1. IN LAB: MIN-MAX Composition
   --POST: Union and Intersection

2. IN LAB: a. Fuzzy Membership (Tall, very tall….)
   b. Gaussian method (White, Moderate, Black)
   --POST: Gaussian Method(Warm, Moderate,..)

3. IN LAB: Trapezoid Membership fn and defuzzify it
   --POST: fuzzification and Defuzzification using Gaussian

4. Implement various activation functions and visualize the results.

5. IN: ANDNOT gate using McCulloch-Pitts neural model.
   --POST: XOR using McCulloch-Pitts neural model.

6. HEBB Rule

7. IN: Perceptron (Sohan)
   --POST: Perceptron Sum

8. XOR gate neural Network

9. IN LAB: Auto Associative Network (SOM)
   --POST: SOM Sums

10. IN LAB: Genetic Algorithms (Sai Passes)
    --POST: Random String from Gene Set (a to z, A to Z)

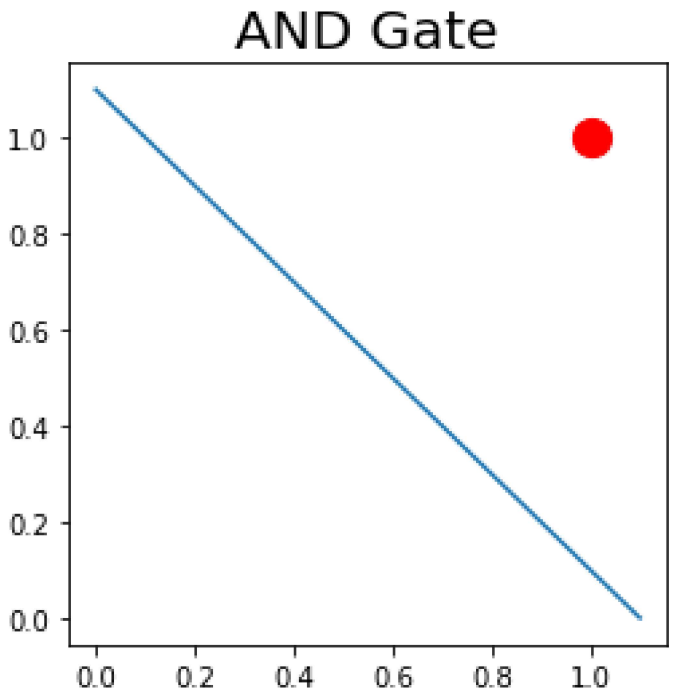
11. Netherlands Genetic Algorithm Population

12. Robot Motion (PSO)

13. Kriya problem (Travelling Salesperson with PSO)

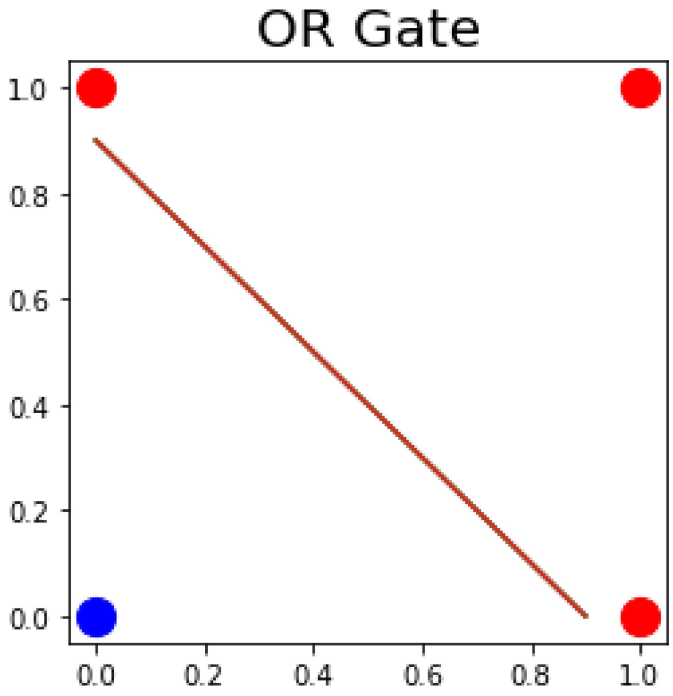
2 logical gates implementation

[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()
```



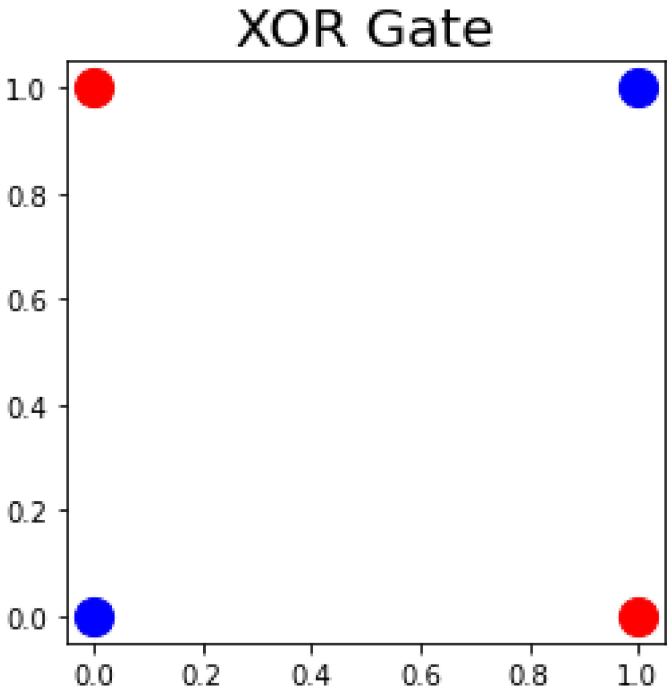
[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
x=np.array([[0,0],[0,1],[1,0],[1,1]])
```

```
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()
```

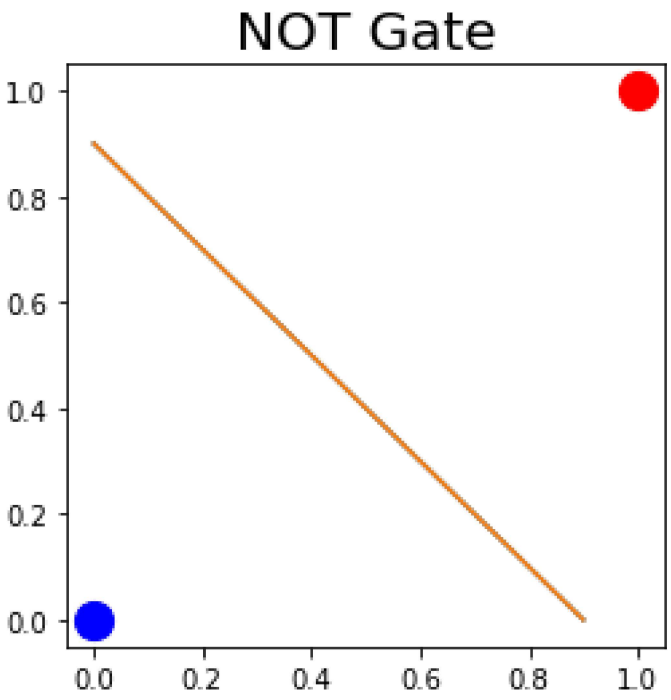


[3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
x=np.array([[0,0],[0,1],[1,0],[1,1]])
y=[0,1,1,0]
area = 200
fig = plt.figure(figsize=(4,4))
plt.title('XOR Gate', fontsize=20)
# color red: is class 1 and color blue is class 0.
for i in range(4):
```

```
    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 MIN MAX PRODUCT

[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 UNION AND INTERSECTION

[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]
```

print("intersection : ", np.intersect1d(arr1,arr2))

OUTPUT: intersection : [4 7]

LAB 2

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

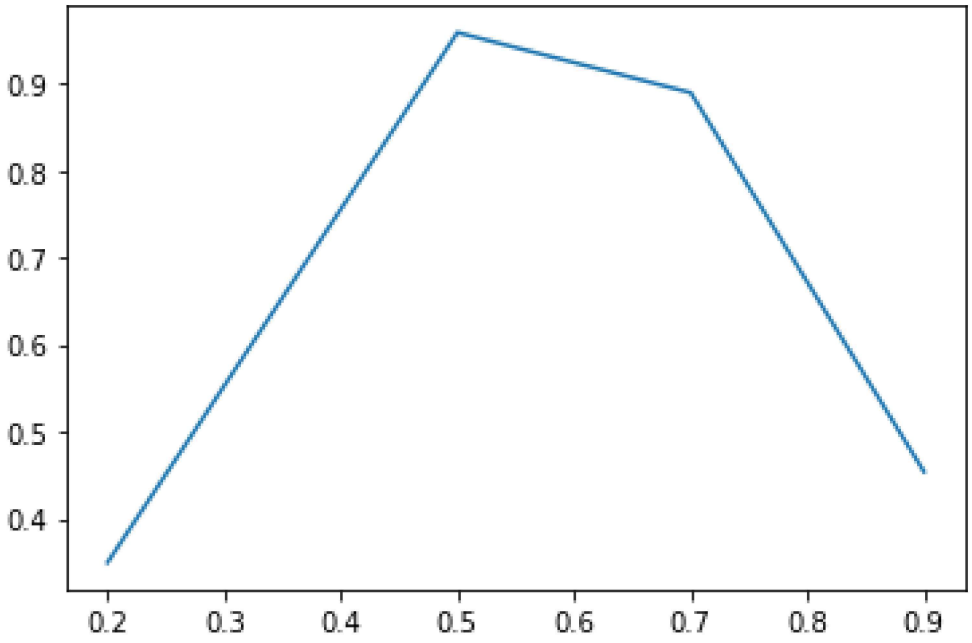
```
import numpy as np
import skfuzzy as fuzz
import matplotlib.pyplot as plt
start = 0
stop = 10 + 0.001
step = 0.25
x = np.arange(start, stop, step)
trapmf = fuzz.trapmf(x, [0, 2, 8, 10])

