Apex Institute Technology (AIT)

Department of Computer Science Engineering

B.E. – CSE (AIML)

SEMESTER	: 5th (July-Nov 2024)
SUBJECT NAME	: Soft Computing
SUBJECT CODE	: 22CSH-345
FACULTY	: Ms. Aarti

LAB MANUALS

Semester Scheme (As per CUIMS)

SN	21CSH-345	Soft Computing Lab	L	T	P	S	C	CH	Course Type
1			3	0	2	0	4	5	Elective
			Course Code(s)						
PRE-REQUISITE		Probability and Statistics Basics of Python and Machine learning/AI	22CSH-345						
CO-REQUISITE		NA							
ANTI-REQUISITE		NA							

List of Experiments

Experiment	Mapped	CO
Unit-I:- Installation and Basics		
1	Experiment 1.1. Getting started with python 3.x and installing libraries of Tensorflow, Keras, Pytorch.	CO1
2	Experiment 1.2 Build a fully connected layer architecture using in-built data sets.	CO2
3	Experiment 1.3 Creating an Artificial Neural Network Classa) Training the model b) Applying the Sigmoid function	CO2, CO4, CO5
Unit 2:-CNN Implementation		
4	Experiment 2.1 Write a program to build Convolutional Neural network.	CO2, CO4, CO5
5	Experiment 2.2 Write a program to implement Multi Layer Perceptron.	CO2, CO4, CO5
6	Experiment 2.3 Write a program to implement feed forward network.	CO2, CO4, CO5
Unit 3:-Advanced Algorithms		
7	Experiment 3.1 Write a program to implement common operations on Fuzzy Set.	CO2
8	Experiment 3.2 Write a program to implement Fuzzy Control System: Tipping Problem.	CO2, CO4, CO5
9	Experiment 3.3 Write a program to implement Fuzzy Inference System.	CO2, CO4, CO5
10	Experiment 3.4 Write a program to implement GANs using genetic algorithms.	CO2, CO4, CO5

Pre-requisites, Course Outcomes and Mapping of COs with POs/PSOs

MODE OF EVALUATION: The performance of students is evaluated as follows:

	Practical	
Components	Continuous Internal Assessment (CAE)	Semester End Examination (SEE)
Marks	60	40
Total Marks	100	

a. COURSE DESCRIPTION

Soft Computing is the use of approximate calculations to provide imprecise but usable solutions to complex computational problems. The approach enable solutions for problems that may be either unsolvable or just too time- consuming to solve with current hardware. Soft computing is sometimes referred to as computational intelligence. Soft computing provides an approach to problem-solving using means other than computers. With the human mind as a role model, soft computing is tolerant of partial truths, uncertainty, imprecision and approximation, unlike traditional computing models. The tolerance of soft computing allows researchers to approach some problems that traditional computing can't process.

b. COURSE OBJECTIVES

The Course aims to:

1. To introduce soft computing concepts and techniques of artificial neural networks, fuzzy sets, fuzzy logic and genetic algorithms.
2. To understand the various techniques from the application point of view.
3. To analyze various soft computing techniques and decide the technique to be used in a particular problem situation.
4. To implement soft computing based solutions for real-world problems.

CO vs PO/PSO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PS
CO1	2	1	1	3	3	1	NA	NA	NA	NA	NA	1	1	1
CO2	2	1	1	3	3	1						1	1	1
CO3	2	2	2	2	2	1	NA	NA	NA	NA	NA	2	2	2
CO4	2	2	2	2	2	1	NA	NA	NA	NA	NA	2	2	2
CO5	2	1	2	2	2	1	NA	NA	NA	NA	NA	2	1	1
Target	2	1.4	1.6	2.4	2.4	1	NA	NA	NA	NA	NA	1.6	1.4	1.4

c. COURSE OUTCOMES

CO1	Identify and describe soft computing techniques and their roles in building intelligent. Machines	**1**
CO2	Recognize the feasibility of applying a soft computing methodology for a particular problem.	**2,4**
CO3	Apply fuzzy logic and reasoning to handle uncertainty and solve engineering problems, genetic algorithms to combinatorial optimization problems and neural networks to pattern classification and regression problems.	**3**
CO4	Effectively use modern software tools to solve real problems using a soft computing approach.	**3**
CO5	Evaluate various soft computing approaches for a given problem.	**4**

***Last Column describes the mapping with BT. the course will go to at max level 4.**

<u>CO/PO/PSO Mapping</u>

Lab Manuals

EXPERIMENT – 1.1

Mapped Course Outcome

CO1: Identify and describe soft computing techniques and their roles in building intelligent. Machines

AIM: Getting started with the python 3.x and installing libraries of Tensorflow, Keras, Pytorch.

Theory

Anaconda is an open-source distribution for python and R. It is used for data science, machine learning, deep learning, etc. With the availability of more than 300 libraries for data science, it becomes fairly optimal for any programmer to work on anaconda for data science. Anaconda helps in simplified package management and deployment. Anaconda comes with a wide variety of tools to easily collect data from various sources using various machine learning and AI algorithms. It helps in getting an easily manageable environment setup which can deploy any project with the click of a single button.

Procedure:

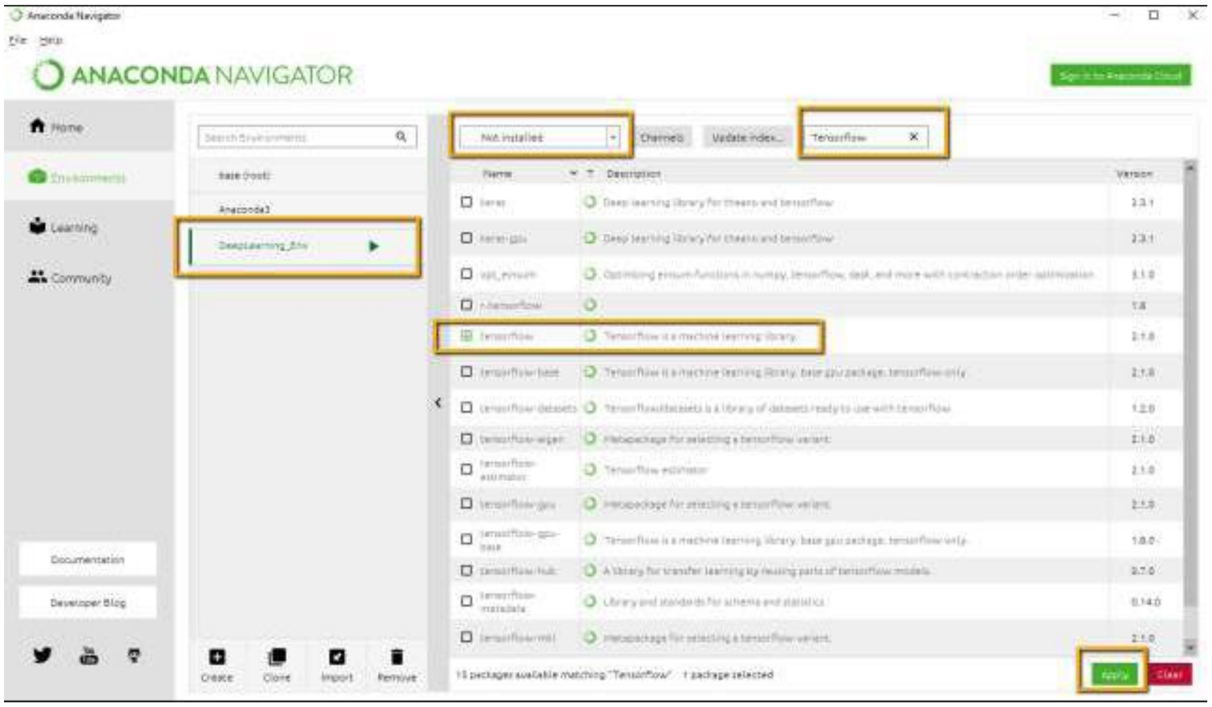
Installation of Anaconda

1. Visit https://www.anaconda.com/products/individual
2. Download the installer.
3. Launch the installer
4. Choose whether to add Anaconda to your PATH environment variable. We recommend not adding Anaconda to the PATH environment variable, since this can interfere with other software. Instead, use Anaconda software by opening Anaconda Navigator or the Anaconda Prompt from the Start Menu.

5. Choose whether to register Anaconda as your default Python. Unless you plan on installing and running multiple versions of Anaconda or multiple versions of Python, accept the default and leave this box checked.
6. Click the Install button. If you want to watch the packages Anaconda is installing, click Show Details.
7. Click the Next button.
8. After a successful installation you will see the "Thanks for installing Anaconda" dialog box:
9. If you wish to read more about Anaconda.org and how to get started with Anaconda, check the boxes "Anaconda Individual Edition Tutorial" and "Learn more about Anaconda". Click the Finish button.

Installing libraries of Tensorflow, Keras, Pytorch

Keras, TensorFlow and PyTorch are among the top three frameworks that are preferred by Data Scientists as well as beginners in the field of Deep Learning.This comparison on Keras vs TensorFlow vs PyTorch will provide you with a crisp knowledge about the top Deep Learning Frameworks and help you find out which one is suitable for you.
1. Open Anaconda Navigator.
2. Type TensorFlow in the search box, select the TensorFlow and apply to install the compatible TensorFlow package.
3. Now type in Keras and install Keras library to our environment.
4. Install other libraries as and when required.



Name	Description	Version
keras	Deep learning library for theano and tensorflow	2.3.1
keras-gpu	Deep learning library for theano and tensorflow	2.3.1
opt_einsum	Optimizing einsum functions in numpy, tensorflow, dask, and more with contraction order optimization	3.1.0
r-tensorflow		1.8
tensorflow	Tensorflow is a machine learning library	2.1.0
tensorflow-base	Tensorflow is a machine learning library, base gpu package, tensorflow only	2.1.0
tensorflow-datasets	tensorflowdatasets is a library of datasets ready to use with tensorflow	1.2.0
tensorflow-eigen	Metapackage for selecting a tensorflow variant.	2.1.0
tensorflow-estimator	Tensorflow estimator	2.1.0
tensorflow-gpu	Metapackage for selecting a tensorflow variant.	2.1.0
tensorflow-gpu-base	Tensorflow is a machine learning library, base gpu package, tensorflow only	1.8.0
tensorflow-hub	A library for transfer learning by reusing parts of tensorflow models	0.7.0
tensorflow-metadata	Library and standards for schema and statistics	0.14.0
tensorflow-mkl	Metapackage for selecting a tensorflow variant.	2.1.0

15 packages available matching "Tensorflow"   1 package selected

Video Tutorial

https://www.youtube.com/watch?v=RgO8BBNGB8w

Further Reading

Rolon-Mérette, D., Ross, M., Rolon-Mérette, T., & Church, K. (2016). Introduction to Anaconda and Python: Installation and setup. *Python for research in psychology*, *16*(5), S5-S11.

Prospective Viva Questions

1. What is Soft Computing?
2. How Soft computing is used in traffic management.
3. What are the advantages of using Jupyter Notebook?
4. What is the usage of Numpy?
5. What do we use Pandas?

EXPERIMENT – 1.2

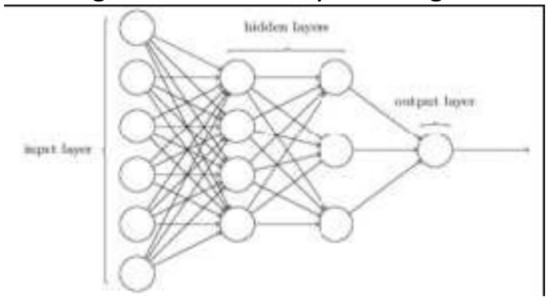
Mapped Course Outcome

CO2 Recognize the feasibility of applying a soft computing methodology for a particular problem.

AIM: Build fully connected layer architecture using in-built data sets.

Theory

A neural network also known as artificial neural network(ANN) is the basic building block of deep learning. It consists of layers of sigmoid neuron stacked together to form a bigger architecture.



Each circle in the above image is a sigmoid neuron. It consists of 3 types of layers, an input layer, an output layer, and hidden layers. All the previous layers are fully connected with the next layer as can be seen in the image so it is sometimes also referred to as the fully connected neural network. Each neuron has its own weight values. The first layer(input) takes the independent variable of data as input. Output layer predicts the class. The number of hidden layers and number of neurons in hidden layers is not fixed and you can choose any number and tinker to get the best results. Neural networks are just the weighted sum of the inputs. So the learning of neural networks is based on updating these weights. We need a method to update the weights.

It is based on how good the neural network is performing. Performance of the neural network means how good are the predictions based on the actual labels that are to be predicted. The value at the output layer is calculated by crossing through the neural network and finding the value of each neuron. Thisprocess of crossing through the neural network is called forward propagation.

Step by Step Working

1. We feed input data into the neural network.
2. The data flows from layer to layer until we have the output.
3. Once we have the output, we can calculate the error which is a scalar.
4. Finally we can adjust a given parameter (weight or bias) by subtracting the derivative of the error with respect to the parameter itself.
5. We iterate through that process.

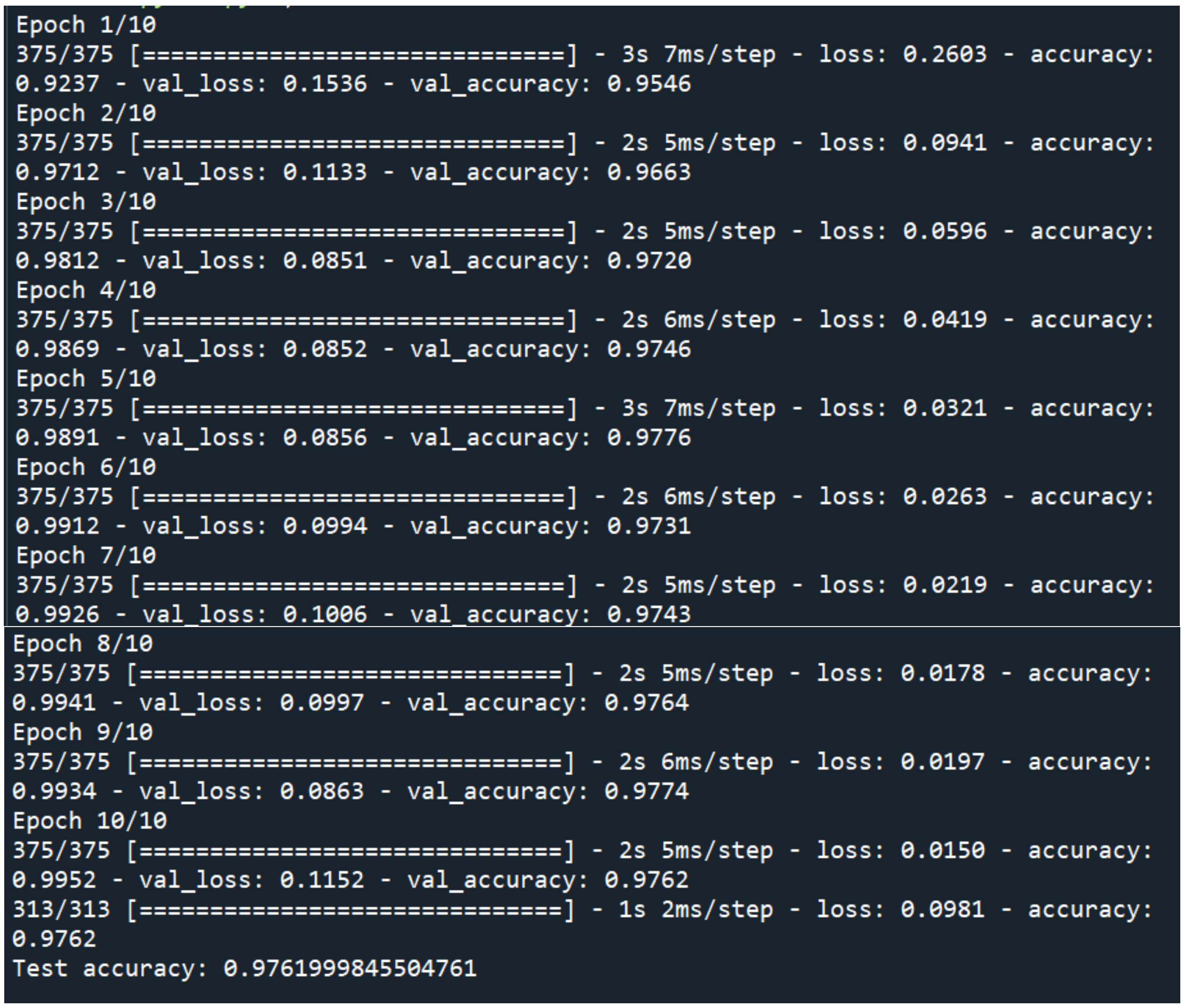
The most important step is the 4th. We want to be able to have as many layers as we want, and of any type. But if we modify/add/remove one layer from the network, the output of the network is going to change, which is going to change the error, which is going to change the derivative of the error with respect to the parameters. We need to be able to compute the derivatives regardless of the network architecture, regardless of the activation functions, regardless of the loss we use.

Procedure:

Python Code

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Step 2: Load and preprocess the dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Normalize the images
train_images = train_images.reshape((60000, 28 * 28)).astype('float32') / 255
test_images = test_images.reshape((10000, 28 * 28)).astype('float32') / 255

# Convert labels to one-hot encoding
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# Step 3: Define the model architecture
model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

# Step 4: Compile the model
model.compile(optimizer='adam',
          loss='categorical_crossentropy',
            metrics=['accuracy'])

# Step 5: Train the model
model.fit(train_images, train_labels, epochs=10, batch_size=128, validation_split=0.2)

# Step 6: Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc}')
```

Output Screen



```
Epoch 1/10
375/375 [==============================] - 3s 7ms/step - loss: 0.2603 - accuracy:
0.9237 - val_loss: 0.1536 - val_accuracy: 0.9546
Epoch 2/10
375/375 [==============================] - 2s 5ms/step - loss: 0.0941 - accuracy:
0.9712 - val_loss: 0.1133 - val_accuracy: 0.9663
Epoch 3/10
375/375 [==============================] - 2s 5ms/step - loss: 0.0596 - accuracy:
0.9812 - val_loss: 0.0851 - val_accuracy: 0.9720
Epoch 4/10
375/375 [==============================] - 2s 6ms/step - loss: 0.0419 - accuracy:
0.9869 - val_loss: 0.0852 - val_accuracy: 0.9746
Epoch 5/10
375/375 [==============================] - 3s 7ms/step - loss: 0.0321 - accuracy:
0.9891 - val_loss: 0.0856 - val_accuracy: 0.9776
Epoch 6/10
375/375 [==============================] - 2s 6ms/step - loss: 0.0263 - accuracy:
0.9912 - val_loss: 0.0994 - val_accuracy: 0.9731
Epoch 7/10
375/375 [==============================] - 2s 5ms/step - loss: 0.0219 - accuracy:
0.9926 - val_loss: 0.1006 - val_accuracy: 0.9743
Epoch 8/10
375/375 [==============================] - 2s 5ms/step - loss: 0.0178 - accuracy:
0.9941 - val_loss: 0.0997 - val_accuracy: 0.9764
Epoch 9/10
375/375 [==============================] - 2s 6ms/step - loss: 0.0197 - accuracy:
0.9934 - val_loss: 0.0863 - val_accuracy: 0.9774
Epoch 10/10
375/375 [==============================] - 2s 5ms/step - loss: 0.0150 - accuracy:
0.9952 - val_loss: 0.1152 - val_accuracy: 0.9762
313/313 [==============================] - 1s 2ms/step - loss: 0.0981 - accuracy:
0.9762
Test accuracy: 0.9761999845504761
```

Video Tutorial
https://www.youtube.com/watch?v=lGLto9Xd7bU

Further Reading

Journal Papers
https://www.sciencedirect.com/science/article/pii/S1877050916325467

Main text books:
- "Neural Networks: A Comprehensive Foundation", S. Haykin
- "Pattern Recognition with Neural Networks", C. Bishop
- "NeuralNetwork Design" by Hagan, Demuth and Beale

Books emphasizing the practical aspects:
- "Neural Smithing", Reeds and Marks
- "Practical Neural Network Recipees in C++"' T. Masters
- "Parallel Distributed Processing" Rumelhart and McClelland et al.

Deep Learning books and tutorials:
- http://www.deeplearningbook.org/
- Introduction to Learning Rules in Neural Network - DataFlair (data-flair.training)

Prospective Viva Questions
1. How neural networks can be used to solve real-time problems. Give any real-time application where NN are used.
2. What are different types of Neural Networks?
3. How ANN is trained?
4. What are the activation functions?

EXPERIMENT – 1.3

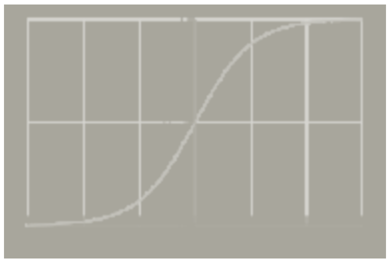
Mapped Course Outcome

CO2 Recognize the feasibility of applying a soft computing methodology for a particular problem.
CO4 Effectively use modern software tools to solve real problems using a soft computing approach.
CO5 Examine and evaluate various soft computing approaches for a given problem.

AIM: Creating an Artificial Neural Network Class.
a) Training the model
b) Applying the Sigmoid function Theory

We'll create a NeuralNetwork class in Python to train the neuron to give an accurate prediction. The class will also have other helper functions.Even though we'll not use a neural network library for this simple neural network example, we'll import the numpy library to assist with the calculations.
The library comes with the following four important methods:
exp—for generating the natural exponential
array—for generating a matrix
dot—for multiplying matrices
random —for generating random numbers. Note that we'll seed the random numbers to ensure their efficient distribution.

We'll use the Sigmoid function, which draws a characteristic "S"-shaped curve, as an activation function to the neural network.



This function can map any value to a value from 0 to 1. It will assist us to normalize the weighted sum of the inputs. Thereafter, we'll create the derivative of the sigmoid function to help in computing the essential adjustments to the weights. The output of a sigmoid function can be employed to generate its derivative. For example, if the output variable is "x", then its derivative will be x * (1-x).

Training of Model

1. We took the inputs from the training dataset, performed some adjustments based on their weights, and siphoned them via a method that computed the output of the ANN.
2. We computed the back-propagated error rate. In this case, it is the difference between neuron's predicted output and the expected output of the training dataset.
3. Based on the extent of the error got, we performed some minor weight adjustments using the Error Weighted Derivative formula.
4. We iterated this process an arbitrary number of 15,000 times. In every iteration, the whole training set is processed simultaneously.

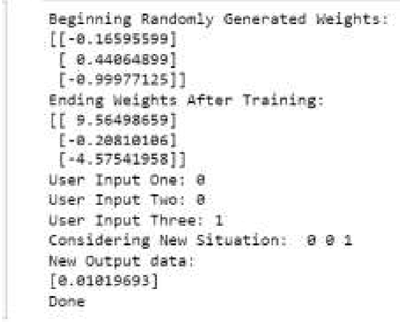
Procedure:

Python Code

```
import numpy as np
class NeuralNetwork():
def __init__(self):
# seeding for random number generation
np.random.seed(1)
#converting weights to a 3 by 1 matrix with values from -1 to 1 and mean of 0
self.synaptic_weights = 2 * np.random.random((3, 1)) - 1
def sigmoid(self, x):
#applying the sigmoid function
return 1 / (1 + np.exp(-x))
def sigmoid_derivative(self, x):
#computing derivative to the Sigmoid function
return x * (1 - x)
def train(self, training_inputs, training_outputs, training_iterations):
#training the model to make accurate predictions while adjusting weights continually
for iteration in range(training_iterations):
#siphon the training data via the neuron
output = self.think(training_inputs)
#computing error rate for back-propagation
error = training_outputs - output
#performing weight adjustments
```

```
adjustments = np.dot(training_inputs.T, error * self.sigmoid_derivative(output))
self.synaptic_weights += adjustments
def think(self, inputs):
#passing the inputs via the neuron to get output
#converting values to floats
inputs = inputs.astype(float)
output = self.sigmoid(np.dot(inputs, self.synaptic_weights))
return output
if __name__ == "__main__":
#initializing the neuron class
neural_network = NeuralNetwork()
print("Beginning Randomly Generated Weights: ")
print(neural_network.synaptic_weights)
#training data consisting of 4 examples--3 input values and 1 output
training_inputs = np.array([[0,0,1],
[1,1,1],
[1,0,1],
[0,1,1]])
training_outputs = np.array([[0,1,1,0]]).T
#training taking place
neural_network.train(training_inputs, training_outputs, 15000)
print("Ending Weights After Training: ")
print(neural_network.synaptic_weights)
user_input_one = str(input("User Input One: "))
user_input_two = str(input("User Input Two: "))
user_input_three = str(input("User Input Three: "))
print("Considering New Situation: ", user_input_one, user_input_two, user_input_three)
print("New Output data: ")
print(neural_network.think(np.array([user_input_one, user_input_two,
user_input_three])))
print("Done")
```

Output Screen



```
Beginning Randomly Generated Weights:
[[-0.16595599]
 [ 0.44064899]
 [-0.99977125]]
Ending Weights After Training:
[[ 9.56498659]
 [-0.20810106]
 [-4.57541958]]
User Input One: 0
User Input Two: 0
User Input Three: 1
Considering New Situation:  0 0 1
New Output data:
[0.01019693]
Done
```

Video Tutorial

https://www.youtube.com/watch?v=3t9lZM7SS7k

Journal Papers

https://www.sciencedirect.com/science/article/pii/S1877050916325467

Books:

- “Neural Networks: A Comprehensive Foundation”, S. Haykin
- “Pattern Recognition with Neural Networks”, C. Bishop

“NeuralNetwork Design” by Hagan, Demuth and Beale

Books emphasizing the practical aspects:

- “Neural Smithing”, Reeds and Marks
- “Practical Neural Network Recipees in C++”’ T. Masters
- “Parallel Distributed Processing” Rumelhart and McClelland et al.

Deep Learning books and tutorials:

- http://www.deeplearningbook.org/
- Introductionto Learning Rules in Neural Network - DataFlair (data-flair.training)

Prospective Viva Questions

1. What is the best activation function that is used in most of the real-time problems?
2. Which activation function can give output in the range between 0 -1.
3. How do we initialize the neural networks.
4. What are hyperparameters?

EXPERIMENT – 2.1

Mapped Course Outcome

CO2 Recognize the feasibility of applying a soft computing methodology for a particular problem.

CO4 Effectively use modern software tools to solve real problems using a soft computing approach.

CO5 Examine and evaluate various soft computing approaches for a given problem.

AIM: Write a program to build a Convolution Neural network.

Theory

A convolution neural network is similar to a multi-layer perceptron network. The major differences are what the network learns, how they are structured and what purpose they are mostly used for. Convolutional neural networks were also inspired from biological processes, their structure has a semblance of the visual cortex present in an animal. CNNs are largely applied in the domain of computer vision and has been highly successful in achieving state of the art performance on various test cases.

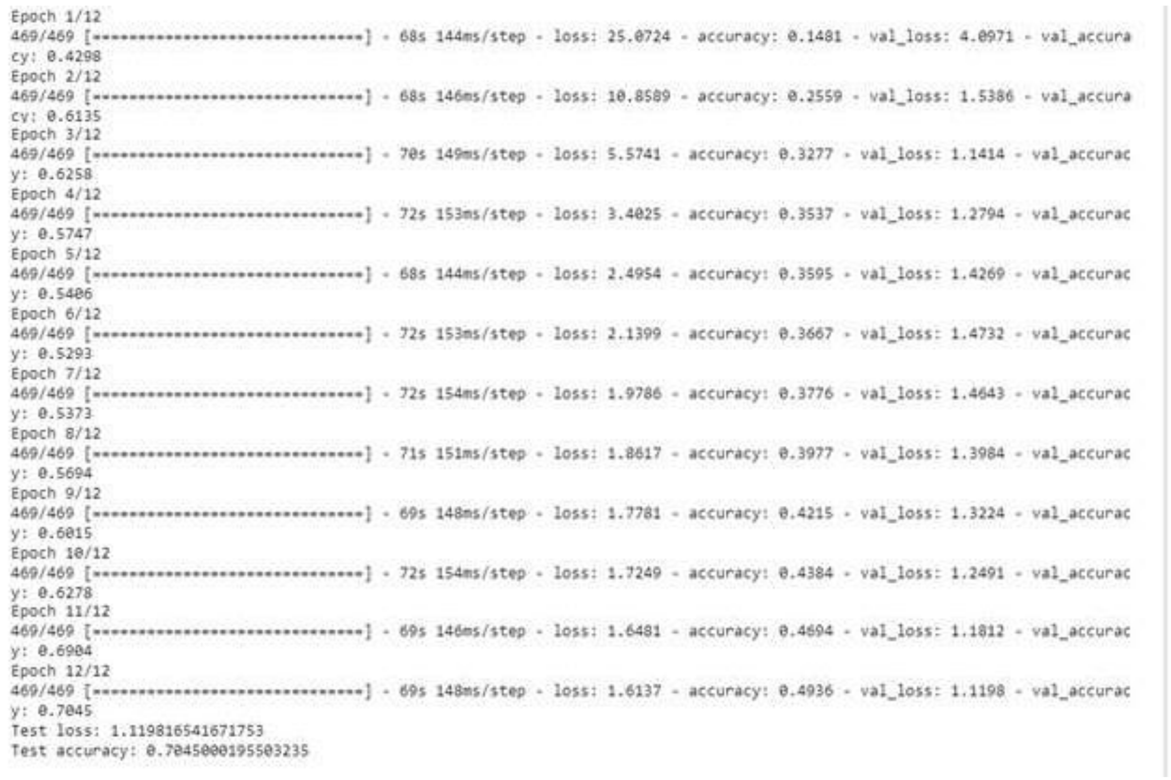
Procedure:

Python Code

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
```

```
import numpy as np
batch_size = 128
num_classes = 10
epochs = 12
# input image dimensions
img_rows, img_cols = 28, 28
# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(60000,28,28,1)
x_test = x_test.reshape(10000,28,28,1)
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
activation='relu',
input_shape=(28,28,1)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.Adadelta(),
metrics=['accuracy'])
model.fit(x_train, y_train,
batch_size=batch_size,
epochs=epochs,
verbose=1,
validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Output Screen



```
Epoch 1/12
469/469 [==============================] - 68s 144ms/step - loss: 25.0724 - accuracy: 0.1481 - val_loss: 4.0971 - val_accura
cy: 0.4298
Epoch 2/12
469/469 [==============================] - 68s 146ms/step - loss: 10.8589 - accuracy: 0.2559 - val_loss: 1.5386 - val_accura
cy: 0.6135
Epoch 3/12
469/469 [==============================] - 70s 149ms/step - loss: 5.5741 - accuracy: 0.3277 - val_loss: 1.1414 - val_accurac
y: 0.6258
Epoch 4/12
469/469 [==============================] - 72s 153ms/step - loss: 3.4025 - accuracy: 0.3537 - val_loss: 1.2794 - val_accurac
y: 0.5747
Epoch 5/12
469/469 [==============================] - 68s 144ms/step - loss: 2.4954 - accuracy: 0.3595 - val_loss: 1.4269 - val_accurac
y: 0.5406
Epoch 6/12
469/469 [==============================] - 72s 153ms/step - loss: 2.1399 - accuracy: 0.3667 - val_loss: 1.4732 - val_accurac
y: 0.5293
Epoch 7/12
469/469 [==============================] - 72s 154ms/step - loss: 1.9786 - accuracy: 0.3776 - val_loss: 1.4643 - val_accurac
y: 0.5373
Epoch 8/12
469/469 [==============================] - 71s 151ms/step - loss: 1.8617 - accuracy: 0.3977 - val_loss: 1.3984 - val_accurac
y: 0.5694
Epoch 9/12
469/469 [==============================] - 69s 148ms/step - loss: 1.7781 - accuracy: 0.4215 - val_loss: 1.3224 - val_accurac
y: 0.6015
Epoch 10/12
469/469 [==============================] - 72s 154ms/step - loss: 1.7249 - accuracy: 0.4384 - val_loss: 1.2491 - val_accurac
y: 0.6278
Epoch 11/12
469/469 [==============================] - 69s 146ms/step - loss: 1.6481 - accuracy: 0.4694 - val_loss: 1.1812 - val_accurac
y: 0.6904
Epoch 12/12
469/469 [==============================] - 69s 148ms/step - loss: 1.6137 - accuracy: 0.4936 - val_loss: 1.1198 - val_accurac
y: 0.7045
Test loss: 1.119816541671753
Test accuracy: 0.7045000195503235
```

References

1. https://towardsdatascience.com/build-your-own-convolution-neural-network-in- **5-mins-4217c2cf964f**

Further Reading

Journal Papers

https://www.sciencedirect.com/science/article/pii/S1877050916325467

Main text books:

- “Neural Networks: A Comprehensive Foundation”, S. Haykin
- “Pattern Recognition with Neural Networks”, C. Bishop
- “NeuralNetwork Design” by Hagan, Demuth and Beale

Books emphasizing the practical aspects:

- “Neural Smithing”, Reeds and Marks
- “Practical Neural Network Recipees in C++”’ T. Masters
- “Parallel Distributed Processing” Rumelhart and McClelland et al.

Deep Learning books and tutorials:

- http://www.deeplearningbook.org/
- Introductionto Learning Rules in Neural Network - DataFlair (data-flair.training)

Prospective Viva Questions

1. What is convolutional NN?
2. How convolutional NN are different from multi layer perceptron?

3. What are the real-time applications of Convolutional NN?
4. What is the difference between CNN and NN?
5. Why we use convolutional neural network?
6. What is convolution neural network CNN explain with examples?
7. Why Convolutional Neural Network is better?
8. Is CNN supervised or unsupervised?

EXPERIMENT 2.2

Mapped Course Outcome

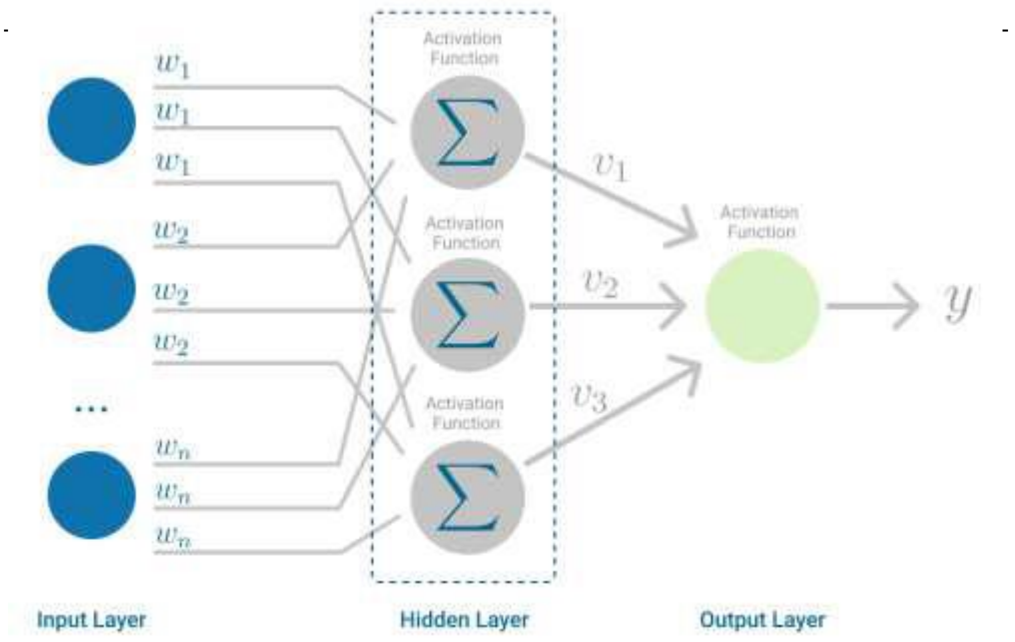
CO2 Recognize the feasibility of applying a soft computing methodology for a particular problem.
CO4 Effectively use modern software tools to solve real problems using a soft computing approach.
CO5 Examine and evaluate various soft computing approaches for a given problem.

AIM: *Write a program to implement Multi Layer Perceptron.*

Theory

Multilayer Perceptron

The Multilayer Perceptron was developed to tackle this limitation. It is a neural network where the mapping between inputs and output is non-linear.
A Multilayer Perceptron has input and output layers, and one or more hidden layers with many neurons stacked together. And while in the Perceptron the neuron must have an activation function that imposes a threshold, like ReLU or sigmoid, neurons in a Multilayer Perceptron can use any arbitrary activation function.

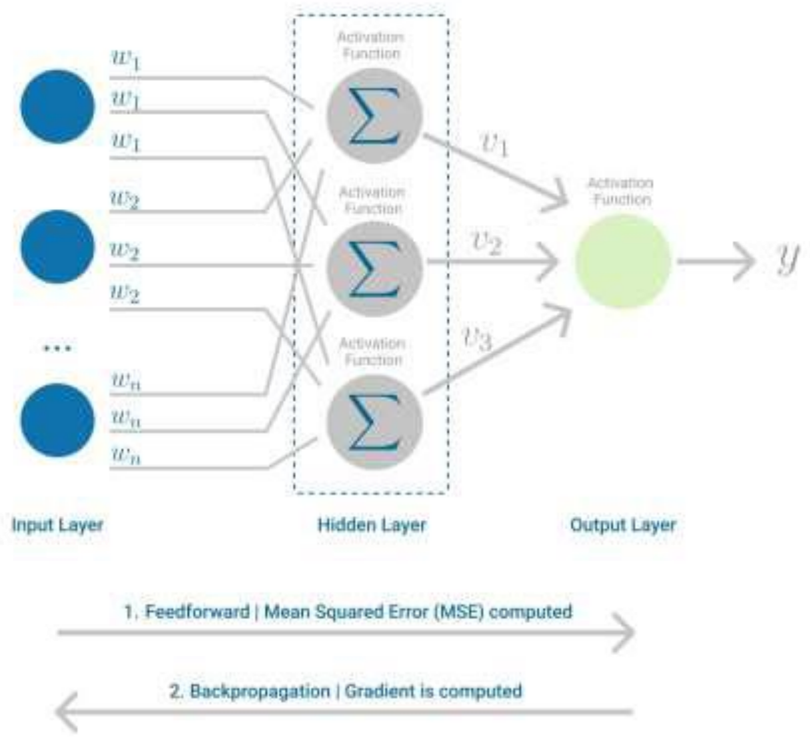


Multilayer Perceptron
Multilayer Perceptron falls under the category of feedforward algorithms, because inputs are combined with the initial weights in a weighted sum and subjected to the activation function, just like in the Perceptron. But the difference is that each linear combination is propagated to the next layer. Each layer is *feeding* the next one with the result of their computation, their internal representation of the data. This goes all the way through the hidden layers to the output layer. If the algorithm only computed the weighted sums in each neuron, propagated results to the output layer, and stopped there, it wouldn't be able to *learn* the weights that minimize the cost function. If the algorithm only computed one iteration, there would be no actual learning.

This is where **Backpropagation** comes into play.

Backpropagation

Backpropagation is the learning mechanism that allows the Multilayer Perceptron to iteratively adjust the weights in the network, with the goal of minimizing the cost function.
There is one hard requirement for backpropagation to work properly. The function that combines inputs and weights in a neuron, for instance the weighted sum, and the threshold function, for instance ReLU, must be differentiable. These functions must have a **bounded derivative**, because Gradient Descent is typically the optimization function used in MultiLayer Perceptron.



Multilayer Perceptron, highlighting the Feedfoward and Backpropagation steps.

In each iteration, after the weighted sums are forwarded through all layers, the gradient of the **Mean Squared Error** is computed across all input and output pairs. Then, to propagate it back, the weights of the first hidden layer are updated with the value of the gradient. That's how the weights are propagated back to the starting point of the neural network!

$$\underbrace{\Delta_w(t)}_{\text{Gradient Current Iteration}} = -\overset{\text{Bias}}{\varepsilon}\frac{\overbrace{dE}^{\text{Error}}}{\underbrace{dw_{(t)}}_{\text{Weight vector}}} + \overset{\text{Learning Rate}}{\alpha}\underbrace{\Delta_{w(t-1)}}_{\text{Gradient Previous Iteration}}$$

One iteration of Gradient Descent.

This process keeps going until gradient for each input-output pair has converged, meaning the newly computed gradient hasn't changed more than a specified *convergence threshold*, compared to the previous iteration.
MLPClassifier supports multi-class classification by applying Softmax as the output function.
Further, the model supports multi-label classification in which a sample can belong to more than one class. For each class, the raw output passes through the logistic function. Values larger or equal to 0.5 are rounded to 1, otherwise to 0. For a predicted output of a sample, the indices where the value is 1 represents the assigned classes of that sample:

```
>>> clf.predict([[0., 0.]])
```

```
array([[0, 1]])
>>> X = [[0., 0.], [1., 1.]]
>>> y = [[0, 1], [1, 1]]
>>> clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
... hidden_layer_sizes=(15,), random_state=1)
>>> clf.fit(X, y)
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(15,), random_state=1,
solver='lbfgs')
>>> clf.predict([[1., 2.]])
array([[1, 1]])
```

Regression

Class MLPRegressor implements a multi-layer perceptron (MLP) that trains using backpropagation with no activation function in the output layer, which can also be seen as using the identity function as activation function. Therefore, it uses the square error as the loss function, and the output is a set of continuous values.
MLPRegressor also supports multi-output regression, in which a sample can have more than one target.

Regularization

Both MLPRegressor and MLPClassifier use parameter alpha for regularization (L2 regularization) term which helps in avoiding overfitting by penalizing weights with large magnitudes.

Algorithms

MLP trains using Stochastic Gradient Descent, Adam, or L-BFGS. Stochastic Gradient Descent (SGD) updates parameters using the gradient of the loss function with respect to a parameter that needs adaptation, i.e. $w \leftarrow w - \eta(\alpha \partial R(w) \partial w + \partial Loss \partial w)$ where η is the learning rate which controls the step-size in the parameter space search. Loss is the loss function used for the network.
Adam is similar to SGD in a sense that it is a stochastic optimizer, but it can automatically adjust the amount to update parameters based on adaptive estimates of lower-order moments.

With SGD or Adam, training supports online and mini-batch learning.

L-BFGS is a solver that approximates the Hessian matrix which represents the second-order partial derivative of a function. Further it approximates the inverse of the Hessian matrix to perform parameter updates. The implementation uses the Scipy version of L-BFGS.
If the selected solver is ‘L-BFGS’, training does not support online nor mini-batch learning.

Procedure:

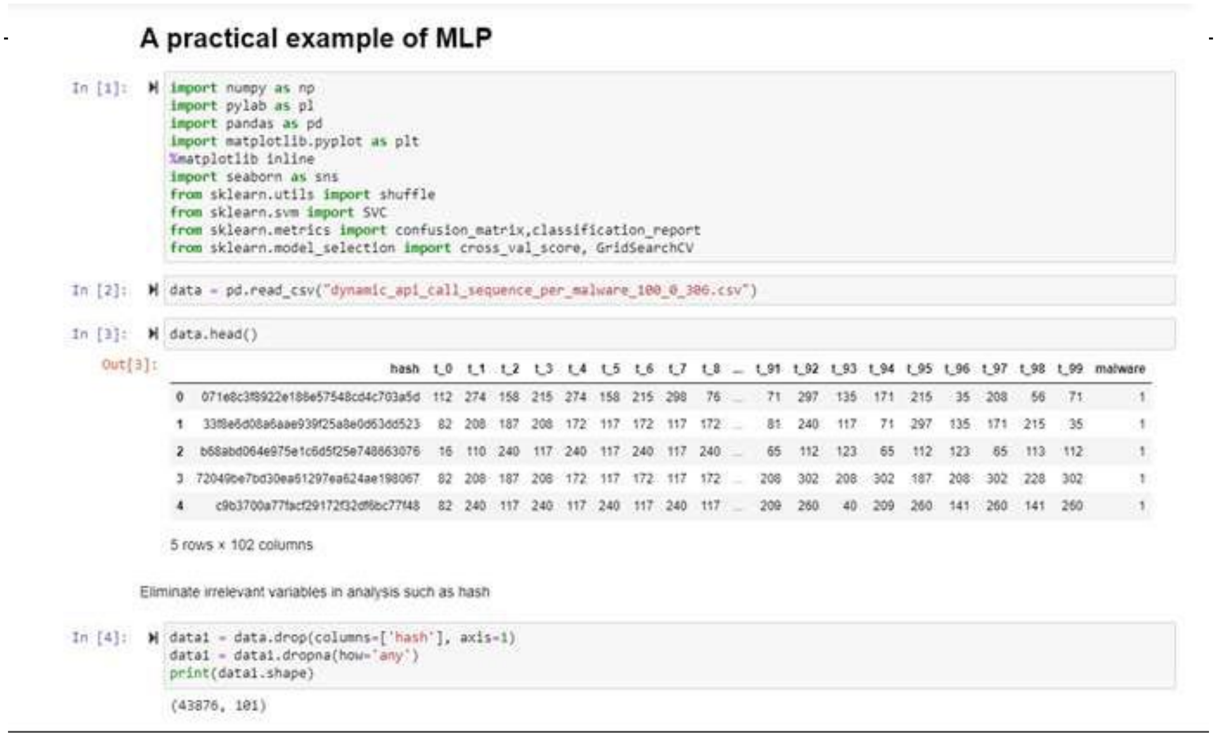
Python Code

```
import pandas
import numpy as np
```

```
def logistic(x):
return 1.0/(1 + np.exp(-x))
def logistic_deriv(x):
return logistic(x) * (1 - logistic(x))
LR = 1
I_dim = 3
H_dim = 4
epoch_count = 1
#np.random.seed(1)
weights_ItoH = np.random.uniform(-1, 1, (I_dim, H_dim))
weights_HtoO = np.random.uniform(-1, 1, H_dim)
preActivation_H = np.zeros(H_dim)
postActivation_H = np.zeros(H_dim)
training_data = pandas.read_excel('MLP_Tdata.xlsx')
target_output = training_data.output
training_data = training_data.drop(['output'], axis=1)
training_data = np.asarray(training_data)
training_count = len(training_data[:,0])
validation_data = pandas.read_excel('MLP_Vdata.xlsx')
validation_output = validation_data.output

validation_data = validation_data.drop(['output'], axis=1)
validation_data = np.asarray(validation_data)
validation_count = len(validation_data[:,0])
#####################
#training
#####################
for epoch in range(epoch_count):
for sample in range(training_count):
for node in range(H_dim):
preActivation_H[node] = np.dot(training_data[sample,:], weights_ItoH[:, node])
postActivation_H[node] = logistic(preActivation_H[node]
preActivation_O = np.dot(postActivation_H, weights_HtoO)
postActivation_O = logistic(preActivation_O)
FE = postActivation_O - target_output[sample]
for H_node in range(H_dim):
S_error = FE * logistic_deriv(preActivation_O)
gradient_HtoO = S_error * postActivation_H[H_node]
for I_node in range(I_dim):
input_value = training_data[sample, I_node]
gradient_ItoH = S_error * weights_HtoO[H_node] *
logistic_deriv(preActivation_H[H_node]) * input_value
weights_ItoH[I_node, H_node] -= LR * gradient_ItoH
weights_HtoO[H_node] -= LR * gradient_HtoO
```

```
#####################
#validation
#####################
correct_classification_count = 0
for sample in range(validation_count):
for node in range(H_dim):
preActivation_H[node] = np.dot(validation_data[sample,:], weights_ItoH[:, node])
postActivation_H[node] = logistic(preActivation_H[node]
preActivation_O = np.dot(postActivation_H, weights_HtoO)
postActivation_O = logistic(preActivation_O)
if postActivation_O > 0.5:
output = 1
else:
output = 0
if output == validation_output[sample]:
correct_classification_count += 1
print('Percentage of correct classifications:')
print(correct_classification_count*100/validation_count)
```

Output Screen



A practical example of MLP

```
In [1]:  import numpy as np
         import pylab as pl
         import pandas as pd
         import matplotlib.pyplot as plt
         %matplotlib inline
         import seaborn as sns
         from sklearn.utils import shuffle
         from sklearn.svm import SVC
         from sklearn.metrics import confusion_matrix,classification_report
         from sklearn.model_selection import cross_val_score, GridSearchCV

In [2]:  data = pd.read_csv("dynamic_api_call_sequence_per_malware_100_0_306.csv")

In [3]:  data.head()

Out[3]:
```

	hash	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	...	t_91	t_92	t_93	t_94	t_95	t_96	t_97	t_98	t_99	malware
0	071e8c3f8922e186e57548cd4c703a5d	112	274	158	215	274	158	215	298	76	...	71	297	135	171	215	35	208	56	71	1
1	33f8e6d08a6aae939f25a8e0d63dd523	82	208	187	208	172	117	172	117	172	...	81	240	117	71	297	135	171	215	35	1
2	b68abd064e975e1c6d5f25e748663076	16	110	240	117	240	117	240	117	240	...	65	112	123	65	112	123	65	113	112	1
3	72049be7bd30ea61297ea624ae198067	82	208	187	208	172	117	172	117	172	...	208	302	208	302	187	208	302	228	302	1
4	c9b3700a77facf29172f32df6bc77f48	82	240	117	240	117	240	117	240	117	...	209	260	40	209	260	141	260	141	260	1

5 rows × 102 columns

Eliminate irrelevant variables in analysis such as hash

```
In [4]:  data1 = data.drop(columns=['hash'], axis=1)
         data1 = data1.dropna(how='any')
         print(data1.shape)

         (43876, 101)
```

SPLITING DATA

Data for training and testing To select a set of training data that will be input in the Machine Learning algorithm, to ensure that the classification algorithm training can be generalized well to new data. For this study using a sample size of 20%.

```
In [6]:  from sklearn.model_selection import train_test_split
         Y = data1['malware']
         X = data1.drop(columns=['malware'])
         X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=9)
```

```
In [7]:  print('X train shape: ', X_train.shape)
         print('Y train shape: ', Y_train.shape)
         print('X test shape: ', X_test.shape)
         print('Y test shape: ', Y_test.shape)
```

```
X train shape:  (35100, 100)
Y train shape:  (35100,)
X test shape:  (8776, 100)
Y test shape:  (8776,)
```

Build Model use MLP

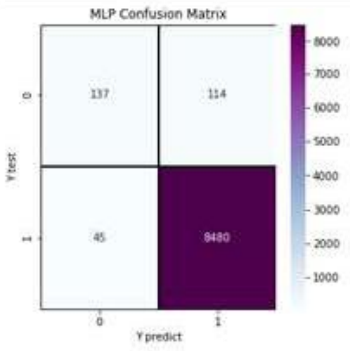
```
In [8]:  from sklearn.neural_network import MLPClassifier

         # We define the model
         mlp = MLPClassifier(hidden_layer_sizes=(100,100,100),max_iter=1000, random_state=42)

         # We train model
         mlp.fit(X_train, Y_train)


         # We predict target values
         prediction = mlp.predict(X_test)
```

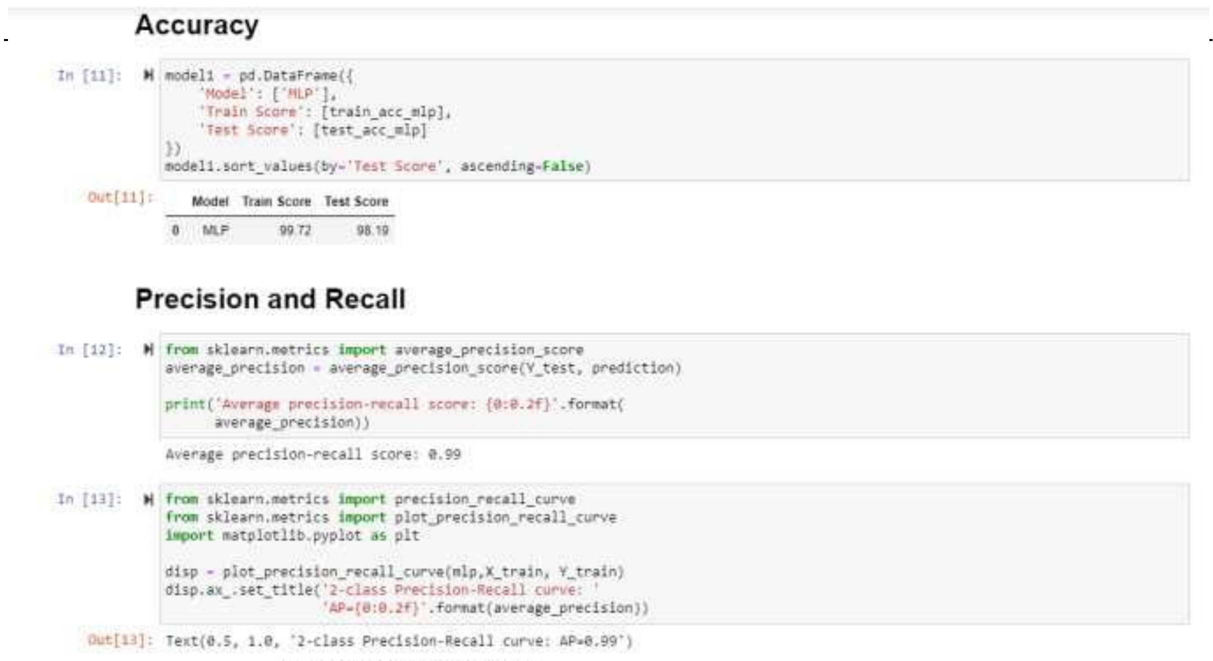
```
In [9]:  test_acc_mlp = round(mlp.fit(X_train,Y_train).score(X_test, Y_test)* 100, 2)
         train_acc_mlp = round(mlp.fit(X_train, Y_train).score(X_train, Y_train)* 100, 2)
```

Measurement

When the classification process was already done. This work evaluated the results using the Confusion Matrix.

```
In [10]:  # The confusion matrix
          mlp_cm = confusion_matrix(Y_test, prediction)
          f, ax = plt.subplots(figsize=(5,5))
          sns.heatmap(mlp_cm, annot=True, linewidth=0.7, linecolor='black', fmt='g', ax=ax, cmap="BuPu")
          plt.title('MLP Confusion Matrix')
          plt.xlabel('Y predict')
          plt.ylabel('Y test')
          plt.show()
```





Accuracy

```
In [11]:  model1 = pd.DataFrame({
              'Model': ['MLP'],
              'Train Score': [train_acc_mlp],
              'Test Score': [test_acc_mlp]
          })
          model1.sort_values(by='Test Score', ascending=False)
```

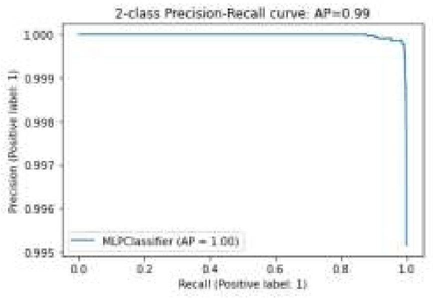
Out[11]:

	Model	Train Score	Test Score
0	MLP	99.72	98.19

Precision and Recall

```
In [12]:  from sklearn.metrics import average_precision_score
          average_precision = average_precision_score(Y_test, prediction)

          print('Average precision-recall score: {0:0.2f}'.format(
                average_precision))

          Average precision-recall score: 0.99
```

```
In [13]:  from sklearn.metrics import precision_recall_curve
          from sklearn.metrics import plot_precision_recall_curve
          import matplotlib.pyplot as plt

          disp = plot_precision_recall_curve(mlp,X_train, Y_train)
          disp.ax_.set_title('2-class Precision-Recall curve: '
                             'AP={0:0.2f}'.format(average_precision))

Out[13]:  Text(0.5, 1.0, '2-class Precision-Recall curve: AP=0.99')
```



Further Reading

1. LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* 521, 436–444 (2015)
2. Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. Deep Learning. The MIT Press.
3. McCulloch, W.S., Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics* 5, 115–133 (1943)
4. Frank Rosenblatt. The Perceptron, a Perceiving and Recognizing Automaton Project Para. *Cornell Aeronautical Laboratory* 85, 460–461 (1957)
5. Minsky M. L. and Papert S. A. 1969. *Perceptrons*. Cambridge, MA: MIT Press.
6. Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. (2013). An introduction to statistical learning : with applications in R. New York :Springer
7. D. Rumelhart, G. Hinton, and R. Williams. Learning Representations by Back-propagating Errors. *Nature* 323 (6088): 533–536 (1986).

Prospective Viva Questions

1. What is multilayer perceptron in neural network?
2. Is multilayer perceptron the same as neural network?
3. What is MLP and why is it used?
4. How does a multi layer perceptron work?
5. Is CNN a MLP?
6. What is the most basic neural network?

EXPERIMENT 2.3

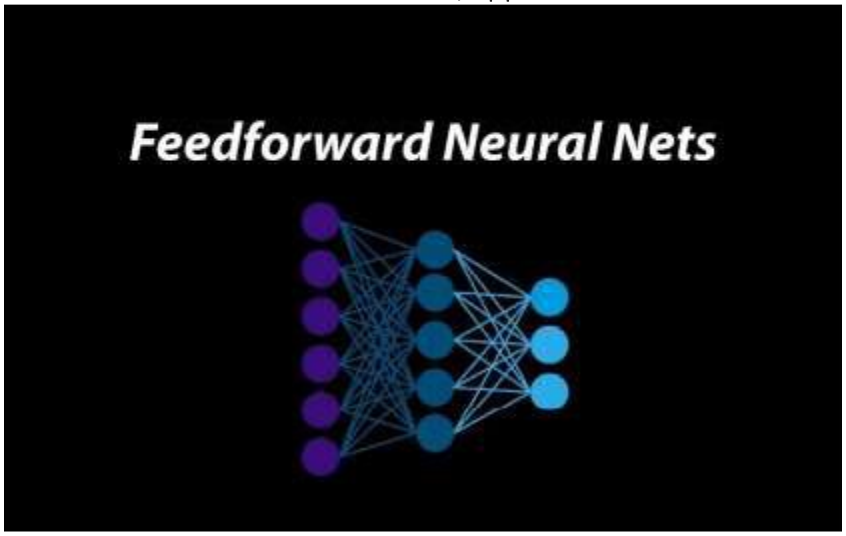
Mapped Course Outcome

CO2 Recognize the feasibility of applying a soft computing methodology for a particular problem.
CO4 Effectively use modern software tools to solve real problems using a soft computing approach.
CO5 Examine and evaluate various soft computing approaches for a given problem.

AIM: Write a program to implement feed forward network

Theory

A Feed Forward Neural Network is an artificial Neural Network in which the nodes are connected circularly. A feed-forward neural network, in which some routes are cycled, is the polar opposite of a Recurrent Neural Network. The feed-forward model is the basic type of neural network because the input is only processed in one direction. The data always flows in one direction and never backwards/opposite.

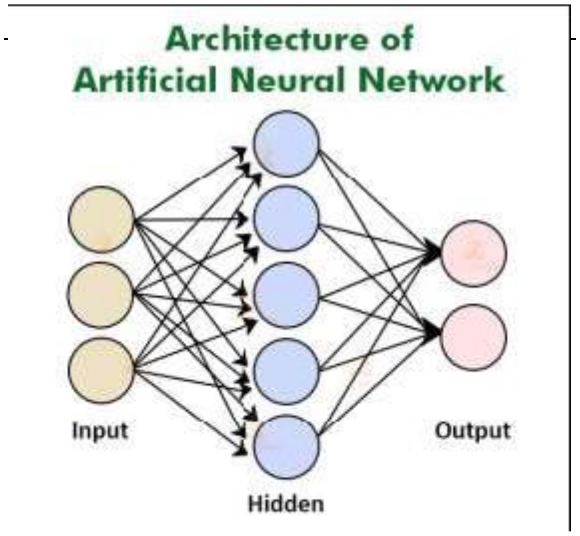


The Neural Network advanced from the perceptron, a prominent machine learning algorithm. Frank Rosenblatt, a physicist, invented perceptrons in the 1950s and 1960s, based on earlier work by Warren McCulloch and Walter Pitts.

Why are Neural Networks used?
Neural Networks are a type of function that connects inputs with outputs. In theory, neural networks should be able to estimate any sort of function, no matter how complex it is.
Nonetheless, supervised learning entails learning a function that translates a given X to a specified Y and then utilising that function to determine the proper Y for a fresh X. If that's the case, how do neural networks differ from typical machine learning methods? Inductive Bias, a psychological phenomenon, is the answer. The phrase may appear to be fresh. However, before applying a machine learning model to it, it is nothing more than our assumptions about the relationship between X and Y. The linear relationship between X and Y is the Inductive Bias of linear regression. As a result, it fits the data to a line or a hyperplane.
When there is a non-linear and complex relationship between X and Y, nevertheless, a Linear Regression method may struggle to predict Y. To approximate that relationship, we may need a curve or a multi-dimensional curve in this scenario.However, depending on the function's complexity, we may need to manually set the number of neurons in each layer and the total number of layers in the network. This is usually accomplished through trial and error methods as well as experience. As a result, these parameters are referred to as hyperparameters.

Neural Network Architecture and Operation ***Before we look at why neural networks work, it's important to understand what neural networks do. Before we can grasp the design of a neural network, we must first understand what a neuron performs***.

A weight is assigned to each input to an artificial neuron. First, the inputs are multiplied by their weights, and then a bias is applied to the outcome. After that, the weighted sum is passed via an activation function, being a non-linear function.

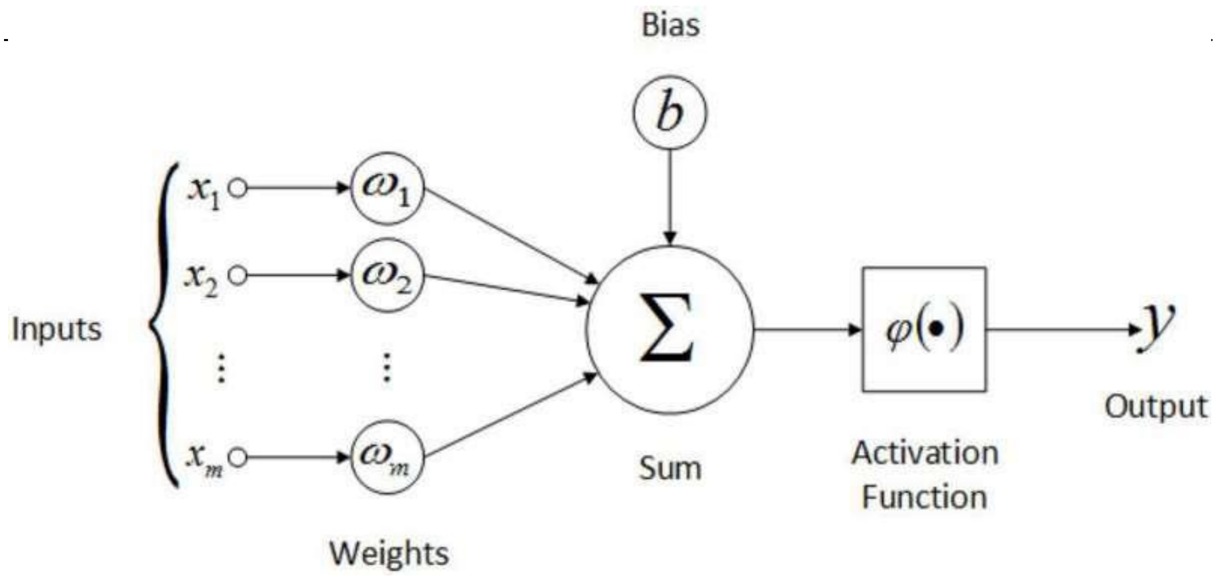


Source: TowardsDatascience

A weight is being applied to each input to an artificial neuron. First, the inputs are multiplied by their weights, and then a bias is applied to the outcome. This is called the weighted sum. After that, the weighted sum is processed via an activation function, as a non-linear function. The first layer is the input layer, which appears to have six neurons but is only the data that is sent into the neural network. The output layer is the final layer. The dataset and the type of challenge determine the number of neurons in the final layer and the first layer. Trial and error will be used to determine the number of neurons in the hidden layers and the number of hidden layers.

All of the inputs from the previous layer will be connected to the first neuron from the first hidden layer. The second neuron in the first hidden layer will be connected to all of the preceding layer's inputs, and so forth for all of the first hidden layer's neurons. The outputs of the previously hidden layer are regarded inputs for neurons in the second hidden layer, and each of these neurons is coupled to all of the preceding neurons.

What is a Feed-Forward Neural Network and how does it work?
In its most basic form, a Feed-Forward Neural Network is a single layer perceptron. A sequence of inputs enter the layer and are multiplied by the weights in this model. The weighted input values are then summed together to form a total. If the sum of the values is more than a predetermined threshold, which is normally set at zero, the output value is usually 1, and if the sum is less than the threshold, the output value is usually -1. The single-layer perceptron is a popular feed-forward neural network model that is frequently used for classification. Single-layer
perceptrons can also contain machine learning features.



Source: Medium.com

The neural network can compare the outputs of its nodes with the desired values using a property known as the delta rule, allowing the network to alter its weights through training to create more accurate output values. This training and learning procedure results in gradient descent. The technique of updating weights in multi-layered perceptrons is virtually the same, however, the process is referred to as back-propagation. In such circumstances, the output values provided by the final layer are used to alter each hidden layer inside the network.

Python Implementation
Representing the feed-forward neural network using Python
Let us create the respective sample weights which are to be applied in the input layer, the first & the second hidden layer

```
import numpy as np
from sklearn import datasets
#
# Generate a dataset and plot it
#
np.random.seed(0)
X, y = datasets.make_moons(200, noise=0.20)
#
# Neural network architecture
# No of nodes in input layer = 4
# No of nodes in output layer = 3
# No of nodes in the hidden layer = 6
#
input_dim = 4 # input layer dimensionality
output_dim = 3 # output layer dimensionality
hidden_dim = 6 # hidden layer dimensionality
```

```
#
# Weights and bias element for layer 1
# These weights are applied for calculating
# weighted sum arriving at neurons in 1st hidden layer
#
W1 = np.random.randn(input_dim, hidden_dim)
b1 = np.zeros((1, hidden_dim))
#
# Weights and bias element for layer 2
# These weights are applied for calculating
# weighted sum arriving at neurons in 2nd hidden layer
#
W2 = np.random.randn(hidden_dim, hidden_dim)
b2 = np.zeros((1, hidden_dim))
#
# Weights and bias element for layer 2
# These weights are applied for calculating
# weighted sum arriving at in the final / output layer
#
W3 = np.random.randn(hidden_dim, output_dim)
b3 = np.zeros((1, output_dim))
```

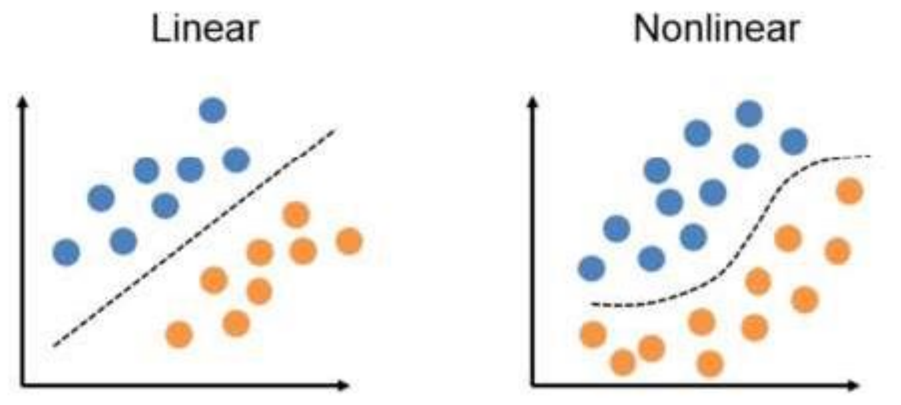
Python code implementation for the propagation of the input signal through different layers towards the output layer

```
#
# Forward propagation of input signals
# to 6 neurons in first hidden layer
# activation is calculated based tanh function
#
z1 = X.dot(W1) + b1
a1 = np.tanh(z1)
#
# Forward propagation of activation signals from first hidden layer
# to 6 neurons in second hidden layer
# activation is calculated based tanh function
#
z2 = a1.dot(W2) + b2
a2 = np.tanh(z2)
#
# Forward propagation of activation signals from second hidden layer
# to 3 neurons in output layer
#
z3 = a2.dot(W3) + b3
#
```

```
# Probability is calculated as an output
# of softmax function
#
probs = np.exp(z3) / np.sum(np.exp(z3), axis=1, keepdims=True)
```

Why does this Strategy Work?

As we've seen, the function of each neuron in the network is similar to that of linear regression. The neuron also has an activation function at the end, and each neuron has its weight vector.



Source: Medium.com

We've seen how the computation works so far. But the major purpose of this blog is to explain why this strategy works. Neural networks should theoretically be able to estimate any continuous function, no matter how complex or non-linear it is.

Importance of the Non-Linearity

When ***two or more linear objects, such as a line, plane, or hyperplane, are combined, the outcome is also a linear object: line, plane, or hyperplane***. No matter how many of these linearthings we add, we'll still end up with a linear object.

However, this is not the case when adding non-linear objects. When two separate curves are combined, the result is likely to be a more complex curve.

We're introducing non-linearity at every layer using these activation functions, in addition to just adding non-linear objects or hyper-curves like hyperplanes. In other words, we're applying a nonlinear function on an already nonlinear object.

What if activation functions were not used in neural networks?
Suppose if neural networks didn't have an activation function, they'd just be a huge linear unit that a single linear regression model could easily replace.

a = m*x + d

Z= k*a + t => k*(m*x+d) + t => k*m*x + k*d + t => (k*m)*x + (k*c+t)

Applications of the Feed Forward Neural Networks

A Feed Forward Neural Network is an artificial neural network in which the nodes are connected circularly. A feed-forward neural network, in which some routes are cycled, is the polar opposite of a recurrent neural network. The feed-forward model is the simplest type of neural network because the input is only processed in one direction. The data always flows in one direction and never backwards, regardless of how many buried nodes it passes through.

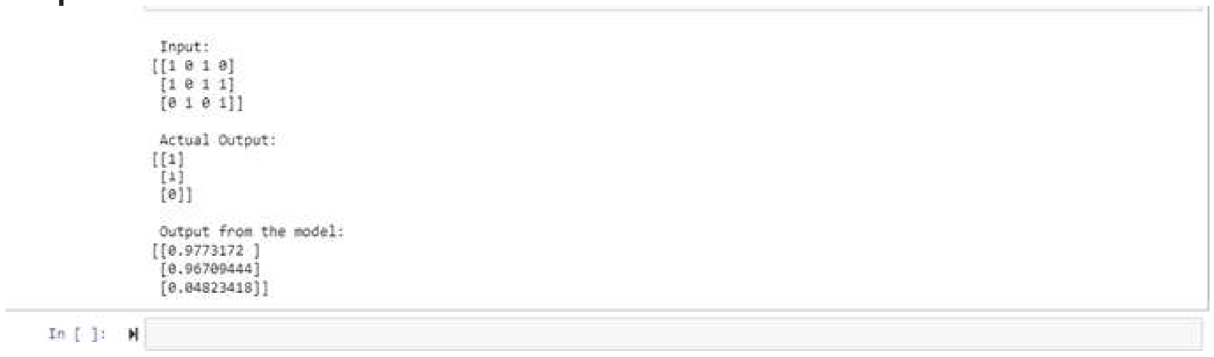
Procedure:

Python Code

