 1, 2, 3.  
8. breast: left, right.  
9. breast-quad: left-up, left-low, right-up, right-low, central.  
10. irradiat: yes, no.  
  
"""

# Screenshot: