SC LABS

1. LAB-1 Max-min, min-dot, union, intersection

2. LAB-2 Fuzzy Membership Fns(heights)

3. LAB-3 Trapezoid Membeship and defuzzification

4. LAB-4 Activation Functions

5. LAB-5 MP Neuron, XOR, mamdhani

6. LAB-6 Hebb Rule

7. LAB-7 Perceptron

8. LAB-8 Feed_Forward_Networks,BPN

9. LAB-9 Auto_Associtive_Network(SOM)

SC_LAB-1

September 12, 2022

1 Pre-Lab

1.define crisp set and fuzzy set

Ans)Crisp set: a set defined using a character function that assigns a value of either 0 or 1 to each element of universe. Fuzzy set: a set having degrees of membership between 1 and 0.

2.Describe the importance of fuzzy sets and it's implications in engineering sector

Ans)It is useful to describe situations in which the data are imprecise or vague. the role of fuzzy sets in engineering has grown in last two decades.THe fuzzy theory indicates the potential application in solvig problems and gives methodical flwxibility to engineers on calculations.

3.why excluded middle law doesnot get satisfied in fuzzy logic:

Ans)The principle of middle law is not accepted as a valid axiom in theory of fuzzt sets because it does not apply to situations in which one deals with clases which do not have sharply defined boundaries.

4) Consider The fuzzy sets:

$$A= \{\frac{1}{2.0} + \frac{0.65}{4.0} + \frac{0.5}{6.0} + \frac{0.35}{8.0} + \frac{0}{10.0}\}$$

$$B= \{\frac{0}{2.0} + \frac{0.35}{4.0} + \frac{0.5}{6.0} + \frac{0.65}{8.0} + \frac{1}{10.0}\}$$

Find: a. Union of A&B b. Intersection of A&B c. Compliment of A d. Compliment of B

Ans) a)Union: C(x) = A B (x) = A(x) B(x) = max(A(x), B(x)), x X

i.e., A B = {1/2.0 + 0.65/40 + 0.5/6.0 + 0.65/8.0 + 1/10.0}

b)Intersection: C(x) = A B (x) = A(x) B(x) = min(A(x), B(x)), x X

i.e., A B = {0/2.0 + 0.35/4.0 + 0.5/6.0 + 0.35/8.0 +0/10.0}

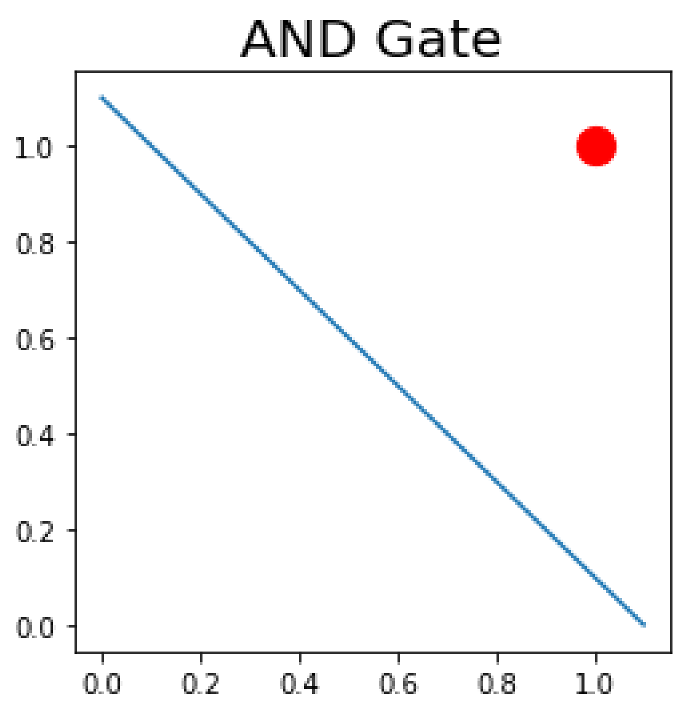
c)Compliment of A: A^c (x) = 1 − A(x)

i.e., {0/2.0 + 0.35/4.0 + 0.5/6.0 + 0.65/8.0 + 1/10.0}

d) Compliment of B is: {1/2.0 + 0.65/4.0 + 0.5/6.0 + 0.35/8.0 + 0/10.0}

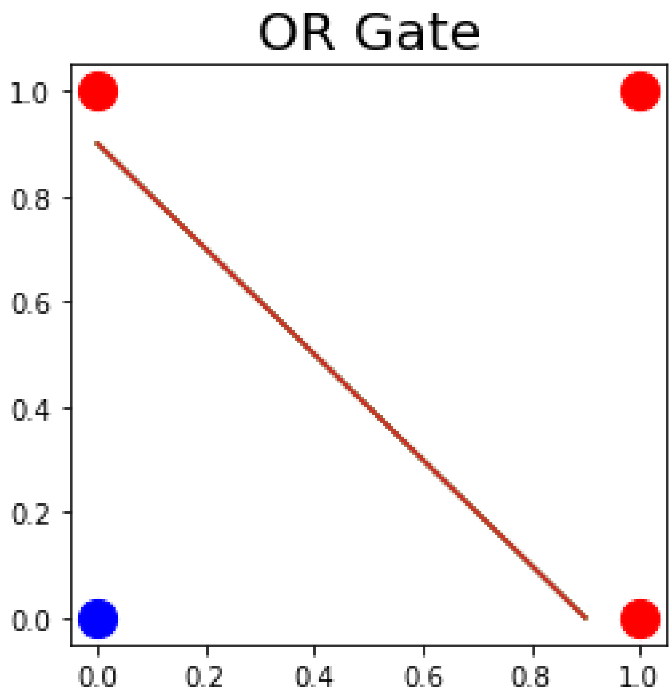
2 logical gates implementation

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
x=np.array([[0,0],[0,1],[1,0],[1,1]])
y=[0,0,0,1]
area = 200
fig = plt.figure(figsize=(4,4))
plt.title('AND Gate', fontsize=20)
# color red: is class 1 and color blue is class 0.
for i in range(4):
    if y[i]==1:
            co='r'
    else:
          co='b'
plt.scatter(x[i][0],x[i][1],s=area,c=co)
plt.plot([0,1.1],[1.1,0])
plt.show()
```



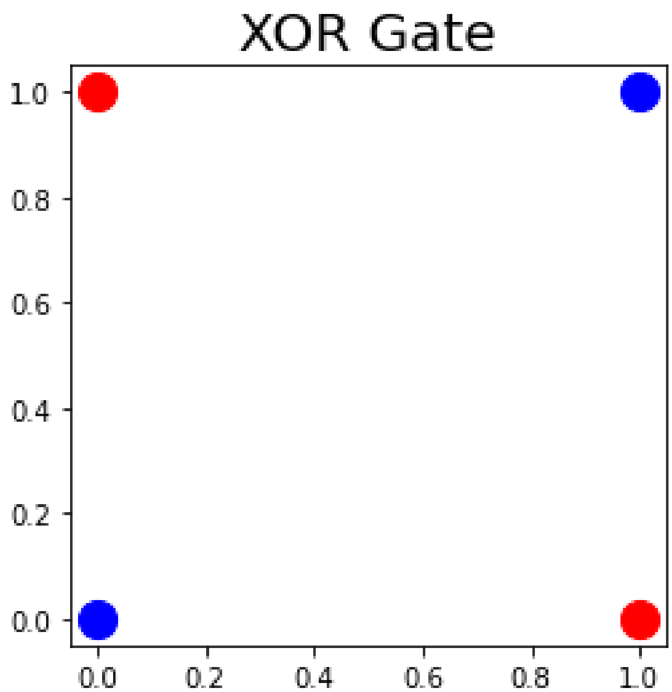
[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
x=np.array([[0,0],[0,1],[1,0],[1,1]])
```

```
y=[0,1,1,1]
area = 200
fig = plt.figure(figsize=(4,4))
plt.title('OR Gate', fontsize=20)
# color red: is class 1 and color blue is class 0.
for i in range(4):
    if y[i]==1:
        co='r'
    else:
        co='b'
    plt.scatter(x[i][0],x[i][1],s=area,c=co)
    plt.plot([0,0.9],[0.9,0])
plt.show()
```

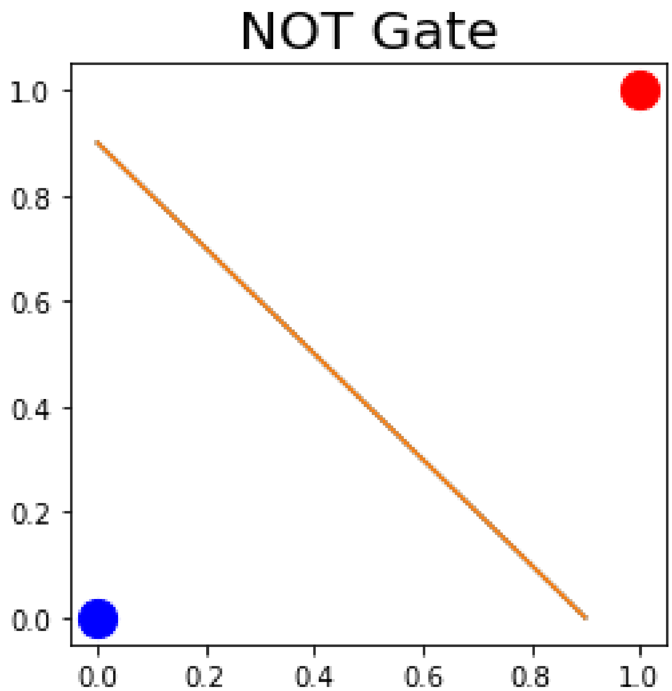


[3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
x=np.array([[0,0],[0,1],[1,0],[1,1]])
y=[0,1,1,0]
area = 200
fig = plt.figure(figsize=(4,4))
plt.title('XOR Gate', fontsize=20)
# color red: is class 1 and color blue is class 0.
for i in range(4):
```

```
    if y[i]==1:
        co='r'
    else:
        co='b'
    plt.scatter(x[i][0],x[i][1],s=area,c=co)
plt.show()
```



[4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
x=np.array([[0,0],[1,1]])
y=[0,1]
area = 200
fig = plt.figure(figsize=(4,4))
plt.title('NOT Gate', fontsize=20)
# color red: is class 1 and color blue is class 0.
for i in range(2):
    if y[i]==1:
        co='r'
    else:
        co='b'
    plt.scatter(x[i][0],x[i][1],s=area,c=co)
    plt.plot([0,0.9],[0.9,0])
plt.show()
```



3 In-Lab

[6]:
```
#inlab
import numpy as np

# Max-Min Composition given by Zadeh
def maxMin(x, y):
    z = []
    for x1 in x:
        for y1 in y.T:
            z.append(max(np.minimum(x1, y1)))
    return np.array(z).reshape((x.shape[0], y.shape[1]))

# Max-Product Composition given by Rosenfeld
def maxProduct(x, y):
    z = []
    for x1 in x:
        for y1 in y.T:
            z.append(max(np.multiply(x1, y1)))
    return np.array(z).reshape((x.shape[0], y.shape[1]))

# 3 arrays for the example
r1 = np.array([[0.6, 0.6, 0.8, 0.9],
```

```
       [0.1, 0.2, 0.9,0.8],
       [0.9, 0.3, 0.4,0.8],
       [0.9,0.8,0.1,0.2]])
r2 = np.array([[0.1, 0.2, 0.7, 0.9],
       [1.0, 1.0, 0.4, 0.6],
       [0, 0, 0.5, 0.9],
       [0.9, 1.0, 0.8, 0.2]])

print ("R1oR2 => Max-Min :\n" + str(maxMin(r1, r2)) + "\n")
print ("R1oR2 => Max-Product :\n" + str(maxProduct(r1, r2)) + "\n\n")
```

```
R1oR2 => Max-Min :
[[0.9 0.9 0.8 0.8]
 [0.8 0.8 0.8 0.9]
 [0.8 0.8 0.8 0.9]
 [0.8 0.8 0.7 0.9]]

R1oR2 => Max-Product :
[[0.81 0.9  0.72 0.72]
 [0.72 0.8  0.64 0.81]
 [0.72 0.8  0.64 0.81]
 [0.8  0.8  0.63 0.81]]
```

4 Post-Lab

[7]:
```
# post lab
arr1 = np.array([1,4,5,7])
arr2 = np.array([4,7,9,10])
arr3 = np.array([[3,5,7,8]])
print("union is ", np.union1d(arr1,arr2))
print("union of 2d and 1d",np.union1d(arr1,arr3))
```

```
union is  [ 1  4  5  7  9 10]
union of 2d and 1d [1 3 4 5 7 8]
```

[8]:
```
# code to find union of more than two arrays
# import libraries
import numpy as np
from functools import reduce


array = reduce(np.union1d, ([1, 2, 3], [1, 3, 5],
                             [2, 4, 6], [0, 0, 0]))
print("Union ", array)
```

```
Union  [0 1 2 3 4 5 6]
```

[9]:
```python
# intersection
print("intersection : ", np.intersect1d(arr1,arr2))
```

```
intersection :  [4 7]
```

5 Other Method

[10]:
```python
import numpy as np
import sys
```

[11]:
```python
'''
r = []
r_rows = int(input("Enter the number of rows of r: "))
r_col = int(input("Enter the number of cols of r: "))

for i in range(0, r_rows):

    r.append([float(i) for i in input().split()])
s = []
s_rows = int(input("Enter the number of rows of s: "))
s_col = int(input("Enter the number of cols of s: "))
for i in range(0, s_rows):

    s.append([float(i) for i in input().split()])
'''
```

[11]:
```
'\nr = []\nr_rows = int(input("Enter the number of rows of r: "))\nr_col =
int(input("Enter the number of cols of r: "))\n\nfor i in range(0, r_rows):\n
\n    r.append([float(i) for i in input().split()])\ns = []\ns_rows =
int(input("Enter the number of rows of s: "))\ns_col = int(input("Enter the
number of cols of s: "))\nfor i in range(0, s_rows):\n    \n
s.append([float(i) for i in input().split()])\n'
```

[12]:
```python
### static input for testing
r = [
        [0.3, 0.5, 0.8],
        [0, 0.7, 1],
        [0.4, 0.6, 0.5]
    ]
s = [
        [0.9, 0.5, 0.7, 0.7],
        [0.3, 0.2, 0, 0.9],
        [1, 0, 0.5, 0.5]
    ]
print(r, s)
```

```
[[0.3, 0.5, 0.8], [0, 0.7, 1], [0.4, 0.6, 0.5]] [[0.9, 0.5, 0.7, 0.7], [0.3,
0.2, 0, 0.9], [1, 0, 0.5, 0.5]]
```

[13]:
```python
def maxmin(r, s):

    if(len(r[0]) != len(s)):
        return "NOT VALID"

    result = np.zeros((len(r), len(s[0])))
    for i in range(len(r)):

        for j in range(len(s[0])):

            current_max = -sys.maxsize
            for k in range(len(s)):
               current_max = max(current_max, min(r[i][k], s[k][j]))

            result[i][j] = current_max

    return (result)


'''
def maxmin(r, s):

    if(len(r[0]) != len(s))
        return "NOT VALID"
    result = np.zeros((len(r), len(s[0])))
    for i in range(len(r)):

        for j in range(len(s[0])):

            min_list = []
            for k in range(len(s)):
               min_list.append(min(r[i][k], s[k][j]))

            result[i][j] = max(min_list)
            min_list.clear()
    return (result)
maxmin(r, s)
'''
```

[13]:
```
'\ndef maxmin(r, s):\n    \n    if(len(r[0]) != len(s))\n        return "NOT
VALID"\n    result = np.zeros((len(r), len(s[0])))\n    for i in
range(len(r)):\n \n        for j in range(len(s[0])):\n            \n
min_list = []\n            for k in range(len(s)):\n
min_list.append(min(r[i][k], s[k][j]))\n\n            result[i][j] =
max(min_list)\n            min_list.clear()\n    return (result)\nmaxmin(r,
s)\n'
```

[14]:
```
def maxprod(r, s):

    if(len(r[0]) != len(s)):
        return "NOT VALID"

    result = np.zeros((len(r), len(s[0])))
    for i in range(len(r)):

        for j in range(len(s[0])):

            current_max = -sys.maxsize
            for k in range(len(s)):
               current_max = max(current_max, (r[i][k] * s[k][j]))

            result[i][j] = current_max

    return (result)
```

[15]:
```
maxmin_result = maxmin(r, s)
maxprod_result = maxprod(r, s)

print("MAX MIN RESULT\n\n", maxmin_result, end="\n\n")
print("MAX PRODUCT RESULT\n\n", maxprod_result, end="\n\n")
```

```
MAX MIN RESULT

 [[0.8 0.3 0.5 0.5]
 [1.  0.2 0.5 0.7]
 [0.5 0.4 0.5 0.6]]

MAX PRODUCT RESULT

 [[0.8  0.15 0.4  0.45]
 [1.   0.14 0.5  0.63]
 [0.5  0.2  0.28 0.54]]
```

SC_LAB-2

September 12, 2022

1 Pre-Lab

1. what is a membership function of a fuzzy set ?

Ans) it is a curve that defines how each point in the input space is mapped to a membership value between 0 & 1.

2.How rank ordering used to define membership functions based in polling ?

Ans)This methodology can be adapted to assign membership values to fuzzy variabble.Pairwise comparisons enable us to determine prefrences and this results in determining the order of membership.

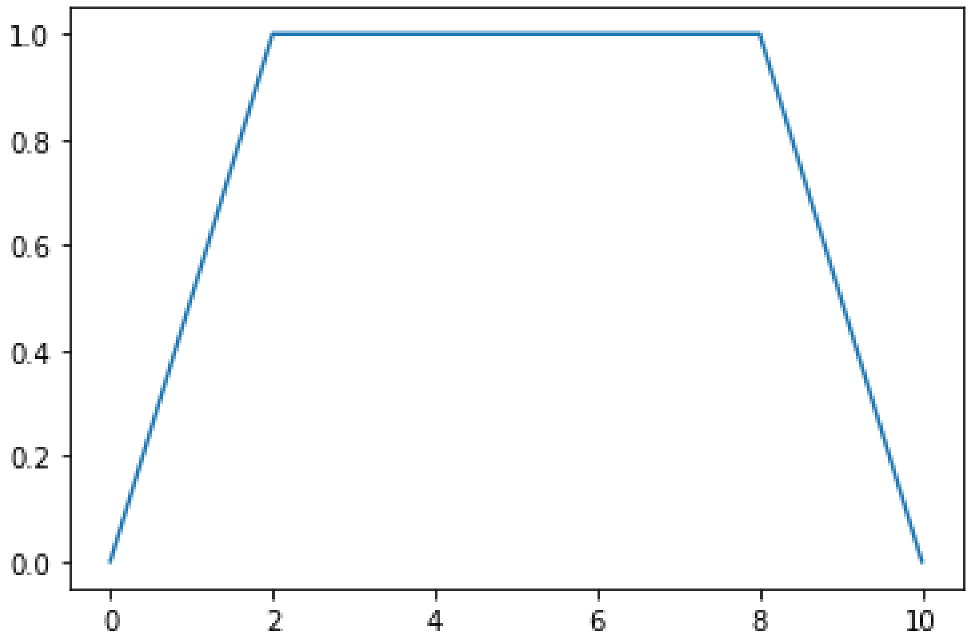
3.List various methods employed for the membership value assignment.

Ans) * Intuition * Inference * Rank Ordering * Angular Fuzzy Sets * Neural Networks * Genetic Algorithm * Inductive Reasoning

4. Write a python code using inference approach.plot the trapezoidal membership function an mention the methods of inference role

[ ]:
```
import numpy as np
import skfuzzy as fuzz
import matplotlib.pyplot as plt


start = 0
stop = 10 + 0.001
step = 0.25
x = np.arange(start, stop, step)
trapmf = fuzz.trapmf(x, [0, 2, 8, 10])
```

[ ]:
```
plt.plot(x, trapmf, label="Trapezoidal")
plt.show()
```



2 In-Lab

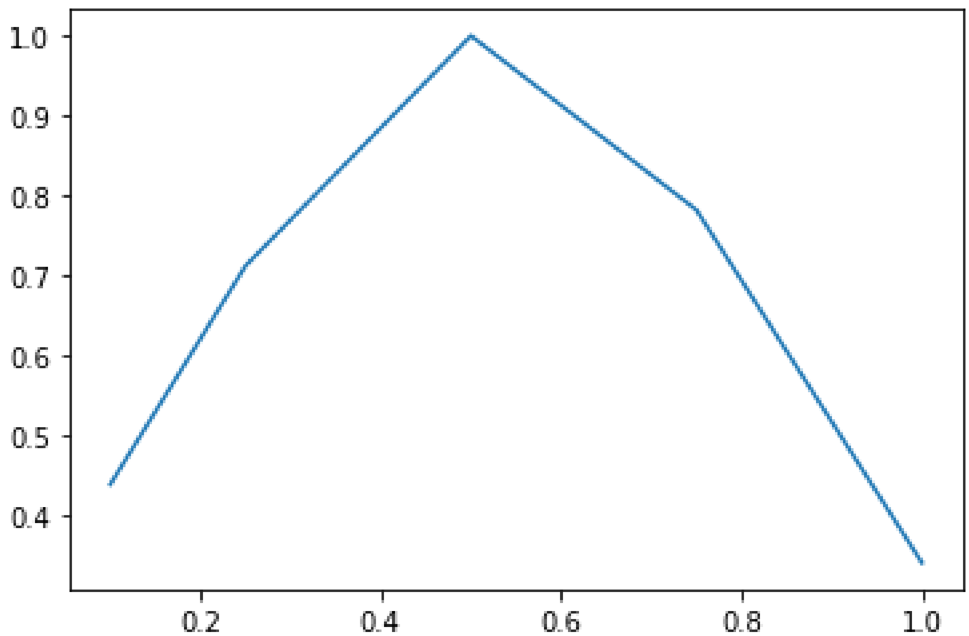
[ ]:
```
'''
Using the inference approach, plot the fuzzy membership functions to the␣
↪following
variables (Linguistic variable) Height of the people:
i)very tall, ii) tall, iii) normal, iv) short, v) very short

Write a python program to implement the inference approach such that the fuzzy
membership functions can be plotted and also display the graph as an output.␣
↪Also
compute the values manually and check out the output with the implementation␣
↪output. '''
```

[ ]:
```
'\nUsing the inference approach, plot the fuzzy membership functions to the
following\nvariables (Linguistic variable) Height of the people:\ni)very tall,
ii) tall, iii) normal, iv) short, v) very short\n\nWrite a python program to
implement the inference approach such that the fuzzy\nmembership functions can
be plotted and also display the graph as an output. Also\ncompute the values
manually and check out the output with the implementation output. '
```

[ ]:
```
#1
# we have to assume the extreme lingusitc variables as 0,1 and mid value as␣
↪normal
arr = np.array([0.1,0.25,0.5,0.75,1])
```

```
print(arr)
srtd = np.std(arr)
mean = np.mean(arr)
print(mean)
print(srtd)
c = fuzz.gaussmf(arr,mean,srtd)
plt.plot(arr,c)
plt.show()
```

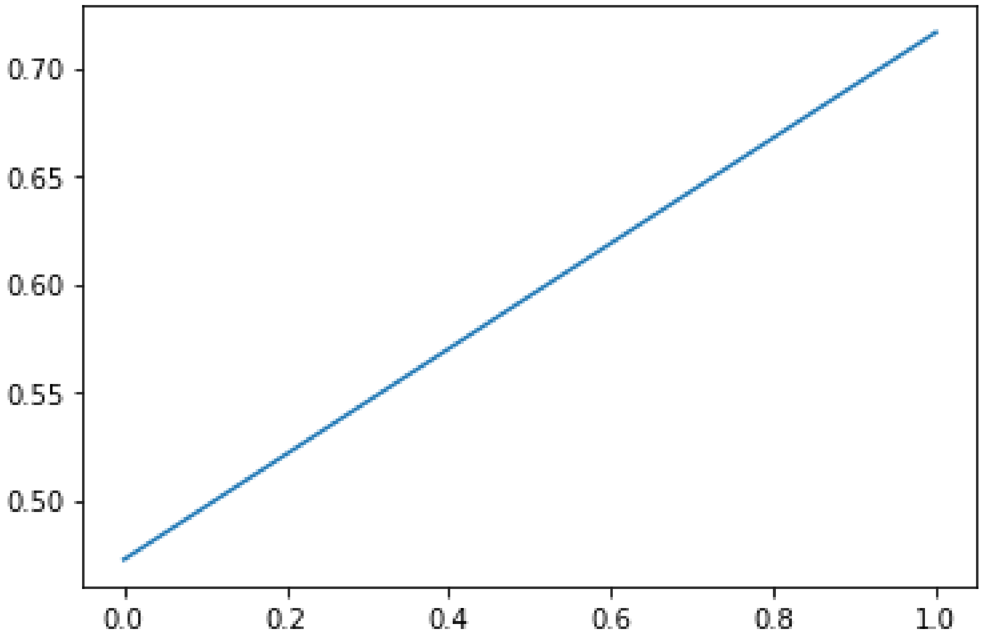
```
[0.1  0.25 0.5  0.75 1.  ]
0.52
0.32649655434629016
```



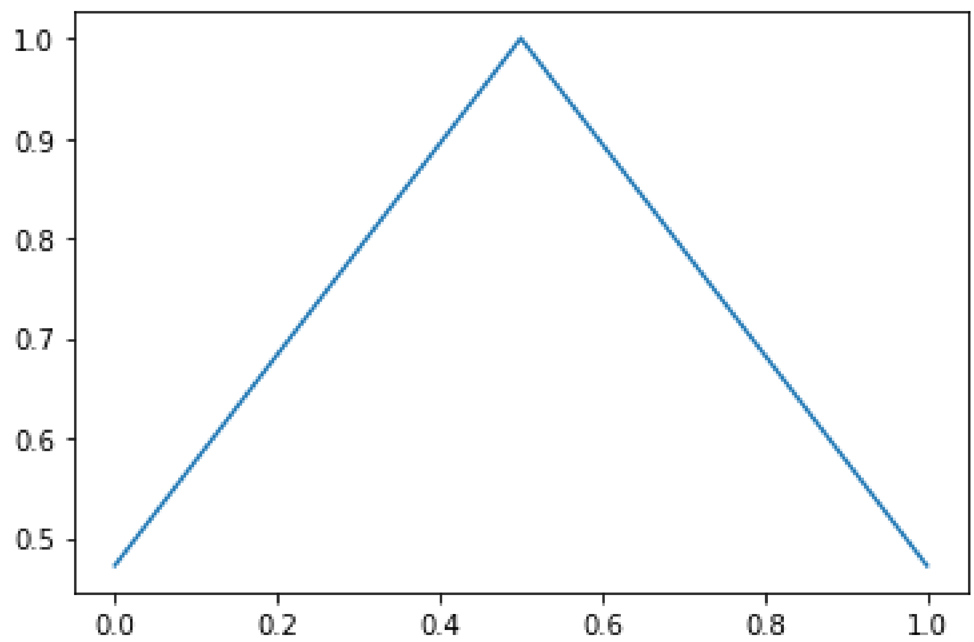
[ ]:
```
# defuzzication
alpha = 0.5
for i in range(len(arr)):
  if arr[i] < alpha:
    arr[i] = 0
  else:
    arr[i] = 1
srtd = np.std(arr)
mean = np.mean(arr)
print(mean)
print(srtd)
c = fuzz.gaussmf(arr,mean,srtd)
```

```
plt.plot(arr,c)
plt.show()
```

```
0.6
0.48989794855663565
```



[ ]:
```
# 2 . by given linguistic constraints we consider the values as 0,0,5,1␣
↪respectively
a = np.array([0,0.5,1])
trimf =fuzz.gaussmf(a,np.mean(a),np.std(a))
plt.plot(a,trimf)
plt.show()
```

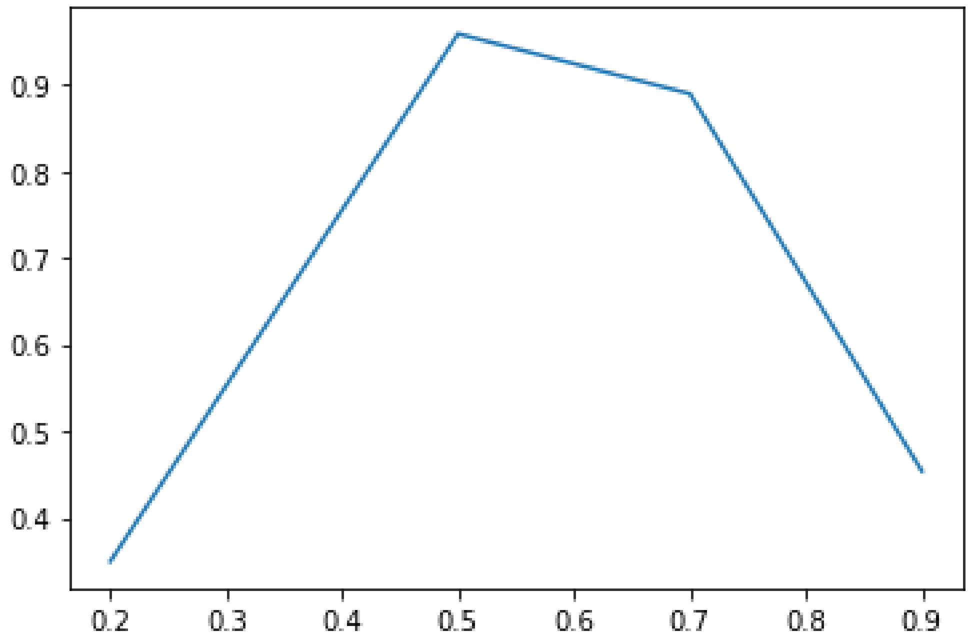


3 Post-Lab

[ ]:
```
"""
Write a python code to implement Gaussian method by plotting the membership␣
↪functions to the
variables of temperatures with:
a) Warm
b) Moderate
c) Cold
d) Hot
"""
```

[ ]:
```
'\nWrite a python code to implement Gaussian method by plotting the membership
functions to the \nvariables of temperatures with:\na) Warm\nb) Moderate\nc)
Cold\nd) Hot\n'
```

[ ]:
```
# let the values be [0.2,0.5,0.7,0.9]
arr = np.array([0.2,0.5,0.7,0.9])
mf = fuzz.gaussmf(arr,np.mean(arr),np.std(arr))
plt.plot(arr,mf)
plt.show()
```



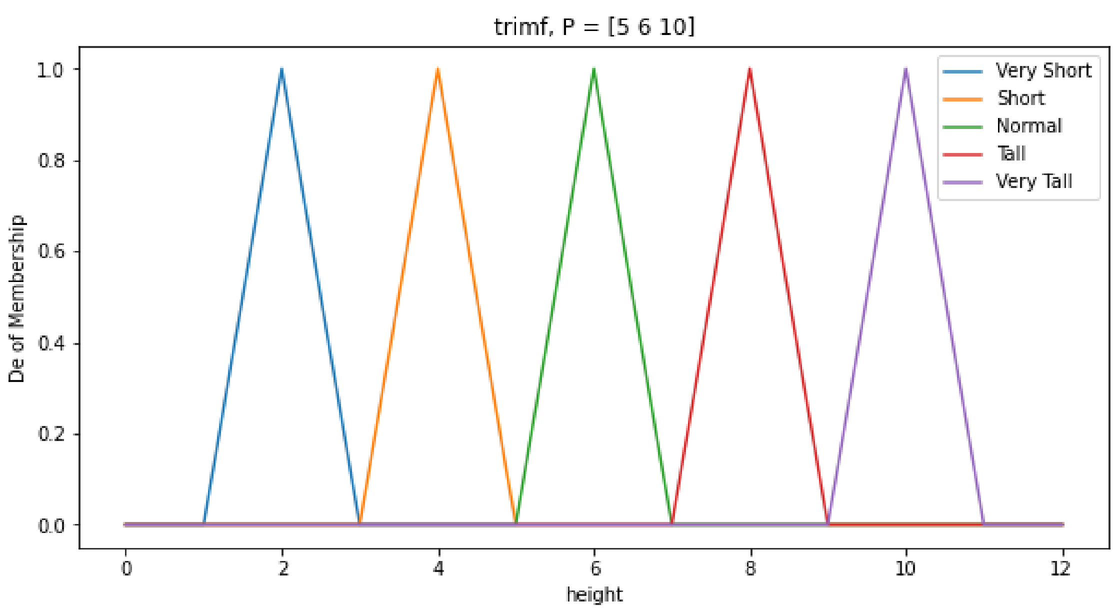
4 Method 2

[ ]:
```
import skfuzzy as fuzz
import numpy as np
x = np.arange(13)
v_s=fuzz.trimf(x,[1,2,3])
s = fuzz.trimf(x,[3,4,5])
n = fuzz.trimf(x,[5,6,7])
t = fuzz.trimf(x,[7,8,9])
v_t = fuzz.trimf(x,[9,10,11])
print(v_s, s, n, t, v_t)


fuzz.trimf?
```

```
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.] [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0.] [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.] [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
0. 0.] [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
Signature:
fuzz.trimf(x,
abc)
Docstring:
Triangular membership function generator.

Parameters
```

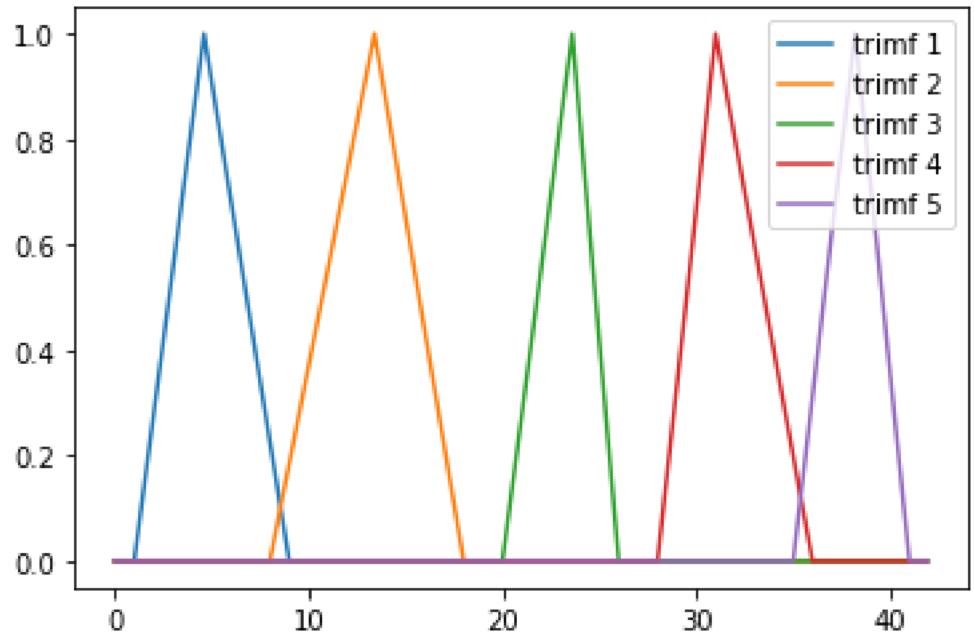
```
----------
x : 1d array
    Independent variable.
abc : 1d array, length 3
    Three-element vector controlling shape of triangular function.
    Requires a <= b <= c.

Returns
-------
y : 1d array
    Triangular membership function.
File:
c:\users\yash\appdata\local\programs\python\python39\lib\site-
packages\skfuzzy\membership\generatemf.py
Type:      function
```

[ ]:
```python
plt.figure(figsize=(10, 5))
plt.plot(x, v_s, label="Very Short")
plt.plot(x, s, label="Short")
plt.plot(x, n, label="Normal")
plt.plot(x, t, label="Tall")
plt.plot(x, v_t, label="Very Tall")
plt.title('trimf, P = [5 6 10]')
plt.xlabel('height')
plt.ylabel('De of Membership')
plt.legend()
plt.show()
```



[ ]:

```python
#Only using Numpy
def get_abc(data):
    abc = []
    for i in data:
        abc.append([np.min(i),np.mean(i),np.max(i)])
    return abc,(data[0][0]-1,data[-1][-1]+1)

import numpy as np
import matplotlib.colors as mcolors
number_divisons = int(input())
data = []
colors = list(mcolors.BASE_COLORS)[:number_divisons]
for i in range(number_divisons):
    data.append([int(num) for num in input().split()])
data = np.array(data)
abc,x = get_abc(data)

def plot(abc,x):
    x = np.arange(x[0],x[1],0.1)
    for i in range(len(abc)):
        plt.plot(x,fuzz.trimf(x,abc[i]),label = 'trimf '+str(i+1))
    plt.legend()
    plt.show()

plot(abc,x)

#trimf 1 => Very Short
#trimf 2 => Short
#trimf 3 => Normal
#trimf 4 => Tall
#trimf 5 => Very Tall

#Input:
'''
5
1 2 3 8 9
8 10 15 16 18
20 24 25 23 26
28 29 30 32 36
35 38 37 40 41
'''
```



[ ]: '\n5\n1 2 3 8 9\n8 10 15 16 18\n20 24 25 23 26\n28 29 30 32 36\n35 38 37 40 41\n'

5 Method 3

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[ ]:
```
def triangularMF(x, ranges):

    y = np.zeros(len(x))
    a, b, c = map(lambda x: x, ranges)
    # Left side
    if a != b:
        '''
            For the first half of the triangle i.e in range [a, b]
            For every x value in the range apply (x-a) / (b-a)

        '''
        index = np.nonzero(np.logical_and(a < x, x < b))[0]
        y[index] = (x[index] - a) / float(b - a)

    # Right side
    if b != c:
        '''
```

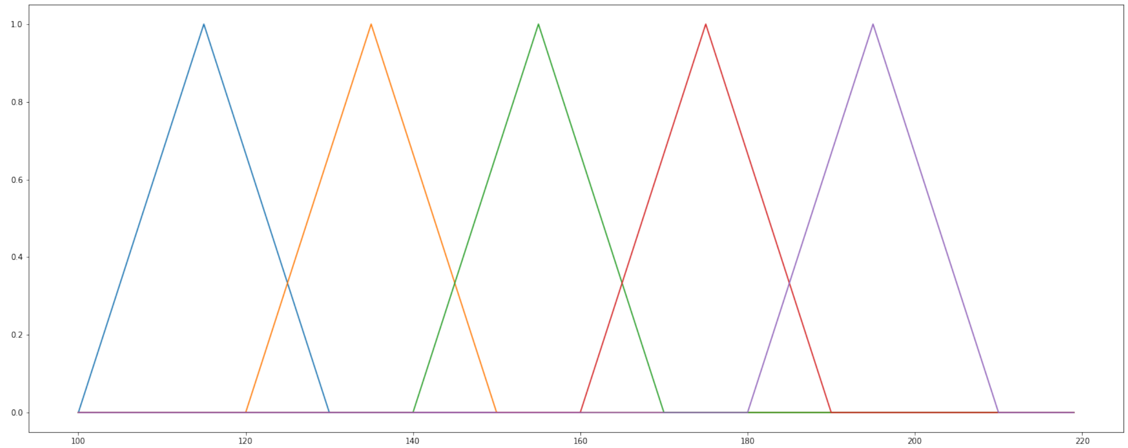
```
            For the second half of the triangle i.e in range [b, c]
            For every x value in the range apply (c-x) / (c-b)

        '''
        index = np.nonzero(np.logical_and(b < x, x < c))[0]
        y[index] = (c - x[index]) / float(c - b)

    index = np.nonzero(x == b)
    '''Change the membership value at b to 1 (i.e. peak value at b)'''
    y[index] = 1

    return y
```

[ ]:
```
"""
def triangularMF(x, a, b, c):

    arr = np.zeros(np.size(x))
    arr[b-x[0]] = 1
    return arr
"""

# Membership for any value of x
# Given ranges
heights = {
    'very_short': (100, 115, 130),
    'short' : (120, 135, 150),
    'medium' : (140, 155, 170),
    'tall' : (160, 175, 190),
    'very_tall' : (180, 195, 210)
}
```

[ ]:
```
x = np.arange(100, 220)

very_short = triangularMF(x, heights['very_short'])

short = triangularMF(x, heights['short'])

medium = triangularMF(x, heights['medium'])

tall = triangularMF(x, heights['tall'])

very_tall = triangularMF(x, heights['very_tall'])
```

[ ]:
```
plt.figure(figsize=(25, 10))
plt.plot(x, very_short)
plt.plot(x, short)
plt.plot(x, medium)
```

```
plt.plot(x, tall)
plt.plot(x, very_tall)
plt.show()
```



[ ]:
```
def getMembership(x, heights):

    '''
            For first half of the triangle
                (x - a) / float(b - a)

            For second half of the triangle
                (c - x) / float(c - b)
    '''
    height_membership = {}
    for height in heights.keys():

        a, b, c = map(lambda x: x, heights[height])

        if(x < a):

            height_membership[height] = 0
            continue
        elif(x > c):
            height_membership[height] = 0
            continue



        isFirstHalf = np.logical_and(a < x, x < b)

        if(isFirstHalf):
```

```python
            height_membership[height] = {(x - a) / float(b - a)}
        else:
            height_membership[height] = {(c - x) / float(c - b)}

    return height_membership
```

[ ]:
```python
membership = getMembership(142, heights)
print(membership)
```

```
{'very_short': 0, 'short': {0.5333333333333333}, 'medium':
{0.13333333333333333}, 'tall': 0, 'very_tall': 0}
```

SC_LAB-3

September 12, 2022

1 Pre-Lab

1.What are the different methods of defuzzification process? Ans) * max membership principle. * centroid method. * weighted average method. * mean max membership. * center of sums. * centre of largest area. * first of maxima, last of maxima.

2.Explain the methods employed for converting fuzzy into crisp Ans)All the methods above can be used for converting fuzzy to crisp , one of the most used method is alpha-cut, in this methodology we seggregate the values using the parameter alpha , it returns either 1 or 0(crisp value) for a given fuzzy value.

3.State the necessity of defuzzification process Ans)Defuzzification helps us to know the crisp value for a fuzzy set such that we can easily identify the outcone in decision making algorithms.

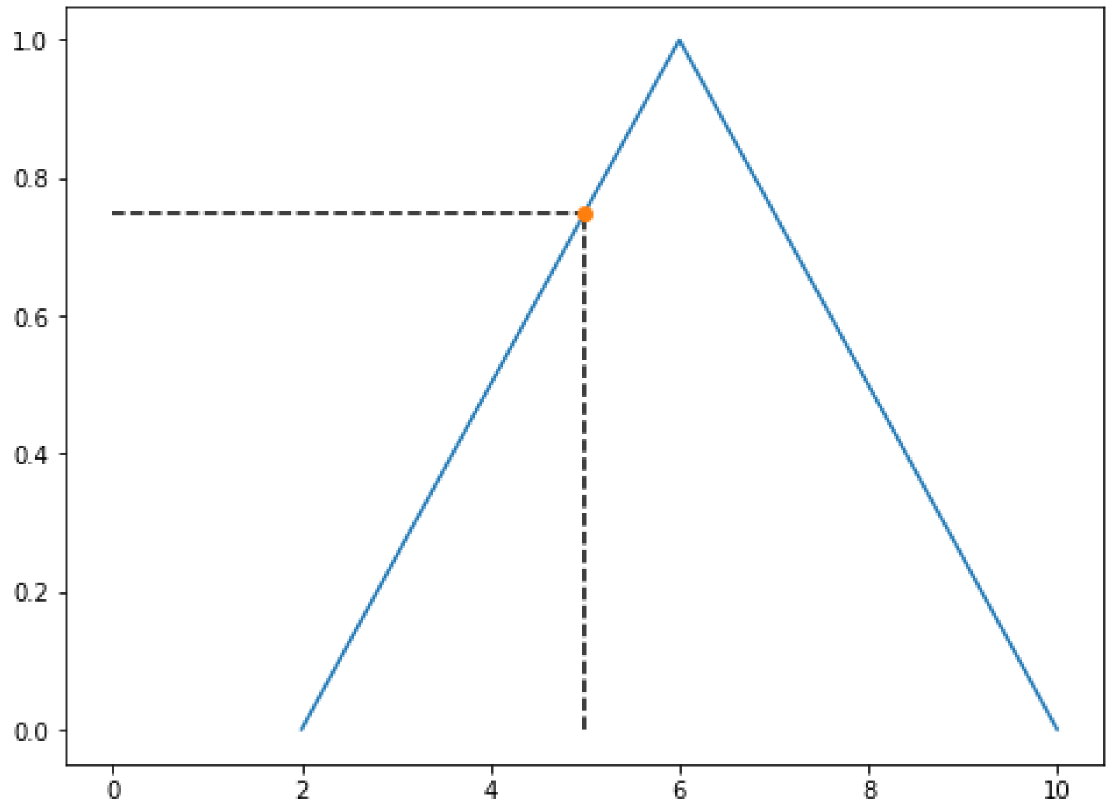
4. Write a python code to implement fuzzification using triangular membership function and then defuzzify by using the centroid. (this solution is implemented in next cell)

[ ]:
```
#triangluar membership function
#preinputs a, b, c
#consider
import skfuzzy as fuzz
import numpy as np
a = 2
b = 6
c = 10
#x can be any value
x = int(input())
ans = 0
if x >= a and x < b:
  ans = (x - a) / (b - a)
elif x >= b and x <= c:
  ans = (c - x) / (c - b)
print(ans)
```

```
5
0.75
```

[ ]:
```
#plot for triangular function
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(8,6))
```

```
plt.plot([a, b, c], [0, 1, 0])
plt.plot([x], [ans], marker="o")
plt.vlines(x, 0, ans, linestyles="dashed")
plt.hlines(ans, 0, x, linestyles="dashed")
plt.show()
```



[ ]:
```
#defuzzification using centroid
defuzz = (x * ans) / ans
defuzz
```

[ ]: 5.0

2 INLAB

[ ]:
```
"""
Write a python code to fuzzify using a trapezoid membership function and then␣
↪defuzzify by using,
a. by bisector technique.
b. mean of the greatest strategy.
"""
```

```python
#trapezoidal fuzzification
a = 2
b = 4
c = 8
d = 10
x = int(input())
ans = 0
if x <= a:
  ans = 0
elif x >= a and x < b:
  ans = (x - a) / (b - a)
elif x >= b and x < c:
  ans = 1
elif x >= c and  x <= d:
  ans = (d - x) / (d - c)
else:
  ans = 0

print(ans)
```

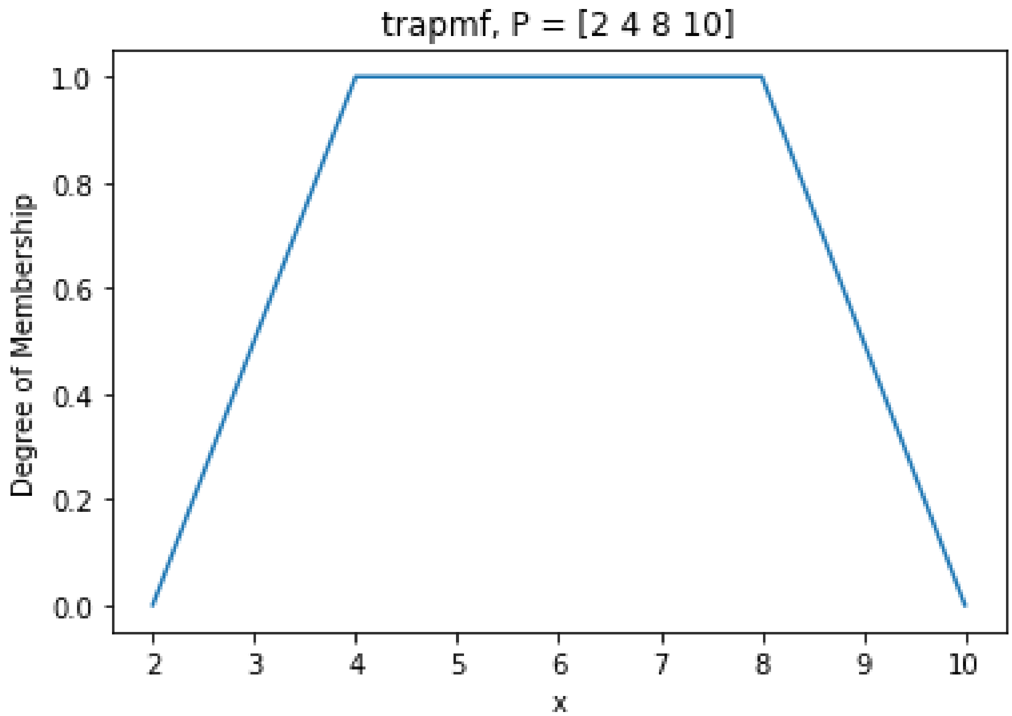
```
3
0.5
```

[ ]:
```python
X = np.array([2, 5, 4])
Y = fuzz.trapmf(X, np.array([2, 4, 8, 10]));
Y
```

[ ]:
```
array([0., 1., 1.])
```

[ ]:
```python
plt.plot([2, 4, 8, 10], [0, 1, 1, 0])
plt.title('trapmf, P = [2 4 8 10]')
plt.xlabel('x')
plt.ylabel('Degree of Membership')
```

[ ]:
```
Text(0, 0.5, 'Degree of Membership')
```



[ ]:
```
#defuzzify using bisector technique
defuzz_bs = fuzz.defuzz(np.array([0.1, 0.5, 0.75]), Y, 'bisector')
defuzz_bs
```

[ ]: 0.525

[ ]:
```
#mean of greatest strategy or mean of maximum
defuzz_mom = fuzz.defuzz(np.array([0.1, 0.5, 0.75]), Y, 'mom')
defuzz_mom
```

[ ]: 0.625

3 Post Lab

[ ]:
```
"""
Write a python code to implement fuzzification using Gaussian membership␣
 ↪function and then
defuzzify
"""

#gaussian membership function
m = 10
sig = 3
```

```
x = int(input())
sig_mem = (2.718281828459045) ** ((-1 / 2) * ((x - m) / sig) ** 2)
sig_mem
```

6

[ ]: 0.41111229050718745

[ ]:
```
#or
sig_mf = fuzz.sigmf(X, m, sig)
sig_mf
```

[ ]: array([3.77513454e-11, 3.05902227e-07, 1.52299795e-08])

[ ]:
```
#defuzzify
defuzz_ct = fuzz.defuzz(np.array([3.77513454e-11, 3.05902227e-07, 1.
 ↪52299795e-08]), sig_mf, 'centroid')
defuzz_ct
```

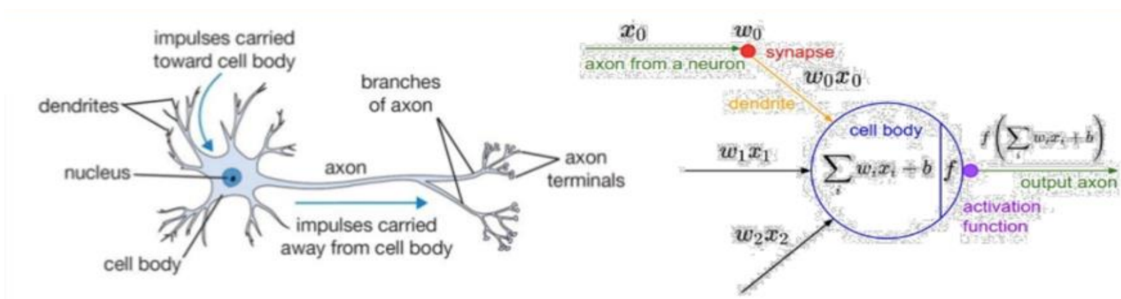
[ ]: 5.3031800314722196e-09

SC_LAB-4

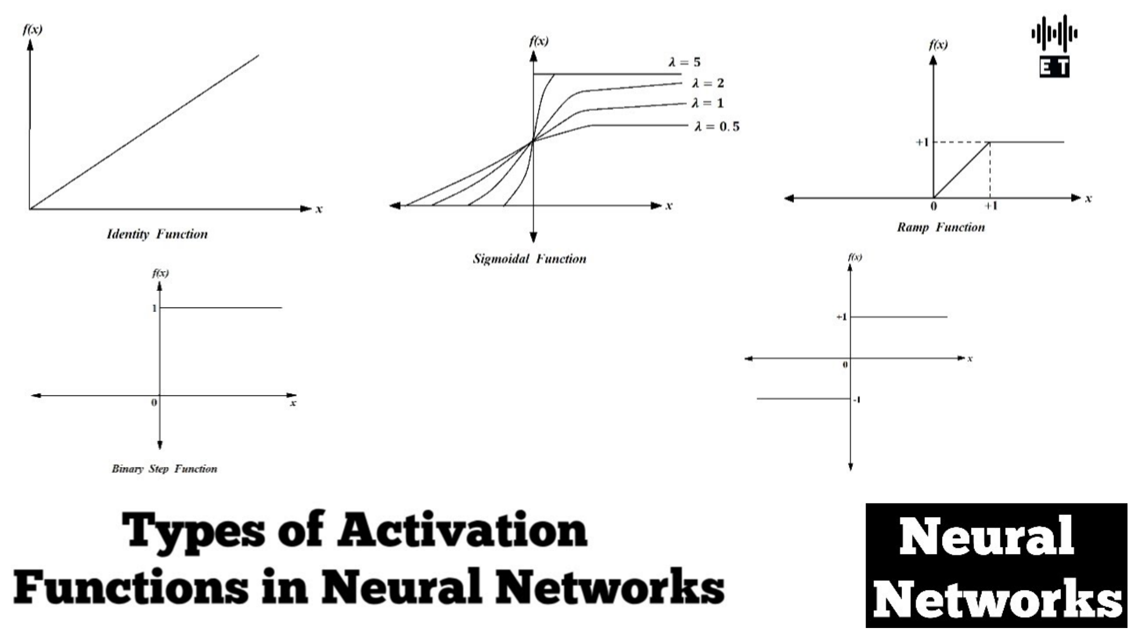
September 12, 2022

1 Pre-Lab

1)Biological Neuron 2,3) Biological Neuron to Artificial Neuron

Ans) 

3)Activation Functions

Ans) 

2 INLAB

[1]:
```
'''
Implement various activation functions and visualize the results.

Binary Sigmoidal:
Bipolar sigmoidal function:
'''
import numpy as np
```

[2]:
```
wt, bias = np.random.random(1)[0], np.random.random(1)[0]
```

[3]:
```
def sigmoid(x):

    return (1/(1+np.exp(-x)))
```

[4]:
```
def bipolar_sigmoid(x):

    bi = -1 + 2 / (1 + np.exp(-x))
    return bi
```

[5]:
```
def summation_function(inp):

    summation = 0
    for idx in range(0, len(inp)):

        summation += inp[idx] * wt[idx]

    bias = np.random.random(1)[0]

    total = summation + bias
    return total, bias
```

[7]:
```
inp = []
wt = []
n = int(input("Enter the number of inputs: "))

for i in range(0, n):

    inp.append(float(input("Enter input: ")))
    wt.append(np.random.random(1)[0])

#Input:
'''
inp = [0.5, 0.9, 0.2, 0.3]
wt = [0.2, 0.3, -0.6, -0.1]
bias = 0.5
```

```
'''
```

[8]:
```python
total, bias = summation_function(inp)
print("sigmoid value:", sigmoid(total))
print("bipolar value:", bipolar_sigmoid(total))
print("weight vetor: ", wt)
print("bias value: ", bias)
```

```
sigmoid value: 0.9938861847055479
bipolar value: 0.9877723694110958
weight vetor:  [0.5612268683814017, 0.7802601155127734, 0.7740951793729846]
bias value:  0.6470390475171633
```

3 Post-Lab

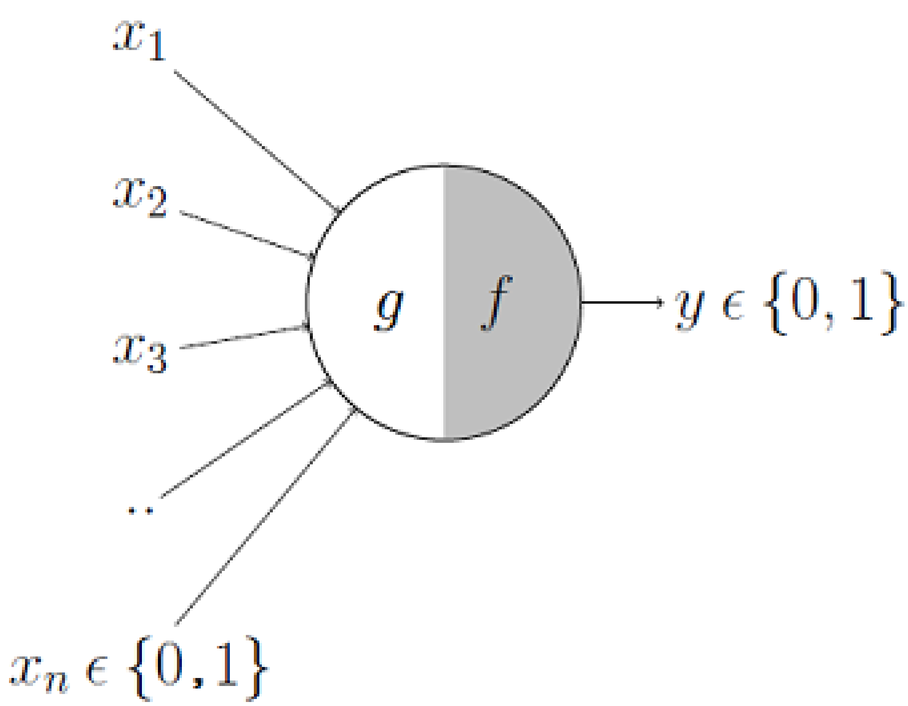
[ ]:
```python
'''
1) Assign appropriate standard notations for the following:
a) Training input vector: x[i]
b) Activation of unit Yj: f(g(x))
c) Activation of unit Xi: g(xi)
d) Weight matrix:  W[i,j]
e) Norm of magnitude vector X: ||X||
f) Training output vector:yi
g) Input vector: xi
h) Learning rate, it controls the amount of weight adjustment at each step of␣
 ↪training:   (alpha) or   (eta)
i) Weight on connection from unit Xi to unit Yj : w[i,j]
j) Change in weights: Δw[i,j]
k) Bias acting on unit j: b[j]
l) Net input to unit Yj: sigma[j] = Σi w[i,j]x[i] + b[j]
m) Threshold for activation of neuron Yj:  [j]
2) Write all the notations that are used in the programs executed above.'''
```

SC_LAB-5

September 12, 2022

1 Pre Lab

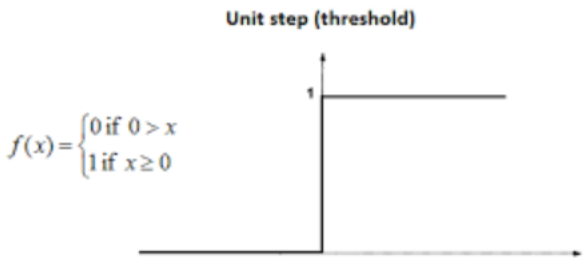
1.) Represent the McCulloch-Pitts Neuron mathematical model with a neat sketch



2.State the disadvantages of artificial neuron. Ans)

- it takes a lot of computational power
- neural network model are hard to explain
- it requiresd a lot of data for training
- Optimizing the models for production can be challenging.
- Data preparation for neural network model needs careful attention.
- it takes more time to compute

3. Give the mathematical representation of activavation function for McCulloch Pitts Neuron.



4.Analyze the effect caused by the activation function based on threshold values

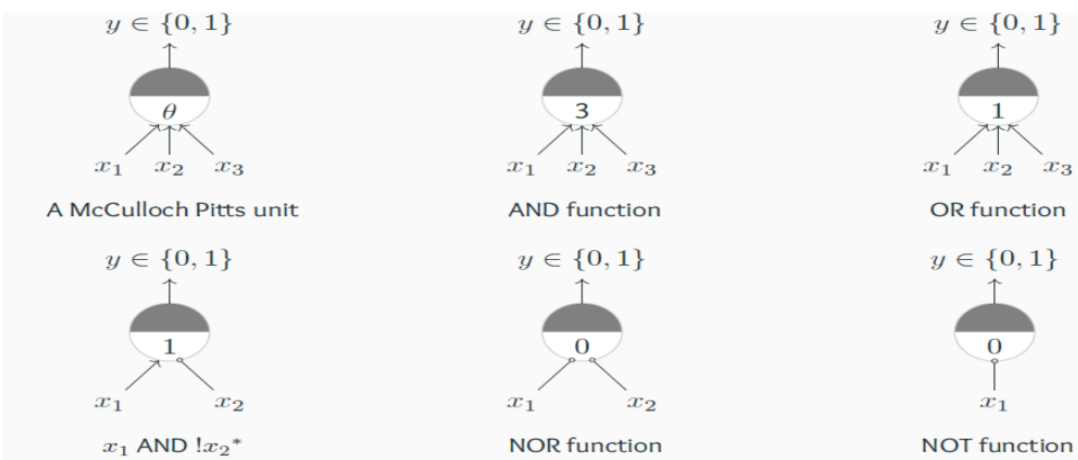
Ans)Activation function is a function that is used to get the output of a node,it is also called transfer function. it is used to determine the output of neural network like yes or no.It maps the resulting values in between 0 or 1 or -1 to 1 etc(depending upon the function)

2 In-Lab

[11]:
```
'''Implement ANDNOT logic gate using McCulloch-Pitts neural model. This neural␣
↪model will
deal with the ANDNOT logic. '''
```

[11]:
```
'Implement ANDNOT logic gate using McCulloch-Pitts neural model. This neural
model will \ndeal with the ANDNOT logic. '
```

#Compare the threshold values of logic gates given below (AND , NOT , OR)



3

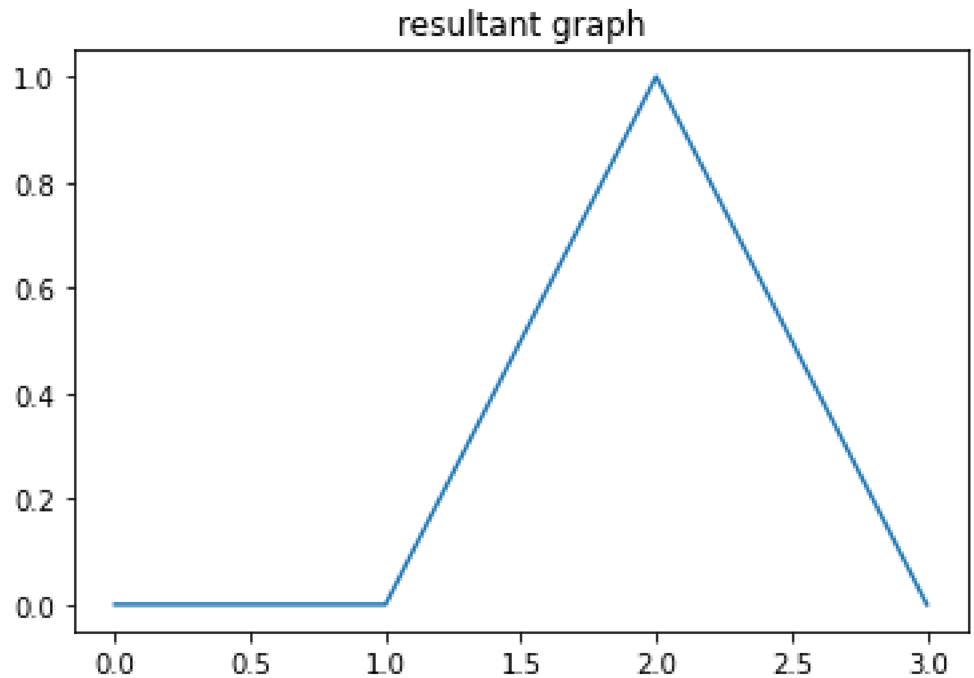
[9]:
```
# implementation of andnot logic using mcculloch pitts neuron
def compute_Neuron(x1,x2,w1,w2):
  res = []
  for i in range(len(x1)):
    res.append(x1[i]*w1+x2[i]*w2)
```

```
    #print(x1[i],w1,x2[i],w1)
    # print(res[i])
  return res
import pandas as pd
w1 = 1
w2 = 1
x1 = [0,0,1,1]
x2 = [0,1,0,1]
# let us consider random weights for meuron model
res = compute_Neuron(x1,x2,w1,w2)
# print(res)
# as we can see that we are having biased outcome hence we reassume our weights␣
 ↪and continue the process
```

[10]:
```
def check(res):
  flag = 0
  for i in range(len(res)):
    if res[i] > 1:
      flag += 1
    if res[i] < 0:
      res[i] = 0
  return flag
flag = check(res)
#print('flag is',flag)
if flag > 0:
  w2 = -1

  res = compute_Neuron(x1,x2,w1,w2)
  check(res)
  print(res)
else:
  flag = 0
  check(res)
  print(res)
```

```
[0, 0, 1, 0]
```

[11]:
```
import matplotlib.pyplot as plt
plt.title('resultant graph')
plt.plot(res)
plt.show()
```



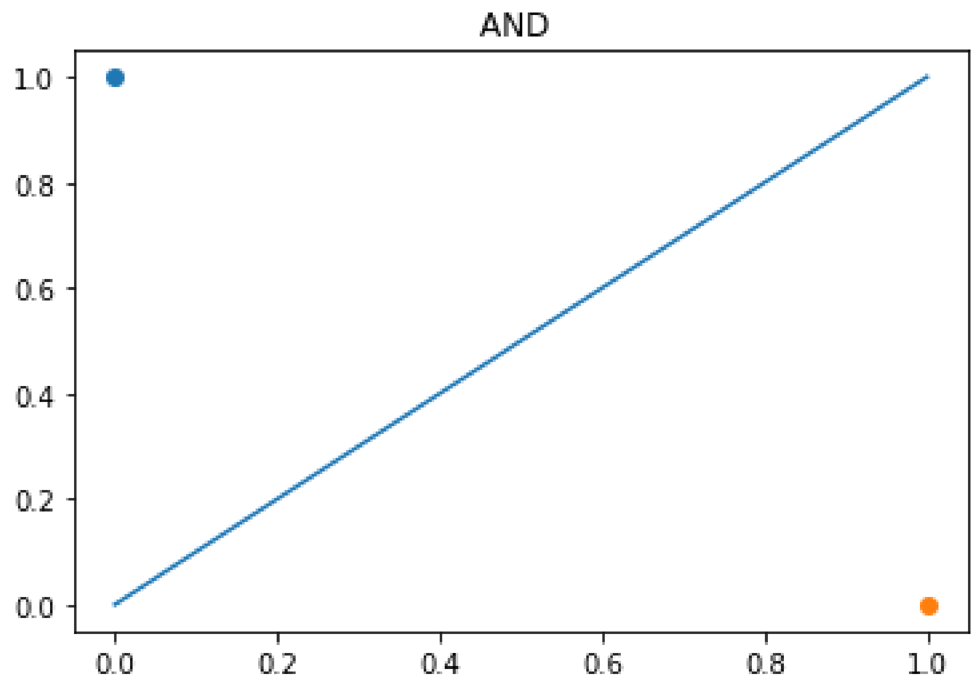
4 post lab

[14]:
```
'''Implement XOR logic gate using McCulloch-Pitts neural model. This neural␣
↪model will
deal with the XOR logic'''

x1 = [0,0,1,1]
x2 = [0,1,0,1]
res = []
ind = []
w1 = 1
w2 = 1
for i in range(len(x1)):
  if(x1 [i]== 0 and x2[i] == 1) or (x1[i] == 1 and x2[i] == 0):
    ind.append([x1[i],x2[i]])
    res.append(1)
  # elif(x1 == 1 and x2 == 0):
  #   res.append(1)
  else:
    res.append(0)
print(res)
print(ind)
print(ind[0][1])
```

```
[0, 1, 1, 0]
[[0, 1], [1, 0]]
1
```

[15]:
```
plt.title('AND ')
plt.plot([0,1])
plt.scatter(ind[0][0],ind[0][1])
plt.scatter(ind[1][0],ind[1][1])
```

[15]:
```
<matplotlib.collections.PathCollection at 0x235ddddda60>
```

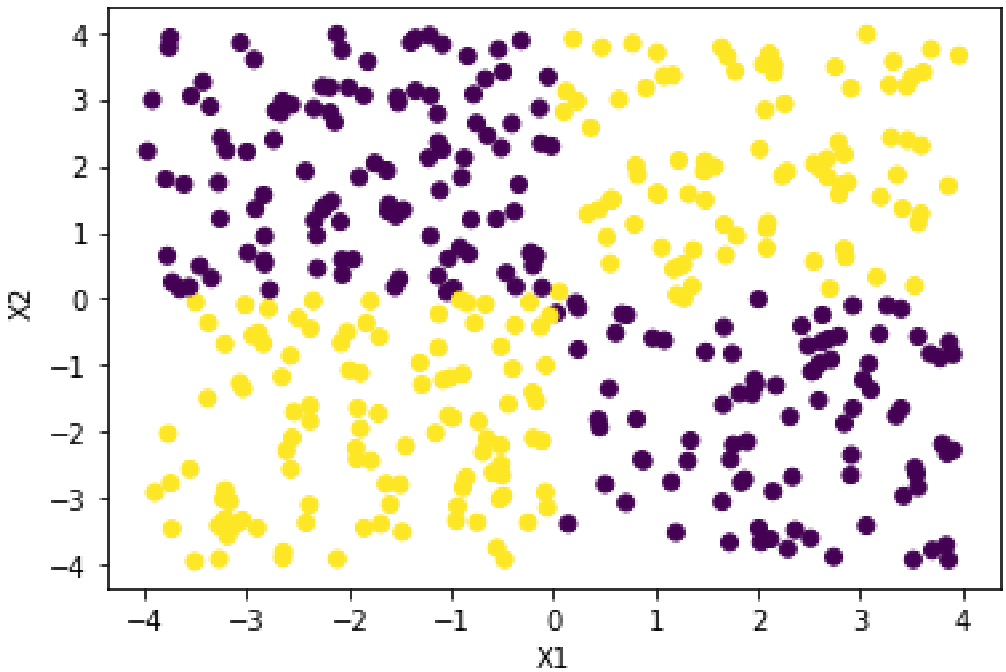


2.what are the drawbacks of McCulloch-Pitts neurons Ans)It can only process binary inputs

5 Other XOR visualizations

[16]:

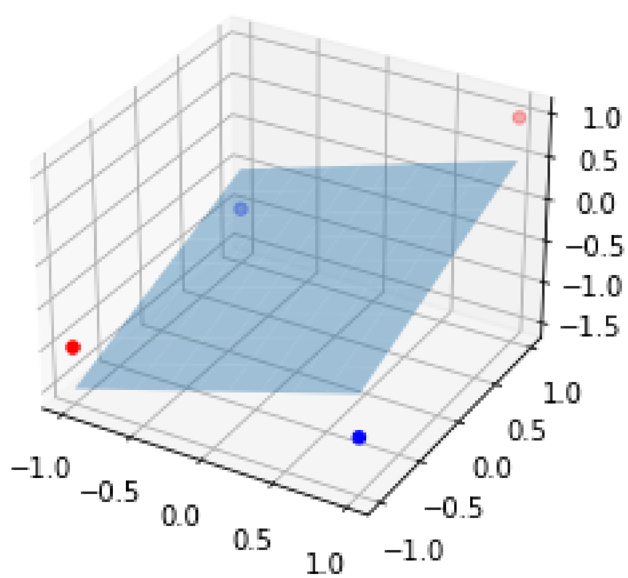
```
import numpy as np
import matplotlib.pyplot as plt

x = np.random.uniform(low=-4, high=4, size=(400,2))
y = np.bitwise_xor(np.sign(x[:,0]).astype(int),np.sign(x[:,1]).astype(int))
plt.scatter(x[:,0],x[:,1],c=y)
plt.xlabel('X1')
plt.ylabel('X2')

plt.show()
```



[17]:

```
#3D plot of XOR

from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# original data
# not linearly separable in 2D
x = np.array([[1, 1], [-1, -1], [-1, 1], [1, -1]])
y = np.array([-1, -1, 1, 1])
```

```
# feature mapping to produce x3 = x1 | x2
# add feature x3             | x3 computed here |
x = np.hstack([x, np.array([x[:, 0] | x[:, 1]]).T])
plus = x[y == 1]
minus = x[y == -1]
ax.scatter(plus[:, 0], plus[:, 1], plus[:, 2], c='b')
ax.scatter(minus[:, 0], minus[:, 1], minus[:, 2], c='r')

# these control the position and norm of the plane
point  = np.array([0, 0, 0.5])
normal = np.array([-1, -1, 2])

# this is copy paste for plotting a plane
# a plane is a*x+b*y+c*z+d=0
# [a,b,c] is the normal. Thus, we have to calculate
# d and we're set
d = -point.dot(normal)
xx, yy = np.meshgrid(np.linspace(-1, 1, 10), np.linspace(-1, 1, 10))
# calculate corresponding z
z = (-normal[0] * xx - normal[1] * yy + d) * 1. /normal[2]
# plot the surface
ax.plot_surface(xx, yy, z, alpha=0.4)
plt.show()
```



6 MP Neuron Method-2

[18]:
```python
import pandas as pd

import numpy as np
from sklearn.metrics import accuracy_score
import operator

class MPModel:
    def __init__(self, function='sum'):
        # We can pass some initial value of threshold.
        self.threshold = None
        if function == 'sum':
            self.function = self.sum_function

    def sum_function(self, x):
        return sum(x) >= self.threshold

    def and_function(self, x):
        return all(x)

    def or_function(self, x):
        return any(x)

    def fit(self, X_DataFrame, y_DataFrame):
        threshold_accuracy_dict = {}
        for threshold in range(len(X_DataFrame.columns) + 1):
            threshold_accuracy_dict[threshold] = None
        for threshold in threshold_accuracy_dict.keys():
            self.threshold = threshold
            predictions = self.predict(X_DataFrame)
            threshold_accuracy_dict[threshold] = accuracy_score(y_DataFrame,␣
 ↪predictions)
        self.threshold = max(threshold_accuracy_dict.items(), key=operator.
 ↪itemgetter(1))[0]
        print(self.threshold, 'threshold', threshold_accuracy_dict)

    def predict(self, X_DataFrame):
        results = np.array([])
        for i in range(len(X_DataFrame)):
            result = self.function(X_DataFrame.iloc[i])
            results = np.append(results, result)
        return results.astype(int)
```

[19]:
```python
df_dict = {
    'Wind': [0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0],
    'Temp': [1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0],
```

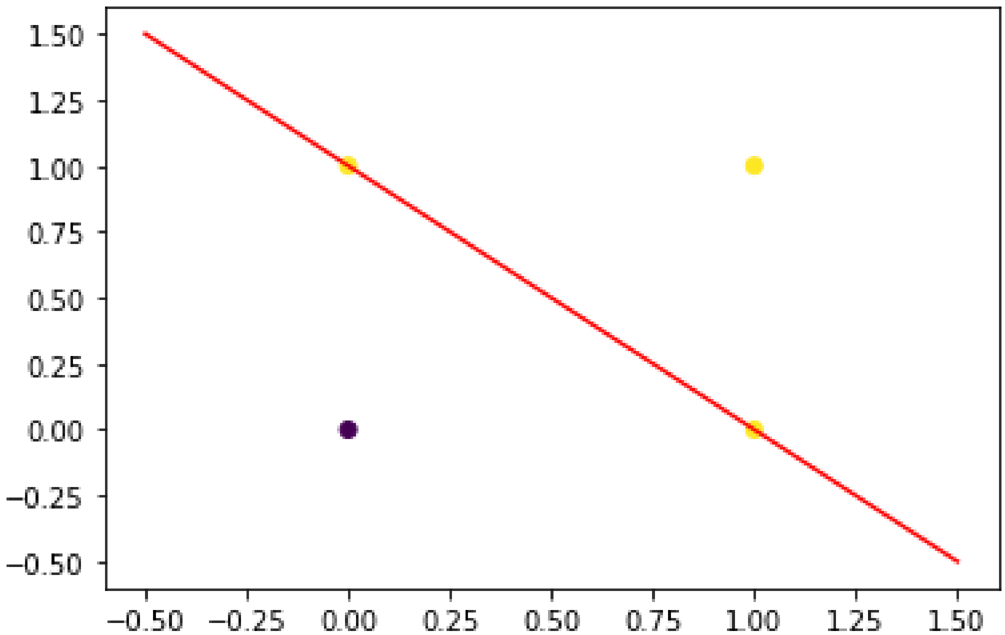
```
    'Played': [0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0]
}
pd_df = pd.DataFrame(df_dict)


mpm = MPModel()
mpm.fit(pd_df[['Wind', 'Temp']], pd_df['Played'])

df_dict_test = {
    'Wind': [0, 1, 1, 0],
    'Temp': [1, 1, 0, 0],
    'Played': [1, 1, 1, 0]
}

pd_df_test = pd.DataFrame(df_dict_test)
predictions = mpm.predict(pd_df_test[['Wind', 'Temp']])
accuracy_score(predictions, pd_df_test['Played'])
```

```
1 threshold {0: 0.5, 1: 0.8571428571428571, 2: 0.5}
```

[19]:
```
1.0
```

[20]:
```
import matplotlib.pyplot as plt

x = np.linspace(-0.5, 1.5, 100)
y = -x + 1
plt.plot(x, y, '-r', label='y=2x+1')
plt.plot()
plt.scatter(df_dict_test['Wind'], df_dict_test['Temp'],
 ↪c=df_dict_test['Played'])
```

[20]:
```
<matplotlib.collections.PathCollection at 0x235ee0d10d0>
```


1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50

SC_LAB-6

September 12, 2022

1 Pre-Lab

1) a)Define learning rule in neural network.

Ans: Learning rule is a mathematical formula that describes how the weights of a neural network are updated during training. The learning rule is used to update the weights of the neural network based on the error between the actual output and the desired output.

b)Match the following(answers)

Delta ->Modification in sympatric weight of a node is equal to multiplication of the error and input

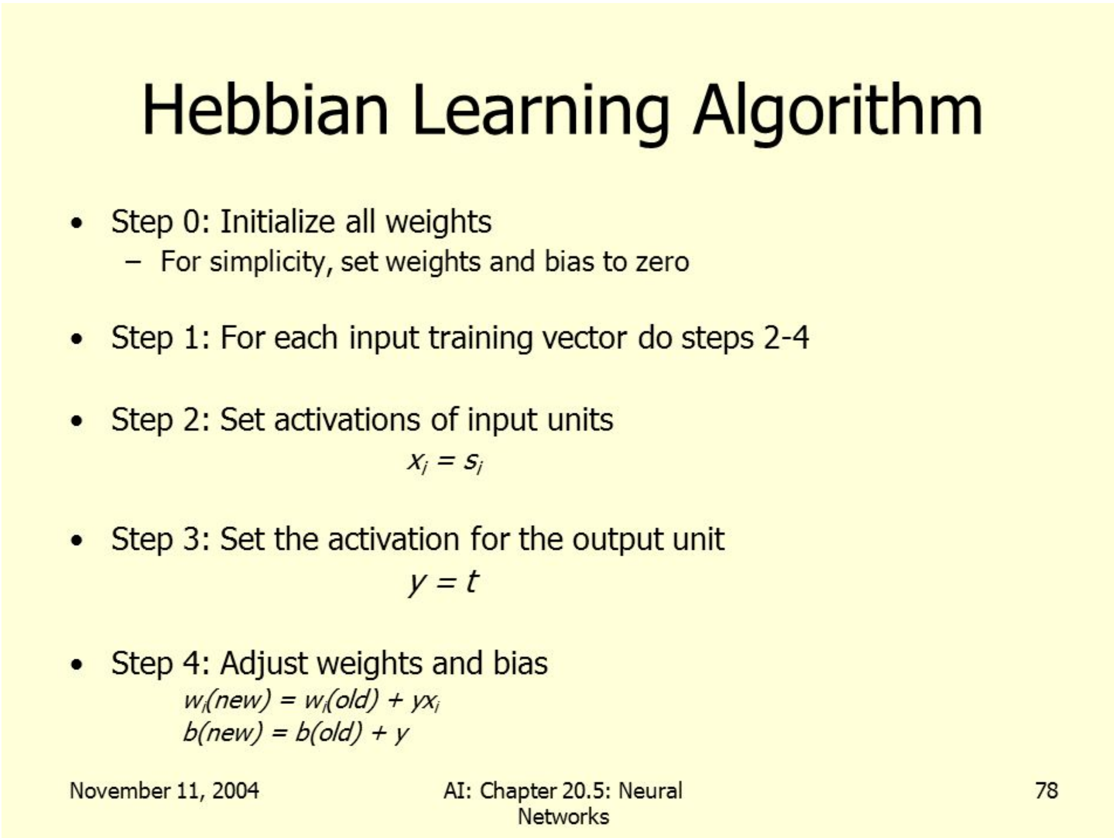
Perceptron ->Learning by assigning a random value to each weight

Hebbian ->Modifying weights of nodes of a network

Outstar ->Assumes that nodes or neurons in a network arranged in a Layer

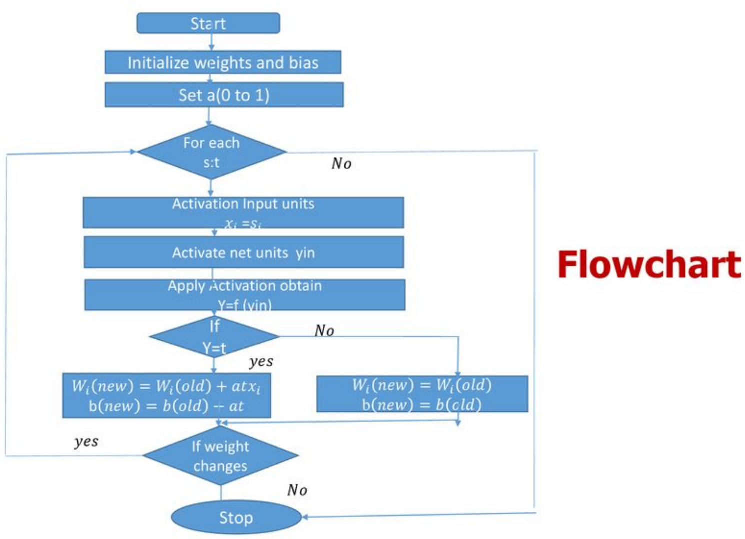
Correlation ->Supervised learning

2)When an axon of cell A is nearly enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A s efficiency, as one of the cells firing B, is increased. Write the algorithm for implementing the above mention rule

Ans) 

3)Hebb Rule Flow chart

Ans)



4)What is the need for modifying weights in a Hebb net?

Ans)The weights are found to increase proportionately to the product of input and output. The weights are modified in the Hebb net to allow for a more generalized learning process.

5)Given below is the neural net whose target value is 1. Implement the net by using the Hebb rule i. e; applying the bipolar step function for obtaining the net input and modify the weights accordingly up to 3 EPOCH

Ans)It is implemented below

[9]:
```
import numpy as np
x1 = 0.2
x2 = 0.6
weight1 = 0.3
weight2 = 0.7
bias = 1
bias_weight = 0.45
target = 1
print("the modified value in inp")
for i in range(0,3):

        print("EPOCH : " ,i)
y = weight1*x1+weight2*x2+ bias*bias_weight
if y == 1:
  print("no weight update")
else:
    weight1 = weight1+x1*target
    weight2 = weight2*x2*target
    bias = bias+target
    print("update weights are: ",weight1,'\t',weight2,'\t')
```

```
the modified value in inp
EPOCH :  0
EPOCH :  1
EPOCH :  2
update weights are:  0.5         0.42
```

2 In-Lab

[10]:
```
x1=[1,1,1,1,1,-1,-1,-1,1,1,1,1,1,-1,-1,-1,1,1,1,1,1]
x2=[1,1,1,1,1,-1,-1,-1,1,-1,-1,-1,1,-1,-1,-1,1,1,1,1]
t1=1
t2=-1
b=0
w=[0]*20
for i in range(20):
#     print("EPOCH: ",i)
#     print('updated weights are:',w[i],'\t',end='')
    w[i]=w[i]+t1*x1[i]
    #     print(w[i])
    b=b+t1
    # print(b)
```

```python
for i in range(20):
    print("EPOCH: ",i)
        #     print('updated weights are:',w[i],end=' ')
    w[i]=w[i]+t2*x2[i]
    print('updated weights are:',w)
b=b+t2
                # print(b)
                # print(w)
```

```
EPOCH:  0
updated weights are: [0, 1, 1, 1, 1, -1, -1, -1, 1, 1, 1, 1, -1, -1, -1, 1, 1,
1, 1, 1]
EPOCH:  1
updated weights are: [0, 0, 1, 1, 1, -1, -1, -1, 1, 1, 1, 1, -1, -1, -1, 1, 1,
1, 1, 1]
EPOCH:  2
updated weights are: [0, 0, 0, 1, 1, -1, -1, -1, 1, 1, 1, 1, -1, -1, -1, 1, 1,
1, 1, 1]
EPOCH:  3
updated weights are: [0, 0, 0, 0, 1, -1, -1, -1, 1, 1, 1, 1, -1, -1, -1, 1, 1,
1, 1, 1]
EPOCH:  4
updated weights are: [0, 0, 0, 0, 0, -1, -1, -1, 1, 1, 1, 1, -1, -1, -1, 1, 1,
1, 1, 1]
EPOCH:  5
updated weights are: [0, 0, 0, 0, 0, 0, -1, -1, 1, 1, 1, 1, -1, -1, -1, 1, 1, 1,
1, 1]
EPOCH:  6
updated weights are: [0, 0, 0, 0, 0, 0, 0, -1, 1, 1, 1, 1, -1, -1, -1, 1, 1, 1,
1, 1]
EPOCH:  7
updated weights are: [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, -1, -1, -1, 1, 1, 1,
1, 1]
EPOCH:  8
updated weights are: [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, -1, -1, -1, 1, 1, 1,
1, 1]
EPOCH:  9
updated weights are: [0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 1, -1, -1, -1, 1, 1, 1,
1, 1]
EPOCH:  10
updated weights are: [0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 1, -1, -1, -1, 1, 1, 1,
1, 1]
EPOCH:  11
updated weights are: [0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, -1, -1, -1, 1, 1, 1,
1, 1]
EPOCH:  12
updated weights are: [0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, -2, -1, -1, 1, 1, 1,
1, 1]
```

```
EPOCH:  13
updated weights are: [0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, -2, 0, -1, 1, 1, 1, 1,
1]
EPOCH:  14
updated weights are: [0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, -2, 0, 0, 1, 1, 1, 1,
1]
EPOCH:  15
updated weights are: [0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, -2, 0, 0, 2, 1, 1, 1,
1]
EPOCH:  16
updated weights are: [0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, -2, 0, 0, 2, 0, 1, 1,
1]
EPOCH:  17
updated weights are: [0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, -2, 0, 0, 2, 0, 0, 1,
1]
EPOCH:  18
updated weights are: [0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, -2, 0, 0, 2, 0, 0, 0,
1]
EPOCH:  19
updated weights are: [0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, -2, 0, 0, 2, 0, 0, 0,
0]
```

3 Method-2

[11]:
```python
import numpy as np

s = [

    [ 1,  1,  1,  1],
    [ 1, -1, -1, -1],
    [ 1,  1,  1,  1],
    [-1, -1, -1,  1],
    [ 1,  1,  1,  1]
]

c = [
    [ 1,  1,  1,  1],
    [ 1, -1, -1, -1],
    [ 1, -1, -1, -1],
    [ 1, -1, -1, -1],
    [ 1,  1,  1,  1]
]
x1 = np.array(s).flatten().reshape((1, 20))
x2 = np.array(c).flatten().reshape((1, 20))

weights = np.zeros((1, 20))
bias_weight = 0.45
```

```
bias = 0

np.dot(x1, weights.T) + bias * bias_weight
```

[11]: array([[0.]])

[12]:
```
epochs = 100
for epoch in range(0, epochs):

    if(epoch % 2 == 0):

        inp = x1
        target = 1
    else:

        inp = x2
        target = -1

    print("Epoch:", epoch)
    y =  np.dot(inp, weights.T) + bias

    print(y)
    if target == y:

        print('no weight update')
        break
    else:

        weights = weights + inp * target
        bias = bias + target
        print("Updated weights are:", weights, '\n', bias)
```

```
Epoch: 0
[[0.]]
Updated weights are: [[ 1.  1.  1.  1.  1. -1. -1. -1.  1.  1.  1.  1. -1. -1.
-1.  1.  1.  1.
   1.  1.]]
 1
Epoch: 1
[[11.]]
Updated weights are: [[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  2.  2.  2. -2.  0.
0.  2.  0.  0.
   0.  0.]]
 0
Epoch: 2
[[10.]]
Updated weights are: [[ 1.  1.  1.  1.  1. -1. -1. -1.  1.  3.  3.  3. -3. -1.
-1.  3.  1.  1.
```

```
   -1.  99.   1.   1.   1.   1.]]
 1
Epoch: 99
[[-479.]]
Updated weights are: [[   0.     0.     0.     0.     0.     0.     0.     0.     0.
100.  100.  100.
  -100.     0.     0.  100.     0.     0.     0.     0.]]
 0
```

4 Method-3

[13]:
```python
import numpy as np
import pandas as pd

def Input(a):
    if(a == 1):
        return[1,1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,1,-1,-1,-1,-1,1,1,1,1,1,1],1 #S
    return [1,1,1,1,1,1,-1,-1,-1,-1,1,-1,-1,-1,-1,1,-1,-1,-1,-1,1,1,1,1,1,1],-1 #C
```

[14]:
```python
w = [0]*20
b =  0
for i in  range(0,600):
    x,y = Input(i%2)
    print("Ecopus:",i+1)

    for j in range(20):
        w[j] = w[j] + x[j] * y

    print("updated weights :",w)
    b = b + y
    print("updated bais:",b)
```

```
Ecopus: 1
updated weights : [-1, -1, -1, -1, -1, -1, 1, 1, 1, 1, -1, 1, 1, 1, 1, -1, 1, 1,
1, 1]
updated bais: -1
Ecopus: 2
updated weights : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, -2, 0, 0, 0, 2]
updated bais: 0
Ecopus: 3
updated weights : [-1, -1, -1, -1, -1, -1, 1, 1, 1, 1, -1, 3, 3, 3, 3, -3, 1, 1,
1, 3]
updated bais: -1
Ecopus: 4
updated weights : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, -4, 0, 0, 0, 4]
```

```
updated weights : [-1, -1, -1, -1, -1, -1, 1, 1, 1, 1, -1, 593, 593, 593, 593,
-593, 1, 1, 1, 593]
updated bais: -1
Ecopus: 594
updated weights : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 594, 594, 594, 594, -594, 0,
0, 0, 594]
updated bais: 0
Ecopus: 595
updated weights : [-1, -1, -1, -1, -1, -1, 1, 1, 1, 1, -1, 595, 595, 595, 595,
-595, 1, 1, 1, 595]
updated bais: -1
Ecopus: 596
updated weights : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 596, 596, 596, 596, -596, 0,
0, 0, 596]
updated bais: 0
Ecopus: 597
updated weights : [-1, -1, -1, -1, -1, -1, 1, 1, 1, 1, -1, 597, 597, 597, 597,
-597, 1, 1, 1, 597]
updated bais: -1
Ecopus: 598
updated weights : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 598, 598, 598, 598, -598, 0,
0, 0, 598]
updated bais: 0
Ecopus: 599
updated weights : [-1, -1, -1, -1, -1, -1, 1, 1, 1, 1, -1, 599, 599, 599, 599,
-599, 1, 1, 1, 599]
updated bais: -1
Ecopus: 600
updated weights : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 600, 600, 600, 600, -600, 0,
0, 0, 600]
updated bais: 0
```

5 Post-Lab

[1]:

```
'''Complete the code given below for updating weights using Hebb rule in␣
↪accordance
with the output '''

'''
The modified values in input
[ 1, -2,    1.5    0  ]
[ 2, -2.5, -0.5,  -1.5]
[ 2, -1.5, -1.5,   0  ]
[ 3,  0,   -3.25, -0.5]
'''

import numpy as np
```

```python
x1=np.array([1.0,-2,1.5,0.0])
x2=np.array([1.0,-0.5,-2.0,-1.5])
x3=np.array([0.0,1.0,-1.0,1.5])
x4=np.array([1.0,1.5,-1.75,-0.5])
inp=np.array([x1,x2,x3,x4])
weight=np.array([0,0,0,0])
weight1=np.array([0,0,0,0])
target=np.array([1,1,1,1])
print('the modified values in inp')

for i in range(0,4):
    print('EPOCH : ',i)
    y=np.dot(inp[i],weight)
    if y==1:
        print('no weight update')
    else:
        weight=weight+inp[i]*target[i]
        print('update weights are: ',weight)
```

```
the modified values in inp
EPOCH :  0
update weights are:  [ 1.  -2.   1.5  0. ]
EPOCH :  1
update weights are:  [ 2.  -2.5 -0.5 -1.5]
EPOCH :  2
update weights are:  [ 2.  -1.5 -1.5  0. ]
EPOCH :  3
update weights are:  [ 3.    0.   -3.25 -0.5 ]
```

SC_LAB-7_Sigmoid_Neuron

September 19, 2022

1 Pre-Lab

1) Define supervised learning and list some examples.

A) It is defined by its use of labeled datasets to train algorithms that to classify data or predict outcomes accurately.As input data is fed into the model, it adjusts its weights until the model has been fitted appropriately, which occurs as part of the cross validation process. Examples: Image- and object-recognition, Predictive analytics, Customer sentiment analysis, Spam detection.

2)Name the neural models that employs supervised learning to classify the data into classes.

Ans)Artificial Neural Network, Convolutional Neural Network, Recurrent Neural Network

3)Link the below units to their respective functionality:

Ans) Sensory Unit - Input unit Associator Unit - Hidden unit Response Unit - Output unit

4)When does a perceptron change its weights and write the equation used by perceptron to update its weight?

Ans) Perceptron changes its weights when the output is not equal to the desired output. The equation used by perceptron to update its weight is:

If y' = y, do nothing.

Otherwise wi=wi+ ∗y∗xi

5)Fill the blanks: Ans)

a)The weight updating process in the perceptron learning rule takes place between the associator unit and response unit

b)Sensory - unit has an activation value of -1,0,1 in perceptron network.

c)The output signals that are sent from the associator unit to the response unit are binary.

6) How are the sensory units connected to associator units?

Ans)The sensory units are connected to associator units with fixed weights having values 1, 0 or -l, which are assigned at random.

7) Implement AND function using perceptron networks for bipolar inputs and targets Answered below

[6]:
```
# importing Python library
import numpy as np

# define Unit Step Function
def unitStep(v):
 if v >= 0:
  return 1
 else:
  return 0

# design Perceptron Model
def perceptronModel(x, w, b):
 v = np.dot(w, x) + b
 y = unitStep(v)
 return y

# AND Logic Function
# w1 = 1, w2 = 1, b = -1.5
def AND_logicFunction(x):
 w = np.array([1, 1])
 b = -1.5
 return perceptronModel(x, w, b)

# testing the Perceptron Model
test1 = np.array([0, 1])
test2 = np.array([1, 1])
test3 = np.array([0, 0])
test4 = np.array([1, 0])

print("AND({}, {}) = {}".format(0, 1, AND_logicFunction(test1)))
print("AND({}, {}) = {}".format(1, 1, AND_logicFunction(test2)))
print("AND({}, {}) = {}".format(0, 0, AND_logicFunction(test3)))
print("AND({}, {}) = {}".format(1, 0, AND_logicFunction(test4)))
```

```
AND(0, 1) = 0
AND(1, 1) = 1
AND(0, 0) = 0
AND(1, 0) = 0
```

2 Perceptron

[20]:
```
import numpy as np
wt, bias = np.random.random(1)[0], np.random.random(1)[0]
```

[21]:
```
def sigmoid(x):

    return (1/(1+np.exp(-x)))
```

[22]:
```python
def bipolar_sigmoid(x):

    bi = -1 + 2 / (1 + np.exp(-x))
    return bi
```

[23]:
```python
def summation_function(inp):

    summation = 0
    for idx in range(0, len(inp)):

        summation += inp[idx] * wt[idx]

    bias = np.random.random(1)[0]

    total = summation + bias
    return total, bias
```

[30]:
```python
inp = []
wt = []
n = int(input("Enter the number of inputs: "))

for i in range(0, n):

    inp.append(float(input("Enter input: ")))
    wt.append(np.random.random(1)[0])

#Inputs :
"""
Enter the number of inputs: 2
Enter input: 1
Enter input: 1"""
```

```
Enter the number of inputs: 2
Enter input: 1
Enter input: 1
```

[31]:
```python
total, bias = summation_function(inp)
print("sigmoid value:", sigmoid(total))
print("bipolar value:", bipolar_sigmoid(total))
print("weight vetor: ", wt)
print("bias value: ", bias)
```

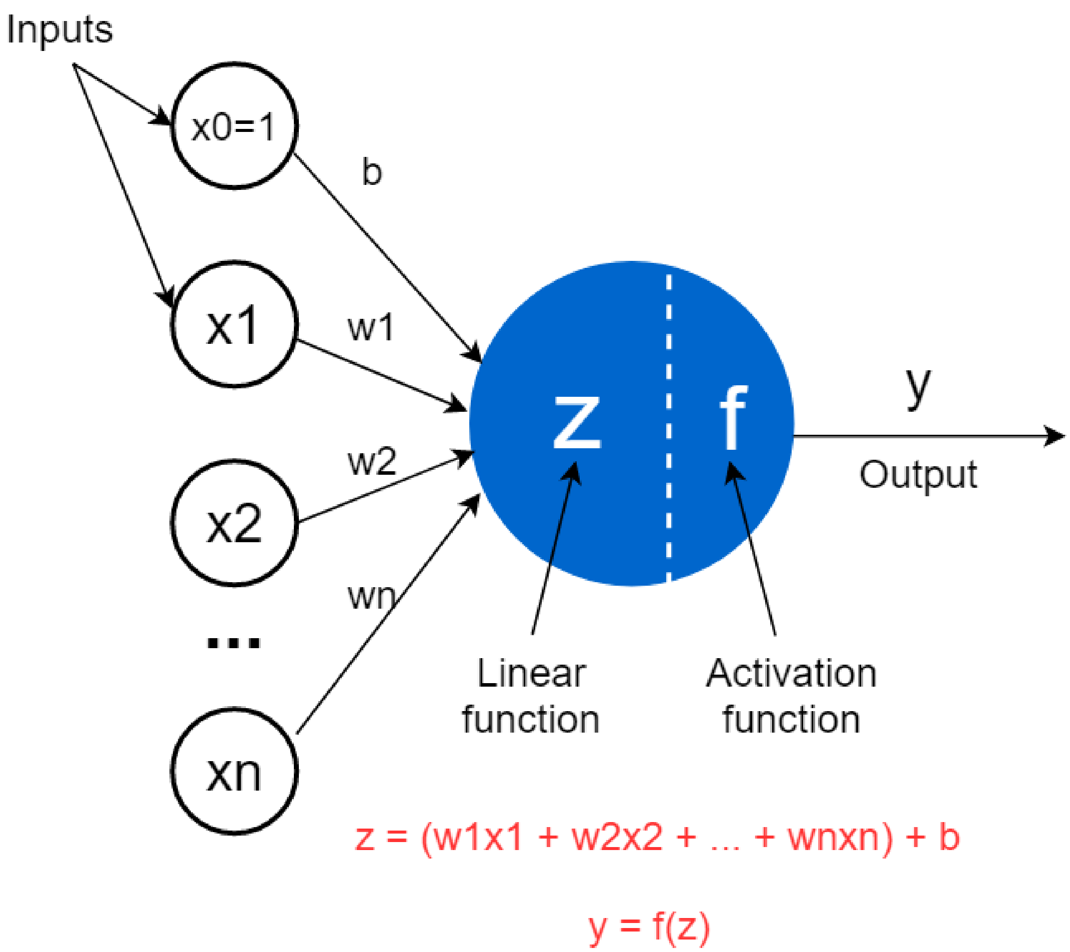
```
sigmoid value: 0.7369929480632201
bipolar value: 0.47398589612644026
weight vetor:  [0.18512235569735258, 0.38651491540832594]
bias value:  0.45876020733650646
```

2.1 implementing OR function using perceptron model.

[7]:

```
# importing Python library
import numpy as np

# define Unit Step Function
def unitStep(v):
        if v >= 0:
                return 1
        else:
                return 0

# design Perceptron Model
def perceptronModel(x, w, b):
        v = np.dot(w, x) + b
        y = unitStep(v)
        return y

# OR Logic Function
# w1 = 1, w2 = 1, b = -0.5
def OR_logicFunction(x):
        w = np.array([1, 1])
        b = -0.5
        return perceptronModel(x, w, b)

# testing the Perceptron Model
test1 = np.array([0, 1])
test2 = np.array([1, 1])
test3 = np.array([0, 0])
test4 = np.array([1, 0])

print("OR({}, {}) = {}".format(0, 1, OR_logicFunction(test1)))
print("OR({}, {}) = {}".format(1, 1, OR_logicFunction(test2)))
print("OR({}, {}) = {}".format(0, 0, OR_logicFunction(test3)))
print("OR({}, {}) = {}".format(1, 0, OR_logicFunction(test4)))
```

```
OR(0, 1) = 1
OR(1, 1) = 1
OR(0, 0) = 0
OR(1, 0) = 1
```



$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

3 Post-Lab

[ ]:
```
'''Find the weights required to perform the following classification using perceptron
network. The vectors [1 1 1 1] and [-1 1 -1 -1] are belonging to the class (so have the
target value 1), vectors [1 1 1 -1] and [1 -1 -1 1] are not belonging to the class (so the
target value is -1).Implement the classification assuming the learning rate as 1 and initial
```

```
weights as 0.'''
```

[10]:
```
import numpy as np

def sigmoid(x):
    return 1/(1 + np.exp(-x))

def error(target, output):
    return target - output

#Weight Update
def weight_update(w, learning_rate, error, input1):
    return w + [(learning_rate * error * input1)]*len(w)

#Classification
def perceptron(input1, target, w, learning_rate):
    output = sigmoid(np.dot(input1, w))
    error1 = error(target, output)
    w = weight_update(w, learning_rate, error, input1)
    return w

print("The weights are: ", perceptron([1, 1, 1, 1], 1, [0, 0, 0, 0], 1))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_11316/4187270011.py in <module>
     18     return w
     19 
---> 20 print("The weights are: ", perceptron([1, 1, 1, 1], 1, [0, 0, 0, 0], 1)

~\AppData\Local\Temp/ipykernel_11316/4187270011.py in perceptron(input1, target
 ↪w, learning_rate)
     15     output = sigmoid(np.dot(input1, w))
     16     error1 = error(target, output)
---> 17     w = weight_update(w, learning_rate, error, input1)
     18     return w
     19 

~\AppData\Local\Temp/ipykernel_11316/4187270011.py in weight_update(w,
 ↪learning_rate, error, input1)
      9 #Weight Update
     10 def weight_update(w, learning_rate, error, input1):
---> 11     return w + [(learning_rate * error * input1)]*len(w)
     12 
     13 #Classification
```

```
TypeError: unsupported operand type(s) for *: 'int' and 'function'
```

[ ]:

```

```

Neural Network from scratch for MNIST

September 19, 2022

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[ ]:
```
data = pd.read_csv("train.csv")
data.head()
```

[ ]:
```
   label  pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  \
0      1       0       0       0       0       0       0       0       0
1      0       0       0       0       0       0       0       0       0
2      1       0       0       0       0       0       0       0       0
3      4       0       0       0       0       0       0       0       0
4      0       0       0       0       0       0       0       0       0

   pixel8  …  pixel774  pixel775  pixel776  pixel777  pixel778  pixel779  \
0       0  …         0         0         0         0         0         0
1       0  …         0         0         0         0         0         0
2       0  …         0         0         0         0         0         0
3       0  …         0         0         0         0         0         0
4       0  …         0         0         0         0         0         0

   pixel780  pixel781  pixel782  pixel783
0         0         0         0         0
1         0         0         0         0
2         0         0         0         0
3         0         0         0         0
4         0         0         0         0

[5 rows x 785 columns]
```

[ ]:
```
data = np.array(data)
m, n = data.shape
np.random.shuffle(data)

data_dev = data[0:1000].T
x_dev = data_dev[1:n]
y_dev = data_dev[0]
```

```
data_train = data[1000:m].T
y_train = data_train[0]
x_train = data_train[1:n]
x_train = x_train/ 255.
```

[ ]:
```
x_train, y_train
```

[ ]:
```
(array([[0., 0., 0., …, 0., 0., 0.],
        [0., 0., 0., …, 0., 0., 0.],
        [0., 0., 0., …, 0., 0., 0.],
        …,
        [0., 0., 0., …, 0., 0., 0.],
        [0., 0., 0., …, 0., 0., 0.],
        [0., 0., 0., …, 0., 0., 0.]]),
 array([0, 3, 3, …, 8, 0, 2]))
```

[ ]:
```
def init_parameters():

    w1 = np.random.rand(10, 784) - 0.5
    # each of 10 nodes has 784 connections
    b1 = np.random.rand(10, 1) - 0.5
    # each of 10 nodes has 1 bias value

    w2 = np.random.rand(10, 10) - 0.5
    b2 = np.random.rand(10, 1) - 0.5

    return w1, b1, w2, b2

def sigmoid(x):

    return 1 / (1 + np.exp(-x))

def softmax(x):

    a = np.exp(x) / sum(np.exp(x))
    return a

def ReLU(Z):
    return np.maximum(Z, 0)
def ReLU_deriv(Z):
    return Z > 0

def forward_prop(w1, b1, w2, b2, x):

    z1 = w1.dot(x) + b1
    a1 = ReLU(z1)
```

```
    z2 = w2.dot(a1) + b2
    a2 = softmax(z2)

    return z1, a1, z2, a2

def one_hot(y):

    one_hot_y = np.zeros((y.size, y.max() + 1))
    one_hot_y[np.arange(y.size), y] = 1
    one_hot_y = one_hot_y.T

    return one_hot_y

def derivative_sigmoid(x):

    return sigmoid(x) *(1-sigmoid(x))

def backward_prop(z1, a1, z2, a2, w1, w2, x, y):

    m = y.size
    one_hot_y = one_hot(y)

    dz2 = a2 - one_hot_y
    dw2 = 1 / m * dz2.dot(a1.T)
    db2 = 1 / m * np.sum(dz2)

    dz1 = w2.T.dot(dz2) * ReLU_deriv(z1)

    dw1 = 1 / m * dz1.dot(x.T)
    db1 = 1 / m * np.sum(dz1)

    return dw1, db1, dw2, db2

def update_parameters(w1, b1, w2, b2, dw1, db1, dw2, db2, alpha):
    w_1 = w1
    w1 = w1 - alpha * dw1
    w2 = w2 - alpha * dw2
    b1 = b1 - alpha * db1
    b2 = b2 - alpha * db2

    return w1, b1, w2, b2
```

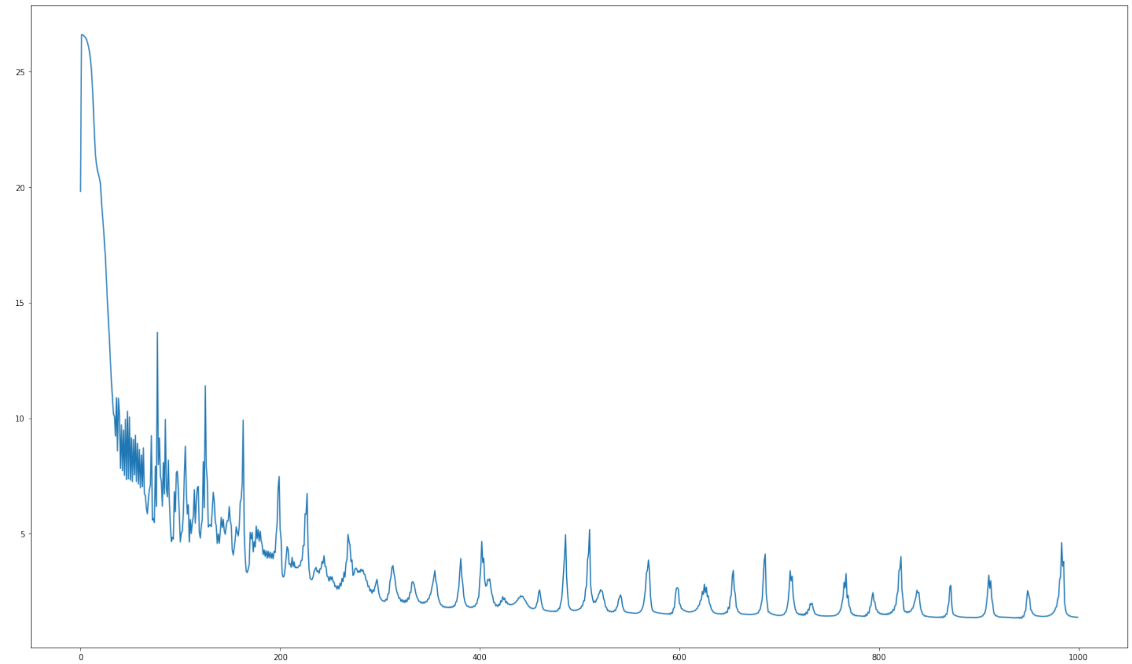
[ ]:
```
def get_predictions(a):
    return np.argmax(a, 0)

def get_accuracy(predictions, y):
```

```python
    return np.sum(predictions == y) / y.size

def calculate_loss_function(predictions, y):

    squared_error = (predictions - y ) ** 2
    sum_squared_error = np.sum(squared_error)
    loss = sum_squared_error / y.size

    return loss

def gradient_descent(x, y, iterations, alpha):


    loss_iteration = []
    w1, b1, w2, b2 = init_parameters()
    for i in range(iterations):

        z1, a1, z2, a2 = forward_prop(w1, b1, w2, b2, x)
        dw1, db1, dw2, db2 = backward_prop(z1, a1, z2, a2, w1, w2, x, y)

        # calculating loss at each iteration
        predictions = get_predictions(a2)
        loss_iteration.append(calculate_loss_function(predictions, y))

        # updating parameters after back-propagation
        w1, b1, w2, b2 = update_parameters(w1, b1, w2, b2, dw1, db1, dw2, db2,␣
 ↪alpha)

        if(i % 100 == 0):

            predictions = get_predictions(a2)
            print("iteration: ", i)
            print("Accuracy: ", get_accuracy(predictions, y), end='\n\n')

    # plot loss function
    plt.figure(figsize = (25, 15))
    plt.plot(np.arange(iterations), loss_iteration)
    return w1, b1, w2, b2
```

[ ]:
```python
w1, b1, w2, b2 = gradient_descent(x_train, y_train, 1000, 0.9)
```

```
iteration:  0
Accuracy:  0.06358536585365854

iteration:  100
Accuracy:  0.7413170731707317
```

```
iteration:  200
Accuracy:  0.6313658536585366

iteration:  300
Accuracy:  0.8601219512195122

iteration:  400
Accuracy:  0.8371951219512195

iteration:  500
Accuracy:  0.8906829268292683

iteration:  600
Accuracy:  0.8844634146341463

iteration:  700
Accuracy:  0.9085853658536586

iteration:  800
Accuracy:  0.9051707317073171

iteration:  900
Accuracy:  0.9136585365853659
```



SOM

September 19, 2022

[ ]:
```
import numpy as np
from numpy.ma.core import ceil
from scipy.spatial import distance #distance calculation
from sklearn.preprocessing import MinMaxScaler #normalisation
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score #scoring
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
from matplotlib import animation, colors
```

[ ]:
```
data_file = "data_banknote_authentication.txt"
data_x = np.loadtxt(data_file, delimiter=",", skiprows=0, usecols=range(0,4)␣
 ↪,dtype=np.float64)
data_y = np.loadtxt(data_file, delimiter=",", skiprows=0, usecols=(4,),dtype=np.
 ↪int64)
```

[ ]:
```
# banknote authentication Data Set
# https://archive.ics.uci.edu/ml/datasets/banknote+authentication
# Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.
 ↪ics.uci.edu/ml].
# Irvine, CA: University of California, School of Information and Computer␣
 ↪Science.

data_file = "data_banknote_authentication.txt"
data_x = np.loadtxt(data_file, delimiter=",", skiprows=0, usecols=range(0,4)␣
 ↪,dtype=np.float64)
data_y = np.loadtxt(data_file, delimiter=",", skiprows=0, usecols=(4,),dtype=np.
 ↪int64)
```

[ ]:
```
train_x, test_x, train_y, test_y = train_test_split(data_x, data_y, test_size=0.
 ↪2, random_state=42)
print(train_x.shape, train_y.shape, test_x.shape, test_y.shape) # check the␣
 ↪shapes
```

```
(1097, 4) (1097,) (275, 4) (275,)
```

[ ]:
```
# Helper functions
```

```
# Data Normalisation
def minmax_scaler(data):
  scaler = MinMaxScaler()
  scaled = scaler.fit_transform(data)
  return scaled

# Euclidean distance
def e_distance(x,y):
  return distance.euclidean(x,y)

# Manhattan distance
def m_distance(x,y):
  return distance.cityblock(x,y)

# Best Matching Unit search
def winning_neuron(data, t, som, num_rows, num_cols):
  winner = [0,0]
  shortest_distance = np.sqrt(data.shape[1]) # initialise with max distance
  input_data = data[t]
  for row in range(num_rows):
    for col in range(num_cols):
      distance = e_distance(som[row][col], data[t])
      if distance < shortest_distance:
        shortest_distance = distance
        winner = [row,col]
  return winner

# Learning rate and neighbourhood range calculation
def decay(step, max_steps,max_learning_rate,max_m_dsitance):
  coefficient = 1.0 - (np.float64(step)/max_steps)
  learning_rate = coefficient*max_learning_rate
  neighbourhood_range = ceil(coefficient * max_m_dsitance)
  return learning_rate, neighbourhood_range
```

[ ]:
```
# hyperparameters
num_rows = 10
num_cols = 10
max_m_dsitance = 4
max_learning_rate = 0.5
max_steps = int(7.5*10e3)

# num_nurons = 5*np.sqrt(train_x.shape[0])
# grid_size = ceil(np.sqrt(num_nurons))
# print(grid_size)
```

[ ]:
```
train_x_norm = minmax_scaler(train_x) # normalisation
```

```python
# initialising self-organising map
num_dims = train_x_norm.shape[1] # numnber of dimensions in the input data
np.random.seed(40)
som = np.random.random_sample(size=(num_rows, num_cols, num_dims)) # map␣
↪construction

# start training iterations
for step in range(max_steps):
  if (step+1) % 1000 == 0:
    print("Iteration: ", step+1) # print out the current iteration for every 1k
  learning_rate, neighbourhood_range = decay(step,␣
↪max_steps,max_learning_rate,max_m_dsitance)

  t = np.random.randint(0,high=train_x_norm.shape[0]) # random index of traing␣
↪data
  winner = winning_neuron(train_x_norm, t, som, num_rows, num_cols)
  for row in range(num_rows):
    for col in range(num_cols):
      if m_distance([row,col],winner) <= neighbourhood_range:
        som[row][col] += learning_rate*(train_x_norm[t]-som[row][col]) #update␣
↪neighbour's weight

print("SOM training completed")
```

```
Iteration:  1000
Iteration:  2000
Iteration:  3000
Iteration:  4000
Iteration:  5000
Iteration:  6000
Iteration:  7000
Iteration:  8000
Iteration:  9000
Iteration:  10000
Iteration:  11000
Iteration:  12000
Iteration:  13000
Iteration:  14000
Iteration:  15000
Iteration:  16000
Iteration:  17000
Iteration:  18000
Iteration:  19000
Iteration:  20000
Iteration:  21000
Iteration:  22000
Iteration:  23000
```

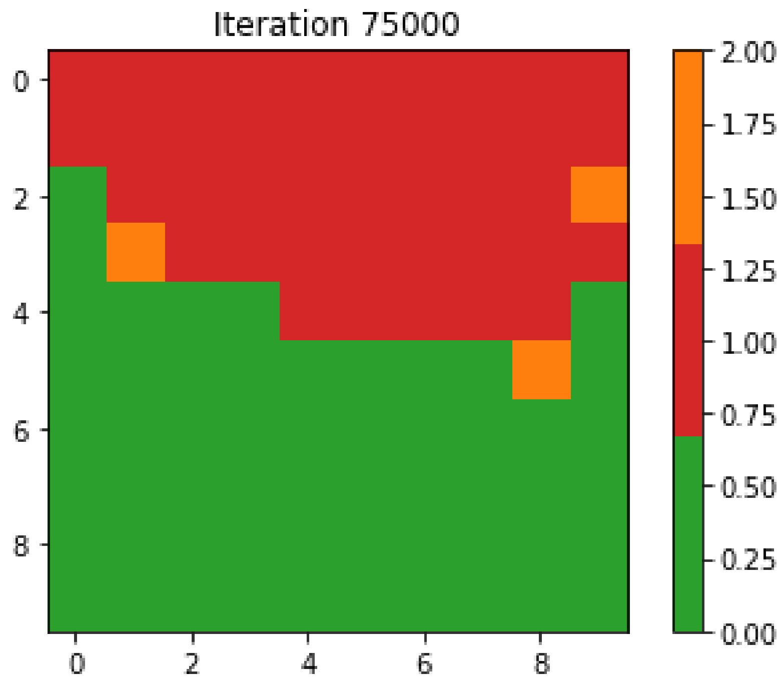
```
Iteration:  24000
Iteration:  25000
Iteration:  26000
Iteration:  27000
Iteration:  28000
Iteration:  29000
Iteration:  30000
Iteration:  31000
Iteration:  32000
Iteration:  33000
Iteration:  34000
Iteration:  35000
Iteration:  36000
Iteration:  37000
Iteration:  38000
Iteration:  39000
Iteration:  40000
Iteration:  41000
Iteration:  42000
Iteration:  43000
Iteration:  44000
Iteration:  45000
Iteration:  46000
Iteration:  47000
Iteration:  48000
Iteration:  49000
Iteration:  50000
Iteration:  51000
Iteration:  52000
Iteration:  53000
Iteration:  54000
Iteration:  55000
Iteration:  56000
Iteration:  57000
Iteration:  58000
Iteration:  59000
Iteration:  60000
Iteration:  61000
Iteration:  62000
Iteration:  63000
Iteration:  64000
Iteration:  65000
Iteration:  66000
Iteration:  67000
Iteration:  68000
Iteration:  69000
Iteration:  70000
Iteration:  71000
```

```
Iteration:  72000
Iteration:  73000
Iteration:  74000
Iteration:  75000
SOM training completed
```

[ ]:
```
# collecting labels

label_data = train_y
map = np.empty(shape=(num_rows, num_cols), dtype=object)

for row in range(num_rows):
  for col in range(num_cols):
    map[row][col] = [] # empty list to store the label

for t in range(train_x_norm.shape[0]):
  if (t+1) % 1000 == 0:
    print("sample data: ", t+1)
  winner = winning_neuron(train_x_norm, t, som, num_rows, num_cols)
  map[winner[0]][winner[1]].append(label_data[t]) # label of winning neuron
```

```
sample data:  1000
```

[ ]:
```
# construct label map
label_map = np.zeros(shape=(num_rows, num_cols),dtype=np.int64)
for row in range(num_rows):
  for col in range(num_cols):
    label_list = map[row][col]
    if len(label_list)==0:
      label = 2
    else:
      label = max(label_list, key=label_list.count)
    label_map[row][col] = label

title = ('Iteration ' + str(max_steps))
cmap = colors.ListedColormap(['tab:green', 'tab:red', 'tab:orange'])
plt.imshow(label_map, cmap=cmap)
plt.colorbar()
plt.title(title)
plt.show()
```



[ ]:
```
#som training
from datetime import datetime
print("Started at: ", datetime.now().strftime("%H:%M:%S"))

train_x_norm = minmax_scaler(train_x) # normalisation

# initialising self-organising map
num_dims = train_x_norm.shape[1] # nummber of dimensions in the input data
np.random.seed(40)
som = np.random.random_sample(size=(num_rows, num_cols, num_dims)) # map␣
 ↪construction

fig = plt.figure(figsize=(num_rows, num_cols))
ax = fig.add_subplot(111)
images = []
cbar_initialized = False
cmap = colors.ListedColormap(['tab:green', 'tab:red', 'tab:orange'])

lr = []
nr = []

# start training iterations
for step in range(max_steps):
  if (step+1) % 1000 == 0 and step > 0:
```

```
    print("Iteration: ", step+1, " at :", datetime.now().strftime("%H:%M:%S"))␣
↪# print out the current iteration for every 1k
  learning_rate, neighbourhood_range = decay(step,␣
↪max_steps,max_learning_rate,max_m_dsitance)

  lr.append(learning_rate)
  nr.append(neighbourhood_range)

  t = np.random.randint(0,high=train_x_norm.shape[0]) # random index of traing␣
↪data
  winner = winning_neuron(train_x_norm, t, som, num_rows, num_cols)
  for row in range(num_rows):
    for col in range(num_cols):
      if m_distance([row,col],winner) <= neighbourhood_range:
        som[row][col] += learning_rate*(train_x_norm[t]-som[row][col])

  if (step ==0) or (step < 1001 and (step+1) %50 ==0) or (step+1) % 1000 == 0:
    # data labelling
    label_data = train_y
    map = np.empty(shape=(num_rows, num_cols), dtype=object)
    for row in range(num_rows):
      for col in range(num_cols):
        map[row][col] = [] # empty list to store the label
        #for t in range(data.shape[0]):
    for t in range(train_x_norm.shape[0]):
      winner = winning_neuron(train_x_norm, t, som, num_rows, num_cols)
      map[winner[0]][winner[1]].append(label_data[t]) # label of winning neuron
    label_map = np.zeros(shape=(num_rows, num_cols),dtype=np.int64)
    for row in range(num_rows):
      for col in range(num_cols):
        label_list = map[row][col]
        if len(label_list)==0:
          label = 2
        else:
          label = max(label_list, key=label_list.count)
        label_map[row][col] = label
    current_step = step + 1
    title = ax.text(0.5, 1.01, 'Iteration '+str(current_step)+' (max␣
↪'+str(max_steps)+' iteration)' , ha='center', va='bottom',transform=ax.
↪transAxes, fontsize='large')
    #image = ax.imshow(map_label, cmap=plt.cm.get_cmap('Accent',3))
    image = ax.imshow(label_map, cmap=cmap)
    if not cbar_initialized:
        cbar_initialized = True # initialise the colour bar only once
        fig.colorbar(image, ax=ax)
    images.append([image]+[title])
```

```
# Generate the animation image and save
animated_image = animation.ArtistAnimation(fig, images)
animated_image.save('./som_training.gif', writer='pillow')

print("SOM training completed")
```

```
Started at:  14:22:13
```

[ ]:
```
data = minmax_scaler(test_x) # normalisation

winner_labels = []

for t in range(data.shape[0]):
 winner = winning_neuron(data, t, som, num_rows, num_cols)
 row = winner[0]
 col = winner[1]
 predicted = label_map[row][col]
 winner_labels.append(predicted)

print("Accuracy: ",accuracy_score(test_y, np.array(winner_labels)))
```

```
Accuracy:  1.0
```

[ ]:

[ ]: