Dr Br Ambedkar National Institute Of Technology

Soft Computing Concepts Lab

LAB

Name - Hashwanth Sutharapu

Branch- Information Technology

Roll No.- 19124025

Group-G1

Submitted To- Dr Neeraj Kumar

File Submission Date-23/02/2022

Index

S.No	Name of Experiment	Date	Remark
1	Install python and import numpy library.	14/02/2022	
2	Implementing linear saturating function/sigmoid function.	21/02/2022	
3	Write a program to implement logic gates OR,AND,XOR	28/02/2022	
4	Write a program to compute the amount of price/tip to be given to a guide while visiting a Historical site.Use any two parameters to compute the tip.	07/03/2022	
5	Write a program to Hebb rule for AND gate	17/03/2022	
6	Write a program to Delta rule for AND gate.	24/03/2022	
7	Implementation Genetic Algorithm.	31/04/2022	

Assignment 1 Date:14/02/2022

Install Python with numpy

AIM:
Install python and import numpy library.

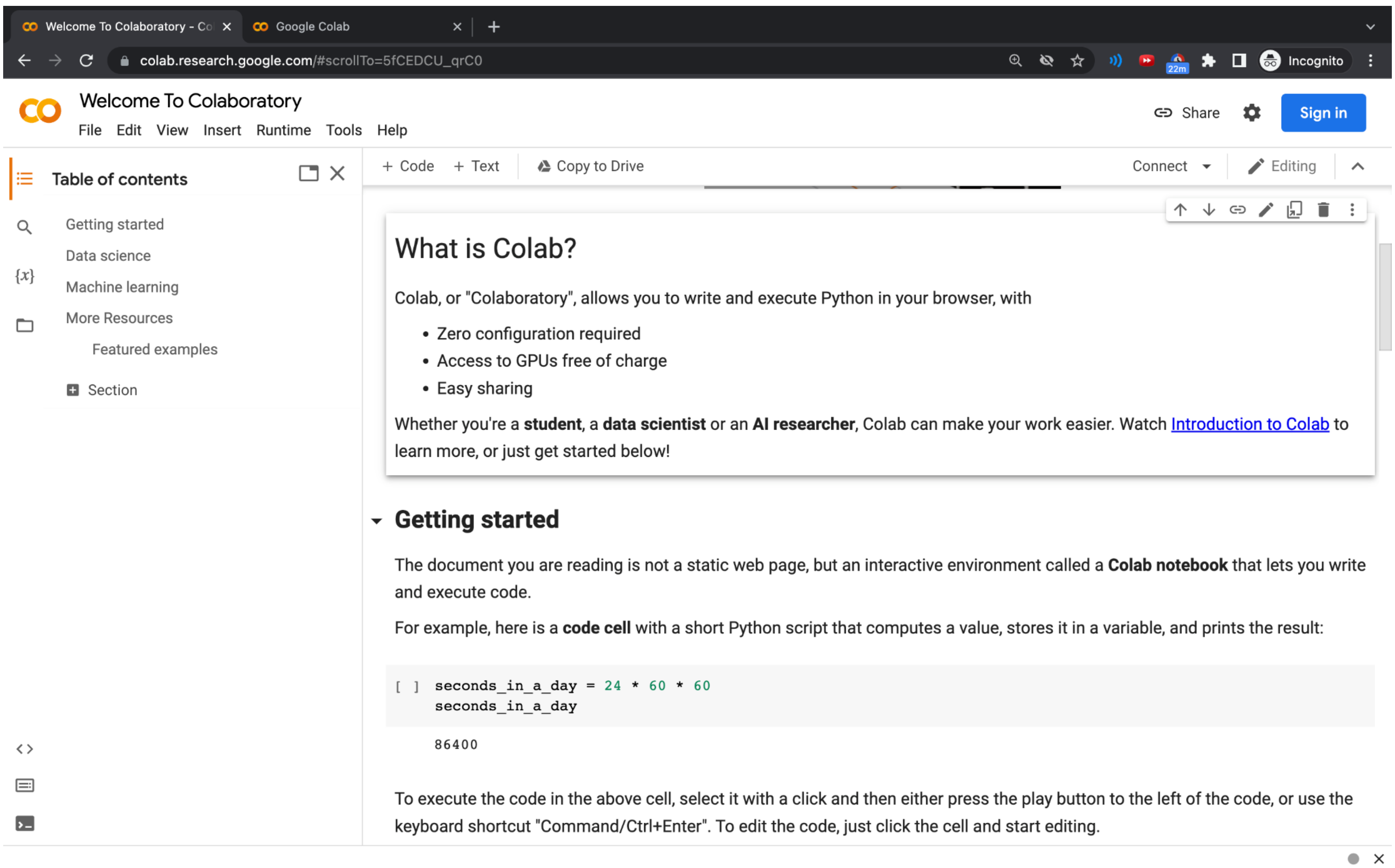
THEORY:

Python is a high-level programming language with versatile utilities such as web development using Django framework, task automation, data science, visualizing data etc.It supports various programming paradigms such as object oriented, procedural, functional and structural programming etc.

Numpy is a python library containing numerous mathematical functions, visualization tool such as plots, scatter diagrams, fourier diagrams etc.

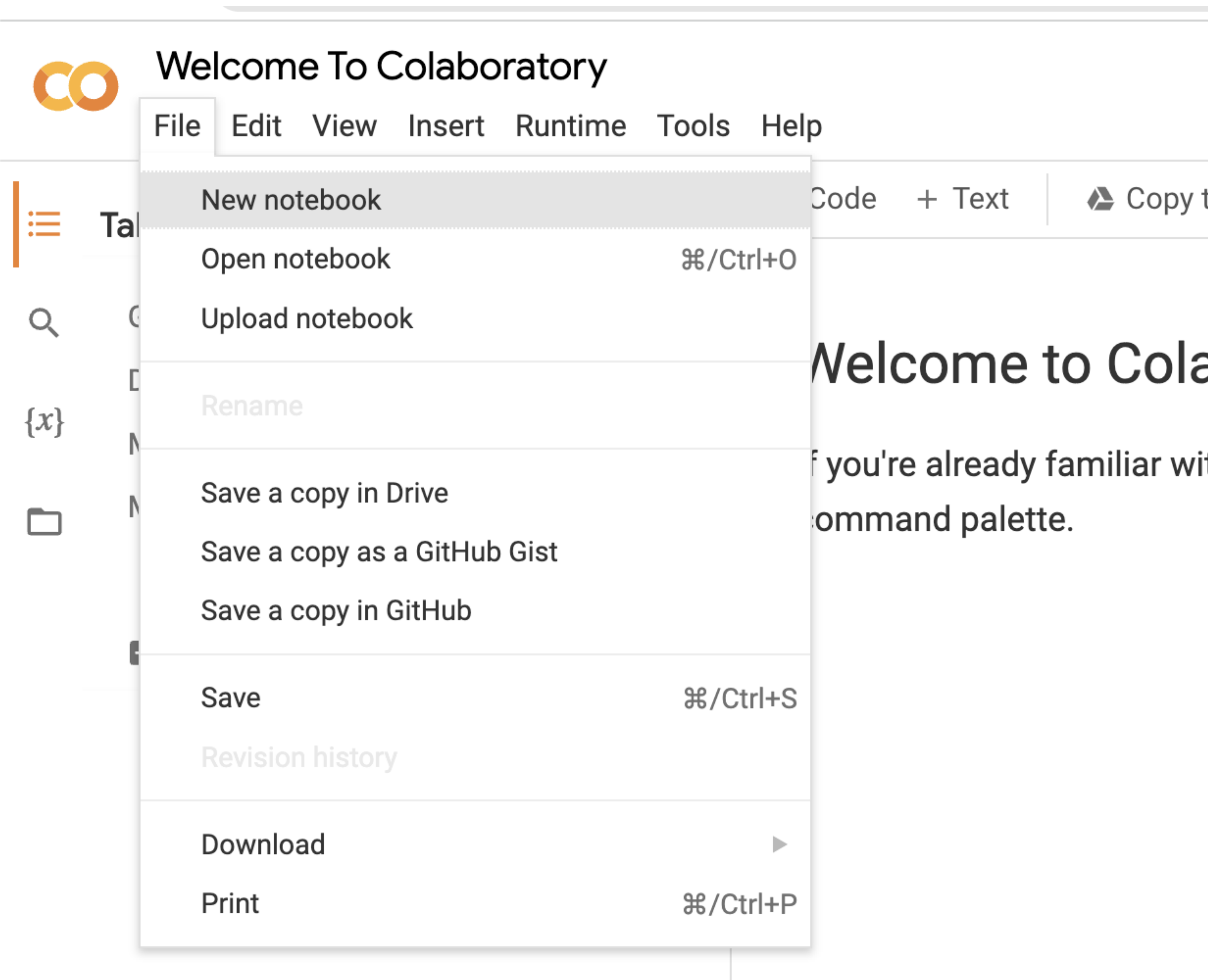
Steps to use Python in Google Colab:

Step-1:
Open the Google Colab using any web browser.



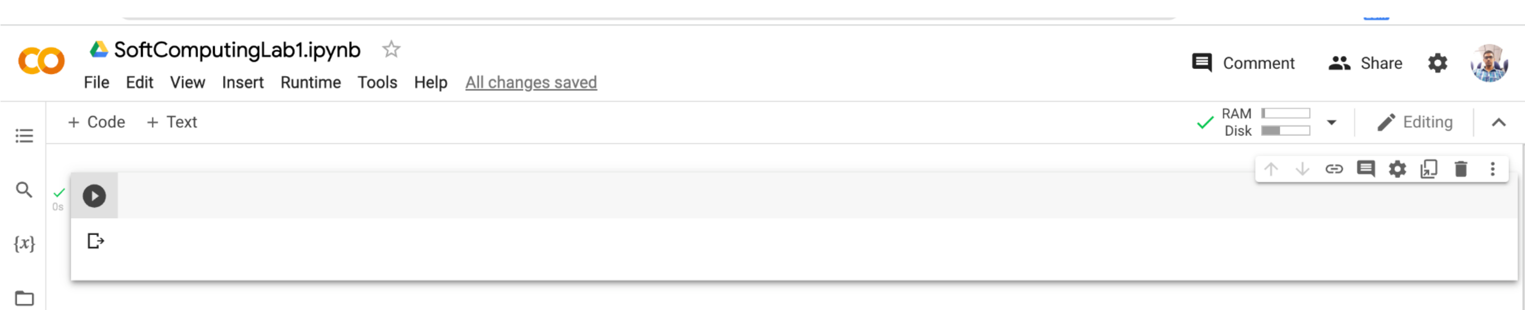
Step-2:

Click on the file button and select new notebook option.



Step-3:

A new note book will be opened with an editor to write and implement python code.



Step-4:

Write a sample python code such as a print statement and run the code using the run button.

From the above screen shot it can be seen that the roll number and name are printed using print statements.

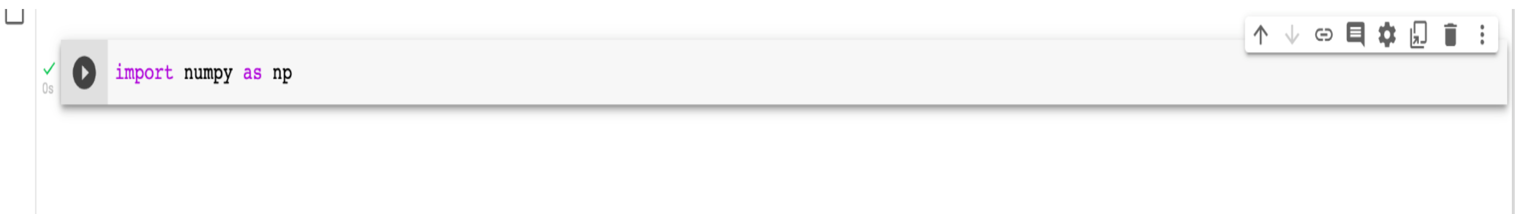
Steps to install numpy library and use it:

Step-1:

Install numpy library using pip.The command to use is “pip install numpy”.

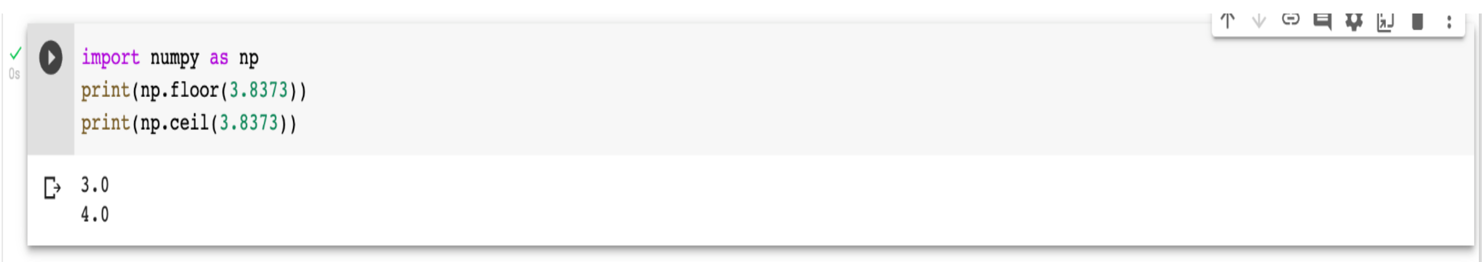
Step-2:

Inorder to use the numpy library import it using command “import numpy as np”.



Step-3:

Use one of the functions to demonstrate the numpy library usage



We have used the floor and ceil of numpy library, which rounds of the figure to upper number or a lower number.

Assignment 2 Date:21/02/2022

Sigmoid Function

AIM:

Implementing linear saturating function/sigmoid function.

THEORY:

Activation Function

In neural networks activation function is used to get output of node.It is also known as the transfer function.
There are two types of activation functions, Linear activation functions and non linear activation function.

The range of linear activation function is unbounded i.e [-infinity to +infinity].

Non linear activation functions are most popular and frequently used, such as Sigmoid or Logistic activation function, Tanh or hyperbolic logic function, Rectified Linear Unit(RELU) activation function, leaky RELU function etc.

Sigmoid or Logistic activation function

$$f(x) = \frac{1}{1+e^{-x}}$$

It is significant as it’s range lies between 0 and 1.Incidentally the range of probability is also [0,1] therefore sigmoid function can be used in model where we have to predict the probability.

Sigmoid function is monotonic and differentiable i.e we can find slope of sigmoid curve between two points.

Derivative of Sigmoid function

$$f'(x) = f(x)(1-f(x))$$

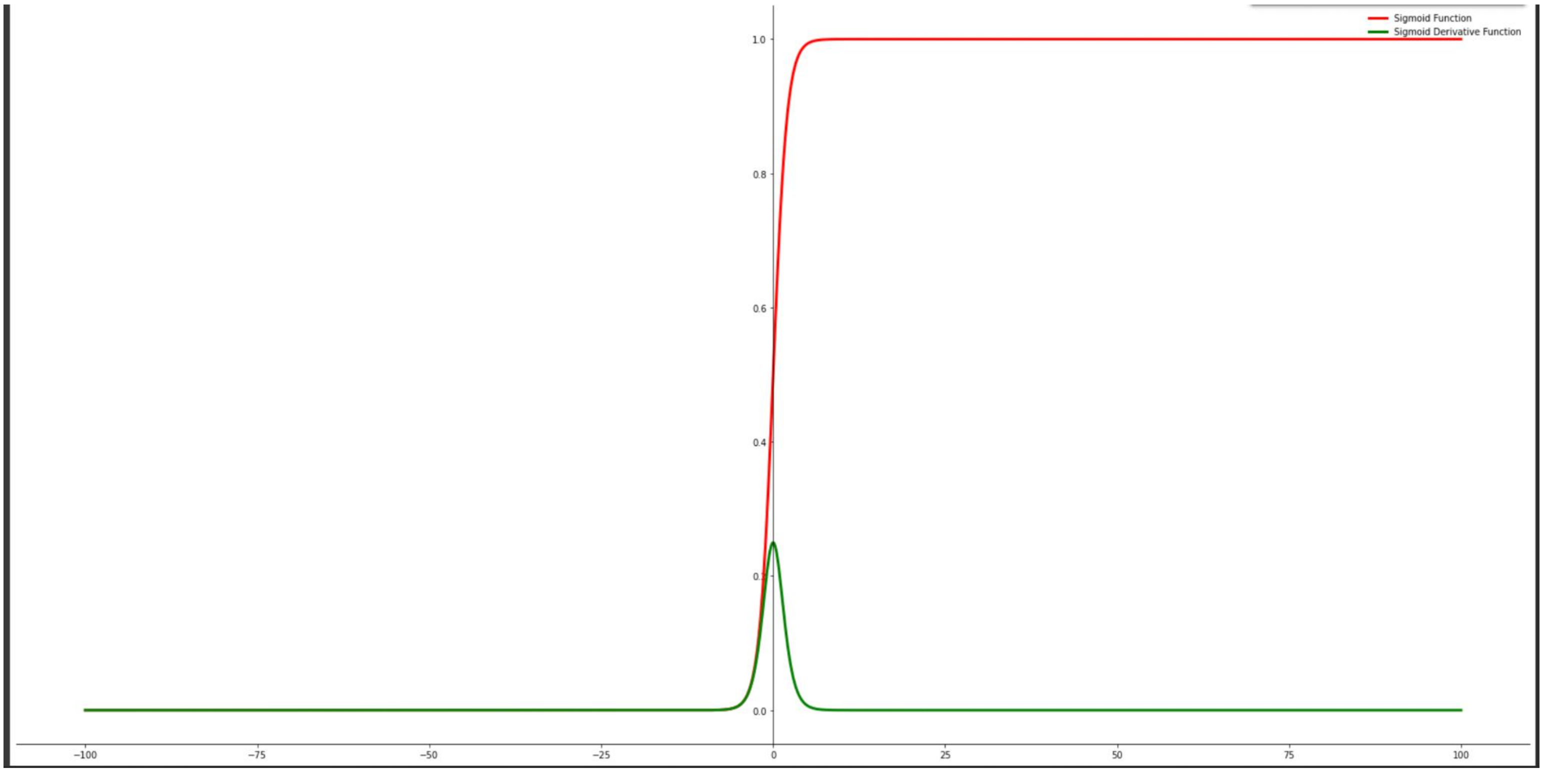
The Derivative of Sigmoid function is not a monotonic function.

Code:

sigmoid.py

```
#Importing Libraries
import matplotlib.pyplot as plt
import numpy as np
import math
#Importing Libraries
x = np.linspace(-100, 100, 1000)
#Sigmoid function
def sigmoid(x):
 s=1/(1+np.exp(-x)) #Sigmoid function
 ds=s*(1-s)        #Derivative of Sigmoid function
 return s,ds
# Setup centered axes
fig, ax = plt.subplots(figsize=(30, 15))
ax.spines['left'].set_position('center')
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')# Create and show plot
## call Sigmoid function for sigmoid
ax.plot(x,sigmoid(x)[0], color="#307EC7", linewidth=3, label="Sigmoid Function")
## call Sigmoid function for derivative sigmoid fun
ax.plot(x,sigmoid(x)[1], color="#9621E2", linewidth=3, label="Sigmoid Derivative Function")
ax.legend(loc="upper right", frameon=False)
plt.show()
```

Output



Sigmoid Function
Sigmoid Derivative Function
1.0
0.8
0.6
0.4
0.0
−100
−75
−50
−25
0
25
50
75
100

Assignment 3 Date:28/02/2022

Logic Gates(OR,AND,XOR)

AIM:
Write a program to implement logic gates OR,AND,XOR

THEORY:
Logic gates such as AND, OR, NAND,XOR, etc have wide range of applications in the daily life beginning from switches in electricity to microprocessors used in computers etc.Decision making step which is most crucial is done with the help of logic gates

Code:

LogicGates.py

```
#--OR function
def OR(a,b):
 if(a!=0 or b!=0):
   return "true"
 return "false"
#--AND function
def AND(a,b):
 if(a==0 or b==0):
   return "false"
 return "true"
#--XOR function
def XOR(a,b):
 if((a==0 and b==0) or (a!=0 and b!=0)):
   return "false"
 return "true"
#--Main function
#Taking input of 2 numbers
a=input("Enter 1st number:");
b=input("Enter 2nd number:");
#convert a,b to integers
a=int(a)
b=int(b)
print("OR of given "+str(a)+" and "+str(b)+" is "+OR(a,b));
print("AND of given "+str(a)+" and "+str(b)+" is "+AND(a,b));
print("XOR of given "+str(a)+" and "+str(b)+" is "+XOR(a,b));
```

Output

```
Enter 1st number:2
Enter 2nd number:0
OR of given 2 and 0 is true
AND of given 2 and 0 is false
XOR of given 2 and 0 is true
```

```
Enter 1st number:0
Enter 2nd number:0
OR of given 0 and 0 is false
AND of given 0 and 0 is false
XOR of given 0 and 0 is false
```

Assignment 4 Date:21/02/2022

Tipping Problem using Fuzzy Logic

AIM:

Write a program to compute the amount of price/tip to be given to a guide while visiting a Historical site.Use any two parameters to compute the tip.

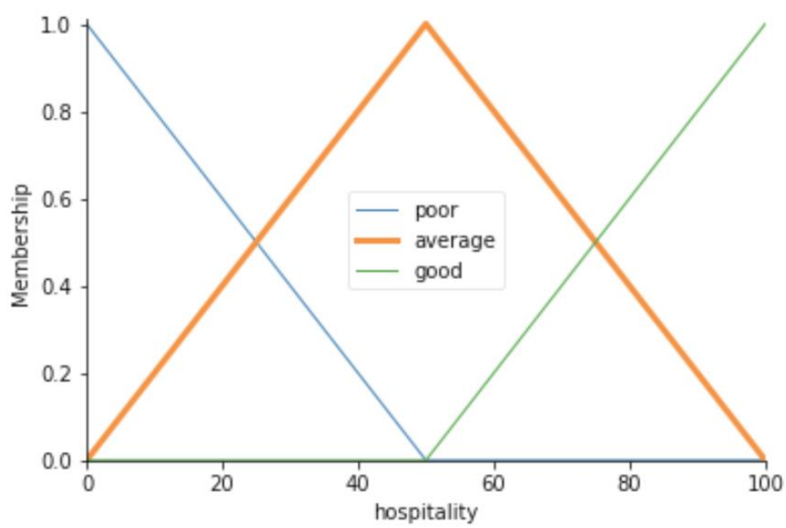
Code:

tipping.py

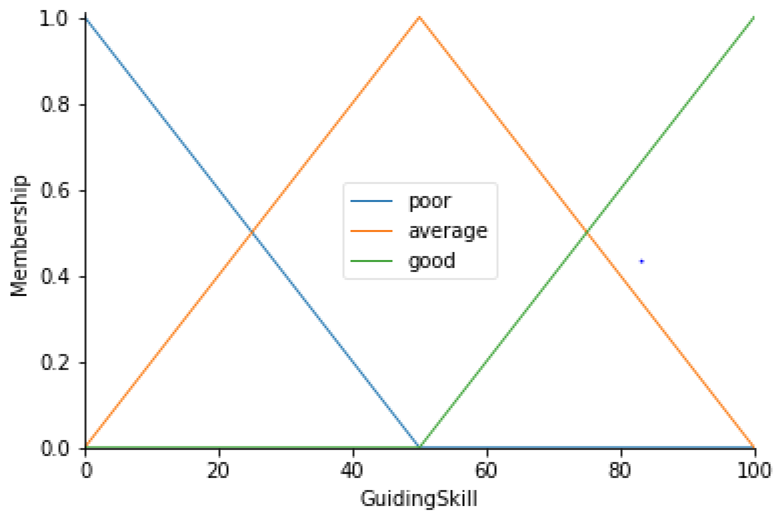
```
pip install scikit-fuzzy #installation of fuzzy package
```

```
#import required libraries
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
#set parameters and range of ratings
hospitality = ctrl.Antecedent(np.arange(0, 101, 1), 'Hospitality')
GuidingSkill = ctrl.Antecedent(np.arange(0, 101, 1), 'Guidingskill')
price = ctrl.Consequent(np.arange(0, 3001, 1), 'price')
# auto membership
hospitality.automf(3)
GuidingSkill.automf(3)
# ------------Python API for fuzzy implementation--------------
price['low'] = fuzz.trimf(price.universe, [0, 0, 1500])
price['medium'] = fuzz.trimf(price.universe, [0, 1500, 3000])
price['high'] = fuzz.trimf(price.universe, [1500, 3000, 3000])
```

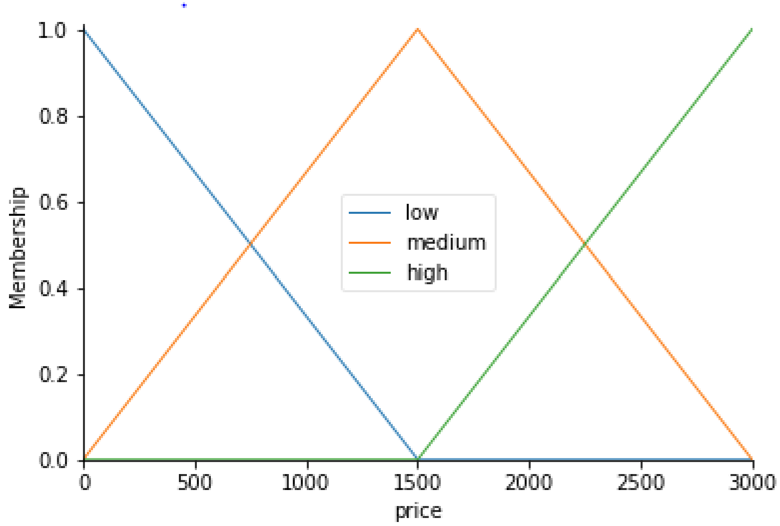
```
hospitality['average'].view()
```




```
GuidingSkill.view()
```

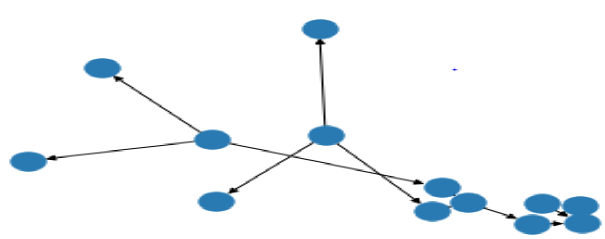



```
prices.view()
```




```
rule1 = ctrl.Rule(hospitality['poor'] | GuidingSkill['poor'], price
['low'])
rule2 = ctrl.Rule(GuidingSkill['average'] | hospitality['average'], price['medium'])
rule3 = ctrl.Rule(hospitality['good'] & GuidingSkill['good'], price['high'])
rule1.view()
```

```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7fa7672fbc10>)
```



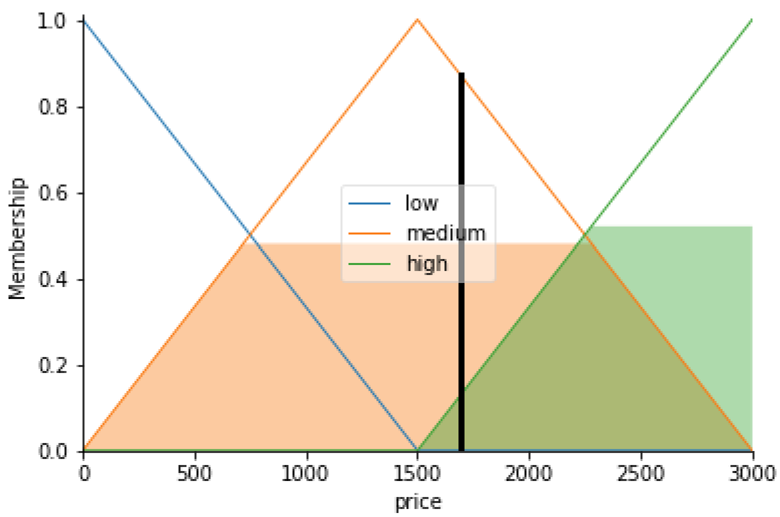
```
judge = ctrl.ControlSystem([rule1, rule2, rule3])
```

```
judgePrice = ctrl.ControlSystemSimulation(judge)
# input passing for hospitality and Guiding skill ratings
hos = input("Enter hospitality rating(range[0 to 100])")
hos=int(hos)
skil = input("Enter GuidingSkills rating(range[0 to 100])")
skil=int(skil)
judgePrice.input['hospitality'] = hos
judgePrice.input['GuidingSkill'] = skil
# find the output
judgePrice.compute()
```

```
Enter hospitality rating(range[0 to 100])76
Enter GuidingSkills rating(range[0 to 100])87
```

```
print(judgePrice.output['price'])
price.view(sim=judgePrice)
```

```
1693.8575393154383
```



Fuzzy set vs Crisp set

Fuzzy set	Crisp set
Fuzzy has a wide range of negative infinity to positive infinity.	The Crisp set has a narrow range of only two values.
Partial inclusion of elements is possible in fuzzy set	Partial inclusion of elements is not possible in crisp set. Elements will either be a part of set or not.
It can be applied in fuzzy controllers.	It is used in digital design.
It has ambiguous or unclear properties.	It has pre defined and well described characteristics.

Assignment 5 Date:17/03/2022

Hebb Rule & AND Gate

AIM:
Write a program to Hebb rule for AND gate.

Theory:
The Hebb rule was proposed by Donald hebb. It states if two neurons are interconnected then weights associated with these neurons can be increased with altering the synaptic gap.It is applicable logic gates, i.e with bipolar data.
Weights are updated using formula:
Here we considered 0(i.e boolean false) as -1 and true as 1.

Wi=Wi-1+x*y
Bias is updated using formula:
Bi=Bi-1+y

Code:

HebbianRule.py

Declaring and initializing inputs X1 and X2 into a 2d array i.e a matrix.The initial bias value is 0.

```
import numpy as np
arr = np.array([1,1,1,-1,-1,1,-1,-1])
x = arr.reshape(4, 2)
print(x)
y= np.array([1,-1,-1,-1])
w=np.array([0,0])
b=0
```

The input Array x

```
[[ 1  1]
 [ 1 -1]
 [-1  1]
 [-1 -1]]
```

Applying hebbian algorithm, where we keep on updating weight and bias.

```
for i in range(4):
 for j in range(2):
   w[j]=w[j]+x[i][j]*y[i]
 b=b+y[i]
```

Printing the values of weight and bias.

```
print(w)
```

```
[2 2]
```

```
print(b)
```

```
-2
```

Verifying the finally outcome of and gate by printing the result of predict and matching it with original result.

```
def predict(x,w,b):
 op=np.array([0,0,0,0])
 for i in range(4):
   op[i]=w[0]*x[i][0]+w[1]*x[i][1]+b
   if(op[i]>0):
     op[i]=1
   else:
     op[i]=-1;
 return op
print(predict(x,w,b))
```

```
[ 1 -1 -1 -1]
```

Remark:
The final outcome of AND gate operation on input X matches with the original answer y.

Assignment 6 Date:24/03/2022

Delta Rule & AND Gate

AIM:
Write a program to Delta rule for AND gate.

Theory:
Delta rule is also known as Perceptron learning rule, the algorithm is similar to hebb rule but the updation step of weight and bias includes an additional parameter alpha i.e learning rate.
Here we considered 0(i.e boolean false) as -1 and true as 1.

Code:

DeltaRule.py

Declaring and initializing inputs X1 and X2 into a 2d array i.e a matrix.The initial bias value is 0.

```
import numpy as np
arr = np.array([1,1,1,-1,-1,1,-1,-1])
x = arr.reshape(4, 2)
print(x)
y= np.array([1,-1,-1,-1])
w=np.array([0,0])
b=0
alph=1; #learning rate
```

The input Array x

```
[[ 1  1]
 [ 1 -1]
 [-1  1]
 [-1 -1]]
```

Applying Delta rule, where we keep on updating weight and bias using learning rate.

```
for i in range(4):
 for j in range(2):
   w[j]=w[j]+alph*x[i][j]*y[i]
 b=b+alph*y[i]
```

Printing the values of weight and bias.

```
print(w)
```

```
[2 2]
```

```
print(b)
```

```
-2
```

Verifying the finally outcome of and gate by printing the result of predict and matching it with original result.

```
def predict(x,w,b):
 op=np.array([0,0,0,0])
 for i in range(4):
   op[i]=w[0]*x[i][0]+w[1]*x[i][1]+b
   if(op[i]>0):
     op[i]=1
   else:
     op[i]=-1;
 return op
print(predict(x,w,b))
```

```
[ 1 -1 -1 -1]
```

Remark:
The final outcome of AND gate operation on input X matches with the original answer y.

Assignment 7 Date:07/04/2022

Genetic Algorithm

AIM:
Write a program to implement Genetic Algorithm.

Theory:
Genetic Algorithms are optimization algorithms, depending on Darwin's "Survival of the fittest" i.e best of the species which can adapt to the changes will survive for a long time.
A few key terms used in Genetic Algorithm are heredity, population diversity, selection of the fittest, survival of the best etc.
The solution has to be found in the given solution space itself.Various steps involved in algorithms are Selection of the best, mutation, crossover, mutation of chromosomes etc.
Evolutionary Algorithms have wide range of applications such as parametric design, DNA analysis, MultiModel Optimizations.
Code:
GeneticAlgo.py
Initializing the population(first generation).

```
#initialize population
import random
best=-100000
populations =([[random.randint(0,1) for x in range(6)] for i in range(4)])
print(type(populations))
parents=[]
new_populations = []
print(populations)
```

The initial population of size 4

```
<class 'list'>
[[1, 1, 0, 1, 1, 0], [1, 1, 1, 1, 0, 0], [1, 0, 1, 1, 0, 0], [1, 0, 0, 1, 0, 1]]
```

Calculating fitness score value

```
#fitness score calculation ............
def fitness_score() :
  global populations,best
  fit_value = []
  fit_score=[]
  for i in range(4) :
     chromosome_value=0
```

```
    for j in range(5,0,-1) :
      chromosome_value += populations[i][j]*(2**(5-j))
    chromosome_value = -1*chromosome_value if populations[i][0]==1 else chromosome_value
    print(chromosome_value)
    fit_value.append(-(chromosome_value**2) + 5 )
  print(fit_value)
  fit_value, populations = zip(*sorted(zip(fit_value, populations) , reverse = True))
  best= fit_value[0]
fitness_score()
```

Printing the chromosome and fitness values.

```
-22
-28
-12
-5
[-479, -779, -139, -20]
```

Selecting the best 2 parents.

```
def selectparent():
  global parents
  #global populations , parents
  parents=populations[0:2]
  print(type(parents))
  print(parents)
selectparent()
```

```
<class 'tuple'>
([1, 0, 0, 1, 0, 1], [1, 0, 1, 1, 0, 0])
```

Performing Crossover at random location

```
#single-point crossover .........

def crossover() :
  global parents

  cross_point = random.randint(0,5)
  parents=parents + tuple([(parents[0][0:cross_point +1] +parents[1][cross_point+1:6])])
  parents=parents + tuple([(parents[1][0:cross_point +1] +parents[0][cross_point+1:6])])

  print(parents)



crossover()
```

Parents(population) after crossover

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

Performing Mutation

```
def mutation() :
  global populations, parents
  mute = random.randint(0,49)
  if mute == 20 :
     x=random.randint(0,3)
     y = random.randint(0,5)
     parents[x][y] = 1-parents[x][y]
  populations = parents
  print(populations)
mutation()
```

Parents(population) after crossover

```
<class 'list'>
[[1, 1, 0, 1, 1, 0], [1, 1, 1, 1, 0, 0], [1, 0, 1, 1, 0, 0], [1, 0, 0, 1, 0, 1]]
```

Performing the iteration for over 1000 times

```
for i in range(1000) :
  fitness_score()
  selectparent()
  crossover()
  mutation()
print("best score :")
print(best)
print("sequence........")
print(populations[0])
```

Final best population after all iterations

```
[5, 5, 5, 5]
<class 'tuple'>
([1, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0])
([1, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0])
([1, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0])
best score :
5
sequence........
[1, 0, 0, 0, 0, 0]
```

Remark:

Genetic Algorithms are the best replacement for solving complex optimization problems for which solutions does not exist in conventional analytics or mathematical modeling i.e hard computing. .