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

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'
```

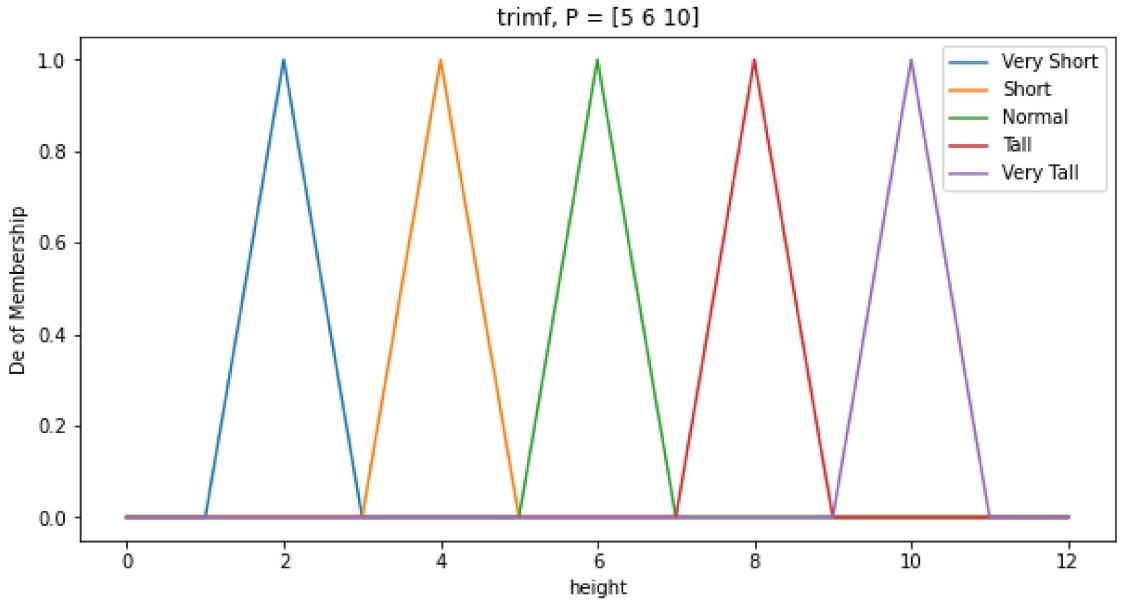
[ ]:
```
# let the values be [0.2,0.5,0.7,0.9]
arr = np.array([0.2,0.5,0.7,0.9])
mf = fuzz.gaussmf(arr,np.mean(arr),np.std(arr))
plt.plot(arr,mf)
plt.show()
```



IN LAB (Vert tall, tall, normal, short, Very Short)

[ ]:
```
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?
```

[ ]:
```
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()
```



SC_LAB-3

1 Pre-Lab

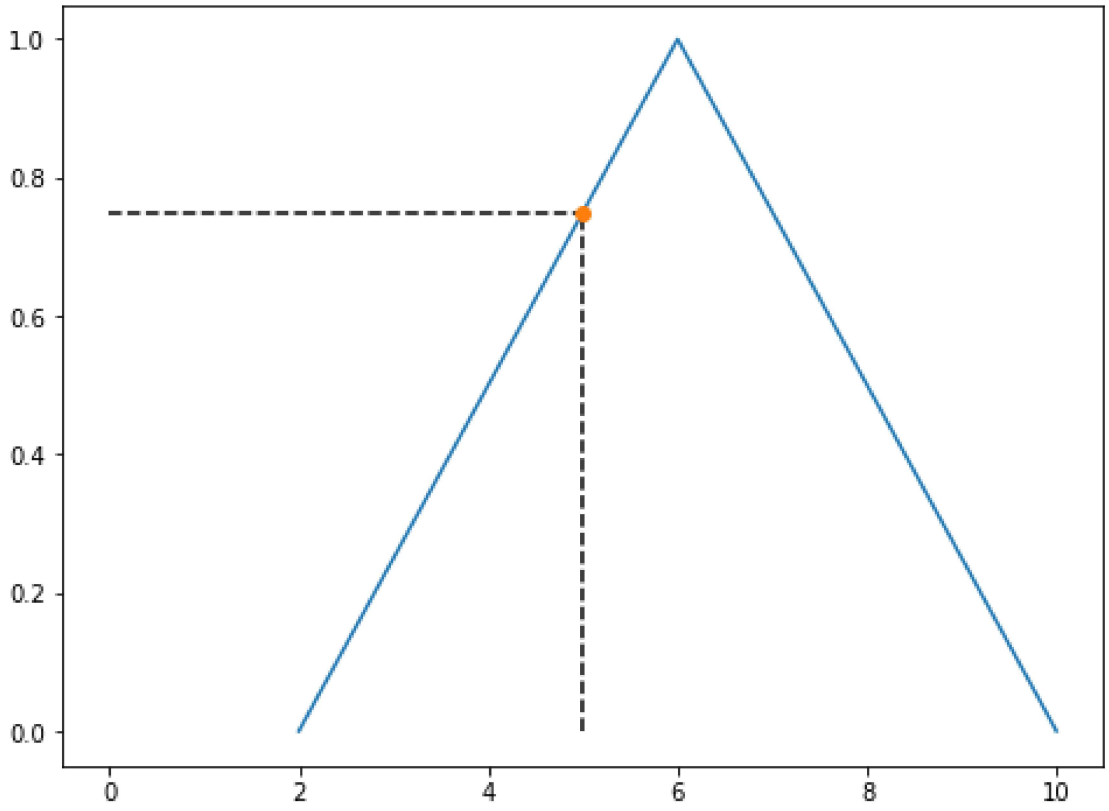
4. Write a python code to implement fuzzification using triangular membership function and then defuzzify by using the centroid. (this solution is implemented in next cell)

[ ]:
```
#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.
"""
```

```
#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)
```

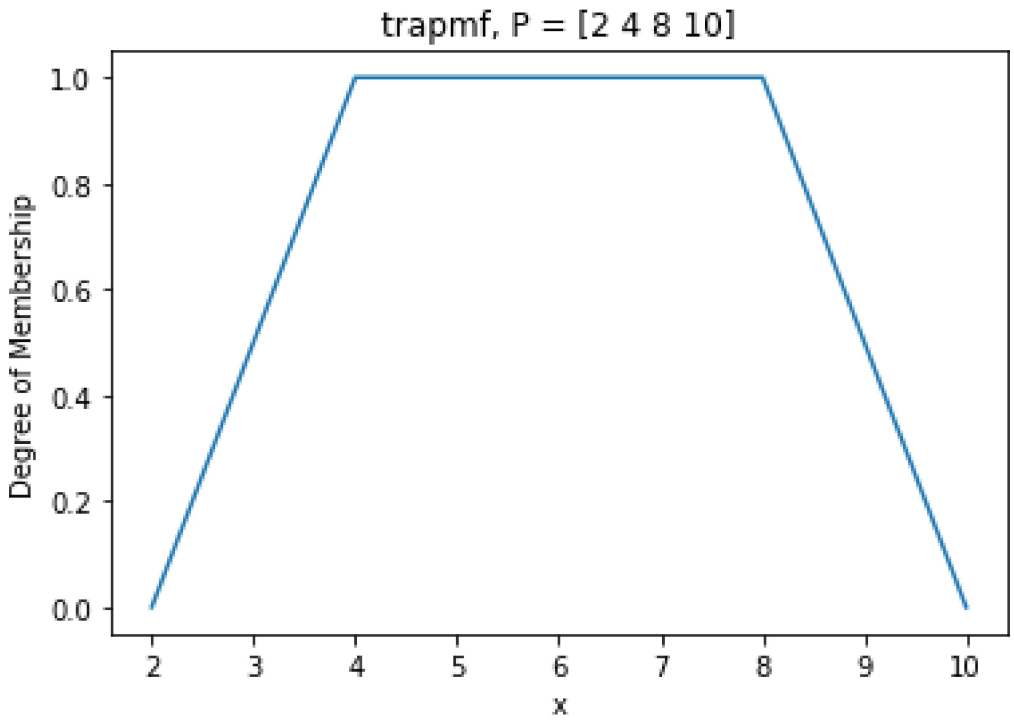
```
3
0.5
```

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

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

[ ]:
```
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

LAB-4

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]:
```
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
```

IN-LAB

LAB - 5

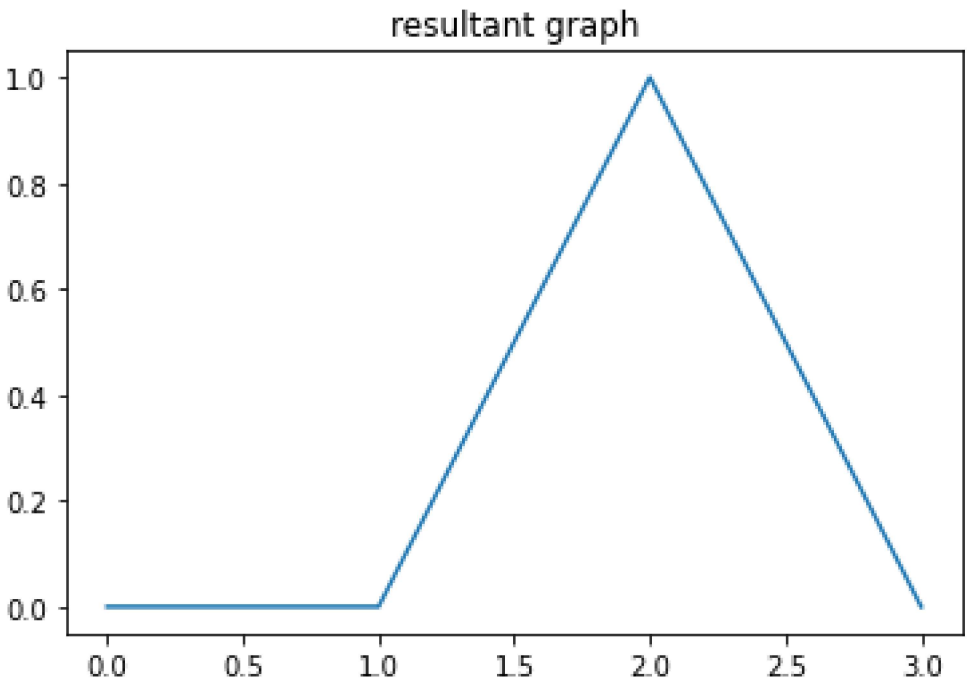
```
def compute_Neuron(x1,x2,w1,w2):          # implementation of andnot logic using mcculloch pitts neuron
    res = []
    for i in range(len(x1)):
        res.append(x1[i]*w1+x2[i]*w2)
```

```
    #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

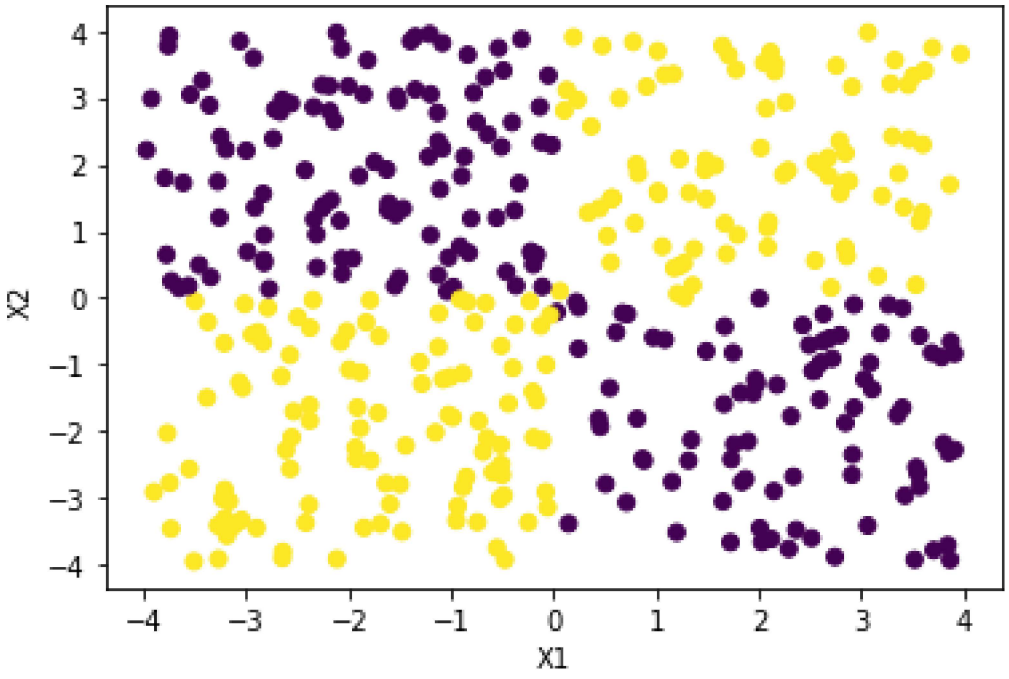
[14]:
```
'''Implement XOR logic gate using McCulloch-Pitts neural model. This neural␣
 ↪model will
deal with the XOR logic'''

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])
```

OUTPUT:

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

XOR Visualization

[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()
```



LAB - 6

Using the Hebb Rule, implement a program to find the weights required to perform the following classifications of the given input patterns below. The pattern is shown as 4*5 matrix form in the square. The “+” symbols represent the value “1” and empty square braces represent “-1”. Consider “S” belongs to the members of class hence its target value is 1 and “C” does not belong to the members of the class hence its target value is “-1”

[11]:
```
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)
```

OUTPUT:
Updated weights are: [[0. 0. 0. 0. 0. 0. 0. 0. 0.
100. 100. 100.
-100. 0. 0. 100. 0. 0. 0. 0.]]

POST LAB

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

```
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 ]
```

LAB - 7

Pre-lab

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 IN - LAB (Sohan Question)

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

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

[23]:
```
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]:
```
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]:
```
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
```

LAB 7 POST LAB

[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))
```

Question for above program (LAB 7 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.'

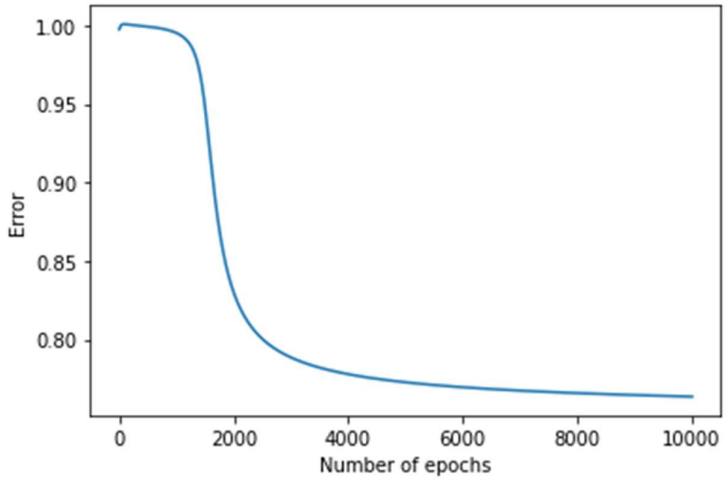
8: XOR gate Neural Network

```
"""Write a neural network program to implement XOR function using back
propagation algorithm."""

import numpy as np
import matplotlib.pyplot as plt

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

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

epochs = 10000
lr = 0.1
input_neurons = 2
hidden_neurons = 1
output_neurons = 1

wh = np.random.uniform(size=(input_neurons, hidden_neurons))#weight matrix
between input and hidden layer
wo = np.random.uniform(size=(hidden_neurons, output_neurons))# weight matrix
between hidden and output layer
bh = np.random.uniform(size=(1, hidden_neurons))
bo = np.random.uniform(size=(1, output_neurons))

#Backpropagation
error = []
for i in range(epochs):
    #Forward Propagation
    hidden_layer_input1 = np.dot(X, wh)
    hidden_layer_input = hidden_layer_input1 + bh
    hidden_layer_activations = sigmoid(hidden_layer_input)
    output_layer_input1 = np.dot(hidden_layer_activations, wo)
    output_layer_input = output_layer_input1+ bo
    output = sigmoid(output_layer_input)

    #Backpropagation
    E = y-output
    slope_output_layer = sigmoid_derivative(output)
    slope_hidden_layer = sigmoid_derivative(hidden_layer_activations)
    d_output = E * slope_output_layer
    Error_at_hidden_layer = d_output.dot(wo.T)
    d_hiddenlayer = Error_at_hidden_layer * slope_hidden_layer
    wo += hidden_layer_activations.T.dot(d_output) *lr
    wh += X.T.dot(d_hiddenlayer) *lr
```

```
    error.append(np.mean(np.abs(E)))
    bo += np.sum(d_output, axis=0,keepdims=True) *lr
    bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr



print("Error: \n" + str(np.mean(np.square(y - output))))

# Plot the error
plt.plot(error)
plt.xlabel("Number of epochs")
plt.ylabel("Error")
plt.show()
```

OUTPUT:

Error: 0.751653470770866




```
def predict(x):
    hidden_layer_input1 = np.dot(x, wh)
    hidden_layer_input = hidden_layer_input1 + bh
    hidden_layer_activations = sigmoid(hidden_layer_input)
    output_layer_input1 = np.dot(hidden_layer_activations, wo)
    output_layer_input = output_layer_input1+ bo
    output = sigmoid(output_layer_input)
    if output > 0.5:
        return 1
    else:
        return -1

print("The prediction is: ", predict(np.array([1,1])))
print("The prediction is: ", predict(np.array([1,-1])))
print("The prediction is: ", predict(np.array([-1,1])))
print("The prediction is: ", predict(np.array([-1,-1])))
```

OUTPUT:

```
The prediction is: -1
The prediction is: 1
The prediction is: 1
The prediction is: -1
```

9.Auto Associative Networks

```
"""
 Train the auto associative network for the input vector [-1 1 1 1] and also
test the
network for
the same input vector. Test the auto associative network with:
• one missing entry
• one wrong entry
• Two wrong entry
• Two missing entries in the test vector.
"""
```

```
import numpy as np
import pandas as pd

x = np.array([[1, 0, 1, 1]])
xT = x.T

w =  np.zeros((4,4))
w = w + xT.dot(x)
w
```

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

```
1  #Testing the network for the same input vector
2
3  y = x.dot(w)
4  print(y)
5
6  #Applying activtion function
7  y_c = np.where(y>0,1,0)
8  print(y_c)
9
10 if (y_c == x).all():
11     print("It is a known vector")
12 else:
13     print("It is an unknown vector")
```

✓ 0.4s Python

```
[[3. 0. 3. 3.]]
[[1 0 1 1]]
It is a known vector
```

```python
# Testing with the one missing entry

x1 = np.array([[0, 0, 1, 1]])
y = x1.dot(w)
print(y)

y_c = np.where(y>0,1,0)
print(y_c)

if (y_c == x).all():
    print("It is a known vector")
else:
    print("It is an unknown vector")
```

Python

```
[[2. 0. 2. 2.]]
[[1 0 1 1]]
It is a known vector
```

```python
#Testing with the one wrong entry

x1 = np.array([[1, 1, 1, 1]])
y = x1.dot(w)
print(y)

y_c = np.where(y>0,1,0)
print(y_c)

if (y_c == x).all():
    print("It is a known vector")
else:
    print("It is an unknown vector")
```

Python

```
[[3. 0. 3. 3.]]
[[1 0 1 1]]
It is a known vector
```

```python
#Testing with the two wrong entry

x1 = np.array([[1, 1, 0, 1]])
y = x1.dot(w)
print(y)

y_c = np.where(y>0,1,0)
print(y_c)

if (y_c == x).all():
    print("It is a known vector")
else:
    print("It is an unknown vector")
```

Python

```
[[2. 0. 2. 2.]]
[[1 0 1 1]]
It is a known vector
```

```python
#Testing with the two wrong entry

x1 = np.array([[1, 0, 0, 0]]) #[0,0,0,0] to get unknown vector
y = x1.dot(w)
print(y)

y_c = np.where(y>0,1,0)
print(y_c)

if (y_c == x).all():
    print("It is a known vector")
else:
    print("It is an unknown vector")
```

Python

```
[[1. 0. 1. 1.]]
[[1 0 1 1]]
It is a known vector
```

POST LAB:

```
"""
Write a python program to find the weight matrix of an hetero associative net to store the
vectors x1= [1 1 1 1 1],x2 =[1 -1 -1 1 -1],x3=[-1 1 -1 -1 -1]. Test the response of the
network by presenting the same pattern and recognize whether it is a known vector or
unknown vector.
"""
```

```python
import numpy as np
import pandas as pd

x= np.array([[1,1,1,1,1],[1,-1,-1,1,-1],[-1,1,-1,-1,-1]])
x_t = x.T

w = np.zeros((5,1))
w = w + np.dot(x_t,x)
#w = w - np.diag(np.diag(w))
w
```

Python

```
array([[ 3., -1.,  1.,  3.,  1.],
       [-1.,  3.,  1., -1.,  1.],
       [ 1.,  1.,  3.,  1.,  3.],
       [ 3., -1.,  1.,  3.,  1.],
       [ 1.,  1.,  3.,  1.,  3.]])
```

```python
#Testing with the same pattern

x1 = np.array([[1,1,1,1,1],[1,-1,-1,1,-1],[-1,1,-1,-1,-1]])
y= np.dot(x1,w)
print(y)
y_c = np.where(y>0,1,0)
print(y_c)

if (y_c == x).all():
    print("It is a known vector")
else:
    print("It is an unknown vector")
```

✓ 0.4s Python

```
[[ 7.  3.  9.  7.  9.]
 [ 5. -7. -5.  5. -5.]
 [-9.  3. -7. -9. -7.]]
[[1 1 1 1 1]
 [1 0 0 1 0]
 [0 1 0 0 0]]
It is an unknown vector
```

10. Implementation of Genetic algorithms

```
import numpy as np
import random
from string import ascii_lowercase,ascii_uppercase
class Genetic_Algorithm:
    def __init__(self,arr,par1,par2):
        self.arr = arr
        self.par1 = par1
        self.par2 = par2
    def mutation(self):
        pos = random.randint(0,self.arr.size-1)
        self.arr[pos] = self.arr[pos]^1

    def crossover(self):
        self.x = self.par1.size//2
        if(random.randint(0,1) == 1):
            self.baby = np.concatenate((self.par1[:self.x],self.par2[self.x:]),axis = 0)
        else:
            self.baby = np.concatenate((self.par2[:self.x],self.par1[self.x:]),axis = 0)
        return self.baby
    def get_arr(self):
        return self.arr
    def get_par1(self):
        return self.par1
    def get_par2(self):
        return self.par2
    def get_baby(self):
        return self.baby
```

```
    def Hello_World(self):
        self.gene_set = ascii_lowercase+' '+ascii_uppercase


lab =
Genetic_Algorithm(np.array([1,1,1,0,0,1]),np.array([1,1,0,1,1,0,1]),np.array([0,1,0,1,1,1,0]
))
print("intial")
print(lab.get_arr())
print("mutation")
lab.mutation()
print(lab.get_arr())
print("crossover")
print("parent1",lab.get_par1())
print("parnent",lab.get_par2())
lab.crossover()
print("result/baby",lab.get_baby())
lab.Hello_World()
```

OUTPUT:

```
intial
[1 1 1 0 0 1]
mutation
[1 0 1 0 0 1]
crossover
parent1 [1 1 0 1 1 0 1]
parnent [0 1 0 1 1 1 0]
result/baby [0 1 0 1 1 0 1]
```

POST LAB 10:

Write a python program to generate a random string from the gene set. Gene Set: "

abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"

Target: "Hello World!".

Code:

```
import random
import datetime


def generate_parent(length, geneSet):
    genes = []
    while len(genes) < length:
        sampleSize = min(length - len(genes), len(geneSet))
        genes.extend(random.sample(geneSet, sampleSize))
    return ''.join(genes)


def mutate_string(parent, geneSet, mutationRate):
    childGenes = list(parent)
    for i in range(len(parent)):
        if random.random() < mutationRate:
            childGenes[i] = random.choice(geneSet)
    return ''.join(childGenes)

def display(guess, fitness, startTime):
    timeDiff = datetime.datetime.now() - startTime
    fitnessPercent = fitness * 100 / len(guess)
    print("{0}\t{1}\t{2}".format(guess, fitnessPercent, str(timeDiff)))

def get_fitness(guess, target):
    return sum(1 for expected, actual in zip(target, guess)
               if expected == actual)


def main():
    random.seed()
    startTime = datetime.datetime.now()
    geneSet = " abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
    target = "Hello World"
    bestParent = generate_parent(len(target), geneSet)
    bestFitness = get_fitness(bestParent, target)
    display(bestParent, bestFitness, startTime)
    while True:
```

```
        child = mutate_string(bestParent, geneSet, 0.01)
        childFitness = get_fitness(child, target)
        if bestFitness >= childFitness:
            continue
        display(child, childFitness, startTime)
        if childFitness >= len(bestParent):
            break
        bestFitness = childFitness
        bestParent = child

if __name__ == '__main__':
    main()
```

OUTPUT:

Hello World 100.0 0:00:00.084964

11. Genetic Algorithms Population

```
import numpy as np
import random
from string import ascii_lowercase,ascii_uppercase
```

```
class Genetic_Algorithm:
    def __init__(self,arr,par1,par2):
        self.arr = arr
        self.par1 = par1
        self.par2 = par2
    def mutation(self):
        pos = random.randint(0,self.arr.size-1)
        self.arr[pos] = self.arr[pos]^1

    def crossover(self):
        self.x = self.par1.size//2
        if(random.randint(0,1) == 1):
            self.baby =
np.concatenate((self.par1[:self.x],self.par2[self.x:]),axis = 0)
        else:
            self.baby =
np.concatenate((self.par2[:self.x],self.par1[self.x:]),axis = 0)
        return self.baby
    def get_arr(self):
        return self.arr
    def get_par1(self):
        return self.par1
    def get_par2(self):
        return self.par2
    def get_baby(self):
        return self.baby
    def Hello_World(self):
        self.gene_set = ascii_lowercase+' '+ascii_uppercase
```

```
lab =
Genetic_Algorithm(np.array([1,1,1,0,0,1]),np.array([1,1,0,1,1,0,1]),np.array([
0,1,0,1,1,1,0]))
print("intial")
print(lab.get_arr())
print("mutation")
lab.mutation()
print(lab.get_arr())
print("crossover")
print("parent1",lab.get_par1())
print("parnent",lab.get_par2())
lab.crossover()
```

```
print("result/baby",lab.get_baby())
lab.Hello_World()
```

OUTPUT:

```
intial
[1 1 1 0 0 1]
mutation
[1 1 1 0 1 1]
crossover
parent1 [1 1 0 1 1 0 1]
parnent [0 1 0 1 1 1 0]
result/baby [1 1 0 1 1 1 0]
```

12. ROBOT MOTION

```
"""Robot motion planning involves finding the path a robot needs to traverse
from a given start
point to a given end point while avoiding obstacles along the way find the
suitable method to
solve this problem and write the python code for that method
"""
```

```
def isEscapePossible(blocked, source, target):
        blocked = {tuple(p) for p in blocked}

        def bfs(source, target):
            bfs, seen = [source], {tuple(source)}
            for x0, y0 in bfs:
                for i, j in [[0, 1], [1, 0], [-1, 0], [0, -1]]:
                    x, y = x0 + i, y0 + j
                    if 0 <= x < 10**6 and 0 <= y < 10**6 and (x, y) not in
seen and (x, y) not in blocked:
                        if [x, y] == target: return True
                        bfs.append([x, y])
                        seen.add((x, y))
                if len(bfs) == 20000: return True
            return False
        return bfs(source, target) and bfs(target, source)
```

Input and Output: