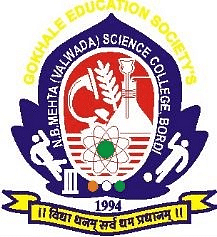
SOFT COMPUTING TECHNIQUES

**MASTERS OF SCIENCE (INFORMATION TECHNOLOGY)**

**BY**

**Name: CHAMPANERI PARTH ASHOK**

**Seat No.**



N.B. MEHTA (VALWADA) SCIENCE COLLEGE BORDI

MAHARASHTRA - 401701

DEPARTMENT OF INFORMATION TECHNOLOGY

MASTERS OF SCIENCE (INFORMATION TECHNOLOGY)

Semester-1

Academic year 2024-2025



**CERTIFICATE**

Class: M.Sc. Information Technology (Semester 1)

Year: 2024-2025

This is to certify that the work entered in this journal is the work of MR. **CHAMPANERI PARTH ASHOK** of **M.Sc.IT Part 1** division **Information Technology** Roll No. **04** Uni. Exam No has satisfactorily completed the required number of practical and worked for the terms of the Year 2024-25 in the college laboratory as laid down by the university.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Head of the External Internal Examiner

Department Examiner Subject teacher

Date : / / 2025 Department of IT-CS

# INDEX

|  |  |  |
| --- | --- | --- |
| **Practical No** | **Details** | **Page No.** |
| **1** | **Implement the following** | **4** |
| A | Design a simple linear neural network model |  |
| B | Calculate the output of neural network using both binary and  bipolar sigmoidal function |  |
| **2** | **Implement the following** | **6** |
| A | Generate AND/NOT function using McCulloch-Pitts neural net. |  |
| B | Generate XOR function using McCulloch-Pitts neural net. |  |
| **3** | **Implement the Following** | **10** |
| A | Write a program to implement Hebb’s rule. |  |
| B | Write a program to implement Delta rule. |  |
| **4** | **Implement the Following** | **13** |
| A | Write a program for Back Propagation Algorithm. |  |
| B | Write a program for error Backpropagation algorithm. |  |
| **5** | **Write a program for Radial Basis function** | **17** |
| **6** | **Kohonen Self organizing map** | **19** |
| **7** | **Implement the Following** | **20** |
| A | Write a program for Linear separation. |  |
| **8** | **Implement the Following** | **22** |
| A | Membership and Identity Operators | in, not in. |  |
| B | Membership and Identity Operators is, is not |  |
| **9** | **Implement the Following** | **23** |
| A | Find ratios using fuzzy logic |  |
| B | Solve Tipping problem using fuzzy logic |  |

**Practical 1a**

**Aim: Design a simple linear neural network model.**

**Code :**

x=float(input("Enter value of x:"))

w=float(input("Enter value of weight w:"))

b=float(input("Enter value of bias b:"))

net = int(w\*x+b)

if(net<0):

out=0

elif((net>=0)&(net<=1)):

out =net

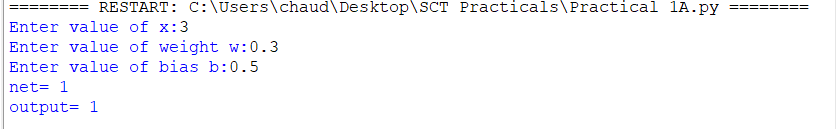
else:

out=1

print("net=",net)

print("output=",out)

**Output :**



**Practical 1b**

**Aim: Calculate the output of neural net using both binary and bipolar sigmoidal function.**

**Code :**

# number of elements as input

n = int(input("Enter number of elements : "))

# In[2]:

print("Enter the inputs")

inputs = [] # creating an empty list for inputs

# iterating till the range

for i in range(0, n):

ele = float(input())

inputs.append(ele) # adding the element

print(inputs)

# In[3]:

print("Enter the weights")

# creating an empty list for weights

weights = []

# iterating till the range

for i in range(0, n):

ele = float(input())

weights.append(ele) # adding the element

print(weights)

# In[4]:

print("The net input can be calculated as Yin = x1w1 + x2w2 + x3w3")

# In[5]:

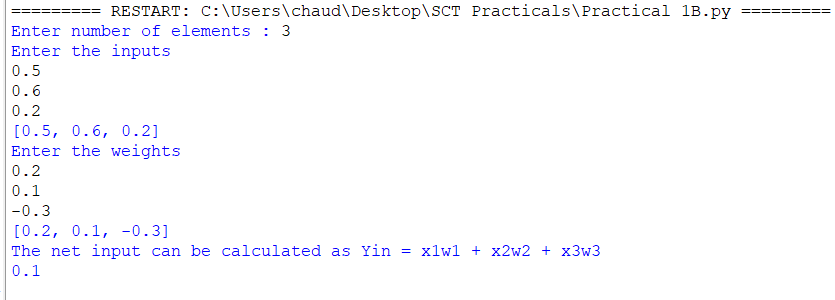
Yin = []

for i in range(0, n):

Yin.append(inputs[i]\*weights[i])

print(round(sum(Yin),3))

**Output:**



**Practical 2a**

**Aim: Generate AND/NOT function using McCulloch-Pitts neural net.**

**Code :**

# enter the no of inputs

num\_ip = int(input("Enter the number of inputs : "))

#Set the weights with value 1

w1 = 1

w2 = 1

print("For the ", num\_ip , " inputs calculate the net input using yin = x1w1 + x2w2 ")

x1 = []

x2 = []

for j in range(0, num\_ip):

ele1 = int(input("x1 = "))

ele2 = int(input("x2 = "))

x1.append(ele1)

x2.append(ele2)

print("x1 = ",x1)

print("x2 = ",x2)

n = x1 \* w1

m = x2 \* w2

Yin = []

for i in range(0, num\_ip):

Yin.append(n[i] + m[i])

print("Yin = ",Yin)

#Assume one weight as excitatory and the other as inhibitory, i.e.,

Yin = []

for i in range(0, num\_ip):

Yin.append(n[i] - m[i])

print("After assuming one weight as excitatory and the other as inhibitory Yin = ",Yin)

#From the calculated net inputs, now it is possible to fire the neuron for input (1, 0)

#only by fixing a threshold of 1, i.e., θ ≥ 1 for Y unit.

#Thus, w1 = 1, w2 = -1; θ ≥ 1

Y=[]

for i in range(0, num\_ip):

if(Yin[i]>=1):

ele= 1

Y.append(ele)

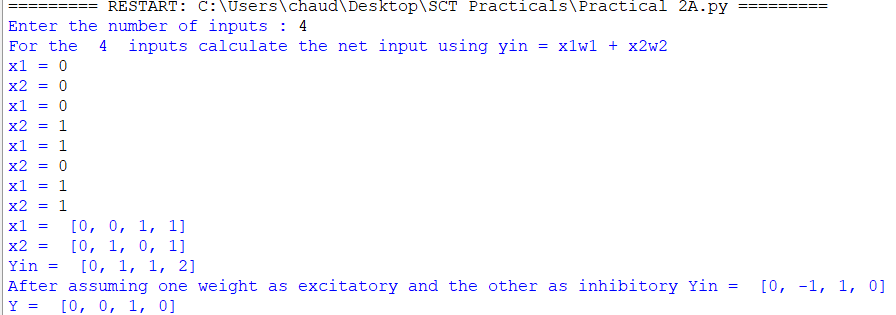
if(Yin[i]<1):

ele= 0

Y.append(ele)

print("Y = ",Y)

**Output:**

****

**Practical 2b**

**Aim : Generate XOR function using McCulloch-Pitts neural net**

**Code:**

import numpy as np

print('Enter weights')

w11=int(input('Weight w11='))

w12=int(input('weight w12='))

w21=int(input('Weight w21='))

w22=int(input('weight w22='))

v1=int(input('weight v1='))

v2=int(input('weight v2='))

print('Enter Threshold Value')

theta=int(input('theta='))

x1=np.array([0, 0, 1, 1])

x2=np.array([0, 1, 0, 1])

z=np.array([0, 1, 1, 0])

con=1

y1=np.zeros((4,))

y2=np.zeros((4,))

y=np.zeros((4,))

while con==1:

zin1=np.zeros((4,))

zin2=np.zeros((4,))

zin1=x1\*w11+x2\*w21

zin2=x1\*w21+x2\*w22

print("z1",zin1)

print("z2",zin2)

for i in range(0,4):

if zin1[i]>=theta:

y1[i]=1

else:

y1[i]=0

if zin2[i]>=theta:

y2[i]=1

else:

y2[i]=0

yin=np.array([])

yin=y1\*v1+y2\*v2

for i in range(0,4):

if yin[i]>=theta:

y[i]=1

else:

y[i]=0

print("yin",yin)

print('Output of Net')

y=y.astype(int)

print("y",y)

print("z",z)

if np.array\_equal(y,z):

con=0

else:

print("Net is not learning enter another set of weights and Threshold value")

w11=input("Weight w11=")

w12=input("weight w12=")

w21=input("Weight w21=")

w22=input("weight w22=")

v1=input("weight v1=")

v2=input("weight v2=")

theta=input("theta=")

print("McCulloch-Pitts Net for XOR function")

print("Weights of Neuron Z1")

print(w11)

print(w21)

print("weights of Neuron Z2")

print(w12)

print(w22)

print("weights of Neuron Y")

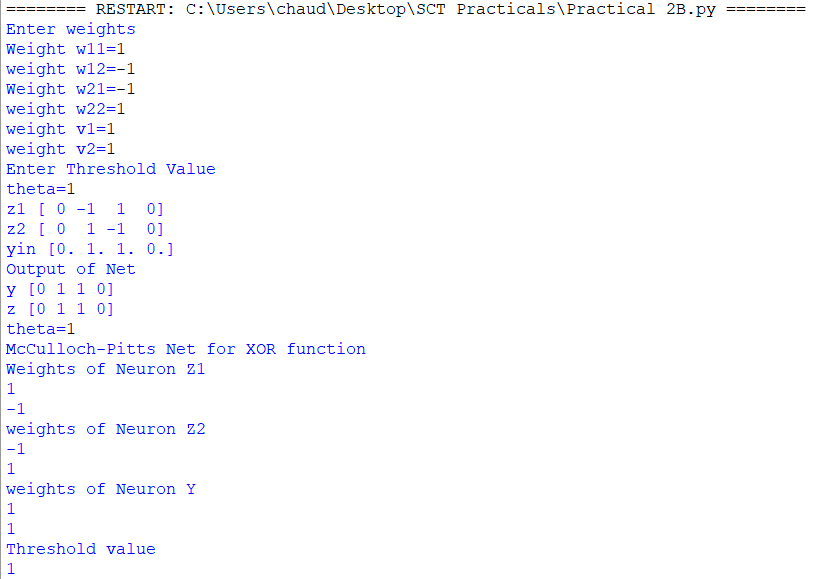
print(v1)

print(v2)

print("Threshold value")

print(theta)

**output:**



**Practical 3a**

**Aim : Write a program to implement Hebb’s rule.**

**Code :**

import numpy as np

#first pattern

x1=np.array([1,1,1,-1,1,-1,1,1,1])

#second pattern

x2=np.array([1,1,1,1,-1,1,1,1,1])

#initialize bais value

b=0

#define target

y=np.array([1,-1])

wtold=np.zeros((9,))

wtnew=np.zeros((9,))

wtnew=wtnew.astype(int)

wtold=wtold.astype(int)

bais=0

print("First input with target =1")

for i in range(0,9):

wtold[i]=wtold[i]+x1[i]\*y[0]

wtnew=wtold

b=b+y[0]

print("new wt =", wtnew)

print("Bias value",b)

print("Second input with target =-1")

for i in range(0,9):

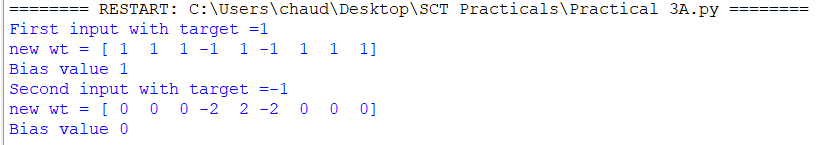
wtnew[i]=wtold[i]+x2[i]\*y[1]

b=b+y[1]

print("new wt =", wtnew)

print("Bias value",b)

**Output:**



**Practical 3b**

**Aim: Write a program to implement of delta rule.**

**Code :**

#supervised learning

import numpy as np

import time

np.set\_printoptions(precision=2)

x=np.zeros((3,))

weights=np.zeros((3,))

desired=np.zeros((3,))

actual=np.zeros((3,))

for i in range(0,3):

x[i]=float(input("Initial inputs:"))

for i in range(0,3):

weights[i]=float(input("Initial weights:"))

for i in range(0,3):

desired[i]=float(input("Desired output:"))

a=float(input("Enter learning rate:"))

actual=x\*weights

print("actual",actual)

print("desired",desired)

while True:

if np.array\_equal(desired,actual):

break #no change

else:

for i in range(0,3):

weights[i]=weights[i]+a\*(desired[i]-actual[i])

actual=x\*weights

print("weights",weights)

print("actual",actual)

print("desired",desired)

print("\*"\*30)

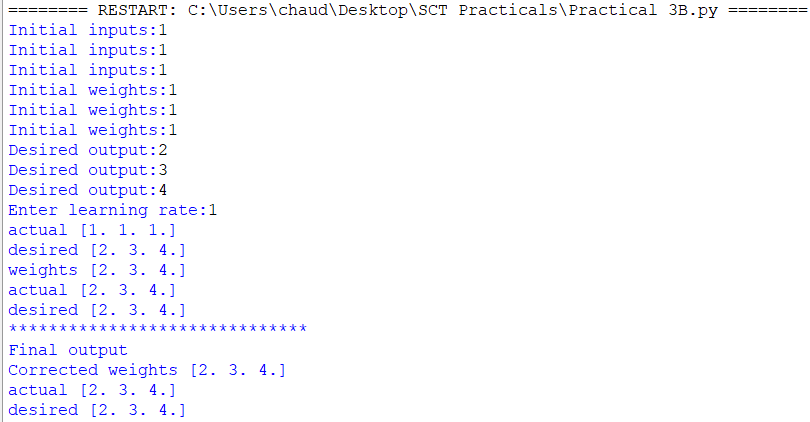
print("Final output")

print("Corrected weights",weights)

print("actual",actual)

print("desired",desired)

**Output:**



**Practical 4a**

**Aim: Write a program for Back Propagation Algorithm**

**Code :**

import numpy as np

import decimal

import math

np.set\_printoptions(precision=2)

v1=np.array([0.6, 0.3])

v2=np.array([-0.1, 0.4])

w=np.array([-0.2,0.4,0.1])

b1=0.3

b2=0.5

x1=0

x2=1

alpha=0.25

print("calculate net input to z1 layer")

zin1=round(b1+ x1\*v1[0]+x2\*v2[0],4)

print("z1=",round(zin1,3))

print("calculate net input to z2 layer")

zin2=round(b2+ x1\*v1[1]+x2\*v2[1],4)

print("z2=",round(zin2,4))

print("Apply activation function to calculate output")

z1=1/(1+math.exp(-zin1))

z1=round(z1,4)

z2=1/(1+math.exp(-zin2))

z2=round(z2,4)

print("z1=",z1)

print("z2=",z2)

print("calculate net input to output layer")

yin=w[0]+z1\*w[1]+z2\*w[2]

print("yin=",yin)

print("calculate net output")

y=1/(1+math.exp(-yin))

print("y=",y)

fyin=y \*(1- y)

dk=(1-y)\*fyin

print("dk",dk)

dw1= alpha \* dk \* z1

dw2= alpha \* dk \* z2

dw0= alpha \* dk

print("compute error portion in delta")

din1=dk\* w[1]

din2=dk\* w[2]

print("din1=",din1)

print("din2=",din2)

print("error in delta")

fzin1= z1 \*(1-z1)

print("fzin1",fzin1)

d1=din1\* fzin1

fzin2= z2 \*(1-z2)

print("fzin2",fzin2)

d2=din2\* fzin2

print("d1=",d1)

print("d2=",d2)

print("Changes in weights between input and hidden layer")

dv11=alpha \* d1 \* x1

print("dv11=",dv11)

dv21=alpha \* d1 \* x2

print("dv21=",dv21)

dv01=alpha \* d1

print("dv01=",dv01)

dv12=alpha \* d2 \* x1

print("dv12=",dv12)

dv22=alpha \* d2 \* x2

print("dv22=",dv22)

dv02=alpha \* d2

print("dv02=",dv02)

print("Final weights of network")

v1[0]=v1[0]+dv11

v1[1]=v1[1]+dv12

print("v=",v1)

v2[0]=v2[0]+dv21

v2[1]=v2[1]+dv22

print("v2",v2)

w[1]=w[1]+dw1

w[2]=w[2]+dw2

b1=b1+dv01

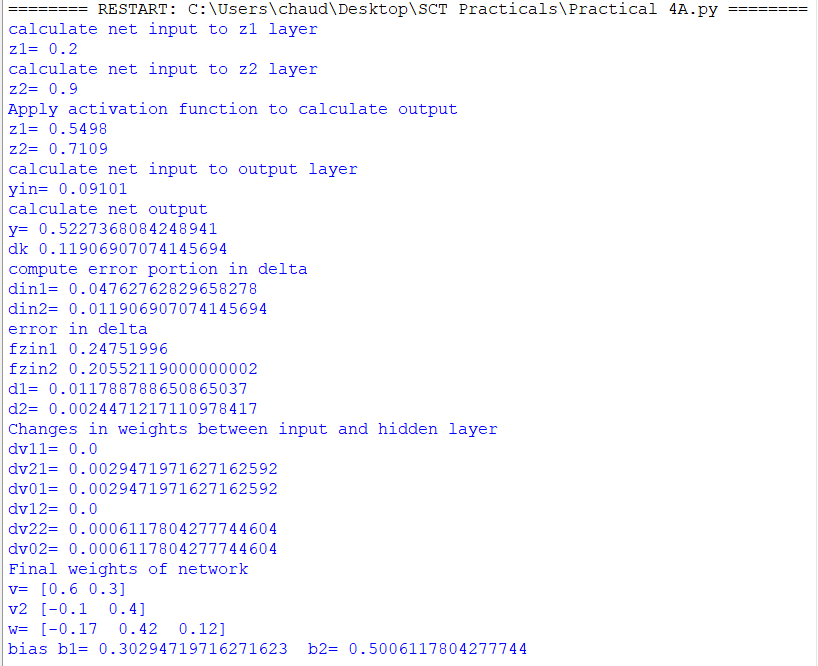
b2=b2+dv02

w[0]=w[0]+dw0

print("w=",w)

print("bias b1=",b1, " b2=",b2)

**Output:**



**Practical 4b**

**Aim : Write a Program For Error Back Propagation Algorithm (Ebpa) Learning**

**Code :**

import math

a0=-1

t=-1

w10=float(input("Enter weight first network"))

b10=float(input("Enter base first network:"))

w20=float(input("Enter weight second network:"))

b20=float(input("Enter base second network:"))

c=float(input("Enter learning coefficient:"))

n1=float(w10\*c+b10)

a1=math.tanh(n1)

n2=float(w20\*a1+b20)

a2=math.tanh(float(n2))

e=t-a2

s2=-2\*(1-a2\*a2)\*e

s1=(1-a1\*a1)\*w20\*s2

w21=w20-(c\*s2\*a1)

w11=w10-(c\*s1\*a0)

b21=b20-(c\*s2)

b11=b10-(c\*s1)

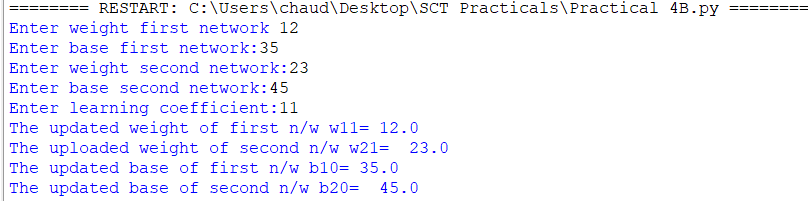
print("The updated weight of first n/w w11=",w11)

print("The uploaded weight of second n/w w21= ",w21)

print("The updated base of first n/w b10=",b10)

print("The updated base of second n/w b20= ",b20)

**Output:**



**Practical 5b**

**Aim: Write a program for Radial Basis function**

**Code:**

from numpy import \*

from scipy import \*

from scipy.linalg import norm, pinv

from matplotlib import pyplot as plt

class RBF:

def \_\_init\_\_(self, indim, numCenters, outdim):

self.indim =indim

self.outdim =outdim

self.numCenters =numCenters

self.centers =[random.uniform(-1, 1, indim) for i in range(numCenters)]

self.beta =8

self.W =random.random((self.numCenters, self.outdim))

def \_basisfunc(self, c, d):

assert len(d) ==self.indim

return exp(-self.beta \*norm(c-d)\*\*2)

def \_calcAct(self, X):

G =zeros((X.shape[0], self.numCenters), float)

for ci, c in enumerate(self.centers):

for xi, x in enumerate(X):

G[xi,ci] =self.\_basisfunc(c, x)

return G

def train(self, X, Y):

""" X: matrix of dimensions n x indim

y: column vector of dimension n x 1 """

# choose random center vectors from training set

rnd\_idx =random.permutation(X.shape[0])[:self.numCenters]

self.centers =[X[i,:] for i in rnd\_idx]

print("center", self.centers)

# calculate activations of RBFs

G =self.\_calcAct(X)

print (G)

