3 INPUT AND 3 WEIGHTS

```
n = int(input("Enter the number of input neurons: "))
# w will take weight & x will take the input
w = [ ]
x = [ ]

# taking the value of input and their weight
for i in range(0,n):
    a = float(input("Enter the input: "))
    x.append(a)
    b = float(input("Enter the weight: "))
    w.append(b)

print("The given weights are: ")
print(w)
print("The given input are: " )
print(x)

y = 0.0
for i in range(0,n):
    y = y + (w[i]*x[i])

print("The net input is ")
print (round(y,3))
```

BINARY AND SIGMOIDAL FUNCTION

```
binary = 1/(1+ (math.exp(-y)))

print("The output after applying binary sigmoidal function activation ")

print (round(binary, 3))
```

```
# Applying Bipolar Sigmoidal function on the net input i.e  y

bipolar = -1+(2/(1+ (math.exp(-y))))
print("The output after applying bipolar sigmoidal function activation ")
print(round(bipolar, 3))
```

MP NEURON

```
n = int(input("Enter number of elements : "))
print("Enter the inputs")
inputs = []
for i in range(0, n):
   ele = float(input())
   inputs.append(ele) # adding the element
print(inputs)
print("Enter the weights")
weights = []
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))
```

HEBB S RULE

```
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("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)
```

DELTA RULE

```
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)
```

BACK PROPOGATION ALGORITHM

```
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("v1=",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)
```

ERROR BACK PROPOGATION

```
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)
```

LINE SEPARATION

```
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()
```

FUZZY LOGIC

```
from fuzzywuzzy import fuzz
from fuzzywuzzy import process

s1 = "I love GeeksforGeeks"
s2 = "I am loving GeeksforGeeks"
print("FuzzyWuzzy Ratio: ", fuzz.ratio( s1, s2 ))
print("FuzzyWuzzy PartialRatio: ", fuzz.partial_ratio( s1, s2 ))
print("FuzzyWuzzy TokenSortRatio: ", fuzz.token_sort_ratio( s1, s2 ))
print("FuzzyWuzzy TokenSetRatio: ", fuzz.token_set_ratio( s1, s2 ))
print("FuzzyWuzzy WRatio: ", fuzz.WRatio( s1, s2 ), '\n\n')

# for process library,
query = 'geeks for geeks'
choices = ['geek for geek', 'geek geek', 'g. for geeks']
print("List of ratios: ")
print(process.extract( query, choices ), '\n')
print("Best among the above list: ", process.extractOne( query, choices
))
```

KOM

```
!pip install minisom
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())
```