Final Year B. Tech (EE)

Trimester: I **Subject: Artificial Intelligence and Machine Learning**

Name: Shreerang Mhatre **Class: TY**

Roll No: 52 **Batch: A3**

Experiment No: 01

Name of the Experiment: Calculate the output of a simple neuron

Marks	Teacher's Signature with date

Performed on: 14/09/2023

Submitted on: 29/09/2023

Aim: To create a simple neural network and calculate its output.

Prerequisite: Knowledge of logic gates, perceptron, various activation functions.

Objective:

To create a simple single layer neural network and calculate its output using Python Programming.

Components and Equipment required:

Python software

Theory:

Based on nature, neural networks are the usual representation we make of the brain: neurons interconnected to other neurons which forms a network. The operation of a complete neural network is straightforward: one enters variables as inputs, and after some calculations, an output is returned. Artificial neural network is usually put on columns, so that a neuron of the column *n* can only be connected to neurons from columns *n-1* and *n+1*. There are few types of networks that use a different architecture.

A simple artificial neural network is represented as below:

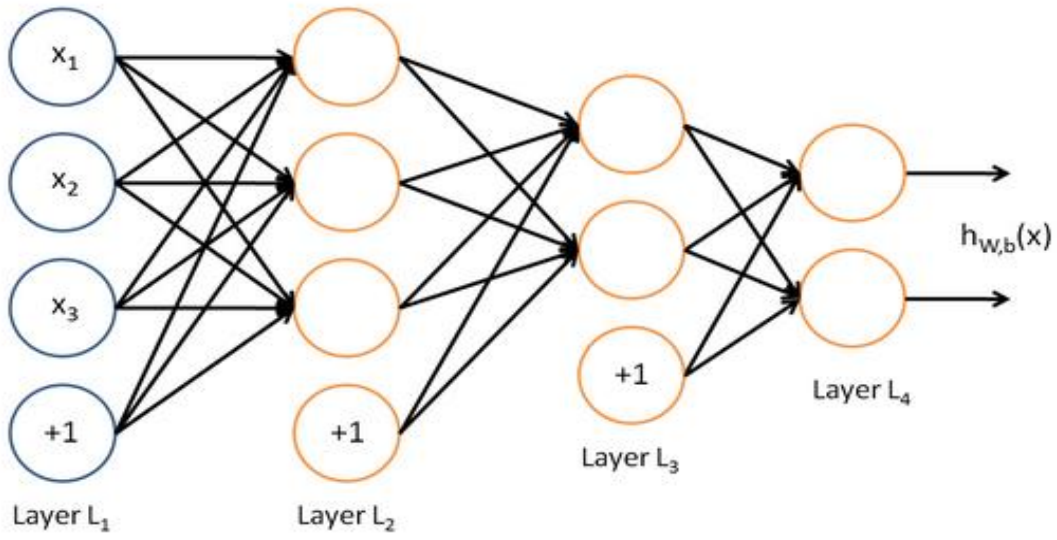


Figure 1 — Representation of a neural network

Neural networks can usually be read from left to right. Here, the first layer is the layer in which inputs are entered. There are 2 internals layers (called hidden layers) that do some math, and one last layer that contains all the possible outputs. “+1” s at the bottom of every column, it is something called “bias”**.**

Operation of a neuron:

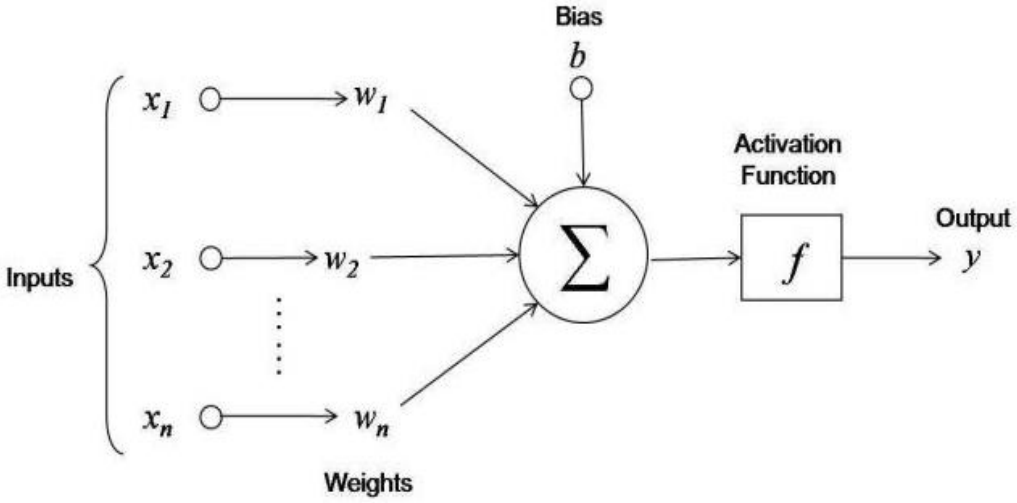


Figure 2 — Operations done by a neuron

First, it adds up the value of every neuron from the previous column it is connected to. On the Figure 2, there are 3 inputs (x1, x2, x3) coming to the neuron, so 3 neurons of the previous column are connected to our neuron. This value is multiplied, before being added, by another variable called “weight” (w1, w2, w3) which determines the connection between the two neurons. Each connection of neurons has its own weight, and those are the only values that will be modified during the learning process.

Moreover, a bias value may be added to the total value calculated. It is not a value coming from a specific neuron and is chosen before the learning phase, but can be useful for the network.

After all those summations, the neuron finally applies a function called “activation function” to the obtained value. Thus a neuron takes all values from connected neurons multiplied by their respective weight, add them, and apply an activation function. Then, the neuron is ready to send its new value to other neurons.

After every neurons of a column did it, the neural network passes to the next column. In the end, the last values obtained should be one usable to determine the desired output.

Network under consideration:

It consists of 2 neurons in the inputs column and 1 neuron in the output column. This configuration allows to create a simple classifier to distinguish 2 groups.

- if A is true and B is true, then A or B is true.
- if A is true and B is false, then A or B is true.
- if A is false and B is true, then A or B is true.
- if A is false and B is false, then A or B is false.

To create a simple Neural Network using Python:

Step-1

```
import numpy, random, os
lr = 1 #learning rate
bias = 1 #value of bias
weights = [random.random(),random.random(),random.random()] #weights generated in a list (3 weights in total for 2 neurons and the bias)
```

In the beginning of the program we define libraries and the values of the parameters, and creates a list which contains the values of the weights that will be modified.

Step-2

```
def Perceptron(input1, input2, output) :
outputP = input1*weights[0]+input2*weights[1]+bias*weights[2]
if outputP > 0 : #activation function (here Heaviside)
outputP = 1
else :
outputP = 0
error = output - outputP
weights[0] += error * input1 * lr
weights[1] += error * input2 * lr
weights[2] += error * bias * lr
```

Here we create a function which defines the work of the output neuron. It takes 3 parameters (the 2 values of the neurons and the expected output). "outputP" is the variable corresponding to the output given by the Perceptron. Then we calculate the error, used to modify the weights of every connection to the output neuron right after.

Step-3

```
for i in range(50) :
Perceptron(1,1,1) #True or true
Perceptron(1,0,1) #True or false
Perceptron(0,1,1) #False or true
Perceptron(0,0,0) #False or false
```

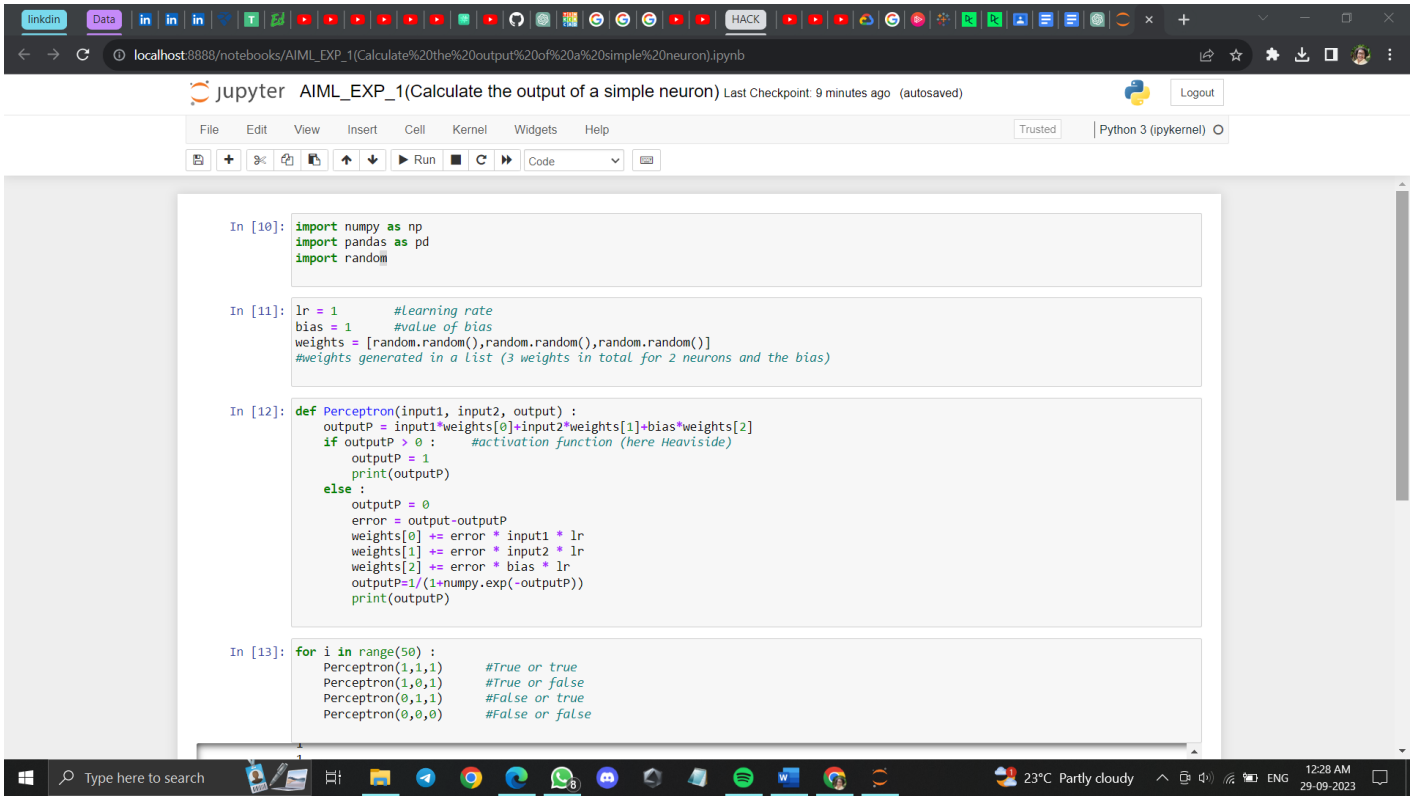
We create a loop that makes the neural network repeat every situation several times. This part is the learning phase. The number of iteration is chosen according to the precision we want.

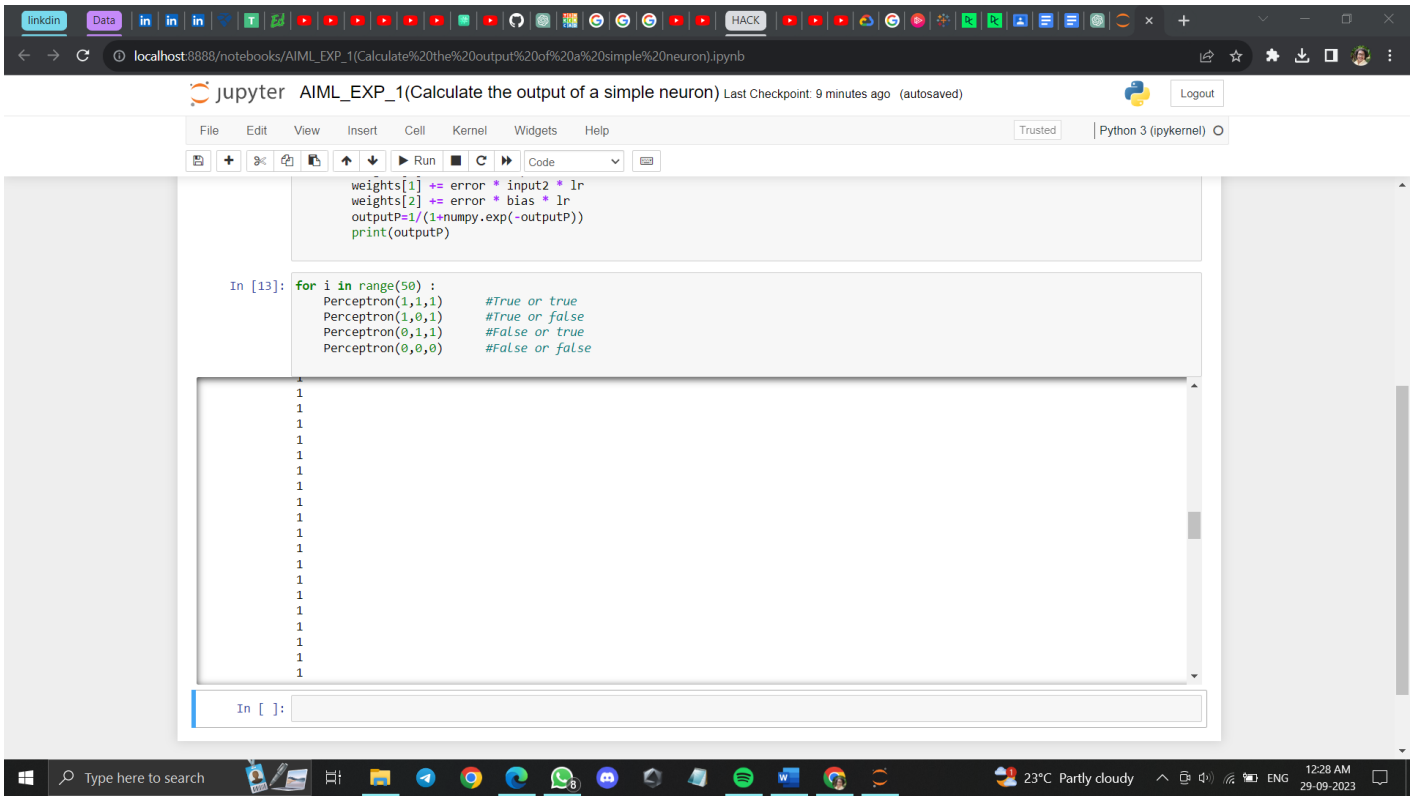
Conclusion:

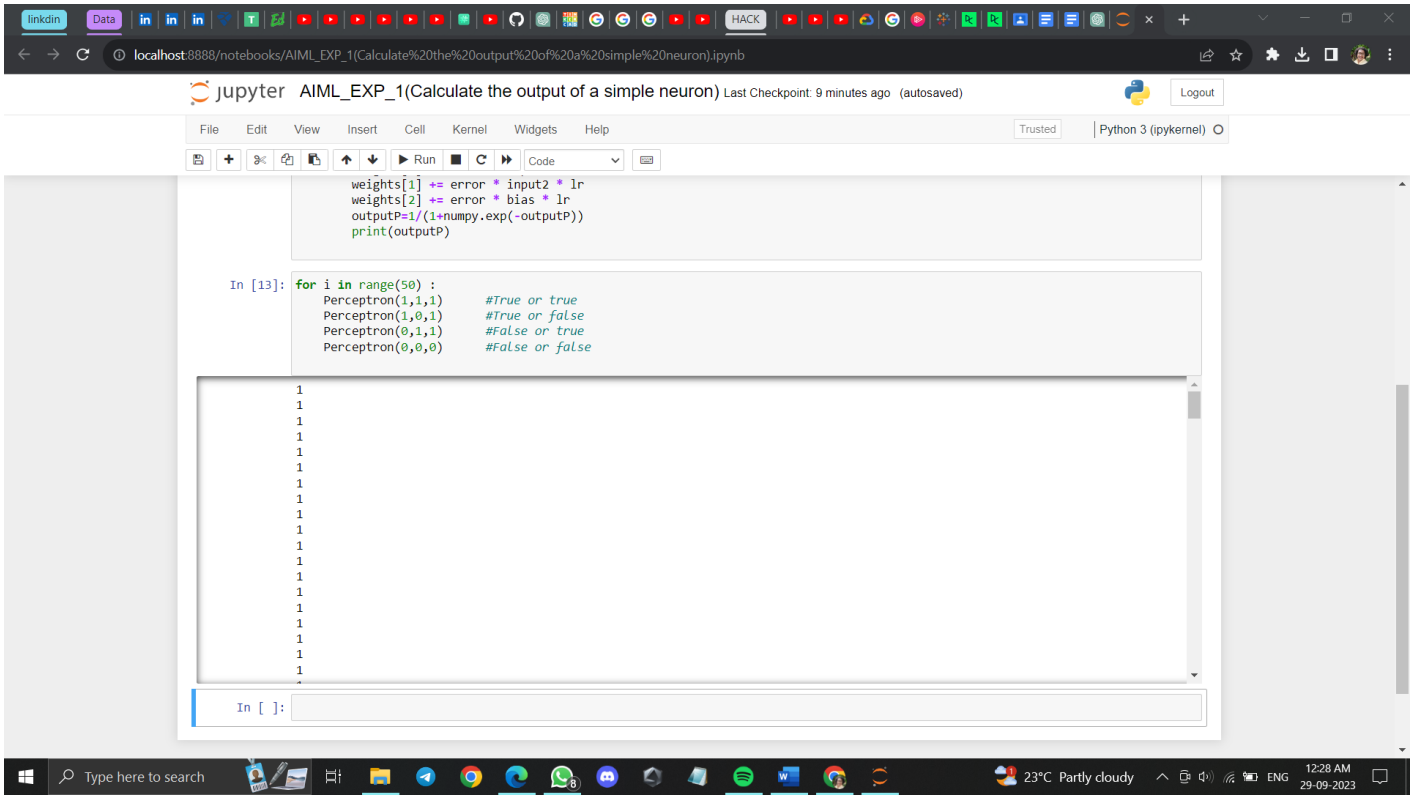
Post Lab Questions:

1. Comment on the similarities between biological neuron and artificial neuron.
2. What do you mean by activation function? State and explain its types.
3. Implement the above code considering sigmoid function.

(`outputP = 1/(1+numpy.exp(-outputP)) #sigmoid function)`


localhost:8888/notebooks/AIML_EXP_1(Calculate%20the%20output%20of%20a%20simple%20neuron).ipynb
jupyter AIML_EXP_1(Calculate the output of a simple neuron) Last Checkpoint: 9 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
In [10]:
import numpy as np
import pandas as pd
import random
In [11]:
lr = 1 #Learning rate
bias = 1 #value of bias
weights = [random.random(),random.random(),random.random()]
#weights generated in a list (3 weights in total for 2 neurons and the bias)
In [12]:
def Perceptron(input1, input2, output) :
outputP = input1*weights[0]+input2*weights[1]+bias*weights[2]
if outputP > 0 : #activation function (here Heaviside)
outputP = 1
print(outputP)
else :
outputP = 0
error = output-outputP
weights[0] += error * input1 * lr
weights[1] += error * input2 * lr
weights[2] += error * bias * lr
outputP=1/(1+numpy.exp(-outputP))
print(outputP)
In [13]:
for i in range(50) :
Perceptron(1,1,1) #True or true
Perceptron(1,0,1) #True or false
Perceptron(0,1,1) #False or true
Perceptron(0,0,0) #False or false
Type here to search
23°C Partly cloudy
ENG
12:28 AM
29-09-2023


localhost:8888/notebooks/AIML_EXP_1(Calculate%20the%20output%20of%20a%20simple%20neuron).ipynb
jupyter AIML_EXP_1(Calculate the output of a simple neuron) Last Checkpoint: 9 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
weights[1] += error * input2 * lr
weights[2] += error * bias * lr
outputP=1/(1+numpy.exp(-outputP))
print(outputP)
In [13]:
for i in range(50) :
Perceptron(1,1,1) #True or true
Perceptron(1,0,1) #True or false
Perceptron(0,1,1) #False or true
Perceptron(0,0,0) #False or false
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
In [ ]:
Type here to search
23°C Partly cloudy
ENG
12:28 AM
29-09-2023


localhost:8888/notebooks/AIML_EXP_1(Calculate%20the%20output%20of%20a%20simple%20neuron).ipynb
jupyter AIML_EXP_1(Calculate the output of a simple neuron) Last Checkpoint: 9 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
weights[1] += error * input2 * lr
weights[2] += error * bias * lr
outputP=1/(1+numpy.exp(-outputP))
print(outputP)
In [13]:
for i in range(50) :
Perceptron(1,1,1) #True or true
Perceptron(1,0,1) #True or false
Perceptron(0,1,1) #False or true
Perceptron(0,0,0) #False or false
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
In [ ]:
Type here to search
23°C Partly cloudy
ENG
12:28 AM
29-09-2023

Expt. 1- 3

Expl - Calculate the output of a simple neuron

PAGE No.
DATE 14/9/23

* Post lab Questions -

Q1) Comment on the similarites between biological neuron & artificial neuron.

-> Biological neurons & artificial neurons share several key similarities -

1. Information Processing - Both process information
2. Input integration - Both integrate incomming signals
3. weighting of inputs - Both assign weights to inputs.
4. Activation / Threshelding - Both use a threshold to decide when to transmit.
5. output Transmission - Transmint information downstream.
6. Learning & Adaptation - Both can adapt & learn from experience
7. Network connectivity - Both are part of larger networks.
8. Non - Linear Transformations - Both can perform non-linear transformations, allowing them to capture complex patterns.

Q2) What do you mean by activation function? State and explain its types

→ An activation function in neural networks is a mathematical function applied to the weighted sum of inputs to a neuron to determine its output. It introduces non-linearity to the model, allowing neural networks to approximate complex relationships in data. Here are some common types of activation functions-

1. Sigmoid Function
2. Hyperbolic Tangent Function
3. Rectified Linear Unit (ReLU)
4. Leaky ReLU
5. Parametric ReLU (PReLU)
6. Exponential Linear Unit (ELU).
7. Swish
8. Gated Recurrent Unit (GRU).

→

PAGE No.
DATE / /

Q3) Implement the above code considering sigmoid function.

→ code -

```
import numpy as np

# Define the output variable
output = 0.5

# Apply the sigmoid function to the output
outputP = 1/(1+np.exp(-outputP)) #sigmoid function

# Print the result
print("Output after applying sigmoid:", outputP)
```

Final Year B. Tech (EE)

Trimester: X **Subject: AIML**

Name: Shreerang Mhatre **Class: Ty**

Roll No: 52 **Batch: A3**

Experiment No: 02

Name of the Experiment: Create and view custom neural networks

Marks	Teacher's Signature with date

Performed on: 24/08/2023

Submitted on: 31/08/2023

Aim: To create and view custom neural networks.

Prerequisite: Knowledge of NN tool in MATLAB, MLP, Activation function.

Objective:

To create and study the Neural Network by varying parameters.

1. Define Input and Output Variable
2. Define and custom Neural Network
3. Define Transfer Function
4. Configure the network
5. Train the network to find output

Components and Equipment required:

MATLAB with NNTool Box

Theory:

Neural networks are a set of algorithms, modelled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labelling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated. Neural networks help us cluster and classify.

In this sense, neural networks refer to systems of neurons, either organic or artificial in nature. Neural networks can adapt to changing input; so the network generates the best possible result without needing to redesign the output criteria. The concept of neural networks, which has its roots in artificial intelligence, is swiftly gaining popularity in the development of trading systems.

A neural network contains layers of interconnected nodes. Each node is a perceptron and is similar to a multiple linear regression. The perceptron feeds the signal produced by a multiple linear regression into an activation function that may be nonlinear.

In a multi-layered perceptron (MLP), perceptrons are arranged in interconnected layers. The input layer collects input patterns. The output layer has classifications or output signals to which input patterns may map.

Hidden layers fine-tune the input weightings until the neural network's margin of error is minimal. It is hypothesized that hidden layers extrapolate salient features in the input data that have predictive power regarding the outputs.

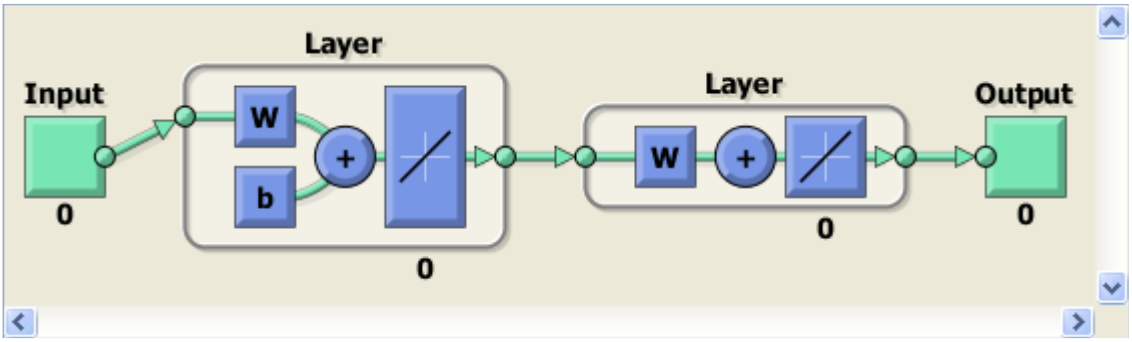
Procedure

Step-1

Define and custom network

Define input and output/ Target variable

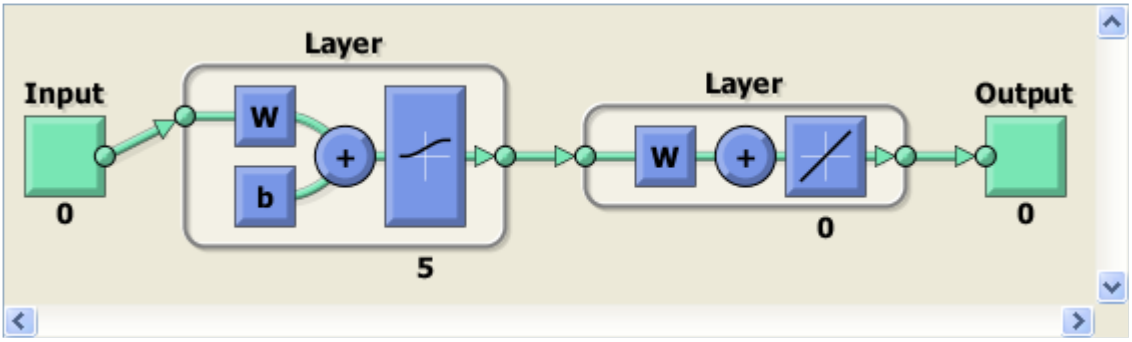
Assign input and target data



Step-2

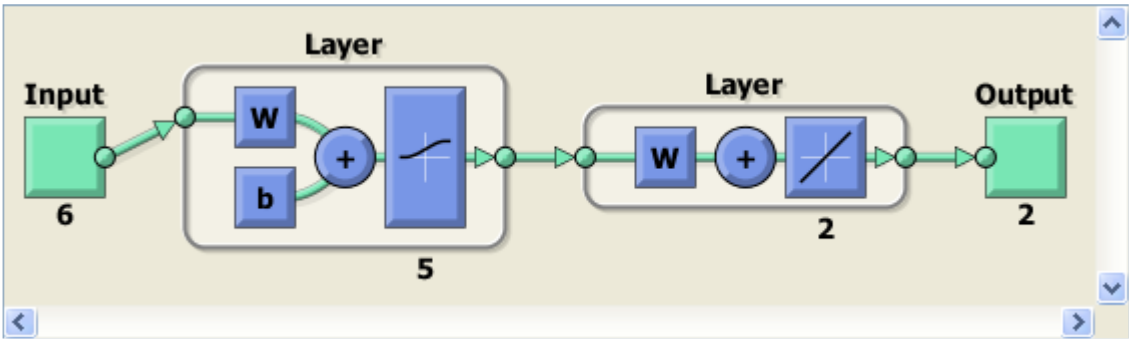
Define topology and transfer function

No. Of hidden layers and their neurons



Step-3

Configure and view network



Step-4

Train net and calculate neuron output

Observations:

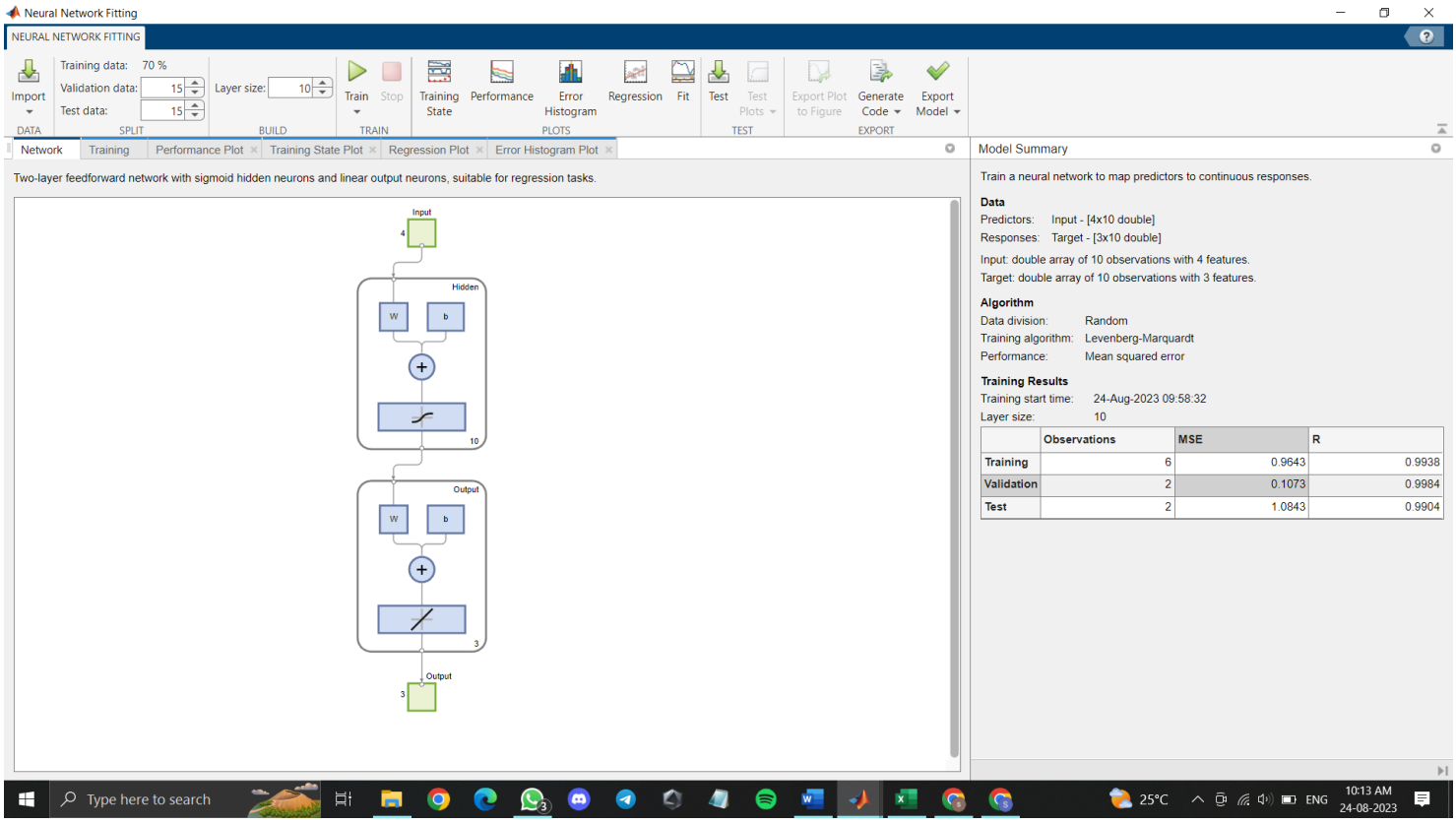
Change the training parameters and view the performance.

Performance Plot

Training Graph

Regression Curves

MATLAB Output:



Neural Network Fitting

NEURAL NETWORK FITTING

Import | Training data: 70 % | Validation data: 15 | Test data: 15 | Layer size: 10 | Train | Stop | Training State | Performance | Error Histogram | Regression | Fit | Test | Test Plots | Export Plot to Figure | Generate Code | Export Model

DATA | SPLIT | BUILD | TRAIN | PLOTS | TEST | EXPORT

Network | Training | Performance Plot | Training State Plot | Regression Plot | Error Histogram Plot

Training Results

Training finished: Reached minimum gradient

Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	7	1000
Elapsed Time	-	00:00:00	-
Performance	13.6	1.61e-26	0
Gradient	27.9	1.11e-13	1e-07
Mu	0.001	1e-10	1e+10
Validation Checks	0	6	6

Model Summary

Train a neural network to map predictors to continuous responses.

Data

Predictors: Input - [4x10 double]
Responses: Target - [3x10 double]

Input: double array of 10 observations with 4 features.
Target: double array of 10 observations with 3 features.

Algorithm

Data division: Random
Training algorithm: Levenberg-Marquardt
Performance: Mean squared error

Training Results

Training start time: 24-Aug-2023 09:58:32
Layer size: 10

	Observations	MSE	R
Training	6	0.9643	0.9938
Validation	2	0.1073	0.9984
Test	2	1.0843	0.9904

Type here to search | 25°C | ENG | 10:13 AM 24-08-2023

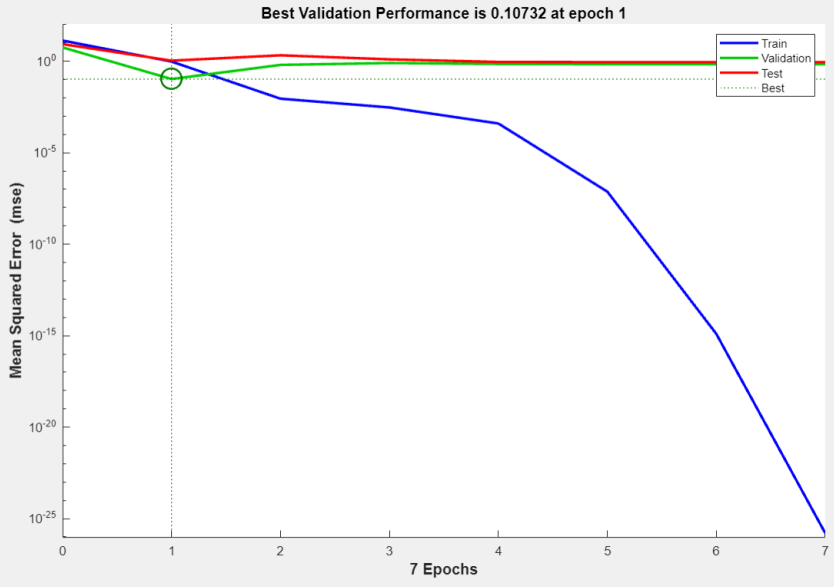
Neural Network Fitting

NEURAL NETWORK FITTING

Import | Training data: 70 % | Validation data: 15 | Test data: 15 | Layer size: 10 | Train | Stop | Training State | Performance | Error Histogram | Regression | Fit | Test | Test Plots | Export Plot to Figure | Generate Code | Export Model

DATA | SPLIT | BUILD | TRAIN | PLOTS | TEST | EXPORT

Network | Training | Performance Plot | Training State Plot | Regression Plot | Error Histogram Plot



Model Summary

Train a neural network to map predictors to continuous responses.

Data

Predictors: Input - [4x10 double]
Responses: Target - [3x10 double]

Input: double array of 10 observations with 4 features.
Target: double array of 10 observations with 3 features.

Algorithm

Data division: Random
Training algorithm: Levenberg-Marquardt
Performance: Mean squared error

Training Results

Training start time: 24-Aug-2023 09:58:32
Layer size: 10

	Observations	MSE	R
Training	6	0.9643	0.9938
Validation	2	0.1073	0.9984
Test	2	1.0843	0.9904

Type here to search | 25°C | ENG | 10:13 AM 24-08-2023

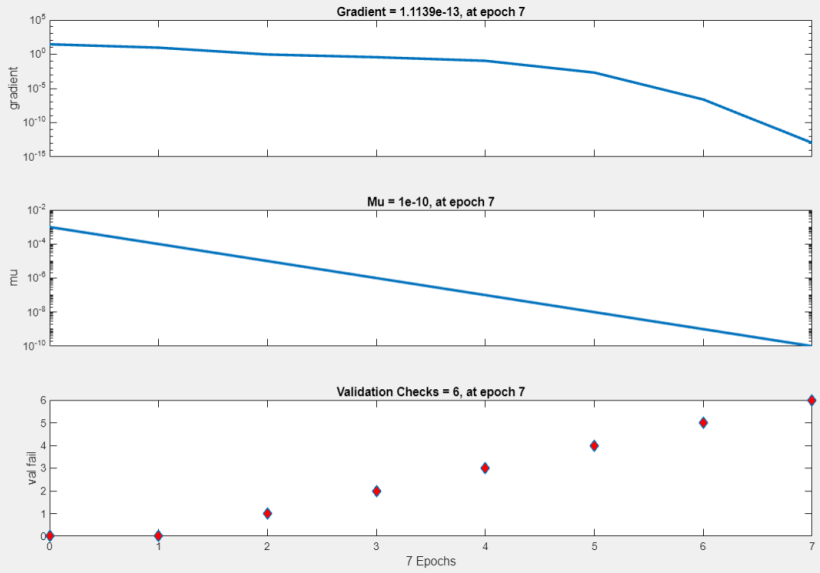
Neural Network Fitting

NEURAL NETWORK FITTING

Import | Training data: 70 % | Validation data: 15 | Test data: 15 | Layer size: 10 | Train | Stop | Training State | Performance | Error Histogram | Regression | Fit | Test | Test Plots | Export Plot to Figure | Generate Code | Export Model

DATA | SPLIT | BUILD | TRAIN | PLOTS | TEST | EXPORT

Network | Training | Performance Plot | Training State Plot | Regression Plot | Error Histogram Plot



Model Summary

Train a neural network to map predictors to continuous responses.

Data
Predictors: Input - [4x10 double]
Responses: Target - [3x10 double]

Input: double array of 10 observations with 4 features.
Target: double array of 10 observations with 3 features.

Algorithm
Data division: Random
Training algorithm: Levenberg-Marquardt
Performance: Mean squared error

Training Results
Training start time: 24-Aug-2023 09:58:32
Layer size: 10

	Observations	MSE	R
Training	6	0.9643	0.9938
Validation	2	0.1073	0.9984
Test	2	1.0843	0.9904

Type here to search | 25°C | ENG | 10:13 AM 24-08-2023

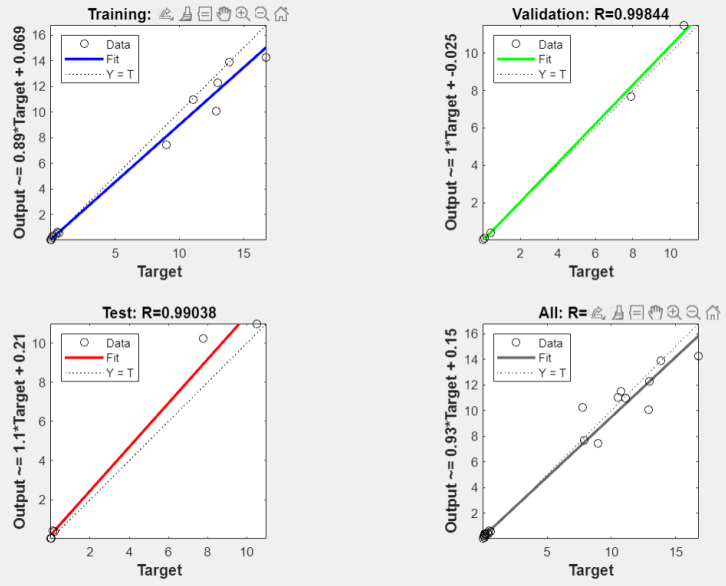
Neural Network Fitting

NEURAL NETWORK FITTING

Import | Training data: 70 % | Validation data: 15 | Test data: 15 | Layer size: 10 | Train | Stop | Training State | Performance | Error Histogram | Regression | Fit | Test | Test Plots | Export Plot to Figure | Generate Code | Export Model

DATA | SPLIT | BUILD | TRAIN | PLOTS | TEST | EXPORT

Network | Training | Performance Plot | Training State Plot | Regression Plot | Error Histogram Plot



Model Summary

Train a neural network to map predictors to continuous responses.

Data
Predictors: Input - [4x10 double]
Responses: Target - [3x10 double]

Input: double array of 10 observations with 4 features.
Target: double array of 10 observations with 3 features.

Algorithm
Data division: Random
Training algorithm: Levenberg-Marquardt
Performance: Mean squared error

Training Results
Training start time: 24-Aug-2023 09:58:32
Layer size: 10

	Observations	MSE	R
Training	6	0.9643	0.9938
Validation	2	0.1073	0.9984
Test	2	1.0843	0.9904

Type here to search | 25°C | ENG | 10:13 AM 24-08-2023

Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

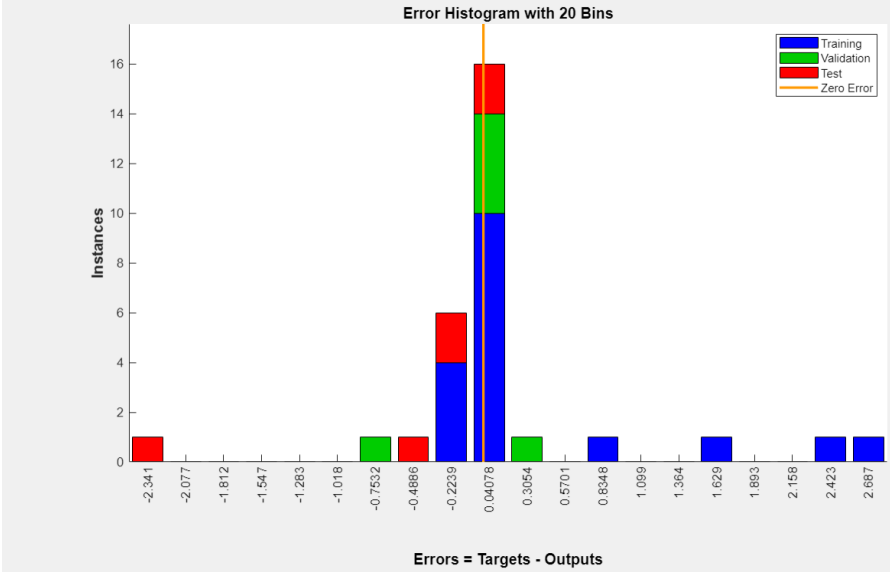
Neural Network Fitting

NEURAL NETWORK FITTING

Import | Training data: 70 % | Validation data: 15 | Test data: 15 | Layer size: 10 | Train | Stop | Training State | Performance | Error Histogram | Regression | Fit | Test | Test Plots | Export Plot to Figure | Generate Code | Export Model

DATA | SPLIT | BUILD | TRAIN | PLOTS | TEST | EXPORT

Network | Training | Performance Plot | Training State Plot | Regression Plot | Error Histogram Plot



Model Summary

Train a neural network to map predictors to continuous responses.

Data

Predictors: Input - [4x10 double]
Responses: Target - [3x10 double]

Input: double array of 10 observations with 4 features.
Target: double array of 10 observations with 3 features.

Algorithm

Data division: Random
Training algorithm: Levenberg-Marquardt
Performance: Mean squared error

Training Results

Training start time: 24-Aug-2023 09:58:32
Layer size: 10

	Observations	MSE	R
Training	6	0.9643	0.9938
Validation	2	0.1073	0.9984
Test	2	1.0843	0.9904

Type here to search | 25°C | ENG | 10:14 AM 24-08-2023

MATLAB R2023a - academic use

HOME | PLOTS | APPS | VARIABLE | VIEW

New Script | New Live Script | New | Open | Find Files | Compare | Import Data | Clean Data | Variable | Save Workspace | Clear Workspace | Favorites | Analyze Code | Run and Time | Clear Commands | Simulink | Layout | Preferences | Set Path | Parallel | Add-Ons | Help | Community | Request Support | Learn MATLAB

FILE | VARIABLE | CODE | SIMULINK | ENVIRONMENT | RESOURCES

Search Documentation | Sign In

C: ▸ Users ▸ SHREERANG ▸ Documents ▸ MATLAB

Current Folder

Name

Details

Select a file to view details

Variables - Input

Input | Target

4x10 double

	1	2	3	4	5	6	7	8	9	10	11	12
1	6	10	6	10	6	10	6	10	6	10		
2	210	210	1010	1010	210	210	1010	1010	210	210		
3	2	2	2	2	6	6	6	6	2	2		
4	0.2000	0.2000	0.2000	0.2000	0.2000	0.2000	0.2000	0.2000	0.6000	0.6000		

Command Window

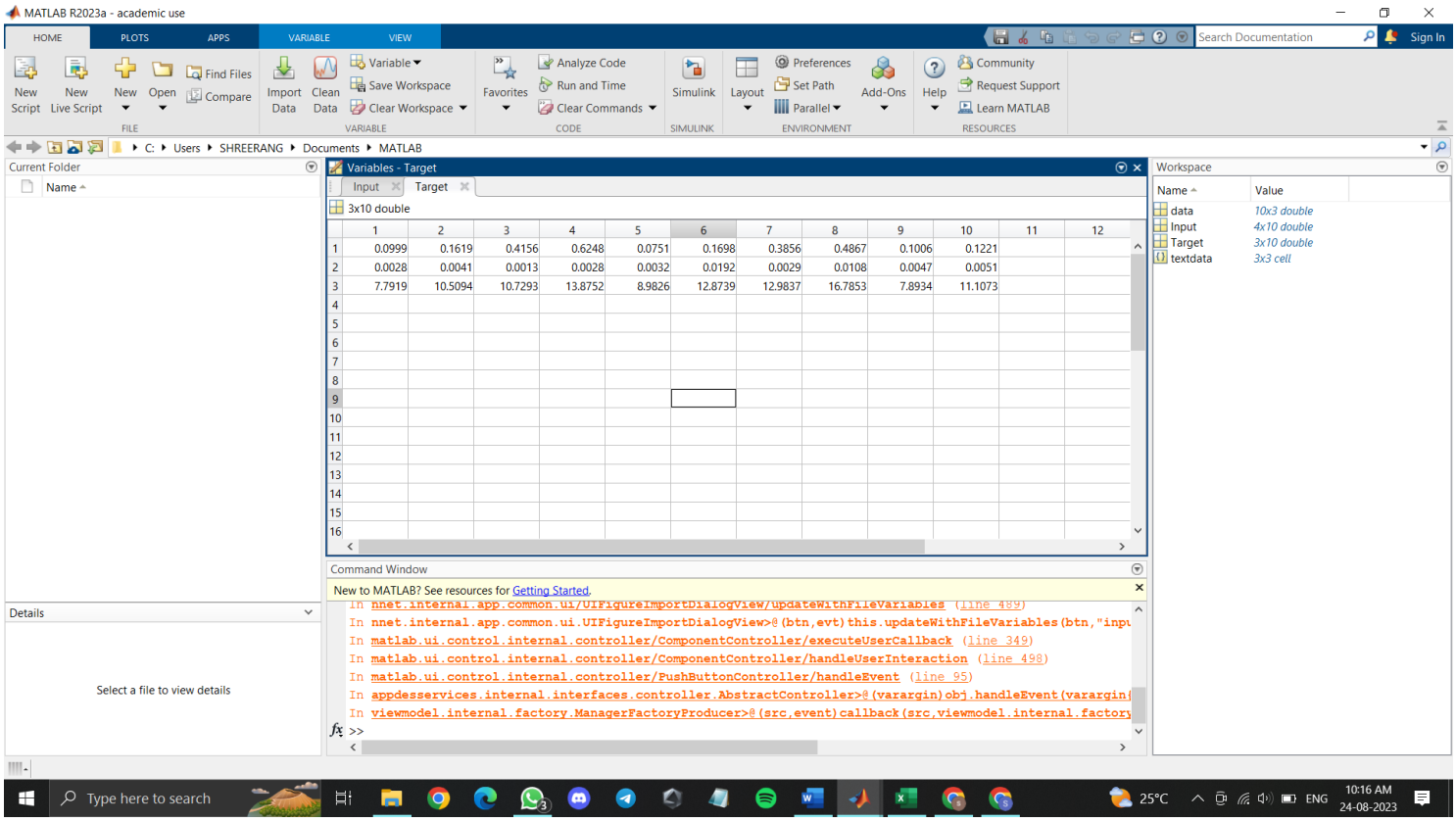
New to MATLAB? See resources for Getting Started.

```
In nnet.internal.app.common.ui.UIFigureImportDialogView/updateWithFileVariables (line 489)
In nnet.internal.app.common.ui.UIFigureImportDialogView>@(btn,evt)this.updateWithFileVariables(btn,"inpu
In matlab.ui.control.internal.controller/ComponentController/executeUserCallback (line 349)
In matlab.ui.control.internal.controller/ComponentController/handleUserInteraction (line 498)
In matlab.ui.control.internal.controller/PushButtonController/handleEvent (line 95)
In appdesservices.internal.interfaces.controller.AbstractController>@(varargin)obj.handleEvent(varargin{
In viewmodel.internal.factory.ManagerFactoryProducer>@(src,event)callback(src,viewmodel.internal.factory
fx >>
```

Workspace

Name	Value
data	10x3 double
Input	4x10 double
Target	3x10 double
textdata	3x3 cell

Type here to search | 25°C | ENG | 10:16 AM 24-08-2023



Conclusions:

Post Lab Questions:

PAGE No.
DATE / /

* Post Lab Questions

1) What is ANN?

An Artificial Neural Network (ANN) is a computational network based on biological neural networks that construct the structure of the human brain. Similar to how a brain has neurons interconnected to each other, artificial neuralnetworks also have neurons that are linked to each other in various layers of the network.

2) Define supervised learning?

Supervised learning is a subcategory of machine learning and artificial intelligence. It is defined by its use of labeled datasets to train algorithms that classify data or predict outcomes accurately. In supervised learning, a training set is used to teach models to yield the desired output. This training dataset includes inputs and correct outputs, which allow the model to learn over time. The algorithm measures its accuracy through the loss function, adjusting until the error has been sufficiently minimized.

PAGE No.
DATE / /

3) Define unsupervised learning

Unsupervised learning is a paradigm in machine learning where, in contrast to supervised learning & semi-supervised learning, algorithms learn patterns exclusively from unlabeled data. These algorithms discover hidden patterns or data groupings without the need for human intervention. Its ability to discover similarities and differences in information make it the ideal solution for exploratory data analysis, cross-selling strategies, customer segmentation, and image recognition.

4) Define semi supervised learning

Semi-supervised learning is a type of machine learning that falls in between supervised & unsupervised learning. It is a method that uses a small amount of labeled data and a large amount of unlabeled data to train a model. This approach to machine learning is a combination of supervised machine learning, which uses labeled training data, and unsupervised learning, which uses unlabeled training data.

PAGE No.
DATE / /

5) Define reinforcement learning.

Reinforcement learning (RL) is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward. It is one of three basic machine learnings paradigms, alongside supervised learning & unsupervised learning. Reinforcement learning differs from supervised learning in not needing labelled input/output pairs to be presented and in not needing sub-optimal actions to be explicitly corrected.

6) What is learning rate? Why it is needed.

In machine and statics, the learning rate is a tuning parameter in an optimization algorithm that determines the step at each iteration while moving toward a minimum of a loss function. It influences to what extent newly acquired information overrides old information, and metaphorically represents the speed at which a machine learning models 'learns'. The learning rate is used to scale the magnitude of parameter updates during gradient descent.

Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

Final Year B. Tech (EE)

Trimester: VII **Subject: AIML**

Name: Shreerang Mhatre

Class: Final Year B. Tech (EE)

Roll No: 52

Batch: A3

A3

Experiment No: 03

Name of the Experiment: Vacuum Cleaner World Agent using Python

Performed on: 11/09/2023

Submitted on: 11/09/2023

Marks	Teacher's Signature withdate

Aim: To develope a simple reflex agent program in Python for the vacuum-cleaner world problem.

Prerequisite: Knowledge of Agents.

Objective:

To create a Vacuum Cleaner World Agent using Python Programming.

Components and Equipment required:Python

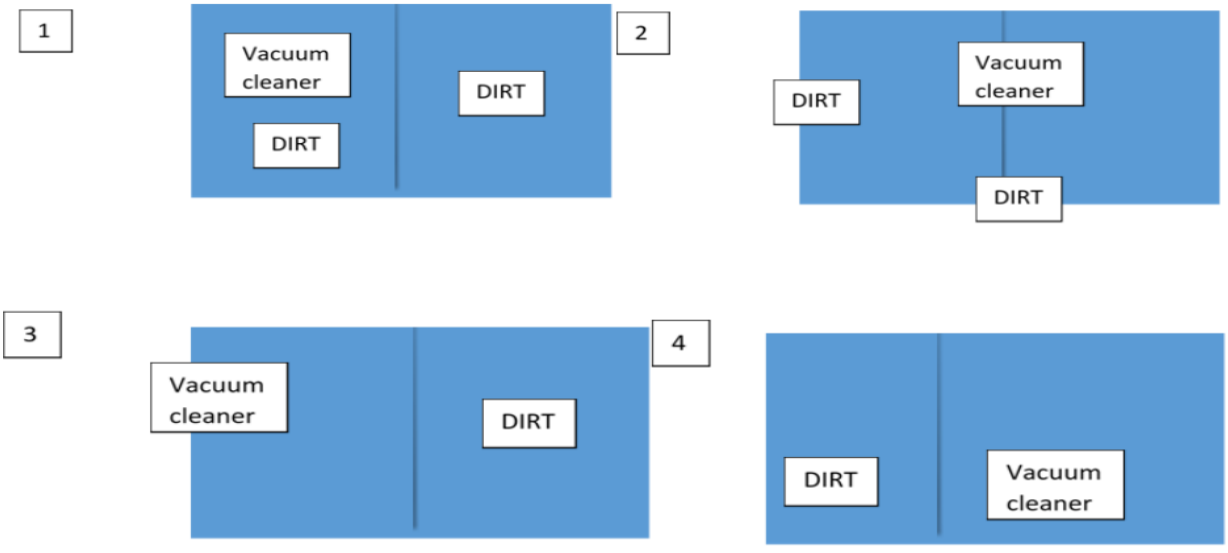
Theory:

Vacuum cleaner problem is a well-known search problem for an agent which works on ArtificialIntelligence. In this problem, our vacuum cleaner is our agent. It is a goal based agent, and the goal of this agent, which is the vacuum cleaner, is to clean up the whole area. So, in the classical vacuum cleaner problem, we have two rooms and one vacuum cleaner. There is dirt in both the roomsand it is to be cleaned. The vacuum cleaner is present in any one of these rooms. So, we have to reacha state in which both the rooms are clean and are dust free.

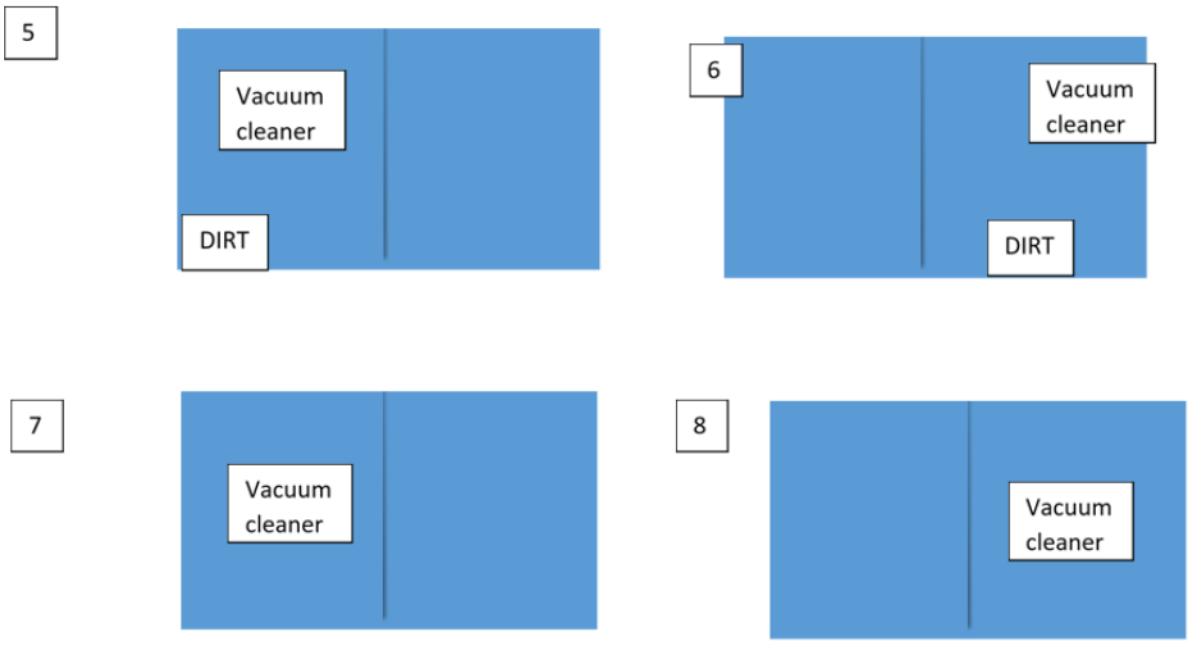
This program defines the States, Goal State, Goal Test, Actions, Transition Model, and Path Cost. Foreach possible initial state, the program returns a sequence of actions that leads to the goal state, along with the path cost. Generates two test cases.

1. **EnterLOCATIONA/BincaptialletterswhereAandBarethetwoadjacentrooms respectively.**
2. **EnterStatusO/1accordinglywhere0meansCLEANand1meansDIRTY.**
3. **VacuumCleanersensesthestatusoftheotherroombeforeperforminganyaction, also known as Environmentsensing.**

So,thereareeightpossiblestatespossibleinourvacuumcleanerproblem**.Thesecanbewellillustratedwiththehelpofthefollowingdiagrams:**



Exp. 3- 2



Here, states 1 and 2 are our initial states and state 7 and state 8 are our final states (goal states).This means that, initially, both the rooms are full of dirt and the **vacuum cleaner** can reside in any room. And to reach the final goal state, both the rooms should be clean and the **vacuum cleaner** again can reside in any of the two rooms.

The **vacuum cleaner** can perform the following functions: move left, move right, move forward,move backward and to suck dust. But as there are only two rooms in our problem, the vacuum cleaner performs only the following functions here: move left, move right and suck.

Here the performance of our agent (vacuum cleaner) depends upon many factors such as timetaken in cleaning, the path followed in cleaning, the number of moves the agent takes intotal,etc. But we consider two main factors for estimating the performance of the agent. Theyare:

1. **Search Cost:** How long the agent takes to come up with thesolution.
2. **Path cost:** How expensive each action in the solutionare.

By considering the above factors, the agent can also be classifies as a utility based agent.

Procedure:

- Create an Environment and Vacuum Cleaner Agent
- Select random location and random status for Vacuum Cleaner Agent
- Verify Initial Location and Environment Status.
- Achieve the goal state.

Programme Code

```
import random
class Environment(object):
   definit(self): self.locationcondition={'A'
      : '1' , 'B' :'1'} #RANDOM
CONDITION
      self.locationcondition['A']=random.randint(0,1)
      self.locationcondition['B']=random.randint(0,1)


class Sreflexagent(Environment):
   definit(self,Environment):
      #print(Environment.locationcondition)#place
      vacum at random location
      vacuumlocation=random.randint(0, 1) #if
      vacuum at A
      if vacuumlocation==0:
         print("vacuum is randomly placed at locationn A")#and if
         location A is dirty
         if Environment.locationcondition['A']==1:
            print("Location A is dirty")
            #suck the dirt and mark it clean
            Environment.locationcondition['A']=0
            print("Location A has been cleaned")
            print("moving to location B")
            vacuumlocation=1
         else:
            print("Location A is clean")#move
         to B
            print("moving to location B")
            vacuumlocation=1
```

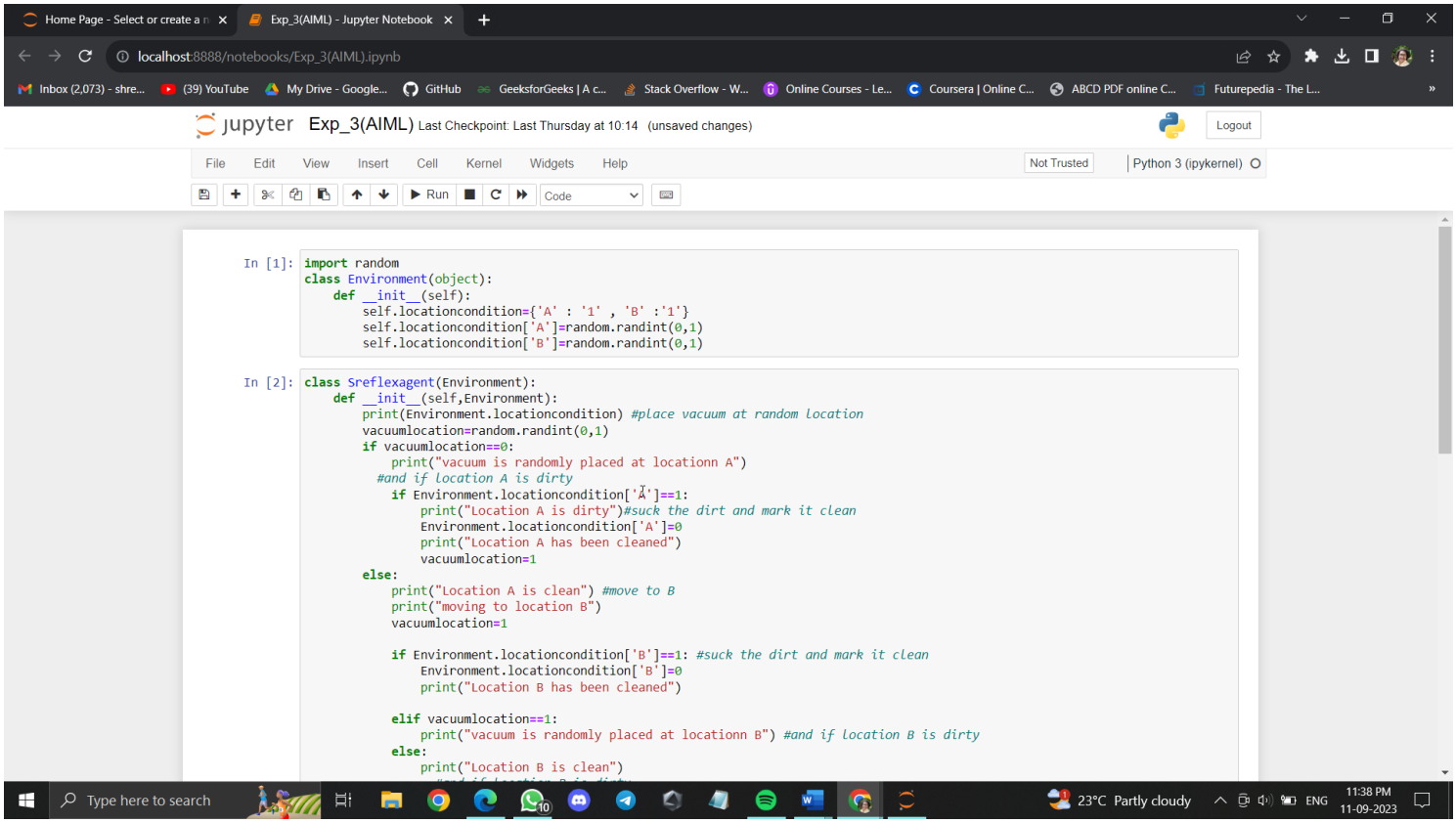
```
if
        Environment.locationcondition['B']==1
        : #suck the dirt and mark itclean
          Environment.locationcondition['B']=0
          print("Location B has been cleaned")
      else:
          print("Location B is clean")elif
    vacuumlocation==1:
      print("vacuum is randomly placed at locationn B")#and if
      location B is dirty
      if Environment.locationcondition['B']==1:
        print("Location B is dirty")
        #suck the dirt and mark it clean
        Environment.locationcondition['B']=0;
        print("Location B has been cleaned")
        print("moving to location A")
        vacuumlocation=0
      else:
        print("Location B is Clean")#Move
        to A
        print("moving to location A")
        vacuumlocation=0
      if vacuumlocation==0:
        if Environment.locationcondition['A']==1:
          print("Location A is dirty")
        #suck the dirt and mark it clean
          Environment.locationcondition['A']=0;
          print("Location A has been cleaned")
        else:
          print("Location A is Clean")



#DONE CLENING

#print(Environment.locationcondition)
theEnvironment=Environment()
thevacuum=Sreflexagent(theEnvironment)
```

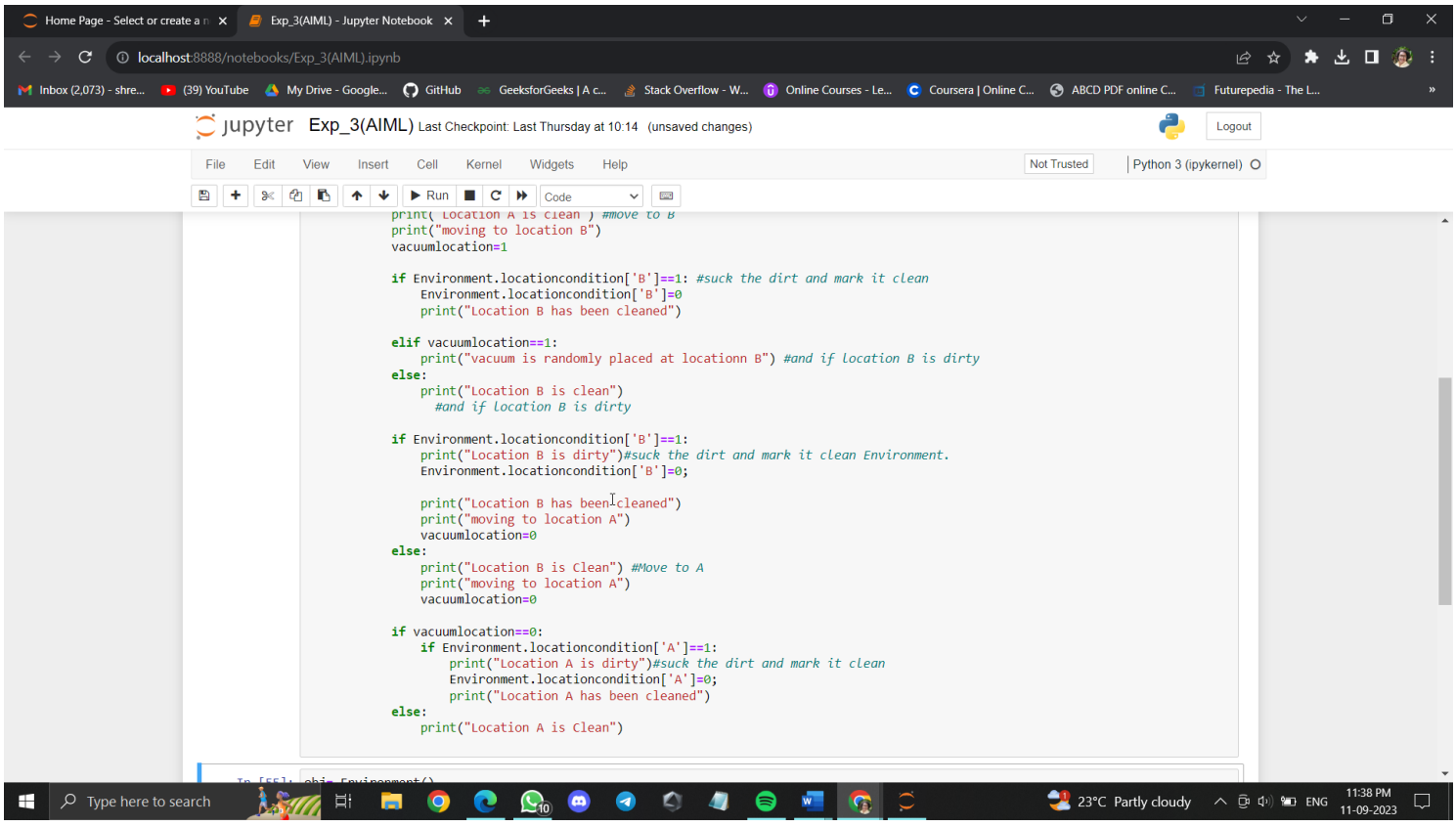
Conclusion:

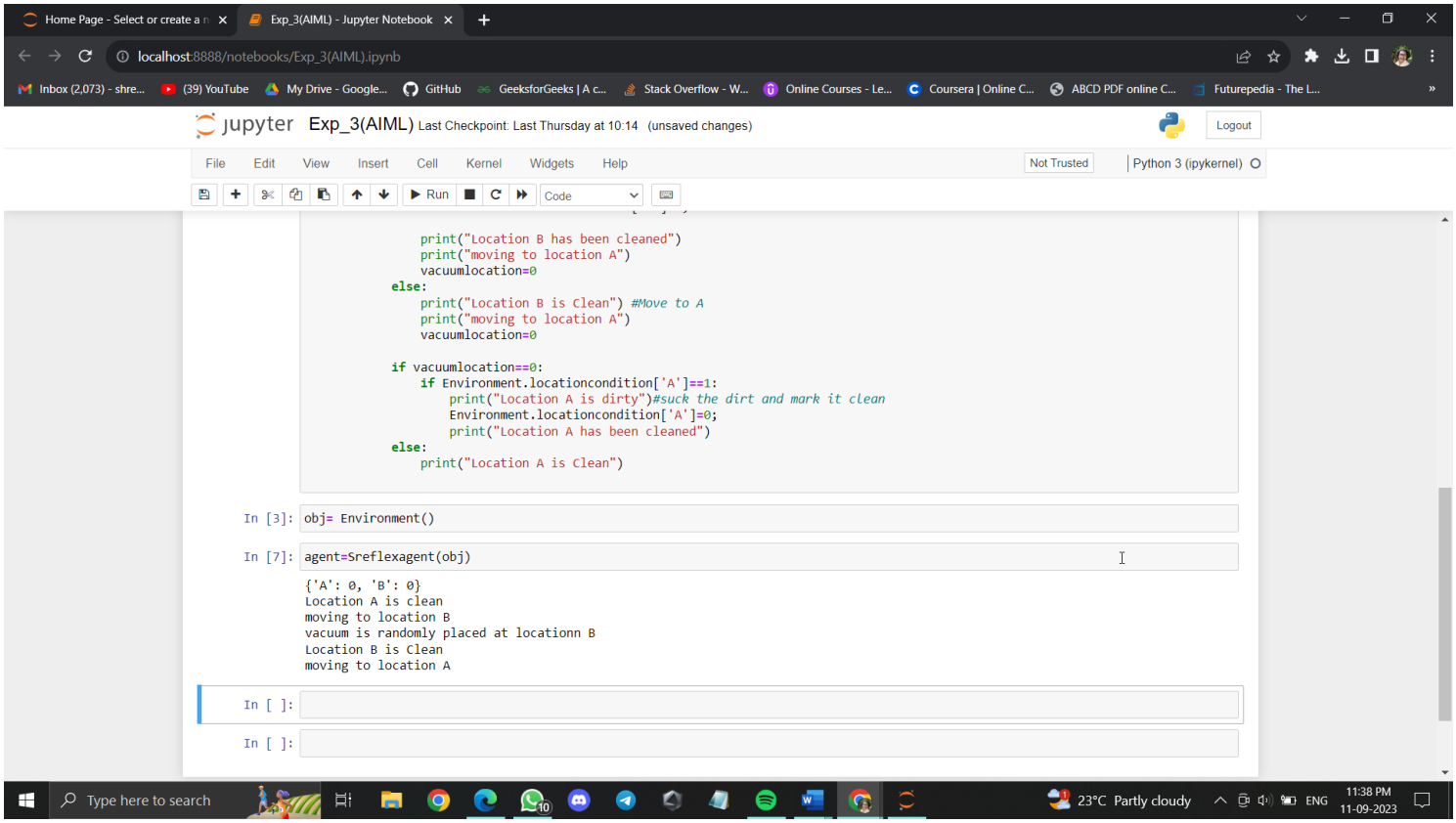
A vacuum cleaner world agent is developed for two locations considering randomstatus using Python Programming.

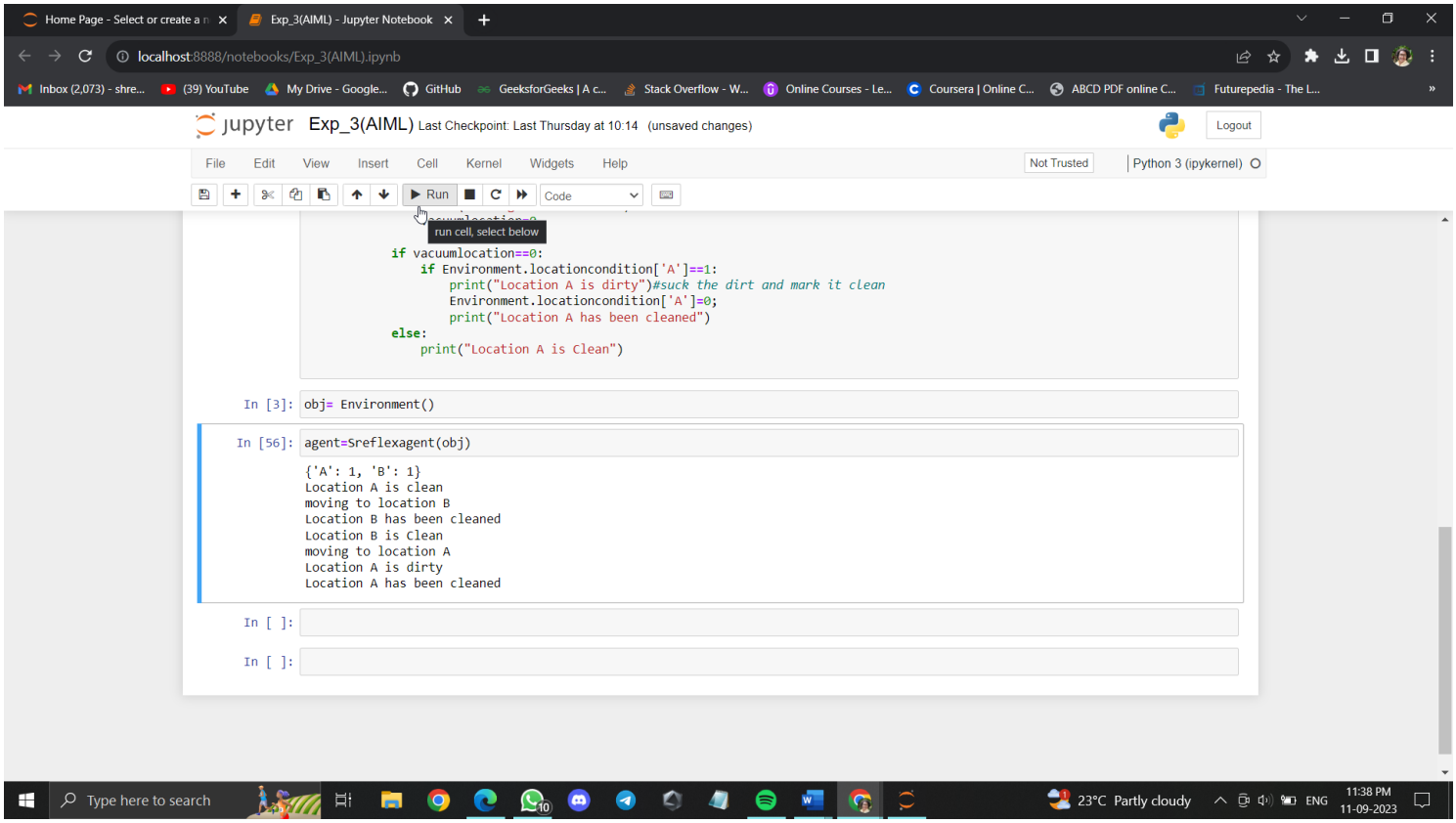
Post Lab Questions:

1. What are the different types of agents? Explain the pseudocode forutility based agent.
2. Explain different terms related to agent.
3. what are the different types of environment?


Home Page - Select or create a n
Exp_3(AIML) - Jupyter Notebook
localhost:8888/notebooks/Exp_3(AIML).ipynb
jupyter Exp_3(AIML) Last Checkpoint: Last Thursday at 10:14 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
In [1]:
import random
class Environment(object):
def __init__(self):
self.locationcondition={'A' : '1' , 'B' :'1'}
self.locationcondition['A']=random.randint(0,1)
self.locationcondition['B']=random.randint(0,1)
In [2]:
class Sreflexagent(Environment):
def __init__(self,Environment):
print(Environment.locationcondition) #place vacuum at random location
vacuumlocation=random.randint(0,1)
if vacuumlocation==0:
print("vacuum is randomly placed at locationn A")
#and if location A is dirty
if Environment.locationcondition['A']==1:
print("Location A is dirty")#suck the dirt and mark it clean
Environment.locationcondition['A']=0
print("Location A has been cleaned")
vacuumlocation=1
else:
print("Location A is clean") #move to B
print("moving to location B")
vacuumlocation=1
if Environment.locationcondition['B']==1: #suck the dirt and mark it clean
Environment.locationcondition['B']=0
print("Location B has been cleaned")
elif vacuumlocation==1:
print("vacuum is randomly placed at locationn B") #and if location B is dirty
else:
print("Location B is clean")
Type here to search
23°C Partly cloudy
ENG
11:38 PM
11-09-2023


Home Page - Select or create a n
Exp_3(AIML) - Jupyter Notebook
localhost:8888/notebooks/Exp_3(AIML).ipynb
jupyter Exp_3(AIML) Last Checkpoint: Last Thursday at 10:14 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
print("moving to location B")
vacuumlocation=1
if Environment.locationcondition['B']==1: #suck the dirt and mark it clean
Environment.locationcondition['B']=0
print("Location B has been cleaned")
elif vacuumlocation==1:
print("vacuum is randomly placed at locationn B") #and if location B is dirty
else:
print("Location B is clean")
#and if location B is dirty
if Environment.locationcondition['B']==1:
print("Location B is dirty")#suck the dirt and mark it clean Environment.
Environment.locationcondition['B']=0;
print("Location B has been cleaned")
print("moving to location A")
vacuumlocation=0
else:
print("Location B is Clean") #Move to A
print("moving to location A")
vacuumlocation=0
if vacuumlocation==0:
if Environment.locationcondition['A']==1:
print("Location A is dirty")#suck the dirt and mark it clean
Environment.locationcondition['A']=0;
print("Location A has been cleaned")
else:
print("Location A is Clean")
Type here to search
23°C Partly cloudy
ENG
11:38 PM
11-09-2023


Home Page - Select or create a n
Exp_3(AIML) - Jupyter Notebook
localhost:8888/notebooks/Exp_3(AIML).ipynb
jupyter Exp_3(AIML) Last Checkpoint: Last Thursday at 10:14 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
print("Location B has been cleaned")
print("moving to location A")
vacuumlocation=0
else:
print("Location B is Clean") #Move to A
print("moving to location A")
vacuumlocation=0
if vacuumlocation==0:
if Environment.locationcondition['A']==1:
print("Location A is dirty")#suck the dirt and mark it clean
Environment.locationcondition['A']=0;
print("Location A has been cleaned")
else:
print("Location A is Clean")
In [3]: obj= Environment()
In [7]: agent=Sreflexagent(obj)
{'A': 0, 'B': 0}
Location A is clean
moving to location B
vacuum is randomly placed at locationn B
Location B is Clean
moving to location A
In [ ]:
In [ ]:
Type here to search
23°C Partly cloudy
ENG
11:38 PM
11-09-2023


Home Page - Select or create a n
Exp_3(AIML) - Jupyter Notebook
localhost:8888/notebooks/Exp_3(AIML).ipynb
jupyter Exp_3(AIML) Last Checkpoint: Last Thursday at 10:14 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
run cell, select below
if vacuumlocation==0:
if Environment.locationcondition['A']==1:
print("Location A is dirty")#suck the dirt and mark it clean
Environment.locationcondition['A']=0;
print("Location A has been cleaned")
else:
print("Location A is Clean")
In [3]: obj= Environment()
In [56]: agent=Sreflexagent(obj)
{'A': 1, 'B': 1}
Location A is clean
moving to location B
Location B has been cleaned
Location B is Clean
moving to location A
Location A is dirty
Location A has been cleaned
In [ ]:
In [ ]:
Type here to search
23°C Partly cloudy
ENG
11:38 PM
11-09-2023

PAGE No.
DATE 11/9/23

Exp 3 Vaccum Cleaner Agent using Python

* Past lab questions.

Q1) what are the different types of agents? Explain the pseudocode for utility based agent.

→ Types of agents are:
1. Simple Reflex agent
2. Model-based reflex agent
3. Goal based agents
4. Utility based agents

*Utility based agent

Define a set of possible actions (A)
Define a set of possible states or world states (s)
Define a utility function U(a,s) for each action a & state s
Repeat until a terminal state is reached.

1. Observe the current state s-current.
2. For each action a in A:
   a) Calculate the expected utility EU(a) for taking action a in the current state:
   EU(a) = Σ (U(a, s-next) * p(s-next/a, s-current)), for all possible s-next in S.

PAGE No.
DATE / /

③ choose the action a-max with the highest expected utility.
a-max argman (EU(a) forall a in A)
④ Execute the chosen action a-max.
⑤ observe the resulting state s-next
⑥ Repeat the process with the new state s-next.
End loop when a terminal state is reached.

(Q2) Explain different terms related to agents

→① Agent -
An agent is a software or hardware entity that perceives its environment, processes information & takes actions to achieve specific goals or objectives

② Environment -
The environment is the external context or surroundings in which an agent operates & interacts. It encompasses everything outside the agent itself.

③ Perception -
Perception is the process through which an agent gathers information about its environment using sensors or other means. Basically it is the input to the agent.

PAGE No.
DATE / /

④ State -
The state represents the current condition or configuration of the environment at a given point in time

⑤ Agent function.
An agent function maps the agent's perception of the current state to an action or a sequence of actions.
- It defines how an agent behaves in response to different states of the environment.

⑥ Agent Program -
An agent program is the specific implementation or code that embodies the agent function. It determines how the agent operates & makes decisions.

⑦ Rational Agent -
A rational agent is an agent that consistently selects actions that maximize its expected utility, given its knowledge & beliefs about the environment.

Q3) what are the different types of environment?

→ ① Physical environment
② Biological environment
③ Climatic environment
④ Marine environment
⑤ Man-made environment
⑥ Social Environment
⑦ virtual environment
⑧ Extreme environment
⑨ Aquatic environment
⑩ Artificial environment.

Final Year B. Tech (EE)

Trimester: I **Subject: Artificial Intelligence and Machine Learning**

Name: Shreerang Mhatre **Class: TY**

Roll No: 52 **Batch: A3**

Experiment No: 04

Name of the Experiment: Classify species of Iris flower using MLP

Performed on:14/09/2023

Submitted on:14/09/2023

Marks	Teacher's Signature with date

Aim: To create a multilayer neural network and classify species of iris flower using Python.

Prerequisite: Knowledge of MLP, iris flower data and its classes.

Objective:

To create a multi-layer neural network and classify iris flower data using Python Programming.

Components and Equipment required:

SkLearn Python module, Python software, NumPy and Panda Libraries

Theory

A hobby botanist is interested in distinguishing the species of some iris flowers that she has found. She has collected some measurements associated with each iris, which are:

- the length and width of the petals
- the length and width of the sepals, all measured in centimetres.

She also has the measurements of some irises that have been previously identified by an expert botanist as belonging to the species setosa, versicolor, or virginica. For these measurements, she can be certain of which species each iris belongs to. We will consider that these are the only species our botanist will encounter.
The goal is to create a machine learning model that can learn from the measurements of these irises whose species are already known, so that we can predict the species for the new irises that she has found.

Building our model

As we have measurements for which we know the correct species of iris, this is a supervised learning problem. We want to predict one of several options (the species of iris), making it an example of a classification problem. The possible outputs (different species of irises) are called classes. Every iris in the dataset belongs to one of three classes considered in the model, so this problem is a three-class classification problem. The desired output for a single data point (an iris) is the species of the flower considering it's features. For a particular data point, the class / species it belongs to is called its label.

Procedure:

- SkLearn is a pack of Python modules built for data science applications

- load_iris: The classic dataset for the iris classification problem. (NumPy array)
- train_test_split: method for splitting our dataset.
- KNeighborsClassifier: method for classifying using the K-Nearest Neighbor approach.
- NumPy is a Python library that makes it easier to work with N-dimensional arrays and has a large collection of mathematical functions at its disposal. It's' base data type is the "numpy.ndarray".

Output

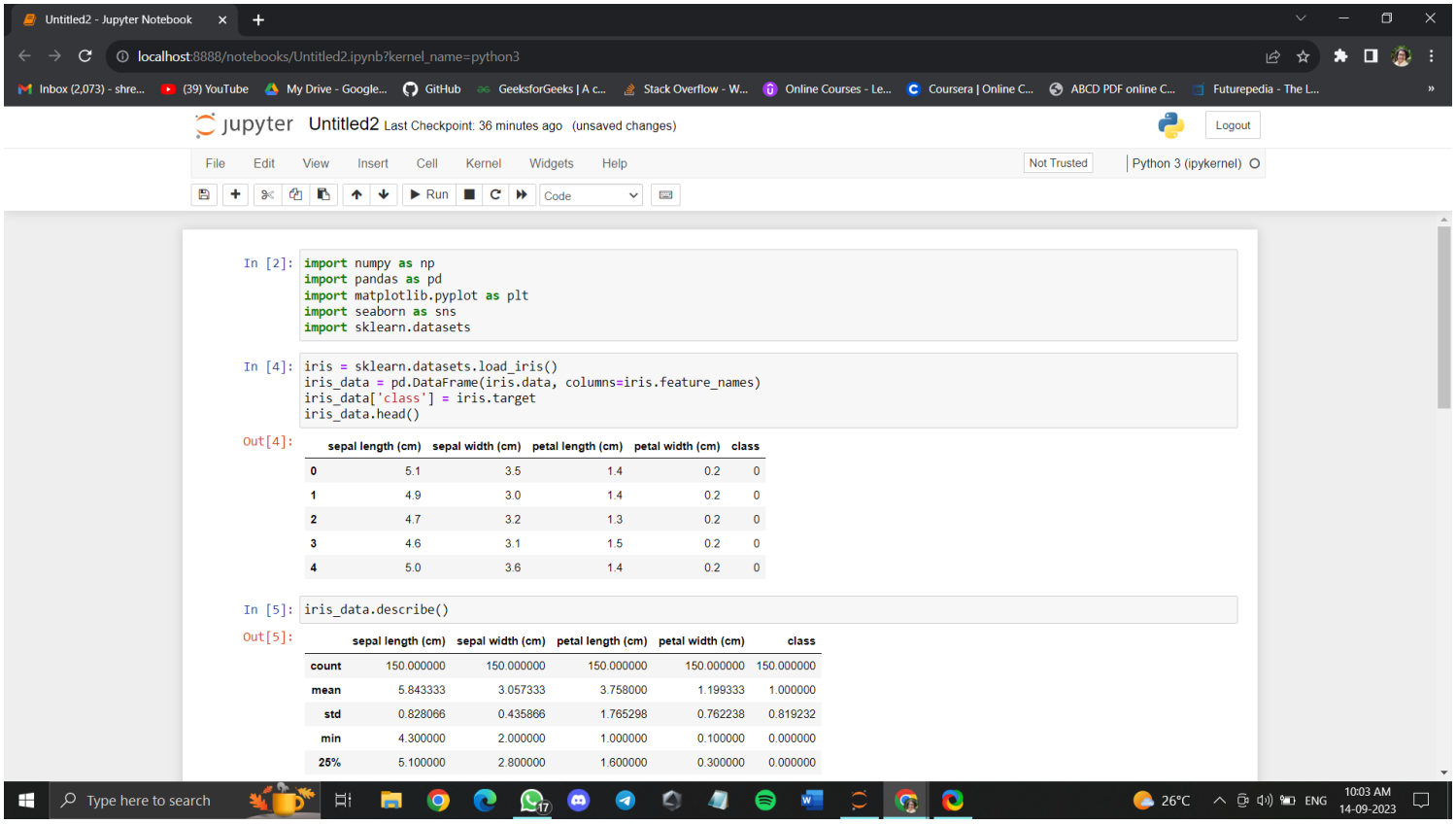
The target array contains the species of each of the flowers that were measured. This array is composed of numbers from 0 to 2.
The meaning of those numbers are directly related to our target names (classes):

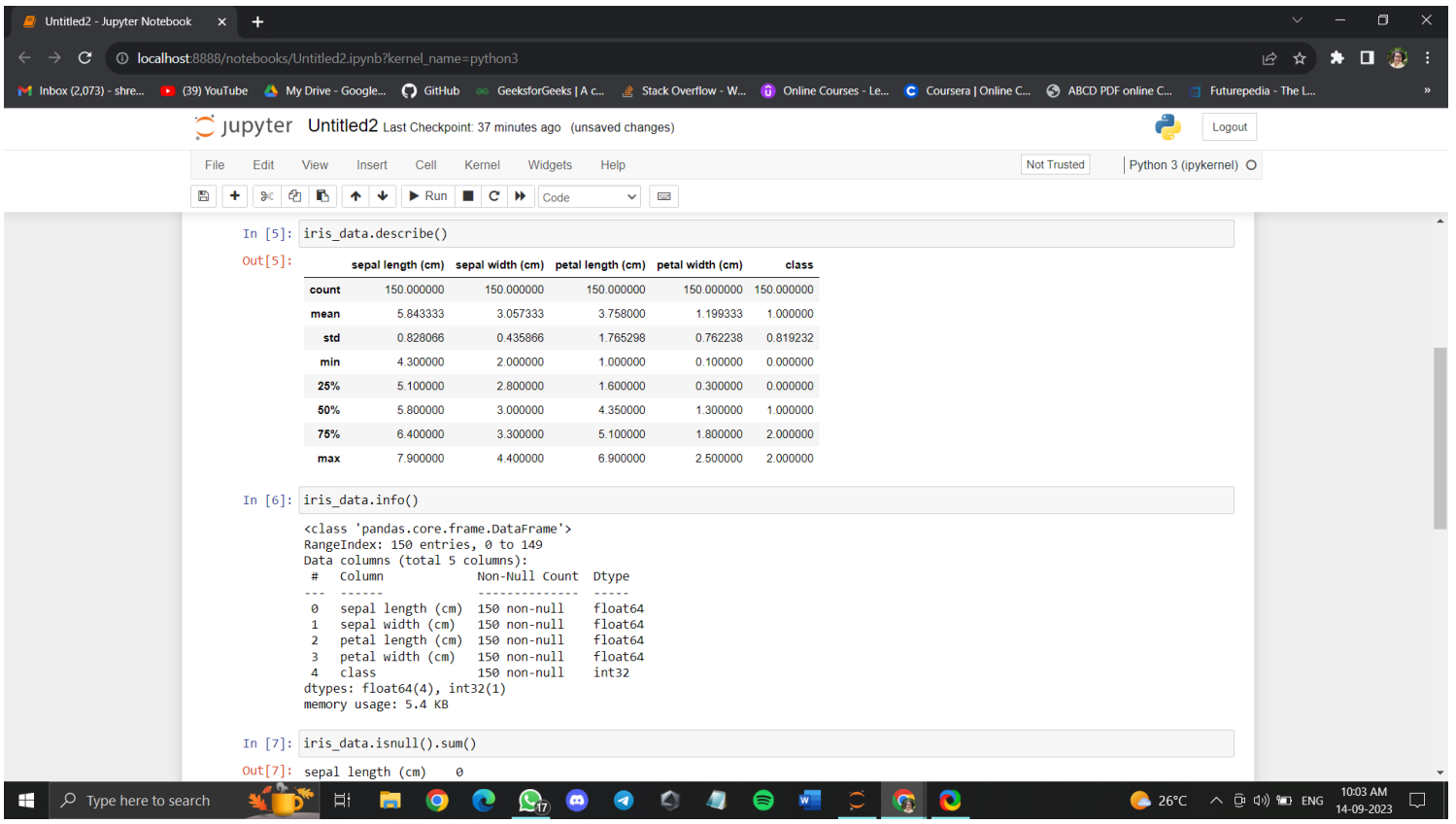
- setosa (0)
- versicolor (1)
- virginica(2)

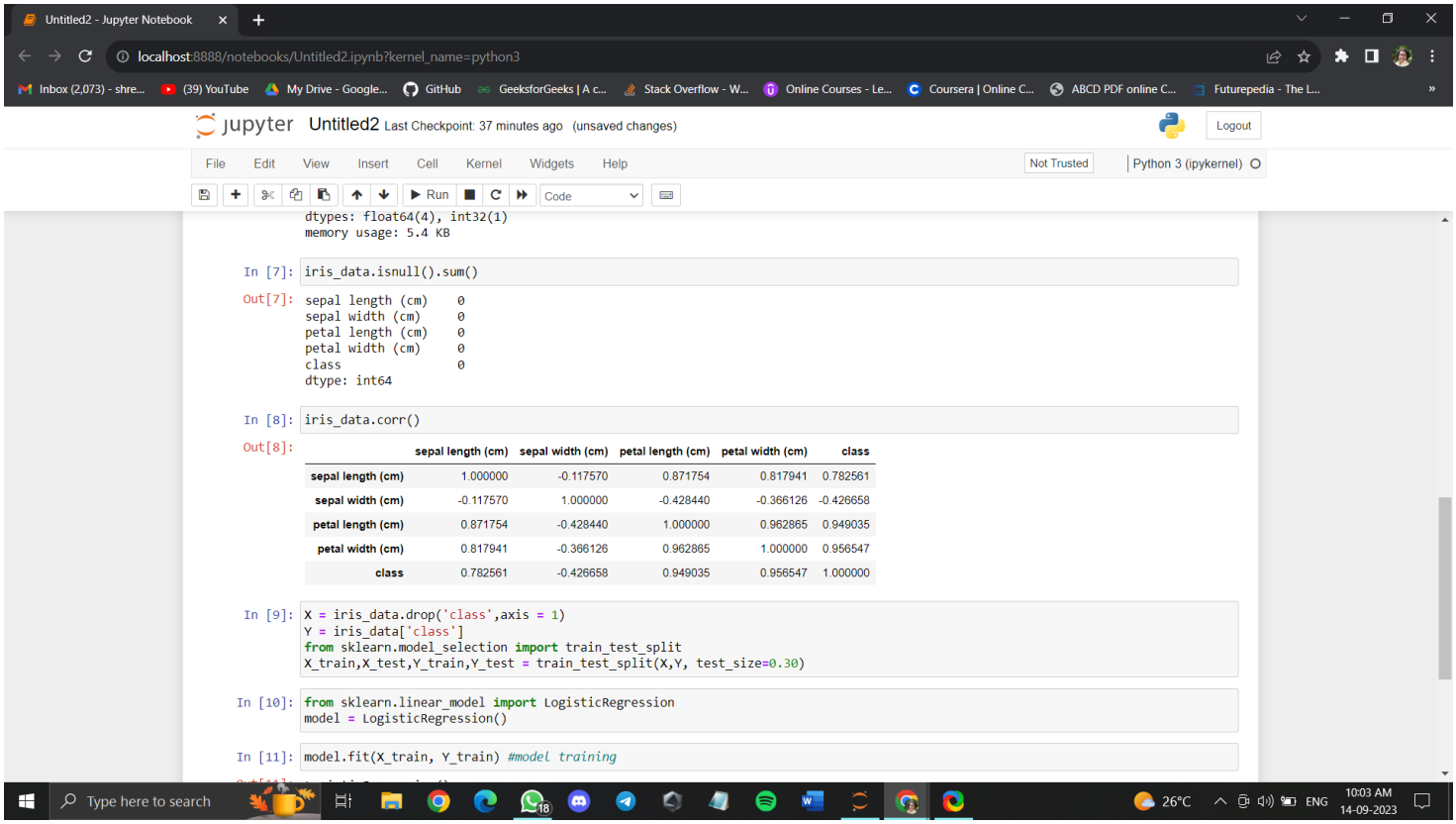
Conclusion:

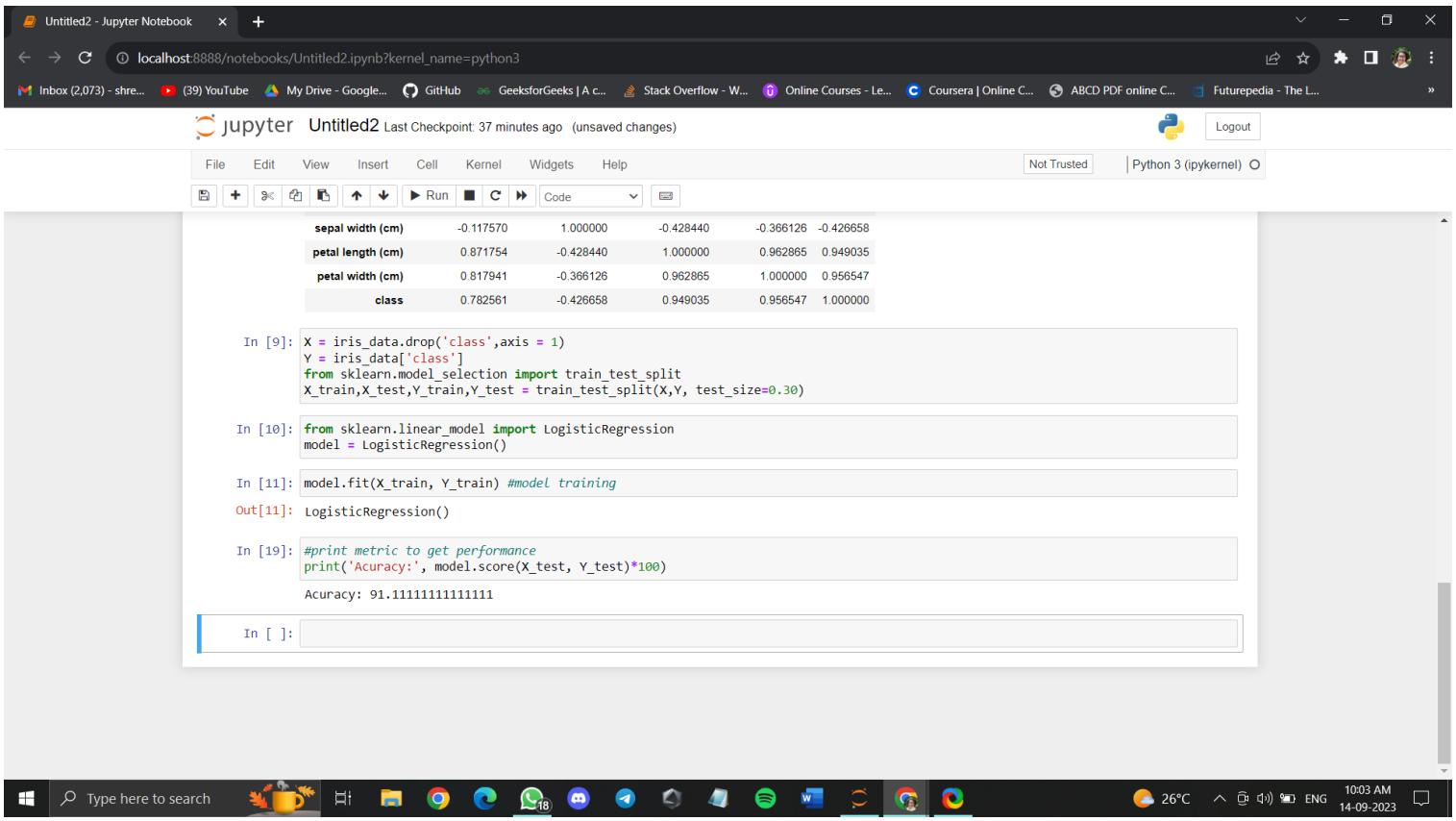
Post Lab Questions:

1. What are the limitations of a perceptron?
2. Explain Generalization.
3. How many training data patterns should be used to train a back propagation network.
4. How to determine the number of Hidden Layer Nodes.


Untitled2 - Jupyter Notebook
localhost:8888/notebooks/Untitled2.ipynb?kernel_name=python3
jupyter Untitled2 Last Checkpoint: 36 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
In [2]:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn.datasets
In [4]:
iris = sklearn.datasets.load_iris()
iris_data = pd.DataFrame(iris.data, columns=iris.feature_names)
iris_data['class'] = iris.target
iris_data.head()
Out[4]:
sepal length (cm) sepal width (cm) petal length (cm) petal width (cm) class
0 5.1 3.5 1.4 0.2 0
1 4.9 3.0 1.4 0.2 0
2 4.7 3.2 1.3 0.2 0
3 4.6 3.1 1.5 0.2 0
4 5.0 3.6 1.4 0.2 0
In [5]:
iris_data.describe()
Out[5]:
sepal length (cm) sepal width (cm) petal length (cm) petal width (cm) class
count 150.000000 150.000000 150.000000 150.000000 150.000000
mean 5.843333 3.057333 3.758000 1.199333 1.000000
std 0.828066 0.435866 1.765298 0.762238 0.819232
min 4.300000 2.000000 1.000000 0.100000 0.000000
25% 5.100000 2.800000 1.600000 0.300000 0.000000
Type here to search
26°C
ENG
10:03 AM
14-09-2023


Untitled2 - Jupyter Notebook
localhost:8888/notebooks/Untitled2.ipynb?kernel_name=python3
jupyter Untitled2 Last Checkpoint: 37 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
In [5]:
iris_data.describe()
Out[5]:
sepal length (cm) sepal width (cm) petal length (cm) petal width (cm) class
count 150.000000 150.000000 150.000000 150.000000 150.000000
mean 5.843333 3.057333 3.758000 1.199333 1.000000
std 0.828066 0.435866 1.765298 0.762238 0.819232
min 4.300000 2.000000 1.000000 0.100000 0.000000
25% 5.100000 2.800000 1.600000 0.300000 0.000000
50% 5.800000 3.000000 4.350000 1.300000 1.000000
75% 6.400000 3.300000 5.100000 1.800000 2.000000
max 7.900000 4.400000 6.900000 2.500000 2.000000
In [6]:
iris_data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
Column Non-Null Count Dtype
--- ------ -------------- -----
0 sepal length (cm) 150 non-null float64
1 sepal width (cm) 150 non-null float64
2 petal length (cm) 150 non-null float64
3 petal width (cm) 150 non-null float64
4 class 150 non-null int32
dtypes: float64(4), int32(1)
memory usage: 5.4 KB
In [7]:
iris_data.isnull().sum()
Out[7]:
sepal length (cm) 0
Type here to search
26°C
ENG
10:03 AM
14-09-2023


Untitled2 - Jupyter Notebook
localhost:8888/notebooks/Untitled2.ipynb?kernel_name=python3
jupyter Untitled2 Last Checkpoint: 37 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
dtypes: float64(4), int32(1)
memory usage: 5.4 KB
In [7]:
iris_data.isnull().sum()
Out[7]:
sepal length (cm) 0
sepal width (cm) 0
petal length (cm) 0
petal width (cm) 0
class 0
dtype: int64
In [8]:
iris_data.corr()
Out[8]:
sepal length (cm) sepal width (cm) petal length (cm) petal width (cm) class
sepal length (cm) 1.000000 -0.117570 0.871754 0.817941 0.782561
sepal width (cm) -0.117570 1.000000 -0.428440 -0.366126 -0.426658
petal length (cm) 0.871754 -0.428440 1.000000 0.962865 0.949035
petal width (cm) 0.817941 -0.366126 0.962865 1.000000 0.956547
class 0.782561 -0.426658 0.949035 0.956547 1.000000
In [9]:
X = iris_data.drop('class',axis = 1)
Y = iris_data['class']
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(X,Y, test_size=0.30)
In [10]:
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
In [11]:
model.fit(X_train, Y_train) #model training
Type here to search
26°C
ENG
10:03 AM
14-09-2023


Untitled2 - Jupyter Notebook
localhost:8888/notebooks/Untitled2.ipynb?kernel_name=python3
jupyter Untitled2 Last Checkpoint: 37 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
sepal width (cm) -0.117570 1.000000 -0.428440 -0.366126 -0.426658
petal length (cm) 0.871754 -0.428440 1.000000 0.962865 0.949035
petal width (cm) 0.817941 -0.366126 0.962865 1.000000 0.956547
class 0.782561 -0.426658 0.949035 0.956547 1.000000
In [9]:
X = iris_data.drop('class',axis = 1)
Y = iris_data['class']
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(X,Y, test_size=0.30)
In [10]:
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
In [11]:
model.fit(X_train, Y_train) #model training
Out[11]:
LogisticRegression()
In [19]:
#print metric to get performance
print('Acuracy:', model.score(X_test, Y_test)*100)
Acuracy: 91.11111111111111
In [ ]:
Type here to search
26°C
ENG
10:03 AM
14-09-2023

Exp 4 classify species of Iris flower using MLP

PAGE No.
DATE 14/9/23

* Post Lab questions-

Q1) What are the limitations of perceptron?

-> Limitations of perceptron -
1. Linear seperability
2. Lack of Hidden Layers
3. Binary outputs
4. sensitivity to Input scale
5. Inability to Learn Complex functions
6. Lack of Representation Learning
7. Unsuitable for Image & Speech Recognition.
8. Training challenges.

Q2) Explain Generalization

-> Generalization, in the context of machine learning, refers to a model's ability to perform well on new, unseen data that it has not been specifically trained on. It means that the model has learned underlying patterns and relationships in the training data without simply memorizing it. A good generalization implies that the model can make accurate predictions or classifications for a wide range of inputs beyond the training set, indicating its ability to capture.

PAGE No.
DATE / /

Q3) How many training data patterns should be used to train a back propagation network

-> The number of training data patterns needed for a back propagation neural network varies based on factors like problem complexity and network design. Generally, thousands to tens of thousands of diverse data points are often required, but it depends on the specific problem. overfitting should be avoided, so having enough data is crucial for good generalization. Experimentation & cross-validation help determine the right dataset size

Q4) How to determine the number of Hidden Layer Nodes

-> The no. of hidden layer Nodes are-
1. start simple
2. Gradually increase
3. Avoid overfitting
4. use of Rule of Thumb
5. Consider Architectural Variations
6. Regularization Techniques
7. Cross-validation
8. Domain knowledge

Final Year B. Tech (EE)

Trimester: X **Subject: AIML**

Name: Shreerang Mhatre **Class: TY**

Roll No: 52 **Batch: A3**

Experiment No: 05

Name of the Experiment: Linear Regression Using Python

Marks	Teacher's Signature with date

Performed on: 21/09/2023

Submitted on: 29/09/2023

Aim: Implement Linear Regression Using Gradient Descent Algorithm.

Prerequisite: Knowledge of Supervised Learning method, MLP, Activation function.

Objective:

To create and study the Neural Network by varying parameters.

1. Define Input and Output Variable
2. Define and custom Neural Network
3. Configure the network
4. Train the network to find output

Components and Equipment required:

Python software

Theory:

Linear Regression:

Linear regression is an approach for modeling relationship between a scalar dependent variable y and one or more independent variables. When there is single independent variable it is called simple linear regression and when there are more than one independent variables, it is called multiple linear regression. Linear regression is a statistical procedure for predicting the value of a dependent variable from an independent variable when the relationship between the variables can be described with linear model given by y = m * x + c.

Gradient Descent

In this method goal is to obtain parameters such that sum of squared error between target output and actual output is minimized.

Gradient descent is an iterative method. Algorithm starts with some set of values for our model parameters (weights and biases) and improves them slowly. Change in weight for each iteration is proportional to derivative of cost function w.r.t. the current weight value.

Thus algorithm moves in the direction of minima of cost function.

Gradient descent is an optimization algorithm that approaches a local minimum of a function by taking steps proportional to the negative of the gradient of the function as the current point.

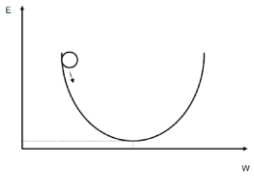


Figure 1: Error and global minima

To run gradient descent on error function, we first need to compute its gradient. The gradient will act like a compass and always point us downhill. To compute it, we will need to differentiate our error function. Since our function is defined by two parameters (w1 and w2), we will need to compute a partial derivative for each. These derivatives work out to be:

$$E(w_{ji}) = \frac{1}{2}(y_{tarj} - y_j)^2 \qquad y_j = f(a_j) = \sum_i w_{ji} x_i$$

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial w_{ji}} = -(y_{tarj} - y_j) x_i = -\delta x_i$$

We now have all the tools needed to run gradient descent. We can initialize our search to start at any pair of w1 and w2 values (i.e., any line) and let the gradient descent algorithm march downhill on our error function towards the best line. Each iteration will update w1 and w2 to a line that yields slightly lower error than the previous iteration.

The learning Rate variable controls how large of a step we take downhill during each iteration. If we take too large of a step, we may step over the minimum. However, if we take small steps, it will require much iteration to arrive at the minimum.

Below are some snapshots of gradient descent running for 2000 iterations for the example problem? We start out at point w1 = -1 w2 = 0. Each iteration w1 and w2 are updated to values that yield slightly lower error than the previous iteration. The left plot displays the current location of the gradient descent search (blue dot) and the path taken to get there (black line). The right plot displays the corresponding line for the current search location. Eventually we ended up with a pretty accurate fit.

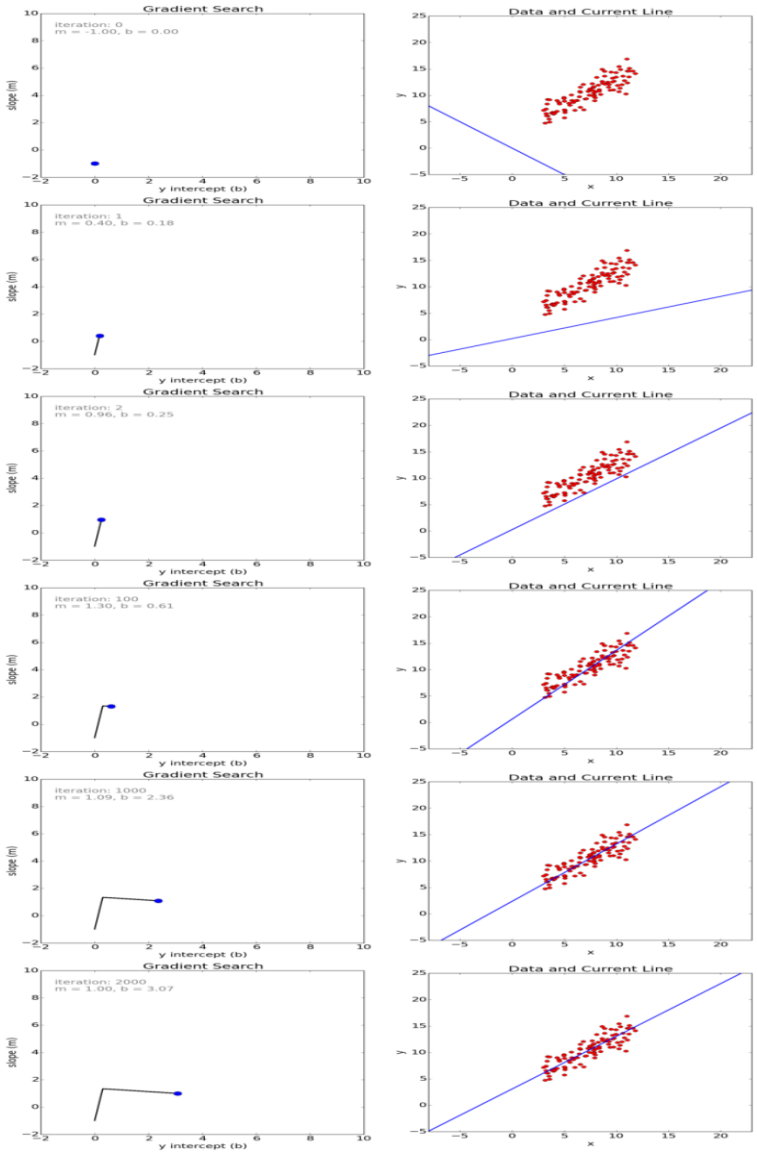


Figure 2: Different iterations of gradient descent

We can also observe how the error changes as we move toward the minimum. A good way to ensure that gradient descent is working correctly is to make sure that the error decreases for each iteration. Below is a plot of error values for the first 100 iterations of the above gradient search.

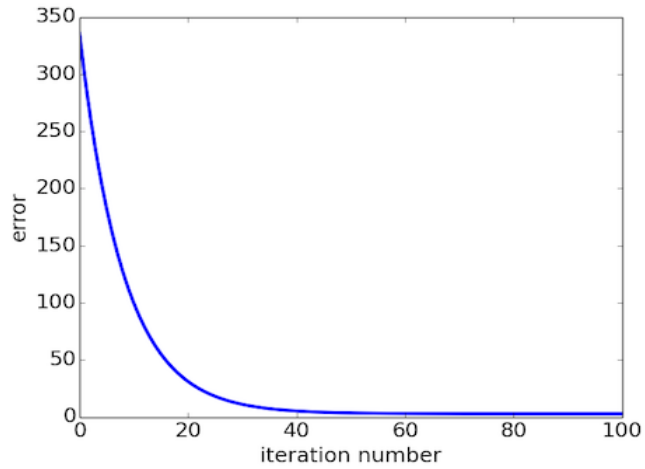


Figure 3: MSE Vs Epochs

While the model in above example was a line, the concept of minimizing a cost function to tune parameters also applies to regression problems that use higher order polynomials and other problems found around the machine learning world.

Procedure

Algorithm:

Step 0: Initialize weights w0,w1.(Set to small random values) and α (learning rate).

Step 1: While stopping condition is false, do steps 2-8.

Step 2: For each training pair with (x_i,y_i), do steps 3-6.

Step 3: Each input unit (X_i, i = 1…….n) receives input signal x_i.

Step 4: Estimate E(w), where $E(w)=1/(2M)* \sum ((w_0+w_1*x_i)-y_i)^2$

Step 5: Estimate new values of $w_{0(new)}$, $w_{1(new)}$ using the following equations and store it in a temporary variable.

Step 6: Perform simultaneous update of $w_{0(new)}=Temp_0$

$w_{1(new)}=Temp_1$

Step 7: Update Mean square error, until the Error function E(w) is minimized

Step 8: Stop.

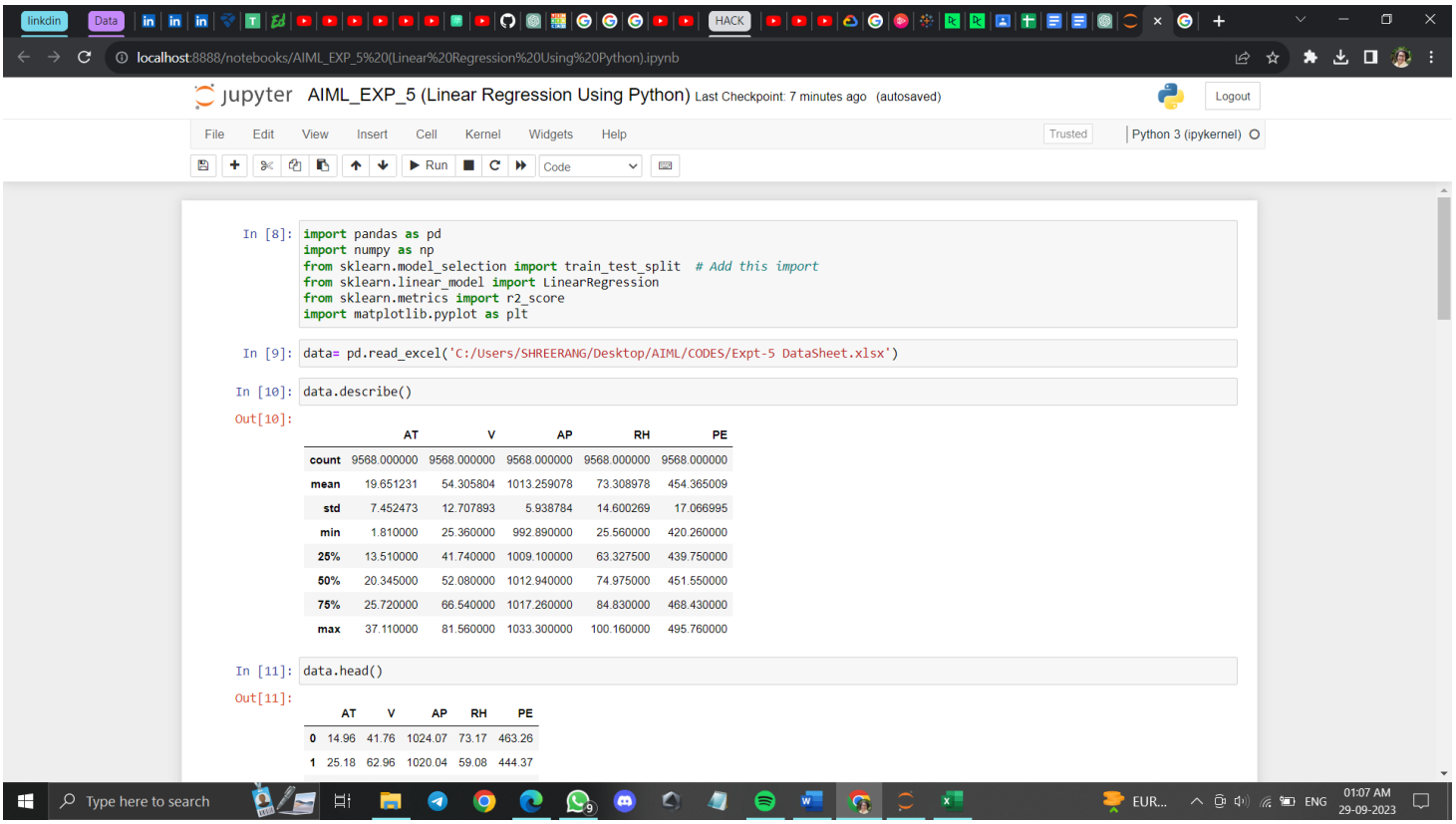
Python Code

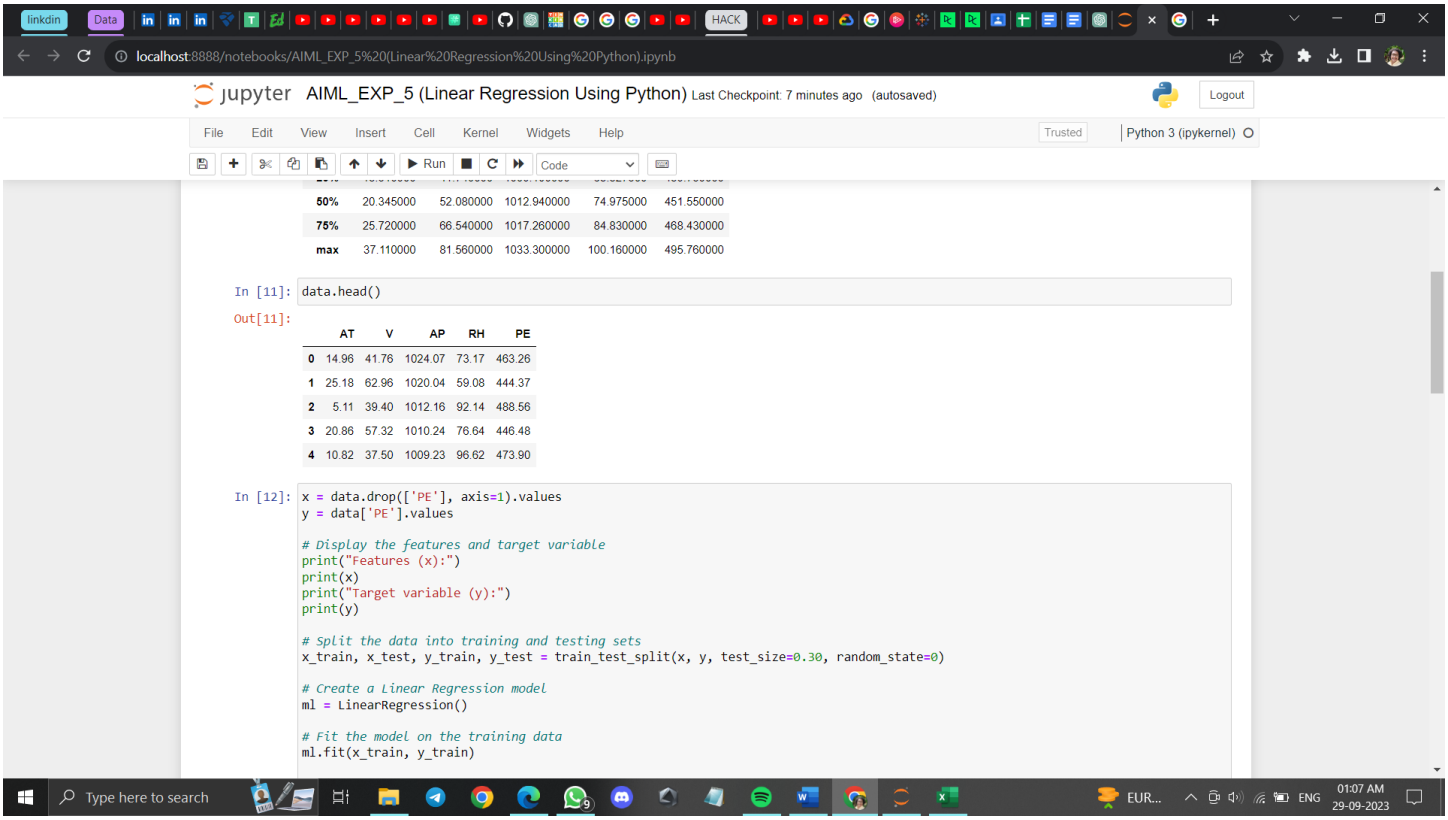
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
data = pd.read_csv('age1.csv')
data.head(5)
data.describe()
plt.scatter(data['AgeGroup'],data['TotalCases'])
plt.xlabel("Age Group of people")
plt.ylabel("Total Confirmed cases")
plt.title("Corona with age Prediction")
data_n=data.values
m=data_n[:,0].size
x=data_n[:,0].reshape(m,1)
y=data_n[:,2].reshape(m,1)
xTrain,xTest,yTrain,yTest=train_test_split(x,y,test_size=0.1,random_state=0)
yTrain
linearRegressor=LinearRegression()
linearRegressor.fit(xTrain,yTrain)
yprediction=linearRegressor.predict(xTest)
xTest,yprediction
import matplotlib.pyplot as plot
plot.scatter(xTest,yTest,color='red')
plot.plot(xTrain,linearRegressor.predict(xTrain))
plot.title('Age vs Corona Confirmed cases')
plot.xlabel('Age Group No')

plot.ylabel('Confirmed cases')
plot.show()
```

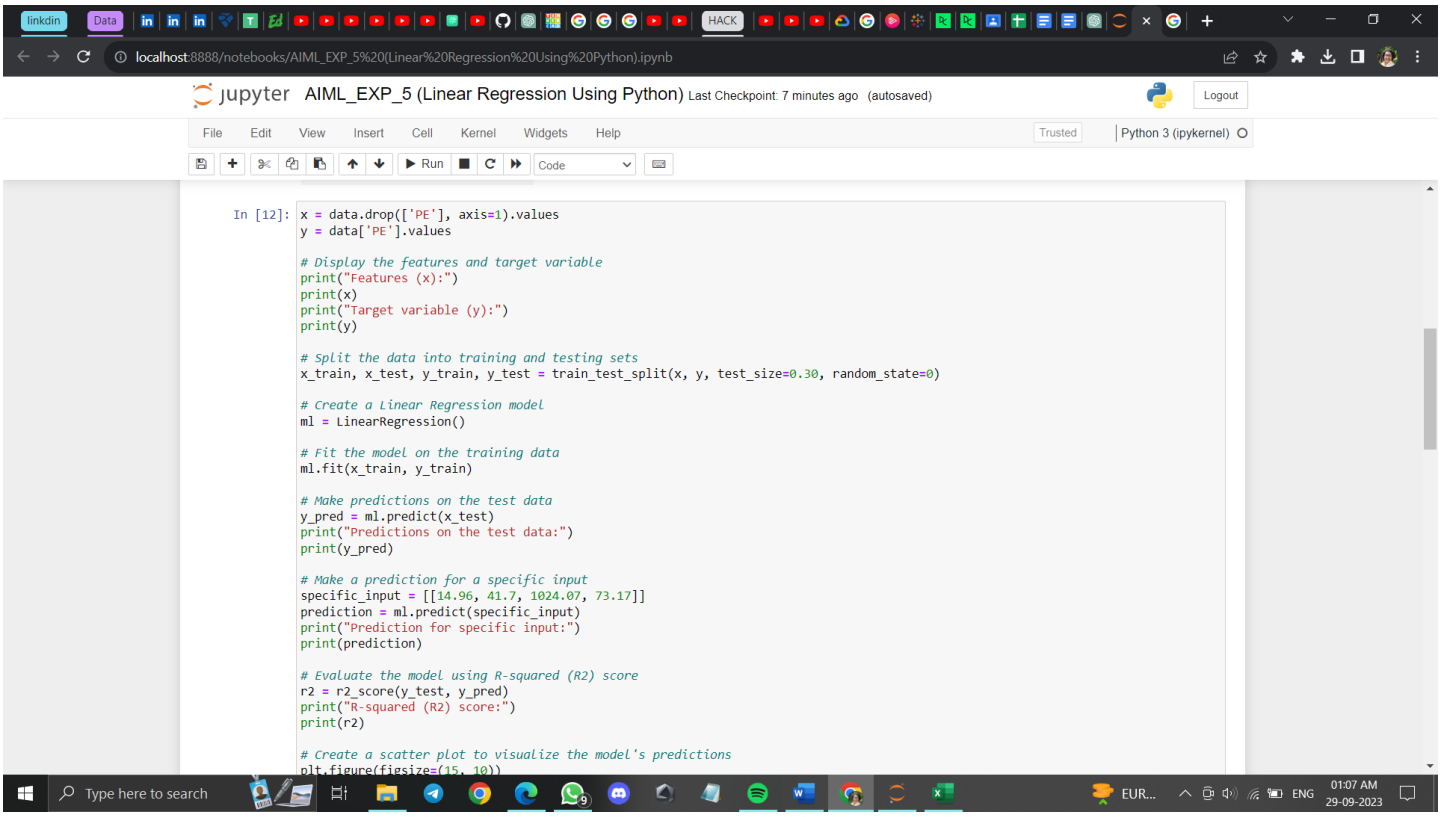
Conclusions:

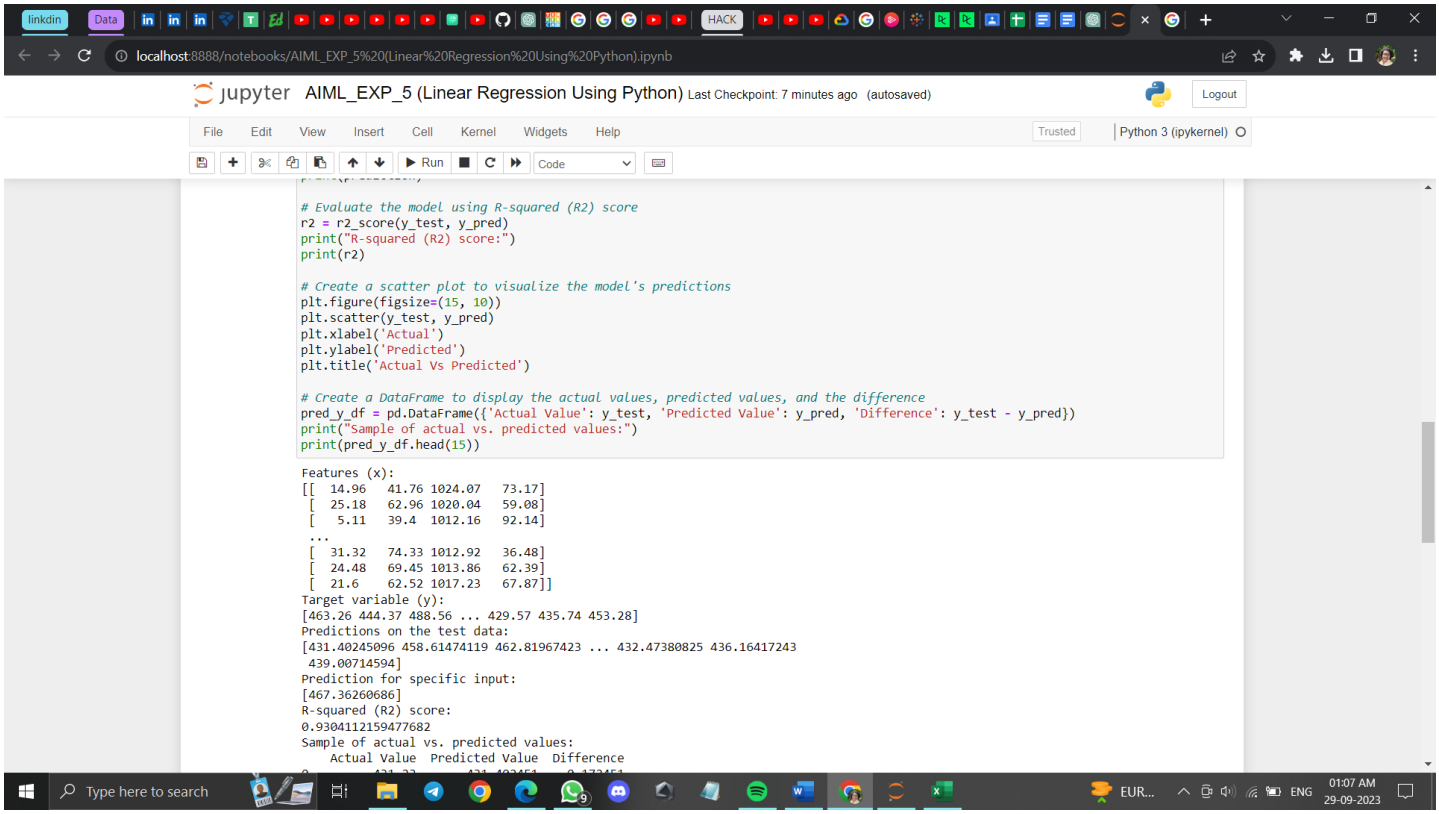
Post Lab Questions:

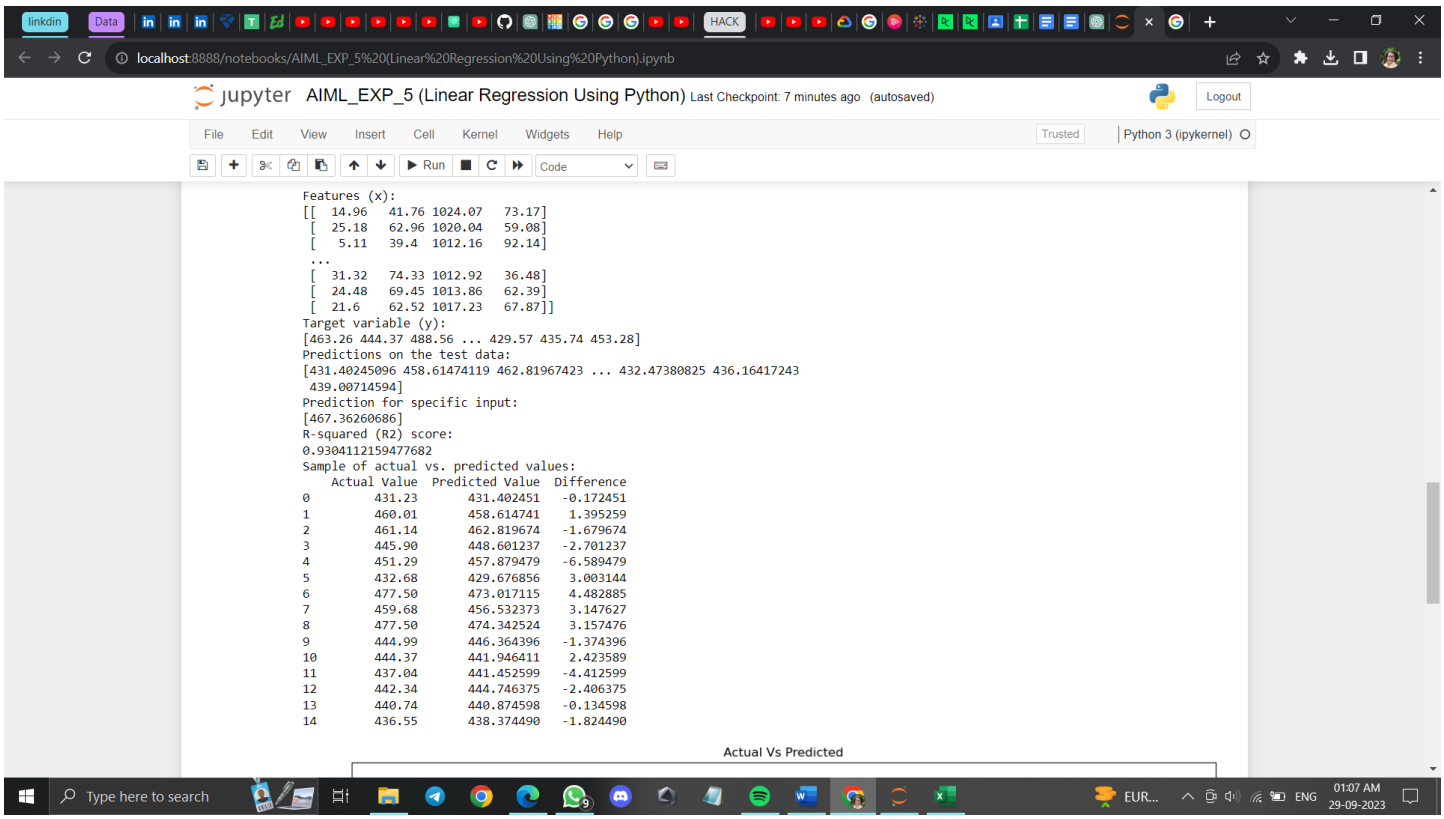
1. 1. Explain the Gradient Descent algorithm.
2. Explain significance of learning rate in case of gradient descent algorithm?
3. What is global minima and local minima?




localhost:8888/notebooks/AIML_EXP_5%20(Linear%20Regression%20Using%20Python).ipynb
jupyter AIML_EXP_5 (Linear Regression Using Python) Last Checkpoint: 7 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
50% 20.345000 52.080000 1012.940000 74.975000 451.550000
75% 25.720000 66.540000 1017.260000 84.830000 468.430000
max 37.110000 81.560000 1033.300000 100.160000 495.760000
In [11]: data.head()
Out[11]:
AT V AP RH PE
0 14.96 41.76 1024.07 73.17 463.26
1 25.18 62.96 1020.04 59.08 444.37
2 5.11 39.40 1012.16 92.14 488.56
3 20.86 57.32 1010.24 76.64 446.48
4 10.82 37.50 1009.23 96.62 473.90
In [12]:
x = data.drop(['PE'], axis=1).values
y = data['PE'].values
# Display the features and target variable
print("Features (x):")
print(x)
print("Target variable (y):")
print(y)
# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30, random_state=0)
# Create a Linear Regression model
ml = LinearRegression()
# Fit the model on the training data
ml.fit(x_train, y_train)
Type here to search
EUR...
ENG
01:07 AM
29-09-2023


localhost:8888/notebooks/AIML_EXP_5%20(Linear%20Regression%20Using%20Python).ipynb
jupyter AIML_EXP_5 (Linear Regression Using Python) Last Checkpoint: 7 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
In [12]:
x = data.drop(['PE'], axis=1).values
y = data['PE'].values
# Display the features and target variable
print("Features (x):")
print(x)
print("Target variable (y):")
print(y)
# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30, random_state=0)
# Create a Linear Regression model
ml = LinearRegression()
# Fit the model on the training data
ml.fit(x_train, y_train)
# Make predictions on the test data
y_pred = ml.predict(x_test)
print("Predictions on the test data:")
print(y_pred)
# Make a prediction for a specific input
specific_input = [[14.96, 41.7, 1024.07, 73.17]]
prediction = ml.predict(specific_input)
print("Prediction for specific input:")
print(prediction)
# Evaluate the model using R-squared (R2) score
r2 = r2_score(y_test, y_pred)
print("R-squared (R2) score:")
print(r2)
# Create a scatter plot to visualize the model's predictions
plt.figure(figsize=(15, 10))
Type here to search
EUR...
ENG
01:07 AM
29-09-2023

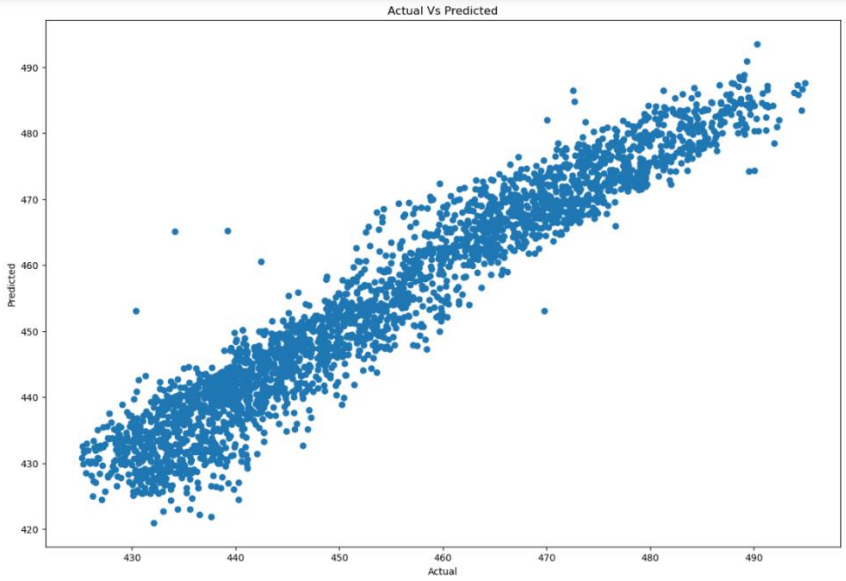

jupyter AIML_EXP_5 (Linear Regression Using Python) Last Checkpoint: 7 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
# Evaluate the model using R-squared (R2) score
r2 = r2_score(y_test, y_pred)
print("R-squared (R2) score:")
print(r2)
# Create a scatter plot to visualize the model's predictions
plt.figure(figsize=(15, 10))
plt.scatter(y_test, y_pred)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual Vs Predicted')
# Create a DataFrame to display the actual values, predicted values, and the difference
pred_y_df = pd.DataFrame({'Actual Value': y_test, 'Predicted Value': y_pred, 'Difference': y_test - y_pred})
print("Sample of actual vs. predicted values:")
print(pred_y_df.head(15))
Features (x):
[[14.96 41.76 1024.07 73.17]
[25.18 62.96 1020.04 59.08]
[5.11 39.4 1012.16 92.14]
...
[31.32 74.33 1012.92 36.48]
[24.48 69.45 1013.86 62.39]
[21.6 62.52 1017.23 67.87]]
Target variable (y):
[463.26 444.37 488.56 ... 429.57 435.74 453.28]
Predictions on the test data:
[431.40245096 458.61474119 462.81967423 ... 432.47380825 436.16417243
439.00714594]
Prediction for specific input:
[467.36260686]
R-squared (R2) score:
0.9304112159477682
Sample of actual vs. predicted values:
Actual Value Predicted Value Difference
Type here to search
EUR...
ENG
01:07 AM
29-09-2023


jupyter AIML_EXP_5 (Linear Regression Using Python) Last Checkpoint: 7 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
Features (x):
[[14.96 41.76 1024.07 73.17]
[25.18 62.96 1020.04 59.08]
[5.11 39.4 1012.16 92.14]
...
[31.32 74.33 1012.92 36.48]
[24.48 69.45 1013.86 62.39]
[21.6 62.52 1017.23 67.87]]
Target variable (y):
[463.26 444.37 488.56 ... 429.57 435.74 453.28]
Predictions on the test data:
[431.40245096 458.61474119 462.81967423 ... 432.47380825 436.16417243
439.00714594]
Prediction for specific input:
[467.36260686]
R-squared (R2) score:
0.9304112159477682
Sample of actual vs. predicted values:
Actual Value Predicted Value Difference
0 431.23 431.402451 -0.172451
1 460.01 458.614741 1.395259
2 461.14 462.819674 -1.679674
3 445.90 448.601237 -2.701237
4 451.29 457.879479 -6.589479
5 432.68 429.676856 3.003144
6 477.50 473.017115 4.482885
7 459.68 456.532373 3.147627
8 477.50 474.342524 3.157476
9 444.99 446.364396 -1.374396
10 444.37 441.946411 2.423589
11 437.04 441.452599 -4.412599
12 442.34 444.746375 -2.406375
13 440.74 440.874598 -0.134598
14 436.55 438.374490 -1.824490
Actual Vs Predicted
Type here to search
EUR...
ENG
01:07 AM
29-09-2023

jupyter AIML_EXP_5 (Linear Regression Using Python) Last Checkpoint: 7 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code



In [ ]:

Exp5- Linear Regression Using Python

PAGE No.
DATE 21/9/23

* Post Lab Questions-

(1) Explain the Gradient Descent algorithm.

-> Gradient Descent is an optimization algorithm used in machine learning to find the best parameters for a model by minimizing a cost function. Here's a consie explanation of the algorithm-

1. Initialize - start with initial parameter values.
2. calculate Gradient - Compute the gradient (slope) of the cost function with respect to each parameter.
3. Update Parameters - Adjust parameters in the opposite direction of the gradient by a small step.
4. Repeat - Continue steps 2 & 3 iteratively until convergence or a set number of iterations.
5. Convergence - Monitor cost function to check if it is minimized, indicating optimal parameters.

Q2) Explain significance of learning rate in case of gradient descent algorithm?

→ The learning rate in the Gradient Descent algorithm plays a pivotal role in the training process of machine learning models. It determines the size of the steps taken to update model parameters during optimization. A higher learning rate can lead to faster convergence but risks overshooting and divergence, while a lower rate provides stability by result in slow convergence. Therefore, selecting the right learning rate is essential; it serves as a critical hyperparameter that must be tuned for each specific problem and dataset. A well-chosen learning rate balances the trade-off between training speed and stability, ensuring the algorithm efficiently finds optimal parameter values and avoids getting stuck in local minima

Q.3) What is global minima & local minima?

-> ① Global minima -

The global minima is the lowest point in the entire function, representing the absolute lowest value of the function across its entire domain. Finding the global minimum is the ~~privacy~~ primary objective in optimization problems.

② Local minima -

Local minima are points within a function where the value is lower than in the surrounding neighborhood but not necessarily the absolute lowest value in the entire function. They can be problematic in optimization as they may trap algorithms if the search starts from a point near a local minimum, preventing them from reaching the global minimum.

Final Year B. Tech (EE)

Trimester: I **Subject:** **Artificial Intelligence and Machine Learning**

Name: Shreerang Mhatre **Class: Ty**

Roll No: 52 **Batch: A3**

Experiment No: 06

Name of the Experiment: Implement and test MLP trained with back – propagation algorithm

Marks	Teacher's Signature with date

Performed on: 11/10/2023

Submitted on: 11/10/2023

Aim: To create a multilayer neural network and train with back propagation algorithm using Python.

Prerequisite: Knowledge of MLP, gradient descent method, Least Mean Square Error

Objective:

To create a multi-layer neural network and train with back propagation algorithm using Python Programming.

Components and Equipment required:

SkLearn Python module, Python software, NumPy and Panda Libraries

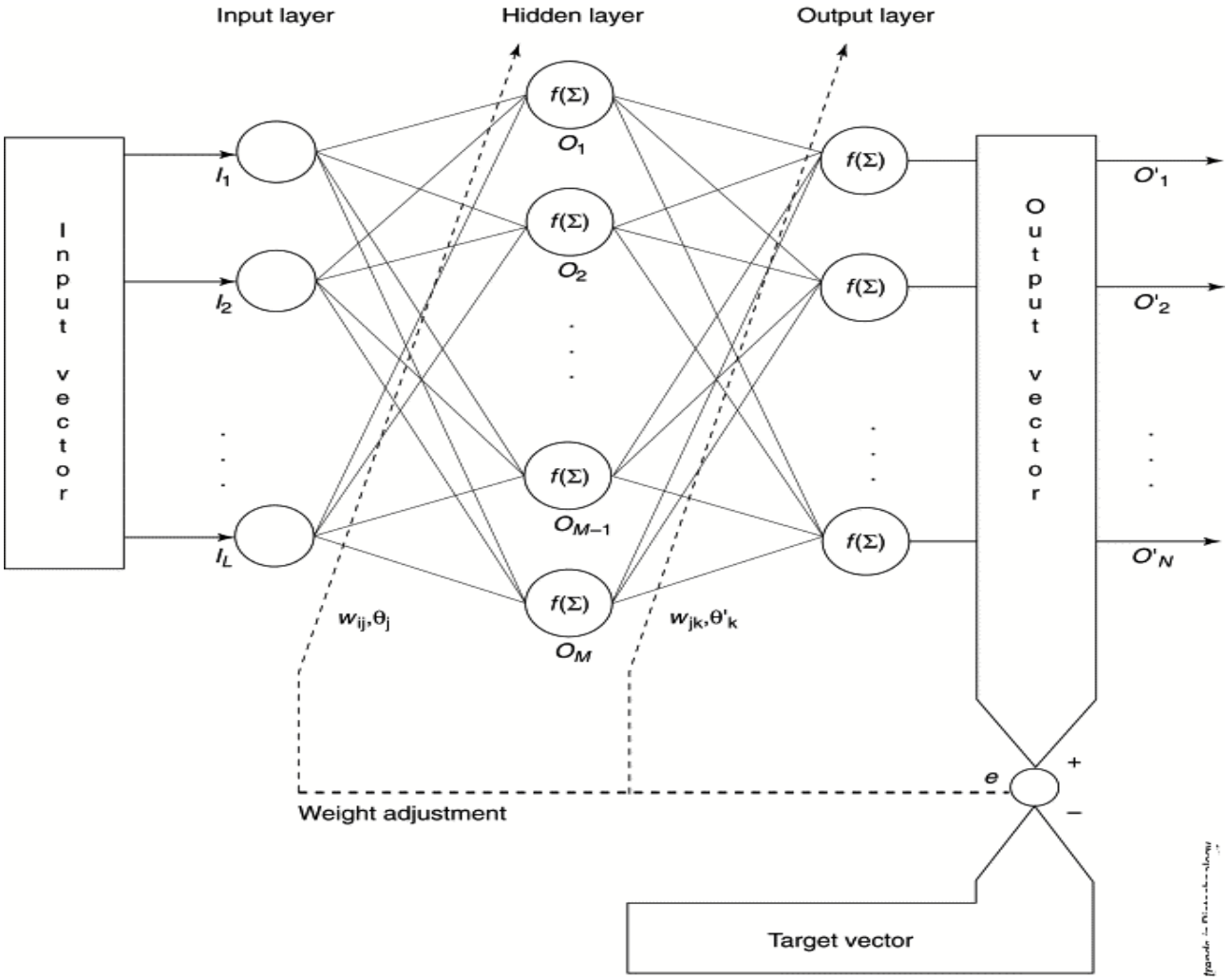
Theory

Multilayer feed forward networks are an important class of neural networks. Typically, the network consists of sensory units (source nodes) that constitute the input layer, one or more hidden layers of computation nodes and an output layer of computation nodes. The input signal propagates through the network in the forward direction on a layer-by-layer basis. These neural networks are commonly known as Multilayer Perceptron's (MLPs)

Multilayer perceptron's have been applied successfully to solve some difficult and diverse problems by training them in a supervised manner with the highly popular algorithm known as the error **back-propagation algorithm**. This algorithm is based on the error-correction learning rule.

Basically error back-propagation learning consists of two passes through the different layers of the network: a forward pass and a backward pass. In the forward pass an activity pattern (input vector) is applied to the sensory nodes of the network and its effect propagates through the network layer by layer. Finally, a set of outputs is produced as the actual response of the network. During the forward pass all synaptic weights of the network are fixed. During the backward pass, all synaptic weights are adjusted in accordance with an error correction rule. Specifically, the actual response of the network is subtracted from the network desired (target) response to produce an error signal. This error signal is then propagated backward through against the direction of the synaptic weights and hence the name "**error back propagation**"..

Building our model



Procedure

Step 0: Initialize weights. (Set to small random values)

Step 1: While stopping condition is false, do steps 2-9.

Step 2: For each training pair, do steps 3-8.

***Feed forward*:**

Step 3: Each input unit (X_i, i = 1……. n) receives input signal $x_{i\ and}$ broadcasts this signal to all units in the hidden layer above (the hidden units.

Step 4: Each hidden unit ($Z_{j,\ j}$ = 1……. p) sums its weighted input signals.

$Z_{inj} = v_{oj} + \sum^{xi} v_{ij\ ;}$ i=1…..n

applies its activation function to compute its output signal.

$$Z_j = f(Z_{inj})$$

and sends this signal to all units in the layer above (output units).

Step 5: Each output unit (Y_k =1……m) sums its weighted input signals.

$$Y_{ink} = w_{ok} + \sum^{zj} w_{jk\ ;}\ i=1…….n$$

and applies its activation function to compute its output signal.

***Back propagation of error*:**

Step 6: Each output unit receives a target pattern corresponding to the input training pattern, computes its error information term.

$$\delta_k = (t_k - y_k)\ f'(y\_in_k)$$

calculates its bias correction term (used to update w_{jk})

$$\Delta\ \mathbf{w}_{jk} = \alpha\delta_k z_j$$

calculates its bias correction term

$$\Delta\mathbf{w}_{ok=}\ \alpha\delta_k$$

and sends δ_k to units in the layer below.

Step 7: Each hidden unit sums its delta inputs (from above in the layer),

$$\Delta\_inj = \sum_{k=1}^{m} \delta_k w_{jk,}$$

Multiplies by the derivative of its activation function to calculate its error information term.

$$\Delta_j = \delta\_inj'(z\_inj),$$

Calculates its weight correction term

$$\Delta v_{jk} = \alpha \delta_k x_j$$

and calculates its bias correction term

$$\Delta v_{oj=} \alpha \delta_j$$

Update weights and biases

Step 8: Each output unit updates its biases and weights

$$w_{jk}(new) = w_{jk(old)} + \Delta w_{jk}$$

each hidden unit updates its biases and weights

$$v_{jk}(new) = v_{jk(old)} + \Delta v_{jk}$$

Test stopping condition.

Python Programming

```
import numpy as np

# X = (hours sleeping, hours studying), y = test score of the student

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)

y = np.array(([92], [86], [89]), dtype=float)

# scale units

X = X/np.amax (X, axis=0) #maximum of X array

y = y/100 # maximum test score is 100
```

```
class NeuralNetwork(object):

    def __init__(self):

        #parameters

        self.inputSize = 2

        self.outputSize = 1

        self.hiddenSize = 3

            #weights

        self. W1 = np.random.randn(self.inputSize, self.hiddenSize) # (2x3) weight matrix from input to
hidden layer

        self. W2 = np.random.randn(self.hiddenSize, self.outputSize) # (3x1) weight matrix from hidden to
output layer

         def feedForward(self, X):

        #forward propogation through the network

      self.z = np.dot(X, self.W1) #dot product of X (input) and first set of weights (3x2)

        self. z2 = self.sigmoid(self.z) #activation function


    self. z3 = np.dot (self. z2, self. W2) #dot product of hidden layer (z2) and second set of weights (3x1)

        output = self.sigmoid(self.z3)

        return output

          def sigmoid (self, s, deriv=False):

        if (deriv == True):

            return s * (1 - s)

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

       def backward (self, X, y, output):

        #backward propogate through the network

        self.output_error = y - output # error in output

        self.output_delta = self.output_error * self.sigmoid(output, deriv=True)

        self. z2_error = self.output_delta.dot (self. W2. T) #z2 error: how much our hidden layer weights
contribute to output error
```

```
        self. z2_delta = self. z2_error * self. Sigmoid (self. z2, deriv=True) #applying derivative of sigmoid to
    z2 error

            self. W1 += X.T.dot (self. z2_delta) # adjusting first set (input -> hidden) weights

self. W2 += self.z2.T.dot (self. output delta) # adjusting second set (hidden -> output) weights

          def train (self, X, y):

 output = self.feedForward(X)

        self.backward(X, y, output)

    NN = Neural Network ()

    for i in range (1000): #trains the NN 1000 times

      if (i % 100 == 0):

         print ("Loss: " + str(np.mean(np.square(y - NN.feedForward(X)))))

      NN.train(X, y)

       print ("Input: " + str(X))

    print ("Actual Output: " + str(y))

    print ("Loss: " + str(np.mean(np.square(y - NN.feedForward(X)))))

    print("\n")

    print ("Predicted Output: " + str(NN.feedForward(X)))
```

Output

```
Loss: 0.0002414175695890420 4

Loss: 0.0002154409437336494 8

Loss: 0.0001960050170361402 6

 Loss: 0.000179502381372854

Loss: 0.0001653813997472701 2

Loss: 0.000153207336120599 3

Loss: 0.0001426350635498208 2

Loss: 0.000133389143354652

Loss: 0.0001252485008011045 8
```

```
Input: [[0.66666667 1.] [0.33333333 0.55555556]

[1. 0.66666667]] Actual Output: [[0.92] [0.86] [0.89]] Loss:
0.00011803465359404784

Predicted Output: [[0.90612361] [0.87271003] [0.89007064]]
```

Conclusion:

Post Lab Questions:

1. Explain the method to initialize weights for a Backpropagation network.
2. Explain the choice of learning rate parameter.
3. Explain Generalization.
4. How many training data patterns should be used to train a backpropagation network?
5. How to determine the number of Hidden Layer Nodes?

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

In [3]:
```
import numpy as np

#Data Preparation
x=np.array(([2,9],[1,5],[3,6]),dtype=float)
y=np.array(([92],[86],[89]),dtype=float)
```

In [4]:
```
# Scale units
x= x/ np.amax(x,axis=0)
#Maximum test score=100
y=y/100
print(x)
print(y)
```

```
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
[[0.92]
 [0.86]
 [0.89]]
```

In [8]:
```
#Neural Network Class
class NeuralNetwork(object):
    def __init__(self):
        #Parameters
        self.inputSize=2
        self.outputSize=1
        self.hiddenSize=3
        #Weights
        self.W1= np.random.randn(self.inputSize, self.hiddenSize)
        self.W2= np.random.randn(self.hiddenSize, self.outputSize)

    def feedForward(self, X):
        self.z=np.dot(X, self.W1)
        self.z2=self.sigmoid(self.z)

        self.z3=np.dot(self.z2, self.W2)
        output=self.sigmoid(self.z3)
        return output

    def sigmoid(self, s, deriv=False):
        if deriv:
            return s*(1-s)
        return 1/ (1+ np.exp(-s))

    def backward(self, X, y, output):
        self.output_error=y-output
        self.output_delta=self.output_error*self.sigmoid(output,deriv=True)
        self.z2_error=self.output_delta.dot(self.W2.T)
        self.z2_delta=self.z2_error*self.sigmoid(self.z2,deriv=True)

        #Update weights
        self.W2+= self.z2.T.dot(self.output_delta)
        self.W1+=X.T.dot(self.z2_delta)

    def train(self,X,y):
        output=self.feedForward(X)
        self.backward(X,y,output)
```

In [9]:
```
#Training the Neural Network
NN= NeuralNetwork()
for i in range(1000):
    if i%100==0:
        print("Loss:" + str(np.mean(np.square(y - NN.feedForward(x)))))
    NN.train(x,y)
```

```
Loss:0.4254937933066482
Loss:0.0001992178622685607 4
Loss:0.0001948173231228523
Loss:0.0001922317381025274 3
Loss:0.0001897257467569777 2
Loss:0.0001872948842807154 8
Loss:0.0001849357010047549 3
Loss:0.0001826449333111930 4
Loss:0.0001804194934161471 3
Loss:0.0001782564590897682 4
```

In [10]:
```
#Final Result
print("Input: "+str(x))
print("Actual output: "+ str(y))
print("Loss: "+str(np.mean(np.square(y-NN.feedForward(x)))))
print("\n")
print("Predicted Output: "+ str(NN.feedForward(x)))
```

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

In [9]:
```
#Training the Neural Network
NN= NeuralNetwork()
for i in range(1000):
    if i%100==0:
        print("Loss:" + str(np.mean(np.square(y - NN.feedForward(x)))))
    NN.train(x,y)
```

```
Loss:0.4254937933066482
Loss:0.0001992178622685607 4
Loss:0.0001948173231228523
Loss:0.0001922317381025274 3
Loss:0.0001897257467569777 2
Loss:0.0001872948842807154 8
Loss:0.0001849357010047549 3
Loss:0.0001826449333111930 4
Loss:0.0001804194934161471 3
Loss:0.0001782564590897682 4
```

In [10]:
```
#Final Result
print("Input: "+str(x))
print("Actual output: "+ str(y))
print("Loss: "+str(np.mean(np.square(y-NN.feedForward(x)))))
print("\n")
print("Predicted Output: "+ str(NN.feedForward(x)))
```

```
Input: [[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual output: [[0.92]
 [0.86]
 [0.89]]
Loss: 0.00017615306388121677


Predicted Output: [[0.90113699]
 [0.8723686 ]
 [0.89443436]]
```

In [ ]:

Exp-6 - MLP trained with back-propagation

PAGE No.
DATE 11/10/22

* Past Lab questions -

Q1) Explain the method to initialize weights for a Back propagation network.

→ Back propagation neural network is an important step to ensure the network learns effectively. There are several methods to initialize weights, and the choice can impact training speed and convergence. One common method is to use small random values for the initial weights.

1. Random Initialization
2. Bias Initialization
3. Xavier / Glorot Initialization
4. He Initialization
5. Custom Initialization.

Q2) Explain the choice of learning rate parameter.

→ The learning rate in training neural networks is a pivotal hyperparameter that dictates the step size at which the model adjusts its weights during optimization. Selecting the ideal learning

rate necessitates careful consideration and often involves methods like grid search or learning rate scheduling. If the learning rate is too high, the model can diverge or oscillate, while a rate that is too low can result in slow convergence and getting stuck in local minima. Learning rate decay and adaptive optimization algorithms offer strategies to strike a balance between fast initial convergence and fine-tuning. Experimentation and continuous monitoring of training progress, including validation performance, are essential to find the most suitable learning rate for a given neural network architecture, dataset, & problem.

Q3) Explain Generalization.

-> Generalization in machine learning refers to the model's ability to perform well on unseen or new data that it hasn't been trained on. It's a fundamental goal of model training, indicating that the model has learned the underlying patterns and relationships within the training data

without merely memorizing it. A well-generalized model can make accurate predictions or classifications for various inputs, not just those it has seen during training.

Q4) How many training data patterns should be used to train a backpropagation network?

→ The number of training data patterns required to train a back propagation network depends on various factors, including the complexity of the problem, the architecture of the neural network, and the desired level of generalization. Some general guidlines -

1. Sufficiency.
2. Complexity of the Problem
3. Data Augmentation
4. Cross-Validation
5. Regularization.

Thus, there's no fixed number of training data patterns that applies universally. The goal is to have enough data to capture the underlying patterns of the problem and achieve good generalization.

PAGE No.
DATE / /

(Q5) How to determine the number of Hidden Layer Nodes?

→ Determining the number of hidden layer nodes in a neural network is a critical aspect of its architecture design. While there is no one-size-fits-all rule, some general guidelines can help. Firstly, start with a minimal number of nodes and gradually increase as needed. Too few nodes can lead to under fitting, while too many may result in over fitting. Consider the complexity of your problem and dataset size more complex problems often require more nodes. Ultimately the number of hidden layer nodes should be chosen based on the specific problem data, and empirical performance evaluation.

Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

Final Year BTech. (EE)

Semester: 5 **Subject: AIML**

Name: Shreerang Mhatre **Class: TY**
Roll No.: 52 **Batch: A3**

Experiment No. 7

Aim: Implementation of Simple Genetic Algorithm

Objective:

To get familiarize with Mathematical foundations for Genetic algorithm, operator.
To study the Applications of Genetic Algorithms

Software Required:

MATLAB

Theory:

Genetic algorithm is a search technique used in computing to find true or approximate solutions to approximate solutions to optimization & search problems.

Genetic algorithms are inspired by Darwin's theory about evolution. Solution to a problem solved by genetic algorithms is evolved.

Algorithm is started with a set of solutions (represented by chromosomes) called population. Solutions from one population are taken and used to form a new population.

This is motivated by a hope, that the new population will be better than the old one. Solutions which are selected to form new solutions (offspring) are selected according to their fitness - the more suitable they are the more chances they have to reproduce. This is repeated until some condition (for example number of populations or improvement of the best solution) is satisfied

Outline of the Basic Genetic Algorithm

1. **[Start]** Generate random population of *n* chromosomes (suitable solutions for the problem)

2. **[Fitness]** Evaluate the fitness *f(x)* of each chromosome *x* in the population

3. **[New population]** Create a new population by repeating following steps until the new population is complete

1. **[Selection]** Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)

2. **[Crossover]** With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.

3. **[Mutation]** With a mutation probability mutate new offspring at each locus (position in chromosome).

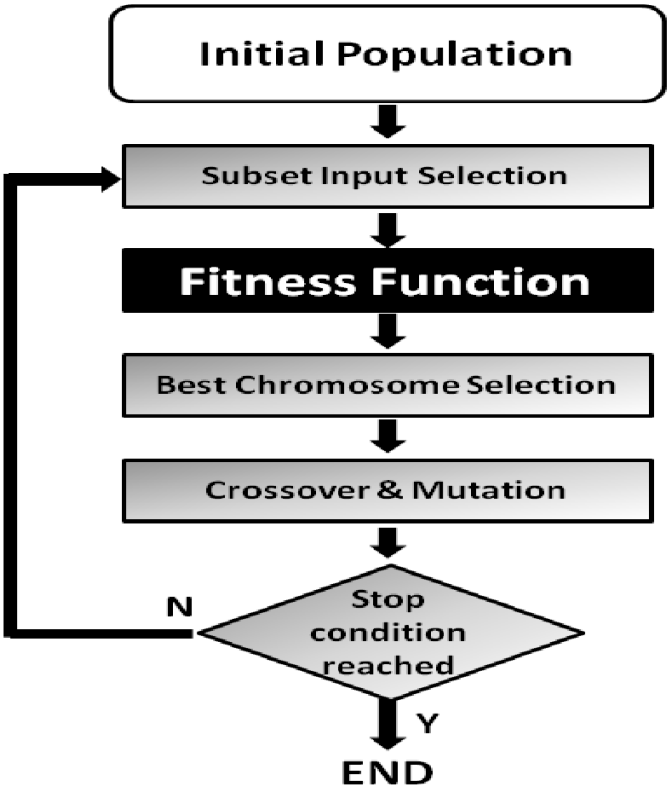
4. [**Accepting**] Place new offspring in a new population

5. [**Replace**] Use new generated population for a further run of algorithm

6. [**Test**] If the end condition is satisfied, stop, and return the best solution in current population

7. [**Loop**] Go to step 2

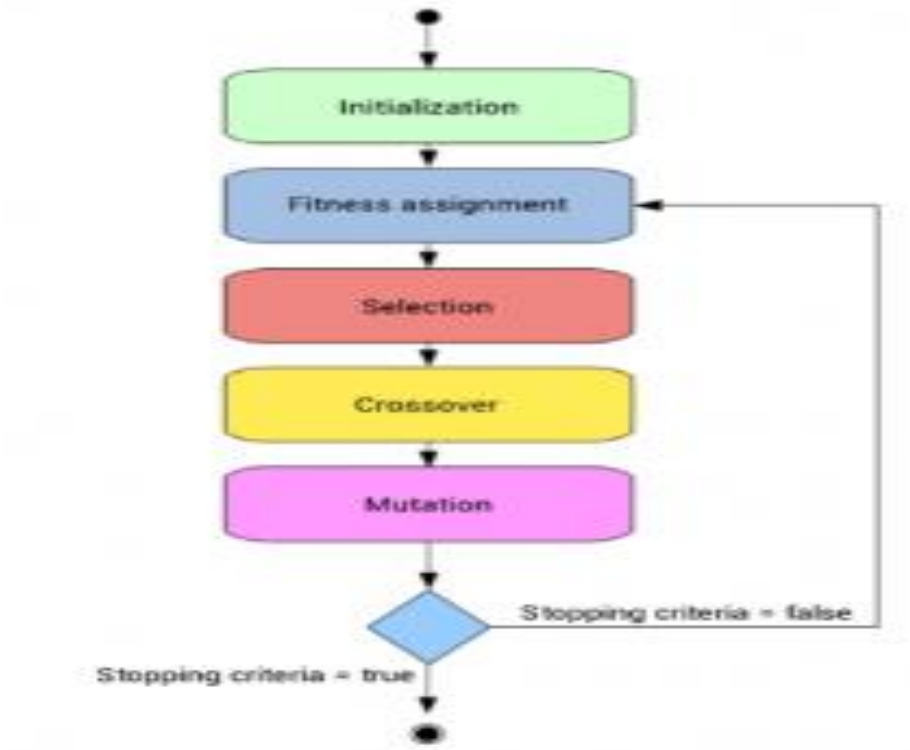
Flowchart



Algorithm:

- **Firstly, we defined our initial population as our countrymen.**
- **We defined a function to classify whether is a person is good or bad.**
- **Then we selected good people for mating to produce their off-springs.**
- **And finally, these off-springs replace the bad people from the population and this process repeats.**
- **This is how genetic algorithm actually works, which basically tries to mimic the human evolution to some extent.**
- **it is an optimization technique, which tries to find out such values of input so that we get the best output values or results.**

The working of a genetic algorithm is also derived from biology



Pseudo code for GA:

START

Generate the initial population

Compute fitness

REPEAT

Selection

Crossover

Mutation

Compute fitness

UNTIL population has converged

STOP

Conclusion:

Post Lab Questions:

1. Name some of the existing search methods.
2. What are the operators involved in a simple genetic algorithm?
3. What is reproduction?
4. What is crossover?
5. Write the code for GA and implement it using Python.

CODE:

```
import random

def fitness_function(individual):
    return sum(individual)

def generate_individual():
    return [random.randint(0, 100) for _ in range(10)]

def mutate(individual):
    index_to_mutate = random.randint(0, len(individual) - 1)
    individual[index_to_mutate] = random.randint(0, 100)
    return individual

def crossover(parent1, parent2):
    midpoint = len(parent1) // 2
    child1 = parent1[:midpoint] + parent2[midpoint:]
    child2 = parent2[:midpoint] + parent1[midpoint:]
    return child1, child2

def select_parents(population):
    fitness_values = [fitness_function(ind) for ind in population]
    total_fitness = sum(fitness_values)
    probabilities = [fit / total_fitness for fit in fitness_values]
    parents = random.choices(population, probabilities, k=2)
    return parents

def genetic_algorithm():
    population_size = 100
    population = [generate_individual() for _ in range(population_size)]

    for generation in range(100):
        parents = select_parents(population)
        offspring = []

        for i in range(population_size // 2):
            parent1, parent2 = parents
            child1, child2 = crossover(parent1, parent2)
            offspring.append(mutate(child1))
            offspring.append(mutate(child2))

        population = parents + offspring
        population.sort(key=fitness_function, reverse=True)
        population = population[:population_size]

    return population[0]
```

```
best_individual = genetic_algorithm()
```

```
print("Best Individual:", best_individual)
print("Fitness Value:", fitness_function(best_individual))
```

```
Best Individual: [25, 17, 38, 13, 51, 38, 95, 79, 99, 99]
Fitness Value: 554
```

Exp-7 Implementation of simple Genetic Algorithm

PAGE No.
DATE 24/10/23

* Post Lab Questions:

Q1) Name some of the existing search methods.

> Some search methods are -

1. Linear search
2. Binary search
3. Depth-First search (DFS)
4. Breadth-First search (BFS)
5. A* search
6. Greedy search
7. Hill Climbing
8. Genetic Algorithms
9. Simulated Annealing
10. Beam search
11. Best-First-search
12. Tabu search
13. Particle swarm optimization (PSO)
14. Ant colony optimization (ACO)
15. Dijkstra's Algorithm
16. Floyd-Warshall Algorithm.

Q2) What are the operators involved in a simple genetic algorithm?

-> The main operators involved in a simple genetic algorithm are -

① Initialization -
In first generation, a population of candidate solutions is randomly generated

② Selection -
Selection is the process of choosing individuals from current population to become parents for the next generation

③ Crossover -
Crossover involves taking two or more parent individuals and combining their genetic information to create one or more offspring.

④ Mutation -
A small random change applied to an individual's genetic information.

⑤ Evaluation -
Each individual's fitness is evaluated to determine how well it solves the problem.

Q3) what is reproduction?

-> Reproduction is a crucial genetic operator in genetic algorithms & evolutionary computation. It involves selecting indiviuals from a population based on their fitness, serving as parents for the next generation. These selected individuals undergo crossover to combine their genetic material, introducing diversity and potentially benefitials traits. In some cases, mutation is applied to farther diversify the offspring.

Q4) what is crossover?

-> In genetic Algorithms & evolutionary computation, "crossover" refers to a genetic operator that combines genetic information from two or more parent individuals to produce one or more offspring. This operation is applied during the reproduction phase, and it mimics the process of recombination or mating in biological genetics crossover is a key mechanism for introducing genetic diversity & potentially combining beneficial traits from the parents, contributing to the evolution of solution.

Q5) Write the code GA & implement it using Python.

->
```
import random
# Define target function to be maximized
def fitness_function(x):
    return x**2
# GA Parameters
population_size = 100
mutation_rate = 0.1
generation = 100
# Initialize the population with
  random individuals
def initialize_population(size):
    return [random.uniform(-10,10)
        for _ in range(size)]
# Select two individuals with a probability
  based on their fitness
def select_parents(population):
    return random.choice(population, k=2,
      weights = [fitness_function(x) for
      x in population])
# Perform single-point crossover
def crossover(parent1, parent2):
    crossover_point = random.randint
        (0, len(parent)-1)
    child1 = parent1[:crossover_point] +
            parent2[crossover_point:]
    child2 = parent2[:crossover_point] +
            parent1[crossover_point:]
    return child1, child2
```

```
# Apply mutation with a probability for
  each gene
def mutate (individual-rate):
    return [gene + random.uniform)
# Main GA loop
population = initialize-population(population-
    size)
for generation in range (generations):
    next_population = []

    for _ in range (population-size // 2):
        parent1, parent2 = select_parents (population)
        child1, child2 = crossover (parent1, parent2)
        child1 = mutate (child1, mutation_rate)
        child2 = mutate (child2, mutation-rate)
        next_population.extend ([child1, child2])

    population = next_population.
# Find the individual with the
  highest fitness
best_individual = max (population,
                key = fitness_function)
best_fitness = function (best_individual)

print (f" Best individual: {best_individual}")
print (f" Best fitness: {best_fitness}").
```

Final Year B. Tech (EE)

Semester: I **Subject: Artificial Intelligence and Machine Learning**

Name: Shreerang Mhatre **Class: TY**

Roll No: 52 **Batch: A3**

Experiment No: 08

Name of the Experiment: **Implement** FIS with Mamdani Interfacing Mechanism Functions

Performed on: 30/10/2023

Submitted on: 4/11/2023

Marks	Teacher's Signature with date

Aim: To implement FIS with mamdani inferencing mechanism using MATLAB/Python code.

Prerequisite: Knowledge of fuzzy sets, membership functions.

Objective:

To implement various fuzzy membership functions using Python code.

Components and Equipment required:

Python software, NumPy and Panda Libraries, MATLAB with Fuzzy toolbox

Theory:

The Mamdani fuzzy inference system is proposed as a first attempt to control a food quality and service by a set of linguistic control rules. To use fuzzy toolbox to model tips value that is given after a dinner based on quality of food (poor, average or good) and service (poor, average or good) and the tip value is decided accordingly in the range from 0 to 25 % of bill value.

As an illustration model we would consider an example in which two input linguistic variables Quality of food and Service provided are considered with poor, average or good membership functions.

Procedure:

INPUTS:

Quality**:** {Poor, Average, Good}

Service: {Poor, Average, Good}

OUTPUT:

Tips**:** Tip value ranging from 0-25 % of bill amount

Use Fuzzy Inference System (FIS) Editor and perform the following

1. Go to command window in Matlab and type fuzzy.
2. New Fuzzy Logic Designer window will be opened.
3. Give Input / Output Variable.
    a. Go to Edit Window and click Add variable
    b. As per our requirements create two input variables namely quality and service

       Quality**:** {Poor, Average, Good}

       Service: {Poor, Average, Good}
    c. Similarly, one output variable as tip value ranges from 0 to 25%.
4. The values for Quality and Service variables are selected for their respective ranges.

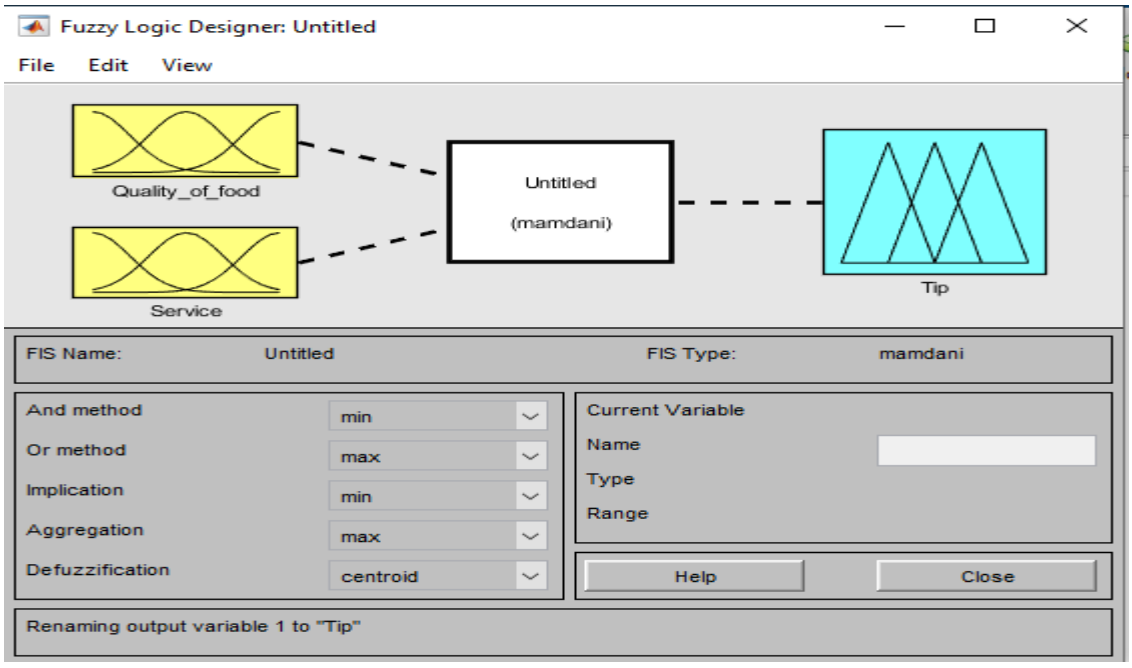
5. Quality:
    a. Double click the Quality input variable.
    b. New window will be opened and remove all the Membership Functions.
    c. Go to Edit and Click Add MFs and select the 4 Parameters for Quality table.

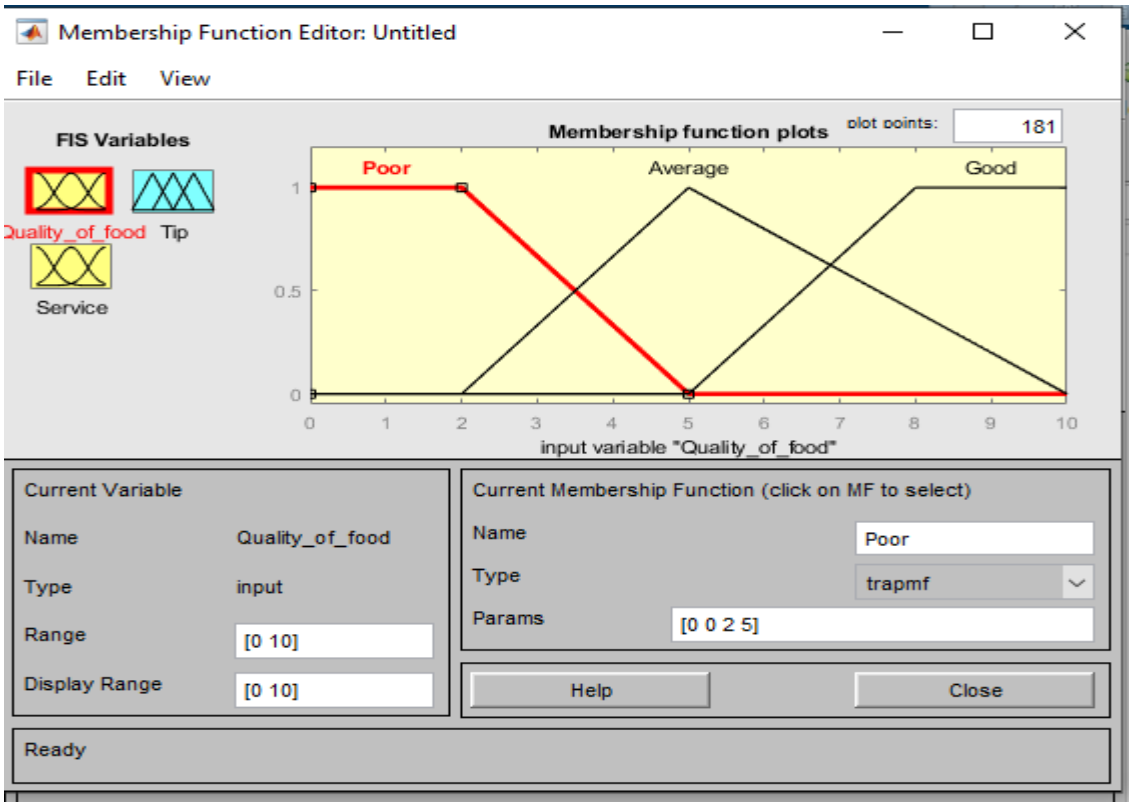
Change the following fields as per the table given below.

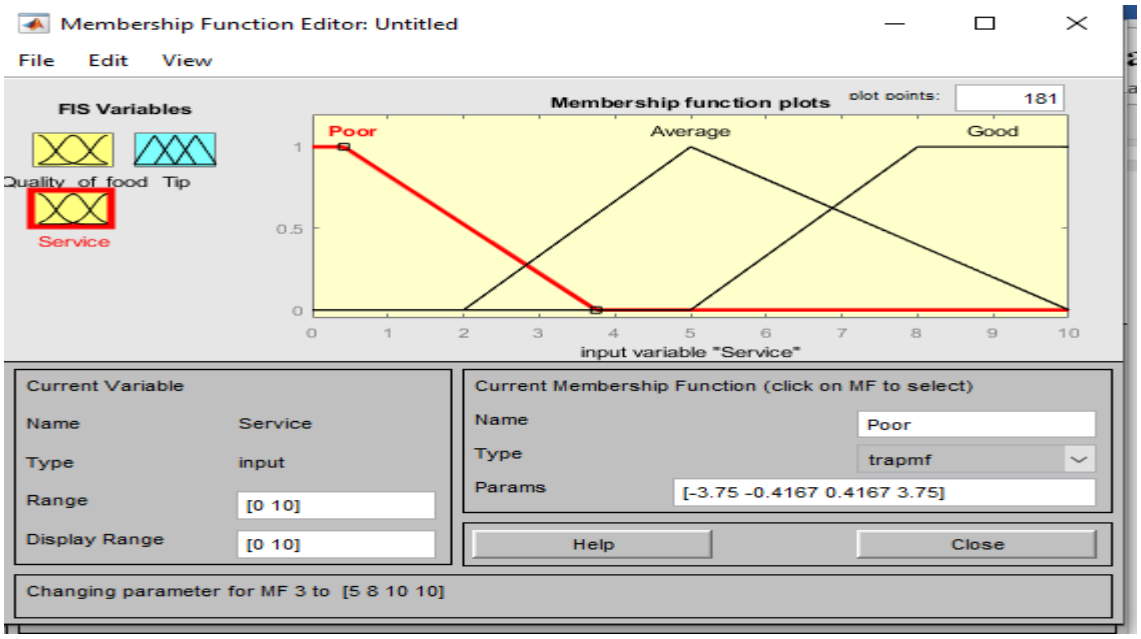
MF1: Range: [0 1 10] Name: Poor Type: trapmf Parameter [0 0 2 5]	MF2: Range: [0 1 10] Name: Average Type: trimf Parameter [2 5 10]	MF3: Range: [0 1 10] Name: Good Type: trapmf Parameter [5 7 10 10]

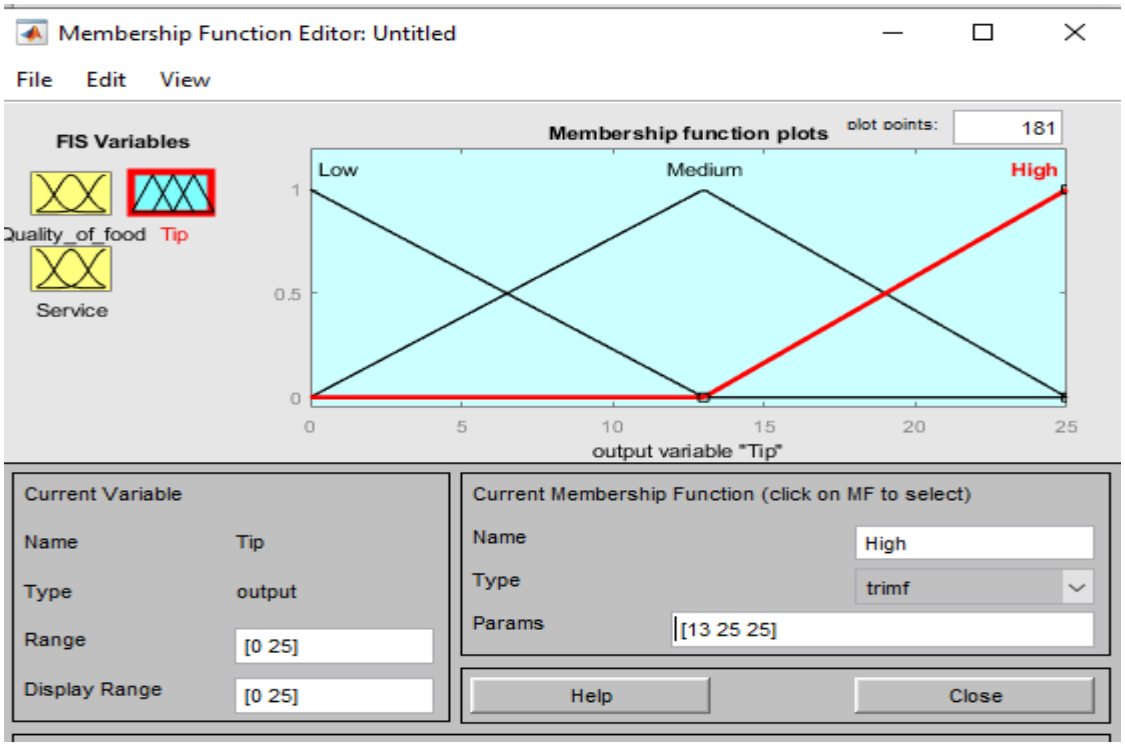
6. Similarly add the data to service and tips variables.
7. Go to Rules: Edit → Rules
8. Add the Rules
9. Go to view → Rules
10. Exit

Sample Input and Output:

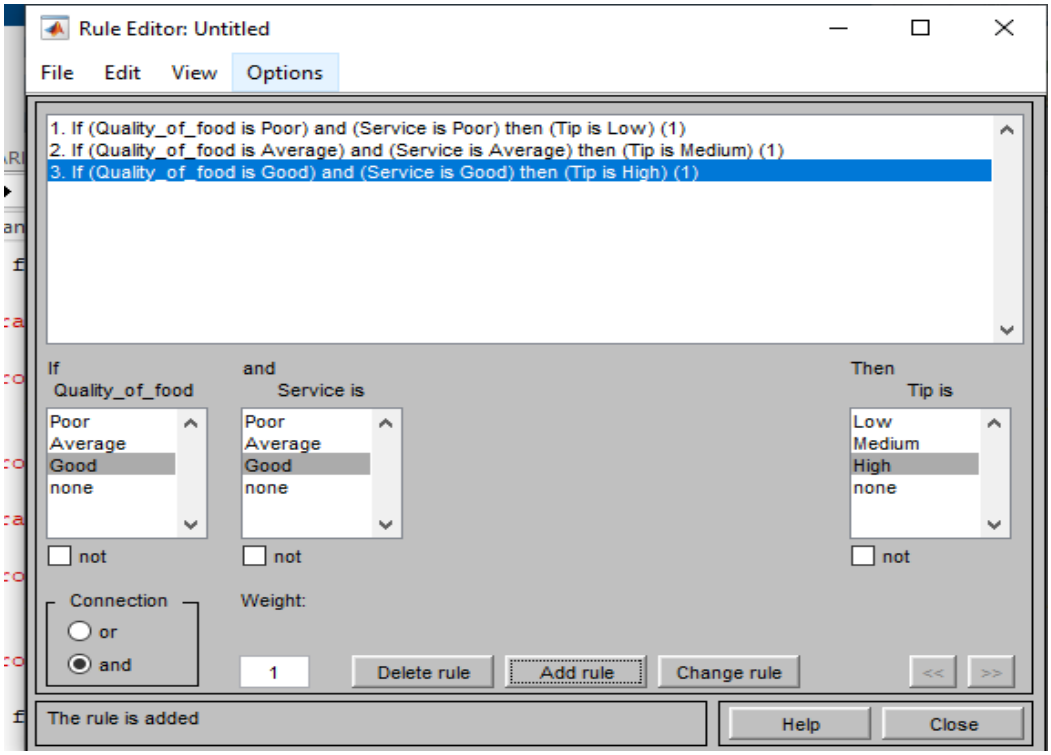



Membership Function Editor: Untitled
File Edit View
FIS Variables
Quality_of_food
Tip
Service
Membership function plots
plot points:
181
Poor
Average
Good
input variable "Quality_of_food"
Current Variable
Name
Quality_of_food
Type
input
Range
[0 10]
Display Range
[0 10]
Current Membership Function (click on MF to select)
Name
Poor
Type
trapmf
Params
[0 0 2 5]
Help
Close
Ready

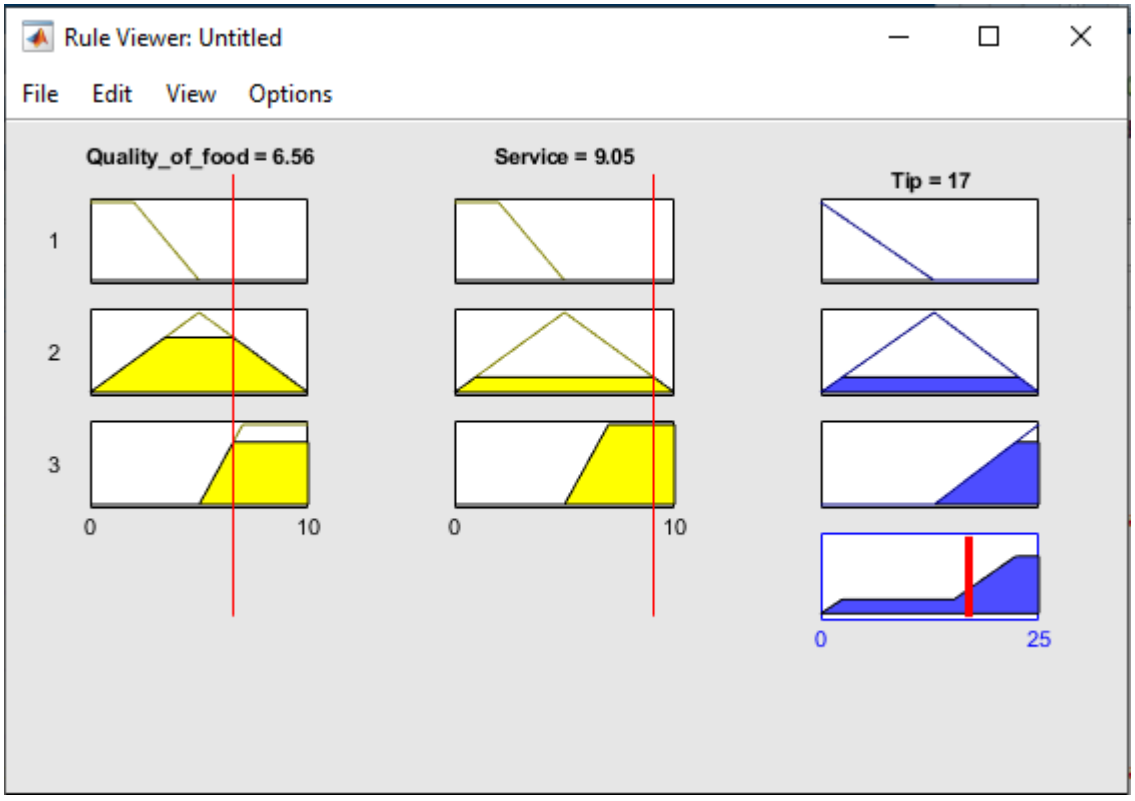

Membership Function Editor: Untitled
File Edit View
FIS Variables
Quality_of_food
Tip
Service
Membership function plots
plot points:
181
Poor
Average
Good
input variable "Service"
Current Variable
Name
Service
Type
input
Range
[0 10]
Display Range
[0 10]
Current Membership Function (click on MF to select)
Name
Poor
Type
trapmf
Params
[-3.75 -0.4167 0.4167 3.75]
Help
Close
Changing parameter for MF 3 to [5 8 10 10]

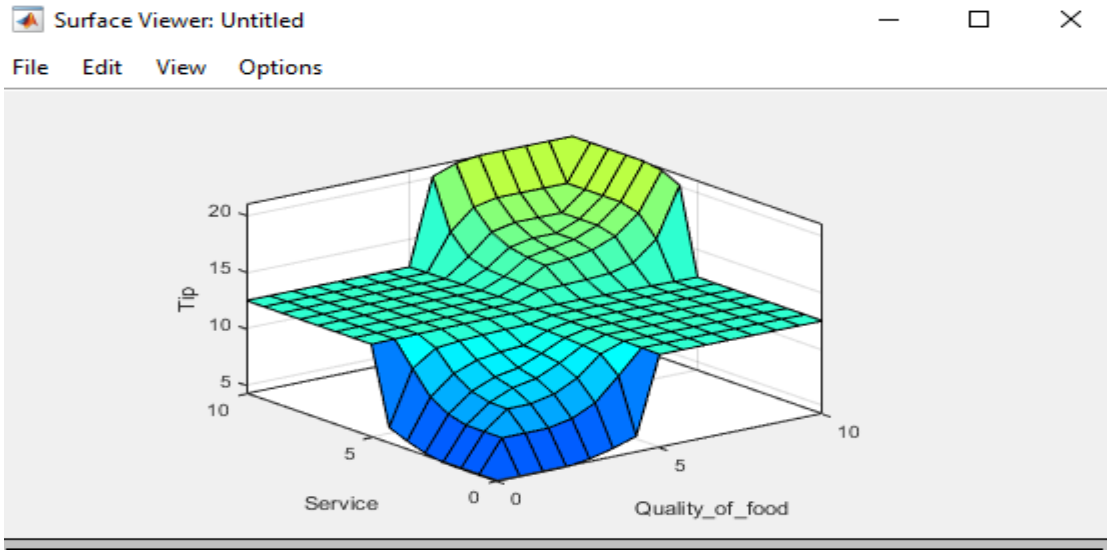


Created Rules:



Output Obtained:





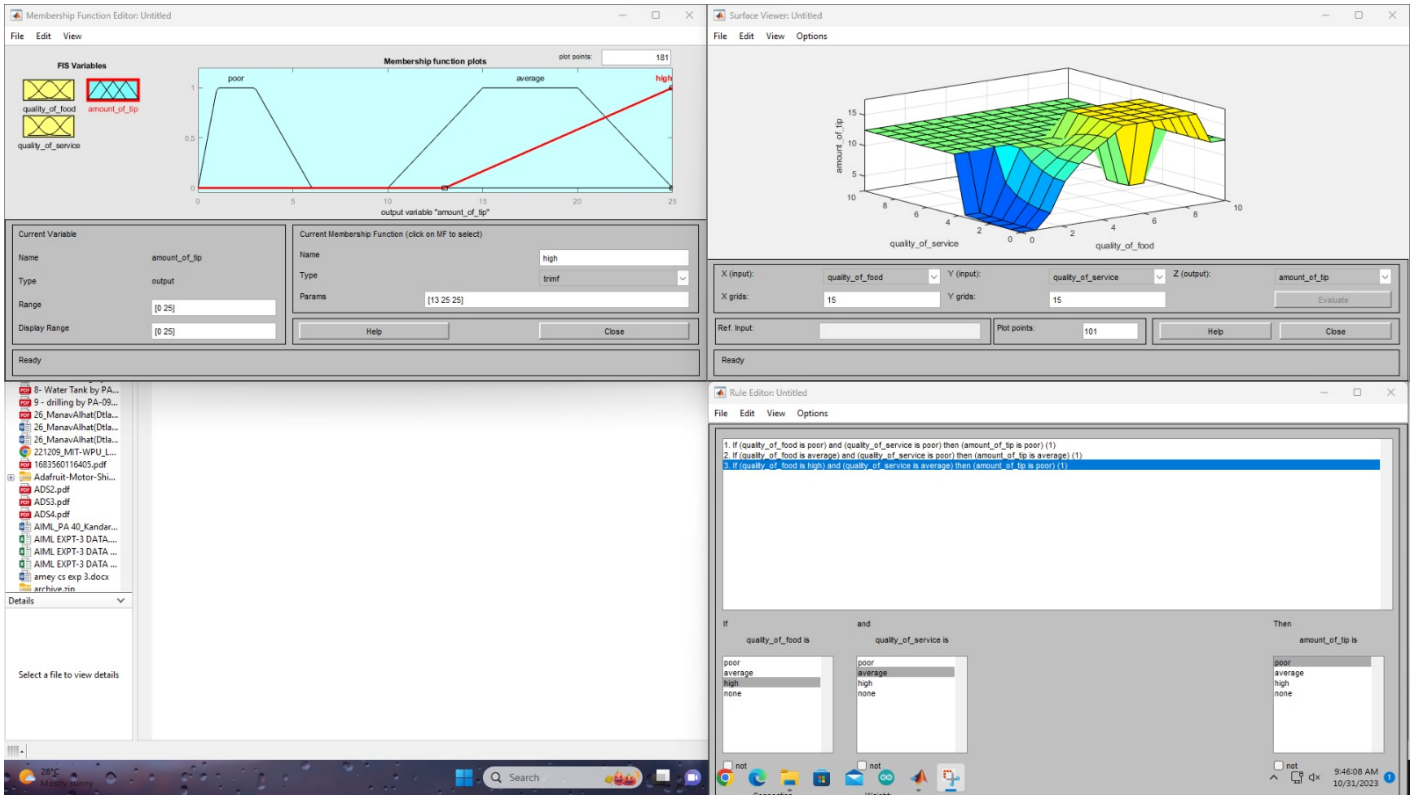
Python Code:

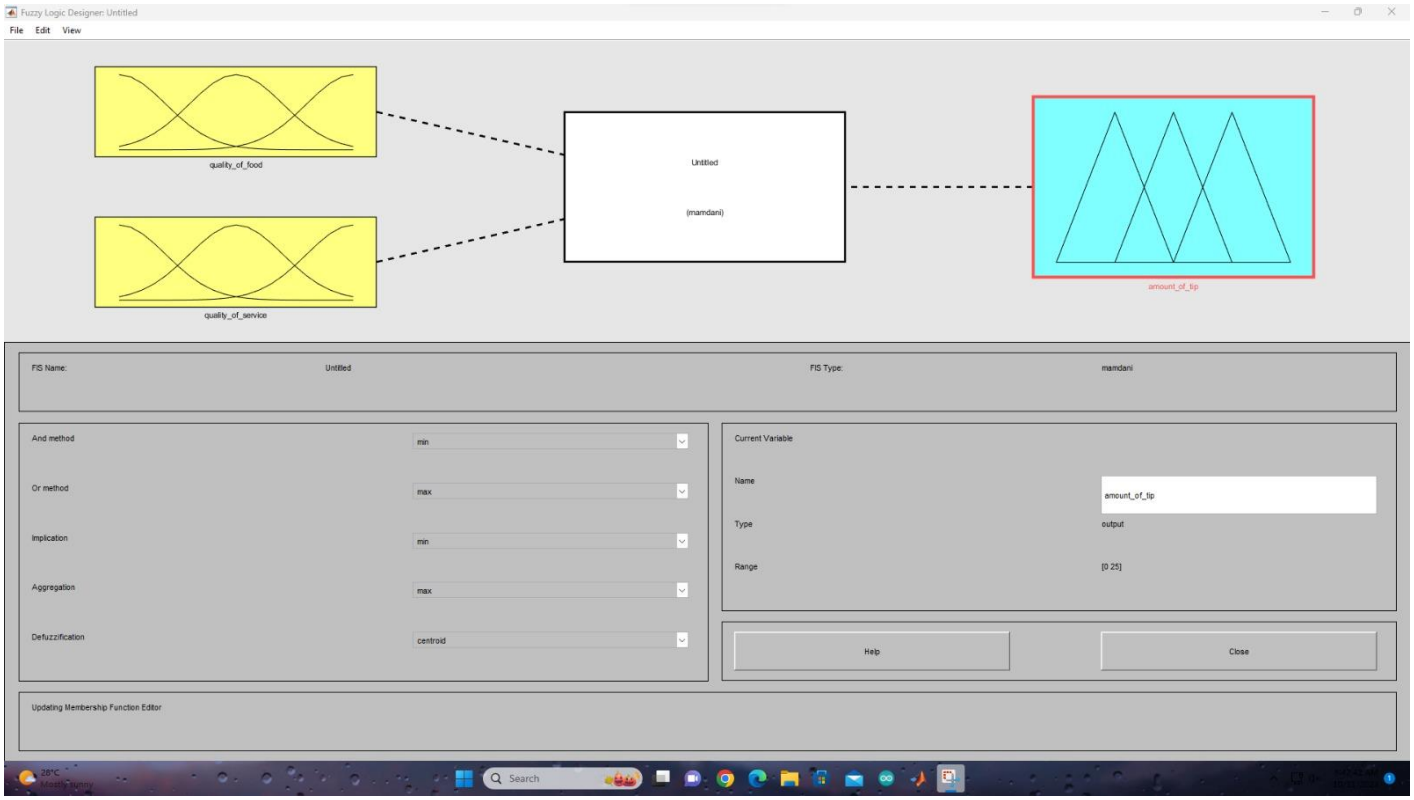
```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
quality = ctrl.Antecedent (np.arange(0, 11, 1), 'quality')
service = ctrl.Antecedent (np.arange(0, 11, 1), 'service')
tip = ctrl.Consequent (np.arange(0, 26, 1), 'tip'
service.automf()
quality.automf(3)
tip ['low'] = fuzz.trimf(tip.universe, [0, 0, 13])
tip ['medium'] = fuzz.trimf(tip.universe, [0, 13, 25])
tip ['high'] = fuzz.trimf(tip.universe, [13, 25, 25])
quality['average'].view()
service.view()
tip.view()
rule1 = ctrl.Rule(quality['poor'] & service['poor'], tip['low'])
rule1.view()
rule2 = ctrl.Rule(quality['average'] & service['average'], tip['medium'])
rule3 = ctrl.Rule(quality['good'] & service['good'], tip['high'])
tipping_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
tipping = ctrl.ControlSystemSimulation(tipping_ctrl)
tipping.input['quality'] = 8.5
tipping.input['service'] = 9.5
tipping.compute()
print(tipping.output['tip'])
tip.view(sim=tipping)
```

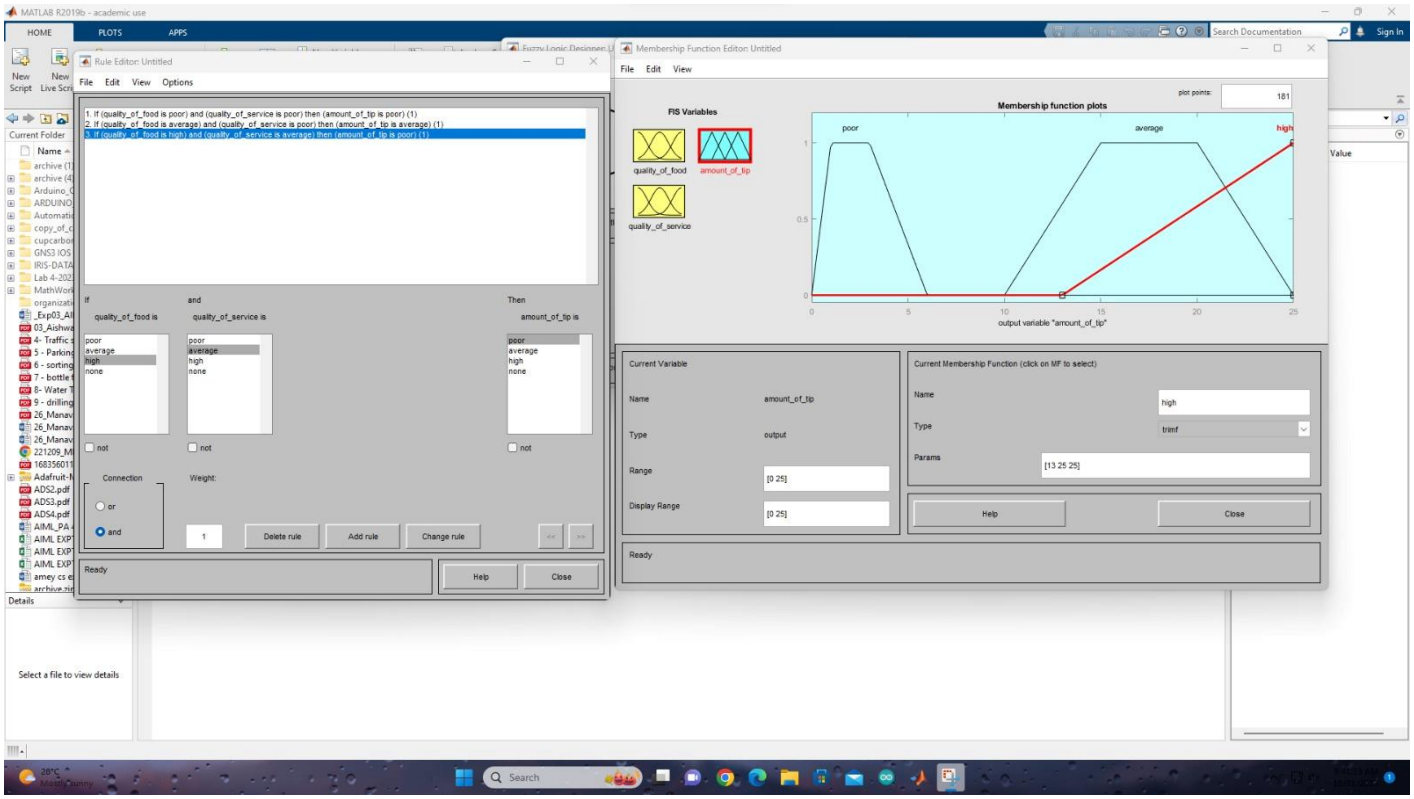
Conclusion:

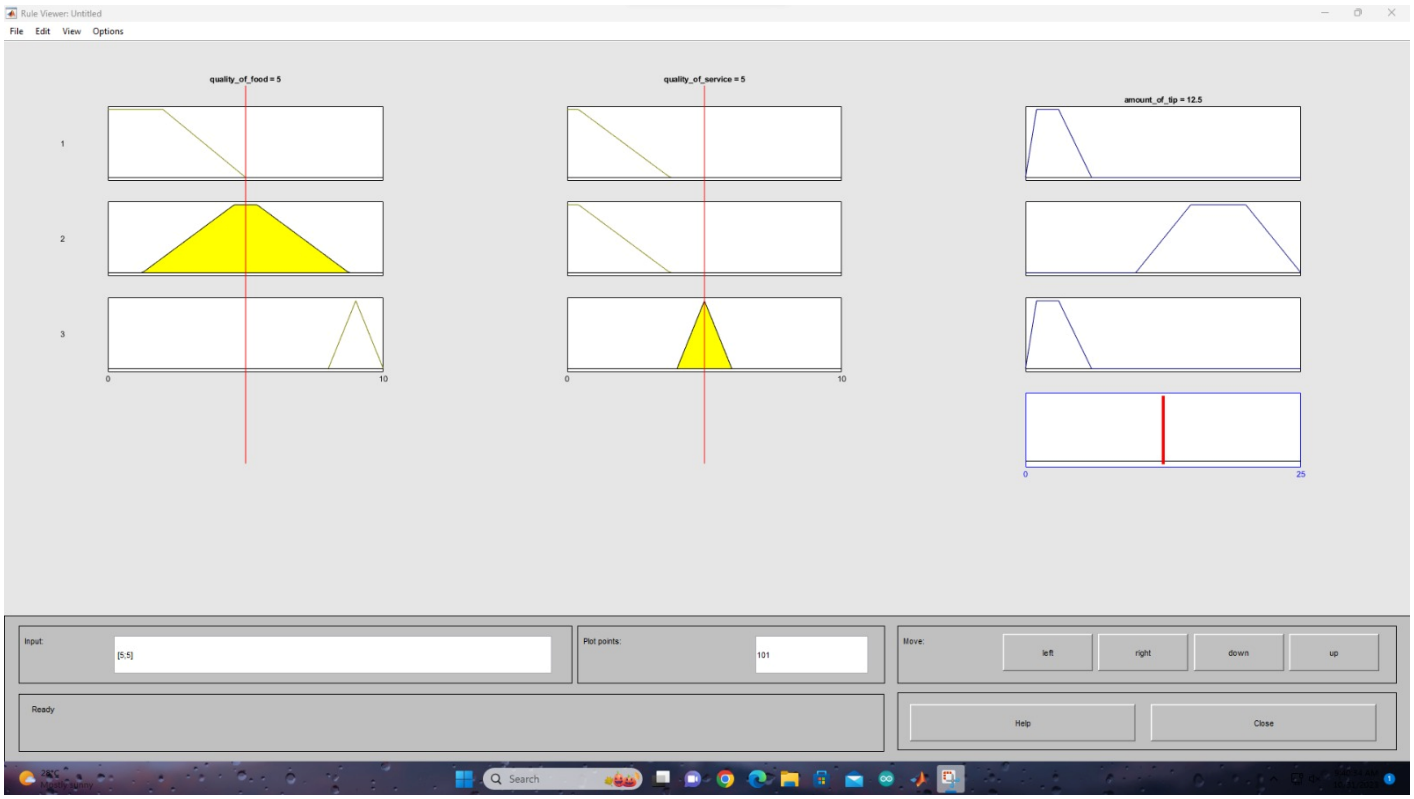
Post Lab Questions:

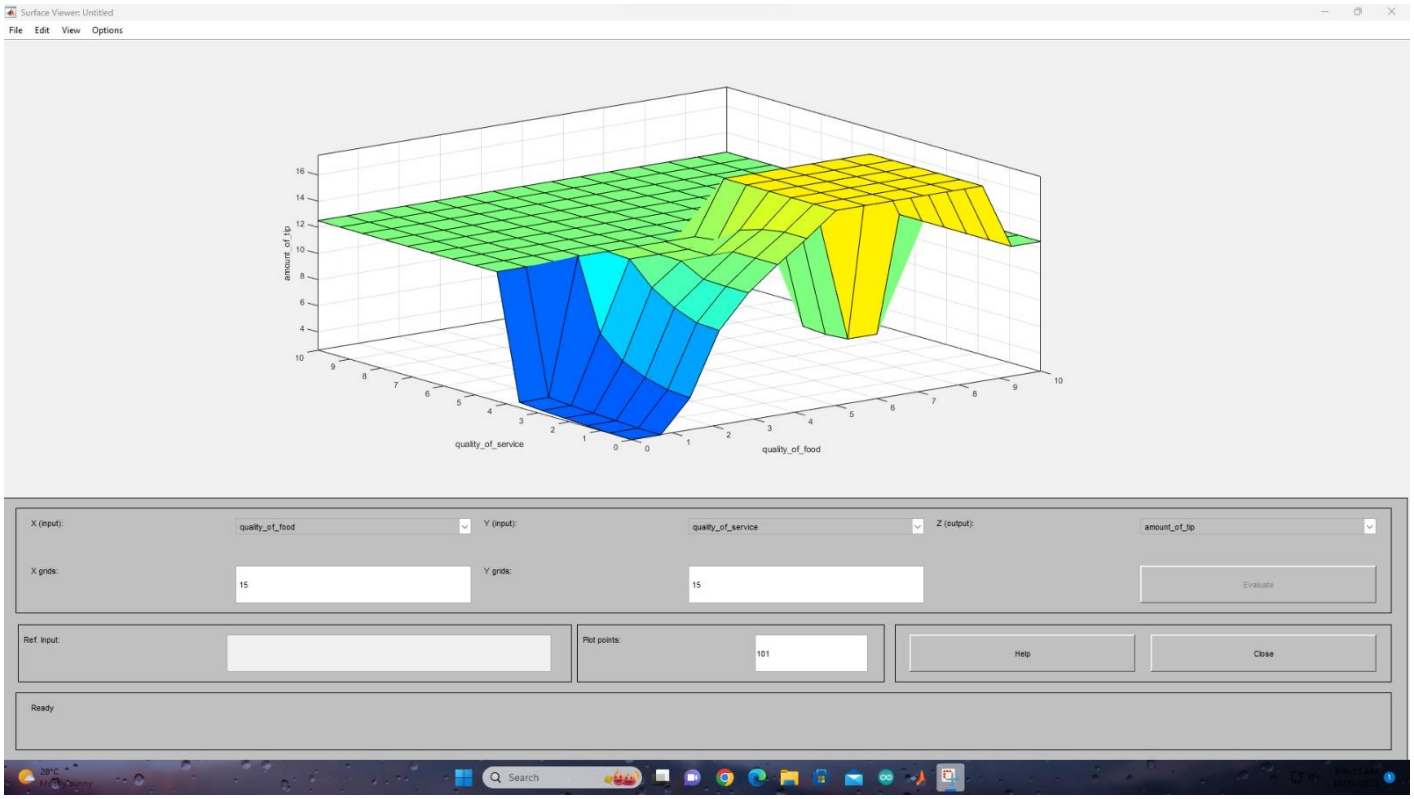
1. Define fuzzy set and crisp set.
2. What are the various operations on fuzzy set?
3. What are the set operations which are violated in fuzzy set theory?
4. Explain FIS in detail.


Membership Function Editor: Untitled
File Edit View
FIS Variables
quality_of_food
amount_of_tip
quality_of_service
Membership function plots
plot points: 181
poor
average
high
output variable "amount_of_tip"
Current Variable
Name amount_of_tip
Type output
Range [0 25]
Display Range [0 25]
Current Membership Function (click on MF to select)
Name high
Type trimf
Params [13.25 25]
Help
Close
Ready
Surface Viewer: Untitled
File Edit View Options
amount_of_tip
quality_of_service
quality_of_food
X (input): quality_of_food
Y (input): quality_of_service
Z (output): amount_of_tip
X grids: 15
Y grids: 15
Evaluate
Ref. Input:
Plot points: 101
Help
Close
Ready
Rule Editor: Untitled
File Edit View Options
1. If (quality_of_food is poor) and (quality_of_service is poor) then (amount_of_tip is poor) (1)
2. If (quality_of_food is average) and (quality_of_service is poor) then (amount_of_tip is average) (1)
3. If (quality_of_food is high) and (quality_of_service is average) then (amount_of_tip is poor) (1)
If
and
Then
quality_of_food is
quality_of_service is
amount_of_tip is
poor
average
high
none
not
Details
Select a file to view details
Search
9:46:08 AM
10/31/2023


Fuzzy Logic Designer: Untitled
File Edit View
quality_of_food
quality_of_service
Untitled
(mamdani)
amount_of_tip
FIS Name: Untitled
FIS Type: mamdani
And method min
Or method max
Implication min
Aggregation max
Defuzzification centroid
Current Variable
Name amount_of_tip
Type output
Range [0 25]
Help
Close
Updating Membership Function Editor
Search


MATLAB R2019b - academic use
HOME
PLOTS
APPS
Rule Editor: Untitled
File Edit View Options
1. If (quality_of_food is poor) and (quality_of_service is poor) then (amount_of_tip is poor) (1)
2. If (quality_of_food is average) and (quality_of_service is poor) then (amount_of_tip is average) (1)
3. If (quality_of_food is high) and (quality_of_service is average) then (amount_of_tip is poor) (1)
If
quality_of_food is
and
quality_of_service is
Then
amount_of_tip is
poor
average
high
none
not
Connection
or
and
Weight:
1
Delete rule
Add rule
Change rule
Ready
Help
Close
Membership Function Editor: Untitled
File Edit View
FIS Variables
quality_of_food
amount_of_tip
quality_of_service
plot points:
181
Membership function plots
poor
average
high
output variable "amount_of_tip"
Current Variable
Name
amount_of_tip
Type
output
Range
[0 25]
Display Range
[0 25]
Current Membership Function (click on MF to select)
Name
high
Type
trimf
Params
[13 25 25]
Help
Close
Ready
Details
Select a file to view details
Value


Rule Viewer: Untitled
File Edit View Options
quality_of_food = 5
quality_of_service = 5
amount_of_tip = 12.5
1
2
3
0
10
0
10
0
25
Input:
[5.5]
Plot points:
101
Move:
left
right
down
up
Ready
Help
Close


Surface Viewer: Untitled
File Edit View Options
amount_of_tip
quality_of_service
quality_of_food
X (input):
quality_of_food
Y (input):
quality_of_service
Z (output):
amount_of_tip
X grids:
15
Y grids:
15
Evaluate
Ref. Input:
Plot points:
101
Help
Close
Ready
Search

Exp 8

FIS with Mamdani Interfacing Mechanism Functions

PAGE No.
DATE 21/11/23

* Post Lab Questions:

① ~~A fuzzy set is a~~
② Define fuzzy set & crisp set

→① Fuzzy set -

A fuzzy set is a mathematical concept that generalizes the idea of a traditional, or "crisp", set by allowing elements to belong to the set to varying degrees.

In a fuzzy set, each element is associated with a membership function that assigns a value between 0 & 1, indicating the degree to which the element belongs to the set.

② Crisp set -

A crisp set, in contrast to a fuzzy set, is a traditional set in which elements are either members or non-members with no intermediate states.

Crisp sets are used in classical set theory and are suitable for modelling well-defined, discrete, and non-ambiguous concepts. They do not accomodate uncertainty or gradual membership.

PAGE No.
DATE / /

Q2) What are the various operations on fuzzy set?

→ Various operations on fuzzy set are.

① Membership Function -
The membership fonction of a fuzzy set defines the degree of membership of each element in the universal set.

② Union (Fuzzy Union) -
The union of two fuzzy sets A & B is a fuzzy set denoted by A∪B.

③ Intersection (Fuzzy Intersection) -
The intersection of two fuzzy set denoted by A∩B

④ Complement (Fuzzy complement) -
The complement of a fuzzy set A is the fuzzy set denoted by A'.

⑤ De Morgan's law -
Fuzzy sets obey De Morgan's laws which relate union and intersection operations for complements.

→

Q3) What are the set operations which are violated in fuzzy set theory?

-> Set operations -

① Commutative law of union -
In classical set theory, the union operation is commutative, meaning that $A \cup B$ is the same as $B \cup A$. But, in fuzzy set theory the commutative law of union is violated.

② Commutative law of intersection -
Similar to the union operation, the commutative law of intersection is violated in fuzzy set theory

③ Idepotent law -
The idepotent law states that $A \cup A$ and $A \cap A$ is equal to A

④ Distributive law -
The distributive law for classical set theory $(A \cup (B \cap C) = (A \cup B) \cap (A \cup C))$ is also violated in fuzzy set theory.

⑤ Absorption law -
The absorption law in classical set theory $(A \cup (A \cap B) = A$ & $A \cap (A \cup B) = A)$ is also violated in fuzzy set theory.

Q4) Explain FIS in detail

→ ① A Fuzzy Inference System (FIS) is a computational framework rooted in fuzzy logic, designed to handle uncertainty and vagueness in data & and decision-making.

② It comprises several key components fuzzification, where crisp data is transformed into fuzzy sets with linguistic terms; a rule base that defines the relationships between input and output fuzzy sets using conditional statements.

③ An inference engine that evaluates rules based on current input values and determines the strength of each rule's conclusion.

④ FIS finds applications in a wide range of fields, including control systems, decision support, pattern recognition, and human-machine interaction, where it excels at handling imprecise & vague information