```
# importing the library
import numpy as np
#load input
#creating the input array
X=np.array([[1,0,1,0],[1,0,1,1],[0,1,0,1]])
print ('\n Input:')
print(X)
# creating the output array
y=np.array([[1],[1],[0]])
print('\n Actual Output:')
print(y)
#defining the Sigmoid Function
def sigmoid (x):
return 1/(1+ np.exp(-x))
# derivative of Sigmoid Function
def derivatives_sigmoid(x):

return x *(1-x)
# initializing the variables
epoch=5000 #number of training iterations
lr=0.1 # learning rate
inputlayer_neurons = X.shape[1] # number of features in data set
hiddenlayer_neurons =3 # number of hidden layers neurons
output_neurons = 1 # number of neurons at output layer.
# initializing weight and bias
wh=np.random.uniform(size=(inputlayer_neurons, hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size= (hiddenlayer_neurons, output_neurons))
bout=np.random.uniform(size=(1, output_neurons))
#training the model
for i in range(epoch):
#Forward Propogation
hidden_layer_input1=np.dot(X, wh)
hidden_layer_input=hidden_layer_input1+ bh
hiddenlayer_activations = sigmoid(hidden_layer_input)
```

```
output_layer_input1=np.dot(hiddenlayer_activations, wout)
output_layer_input= output_layer_input1+ bout
output= sigmoid(output_layer_input)
#Backpropagation
E = y-output
slope_output_layer= derivatives_sigmoid(output)
slope_hidden_layer= derivatives_sigmoid(hiddenlayer_activations)
d_output= E*slope_output_layer
Error_at_hidden_layer=d_output.dot(wout.T)
d_hiddenlayer=Error_at_hidden_layer*slope_hidden_layer
wout += hiddenlayer_activations.T.dot(d_output)*lr
bout += np.sum(d_output, axis=0, keepdims=True)*lr
wh += X.T.dot (d_hiddenlayer)*lr
bh+= np.sum(d_hiddenlayer, axis=0, keepdims=True)*lr
print ('\n Output from the model:')
print (output)
```

Output Screen



```
 Input:
[[1 0 1 0]
 [1 0 1 1]
 [0 1 0 1]]

 Actual Output:
[[1]
 [1]
 [0]]

 Output from the model:
[[0.9773172 ]
 [0.96709444]
 [0.04823418]]
```

Further Reading

Deep Learning Book" by Goodfellow, Bengio, and Courville.
http://neuralnetworksanddeeplearning.com/

Journal Papers

https://www.sciencedirect.com/science/article/pii/S1877050916325467

Main text books:

• "Neural Networks: A Comprehensive Foundation", S. Haykin
• "Pattern Recognition with Neural Networks", C. Bishop
• "NeuralNetwork Design" by Hagan, Demuth and Beale

Books emphasizing the practical aspects:

• "Neural Smithing", Reeds and Marks
• "Practical Neural Network Recipees in C++"' T. Masters
• "Parallel Distributed Processing" Rumelhart and McClelland et al.

Deep Learning books and tutorials:

• http://www.deeplearningbook.org/

• Introductionto Learning Rules in Neural Network - DataFlair (data-flair.training)

Prospective Viva Questions
1. Is RNN a feed forward neural network?
2. Why we use CNN instead of ANN?
3. Why is CNN better than RNN?
4. Is RNN a feed forward neural network?
5. Why we use CNN instead of ANN?
6. Why is CNN better than RNN?
7. How do I create a feedforward neural network?
8. How do I train feed forward in neural network?
9. What is feed forward backpropagation?

EXPERIMENT - 3.1

Mapped Course Outcome

CO3	Apply fuzzy logic and reasoning to handle uncertainty and solve engineering problems, genetic algorithms to combinatorial optimization problems and neural networks to pattern classification and regression problems.

AIM: Write a program to implement common operations on Fuzzy Set.

Union

Consider 2 Fuzzy Sets denoted by A and B, then let’s consider Y be the Union of them, then for every member of A and B, Y will be:

degree_of_membership(Y)= max(degree_of_membership(A), degree_of_membership(B))

Intersection

Consider 2 Fuzzy Sets denoted by A and B, then let’s consider Y be the Intersection of them, then for every member of A and B, Y will be: degree_of_membership(Y)= min(degree_of_membership(A), degree_of_membership(B))

Complement :

Consider a Fuzzy Sets denoted by A , then let’s consider Y be the Complement of it, then for every member of A , Y will be:
degree_of_membership(Y)= 1 - degree_of_membership(A)

Difference :

Consider 2 Fuzzy Sets denoted by A and B, then let’s consider Y be the Intersection of them, then for every member of A and B, Y will be:

degree_of_membership(Y)= min(degree_of_membership(A), 1- degree_of_membership(B))

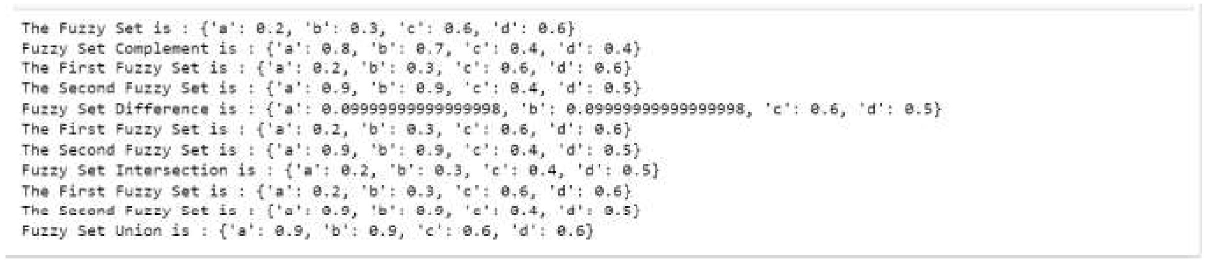
Procedure:

Python Code

```
# degree_of_membership(Y)= 1 - degree_of_membership(A)
# Example to Demonstrate the
# Difference Between Two Fuzzy Sets
A = dict()
Y = dict()
A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}
print('The Fuzzy Set is :', A)
for A_key in A:
Y[A_key]= 1-A[A_key]
print('Fuzzy Set Complement is :', Y)
# degree_of_membership(Y)= min(degree_of_membership(A), 1- degree_of_membership(B))
# Example to Demonstrate the
# Difference Between Two Fuzzy Sets
A = dict()
B = dict()
Y = dict()
A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}
B = {"a": 0.9, "b": 0.9, "c": 0.4, "d": 0.5}
print('The First Fuzzy Set is :', A)
print('The Second Fuzzy Set is :', B)
for A_key, B_key in zip(A, B):
A_value = A[A_key]
B_value = B[B_key]
B_value = 1 - B_value
if A_value < B_value:
Y[A_key] = A_value
```

```
else:
Y[B_key] = B_value
Print('Fuzzy Set Difference is :', Y)
# degree_of_membership(Y)= min(degree_of_membership(A), degree_of_membership(B))
# Example to Demonstrate
# Intersection of Two Fuzzy Sets
A = dict()
B = dict()
Y = dict()
A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}
B = {"a": 0.9, "b": 0.9, "c": 0.4, "d": 0.5}
print('The First Fuzzy Set is :', A)
print('The Second Fuzzy Set is :', B)
for A_key, B_key in zip(A, B):
A_value = A[A_key]
B_value = B[B_key]
if A_value < B_value:
Y[A_key] = A_value
else:
Y[B_key] = B_value
print('Fuzzy Set Intersection is :', Y)
# Example to Demonstrate the
# Union of Two Fuzzy Sets
A = dict()
B = dict()
Y = dict()
A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}
B = {"a": 0.9, "b": 0.9, "c": 0.4, "d": 0.5}
print('The First Fuzzy Set is :', A)
print('The Second Fuzzy Set is :', B)
for A_key, B_key in zip(A, B):
A_value = A[A_key]
B_value = B[B_key]
if A_value > B_value:
Y[A_key] = A_value
else:
Y[B_key] = B_value
print('Fuzzy Set Union is :', Y)
```

Output Screen:



```
The Fuzzy Set is : {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}
Fuzzy Set Complement is : {'a': 0.8, 'b': 0.7, 'c': 0.4, 'd': 0.4}
The First Fuzzy Set is : {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}
The Second Fuzzy Set is : {'a': 0.9, 'b': 0.9, 'c': 0.4, 'd': 0.5}
Fuzzy Set Difference is : {'a': 0.09999999999999998, 'b': 0.09999999999999998, 'c': 0.6, 'd': 0.5}
The First Fuzzy Set is : {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}
The Second Fuzzy Set is : {'a': 0.9, 'b': 0.9, 'c': 0.4, 'd': 0.5}
Fuzzy Set Intersection is : {'a': 0.2, 'b': 0.3, 'c': 0.4, 'd': 0.5}
The First Fuzzy Set is : {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}
The Second Fuzzy Set is : {'a': 0.9, 'b': 0.9, 'c': 0.4, 'd': 0.5}
Fuzzy Set Union is : {'a': 0.9, 'b': 0.9, 'c': 0.6, 'd': 0.6}
```

Video Tutorial
https://www.youtube.com/watch?v=a2i-lHS-c_I

Further Reading

Çakıt, E. (2022). A Systematic Review of Fuzzy Logic Applications for the COVID-19 Pandemic. *Big Data Analytics and Artificial Intelligence in the Healthcare Industry*, 101-128.

Hosseinpour, S., & Martynenko, A. (2022). Application of fuzzy logic in drying: A review. *Drying Technology*, *40*(5), 797-826.

Prospective Viva Questions

1. What is a fuzzy set explain with example?
2. What is fuzzy set and its operations?
3. What is fuzzy and crisp set?
4. What is a fuzzy value?
5. What is a normal fuzzy set?
6. Why is fuzzy number important?
7. What is fuzzy theory?
8. What is a fuzzy rule base?
9. What are the types of fuzzy logic sets?

EXPERIMENT – 3.2

Mapped Course Outcome

CO3	Apply fuzzy logic and reasoning to handle uncertainty and solve engineering problems, genetic algorithms to combinatorial optimization problems and neural networks to pattern classification and regression problems.
CO4	Effectively use modern software tools to solve real problems using a soft computing approach.

CO5	Evaluate various soft computing approaches for a given problem.

AIM: Write a program to implement Fuzzy Control System: Tipping Problem.

Theory

The 'tipping problem' is commonly used to illustrate the power of fuzzy logic principles to generate complex behavior from a compact, intuitive set of expert rules.

Example The Tipping Problem

Let’s create a fuzzy control system which models how you might choose to tip at a restaurant. When tipping, you consider the service and food quality, rated between 0 and 10. You use this to leave a tip of between 0 and 25%.
We would formulate this problem as:
* Antecednets (Inputs)
- `service`
* Universe (ie, crisp value range): How good was the service of the wait staff, on a scale of 0 to 10?
* Fuzzy set (ie, fuzzy value range): poor, acceptable, amazing
- `food quality`
* Universe: How tasty was the food, on a scale of 0 to 10?

* Fuzzy set: bad, decent, great
* Consequents (Outputs)
- `tip`
* Universe: How much should we tip, on a scale of 0% to 25%
* Fuzzy set: low, medium, high
* Rules
- IF the *service* was good *or* the *food quality* was good,
THEN the tip will be high.
- IF the *service* was average, THEN the tip will be medium.
- IF the *service* was poor *and* the *food quality* was poor
THEN the tip will be low.
* Usage
- If I tell this controller that I rated:
* the service as 9.8, and
* the quality as 6.5,
- it would recommend I leave:
* a 20.2% tip.
Creating the Tipping Controller Using the skfuzzy control API
We can use the `skfuzzy` control system API to model this. First, let's
define fuzzy variables

```
"""
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
# New Antecedent/Consequent objects hold universe variables and membership
# functions
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')
# Auto-membership function population is possible with .automf(3, 5, or 7)
quality.automf(3)
service.automf(3)
# Custom membership functions can be built interactively with a familiar,
# Pythonic API
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])
"""
To help understand what the membership looks like, use the ``view`` methods.
"""
# You can see how these look with .view()
quality['average'].view()
"""
.. image:: PLOT2RST.current_figure
"""
service.view()
"""
.. image:: PLOT2RST.current_figure
"""
tip.view()
"""
.. image:: PLOT2RST.current_figure
Fuzzy rules
Now, to make these triangles useful, we define the *fuzzy relationship* between input and
output variables. For the purposes of our example, consider three simple rules:
1. If the food is poor OR the service is poor, then the tip will be low
2. If the service is average, then the tip will be medium
3. If the food is good OR the service is good, then the tip will be high.
Most people would agree on these rules, but the rules are fuzzy. Mapping the
imprecise rules into a defined, actionable tip is a challenge. This is the
kind of task at which fuzzy logic excels.
"""
rule1 = ctrl.Rule(quality['poor'] | service['poor'], tip['low'])
rule2 = ctrl.Rule(service['average'], tip['medium'])
rule3 = ctrl.Rule(service['good'] | quality['good'], tip['high'])
rule1.view()
```

.. image:: PLOT2RST.current_figure
Control System Creation and Simulation

We can simply create a control system via:
tipping_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
In order to simulate this control system, we will create a
``ControlSystemSimulation``. Think of this object representing our controller
applied to a specific set of cirucmstances. For tipping, this might be tipping
Sharon at the local brew-pub. We would create another
``ControlSystemSimulation`` when we're trying to apply our ``tipping_ctrl``
for Travis at the cafe because the inputs would be different.
"""
tipping = ctrl.ControlSystemSimulation(tipping_ctrl)
"""
We can now simulate our control system by simply specifying the inputs
and calling the ``compute`` method. Suppose we rated the quality 6.5 out of 10
and the service 9.8 of 10.
"""
# Pass inputs to the ControlSystem using Antecedent labels with Pythonic API
# Note: if you like passing many inputs all at once, use .inputs(dict_of_data)
tipping.input['quality'] = 6.5
tipping.input['service'] = 9.8
# Crunch the numbers
tipping.compute()
Once computed, we can view the result as well as visualize it.
print tipping.output['tip']
tip.view(sim=tipping)
image:: PLOT2RST.current_figure
The power of fuzzy systems is allowing complicated, intuitive behavior based
on a sparse system of rules with minimal overhead. Note our membership
function universes were coarse, only defined at the integers, but
``fuzz.interp_membership`` allowed the effective resolution to increase on
demand. This system can respond to arbitrarily small changes in inputs,
and the processing burden is minimal.

Procedure:

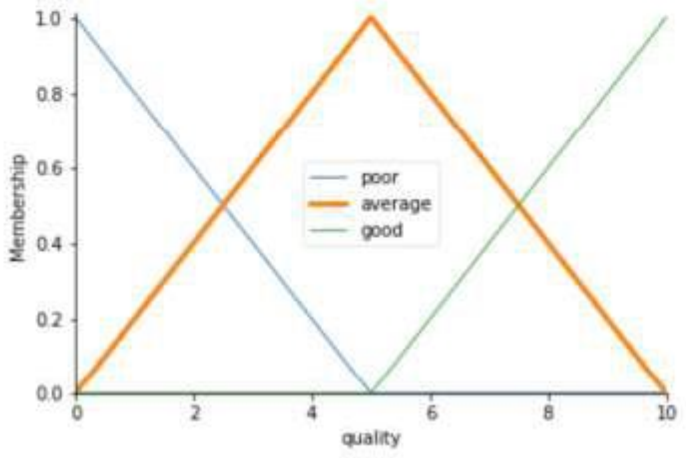
Python Code

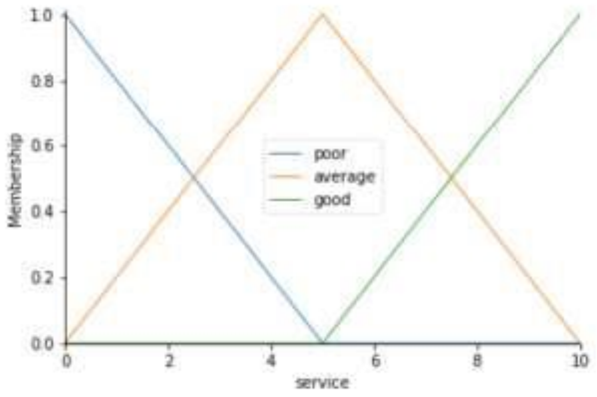
```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
# New Antecedent/Consequent objects hold
#universe variables and membership
# functions
quality = ctrl.Antecedent(np.arange(0, 11, 1), 'quality')
```

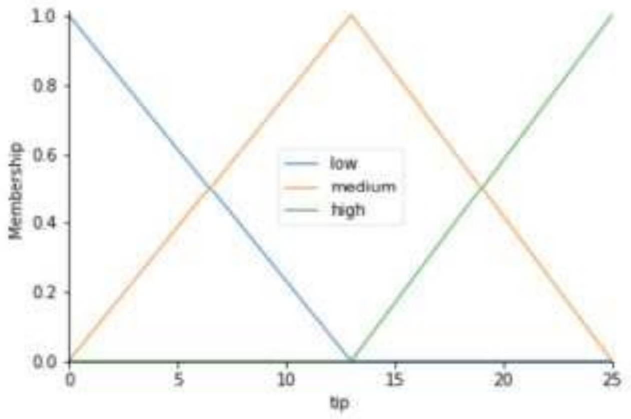
```
service = ctrl.Antecedent(np.arange(0, 11, 1), 'service')
tip = ctrl.Consequent(np.arange(0, 26, 1), 'tip')
# Auto-membership function population
# is possible with .automf(3, 5, or 7)
quality.automf(3)
service.automf(3)
# Custom membership functions can be
# built interactively with a familiar,
# Pythonic API
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'])
rule2 = ctrl.Rule(
service['average'], tip['medium'])
rule3 = ctrl.Rule(
service['good'] | quality['good'], tip['high'])
rule1.view()
tipping_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
tipping = ctrl.ControlSystemSimulation(tipping_ctrl)
tipping.input['quality'] = 6.5
tipping.input['service'] = 9.8
# Crunch the numbers
tipping.compute()
print(tipping.output['tip'])
tip.view(sim=tipping)
```

Output Screen:

```
19.847607361963192
```







Video Tutorial

https://www.youtube.com/watch?v=R4TPFpYXvS0

Further Reading

Papers

Chakraborty, D., & Debnath, S. S. (2022). Determination of Quality of Food and Service in Restaurants by Obtaining Tipping Percentage with the Help of Fuzzy Inference System. *AIJR Abstracts*, 81.

Prospective Viva Questions

1. What are the components of fuzzy logic?
2. How many parts are there in fuzzy logic architecture?
3. What is Type 2 fuzzy logic?
4. What is fuzzy rule base?
5. What is fuzzy data?
6. What are the advantages of fuzzy logic?
7. What are the types of fuzzy logic sets?
8. Does A.I use fuzzy logic?
9. Is fuzzy logic still used?

EXPERIMENT – 3.3

Mapped Course Outcome

CO3	Apply fuzzy logic and reasoning to handle uncertainty and solve engineering problems, genetic algorithms to combinatorial optimization problems and neural networks to pattern classification and regression problems.	**3**
CO4	Effectively use modern software tools to solve real problems using a soft computing approach.	**3**
CO5	Evaluate various soft computing approaches for a given problem.	**4**

AIM: Write a program to implement Fuzzy Inference System

Theory

Fuzzy Inference System is the key unit of a fuzzy logic system having decision making as its primary work. It uses the "IF…THEN" rules along with connectors "OR" or "AND" for drawing essential decision rules.

Characteristics of Fuzzy Inference System

Following are some characteristics of FIS −

• The output from FIS is always a fuzzy set irrespective of its input which can be fuzzy or crisp.

• It is necessary to have fuzzy output when it is used as a controller.

• A defuzzification unit would be there with FIS to convert fuzzy variables into crisp variables.

Functional Blocks of FIS

The following five functional blocks will help you understand the construction of FIS −

• **Rule Base** − It contains fuzzy IF-THEN rules.

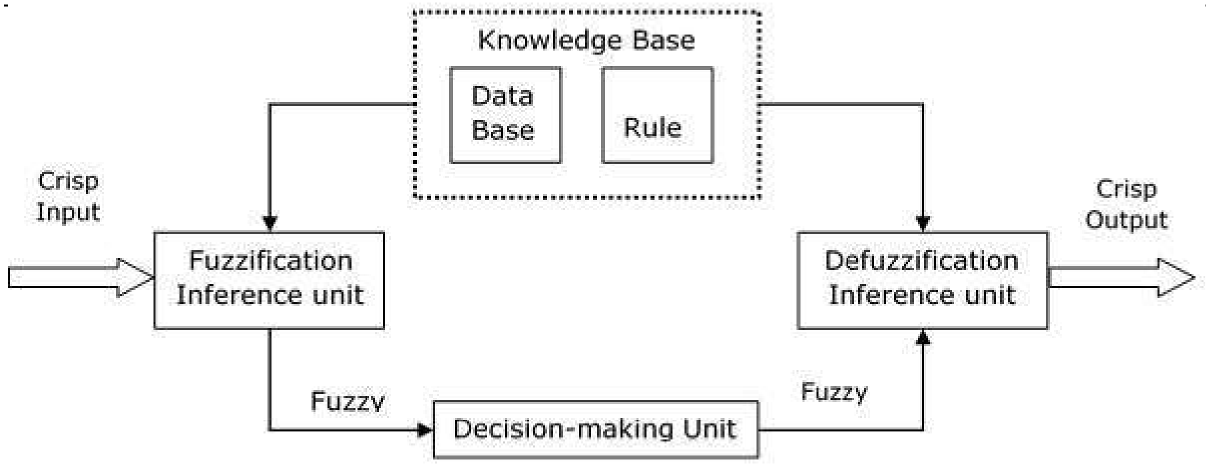
• **Database** − It defines the membership functions of fuzzy sets used in fuzzy rules.

• **Decision-making Unit** − It performs operation on rules.

Fuzzification Interface Unit − It converts the crisp quantities into fuzzy quantities.

• **Defuzzification Interface Unit** − It converts the fuzzy quantities into crisp quantities.

Following is a block diagram of fuzzy interference system.



Working of FIS

The working of the FIS consists of the following steps −

• A fuzzification unit supports the application of numerous fuzzification methods, and converts the crisp input into fuzzy input.

• A knowledge base - collection of rule base and database is formed upon the conversion of crisp input into fuzzy input.

• The defuzzification unit fuzzy input is finally converted into crisp output.

Methods of FIS

Let us now discuss the different methods of FIS. Following are the two important methods of FIS, having different consequent of fuzzy rules −

• Mamdani Fuzzy Inference System

• Takagi-Sugeno Fuzzy Model (TS Method)

Mamdani Fuzzy Inference System

This system was proposed in 1975 by Ebhasim Mamdani. Basically, it was anticipated to control a steam engine and boiler combination by synthesizing a set of fuzzy rules obtained from people working on the system.

Steps for Computing the Output

Following steps need to be followed to compute the output from this FIS −

• **Step 1** − Set of fuzzy rules need to be determined in this step.

• **Step 2** − In this step, by using input membership function, the input would be made fuzzy.

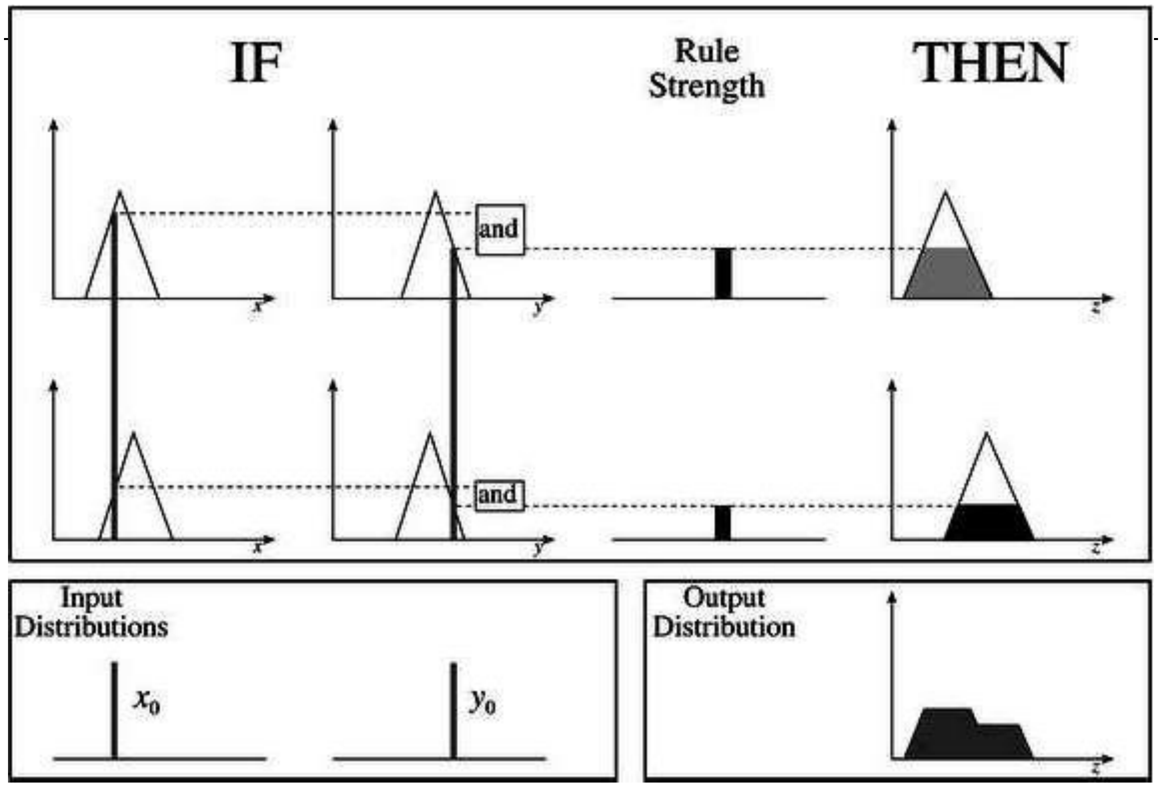
• **Step 3** − Now establish the rule strength by combining the fuzzified inputs according to fuzzy rules.

• **Step 4** − In this step, determine the consequent of rule by combining the rule strength and the output membership function.

• **Step 5** − For getting output distribution combine all the consequents.

• **Step 6** − Finally, a defuzzified output distribution is obtained.

Following is a block diagram of Mamdani Fuzzy Interface System.



Takagi-Sugeno Fuzzy Model (TS Method)

This model was proposed by Takagi, Sugeno and Kang in 1985. Format of this rule is given as −

IF x is A and y is B THEN Z = f(x,y)

Here, *AB* are fuzzy sets in antecedents and *z = f(x,y)* is a crisp function in the consequent.

Fuzzy Inference Process

The fuzzy inference process under Takagi-Sugeno Fuzzy Model (TS Method) works in the following way:

• **Step 1: Fuzzifying the inputs** − Here, the inputs of the system are made fuzzy.

• **Step 2: Applying the fuzzy operator** − In this step, the fuzzy operators must be applied to get the output.

Rule Format of the Sugeno Form

The rule format of Sugeno form is given by −

if 7 = x and 9 = y then output is z = ax+by+c

omparison between the two methods

Let us now understand the comparison between the Mamdani System and the Sugeno Model.

• **Output Membership Function** − The main difference between them is on the basis of output membership function. The Sugeno output membership functions are either linear or constant.

• **Aggregation and Defuzzification Procedure** − The difference between them also lies in the consequence of fuzzy rules and due to the same their aggregation and defuzzification procedure also differs.

• **Mathematical Rules** − More mathematical rules exist for the Sugeno rule than the Mamdani rule.

• **Adjustable Parameters** − The Sugeno controller has more adjustable parameters than the Mamdani controller.

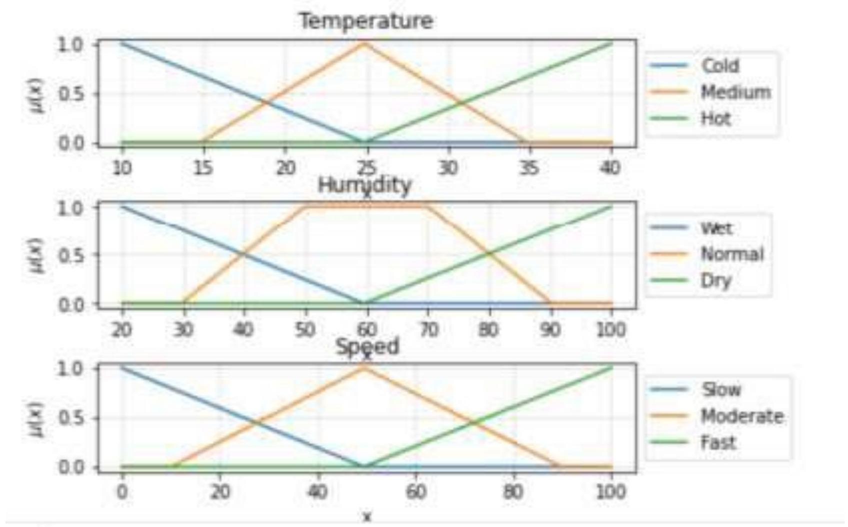
Procedure:

Python Code

```
import numpy as np
import copy
import matplotlib.pyplot as plt
from fuzzy_system.fuzzy_variable_output import FuzzyOutputVariable
from fuzzy_system.fuzzy_variable_input import FuzzyInputVariable
# from fuzzy_system.fuzzy_variable import FuzzyVariable
from fuzzy_system.fuzzy_system import FuzzySystem
temp = FuzzyInputVariable('Temperature', 10, 40, 100)
temp.add_triangular('Cold', 10, 10, 25)
temp.add_triangular('Medium', 15, 25, 35)
temp.add_triangular('Hot', 25, 40, 40)
humidity = FuzzyInputVariable('Humidity', 20, 100, 100)
humidity.add_triangular('Wet', 20, 20, 60)
humidity.add_trapezoidal('Normal', 30, 50, 70, 90)
humidity.add_triangular('Dry', 60, 100, 100)
motor_speed = FuzzyOutputVariable('Speed', 0, 100, 100)
motor_speed.add_triangular('Slow', 0, 0, 50)
motor_speed.add_triangular('Moderate', 10, 50, 90)
motor_speed.add_triangular('Fast', 50, 100, 100)
system = FuzzySystem()
system.add_input_variable(temp)
system.add_input_variable(humidity)
system.add_output_variable(motor_speed)
ystem.add_rule(
{ 'Temperature':'Cold',
'Humidity':'Wet' },
{ 'Speed':'Slow'})
system.add_rule(
{ 'Temperature':'Cold',
'Humidity':'Normal' },
{ 'Speed':'Slow'})
system.add_rule(
{ 'Temperature':'Medium',
'Humidity':'Wet' },
{ 'Speed':'Slow'})
system.add_rule(
{ 'Temperature':'Medium',
'Humidity':'Normal' },
{ 'Speed':'Moderate'})
system.add_rule(
{ 'Temperature':'Cold',
'Humidity':'Dry' },
{ 'Speed':'Moderate'})
system.add_rule(
{ 'Temperature':'Hot',
'Humidity':'Wet' },
{ 'Speed':'Moderate'})
system.add_rule(
{ 'Temperature':'Hot',
'Humidity':'Normal' },
{ 'Speed':'Fast'})
system.add_rule(
{ 'Temperature':'Hot',
'Humidity':'Dry' },
{ 'Speed':'Fast'})
system.add_rule(
{ 'Temperature':'Medium',
'Humidity':'Dry' },
{ 'Speed':'Fast'})
output = system.evaluate_output({
'Temperature':18,
'Humidity':60
})
print(output)
# print('fuzzification\n-------------\n', info['fuzzification'])
# print('rules\n-----\n', info['rules'])
system.plot_system()
from fuzzy_system.fuzzy_variable_output import FuzzyOutputVariable
from fuzzy_system.fuzzy_variable_input import FuzzyInputVariable
from fuzzy_system.fuzzy_system import FuzzySystem
x1 = FuzzyInputVariable('x1', 0, 100, 100)
x1.add_triangular('S', 0, 25, 50)
x1.add_triangular('M', 25, 50, 75)
x1.add_triangular('L', 50, 75, 100)
x2 = FuzzyInputVariable('x2', 0, 100, 100)
x2.add_triangular('S', 0, 25, 50)
x2.add_triangular('M', 25, 50, 75)
x2.add_triangular('L', 50, 75, 100)
y = FuzzyOutputVariable('y', 0, 100, 100)
y.add_triangular('S', 0, 25, 50)
y.add_triangular('M', 25, 50, 75)
y.add_triangular('L', 50, 75, 100)
z = FuzzyOutputVariable('z', 0, 100, 100)
z.add_triangular('S', 0, 25, 50)
z.add_triangular('M', 25, 50, 75)
z.add_triangular('L', 50, 75, 100)
system = FuzzySystem()
system.add_input_variable(x1)
system.add_input_variable(x2)
system.add_output_variable(y)
system.add_output_variable(z)
system.add_rule(
{ 'x1':'S',
'x2':'S' },
{ 'y':'S',
'z':'L' })
system.add_rule(
{ 'x1':'M',
'x2':'M' },
{ 'y':'M',
'z':'M' })
system.add_rule(
{ 'x1':'L',
'x2':'L' },
{ 'y':'L',
'z':'S' })
system.add_rule(
{ 'x1':'S',
'x2':'M' },
{ 'y':'S',
'z':'L' })
system.add_rule(
{ 'x1':'M',
'x2':'S' },
{ 'y':'S',
'z':'L' })
system.add_rule(
{ 'x1':'L',
'x2':'M' },
{ 'y':'L',
'z':'S' })
system.add_rule(
{ 'x1':'M',
'x2':'L' },
{ 'y':'L',
'z':'S' })
system.add_rule(
{ 'x1':'L',
'x2':'S' },
{ 'y':'M',
'z':'M' })
system.add_rule(
{ 'x1':'S',
'x2':'L' },
{ 'y':'M',
'z':'M' })
output = system.evaluate_output({
'x1':35,
'x2':75
})
print(output)
```

Output Screen

```
{'Speed': 37.24647662394699}
```



Video Tutorial
https://www.youtube.com/watch?v=5UX1w16VQCg

Papers:
Hussain, S., Kim, Y. S., Thakur, S., & Breslin, J. G. (2022). Optimization of waiting time for electric vehicles using a fuzzy inference system. IEEE Transactions on Intelligent Transportation Systems.

Prospective Viva Questions

1. What do you mean by fuzzy inference system?
2. What are the two types of fuzzy inference systems ?
3. What are the main steps in fuzzy inference system?
4. What are the application of fuzzy inference system?

EXPERIMENT – 3.4

Mapped Course Outcome

CO3	Apply fuzzy logic and reasoning to handle uncertainty and solve engineering problems, genetic algorithms to combinatorial optimization problems and neural networks to pattern classification and regression problems.	**3**
CO4	Effectively use modern software tools to solve real problems using a soft computing approach.	**3**
CO5	Evaluate various soft computing approaches for a given problem.	**4**

AIM: Write a program to implement GANs using genetic algorithms

Theory

What are genetic algorithms? Genetic Algorithms are a type of learning algorithm, that uses the idea that crossing over the weights of two good neural networks, would result in a better neural network. The reason that genetic algorithms are so effective is that there is no direct optimization algorithm, allowing for the possibility to have extremely varied results. Additionally, they often come up with very interesting solutions that often give valuable insight into the problem. How do they work? A set of random weights are generated. This is the neural network of the first agent. A set of tests are performed on the agent. The agent receives a score based on the tests. Repeat this several times to create a population. Select the top 10% of the population to be available to crossover. Two random parents are chosen from the top 10% and their weights are crossover. Every time crossover occurs, there is a small chance of mutation: That is a random value that is in neither of the parent's weights. This process slowly optimizes the agent's performance, as the agents slowly adapt to the environment.

Procedure:

Python Code

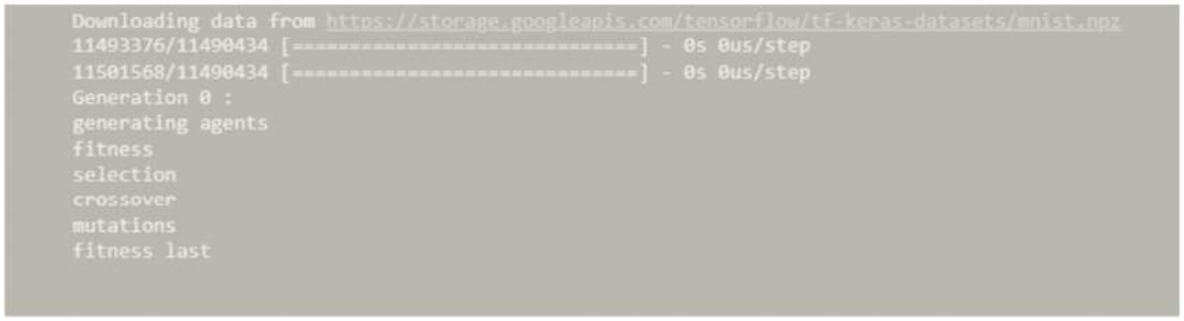
```
import random
import numpy as np
from IPython.display import clear_output
from keras.layers import Reshape
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import Conv2DTranspose
from keras.layers import LeakyReLU
from keras.layers import Dropout,Dense
from tensorflow.keras.optimizers import Adam
from keras.models import Sequential
from keras.datasets.mnist import load_data
from math import exp
from tensorflow.math import sigmoid
(trainX, trainy), (testX, testy) = load_data()
def sigmoid(x):
return 1/(1 + np.exp(-x))
class genetic_algorithm:
def execute(pop_size,generations,threshold,network):
class Agent:
def __init__(self,network):
class neural_network:
def __init__(self,network):
self.weights = []
self.activations = []
for layer in network:
if layer[0] != None:
input_size = layer[0]
else:
```

```
input_size = network[network.index(layer)-1][1]
output_size = layer[1]
activation = layer[2]
self.weights.append(np.random.randn(input_size,output_size))
self.activations.append(activation)
def propagate(self,data):
input_data = data
for i in range(len(self.weights)):
z = np.dot(input_data,self.weights[i])
a = self.activations[i](z)
input_data = a
yhat = a
return yhat
self.neural_network = neural_network(network)
self.fitness = 0
# def generate_agents(population, network):
# return [Agent(network) for _ in range(population)]
def generate_agents(population, network):
print(Agent(network))
return [Agent(network) for _ in range(population)]
def fitness(agents):
for agent in agents:
dataset_len = 100
fake = []
real = []
y = []
for i in range(dataset_len//2):
fake.append(agent.neural_network.propagate(np.random.randn(latent_size)).reshape(28,28))
y.append(0)
real.append(random.choice(trainX))
y.append(1)
X = fake+real
X = np.array(X).astype('uint8').reshape(len(X),28,28,1)
y = np.array(y).astype('uint8')
model.fit(X,y,verbose = 0)
fake = []
real = []
y = []
for i in range(dataset_len//2):
fake.append(agent.neural_network.propagate(np.random.randn(latent_size)).reshape(28,28))
y.append(0)
real.append(random.choice(trainX))
y.append(1)
X = fake+real
```

```
X = np.array(X).astype('uint8').reshape(len(X),28,28,1)
y = np.array(y).astype('uint8')
agent.fitness = model.evaluate(X,y,verbose = 0)[1]*100
return agents
def selection(agents):
agents = sorted(agents, key=lambda agent: agent.fitness, reverse=False)
print('\n'.join(map(str, agents)))
agents = agents[:int(0.2 * len(agents))]
return agents
def unflatten(flattened,shapes):
newarray = []
index = 0
for shape in shapes:
size = np.product(shape)
newarray.append(flattened[index : index + size].reshape(shape))
index += size
return newarray
def crossover(agents,network,pop_size):
offspring = []
for _ in range((pop_size - len(agents)) // 2):
parent1 = random.choice(agents)
parent2 = random.choice(agents)
child1 = Agent(network)
child2 = Agent(network)
shapes = [a.shape for a in parent1.neural_network.weights]
genes1 = np.concatenate([a.flatten() for a in parent1.neural_network.weights])
genes2 = np.concatenate([a.flatten() for a in parent2.neural_network.weights])
split = random.randint(0,len(genes1)-1)
child1_genes = np.array(genes1[0:split].tolist() + genes2[split:].tolist())
child2_genes = np.array(genes1[0:split].tolist() + genes2[split:].tolist())
child1.neural_network.weights = unflatten(child1_genes,shapes)
child2.neural_network.weights = unflatten(child2_genes,shapes)
offspring.append(child1)
offspring.append(child2)
agents.extend(offspring)
return agents
def mutation(agents):
for agent in agents:
if random.uniform(0.0, 1.0) <= 0.1:
weights = agent.neural_network.weights
shapes = [a.shape for a in weights]
flattened = np.concatenate([a.flatten() for a in weights])
randint = random.randint(0,len(flattened)-1)
flattened[randint] = np.random.randn()
```

```
newarray = []
indeweights = 0
for shape in shapes:
size = np.product(shape)
newarray.append(flattened[indeweights : indeweights + size].reshape(shape))
indeweights += size
agent.neural_network.weights = newarray
return agents
for i in range(generations):
print('Generation',str(i),':')
agents = generate_agents(pop_size,network)
print(agents)
agents = fitness(agents)
agents = selection(agents)
agents = crossover(agents,network,pop_size)
agents = mutation(agents)
agents = fitness(agents)
if any(agent.fitness < threshold for agent in agents):
print('Threshold met at generation '+str(i)+' !')
if i % 100:
clear_output()
return agents[0]
image_size = 28
latent_size = 100
model = Sequential()
model.add(Conv2D(64, (3,3), strides=(2, 2), padding='same',
input_shape=(image_size,image_size,1)))
model.add(LeakyReLU(alpha=0.2))
model.add(Dropout(0.4))
model.add(Conv2D(64, (3,3), strides=(2, 2), padding='same'))
model.add(LeakyReLU(alpha=0.2))
model.add(Dropout(0.4))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
opt = Adam(learning_rate=0.0002, beta_1=0.5)
model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
network = [[latent_size,100,sigmoid],[None,image_size**2,sigmoid]]
ga = genetic_algorithm
agent = ga.execute(1000,10,90,network)
print(type(agent))
(trainX, trainy), (testX, testy) = load_data()
weights = agent.neural_network.weights
```

Output Screen


```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
Generation 0 :
generating agents
fitness
selection
crossover
mutations
fitness last
```

Video Tutorial
https://www.youtube.com/watch?v=FKbarpAlBkw

Further Reading
Mirjalili, S. (2019). Genetic algorithm. In *Evolutionary algorithms and neural networks* (pp. 43- 55). Springer, Cham.

Prospective Viva Questions
1. Is genetic algorithm part of machine learning?
2. Is genetic algorithm neural network?
3. What is genetic programming in machine learning?
4. How does a genetic algorithm work?
5. What are genetic algorithms in AI?
6. Is genetic algorithm a form of reinforcement learning?
7. Is evolutionary algorithm supervised learning?
8. Where is genetic programming used?
9. Is genetic algorithm the best?