# calculate output weights (pseudoinverse)

self.W =dot(pinv(G), Y)

def test(self, X):

""" X: matrix"""

G =self.\_calcAct(X)

Y =dot(G, self.W)

return Y

if \_\_name\_\_ =='\_\_main\_\_':

# ----- 1D Example ------------------------------------------------

n =100

x =mgrid[-1:1:complex(0,n)].reshape(n, 1)

# set y and add random noise

y =sin(3\*(x+0.5)\*\*3-1)

# y += random.normal(0, 0.1, y.shape)

# rbf regression

rbf =RBF(1, 10, 1)

rbf.train(x, y)

z =rbf.test(x)

# plot original data

plt.figure(figsize=(12, 8))

plt.plot(x, y, 'k-')

# plot learned model

plt.plot(x, z, 'r-', linewidth=2)

# plot rbfs

plt.plot(rbf.centers, zeros(rbf.numCenters), 'gs')

for c in rbf.centers:

# RF prediction lines

cx =arange(c-0.7, c+0.7, 0.01)

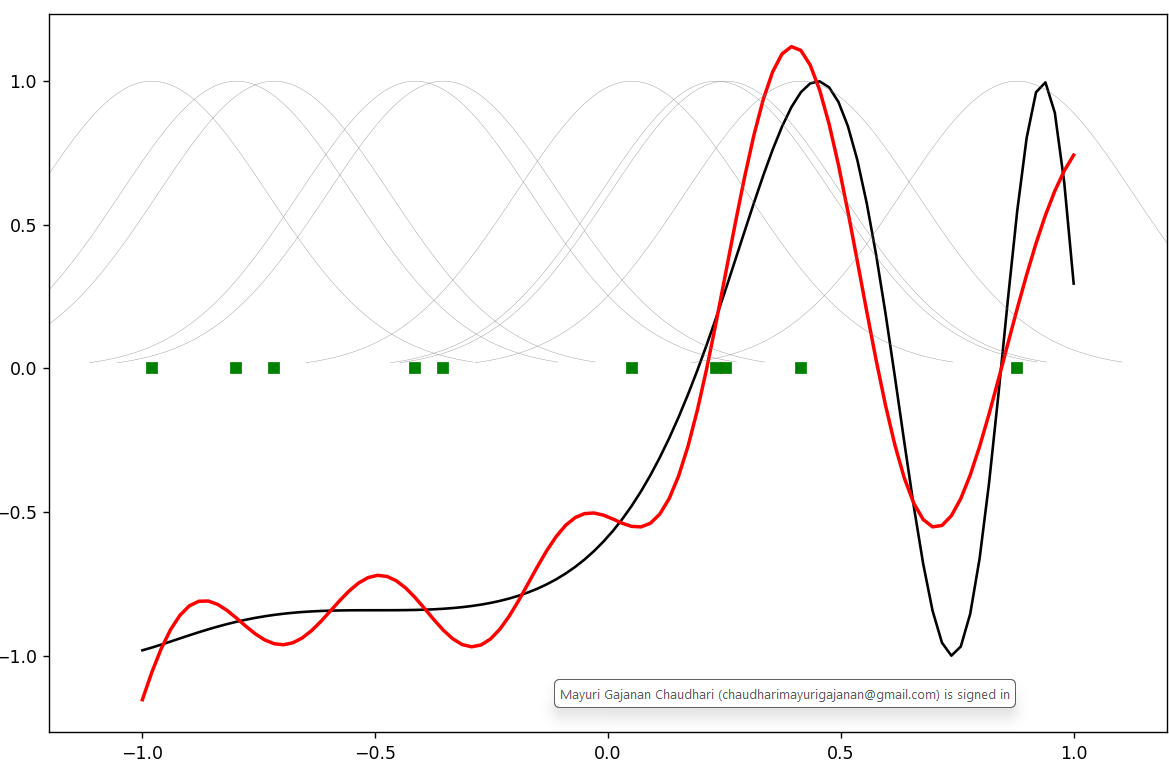
cy =[rbf.\_basisfunc(array([cx\_]), array([c])) for cx\_ in cx]

plt.plot(cx, cy, '-', color='gray', linewidth=0.2)

plt.xlim(-1.2, 1.2)

plt.show()

**Output:**



**Practical 6a**

**Aim: Write a program for Kohonen Self organizing map**

**Code :**

from minisom import MiniSom

import matplotlib.pyplot as plt

data = [[ 0.80, 0.55, 0.22, 0.03],

[ 0.82, 0.50, 0.23, 0.03],

[ 0.80, 0.54, 0.22, 0.03],

[ 0.80, 0.53, 0.26, 0.03],

[ 0.79, 0.56, 0.22, 0.03],

[ 0.75, 0.60, 0.25, 0.03],

[ 0.77, 0.59, 0.22, 0.03]]

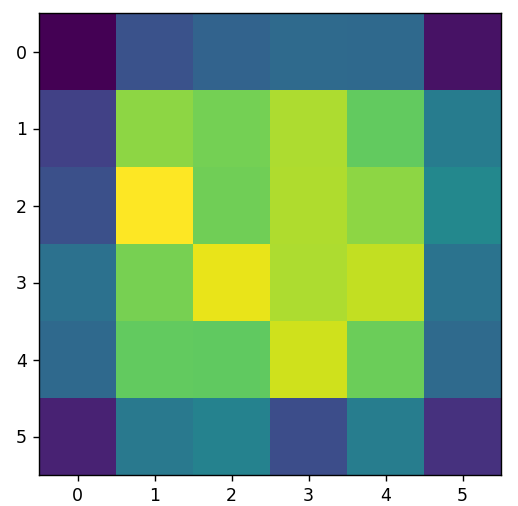
som = MiniSom(6, 6, 4, sigma=0.3, learning\_rate=0.5) # initialization of 6x6 SOM

som.train\_random(data, 100) # trains the SOM with 100 iterations

plt.imshow(som.distance\_map())

plt.show()

**Output:**



**Practical 7a**

**Aim: Write a program for Linear separation.**

**Code :**

import numpy as np

import matplotlib.pyplot as plt

def create\_distance\_function(a, b, c):

""" 0 = ax + by + c """

def distance(x, y):

""" returns tuple (d, pos)

d is the distance

If pos == -1 point is below the line,

0 on the line and +1 if above the line

"""

nom = a \* x + b \* y + c

if nom == 0:

pos = 0

elif (nom<0 and b<0) or (nom>0 and b>0):

pos = -1

else:

pos = 1

return (np.absolute(nom) / np.sqrt( a \*\* 2 + b \*\* 2), pos)

return distance

points = [ (3.5, 1.8), (1.1, 3.9) ]

fig, ax = plt.subplots()

ax.set\_xlabel("sweetness")

ax.set\_ylabel("sourness")

ax.set\_xlim([-1, 6])

ax.set\_ylim([-1, 8])

X = np.arange(-0.5, 5, 0.1)

colors = ["r", ""] # for the samples

size = 10

for (index, (x, y)) in enumerate(points):

if index== 0:

ax.plot(x, y, "o", color="darkorange", markersize=size)

else:

ax.plot(x, y, "oy", markersize=size)

step = 0.05

for x in np.arange(0, 1+step, step):

slope = np.tan(np.arccos(x))

dist4line1 = create\_distance\_function(slope, -1, 0)

#print("x: ", x, "slope: ", slope)

Y = slope \* X

results = []

for point in points:

results.append(dist4line1(\*point))

#print(slope, results)

if (results[0][1] != results[1][1]):

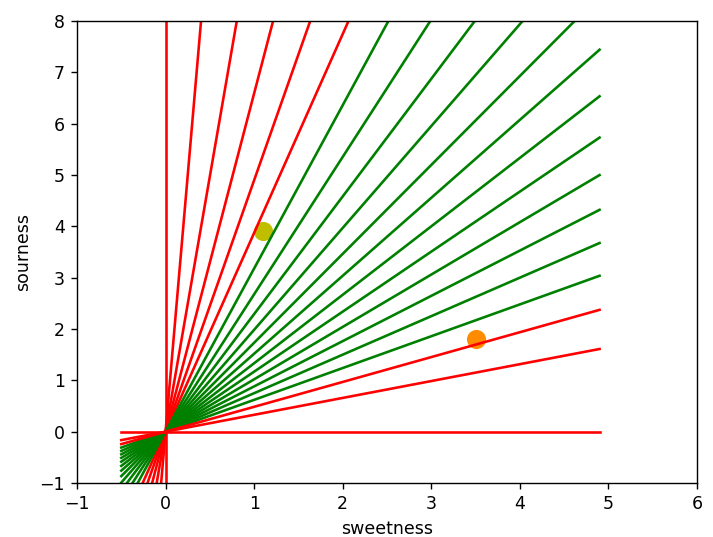
ax.plot(X, Y, "g-")

else:

ax.plot(X, Y, "r-")

plt.show()

**Output:**



**Practical 8a**

**Aim: Membership and Identity Operators | in, not in,**

**Code :**

# Python program to illustrate

# Finding common member in list

# without using 'in' operator

# Define a function() that takes two lists

def overlapping(list1,list2):

c=0

d=0

for i in list1:

c+=1

for i in list2:

d+=1

for i in range(0,c):

for j in range(0,d):

if(list1[i]==list2[j]):

return 1

return 0

list1=[1,2,3,4,6]

list2=[6,7,8,9]

if(overlapping(list1,list2)):

print("overlapping")

else:

print("not overlapping")

**Output: overlapping**

**Practical 8 b**

**Aim: Membership and Identity Operators is, is not**

**Code :**

# Python program to illustrate the use

# of 'is' identity operator

'''x = 5

if (type(x) is int):

print ("true")

else:

print ("false")'''

# Python program to illustrate the

# use of 'is not' identity operator

x = 1

if (type(x) is not int):

print ("true")

else:

print ("false")

**Output: false**

**Practical 9a**

**Aim: Find the ratios using fuzzy logic**

**Code :**

# Python code showing all the ratios together,

# make sure you have installed fuzzywuzzy module

from fuzzywuzzy import fuzz

from fuzzywuzzy import process

s1 = "I love fuzzysforfuzzys"

s2 = "I am loving fuzzysforfuzzys"

print ("FuzzyWuzzy Ratio:", fuzz.ratio(s1, s2))

print ("FuzzyWuzzyPartialRatio: ", fuzz.partial\_ratio(s1, s2))

print ("FuzzyWuzzyTokenSortRatio: ", fuzz.token\_sort\_ratio(s1, s2))

print ("FuzzyWuzzyTokenSetRatio: ", fuzz.token\_set\_ratio(s1, s2))

print ("FuzzyWuzzyWRatio: ", fuzz.WRatio(s1, s2),'\n\n')

# for process library,

query = 'fuzzys for fuzzys'

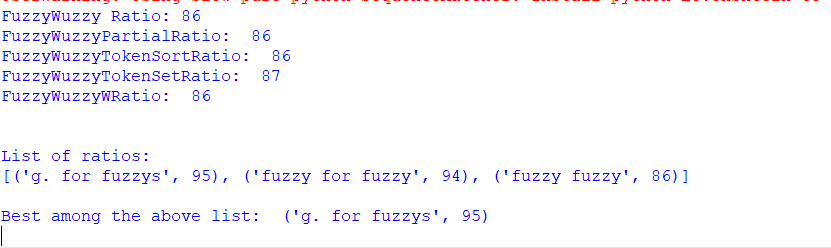
choices = ['fuzzy for fuzzy', 'fuzzy fuzzy', 'g. for fuzzys']

print ("List of ratios: ")

print (process.extract(query, choices), '\n')

print ("Best among the above list: ",process.extractOne(query, choices))

**Output:**



**Practical 9 b**

**Practical 9b**

**Aim: Solve Tipping Problem using fuzzy logic**

**Code :**

import numpy as np

import skfuzzy as fuzz

from skfuzzy import control as ctrl

quality = ctrl.Antecedent(np.arange(0, 11, 1), 'quality')

service = ctrl.Antecedent(np.arange(0, 11, 1), 'service')

tip = ctrl.Consequent(np.arange(0, 26, 1), 'tip')

quality.automf(3)

service.automf(3)

tip['low'] = fuzz.trimf(tip.universe, [0, 0, 13])

tip['medium'] = fuzz.trimf(tip.universe, [0, 13, 25])

tip['high'] = fuzz.trimf(tip.universe, [13, 25, 25])

quality['average'].view()

service.view()

tip.view()

rule1 = ctrl.Rule(quality['poor'] | service['poor'], tip['low'])

rule2 = ctrl.Rule(service['average'], tip['medium'])

rule3 = ctrl.Rule(service['good'] | quality['good'], tip['high'])

rule1.view()

tipping\_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])

tipping = ctrl.ControlSystemSimulation(tipping\_ctrl)

# Pass inputs to the ControlSystem using Antecedent labels with Pythonic API

tipping.input['quality'] = 6.5

tipping.input['service'] = 9.8

# Crunch the numbers

tipping.compute()

print(tipping.output['tip'])

tip.view(sim=tipping)

**Output:**

