SOFT COMPUTING LAB

ETCS-456

Faculty name: Dr. Sandeep Tayal

Student name: Syeda Reeha Quasar

Roll No.: 14114802719

Semester: 8th

Group: 8C7

Maharaja Agrasen Institute of Technology, PSP Area,

Sector – 22, Rohini, New Delhi – 110085

Index

Exp No	Experiment Name	Date of performance	Date of checking	Marks	Sign.
1.	Implementation of Fuzzy Operations.				
2.	Implementation of Fuzzy Relations (Maxmin Composition).				
3.	Implementation of Fuzzy Controller (Washing Machine).				
4.	Implementation of Simple Neural Network (McCullohPitts model).				
5.	Implementation of Perceptron Learning Algorithm.				
6.	Implementation of Unsupervised Learning Algorithm.				
7.	Implementation of Simple Genetic Application.				
8.	Study of ANFIS Architecture.				
	Beyond the Syllabus Questions				
1.	Study of Derivative-free Optimization				
2.	Study of research paper on Soft Computing				

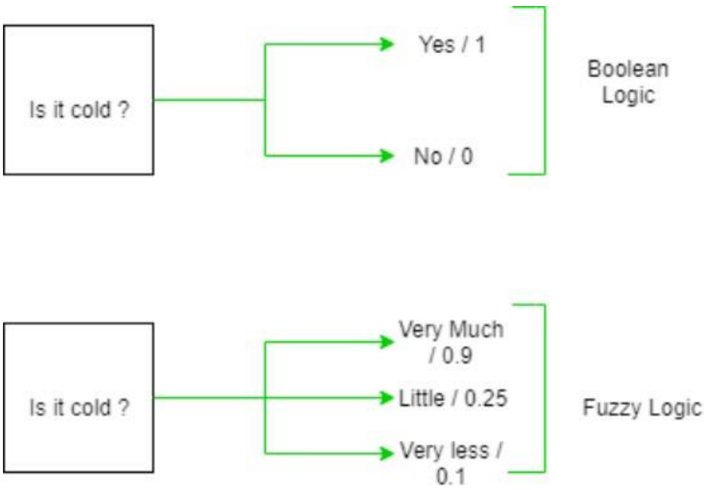
Experiment 1

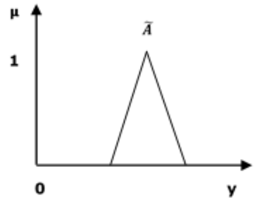
Aim : Implementation of Fuzzy Operations

Theory :

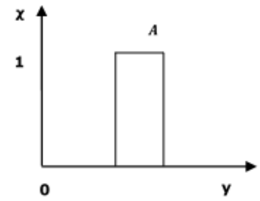
<u>Fuzzy Logic:</u>

The term fuzzy refers to things which are not clear or are vague. In the real world many times we encounter a situation when we can’t determine whether the state is true or false, their fuzzy logic provides a very valuable flexibility for reasoning. In this way, we can consider the inaccuracies and uncertainties of any situation.
In boolean system truth value, 1.0 represents absolute truth value and 0.0 represents absolute false value. But in the fuzzy system, there is no logic for absolute truth and absolute false value. But in fuzzy logic, there is intermediate value too present which is partially true and partially false.





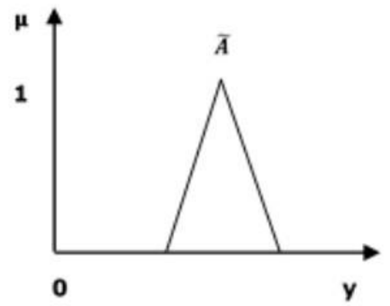
Membership Function of Fuzzy set Ã



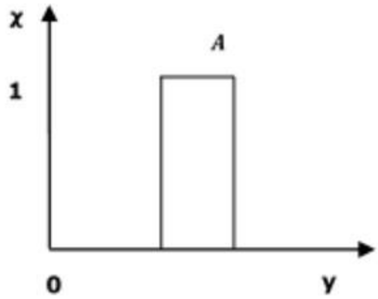
Membership Function of classical set A

<u>Fuzzy Sets:</u>

Fuzzy sets can be considered as an extension and gross oversimplification of classical sets. It can be best understood in the context of set membership. Basically it allows partial membership which means that it contain elements that have varying degrees of membership in the set. From this, we can understand the difference between classical set and fuzzy set. Classical set contains elements that satisfy precise properties of membership while fuzzy set contains elements that satisfy imprecise properties of membership.



Membership Function of Fuzzy set Ã



Membership Function of classical set A

Mathematical Concept:

A fuzzy set Ã in the universe of information U can be defined as a set of ordered pairs and it can be represented mathematically as:

$\tilde{A} = \{ (y , \mu\tilde{A} (y)) \mid y \, \varepsilon \, U \}$

Here $\mu\tilde{A} (y)$ = degree of membership of y in \widetilde{A}, assumes values in the range from 0 to 1, i.e., $\mu\tilde{A} (y) \, \varepsilon \, [0,1]$.

Operations on Fuzzy Sets:

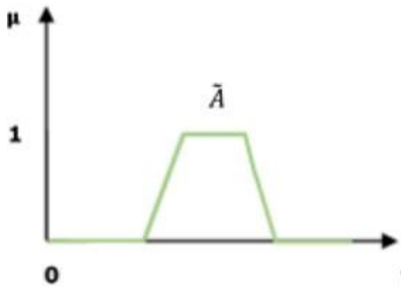
Having two fuzzy sets A˜ and B˜, the universe of information U and an element 8 of the universe, the following relations express the union, intersection and complement operation on fuzzy sets.

Union/Fuzzy 'OR'

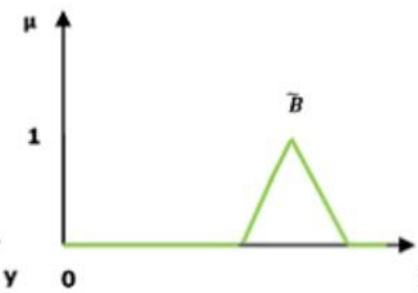
Let us consider the following representation to understand how the Union/Fuzzy 'OR' relation works −

$\mu\tilde{A} \cup \tilde{B} (y) = \mu\tilde{A} \vee \mu\tilde{B} \;\; \forall \; y \, \varepsilon \, U$

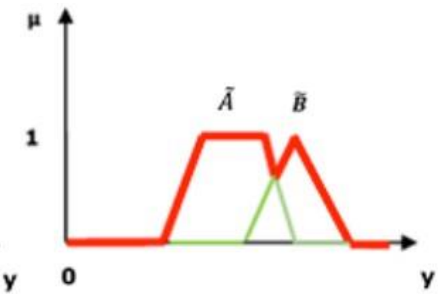
Here ∨ represents the 'max' operation.



Fuzzy set Ã



Fuzzy set B̃



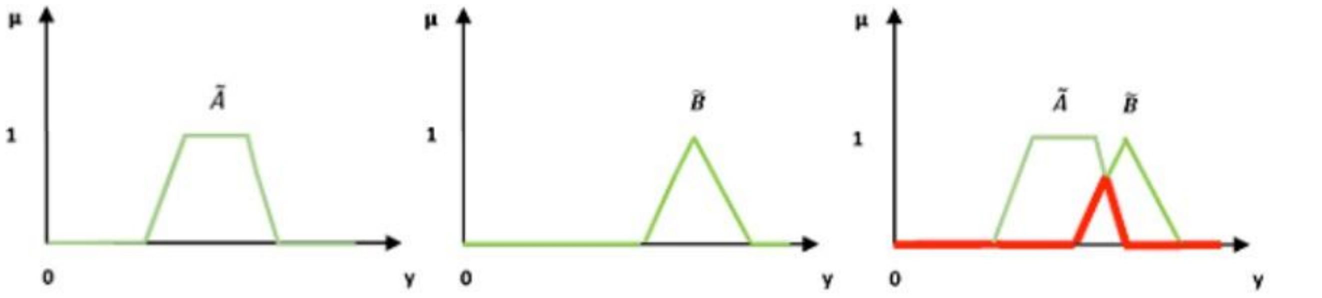
Union of two Fuzzy sets

Intersection/Fuzzy ‘AND’

Let us consider the following representation to understand how the Intersection/Fuzzy ‘AND’ relation works −

$$\mu\tilde{A} \cup \tilde{B}\ (y) = \mu\tilde{A} \wedge \mu\tilde{B} \quad \forall\ y\ \varepsilon\ U$$

Here $\wedge$ represents the ‘min’ operation.

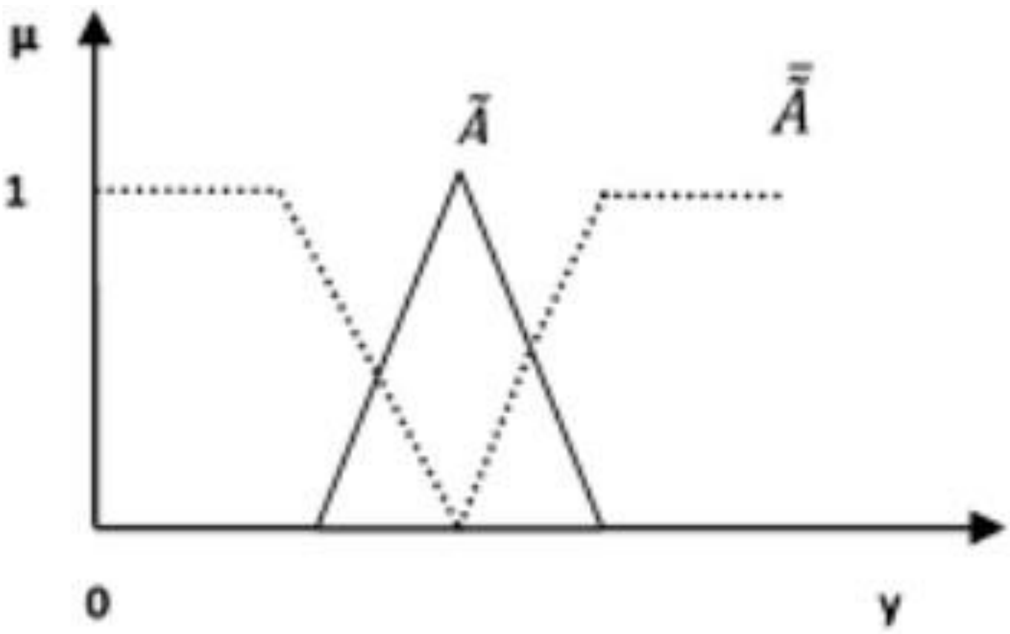


Fuzzy set $\tilde{A}$ Fuzzy set $\tilde{B}$ Intersection of two Fuzzy sets

Complement/Fuzzy ‘NOT’

Let us consider the following representation to understand how the complement/Fuzzy ‘NOT’ relation works −

$$\mu\tilde{A} = 1- \mu\tilde{A}\ (y) \quad \forall\ y\ \varepsilon\ U$$

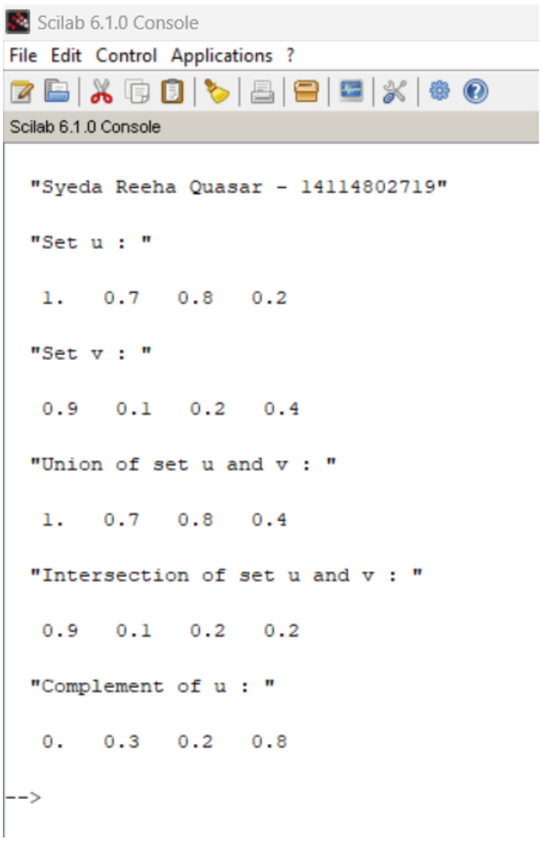


Complement of a fuzzy set

Code :

```
clc;
disp('Syeda Reeha Quasar - 14114802719')

u=[1 0.7 0.8 0.2]
v=[0.9 0.1 0.2 0.4]

disp('Set u : ')
disp(u)

disp('Set v : ')
disp(v)

disp('Union of set u and v : ')
w=max(u,v)
disp(w)

disp('Intersection of set u and v : ')
p=min(u,v)
disp(p)

disp('Complement of u : ');
m=length(u);
q1=ones(m)-u
disp(q1)
```

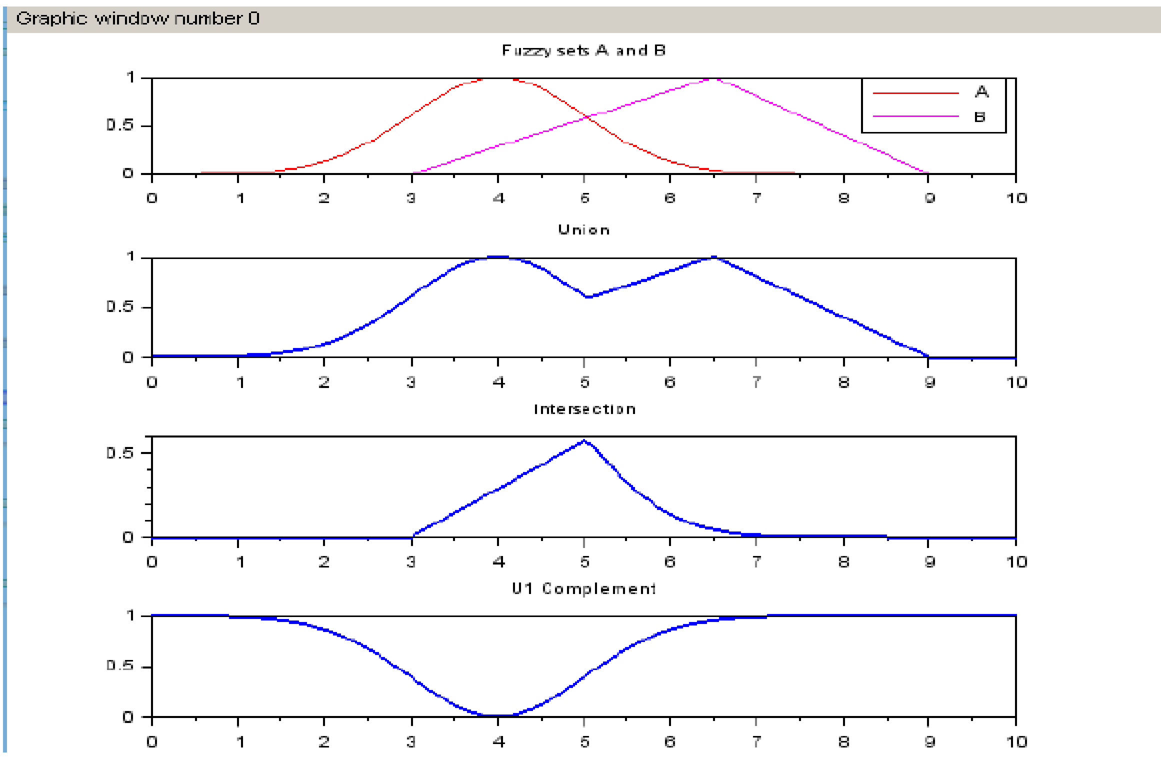
Output :



Code :

```
clear;
clc;
x = (0:0.1:10)'; // The universe of discourse is [0,10]; the points are defined with a step of
0.1
u1 = gaussmf(x,[1,4]); // first membership function, gaussian type
u2 = trimf(x,[3 6.5 9]); // second membership function, triangular type
u_union = max(u1,u2); // compute the membership degrees for the union using the "MAX"
operator
u_intersect = min(u1,u2);
u1_fnot = 1-u1;
set(gca(),"auto_clear","off");
subplot(4,1,1); // breaks the figure in four windows
plot(x,u1,'r');
set(gca(),"auto_clear","off");
plot (x,u2,'m');
set(gca(),"auto_clear","off");
legend('A','B');
title('Fuzzy sets A and B');
subplot(4,1,2); // the current plot appears in the second window
plot(x, u_union,'color','b','linewidth',2);
title('Union');
subplot(4,1,3); // the current plot appears in the third window
plot(x, u_intersect,'color','b','linewidth',2);
title('Intersection');
subplot(4,1,4); // the current plot appears in the fourth window
plot(x, u1_fnot,'color','b','linewidth',2);
title('U1 Complement');
```

Output :



:

Viva Questions:

1. What are the properties of the Fuzzy set?
Commutativity
Associativity
Distributivity
Idempotency
Identity
Transitivity

2. What is De Morgan's Law in a Crisp Set?
For any two finite sets A and B;
- (A U B)' = A' ∩ B' (which is a De Morgan's law of union).
- (A ∩ B)' = A' U B' (which is a De Morgan's law of intersection).

3. What is the difference between the crisp set and fuzzy set?

FUZZY SET	CRISP SET
Prescribed by vague or ambiguous properties.	Defined by precise and certain characteristics.
Elements are allowed to be partially included in the set.	Element is either the member of a set or not
Used in fuzzy controllers	Digital design
Infinite-valued	bi-valued

4. List the different fuzzy set operations?
Union, intersection, complement, scalar product, vector product, Cartesian product and power.

5. What is fuzzy logic?
Fuzzy logic is a form of many-valued logic in which the truth values of variables may be any real number between 0 and 1 both inclusive. It is employed to handle the concept of partial truth, where the truth value may range between completely true and completely false

Experiment 2

Aim : Implementation of fuzzy relations (Max-Min Composition)

Theory :

Max-Min Composition of fuzzy Relations:
Fuzzy relation in different product space can be combined with each other by the operation called ―Composition‖. There are many composition methods in use ,

e.g. max product method, max-average method and max-min method. But max-min composition method is best known in fuzzy logic applications.

Crisp relation

Crisp relation is defined on the Cartesian product of two sets. Consider,

$$X \times Y = \{(x,y) | x \in X, y \in Y\}$$

The relation on this Cartesian product will be,

$$\mu_R = \begin{cases} 1, (x,y) \in R \\ 0, (x,y) \notin R \end{cases}$$

Example: Let X={1,4,5} and Y={3,6,7} then for relation R=x<y,

$$R = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

Fuzzy relation

Let $X, Y \subseteq R$ be universal sets then,

$$R = \{((x,y), \mu_R(x,y)) \mid (x,y) \in X \times Y\}$$

Is called a fuzzy relation in $X \times Y \subseteq R$

Example: Let X={1,2,3} and Y={1,2}

If $\mu_R(x,y) = e^{-(x-y)^2}$, then

$$R = \left\{ \frac{e^{-(1-1)^2}}{(1,1)}, \frac{e^{-(1-2)^2}}{(1,2)}, \frac{e^{-(2-1)^2}}{(2,1)}, \frac{e^{-(2-2)^2}}{(2,2)}, \frac{e^{-(3-1)^2}}{(3,1)}, \frac{e^{-(3-2)^2}}{(3,2)} \right\}$$

$$R = \begin{bmatrix} 1 & 0.37 \\ 0.37 & 1 \\ 0.02 & 0.37 \end{bmatrix}$$

Max-Min Composition

Let X, Y and Z be universal sets and let R and Q be relations that relate them as,

$$R = \{ (x,y) | x \in X, y \in Y, R \subset X \times Y \}$$
$$Q = \{ (y,z) | y \in Y, z \in Z, Q \subset Y \times Z \}$$

Then S will be a relation that relates elements of X with elements of Z as,

$$S = R \circ Q$$
$$S = \{ (x,z) | x \in X, z \in Z, S \subset X \times Z$$

Max min composition is then defined as,

$$\mu_S(x,z) = \max\left(\min\left(\mu_R(x,y), \mu_Q(y,z)\right)\right)$$

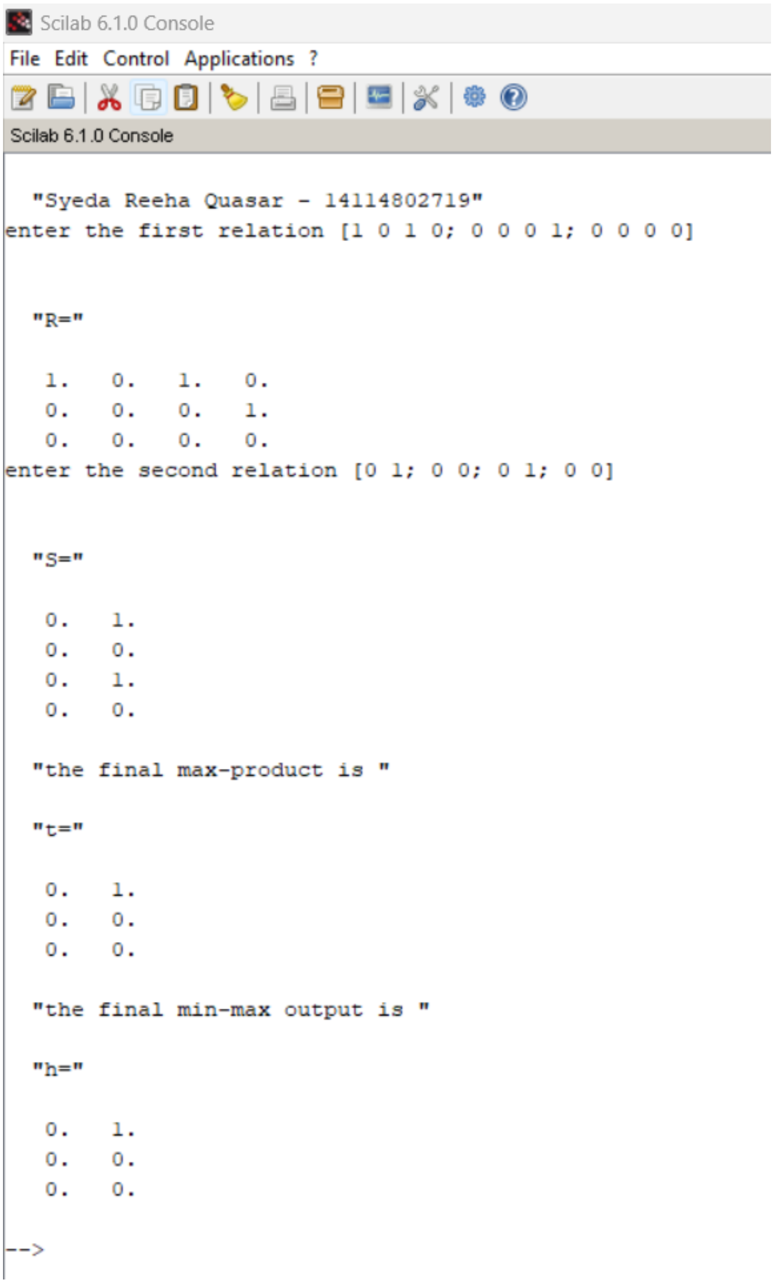
Example: $R = \begin{bmatrix} 0.6 & 0.5 & 0.4 \\ 0.2 & 0.1 & 0.2 \end{bmatrix}$ and $Q = \begin{bmatrix} 0.2 & 0.6 \\ 0.1 & 0.3 \\ 0.7 & 0.5 \end{bmatrix}$ $S = \begin{bmatrix} 0.4 & 0.6 \\ 0.2 & 0.2 \end{bmatrix}$

Code :

```
clear; clc;
disp('Syeda Reeha Quasar - 14114802719')

R=input("enter the first relation ");
disp("R=",R);
S=input("enter the second relation ");
disp("S=",S);

[m,n]=size(R);
[a,b]=size(S);

if(n==a)
   for i=1:m
      for j=1:b
         c=R(i,:);
         d=S(:,j);
         [f,g]=size(c);
         [h,q]=size(d); for l=1:g
         e(1,l)=c(1,l)*d(l,1); end
      t(i,j)=max(e); end
   end
   disp("the final max-product is ")
   disp("t=",t);
else
   disp("cannot find max-product");
end
if(n==a)
```

```
    for i=1:m
        for j=1:b
            c=R(i,:);
            d=S(:,j);
            f=mtlb_t(d);
            e=min(c,f);
            h(i,j)=max(e);
        end
    end
    disp("the final min-max output is ")
    disp("h=",h);
else
    disp("cannot find min-max");
end
```

Output :



Viva Questions:

1. What is the main difference between probability and fuzzy logic?
Fuzzy Logic is all about the degree of truth. Probability theory has nothing to reason about things that aren't entirely true or false. In short, we can say that Fuzzy Logic captures the meaning of partial truth whereas Probability theory captures partial knowledge.

2. What are the types of fuzzy logic sets?
L-fuzzy sets
Neutrosophic fuzzy sets
Pythagorean fuzzy sets

3. Who is the founder of fuzzy logic?
Lotfi Zadeh

4. What is fuzzy arithmetic?
Fuzzy arithmetic or arithmetic of fuzzy numbers is generalization of interval arithmetic, where rather than considering intervals at one constant level only, several levels are considered in [0, 1].

Experiment 3

Aim : Implementation of fuzzy controller (Washing Machine)

Theory :

Washing Machine Controller:

To design a system using fuzzy logic, input & output is necessary part of the system. Main function of the washing machine is to clean cloth without damaging the cloth. In order to achieve it, the output parameters of fuzzy logic, which are the washing parameters, must be given more importance.

The identified input & output parameters are:

Input: 1. Type of cloth 2. Type of dirt 3. Degree of dirt

Output: Wash time

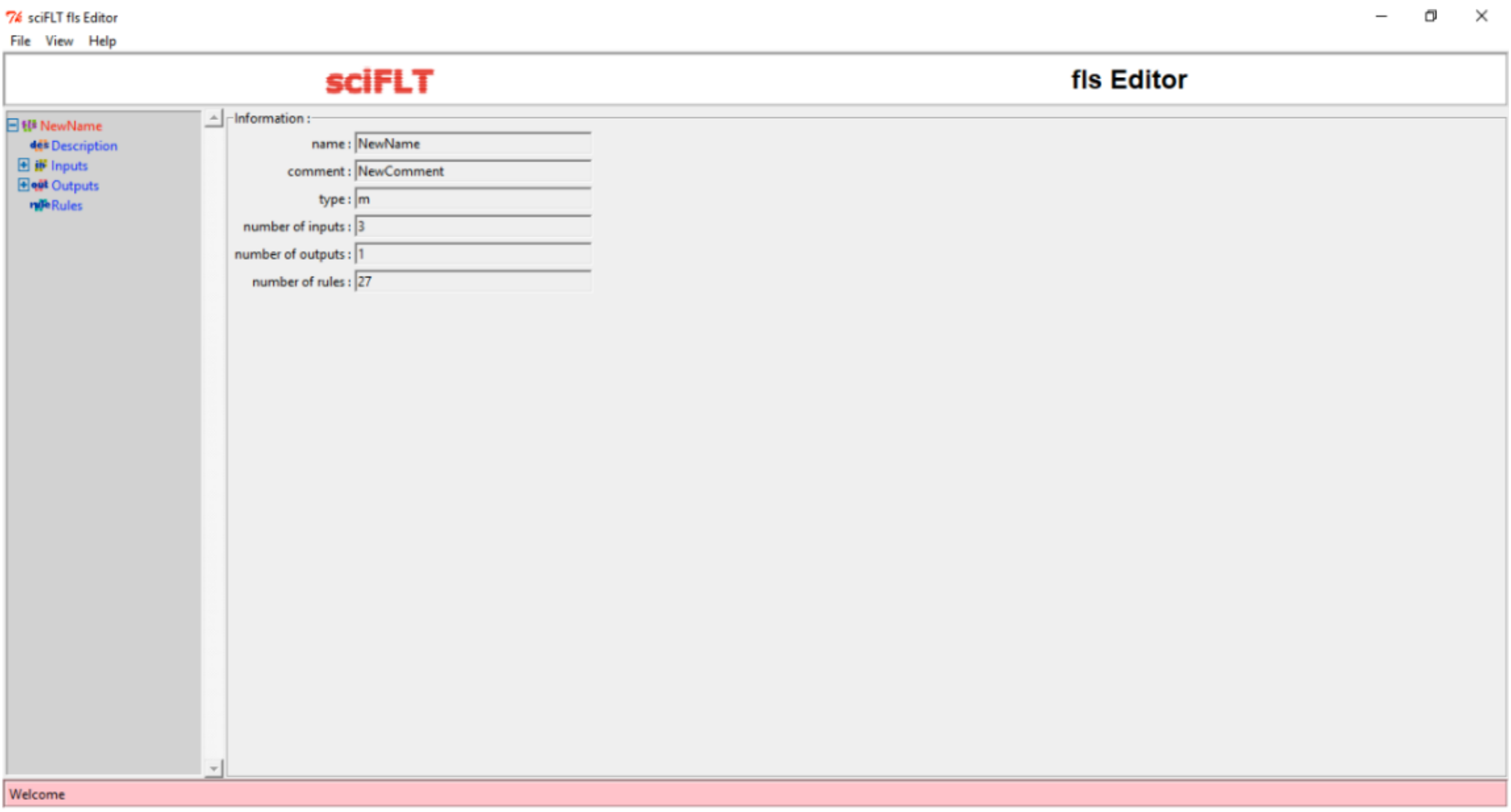
Rules:

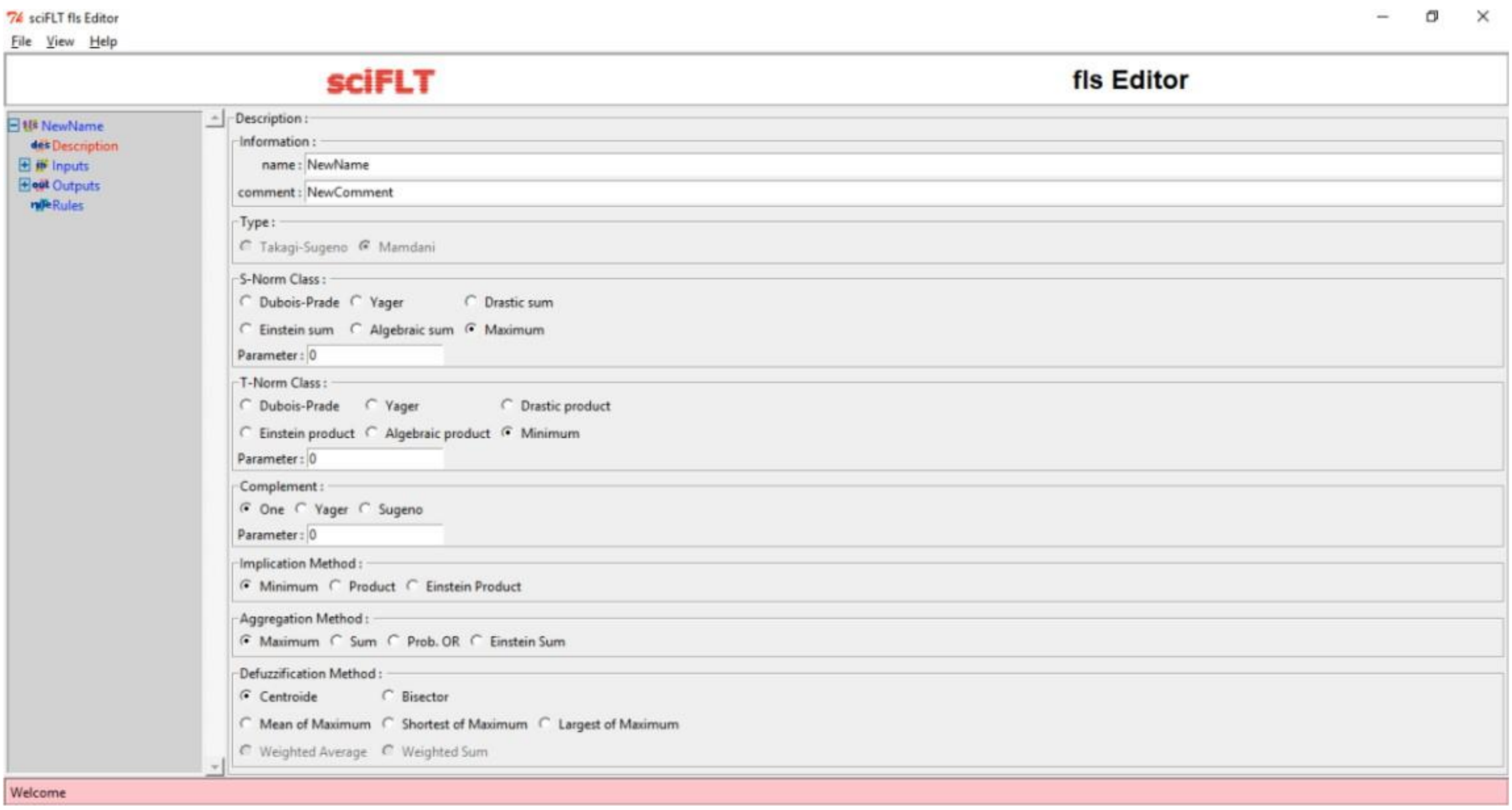
Type of Cloth	Type of Dirt	Degree of Dirt	Washing Time
Silk	Non greasy	Small	Very Short
Silk	Non greasy	Medium	Short
Silk	Non greasy	Large	Medium
Silk	Medium	Small	Medium
Silk	Medium	Medium	Long
Silk	Medium	Large	Long
Silk	Greasy	Small	Medium
Silk	Greasy	Medium	Long
Silk	Greasy	Large	Very Long
Woollen	Non greasy	Small	Short
Woollen	Non greasy	Medium	Medium
Woollen	Non greasy	Large	Long
Woollen	Medium	Small	Medium
Woollen	Medium	Medium	Medium
Woollen	Medium	Large	Long
Woollen	Greasy	Small	Long
Woollen	Greasy	Medium	Long
Woollen	Greasy	Large	Very Long
Cotton	Non greasy	Small	Short
Cotton	Non greasy	Medium	Medium
Cotton	Non greasy	Large	Long
Cotton	Medium	Small	Medium
Cotton	Medium	Medium	Long
Cotton	Medium	Large	Very Long
Cotton	Greasy	Small	Long
Cotton	Greasy	Medium	Long
Cotton	Greasy	Large	Very Long

Execution:

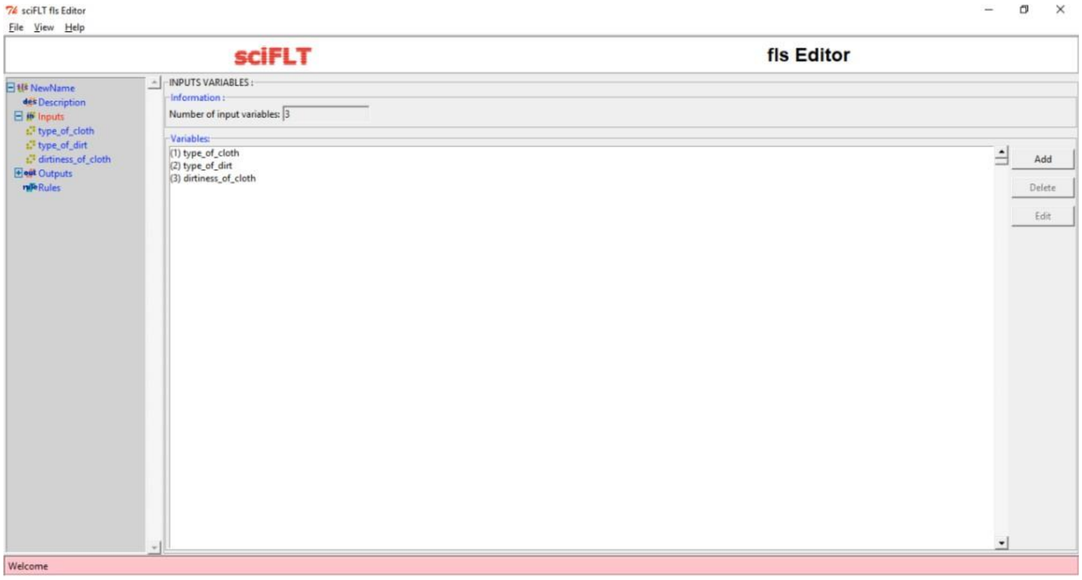
1. Run sciFLT Editor using sciFLTEditor() command.

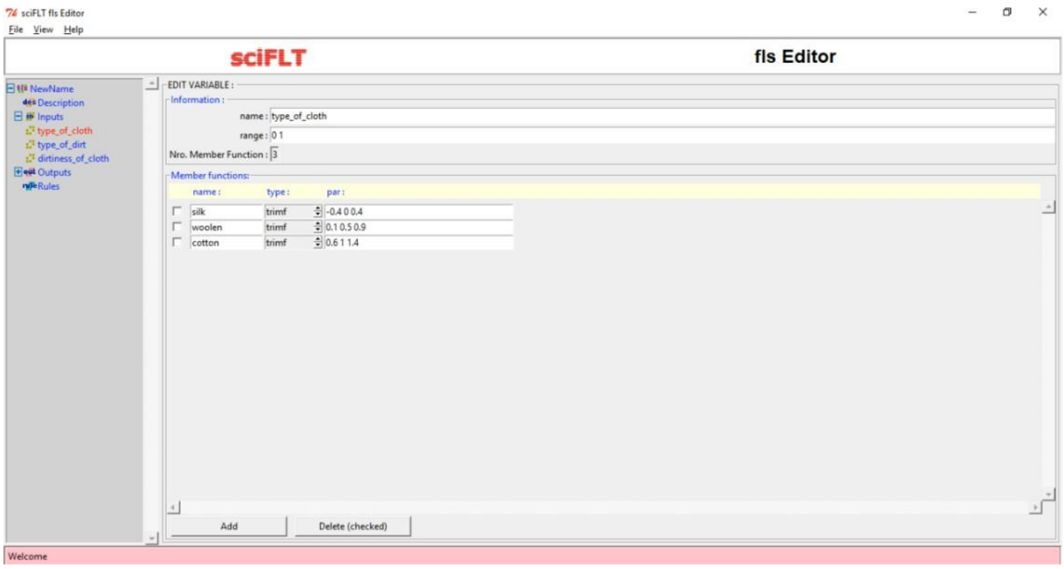
2. Create New File.

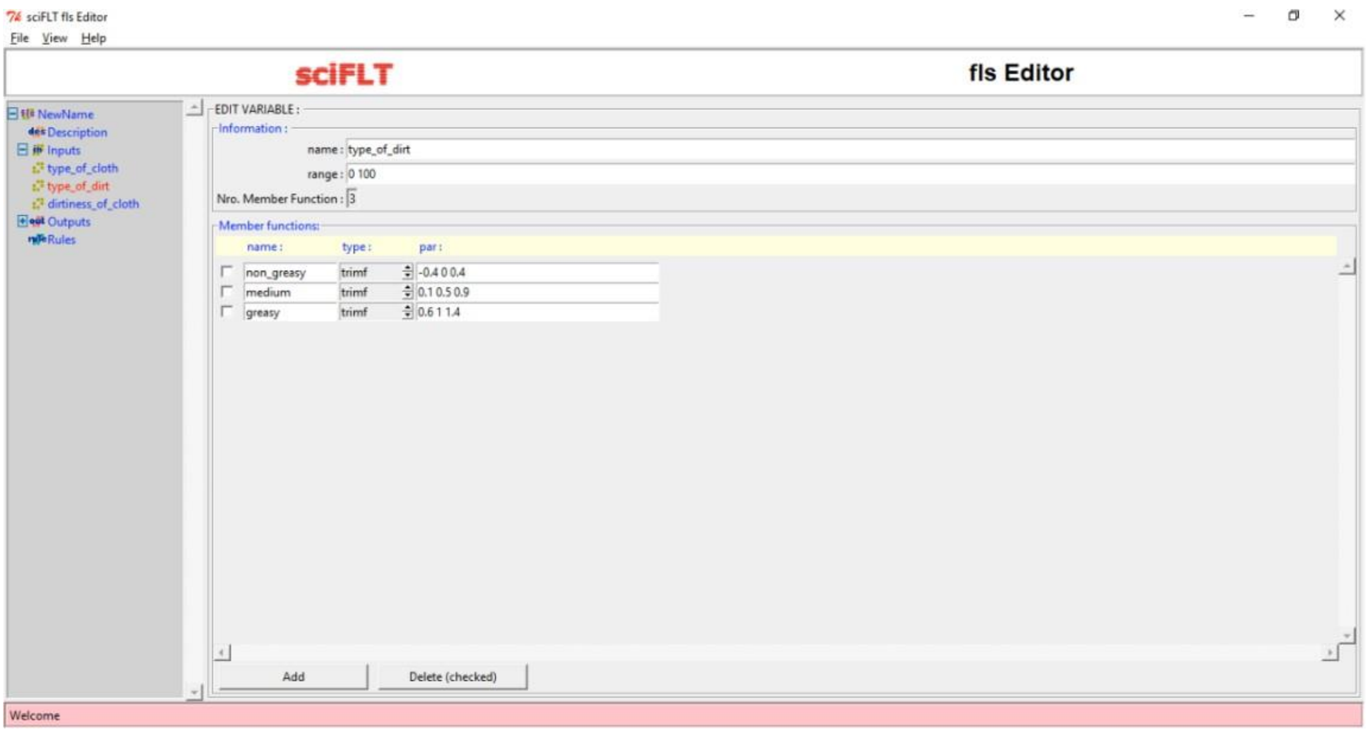


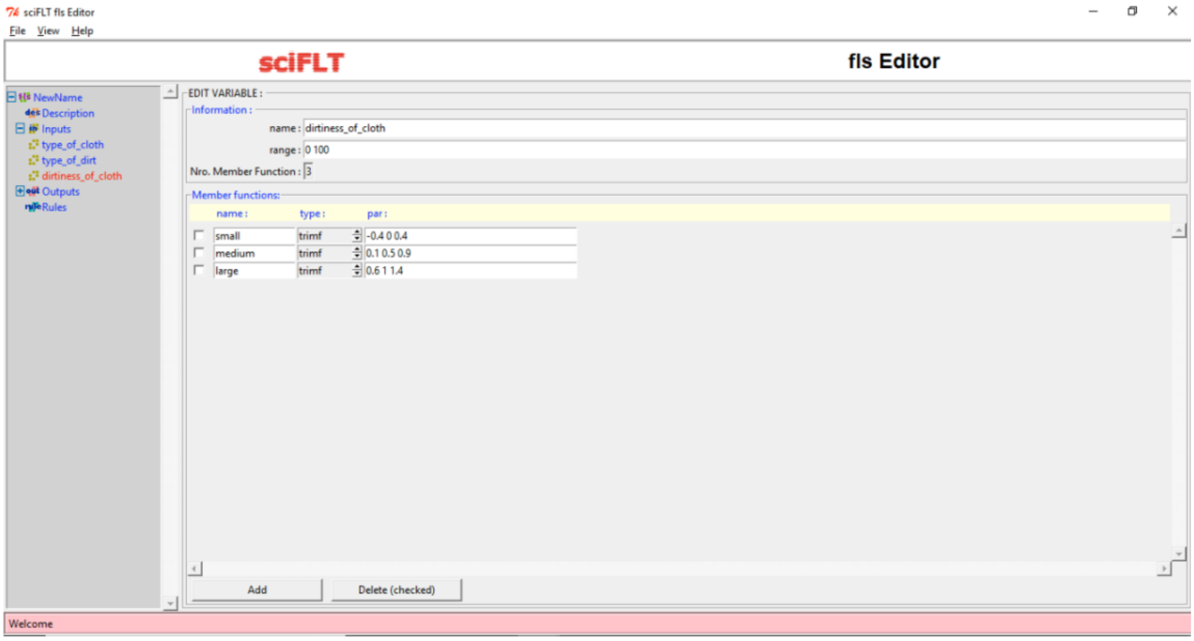


3. Design the input variables.

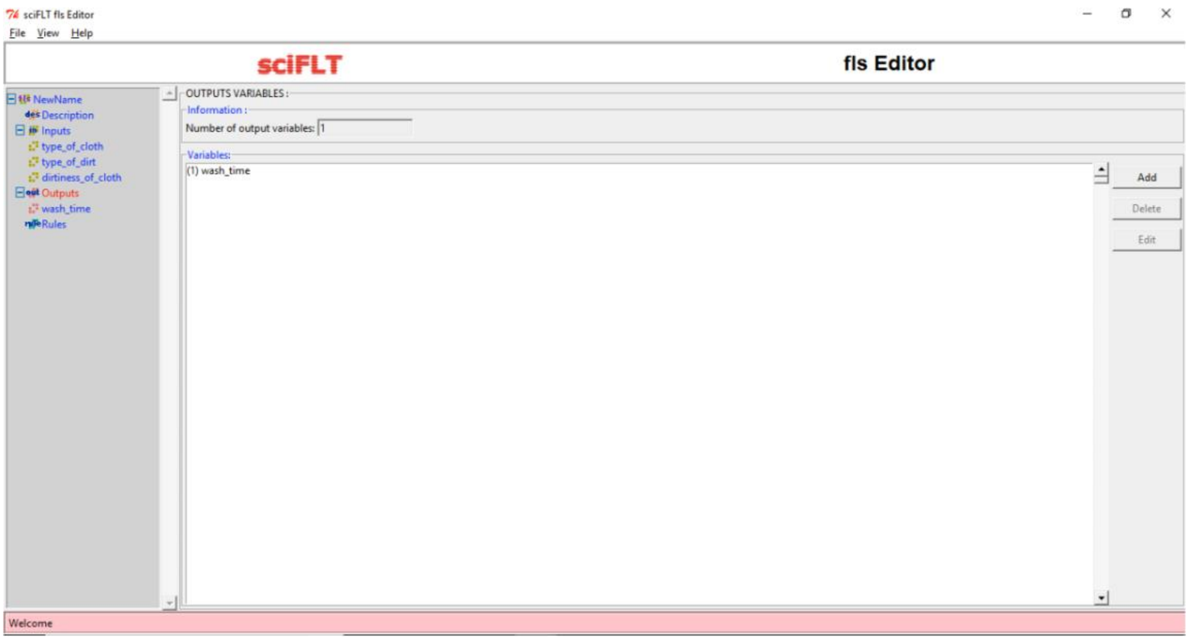


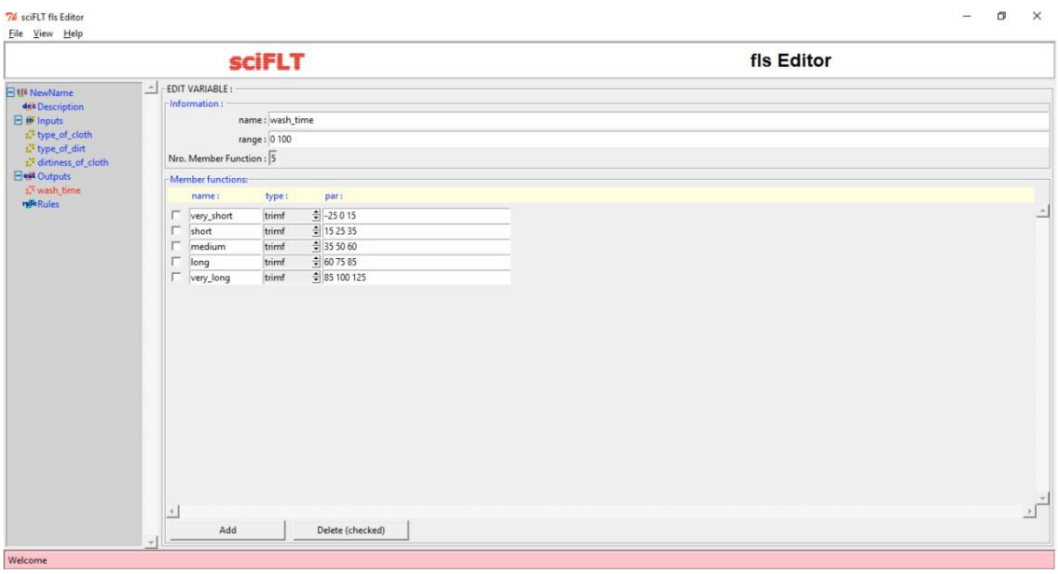




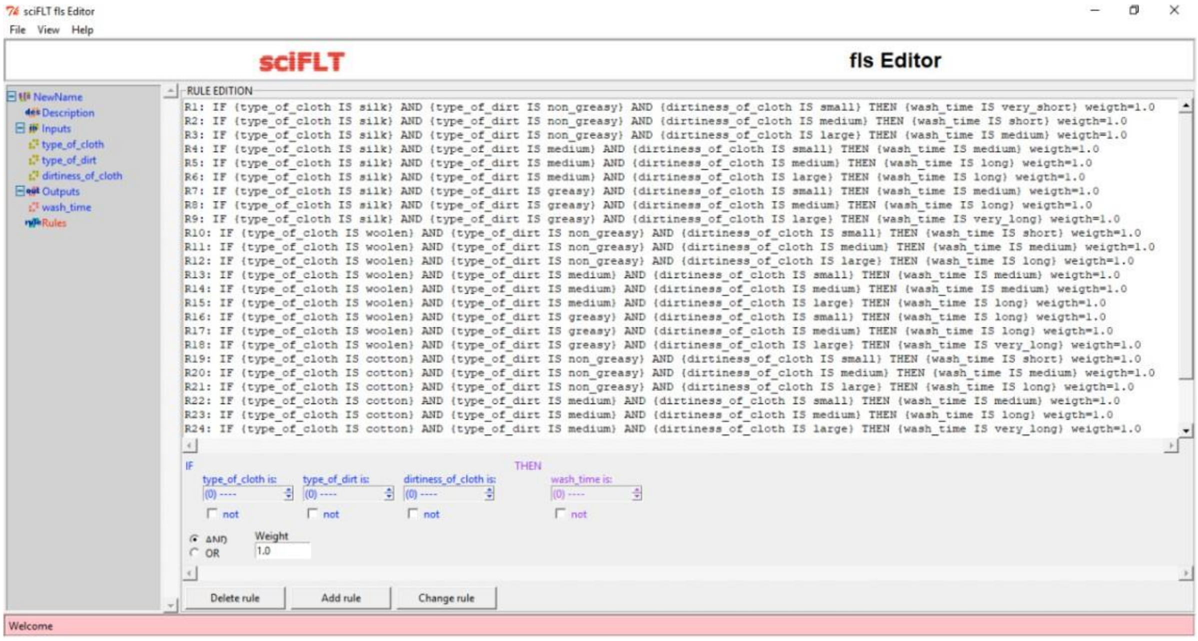


4. Design Output Variable.





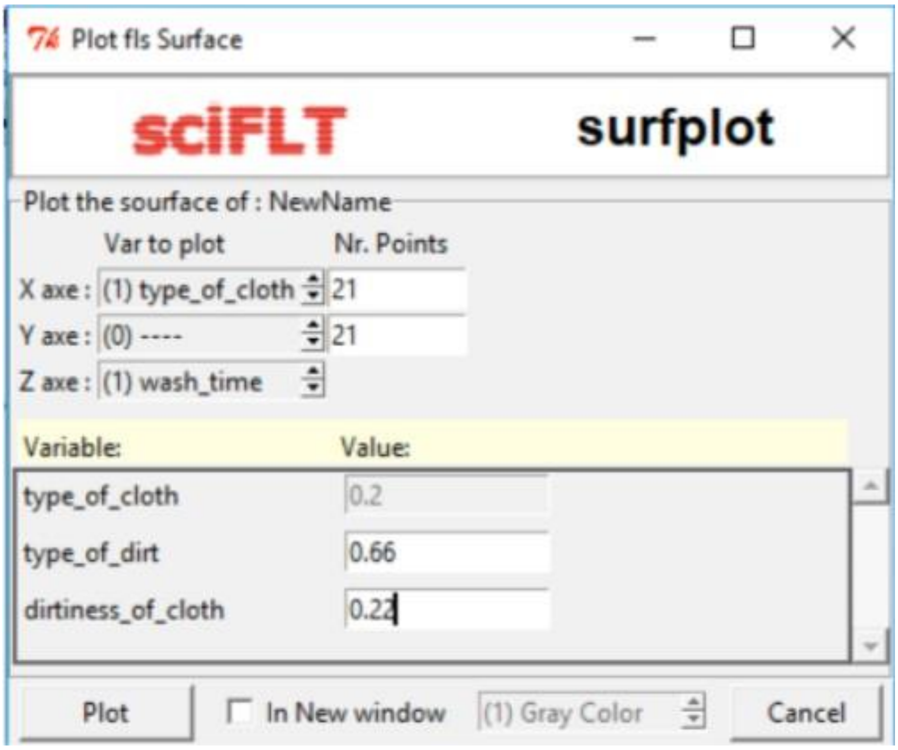
5. Define the rules.



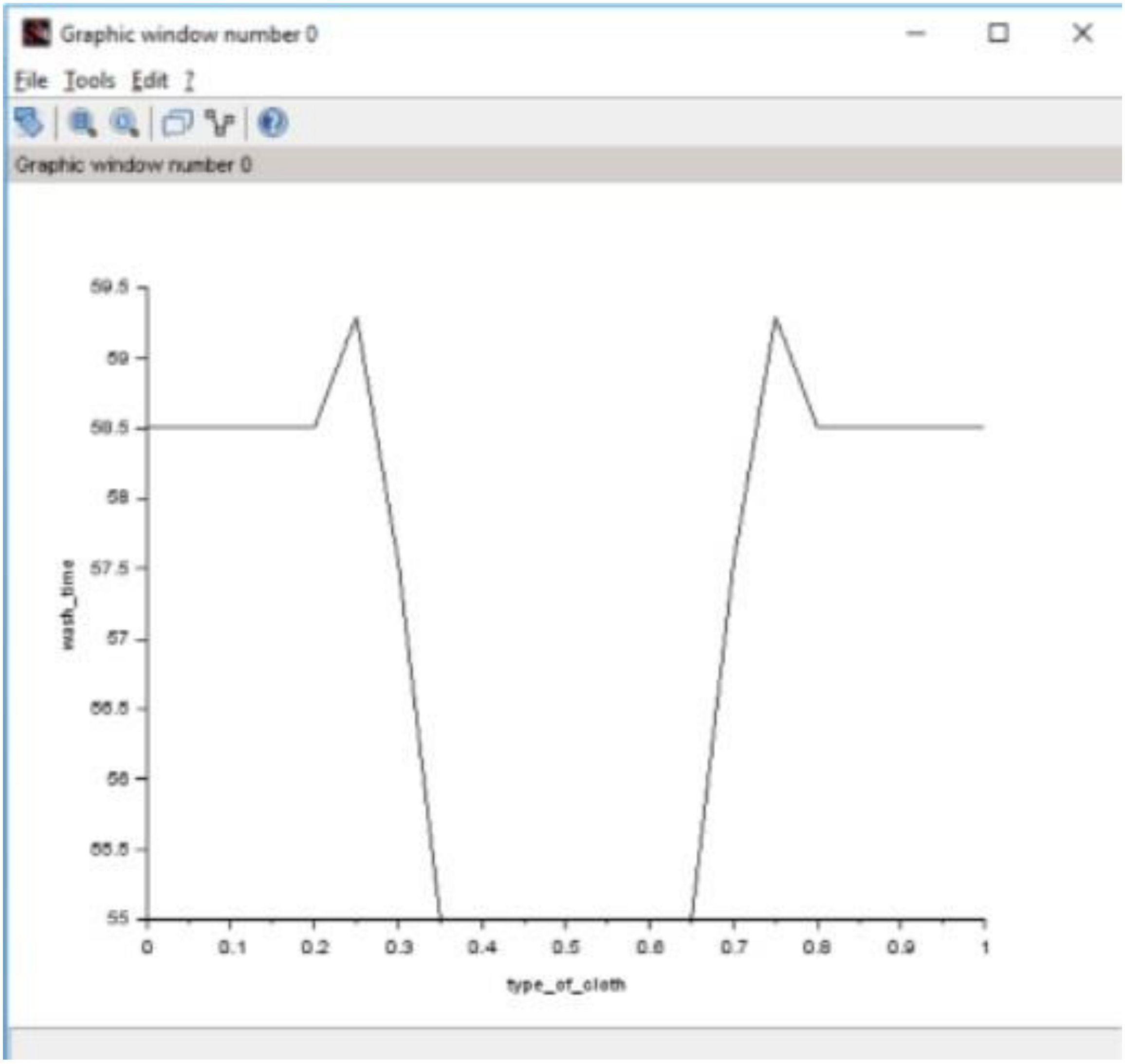
6. Execute using loadfls and evalfls commands.

Output :

```
Scilab 6.1.1 Console
-->flsl=loadfls("washing.fls")
 flsl  =

        name  : 'NewName'
     comment  : 'NewComment'
        type  : 'm'
       SNorm  : 'max'
    SNormPar  : [0]
       TNorm  : 'min'
    TNormPar  : [0]
        Comp  : 'one'
     CompPar  : [0]
   ImpMethod  : 'min'
   AggMethod  : 'max'
 defuzzMethod : 'centroide'
       input  : 3 input(s)
      output  : 1 output(s)
        rule  : 27 rule(s)

-->evalfls([0.2 0.66 0.22],flsl)
 ans  =

   58.507743

-->plotsurf(flsl)
```



Plot fls Surface
sciFLT
surfplot
Plot the sourface of : NewName
Var to plot
Nr. Points
X axe : (1) type_of_cloth
21
Y axe : (0) ----
21
Z axe : (1) wash_time
Variable:
Value:
type_of_cloth
0.2
type_of_dirt
0.66
dirtiness_of_cloth
0.22
Plot
In New window
(1) Gray Color
Cancel



Graphic window number 0
File Tools Edit ?
Graphic window number 0
59.5
59
58.5
58
57.5
57
56.5
56
55.5
55
wash_time
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
type_of_cloth

Viva Questions:

1. **What is the reason that logic function has rapidly become one of the most successful technologies for developing sophisticated control systems?**

   There are mainly two reasons:

   (i) Fuzzy logic applies the concept of 'certain degree' which is similar to the way human beings think. Instead of just being either true or false, fuzzy logic can be true partially andalso false partially at the same time. This is similar to the human mind.

   (ii) Fuzzy logic can use exact points representing to what degree an event occurs and withfuzzy rules it generates precise outcomes.

2. **What is the sequence of steps taken in designing a fuzzy logic machine?**

   Following is the sequence for the designing a fuzzy logic machine:

   Fuzzification -> Rule Evaluation -> Defuzzification

   When designing a fuzzy logic, we first have to define the fuzzy sets and make appropriate member functions. The rule evaluation comes in which matches the set to its corresponding rules.

3. What is the fuzzy inference system (FIS)?

A fuzzy inference system (FIS) is defined as a system that uses fuzzy membership functions to make a decision.

4. What is defuzzification and fuzzy controller?

Defuzzification is the process of producing a quantifiable result in Crisp logic, given fuzzy sets and corresponding membership degrees. It is the process that maps a fuzzy set to a crisp set

Experiment 4

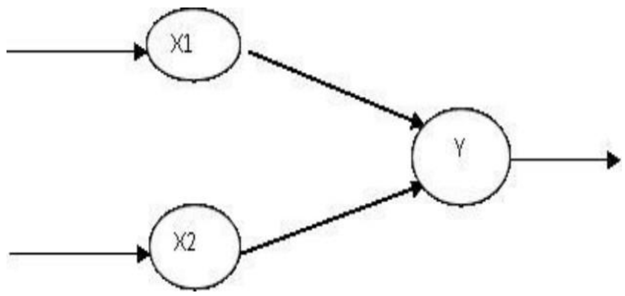
Aim : To implement Mc-Culloch pitts Model using XOR

Theory :

Neural network was inspired by the design and functioning of human brain and components.

Definition:
Information processing model that is inspired by the way biological nervous system (i.e the brain) process information, is called Neural Network.
Neural Network has the ability to learn by examples. It is not designed to perform fix /specific task, rather task which need thinking (e.g. Predictions). ANN is composed of large number of highly interconnected processing elements(neurons) working in unison to solve problems. It mimic human brain. It is configured for special application such as pattern recognition and data classification through a learning process. ANN is 85-90% accurate.

Basic Operation of a Neural Network:



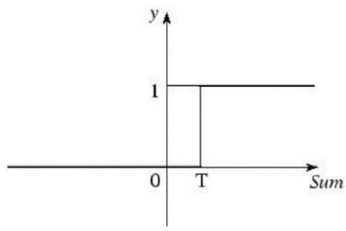
X1 and X2 – input neurons.
Y- output neuron
Weighted interconnection links- W1 and W2.
Net input calculation is :
Yin= x1w1+x2w2
Output is :
y=f(Yin)
Output= function

The McCulloch-Pitts Model of Neuron:

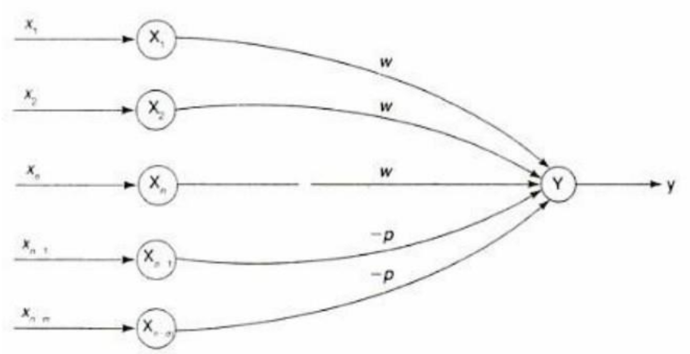
The early model of an artificial neuron is introduced by Warren McCulloch and Walter Pitts in 1943. The McCulloch-Pitts neural model is also known as linear threshold gate. It is a neuron of a set of inputs I1,I2,I3...Im and one output y . The linear threshold gate simply classifies the set of inputs into two different classes. Thus the output y is binary. Such a function can be described mathematically using these equations:

$$Sum = \sum_{i=1}^{N} I_i W_i,$$

$$y = f(Sum).$$



W1,W2...Wm are weight values normalized in the range of either (0,1) or (-1,1) and associated with each input line, Sum is the weighted sum, and T is a threshold constant. The function f is a linear step function at threshold T as shown in figure



A simple M-P neuron is shown in the figure.
It is excitatory with weight (w>0) / inhibitory with weight –p (p<0).
In the Fig., inputs from x1 to xn possess excitatory weighted connection and Xn+1 toxn+m has inhibitory weighted interconnections.
Since the firing of neuron is based on threshold, activation function is defined as

$$f(y_{in}) = \begin{Bmatrix} 1 \;\; if\; y_{in} \geq \theta \\ 0 \;\; if\; y_{in} < \theta \end{Bmatrix}$$

For inhibition to be absolute, the threshold with the activation function should satisfy the following condition:

$$\theta > nw - p$$

Output will fire if it receives ―k‖ or more excitatory inputs but no inhibitory inputs where

$$kw \geq \theta > (k-1)\, w$$

- The M-P neuron has no particular training algorithm.
- An analysis is performed to determine the weights and the threshold.
- It is used as a building block where any function or phenomenon is modelled based on alogic function.

Problem Statement: Generate XOR function using McCulloch-Pitts neural net by a SCILAB program.

Code :

```
clear ; clc ;

// Getting weights and thresholds value

disp('Syeda Reeha Quasar- 14114802719');

disp ('Enter weights') ;
w11 = input ( 'Weight w11= ') ;
w12 = input ('Weight w12= ') ;
w21 = input ('Weight w21= ') ;
w22 = input ('Weight w22= ') ;
v1 = input ('Weight v1 = ') ;
v2 = input ('Weight v2= ') ;

disp ('Enter threshold value') ;
theta = input ('theta= ') ;
x1 =[0 0 1 1];
x2 =[0 1 0 1];
z =[0;1;1;0];

con =1;

while con
   zin1 = x1 * w11 + x2 * w21 ;
   zin2 = x1 * w21 + x2 * w22 ;
   for i =1:4
      if zin1 ( i ) >= theta
         y1 ( i ) =1;
      else
         y1 ( i ) =0;
      end
      if zin2 ( i ) >= theta
         y2 ( i ) =1;
      else
         y2 ( i ) =0;
      end
   end
   yin = y1 * v1 + y2 * v2 ;
   for i =1:4
      if yin ( i ) >= theta ;
         y ( i ) =1;

      else
         y ( i ) =0;
      end
   end
   disp ('Output of Net ') ;
   disp ( y );
   if y == z con =0;
   else
      disp ('Net is not learning to enter another set of weights and threshold values' ) ;
      w11 = input ( 'Weight w11= ' ) ;
```

```
        w12 = input ( 'Weight w12= ' ) ;
        w21 = input ( 'Weight w21= ' ) ;
        w22 = input ( 'Weight w22= ' ) ;
        v1 = input ( 'Weight v1= ' ) ;
        v2 = input ( 'Weight v2= ' ) ;
        theta = input ( 'theta= ' ) ;
    end
end

disp ('McCulloch−Pitts Net for XOR function') ;
disp ( 'Weights of Neuron Z1 ' ) ;
disp ( w11 ) ;
```

Output :

```
Scilab 6.1.0 Console
File  Edit  Control  Applications  ?
Scilab 6.1.0 Console

  "Syeda Reeha Quasar- 14114802719"

  "Enter weights"
Weight w11= 1

Weight w12= -1

Weight w21= -1

Weight w22= 1

Weight v1 = 1

Weight v2= 1


  "Enter threshold value"
theta= 1


  "Output of Net "

   0.
   1.
   1.
   0.

  "McCulloch-Pitts Net for XOR function"

  "Weights of Neuron Z1 "

   1.

-->
```

Viva Questions:

1. What are Neural Networks? What are the types of neural networks?
A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates.
Types:
Feedforward Neural Network – Artificial Neuron.
Radial Basis Function Neural Network.
Multilayer Perceptron.
Convolutional Neural Network.

2. How are Artificial Neural Networks different from Normal Computers?
Difference between traditional computers and artificial neural networks is the way in which they function. While computers function logically with a set of rules and calculations, artificial neural networks can function via images, pictures, and concepts

3. What is a simple Artificial Neuron?
An artificial neuron is a mathematical function conceived as a model of biological neurons, a neural network. Usually each input is separately weighted, and the sum is passed through a nonlinear function known as an activation function or transfer function.

4. What is meant by training in artificial neural networks?
Once a network has been structured for a particular application, that network is rained. To start this process the initial weights are chosen randomly. Then, the training, or learning, begins. There are two approaches to training - supervised and unsupervised

Experiment 5

Aim : Implementation of Single layer Perceptron Learning Algorithm

Theory :

Neural networks are a branch of ―Artificial Intelligence". Artificial Neural Network is a system loosely modelled based on the human brain. Neural networks are a powerful technique to solve many real world problems. They have the ability to learn from experience in order to improve their performance and to adapt themselves to changes in the environment. In addition to that they are able to deal with incomplete information or noisy data and can be very effective especially in situations where it is not possible to define the rules or steps that lead to the solution of a problem. In a nutshell a Neural network can be considered as a black box that is able to predict an output pattern when it recognizes a given input pattern. Once trained, the neural network is able to recognize similarities when presented with a new input pattern, resulting in a predicted output pattern.

In late 1950s, Frank Rosenblatt introduced a network composed of the units that were enhanced version of McCulloch-Pitts Threshold Logic Unit (TLU) model. Rosenblatt's model of neuron, a perceptron, was the result of merger between two concepts from the 1940s, McCulloch-Pitts model of an artificial neuron and Hebbian learning rule of adjusting weights. In addition to the variable weight values, the perceptron model added an extra input that represents bias. Thus, the modified equation is now as follows:

$$Sum = \sum_{i=1}^{N} I_i W_i + b,$$

where b represents the bias value.

<u>Algorithm:</u>

Perceptron Learning Algorithm

The perceptron learning rule was originally developed by Frank Rosenblatt in the late 1950s. Training patterns are presented to the network's inputs; the output is computed. Then the connection weights wj are modified by an amount that is proportional to the product of the difference between the actual output, y, and the desired output, d, and the input pattern, x.

The algorithm is as follows:

1. Initialize the weights and threshold to small random numbers.
2. Present a vector x to the neuron inputs and calculate the output.
3. Update the weights according to:

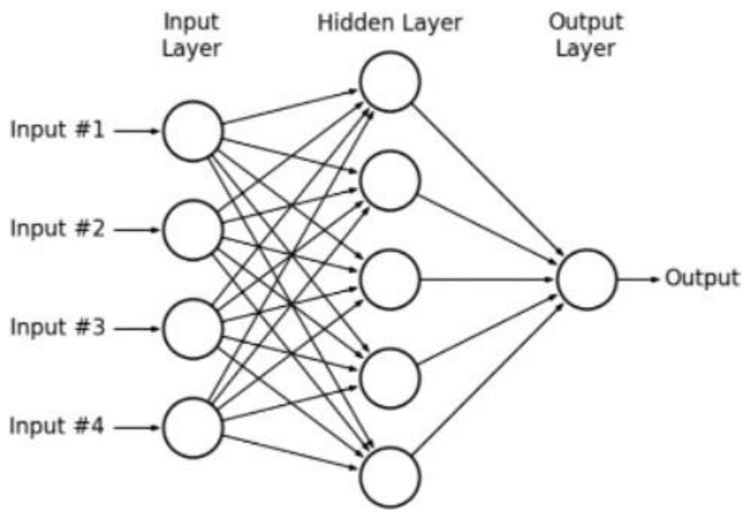
$$w_j(t+1) = w_j(t) + \eta(d-y)\mathbf{x}$$

where
• d is the desired output,
• t is the iteration number, and
• eta is the gain or step size, where 0.0 < n < 1.0

4. Repeat steps 2 and 3 until:
1. the iteration error is less than a user-specified error threshold or
2. a predetermined number of iterations have been completed.

Learning only occurs when an error is made; otherwise the weights are left unchanged.

Multilayer Perceptron:



Code :

```
clear ; clc ;

// Getting weights and thresholds value

disp('Syeda Reeha Quasar- 14114802719');

x=[0 0 1 1; 0 1 0 1];
//input variable pass
d=[1 1 0 0];
//target output
w=[-20 3 3 -5];
//initialize weight for per input
z=[0 0 0 0];
//vector to store the calculated value of the sigma input*weight + bias
bias=0.2;
//iniatlize of bias to store the value

//calculate the values total value
for j= 1:2
    sigma=0;
```

```
    for i=1 : 4
        sigma=bias + x(j,i)*mtlb_t(w);
    end
end

disp('final calculation');
disp(sigma);

//set the theta value for step function
theta=0.3;
for i=1:4
    if sigma(i)> theta
        z(i)=1;
    else if sigma(i)<=theta
            z(i)=0;
        end
    end
end
disp('Final output of the computed net value');
disp(z);
disp('oldweight');
disp(w);

//updation to minimize the error
eta=1.2; // learning rate ;
for j= 1:4
    lr=0;
    for i=1 : 2
        lr= x(i,j)*eta;
    end
end

disp(lr);

for i=1:4
    if z(i)==1 & d(i)==0
        w(i)=w(i)-lr;
    else if z(i)==0 & d(i)==1
            w(i)=w(i)+lr;
        end
    end
end

//final weight
disp('final updated weight');
disp(w);
```

Output :

```
Scilab 6.1.0 Console
File  Edit  Control  Applications  ?
Scilab 6.1.0 Console

 "Syeda Reeha Quasar- 14114802719"

 "final calculation"

 -19.8
  3.2
  3.2
 -4.8

 "Final output of the computed net value"

  0.   1.   1.   0.

 "oldweight"

 -20.   3.   3.  -5.

  1.2

 "final updated weight"

 -18.8   3.   1.8  -5.

-->
```

Viva Questions:

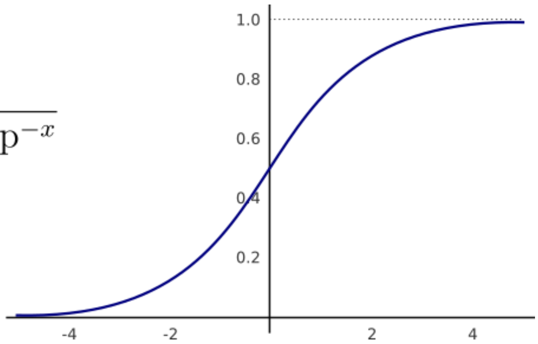
1. What is a feed forward network?

Deep feedforward networks, also often called feedforward neural networks, or multilayer perceptrons(MLPs), are the quintessential deep learning models. The goal of a feedforward network is to approximate some function f*. These models are called feedforward because information flows through the function being evaluated from x, through the intermediate computations used to define f, and finally to the output y.

2. Write the logistic sigmoid function?

The logistic sigmoid has the following form.

$$\sigma(x) = \frac{1}{1 + \exp^{-x}}$$



3. Why use Artificial Neural Networks? What are its advantages?

ANNs have some key advantages that make them most suitable for certain problems and situations:

• ANNs have the ability to learn and model non-linear and complex relationships, which is really important because in real-life, many of the relationships between inputs and outputs are non-linear as well as complex.

• ANNs can generalize — After learning from the initial inputs and their relationships, it can infer unseen relationships on unseen data as well, thus making the model generalize and predict on unseen data.

• Unlike many other prediction techniques, ANN does not impose any restrictions on the input variables.

4. List some commercial practical applications of Artificial Neural Networks.

ANNs, due to some of its wonderful properties have many applications:

• Image Processing and Character recognition. • Forecasting

5. What are the disadvantages of Artificial Neural Networks?

• Hardware dependence

• Unexplained behavior of the network

• Determination of proper network structure

Experiment 6

Aim : Implementation of unsupervised learning algorithm – Hebbian Learning

Theory :

<u>Unsupervised Learning Algorithm:</u>

These types of model are not provided with the correct results during the training. It can be used to cluster the input data in classes on the basis of their statistical properties only. The labelling can be carried out even if the labels are only available for a small number of objects represented of the desired classes. All similar input patters are grouped together as clusters. If matching pattern is not found, a new cluster is formed. In contrast to supervised learning, unsupervised or self-organized learning does not require an external teacher. During the training session, the neural network receives a number of different patterns & learns how to classify input data into appropriate categories. Unsupervised learning tends to follow the neuro-biological organization of brain. It aims to learn rapidly & can be used in real-time.

Hebbian Learning Rule, also known as Hebb Learning Rule, was proposed by Donald O Hebb. It is one of the first and also easiest learning rules in the neural network. It is used for pattern classification. It is a single layer neural network, i.e. it has one input layer and one output layer. The input layer can have many units, say n. The output layer only has one unit. Hebbian rule works by updating the weights between neurons in the neural network for each training sample.

<u>Hebbian Learning Rule Algorithm :</u>

- Set all weights to zero, wi = 0 for i=1 to n, and bias to zero.
- For each input vector, S(input vector) : t(target output pair), repeat steps 3-5.
- Set activations for input units with the input vector Xi = Si for i = 1 to n.
- Set the corresponding output value to the output neuron, i.e. y = t.
- Update weight and bias by applying Hebb rule for all i = 1 to n:

$$w_i\,(\text{new}) = w_i\,(\text{old}) + x_i\,y$$

$$b\,(\text{new}) = b\,(\text{old}) + y$$

Hebb's law provide basis for learning without a teacher. Learning here is a local phenomenon occurring without feedback from the environment.

- ➢ Using Hebb's Law we can express the adjustment applied to weight w_{ij} at iteration p in the following form: $\Delta w_{ij}(p) = F[y_i(p), x_i(p)]$
- ➢ As a special case, we can represent Hebb's Law as follows: $\Delta w_{ij}(p) = \alpha y_i(p) x_i(p)$
  where α is the learning rate parameter.
- ➢ Hebbian learning implies that weights can only increase. To resolve this problem, we might impose a limit on the growth of synaptic weights. It can be done by introducing non-linear forgetting factor into Hebb's Law:

$$\Delta w_{ij}(p) = \alpha y_i(p) x_i(p) - \varphi y_i(p), w_{ij}(p)$$

  where φ is the forgetting factor.

Hebbian learning algorithm
Step 1: Initialization
Set initial synaptic weights and thresholds to small random values, say in an interval [0, 1].
Step 2: Activation
Compute the neuron output at iteration $y_j(p) = \sum_{i=1}^{n} x_i(p)\, w_{ij}(p) - \theta_j$

Where n is number of neuron inputs, & θ_j is the threshold value of neuron j.
Step 3: Learning
Update the weights in the $w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$ network

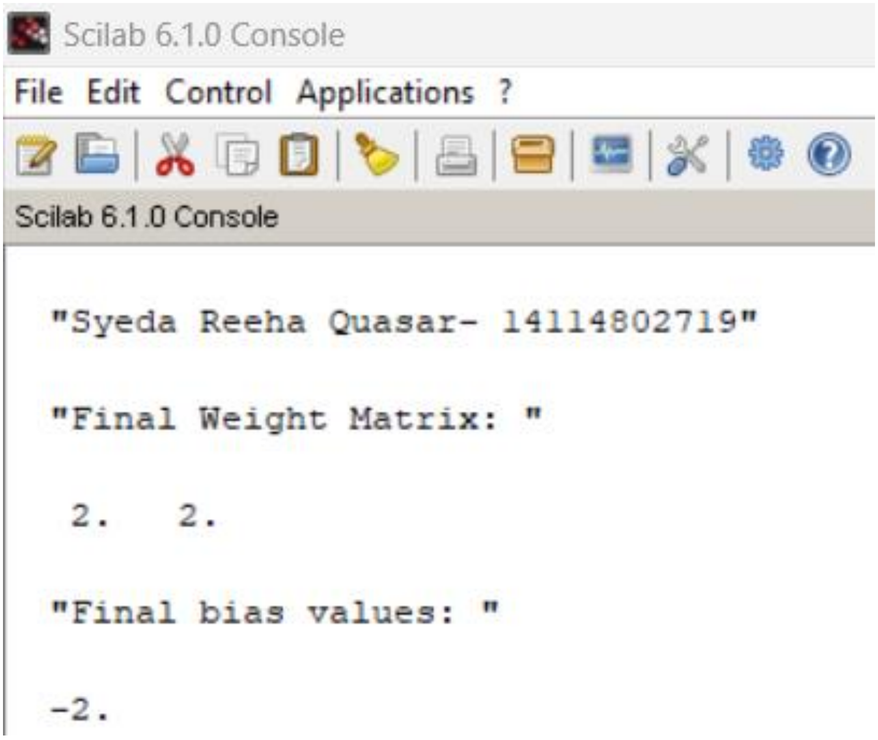
Where $\Delta w_{ij}(p)$ is the weight correction at iteration p. $\Delta w_{ij}(p) = \varphi\, y_j(p)[\lambda\, x_i(p) - w_{ij}(p)]$
Step 4: Iteration
Increase iteration p by one, go back to step 2.

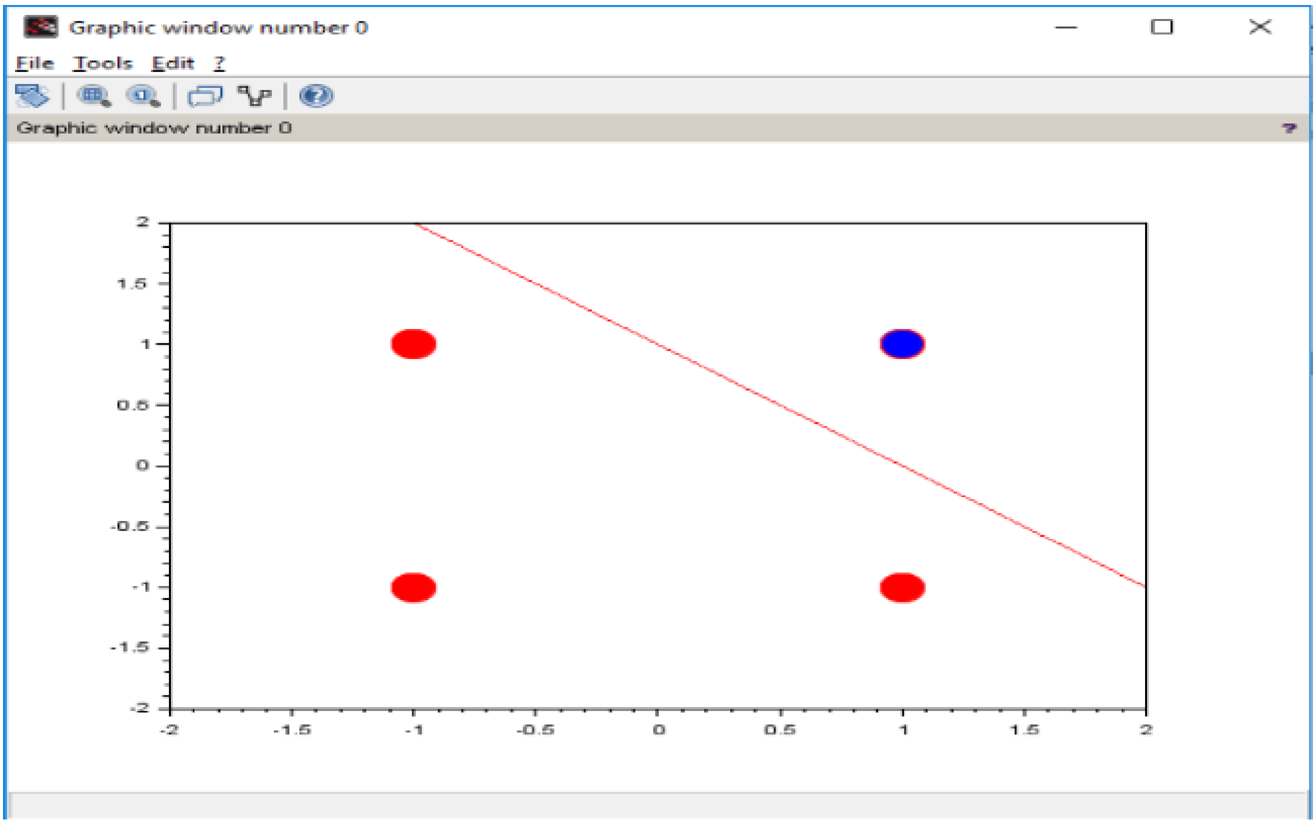
Code :

```
clear ; clc ;

// Getting weights and thresholds value

disp('Syeda Reeha Quasar- 14114802719');

x=[1 1 -1 -1;1 -1 1 -1];
t=[1 -1 -1 -1];
w=[0 0];
b=0;

for i=1:4
   for j=1:2
      w(j)=w(j)+t(i)*x(j,i);
   end
   b=b+t(i);
end

disp("Final Weight Matrix: ");
disp(w);
disp("Final bias values: ");
disp(b);

plot(x(1,1),x(2,1),'or','MarkerSize',20,'MarkerFaceColor',[0 0 1]);
set(gca(),"auto_clear","off");
plot(x(1,2),x(2,2),'or','MarkerSize',20,'MarkerFaceColor',[1 0 0]);
set(gca(),"auto_clear","off");
plot(x(1,3),x(2,3),'or','MarkerSize',20,'MarkerFaceColor',[1 0 0]);
set(gca(),"auto_clear","off");
plot(x(1,4),x(2,4),'or','MarkerSize',20,'MarkerFaceColor',[1 0 0]);
set(gca(),"auto_clear","off");

m=-(w(1)/w(2));
c=-b/w(2);
```

```
x1=linspace(-2,2,100);
x2=m*x1+c; plot(x2,x1,'r');
a=gca() ;//get the current axes
a.box="on";
a.data_bounds=[-2,-2;2,2]; //define the bounds
```

Output :





Viva Questions:

1. What is unsupervised and supervised training?

- In Supervised learning, you train the machine using data which is well "labeled."
- Unsupervised learning is a machine learning technique, where you do not need to supervise the model.

2. How do Artificial Neurons learn?

The smallest and most important unit of the artificial neural network is the neuron. As in biological neural systems, these neurons are connected with each other and together they have great processing power. Every neuron has input connections and output connections. These connections simulate the behavior of the synapses in the brain. The same way that synapses in the brain transfer the signal from one neuron to another, connections pass information between artificial neurons. These connections have weights, meaning that the value that is sent to every connection is multiplied by this factor.

3. What is the difference between neural network and fuzzy logic?

Fuzzy Logic

It is an extension of convention set theory (also called crisp set) In conventional set logic true is 1(in general) and false is 0. But in fuzzy terms there is no concept of exact true or exact false; they are given membership.

Neural Network

It is nothing but trying to simulate our brain in the electronic domain so that it can learn like we do.

4. What is Unsupervised Hebbian learning algorithm

It is a linear feedforward neural network model for unsupervised learning with applications primarily in principal components analysis.

Experiment 7

Aim : Implementation Genetic Application – Match Word Finding.

Theory :

<u>Genetic algorithm:</u>

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

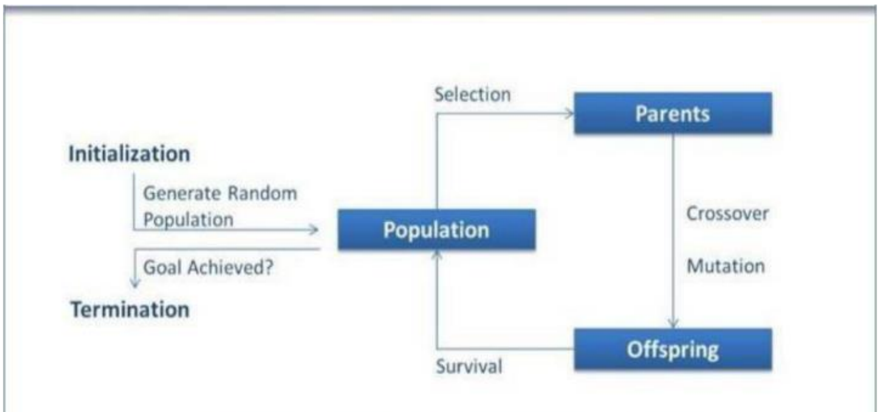
• Genetic algorithms are inspired by Darwin's theory about evolution. Solution to a problem solved by genetic algorithms is evolved.

• 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.

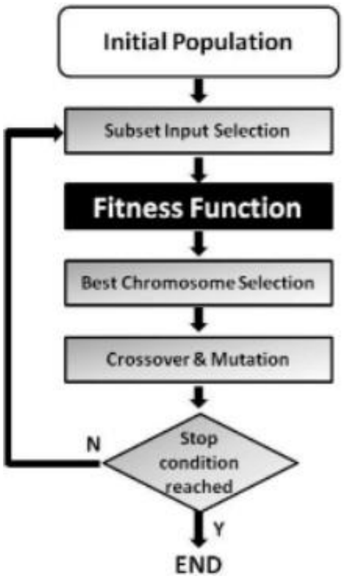
Outline of the Basic Genetic Algorithm:

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
    - **[Selection]** Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)
    - **[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.
    - **[Mutation]** With a mutation probability mutate new offspring at each locus (position in chromosome).
    - [**Accepting**] Place new offspring in a new population
    - [**Replace**] Use new generated population for a further run of algorithm
    - **[Test]** If the end condition is satisfied, stop, and return the best solution in current population
    - **[Loop]** Go to step 2



General Scheme of Evolutionary Process

Flowchart:



Problem Statement: Match Word Finding
Here we try to guess a word from the given population of word.

Algorithm: **Match Word Finding Algorithm**

Step 1: Select the word to be guessed. This value is taken through user input.
Step 2: Initialize the population. User inputs the population.
Step 3: Evaluate the population. Fitness is assigned based on number of correct letters in correct place.
Step 4: Select breeding population. Selection is done on the basis of fitness.
Step 5: Create new population. Population is created by using uniform crossover between breeding populations.
Step 6: Check for stopping condition. Here maximum fitness value in population is checked. If it is 60%.
Step 7: If stopping condition is not true, goto Step 3; else return the offspring with highest fitness value.

Code :

```
# Python3 program to create target string, starting from
# random string using Genetic Algorithm

import random

# Number of individuals in each generation
POPULATION_SIZE = 100

# Valid genes
GENES = '''abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOP
QRSTUVWXYZ 1234567890, .-;:_!"#%&/()=?@${[]}'''
```

```
# Target string to be generated
TARGET = "Soft Computing"

class Individual(object):
    '''
    Class representing individual in population
    '''
    def __init__(self, chromosome):
        self.chromosome = chromosome
        self.fitness = self.cal_fitness()

    @classmethod
    def mutated_genes(self):
        '''
        create random genes for mutation
        '''
        global GENES
        gene = random.choice(GENES)
        return gene

    @classmethod
    def create_gnome(self):
        '''
        create chromosome or string of genes
        '''
        global TARGET
        gnome_len = len(TARGET)
        return [self.mutated_genes() for _ in range(gnome_len)]

    def mate(self, par2):
        '''
        Perform mating and produce new offspring
        '''


        # chromosome for offspring
        child_chromosome = []
        for gp1, gp2 in zip(self.chromosome, par2.chromosome):

            # random probability
            prob = random.random()

            # if prob is less than 0.45, insert gene
            # from parent 1
            if prob < 0.45:
                child_chromosome.append(gp1)

            # if prob is between 0.45 and 0.90, insert
```

```
            # gene from parent 2
            elif prob < 0.90:
                child_chromosome.append(gp2)

            # otherwise insert random gene(mutate),
            # for maintaining diversity
            else:
                child_chromosome.append(self.mutated_genes())

        # create new Individual(offspring) using
        # generated chromosome for offspring
        return Individual(child_chromosome)

    def cal_fitness(self):
        '''
        Calculate fittness score, it is the number of
        characters in string which differ from target
        string.
        '''
        global TARGET
        fitness = 0
        for gs, gt in zip(self.chromosome, TARGET):
            if gs != gt: fitness+= 1
        return fitness

# Driver code
def main():
    global POPULATION_SIZE

    print("Syeda Reeha Quasar - 14114802719")

    #current generation
    generation = 1

    found = False
    population = []

    # create initial population
    for _ in range(POPULATION_SIZE):
                gnome = Individual.create_gnome()
                population.append(Individual(gnome))

    while not found:

        # sort the population in increasing order of fitness score
        population = sorted(population, key = lambda x:x.fitness)
```

```
        # if the individual having lowest fitness score ie.
        # 0 then we know that we have reached to the target
        # and break the loop
        if population[0].fitness <= 0:
            found = True
            break

        # Otherwise generate new offsprings for new generation
        new_generation = []

        # Perform Elitism, that mean 10% of fittest population
        # goes to the next generation
        s = int((10*POPULATION_SIZE)/100)
        new_generation.extend(population[:s])

        # From 50% of fittest population, Individuals
        # will mate to produce offspring
        s = int((90*POPULATION_SIZE)/100)
        for _ in range(s):
            parent1 = random.choice(population[:50])
            parent2 = random.choice(population[:50])
            child = parent1.mate(parent2)
            new_generation.append(child)

        population = new_generation

        print("Generation: {}\tString: {}\tFitness: {}".\
              format(generation,
                     "".join(population[0].chromosome),
                     population[0].fitness))

        generation += 1


    print("Generation: {}\tString: {}\tFitness: {}".\
          format(generation,
          "".join(population[0].chromosome),
          population[0].fitness))

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

Output :

```
TERMINAL   AZURE   PROBLEMS   OUTPUT   DEBUG CONSOLE

PS E:\sem 7\sc> python -u "e:\sem 7\sc\exp.py"
Syeda Reeha Quasar - 14114802719
Generation: 1   String: iHs
 C#=gFK[;O      Fitness: 12
Generation: 2   String: iHs
 C#=gFK[;O      Fitness: 12
Generation: 3   String: !-"od]Wm?R#%ng  Fitness: 11
Generation: 4   String: n/Z-:4ompf?Hmg  Fitness: 10
Generation: 5   String: n28! ]ompR?H;g  Fitness: 9
Generation: 6   String: Do!q CTipRvAng  Fitness: 8
Generation: 7   String: noJ( C!mpsvinI  Fitness: 7
Generation: 8   String: noJ( C!mpsvinI  Fitness: 7
Generation: 9   String: 5 8
 CompoKing      Fitness: 6
Generation: 10  String: 5 8
 CompoKing      Fitness: 6
Generation: 11  String: no@0 Comp/2ing  Fitness: 5
Generation: 12  String: no@0 Comp/2ing  Fitness: 5
Generation: 13  String: no@0 Comp/2ing  Fitness: 5
Generation: 14  String: SoJq CompN?ing  Fitness: 4
Generation: 15  String: SoJq CompN?ing  Fitness: 4
Generation: 16  String: SoJq CompN?ing  Fitness: 4
Generation: 17  String: SoJq CompN?ing  Fitness: 4
Generation: 18  String: SoJq CompN?ing  Fitness: 4
Generation: 19  String: SoUt Compc?ing  Fitness: 3
Generation: 20  String: SoUt Compc?ing  Fitness: 3
Generation: 21  String: SoUt Compc?ing  Fitness: 3
Generation: 22  String: SoUt Compc?ing  Fitness: 3
Generation: 23  String: SoUt Compc?ing  Fitness: 3
Generation: 24  String: SoUt Compc?ing  Fitness: 3
Generation: 25  String: SoUt Compc?ing  Fitness: 3
Generation: 26  String: SoUt Compc?ing  Fitness: 3
Generation: 27  String: SoKt Compu?ing  Fitness: 2
Generation: 28  String: SoKt Compu?ing  Fitness: 2
Generation: 29  String: SoKt Compu?ing  Fitness: 2
Generation: 30  String: SoKt Compu?ing  Fitness: 2
Generation: 31  String: SoKt Compu?ing  Fitness: 2
Generation: 32  String: SoKt Compu?ing  Fitness: 2
Generation: 33  String: SoKt Compu?ing  Fitness: 2
Generation: 34  String: SoKt Compu?ing  Fitness: 2
Generation: 35  String: SoKt Compu?ing  Fitness: 2
Generation: 36  String: SoKt Compu?ing  Fitness: 2
Generation: 37  String: SoKt Compu?ing  Fitness: 2
Generation: 38  String: SoKt Compu?ing  Fitness: 2
Generation: 39  String: SoKt Compu?ing  Fitness: 2
Generation: 40  String: SoKt Compu?ing  Fitness: 2
Generation: 41  String: SoKt Compu?ing  Fitness: 2
Generation: 42  String: SoKt Compu?ing  Fitness: 2
Generation: 43  String: SoKt Compu?ing  Fitness: 2
Generation: 44  String: SoKt Compu?ing  Fitness: 2
Generation: 45  String: SoKt Compu?ing  Fitness: 2
Generation: 46  String: SoUt Computing  Fitness: 1
Generation: 47  String: SoUt Computing  Fitness: 1
Generation: 48  String: SoUt Computing  Fitness: 1
Generation: 49  String: SoUt Computing  Fitness: 1
Generation: 50  String: SoUt Computing  Fitness: 1
Generation: 51  String: SoUt Computing  Fitness: 1
Generation: 52  String: SoUt Computing  Fitness: 1
Generation: 53  String: SoUt Computing  Fitness: 1
Generation: 83  String: SoUt Computing  Fitness: 1
Generation: 84  String: SoUt Computing  Fitness: 1
Generation: 85  String: SoUt Computing  Fitness: 1
Generation: 86  String: SoUt Computing  Fitness: 1
```

TERMINAL AZURE PROBLEMS OUTPUT DEBUG CONSOLE

```
Generation: 26  String: SoUt Compc?ing  Fitness: 3
Generation: 27  String: SoKt Compu?ing  Fitness: 2
Generation: 28  String: SoKt Compu?ing  Fitness: 2
Generation: 29  String: SoKt Compu?ing  Fitness: 2
Generation: 30  String: SoKt Compu?ing  Fitness: 2
Generation: 31  String: SoKt Compu?ing  Fitness: 2
Generation: 32  String: SoKt Compu?ing  Fitness: 2
Generation: 33  String: SoKt Compu?ing  Fitness: 2
Generation: 34  String: SoKt Compu?ing  Fitness: 2
Generation: 35  String: SoKt Compu?ing  Fitness: 2
Generation: 36  String: SoKt Compu?ing  Fitness: 2
Generation: 37  String: SoKt Compu?ing  Fitness: 2
Generation: 38  String: SoKt Compu?ing  Fitness: 2
Generation: 39  String: SoKt Compu?ing  Fitness: 2
Generation: 40  String: SoKt Compu?ing  Fitness: 2
Generation: 41  String: SoKt Compu?ing  Fitness: 2
Generation: 42  String: SoKt Compu?ing  Fitness: 2
Generation: 43  String: SoKt Compu?ing  Fitness: 2
Generation: 44  String: SoKt Compu?ing  Fitness: 2
Generation: 45  String: SoKt Compu?ing  Fitness: 2
Generation: 46  String: SoUt Computing  Fitness: 1
Generation: 47  String: SoUt Computing  Fitness: 1
Generation: 48  String: SoUt Computing  Fitness: 1
Generation: 49  String: SoUt Computing  Fitness: 1
Generation: 50  String: SoUt Computing  Fitness: 1
Generation: 51  String: SoUt Computing  Fitness: 1
Generation: 52  String: SoUt Computing  Fitness: 1
Generation: 53  String: SoUt Computing  Fitness: 1
Generation: 83  String: SoUt Computing  Fitness: 1
Generation: 84  String: SoUt Computing  Fitness: 1
Generation: 85  String: SoUt Computing  Fitness: 1
Generation: 86  String: SoUt Computing  Fitness: 1
Generation: 87  String: SoUt Computing  Fitness: 1
Generation: 88  String: SoUt Computing  Fitness: 1
Generation: 89  String: SoUt Computing  Fitness: 1
Generation: 90  String: SoUt Computing  Fitness: 1
Generation: 91  String: SoUt Computing  Fitness: 1
Generation: 92  String: SoUt Computing  Fitness: 1
Generation: 93  String: SoUt Computing  Fitness: 1
Generation: 94  String: SoUt Computing  Fitness: 1
Generation: 95  String: SoUt Computing  Fitness: 1
Generation: 96  String: SoUt Computing  Fitness: 1
Generation: 97  String: SoUt Computing  Fitness: 1
Generation: 98  String: SoUt Computing  Fitness: 1
Generation: 99  String: SoUt Computing  Fitness: 1
Generation: 100 String: SoUt Computing  Fitness: 1
Generation: 101 String: SoUt Computing  Fitness: 1
Generation: 102 String: SoUt Computing  Fitness: 1
Generation: 103 String: SoUt Computing  Fitness: 1
Generation: 104 String: SoUt Computing  Fitness: 1
Generation: 105 String: SoUt Computing  Fitness: 1
Generation: 106 String: SoUt Computing  Fitness: 1
Generation: 107 String: SoUt Computing  Fitness: 1
Generation: 108 String: SoUt Computing  Fitness: 1
Generation: 109 String: SoUt Computing  Fitness: 1
Generation: 110 String: SoUt Computing  Fitness: 1
Generation: 111 String: SoUt Computing  Fitness: 1
Generation: 112 String: SoUt Computing  Fitness: 1
Generation: 113 String: SoUt Computing  Fitness: 1
Generation: 114 String: SoUt Computing  Fitness: 1
Generation: 115 String: Soft Computing  Fitness: 0
PS E:\sem 7\sc>
```

Viva Questions:

1. Name some of the existing search methods.
- Calculus based Search
- Enumerative Search
- Random Search

2. What are the operators involved in a simple genetic algorithm?
There are three main types of operators which must work in conjunction with one another in order for the algorithm to be successful –
- Mutation
- Crossover
- Selection

3. What is reproduction?
Genetic algorithm reproduction methods for distribution system loss reduction and load balancing problems. Selected fittest parents create a new child.

4. What is crossover?
Crossover is a genetic operator used to combine the genetic information of two parents to generate new offspring

Experiment 8

Aim :Study of ANFIS Architecture.

Theory :

ANFIS:

An adaptive neuro-fuzzy inference system or adaptive network-based fuzzy inference system (ANFIS) is a kind of artificial neural network that is based on Takagi–Sugeno fuzzy inference system. Since it integrates both neural networks and fuzzy logic principles, it has potential to capture the benefits of both in a single framework. Its inference system corresponds to a set of fuzzy IF–THEN rules that have learning capability to approximate nonlinear functions. Hence, ANFIS is considered to be a universal estimator. For using the ANFIS in a more efficient and optimal way, one can use the best parameters obtained by genetic algorithm.

ANFIS Architecture:

Representing Takagi-Sugeno Fuzzy Model

For simplicity, we assume that the fuzzy inference sytem under consideration has two inputs x and y and one output z. For a first-order Takagi-Sugeno fuzzy model, a common rule set with two fuzzy if-then rules is the following:

- Rule 1: If x is A1 and y is B1, then f1=p1x+q1y+r1;
- Rule 2: If x is A2 and y is B2, then f2=p2x+q2y+r2;

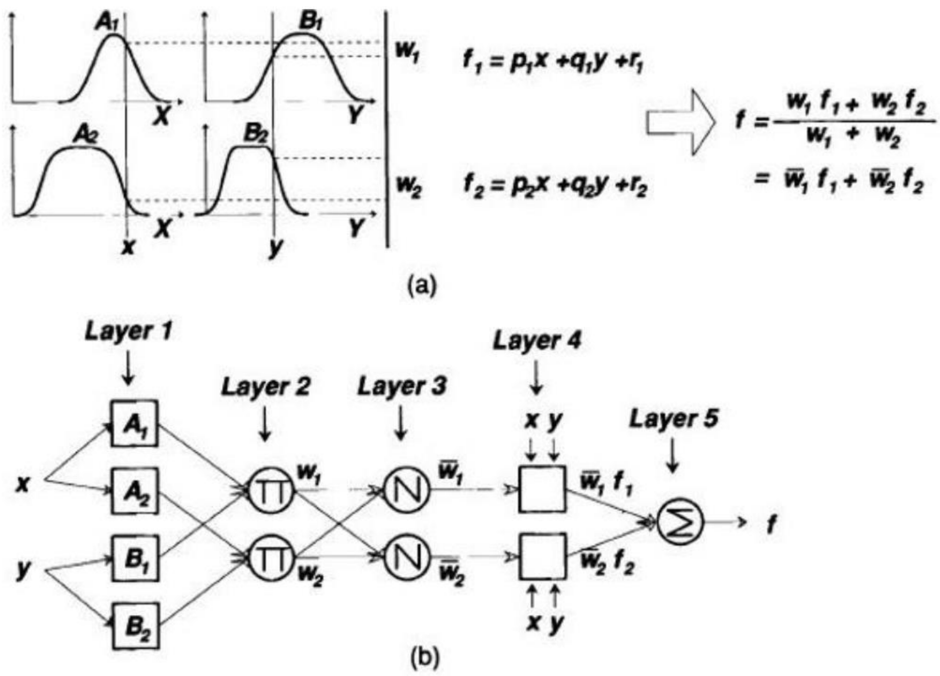


Figure 1: (a) A two inputs first order Takagi-Sugeno fuzzy model with two rules; (b) The equivalent ANFIS architecture.

Layer 1: Every node i in this layer is an adaptive node with a node function. O1, i=μAi(x), for i=1,2, or O1, i=μBi−2(y), for i=3, 4, O1, i=μAi(x), for i=1, 2, or O1, i=μBi−2(y), for i=3, 4, where x (or y) is the input to node i and Ai(or Bi-2) is a linguistic label (such as "small" or "large") associated with this node. In other words, O1, i is the membership grade of a fuzzy set A (=A1, A2, B1 or B2) and it specifies the degree to which the given input x (or y) satifies the quantifier A.

μA(x)=11+|x−ciai|2b,μA(x)=11+|x−ciai|2b,
Where {ai, bi, ci} is the parameter set. As the values of these parameters change, the bell-shaped function varies accordingly, thus exhibiting various forms of membership function for fuzzy set A. Parameters in this layer are referred to as premise parameters.

Layer 2: Every node in this layer is a fixed node labeledanfis, whose output is the product of all the incoming signals:
O2, i = wi = μAi(x) μBi(y), i= 1, 2. O2, I = wi = μAi(x) μBi(y), i = 1, 2.
Each node output represents the firing strength of a rule. In general, any other T-norm operators that perform fuzzy AND can be used as the node function in this layer.

Layer 3: Every node in this layer is a fixed node labeled N. The ith node calculates the ratio of the ith rule's firing strength to the sum of all rules' firing strenghts:
O3, i=w¯I = wiw1+w2′I = 1, 2. O3, I = w¯I = wiw1+w2′I = 1, 2.
For convenience, outputs of this layer are called normalized firing strengthes.

Layer 4: Every node i in this layer is an adaptive node with a node function:
O4, i= wifi¯= w¯i (pix + qiy + ri), O4, i= wifi¯= w¯i (pix + qiy + ri),
Where anfis is a normalized firing strength from layer 3 and {pi, qi, ri} is the parameter set of this node. Parameters in this layer are referred to as consequent parameters.

Layer 5: The single node in this layer is a fixed node labeledanfis, which computes the overall output as the summation of all incoming singals:
Overall output=O5, i=Σiw¯ifi=ΣiwifiΣiwioverall output=O5, i=Σiw¯ifi=ΣiwifiΣiwi
Thus we have constructed an adaptive network that is functionally equivalent to a Sugeno fuzzy model.

Representing Tsukamoto Fuzzy Models:

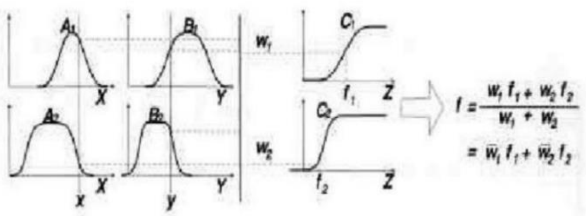


Figure 2(a)

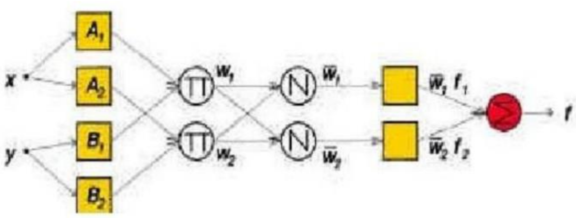


Figure 2: (b)

Figure 2: (a) A two-rule Tsukamoto fuzzy model; (b) The equivalent ANFIS architecture

The extension from TS ANFIS to Tsukamoto ANFIS is straightforward, as show in Figure 2, where the output of each rule (fi, i=1, 2) is induced jointly by a consequent membership function and a firing strength.

Representing Mamdani Fuzzy Model:

For the Mamdani fuzzy inference system with max-min composition, a corresponding ANFIS can be constructed if discrete approximations are used to replace the integrals in the centroid defuzzification scheme introduced in here. However, the resulting ANFIS is much more complicated than either TS ANFIS or Tsukamoto ANFIS. The extra complexity in structure and computation of Mamdani ANFIS with max-min composition does not necessarily imply better learning capability or approximation power. If we adopt sum-product composition and centroid defuzzification for a Mamdani fuzzy model, a corresponding ANFIS can be constructed easily based on Theorem directly without using any approximation at all.

Feature	Description
Name	Adaptive Neuro-Fuzzy Inference System
Architecture	A five-layer network, consisting of an input layer, a fuzzification layer, an inference layer, a defuzzification layer, and an output layer
Input layer	Receives the input data
Fuzzification layer	Converts the input data into fuzzy sets
Inference layer	Uses fuzzy logic to infer the output
Defuzzification layer	Converts the output of the inference layer into a crisp value
Output layer	Provides the output of the network
Training	The network is trained using a backpropagation algorithm
Applications	AFNIS can be used for a variety of applications, including: • Classification • Regression • Prediction • Control • Optimization

| Advantages |

AFNIS has a number of advantages, including:

- It is easy to understand and use
- It can be trained using a variety of data types
- It is able to handle nonlinear relationships
- It is robust to noise

| Disadvantages |

AFNIS also has a number of disadvantages, including:

- It can be computationally expensive to train
- It can be sensitive to the choice of parameters
- It can be difficult to interpret the results

Viva Questions:

1. What are hybrid systems?
A hybrid system is a dynamical system that exhibits both continuous and discrete dynamic behavior – a system that can both flow (described by a differential equation) and jump (described by a state machine or automaton)

2. What are fuzzy inference systems?
Fuzzy Inference Systems take inputs and process them based on the pre-specified rules to produce the outputs. Both the inputs and outputs are real valued, whereas the internal processing is based on fuzzy rules and fuzzy arithmetic.

3. How do neuro fuzzy inference systems work?
A neuro-fuzzy system is based on a fuzzy system which is trained by a learning algorithm derived from neural network theory. The learning procedure of a neuro-fuzzy system takes the semantical properties of the underlying fuzzy system into account. This results in constraints on the possible modifications applicable to the system parameters.

4. What is ANFIS architecture?
An adaptive neuro-fuzzy inference system or adaptive network-based fuzzy inference system (ANFIS) is a kind of artificial neural network that is based on Takagi–Sugeno fuzzy inference system. Since it integrates both neural networks and fuzzy logic principles, it has potential to capture the benefits of both in a single framework. Its inference system corresponds to a set of fuzzy IF–THEN rules that have learning capability to approximate nonlinear functions. Hence, ANFIS is considered to be a universal estimator

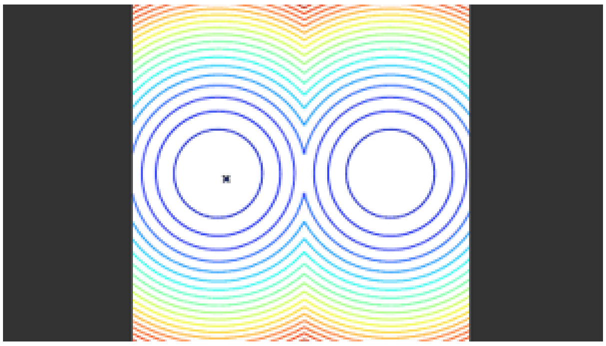
BEYOND THE SYLLABUS EXPERIMENTS

Experiment 1

Aim: Study of Derivative-free Optimization.

Theory:

Optimization Illustrations defined by functions for which derivatives are unavailable or available at a prohibitive cost are appearing more and more frequently in computational science and engineering. Increasing complexity in mathematical modelling, higher sophistication of scientific computing, and abundance of legacy codes are some of the reasons why derivative-free optimization is currently an area of great demand. Difficulties and challenges arise from multiple sources: expensive function evaluation, black-box/legacy codes, noise and uncertainty, unknown a priori function domains, and hidden constraints.



Derivative-free optimization (DFO) is a type of optimization problem in which the objective function is not differentiable. This can make the problem much more difficult to solve, as traditional optimization methods rely on derivative information to guide the search for the optimal solution.

There are a number of different DFO methods available, each with its own strengths and weaknesses. Some of the most common methods include:

- Direct search methods: These methods start with a random guess and then iteratively improve the solution by making local changes.
- Model-based methods: These methods build a model of the objective function and then use the model to guide the search for the optimal solution.
- Metaheuristic methods: These methods are based on heuristics, or rules of thumb, that have been shown to be effective in solving optimization problems.

The choice of DFO method depends on the specific problem being solved. Some factors to consider include the smoothness of the objective function, the number of constraints, and the availability of computing resources.
DFO has been used to solve a wide variety of problems, including:

- Engineering design
- Financial optimization
- Machine learning
- Robotics
- Scientific computing

In many cases, DFO has been shown to be more effective than traditional optimization methods, especially for problems with non-smooth objective functions or large numbers of constraints.

Here are some of the advantages of using DFO:

- It can be used to solve problems with non-smooth objective functions.
- It can be used to solve problems with large numbers of constraints.
- It is often more robust to noise than traditional optimization methods.

Here are some of the disadvantages of using DFO:

- It can be more time-consuming than traditional optimization methods.
- It can be more difficult to find the global optimum.
- It can be more sensitive to the choice of starting point.

Overall, DFO is a powerful tool that can be used to solve a wide variety of optimization problems. However, it is important to be aware of its limitations before using it.

Derivative free Optimization algorithm

- Genetic algorithms (GA)
- Simulated Annealing (SA)

Genetic algorithms (GA)

In the field of artificial intelligence, a genetic algorithm (GA) is a search heuristic that mimics the process of natural selection. This heuristic (also sometimes called a metaheuristic) is routinely used to generate useful solutions to optimization and search Illustrations.[1] Genetic algorithms belong to the larger class of evolutionary algorithms (EA), which generate solutions to optimization Illustrations using techniques inspired by natural evolution, such as inheritance, mutation, selection and crossover.

Optimization Illustrations

In a genetic algorithm, a population of candidate solutions (called individuals, creatures, or phenotypes) to an optimization Illustration is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible.[2]

The evolution usually starts from a population of randomly generated individuals, and is an iterative process, with the population in each iteration called a generation. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization Illustration being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

A typical genetic algorithm requires:
a genetic representation of the solution domain, a fitness function to evaluate the solution domain. A standard representation of each candidate solution is as an array of bits.[2] Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operations. Variable length representations may also be used, but crossover implementation is more complex in this case. Tree-like representations are explored in genetic programming and graph-form representations are explored in evolutionary programming; a mix of both linear chromosomes and trees is explored in gene expression programming.
Once the genetic representation and the fitness function are defined, a GA proceeds to initialize a population of solutions and then to improve it through repetitive application of the mutation, crossover, inversion and selection operators.

Initialization
The population size depends on the nature of the Illustration, but typically contains several hundreds or thousands of possible solutions. Often, the initial population is generated randomly, allowing the entire range of possible solutions (the search space). Occasionally, the solutions may be "seeded" in areas where optimal solutions are likely to be found.

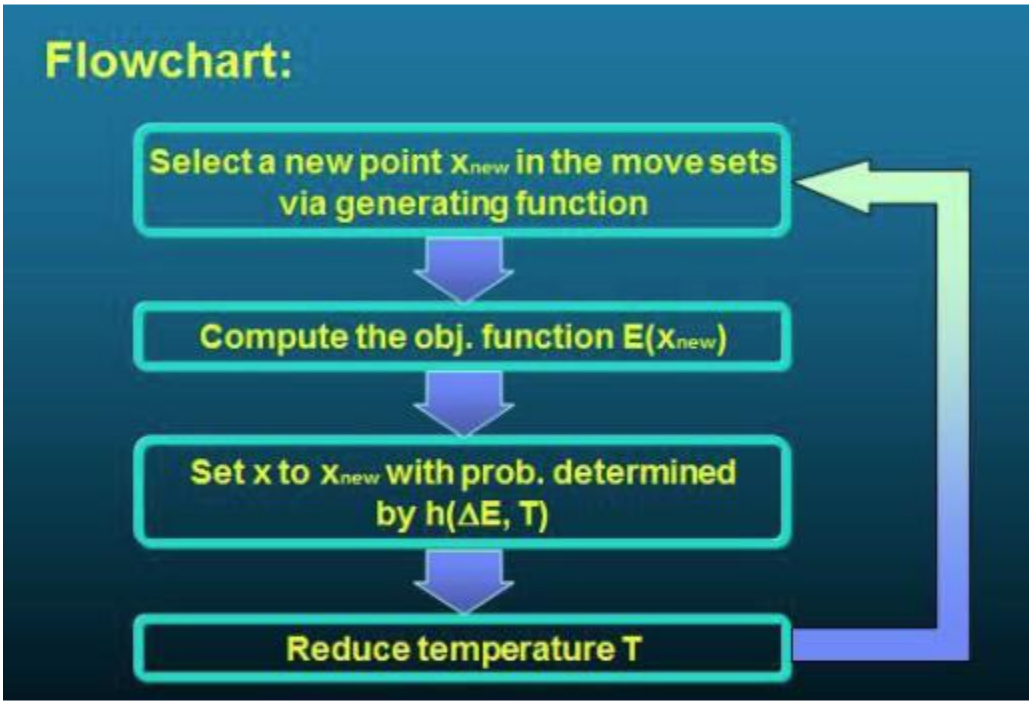
Selection
During each successive generation, a proportion of the existing population is selected to breed a new generation. Individual solutions are selected through a fitness-based process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected. Certain selection methods rate the fitness of each solution and preferentially select the best solutions. Other methods rate only a random sample of the population, as the former process may be very time-consuming.
The fitness function is defined over the genetic representation and measures the quality of the represented solution. The fitness function is always Illustration dependent. For instance, in the knapsack Illustration one wants to maximize the total value of objects that can be put in a knapsack of some fixed capacity. A representation of a solution might be an array of bits, where each bit represents a different object, and the value of the bit (0 or 1) represents whether or not the object is in the knapsack. Not every such representation is valid, as the size of objects may exceed the capacity of the knapsack. The fitness of the solution is the sum of values of all objects in the knapsack if the representation is valid, or 0 otherwise.
In some Illustrations, it is hard or even impossible to define the fitness expression; in these cases, a simulation may be used to determine the fitness function value of a phenotype (e.g. computational fluid dynamics is used to determine the air resistance of a vehicle whose shape is encoded as the phenotype), or even interactive genetic algorithms are used.

Simulated annealing (SA) is a probabilistic technique for approximating the global optimum of a given function. Specifically, it is a metaheuristic to approximate global optimization in a large search space. It is often used when the search space is discrete (e.g., all tours that visit a given set of cities). For Illustrations where finding the precise global optimum is less important than finding an acceptable local optimum in a fixed amount of time, simulated annealing may be preferable to alternatives such as brute-force search or gradient descent.
Simulated annealing interprets slow cooling as a slow decrease in the probability of accepting worse solutions as it explores the solution space. Accepting worse solutions is a fundamental property of metaheuristics because it allows for a more extensive search for the optimal solution.

Terminology:

1. Objective function E(x): function to be optiimized
2. Move set: set of next points to explore
3. Generating function: to select next point
4. Acceptance function h(DE, T): to determine if the selected point should be accept or not. Usually h(DE, T) = 1/(1+exp(DE/(cT)).
5. Annealing (cooling) schedule: schedule for reducing the temperature T

Flowchart:



Result/Conclusion:

This study experiments describe the various techniques used for derivative free optimization. It also describes how to use optimization techniques in soft computing domain.

Viva Questions:

1. What is the meaning of optimization?
Optimization is the process of finding the best possible solution to a problem. This can be done by finding the minimum or maximum value of a function, or by finding the best possible combination of variables.

2. What do you mean by simulated annealing?
Simulated annealing is a metaheuristic optimization algorithm that is based on the annealing process in metallurgy. The algorithm starts with a random solution and then iteratively improves it by making small changes. If the change results in a better solution, it is accepted. If the change results in a worse solution, it is accepted with a certain probability. The probability of accepting a worse solution decreases as the temperature of the system decreases.

3. What are the algorithms used for derivative free optimization?
There are many algorithms that can be used for derivative free optimization. Some of the most common algorithms include:

- Genetic algorithms
- Tabu search
- Simulated annealing
- Random search

4. What are the algorithms used for derivative based optimization?
There are also many algorithms that can be used for derivative based optimization. Some of the most common algorithms include:

- Gradient descent
- Newton's method
- Quasi-Newton methods
- Conjugate gradient methods

These are just a few of the many optimization algorithms that are available. The best algorithm to use for a particular problem will depend on the specific characteristics of the problem.

Experiment 2

Aim: Study of research paper on Soft Computing and give a review report consists of abstract, introduction, state of the art, methodology, results, conclusion, reference.

Theory:

Paper citation: Grossi, Enzo & Buscema, Massimo. (2008). Introduction to artificial neural networks. European journal of gastroenterology & hepatology. 19. 1046-54. 10.1097/MEG.0b013e3282f198a0.

Title: Introduction to Artificial Neural Networks
Authors: Enzo Grossi and Massimo Buscema
Published: 2008
Published in: European Journal of Gastroenterology & Hepatology
Volume: 19
Pages: 1046-1054
DOI: 10.1097/MEG.0b013e3282f198a0

Title:
Review of "Introduction to Artificial Neural Networks" by Enzo Grossi and Massimo Buscema

Abstract:
The paper titled "Introduction to Artificial Neural Networks" provides an overview of artificial neural networks (ANNs) and their applications in addressing complex problems. The authors emphasize the advantages of ANNs in healthcare and medicine, particularly in the field of gastroenterology. This review aims to summarize the key points covered in the paper, including the introduction to ANNs, their functioning principles, state of the art, methodology, results, and conclusions.

Introduction:
The paper introduces ANNs as intelligent agents inspired by the human brain, capable of adapting dynamically to complex problems. It highlights the ability of ANNs to model the dynamic interaction of multiple factors and draw individualized conclusions. The authors emphasize the advantages of ANNs over traditional statistical techniques, particularly in the context of healthcare and medicine.

State of the Art:
The authors discuss the increasing complexity of clinical data and the need for mathematical models to capture the key properties of the entire ensemble. They highlight the ubiquity of complexity and nonlinearity in living organisms and the importance of using ANNs to understand and analyze such systems. The paper emphasizes the need for new computational tools, including ANNs, to address the quantity and quality of medical information.

Methodology:
The paper describes the basic elements of ANNs, including nodes (processing elements) and connections. It explains how ANNs modify their internal structure through a learning process based on the environment's interactions. The authors discuss the topological arrangements of neurons within ANNs, with a focus on feedforward neural networks. They also mention the availability of specialized software for working with ANNs.

Results:
The paper highlights the pattern recognition-like abilities of ANNs, their robust classification capabilities, and their effectiveness in solving nonlinear problems. It emphasizes that ANNs can

extract valuable information from complex, dynamic, and multidimensional phenomena, which are often poorly predictable using traditional methods. The authors discuss the potential applications of ANNs in diagnosis, prognosis, and decision-making based on large and fuzzy input data.

Conclusion:
The authors conclude that ANNs offer specific advantages in addressing complex problems in healthcare and medicine. They emphasize the potential of ANNs to maximize the information derived from complex phenomena and provide individualized insights. The paper emphasizes the importance of ANNs as computational tools that can complement traditional statistical techniques and contribute to the advancement of medical science.

Key takeaways from the paper:

- ANNs are a powerful tool for medical diagnosis and prognosis.
- ANNs can learn complex relationships between input and output data.
- ANNs are not limited by the assumptions of linear regression.
- ANNs can be difficult to train, and they can be sensitive to the choice of parameters.

Summary:
The paper titled "Introduction to Artificial Neural Networks" provides a comprehensive overview of artificial neural networks (ANNs). The authors aim to introduce ANNs as a powerful computational tool for solving complex problems and highlight their applications in various fields, particularly in healthcare and medicine.
The paper begins by presenting the basic principles of ANNs, emphasizing their resemblance to the structure and functioning of the human brain. It explains how ANNs are composed of interconnected nodes called artificial neurons or perceptrons. These perceptrons receive input signals, perform mathematical operations, and generate output signals. The authors describe the importance of activation functions, which determine the output of a perceptron based on its inputs.
The authors then delve into the learning process of ANNs, focusing on two major types of learning algorithms: supervised learning and unsupervised learning. In supervised learning, ANNs are trained using input-output pairs, allowing them to learn patterns and make predictions. Unsupervised learning, on the other hand, involves training ANNs on unlabeled data to discover hidden structures and patterns.
Furthermore, the paper explores different types of neural network architectures, such as feedforward neural networks, recurrent neural networks, and self-organizing maps. It highlights the significance of the backpropagation algorithm for adjusting the weights of the network to minimize errors and improve performance.
The authors discuss the role of ANNs in healthcare and medicine, including applications in disease diagnosis, prognosis, and treatment prediction. They provide examples of studies that have successfully utilized ANNs for medical image analysis, prediction of patient outcomes, and drug discovery.
The authors highlight the advantages of ANNs in addressing complex and dynamic problems, particularly in the field of healthcare and medicine.
The paper emphasizes that ANNs are intelligent agents that can adapt dynamically to high complexity problems. ANNs have the ability to model the dynamic interaction of multiple factors simultaneously, enabling the study of complexity and individualized conclusions. They offer specific advantages compared to traditional statistical techniques.
The authors introduce ANNs as systems inspired by the functioning processes of the human brain. ANNs consist of interconnected nodes called processing elements (PEs) that receive input signals and generate output signals. The connections between nodes can modify themselves over time, leading to a learning process within the ANN. ANNs are particularly suitable for solving nonlinear problems and can reconstruct fuzzy rules governing optimal solutions.
The paper discusses the various topological arrangements of neurons within ANNs, with feedforward neural networks being the most common architecture. The learning process, facilitated by connection weights, allows ANNs to adapt to the data structure and understand the environment and its relations.

Furthermore, the paper emphasizes the application of ANNs in healthcare and medicine, particularly in the field of gastroenterology. ANNs offer a means to maximize the information derived from complex, dynamic, and multidimensional phenomena, which are often poorly predictable using traditional approaches. They can be used for diagnosis, prognosis, and decision-making based on large and fuzzy input data.

The authors highlight the pattern recognition-like abilities of ANNs, their robust classification capabilities, and their suitability for solving nonlinear problems. ANNs are described as valuable tools for understanding complex systems and capturing the rules that govern their behavior.

In summary, this paper provides an introduction to ANNs, their functioning principles, and their applications in healthcare and medicine. It emphasizes the advantages of ANNs in addressing complex problems and highlights their potential for advancing medical science.

Additionally, the paper touches upon some challenges associated with ANNs, including the selection of appropriate network architectures, avoidance of overfitting, and interpretability of network decisions. The authors emphasize the need for robust validation techniques and caution against the potential pitfalls of blindly relying on ANN predictions without domain knowledge.

In conclusion, the paper provides a comprehensive introduction to ANNs, covering their fundamental principles, learning algorithms, network architectures, and applications in healthcare and medicine. It serves as a valuable resource for researchers and practitioners interested in understanding and utilizing ANNs for solving complex problems in various domains.

Important Points:

- Artificial neural networks (ANNs) are a type of machine learning algorithm that are inspired by the human brain. They are made up of a series of interconnected nodes, each of which can represent a different feature of the input data. ANNs can be trained to learn complex relationships between the input and output data.
- ANNs have been used successfully in a variety of applications, including medical diagnosis, financial forecasting, and image recognition.
- In the field of medicine, ANNs have been used to diagnose diseases, predict patient outcomes, and develop new treatments.
- ANNs offer a number of advantages over traditional statistical methods. They are able to learn complex relationships between input and output data, and they are not limited by the assumptions of linear regression.
- However, ANNs also have some disadvantages. They can be difficult to train, and they can be sensitive to the choice of parameters.
- Despite these disadvantages, ANNs are a powerful tool that can be used to solve a wide range of problems.

Result/Conclusion: Through this experiment, we have understood the recent advancements and applications of various subdomains of soft computing