

EXPERIMENT NO-1

Aim:

Introduction to Prolog as AI programming language.

- a) Learning facts rules and built in predicates in Prolog. (Family Tree)
- b) Understanding recursion and backtracking in Prolog. (GCD, Factorial, Fibonacci Series).
- c) Program in List Processing. (8 queens).

Objectives:

To learn and understand the program by defining the necessary predicates that will make use of following procedures in Prolog:

1. Rules to infer facts in Prolog
2. Predicate Logic
3. Program code sections in Prolog
4. List Processing in Prolog.

Software Required : SWI-Prolog

Theory:

Prolog is a logical programming language. It is based on predicate calculus however it is restricted to allow only horn clauses.

Prolog is declarative: the program logic is expressed in terms of relations, represented as facts and rules. A computation is initiated by running a query over these relations.

Prolog is well-suited for specific tasks that benefit from rule-based logical queries such as searching databases, voice control systems, and filling templates.

Execution of Prolog program is effectively an application of theorem proving by first order resolution.

Rules and facts:

Prolog programs describe relations, defined by means of clauses. Pure Prolog is restricted to Horn clauses. There are two types of clauses: facts and rules.

Facts: Facts are nothing but knowledge that is already known. The fact can be described as symbolic relationship between different objects. In Prolog a database of facts and rules is supplied. The basic unit of Prolog is a predicate that is defined to be true. e.g. Father (john, bill)

Rules: Rules are used to infer the fact(s). They are also used to write and differentiate one fact from other facts. A rule is a conclusion that is known to be true if one or more other conclusions or facts are found to be true.

e.g. Father (x, y):-parent(x, y),male(x).

This means that if x is a parent of y and x is male then x is father of y.

RECURSION:

Recursion is a way of specifying something by reference to itself. More precisely, complicated instances are defined in terms of simpler instances and the simplest instance is specified explicitly.

Eg.

```
fact (N, R):- W is
            N-1, fact
            (w, Z), R
            is N*Z.
```

RECURSIVE RULES:

In a rule, if a relation calls itself, then it is called a recursive rule.

E.g

```
predecessor( X, Z ) :-
    parent(X, Z).
predecessor (X, Z):-
parent( X, Y), predecessor ( Y, Z).
```

List: - A list in Prolog is an ordered sequence of elements. Elements are terms, i.e. constants, variables, structures, including other lists.

List Syntax: the elements are enclosed in square brackets, separated by commas.

Examples:

List of mixed terms	:	[a, 2, c(3, d), Variable]
List of one kind of term	:	[atom1, atom2, atom3, atom4]
List of lists	:	[a, [b, c, [d, [e, f]], [g],1,3]
Empty list	:	[]

In Prolog, list split into two parts, the first part of the list known as head and second part known as the tail. We use symbol | (pronounced 'bar') to distinguish between head from tail.

Note: in head we have element of list and tail always have list.

[1 2, 3]	head = 1	tail = [2,3]
[1,2 3]	head = 1, 2	tail = [3]
[1]	head = 1	tail = []

Processing a List:

Elements in a list can be accessed through the head of the list. It is possible to extract more than one head element at a time.

List processing is a recursive because *either*: –you
 want to process the head of the list *or* –
 you want to process something in the tail.

Example:

1. List Member: Is given element present in list or not?

$member(X, [X|R]) .$

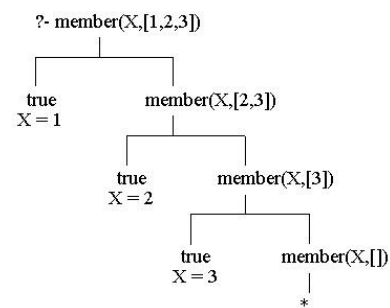
$member(X, [Y|R]) :- member(X, R) .$

One can read the clauses the following way, respectively:

- X is a member of a list whose first element is X.
- X is a member of a list whose tail is R if X is a member of R. Run:

$?- member(2, [1,2,3]) .$
 $true$

$?- member(X, [1,2,3]) .$
 $X = 1 ;$
 $X = 2 ;$
 $X = 3 ; false$



Member also can be written as

$member(X, [X|_]) .$

$member(X, [_|R]) :- member(X, R) .$ where '_' (underscore) designates a "don't-care" variable, usually called anonymous variables.

1. Delete: delete an element from list.

$del(X, [X|R], R) .$

$del(X, [F|R], [F|S]) :- del(X, R, S) .$

These clauses can be paraphrased in English as follows:

- When X is taken out of [X|R], R results.
- When X is taken out of the tail of [X|R], [X|S] results, where S is the result of taking X out of R. Run:

$?- del(2, [1,2,3], L) .$ $L=[2,3] .$

2. Permutation: delete an element from list.

$perm(List, [H|Perm]) :- del(H, List, Rest), perm(Rest, Perm) .$
 $perm([], []) .$

Run:

$?- perm([1,2,3], P) .$

```

P = [1,2,3] ;
P = [2,1,3] ;
P = [2,3,1] ;
P = [1,3,2] ;
P = [3,1,2] ; P
= [3,2,1] ;
false.

```

8-Queens Problem

Eight queens problem is a constraint satisfaction problem. The task is to place eight queens in the 8x8 matrix in such a way that no queen attacks each other.

Now a solution for this problem is to assign values for pair (x, y) where x = row number and y = column number such that the constraint is satisfied. The constraints are

1. No two queens should be in the same row
i.e $y_i \neq y_j$ for $i=1$ to $8; j=1$ to $8; i \neq j$
2. No two queens should be in the same column
i.e $x_i \neq x_j$ for $i=1$ to $8; j=1$ to $8; i \neq j$
3. There should not be two queens placed on the same diagonal line
i.e $(y_i - y_j) \neq \pm(x_i - x_j)$

Algorithm to solve eight queens problem

Start

Step 1: Represent the Queens vector, i.e., [1,2,3,4,5,6,7,8].

Step 2: Calculate the permutation of the above eight numbers stored in set P.

Step 3: Let the position where the first queen to be placed be (1,Y), for second be (2,Y1), and so on and store the position in S.

Step 4: Check for the safety of the queens through the predicate, 'noattack()'.

Step 5: Calculate $Y1-y$ and $Y-Y1$. If both are not equal to Xdist, which is the X-distance between the first queen and others, then goto step 6 else goto step 7.

Step 6: Increment Xdist by 1.

Step 7: Repeat above for the rest of the queens, until the end of the list is reached.

Step 8: Print S as answer.

Stop

CONCLUSION

Prolog is declarative. Prolog can make program design very like program specification. This makes rules concise and amenable to verification by inspection. Any assignment of variables is effected by unification. There are no explicit decisions or branches. There is virtually no logic to specify. This

style of specification is ideal development of a rule based on an expert system that allows real world rules to translate in fairly direct way to program rules. There are 92 solutions for 8 queens problem. Out of that only 12 solutions are unique. A solution to this puzzle can be represented as a special permutation of the list [1, 2, 3, 4, 5, 6, 7, 8] where each value indicates column position of every queen. Prolog provides the List data structure to facilitate this. To test whether a given permutation is a solution, one needs to calculate whether the permutation has two or more queens lie on the same diagonal. The representation itself prevents two or more queens in the same row or column.

Program Execution/formation/correction/ethical practices (07)	Documentation (02)	Timely Submission (03)	Viva Answer (03)	Experiment Marks (15)	Teacher Signature with date

EXPERIMENT 1

EXPERIMENT 1A: FAMILY TREE

Code:

```
male(sunil). male(ajay).  
male(swapnil).  
male(john). male(ram).  
female(sita).  
female(monica).  
female(rachel).  
female(phoebe).  
female(disha).  
parent_of(sunil,monica).  
parent_of(ajay,disha).  
parent_of(john,rachel).  
parent_of(ram,phoebe) .  
parent_of(sunil,ajay) .  
parent_of(ram,monica) .  
parent_of(john,swapnil) .  
parent_of(disha,sita) .
```

```
/*Rules*/ father_of(X,Y)  
:- male(X),  
parent_of(X,Y) .  
mother_of(X,Y) :- female(X),  
parent_of(X,Y) .  
grandfather_of(X,Y) :- male(X),  
parent_of(X,Z),  
parent_of(Z,Y).  
grandmother_of(X,Y) :- female(X),  
parent_of(X,Z),  
parent_of(Z,Y).  
sister_of(X,Y):- female(X),  
father_of(F, Y), father_of(F,X),X \= Y.  
sister_of(X,Y):- female(X),  
mother_of(M, Y),  
mother_of(M,X),X \= Y.  
aunt_of(X,Y):- female(X),
```

```

parent_of(Z,Y), sister_of(Z,X),!.
brother_of(X,Y):-
    %(X,YorY,X)%male(X),
    father_of(F, Y), father_of(F,X),X \= Y.
brother_of(X,Y):- male(X), mother_of(M,
Y), mother_of(M,X),X \= Y.
uncle_of(X,Y):- parent_of(Z,Y),
brother_of(Z,X).
ancestor_of(X,Y):- parent_of(X,Y).
ancestor_of(X,Y):- parent_of(X,Z),
ancestor_of(Z,Y).

```

Output :-

```

?- father_of(sunil,monica).
true .

?- sister_of(disha,phoebe).
false .

?- brother_of(rachel,swapnil).
false .

?- brother_of(swapnil,rachel).
true

```

EXPERIMENT 1B : FACTORIAL

Code:

```

fact(0,1). fact(N,F):-
(
    N>0 -
    >(
        N1 is N-1, fact(N1,F1),
        F is N*F1
    )
;
    N<0 -
    >(
        N1 is N+1,
        fact(N1,F1), F
        is N*F1
    )
)

```

)).

Output :-

```
SWI-Prolog (AMD64, Multi-threaded, version 8.3.5)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 8.3.5)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?-
% c:/Users/Shivani/Desktop/AISC practicals/EXP1B/factorial.pl compiled 0.00 sec, 2 clauses
?- fact(3,R).
R = 6 .
?- fact(5,R).
R = 120 .
?- fact(-3,R).
R = -6
```

EXPERIMENT 1B : GCD

Code:

$\text{gcd}(X,0,X).$
 $\text{gcd}(X,Y,Z):- R \text{ is mod}(X,Y),$
 $\text{gcd}(Y,R,Z).$

Output :-

```
?- gcd(12,6,X).
X = 6 .
?- gcd(15,6,X).
X = 3 .
?- gcd(12,13,X).
X = 1
```


EXPERIMENT 1B : LCM

Code:

```
gcd(X, Y, G) :- X = Y, G = X.  
gcd(X, Y, G): G:- X < Y, Y1 is Y-X,  
gcd(X, Y1, G).  
gcd(X, Y, G) :- X > Y, gcd(Y, X, G).  
lcm(X, Y, LCM):-gcd(X, Y, GCD), LCM is X*Y//GCD
```

```
?- lcm(3, 4, M).  
M = 12 .  
?- lcm(5, 6, M).  
M = 30 .  
?- lcm(8, 2, M).  
M = 8
```

EXPERIMENT 1C: 8 QUEENS

Code:

```
:- use_rendering(chess). queens(N,  
  
Queens) :- length(Queens, N),  
  
board(Queens, Board, 0, N, _, _),  
  
queens(Board, 0, Queens).  
  
board([], [], N, N, _, _).  
board([|Queens|, [Col-Vars|Board], Col0, N, [|VR|, VC]):-  
  
Col is Col0+1, functor(Vars, f, N), constraints(N, Vars,  
  
VR, VC),  
  
board(Queens, Board, Col, N, VR, [|_|VC]).
```

```
constraints(0, _, _) :- !.  
constraints(N, Row, [R|Rs], [C|Cs]) :-  
    arg(N, Row, R-C), M is N-1,  
  
    constraints(M, Row, Rs, Cs).
```

```
queens([], _, []).  
queens([C|Cs], Row0, [Col|Solution]) :-  
    Row is Row0+1, select(Col-Vars,  
    [C|Cs], Board), arg(Row, Vars, Row-  
    Row), queens(Board, Row, Solution)
```

Output

```
?- queens(8, Queens).  
Queens = [1, 5, 8, 6, 3, 7, 2, 4] ■
```

EXPERIMENT NO-2

Aim:

Implement logical gates using McCulloch pitts neuron model.

Theory:

McCulloch and Pitts tried to understand how the brain could produce highly complex patterns by using many basic cells that are connected together. These basic brain cells are called neurons, and McCulloch and Pitts gave a highly simplified model of a neuron . A group of MCP neurons that are connected together is called an **artificial neural network**. In a sense, the brain is a very large neural network. It has billions of neurons, and each neuron is connected to thousands of other neurons. McCulloch and Pitts showed how to encode any logical proposition by an appropriate network of MCP neurons. And so in theory anything that can be done with a computer can also be done with a network of MCP neurons. McCulloch and Pitts also showed that every network of MCP neurons encodes some logical proposition. So if the brain were a neural network, then it would encode some complicated computer program. But the MCP neuron is not a real neuron; it's only a highly simplified model.

Examples:

Suppose there is a neuron in a bird's brain that has two receivers, which are connected somehow to the bird's eyes. If the bird sees a round object, a signal is sent to the first receiver. But if any other shape is seen, no signal is sent. So the first receiver is a roundness detector. If the bird sees a purple object, a signal is sent to the second receiver of the neuron. But if the object is any other color, then no signal is sent. So the second receiver is a purple detector. Notice that for either receiver there is a question that can be answered "yes" or "no," and a signal is only sent if the answer is "yes." The first receiver corresponds to the question "Is the object round?" The second receiver corresponds to the question "Is the object purple?" We would like to produce an MCP neuron that will tell the bird to eat a blueberry, but to avoid eating red berries or purple violets. In other words, we want the MCP neuron to send an "eat" signal if the object is both round and purple, but the MCP neuron will send no signal if the object is either not round or not purple, or neither round nor purple. So the bird will only eat an object if the MCP neuron sends a signal. If no signal is sent, then the bird will not eat the object. Here is a table that summarizes how the MCP neuron would work in several cases.

Object	Purple?	Round?	Eat?
Blueberry	Yes	Yes	Yes
Golf ball	No	Yes	No
Violet	Yes	No	No

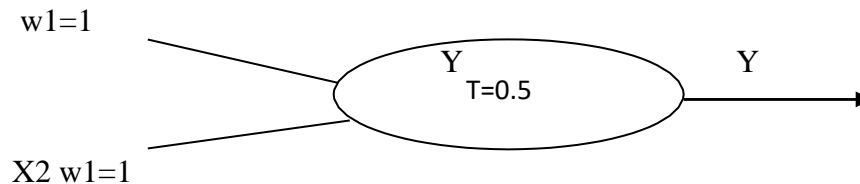
Hot Dog	No	No	No
---------	----	----	----

Notice that all the signals sent to the MCP neuron and the signal that it sends out are all "yes" or "no" signals. This "all or nothing" feature is one of the assumptions that McCulloch and Pitts made about the workings of a real neuron. They also assumed that somehow a real neuron "adds" the signals from all its receivers, and it decides whether to send out a "yes" or "no" signal based on the total of the signals it receives. If the total of the received signals is high enough, the neuron sends out a "yes" signal; otherwise, the neuron sends a "no" signal. In order to "add" the signals that the MCP neuron is receiving, we will use the number 1 for a "yes" and the number 0 for a "no." Then Table 1 will now look like this.

1. OR GATE The OR gate is a digital logic gate that implements logical disjunction-it behaves according to the truth table. Table -1: Truth table

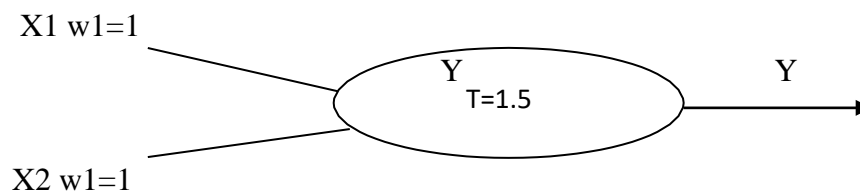
X1	X2		Y	E
0	0	0	0	0
0	1	1	1	1
1	0	1	1	1
1	1	2	1	1

Fig -2: OR Gate IMPLIMENTATION OF MCCULLOCH PITTS X1



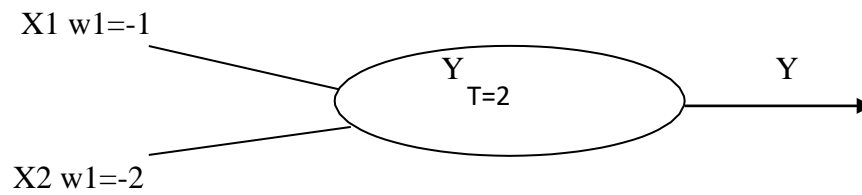
2. AND GATE-

X1	X2		Y	E
0	0	0	0	0
0	1	1	0	0
1	0	1	0	0
1	1	2	1	1



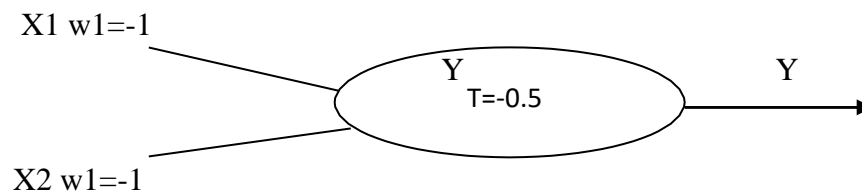
3. NAND Gate

X1	X2		Y	E
0	0	0	1	1
0	1	-2	1	1
1	0	-1	1	1
1	1	-3	0	0



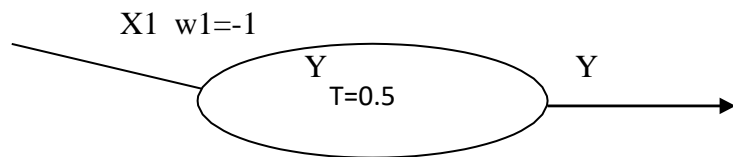
4. NOR Gate

X1	X2		Y	E
0	0	0	1	1
0	1	-1	0	0
1	0	-1	0	0
1	1	-2	0	0



5. NOT Gate

X1		Y	E
0	0	1	1
0	-1	0	0



6. XOR Gate

X1	X2		Y1		Y2		Y3	
0	0	0	0	0	1	1	0	0
0	1	-1	1	-2	1	2	1	1
1	0	-1	1	-5	1	2	1	1
1	1	-2	1	-7	0	1	0	1

CONCLUSION

Hence we have implemented OR,NOT,XOR,AND and NAND logic gates using mc llutoch pitts neuron model and shown how to encode any logical proposition by an appropriate network of MCP neurons.

Program Execution/formation/correction/ethical practices (07)	Documentation (02)	Timely Submission (03)	Viva Answer (03)	Experiment Marks (15)	Teacher Signature with date

NOT using MCCulloch pitts model :- Code:

```
clear all;
disp("Enter weights : ");
w1=input("Enter w1 : ");
disp("Enter Threshold : ");
theta=input("theta : ");
y=[0 0];
x1=[0 1];
z=[1 0];
con=1;
while con zin=x1*w1;
for i=1:2 if zin(i)>=theta
y(i)=0; else y(i)=1; end
end
disp("Output of net is : ");
disp(y);
if y==z con=0;
else
disp("Enter another set of Weights and Threshold");
w1=input("Enter Weight w1 : ");
theta=input("Enter theta : ");
end
end
disp("McCulloch pitts net for NOT");
disp("Weight of neuron : ");
disp(w1);
disp("theta : ");
disp(theta);
```

Output:

```

    "Enter weights : "
Enter w1 : 1

    "Enter Threshold : "
theta : 0

    "Output of net is : "

    0.    0.

    "Enter another set of Weights and Threshold"
Enter Weight w1 : 1
Enter theta : 1

    "Output of net is : "

    1.    0.

    "McCulloch pitts net for NOT"

    "Weight of neuron : "

    1.

    "theta : "

    1.

```

Or using using MCCulloch pitts model :- Code:

```

clear all;
disp("Enter weights :");
w1=input("Enter weight w1: ");
w2=input("Enter weight w2: ");
disp("Enter threshold : ");
theta=input("theta =");
y=[0 0 0 0];

```



```

x1=[0 0 1 1];
x2=[0 1 0 1];
z=[0 1 1 1];
con=1;
while con zin=x1*w1+x2*w2;
for i=1:4
    if zin(i)>=theta y(i)=1;
    else y(i)=0;
    end
end
disp("Output of net is :
");
disp(y); if y==z con=0;
else
disp("Enter another set of weights and threshold");
w1=input("Enter Weight w1: ");
w2=input("Enter Weight w2: ");
theta=input("theta =");
end
end
disp("McCulloch pitts net for OR Function");
disp("Weights of neuron are :");
disp(w1);
disp(w2);
disp("theta=");
disp(theta);

```

Output:

```

    "Enter weights : "
Enter weight w1: 1

Enter weight w2: 0

    "Enter threshold : "
theta =1

    "Output of net is : "

    0.    0.    1.    1.

    "Enter another set of weights and threshold"
Enter Weight w1: 1

Enter Weight w2: 1

theta =1

    "Output of net is : "

    0.    1.    1.    1.

    "McCulloch pitts net for OR Function"

    "Weights of neuron are : "

    1.

    1.

    "theta="

    1.

```

AND function using MCCulloch pitts model

Code:

```

clear all;
disp("Enter weights : ");
w1=input("Enter Weight w1: ");
w2=input("Enter Weight w2: ");
disp("Enter threshold: ");
theta=input("theta=");
y=[0 0 0 0];
x1=[0 0 1 1];
x2=[0 1 0 1];
z=[0 0 0 1];
con=1;
while con zin=x1*w1+x2*w2;
for i=1:4
if zin(i)>=theta y(i)=1;
else
y(i)=0;
end
end
disp("Output of net is :
");
disp(y);
if y==z con=0;
else
disp("Enter another set of weights and threshold");
w1=input("Enter Weight w1: ");
w2=input("Enter Weight w2: ");
theta=input("theta=");

end
end

disp("McCulloch pitts net for AND Function");
disp("Weights of neuron are : ");

disp(w1);
disp(w2);
disp("theta=");
disp(theta);

```

```

    "Enter weights : "
Enter Weight w1: 1

Enter Weight w2: 0

    "Enter threshold: "
theta=1

    "Output of net is : "

    0.    0.    1.    1.

    "Enter another set of weights and threshold"
Enter Weight w1: 1

Enter Weight w2: 1

theta=2

    "Output of net is : "

    0.    0.    0.    1.

    "McCulloch pitts net for AND Function"

    "Weights of neuron are : "

    1.

    1.

    "theta="

    2.

```

AndNot Function using MCCulloch pitts model :- Code:

```

clear all; disp("Enter
weights ");
w1=input("Enter w1: ");

```

```

w2=input("Enter w : ");
disp("Enter Threshold: ");
theta=input("theta=");
y=[0 0 0 0]; x1=[0 0 1 1];
x2=[0 1 0 1]; z=[0 0 1 0];
con=1; while con
zin=x1*w1+x2*w2;
for i=1:4 if
zin(i)>=theta
y(i)=1; else
y(i)=0; end
end
disp("Output of net is : ");
disp(y); if y==z con=0;
else
disp("Enter another set of weights and threshold");
w1=input("Enter w1: "); w2=input("Enter w2: ");
theta=input("Enter theta: ");
end
end
disp("McCulloch pitts net for ANDNOT Function");
disp("Weights of neuron are:"); disp(w1);
disp(w2); disp("theta : "); disp(theta);

```

```
"Enter weights "  
Enter w1: -1  
  
Enter w : -1  
  
"Enter Threshold: "  
theta=1  
  
"Output of net is : "  
  
0.    0.    0.    0.  
  
"Enter another set of weights and threshold"  
Enter w1: 1  
  
Enter w2: -1  
  
Enter theta: 1  
  
"Output of net is : "  
  
0.    0.    1.    0.  
  
"McCulloch pitts net for ANDNOT Function"  
  
"Weights of neuron are:"  
  
1.  
  
-1.  
  
"theta : "  
  
1.
```

Nor using MCCulloch pitts model :-

Code:

```
clear all;
disp("Enter weights: ");
w1=input("Enter Weight w1: ");
w2=input("Enter Weight w2: ");
disp("Enter threshold : ");
theta=input("theta="); y=[0 0 0
0]; x1=[0 0 1 1]; x2=[0 1 0 1];
z=[1 0 0 0]; con=1; while con
zin=x1*w1+x2*w2;
for i=1:4 if zin(i)>=theta
y(i)=0; else y(i)=1; end
end disp("Output of net
is : "); disp(y); if y==z
con=0; else
disp("Enter another set of weights and threshold");
w1=input("Enter Weight w1: "); w2=input("Enter
Weight w2: "); theta=input("theta="); end end
disp("McCulloch pitts net for NOR function");
disp("Weights of neuron are:"); disp(w1);
disp(w2); disp("theta="); disp(theta);
```

```

Enter Weight w1: 2
Enter Weight w2: 1
theta=2

"Output of net is : "

1. 1. 0. 0.

"Enter another set of weights and threshold"
Enter Weight w1: 2
Enter Weight w2: 2
theta=2

"Output of net is : "

1. 0. 0. 0.

"McCulloch pitts net for NOR function"

"Weights of neuron are:"

2.

2.

"theta="

2.

```

XOR function using MCCulloch pitts model :- Code:


```

clear all; disp("Enter weights : ");
w11=input("Enter Weight w11 : ");
w12=input("Enter Weight w12 : ");
w21=input("Enter Weight w21 : ");
w22=input("Enter Weight w22 : ");
v1=input("Enter Weight v1 : ");
v2=input("Enter Weight v2 : ");
disp("Enter threshold : ");
theta=input("theta="); y1=[0 0 0
0]; y2=[0 0 0 0]; x1=[0 0 1 1];
x2=[0 1 0 1]; z=[0 1 1 0]; con=1;
while con zin1=x1*w11+x2*w12;
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
is : "); disp(y); if y==z
con=0; else
disp("Enter another set of weights and threshold");
w11=input("Enter Weight w11 : ");
w12=input("Enter Weight w12 : ");
w21=input("Enter Weight w21 : ");
w22=input("Enter Weight w22 : "); v1=input("v1 :
");

```

```

v2=input("v2 : "); theta=input("theta
= ");
end end
disp("McCulloch pitts net for XOR function");
disp("Weights of neuron Z1 : ");
disp(w11); disp(w12);
disp("Weights of neuron Z2 : ");
disp(w21); disp(w22);
disp("Weights of neuron Y : ");
disp(v1); disp(v2);
disp("theta = "); disp(theta);

```

Output:

```

--> exec('C:\Users\riya amit manek\Documents\Xor.sce', -1)

    "Enter weights : "
Enter Weight w11 : 1

Enter Weight w12 : 1

Enter Weight w21 : -1

Enter Weight w22 : 1

Enter Weight v1 : 1

Enter Weight v2 : 1

    "Enter threshold : "
theta=1

    "Output of net is : "

    0.    1.    1.    1.

```

"Enter another set of weights and threshold"

Enter Weight w11 : 1

Enter Weight w12 : -1

Enter Weight w21 : -1

Enter Weight w22 : 1

v1 : 1

v2 : 1

theta = 1

"Output of net is : "

0. 1. 1. 0.

"McCulloch pitts net for XOR function"

"Weights of neuron Z1 : "

1.

-1.

"Weights of neuron Z2 : "

-1.

1.

"Weights of neuron Y : "

1.

1.

"theta = "

1.

EXPERIMENT NO-3

Aim:

- Identify an AI problem
- Give PEAS description for the problem
- Give problem Formulation for it

Theory:

N-Puzzle or sliding puzzle is a popular puzzle that consists of N tiles where N can be 8, 15, 24 and so on. In our example $N = 8$. The puzzle is divided into $\sqrt{N+1}$ rows and $\sqrt{N+1}$ columns. Eg. 15Puzzle will have 4 rows and 4 columns and an 8-Puzzle will have 3 rows and 3 columns. The puzzle consists of N tiles and one empty space where the tiles can be moved. Start and Goal configurations (also called state) of the puzzle are provided. The puzzle can be solved by moving the tiles one by one in the single empty space and thus achieving the Goal configuration.

Initial State			Goal State		
1	2	3	2	8	1
8		4		4	3
7	6	5	7	6	5

Figure: Start and Goal configurations of an 8-Puzzle.

The tiles in the initial(start) state can be moved in the empty space in a particular order and thus achieve the goal state.

PEAS description

PEAS stands for Performance Measures, Environment, Actuators, and Sensors.

Performance Measure: If the objective function to judge the performance of the agent. For example, in case of pick and place robot, no of correct parts in a bin can be the performance measure.

Environment: It the real environment where the agent need to deliberate actions.

Actuators: These are the tools, equipment or organs using which agent performs actions in the environment. This works as output of the agent.

Sensors: These are tools, organs using which agent captures the state of the environment. This works as input to the agent.

PEAS description for 8 Puzzle Problem:

Performance measure:

Each step cost 1, so the path cost is the number of steps in the path.

Environment:

3 X 3 grid having 8 tiles and one blank space Each tile is numbered 1 to 8 We can move only tiles adjacent to blank space.

Actuators:

Motors for moving the blank tiles to appropriate position.

Sensors:

Camera for finding blank tiles and reading the tile numbers.

Problem Formulation:

A problem can be defined formally by 4 components:

1. Initial State:

- It is the state from which our agents start solving the problem {e.i: $in(A)$ }.

2. State Description:

- a description of the possible actions available to the agent, it is common to describe it by means of a successor function, given state x then $SUCCESSOR-FN(x)$ returns a set of ordered pairs $\langle action, successor \rangle$ where action is a legal action from state x and successor is the state in which we can be by applying action.
- The initial state and the successor function together defined what is called state space which is the set of all possible states reachable from the initial state {e.i: $in(A)$, $in(B)$, $in(C)$, $in(D)$, $in(E)$ }.

3. Goal Test:

- we should be able to decide whether the current state is a goal state {e.i: is the current state is $in(E)$?}.

4. Path cost:

- A function that assigns a numeric value to each path, each step we take in solving the problem should be somehow weighted, so If I travel from A to E our agent will pass by many cities, the cost to travel between two consecutive cities should have some cost measure, {e.i: Traveling from 'A' to 'B' costs 20 km or it can be typed as $c(A, 20, B)$ }.

A solution to a problem is path from the initial state to a goal state, and solution quality is measured by the path cost, and the optimal solution has the lowest path cost among all possible solutions.

Problem formulation for 8 Puzzle Problem:

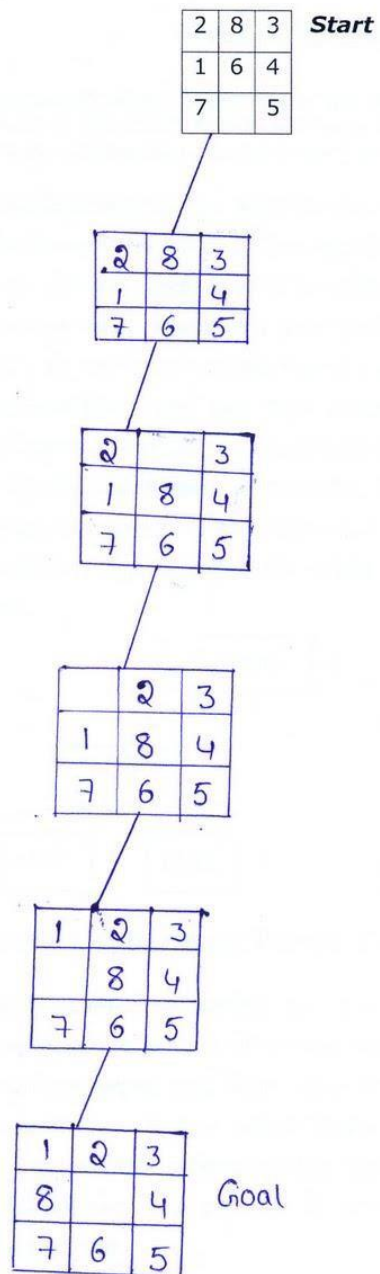
States: Description of the location of each of the eight tiles and (for efficiency) the blank square.

Initial State: Initial configuration of the puzzle.

Actions or Successor function: Moving the blank left, right, up, or down.

Goal Test: Does the state match the configuration on the right (or any other configuration)?

Path Costs: Each step costs 1 unit (path costs corresponds to its length).



Search tree for 8-puzzle

CONCLUSION: Different problems can be solved such as puzzle problem and its PEAS description can be generated

Program Execution/ formation/ correction/ ethical practices (07)	Documentation (02)	Timely Submission (03)	Viva Answer (03)	Experiment Marks (15)	Teacher Signature with date

Experiment No.3

1. Vacuum Cleaner Agent

- **Performance:** Performance can be evaluated on the basis of how well the cleaning work is performed by the agent.
- **Environment:** The agent works in a closed environment where the only focus is on the tile and the dirt present on it.
- **Actuators:** The devices that the agent uses are the motor legs to move to the desired tile block and the cleaning devices used to clean a dirty tile.
- **Sensors:** Dirt sensor that can be accumulation of many other sensors and also obstacle detection sensor for the agent to move properly.

2. English Language Tutor:

- **Performance:** Performance of the agent is evaluated on the basis of how well the language is understood by the students.
- **Environment:** The agent would work in the classroom where there would be benches, board and students too in the environment.
- **Actuators:** Test score evaluators and speech to text convertors and result generation comes under actuators.
- **Sensors:** The scanner to scan the test paper and the speech to text convertor for voice based analysis (speaking skills).

3. Part Picking Robot:

- **Performance:** The performance of this agent can be evaluated on the basis of the perfection with which the different parts are picked.
- **Environment:** The environment will consists of the factory and the conveyor belt to pick the parts.
- **Actuators:** The part picking robotic arm, sensor to determine the orientation of the object.
- **Sensors:** A sensor will be used to check the orientation of the object and a camera might be used in order to see the image and recognize the part to be picked.

4. Text Summarization Agent:

- **Performance:** The performance can be evaluated on the criteria that whether an important part of the text included in the summarization or not.
- **Environment:** The Agent will work in a digital environment with the different document types for getting the text and summarizing it.
- **Actuators:** The read/write head or pointer to parse through the entire document and the algorithm to summarize the text. ○ **Sensors:** A camera can be used to read the text on real life objects and different documents will be the input to it.

5. Career Guidance Agent:

- **Performance:** The performance is evaluated by checking with the students whether their choice and the career guidance predicted are same or not.
- **Environment:** The agent will deal with various kinds of data regarding the student and also the details or a summary about a student.

- **Actuators:** The parameter evaluators and the data extractors will act as an actuator and the predicted output will also act as an actuator.
- **Sensors:** The dataset of student having various entries for all the parameters is the input to the agent.

6. **Admission Taking Guide:**

- **Performance:** The performance can be evaluated on the basis of the student getting the admission in the correct Engineering College as told by the agent. ○
- **Environment:** The agent deals with the scores obtained by the student in the entrance exam and the 12th standard exams or the diploma scores.
- **Actuators:** The eligibility criteria given by the DTE and then the score checker and the cut-off comparison maker acts as an actuator.
- **Sensors:** The dataset of a student having the entries for marks for the needs mentioned above.

EXPERIMENT NO-4

Aim:

Implement search algorithm to reach goal state.

- a) Depth Limited Search
- b) A* Algorithm

Objectives:

- ✦ To formulate a given a problem description in terms of a state space search problem.
- ✦ To implement the uninformed or blind search strategies to find the solution for a given problem.
- ✦ To analyze the properties of these algorithms in terms of time complexity, space complexity, termination and optimality.

- ✦ Given a problem description, to draw the state space representation of the problem.
- ✦ To select a suitable heuristics for the given problem.
- ✦ To implement an A*(informed search) algorithm to find an optimal path from initial state to goal state.

Theory:

Uninformed search means that there is no additional information about the states beyond that provided by the problem definition. All that can be done is to generate successors and distinguish a goal state from a non-goal state. There are many uninformed search strategies. All search strategies are distinguished by the order in which nodes are expanded. They are- Breadth First Search

- Uniform cost Search
- Depth First Search
- Backtracking DFS
- Depth Limited Search
- Iterative Deepening DFS
- Bidirectional search

Depth-limited search (DLS)

- **Description:**

- The unbounded tree problem appeared in DFS can be fixed by imposing a limit on the depth that DFS can reach, this limit we will call depth limit l , this solves the infinite path problem.

- **Performance Measure:**

- Completeness:
 - ✦ The limited path introduces another problem which is the case when we choose $l < d$, in which is our DLS will never reach a goal, in this case we can say that DLS is not complete.
- Optimality:
 - ✦ One can view DFS as a special case of the depth DLS, that DFS is DLS with $l = \text{infinity}$.

- ✦ DLS is not optimal even if $l > d$.
- Time Complexity: $O(b^l)$
- Space Complexity: $O(b^l)$
- **Conclusion:**
 - DLS can be used when there is a prior knowledge to the problem, which is always not the case. Typically, we will not know the depth of the shallowest goal of a problem unless we solved this problem before.

LOGIC:

The Example is of 'Water Jug Problem'.

E.g. Given a full 5-gallon jug and an empty 2-gallon jug, the goal is to fill the 2-gallon jug with exactly one gallon of water.

State = (x,y) , where x is the number of gallons of water in the 5-gallon jug and y is # of gallons in the 2-gallon jug

Initial State = $(5,0)$

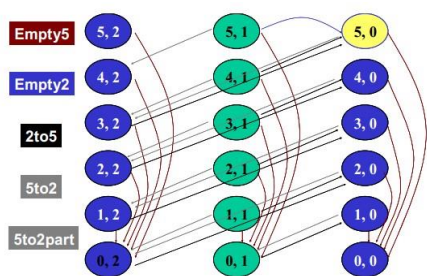
Goal State = $(*,1)$, where $*$ means any amount

Solution:

The table below shows the different operators and their effects

NAME	CONDITION	TRANSITION	EFFECT
Empty5	-	$(x,y) \rightarrow (0,y)$	Empty 5-gal. jug
Empty2	-	$(x,y) \rightarrow (x,0)$	Empty 2-gal. jug
2to5	$x \leq 3$	$(x,2) \rightarrow (x+2,0)$	Pour 2-gal. into 5-gal.
5to2	$x \geq 2$	$(x,0) \rightarrow (x-2,2)$	Pour 5-gal. into 2-gal.
5to2part	$y < 2$	$(1,y) \rightarrow (0,y+1)$	Pour partial 5-gal. into 2-gal.

The figure below shows the different states and the transitions between the states using the operators above. The transitions corresponding to Empty2 have not been marked to keep the figure clean.



$(5,0)$ is the initial state.

$(0,1)$ is the goal state.

A solution to this problem is given by the path

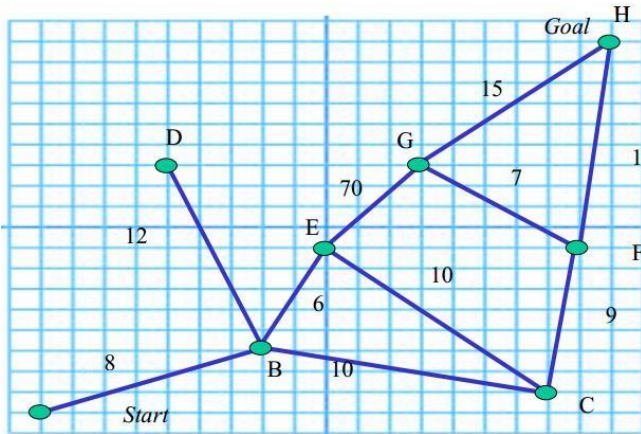
$(5,0) - (3,2) - (3,0) - (1,2) - (1,0) - (0,1)$.

Informed (Heuristics) search methods use problem specific knowledge, and may be more efficient. Heuristic means "rule of thumb". Heuristics are used to identify the most promising search path.

A heuristic function $h(n)$ = estimated cost of the cheapest path from node n to a goal node **EXAMPLE:**

A* Search Technique

A* is a best first search algorithm with $f(n) = g(n) + h(n)$ where $g(n)$ = sum of edge costs from start to n $h(n)$ = estimate of lowest cost path from n $h(n)$ is said to be admissible if it underestimates the cost of any solution that can be reached from n. If $C^*(n)$ is the cost of the cheapest solution from n to a goal and if $h(n)$ is admissible then $h(n) \leq C^*(n)$. If $h(n)$ is admissible, then the search will find an optimal solution.



The heuristics function used is the straight line distance

LOGIC:

Algorithm A*

OPEN = nodes on frontier. CLOSED = expanded nodes.

OPEN = {<s, NIL>}

while OPEN is not empty place <n,p> remove from open the node <n,p> with minimum $f(n)$ place <n, p> on CLOSED if n is a goal node, ss (path p) return success (path p)

for each edge connecting n and m with cost c

if <m, q> is on CLOSED and {p|e} is cheaper than q LOSED,
then remove n from C|e}>on OPEN put <m,{p|e}> on OPEN

else if <m,q> is on OPEN and {p|e} is cheaper than q then
replace q with {p|e}

else if m is not on OPEN then
put <m,{p|e}>on OPEN

SOLUTION

The order of nodes expanded, and the status of Fringe is shown in the following table.

	Fringe	Node expanded	Comments
1	A		
2	B(26.6)	A	
3	E(27.5), C(35.1), D(35.2)	B	

4	C(35.1), D(35.2), C(41.2) , G(92.5)	E	C is not inserted as there is another C with lower cost
5	D(35.2), F(37), G(92.5)	C	
6	F(37), G(92.5)	D	
7	H(39), G(42.5)	F	G is replaced with a lower cost node
8	G(42.5)	H	Goal test successful

CONCLUSION:

Depth search is not appropriate. The search space is a graph. The search tree has many repeated states. Breadth first search is appropriate

For informed search technique A* - The path returned is A-B-C-F-H. The path cost is 39. This is an optimal path

Program Execution/formation/correction/ethical practices (07)	Documentation (02)	Timely Submission (03)	Viva Answer (03)	Experiment Marks (15)	Teacher Signature with date

DFID :-

Code:

```
from collections import defaultdict

r=[]

class Graph:

    def __init__(self,vertices):

        self.v = vertices    self.graph

        = defaultdict(list) def

        addEdge(self,u,v):

            self.graph[u].append(v) def

        DLS(self,src,target,maxDepth):

            if src == target :

                r.append(src)

                return True if

                maxDepth <= 0 :

                return False for i in

                self.graph[src]:

                    if(self.DLS(i,target,maxDepth-1)):

                        r.append(src) return True

            return False def IDDFS(self,src,

            target, maxDepth): for i in

            range(maxDepth):

                if (self.DLS(src, target, i)):

                    return True

            return False

node = int(input("Number of nodes:

")) g = Graph(node) print(" Enter

Graph nodes: ")
```

```

for i in range (node-1):
    a,b=map(str, input().split())
    g.addEdge(a,b)
target = (input("Specify Target node: "))
maxDepth = int(input("Max depth of the tree is: "))
src = (input("Enter Source node: "))
if g.IDDFS(src, target, maxDepth) == True:
    print ("Target can be reached !!")
    print("For reaching Target Path is: ",end="")
    for i in range(len(r)-1,-1,-1):
        print(r[i],end="")
    else :
        print ("Target is Not Reachable")

```

Output:

```

Number of nodes: 12
Enter Graph nodes:
a b
a c
b d
b e
c f
c g
d i
d h
e j
f l
e k
Specify Target node: j
Max depth of the tree is: 4
Enter Source node: a
Target can be reached !!
For reaching Target Path is: abej
>>> |

```

A* :- Code:

```

m_tree = {'S': [['A', 1], ['B', 7], ['C', 4]],
'A': [['S', 1], ['D', 2], ['E', 9], ['G', 3]],

```



```

'B': [['S', 3], ['G', 4]],
'C': [['S', 2], ['G', 8]],
'D': [['A', 4]], 'E': [['A', 2]] heuristic = {'S': 8, 'A': 8, 'B': 4,
'C': 3, 'D': 40, 'E': 40, 'G': 0} cost = {'S': 0}
def AStarSearch():
    global m_tree, heuristic
    closed = [] opened =
[['S', 8]] while True:
    fn = [i[1] for i in opened]
    chosen_index = fn.index(min(fn))
    node = opened[chosen_index][0]
    closed.append(opened[chosen_index])
    del opened[chosen_index]
    if closed[-1][0] == 'G':
        break
    for item in m_tree[node]: if item[0] in [closed_item[0]
    for closed_item in closed]:
        continue cost.update({item[0]: cost[node] +
    item[1]}) fn_node = cost[node] + heuristic[item[0]]
    + item[1] temp = [item[0], fn_node]
    opened.append(temp)
    trace_node = 'G'
optimal_sequence = ['G']
for i in range(len(closed)-2, -1, -1):
    check_node = closed[i][0] if trace_node in [children[0] for
    children in m_tree[check_node]]: children_costs = [temp[1] for
    temp in m_tree[check_node]] children_nodes = [temp[0] for
    temp in m_tree[check_node]]
    if cost[check_node] + children_costs[children_nodes.index(trace_node)] ==

```

```

cost[trace_node]:
    optimal_sequence.append(check_node)
    trace_node = check_node
optimal_sequence.reverse()

opt_cost=0
l1=[]
for i in optimal_sequence:
    for x in m_tree['S']:
        if(x[0]==i):
            k=x[0]
            opt_cost=opt_cost+x[1]
            break
    for y
in m_tree[k]:
    if(y[0]=='G'):
        opt_cost=opt_cost+y[1]
        break

return closed, optimal_sequence, opt_cost

if __name__ == '__main__':
    visited_nodes, optimal_nodes,opt_cost = AStarSearch()
    print('Visited nodes are : ' + str(visited_nodes))
    print('Optimal Path is : ' + str(optimal_nodes))
    print('Minimum Cost is :'+str(opt_cost))

```

Output:

```
Visited nodes are : [['S', 8], ['C', 7], ['A', 9], ['G', 4]]
```

```
Optimal Path is : ['S', 'A', 'G']
```

```
Minimum Cost is :4
```

```
>>> |
```

EXPERIMENT NO-5

Aim: To implement

- a) Backpropagation Algorithm
- b) Self Organizing Maps

Theory:

- Propagates inputs forward in the usual way, i.e.
 - All outputs are computed using sigmoid thresholding of the inner product of the corresponding weight and input vectors.
 - All outputs at stage n are connected to all the inputs at stage $n+1$
- Propagates the errors backwards by apportioning them to each unit according to the amount of this error the unit is responsible for.

We now derive the stochastic Backpropagation algorithm for the general case. Input vector for unit j (x_{ji} = i th input to the j th unit) and weight vector for unit j (w_{ji} = weight on x_{ji}) and the weighted sum of inputs for unit j o_j = output of unit j ($o_j = \sigma(z_j)$)

t_j = target for unit j

$Downstream(j)$ = set of units whose immediate inputs include the output of j Outputs
= set of output units in the final layer

Since we update after each training example, we can simplify the notation somewhat by imagining that the training set consists of exactly one example and so the error can simply be denoted by E .

We want to calculate $\frac{\partial E}{\partial w_{ji}}$ for each input weight w_{ji} for each output unit j . Note first that since z_j is a function of w_{ji} regardless of where in the network unit j is located,

$$\begin{aligned} \frac{\partial E}{\partial w_{ji}} &= \frac{\partial E}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{ji}} \\ &= \frac{\partial E}{\partial z_j} x_{ji} \end{aligned}$$

$$\frac{\partial E}{\partial z_j}$$

Furthermore, is the same regardless of which input weight of unit j we are trying to update. So we denote this quantity by δ_j .

Consider the case when j belongs to outputs. We know

$$E = \frac{1}{2} \sum_{k \in \text{Outputs}} (t_k - \sigma(z_k))^2$$

Since the outputs of all units $k \neq j$ are independent of w_{ji} , we can drop the summation and consider just the contribution to E by j .

$$\begin{aligned} \delta_j = \frac{\partial E}{\partial z_j} &= \frac{\partial}{\partial z_j} \frac{1}{2} (t_j - o_j)^2 \\ &= -(t_j - o_j) \frac{\partial o_j}{\partial z_j} \\ &= -(t_j - o_j) \frac{\partial}{\partial z_j} \sigma(z_j) \\ &= -(t_j - o_j) (1 - \sigma(z_j)) \sigma(z_j) \\ &= -(t_j - o_j) (1 - o_j) o_j \end{aligned}$$

Thus

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} = \eta \delta_j x_{ji} \quad (17)$$

Now consider the case when j is a hidden unit. Like before, we make the following two important observations.

- For each unit k downstream from j , z_k is a function of z_j
- The contribution to error by all units in the same layer as j is independent of w_{ji} . We want

to calculate $\frac{\partial E}{\partial w_{ji}}$ for each input weight w_{ji} for each hidden unit j . Note that w_{ji} influences just z_j which influences o_j which influences $z_k \forall k \in \text{Downstream}(j)$ each of which influence E . So we can write

$$\begin{aligned}
\frac{\partial E}{\partial w_{ji}} &= \sum_{k \in \text{Downstream}(j)} \frac{\partial E}{\partial z_k} \cdot \frac{\partial z_k}{\partial o_j} \cdot \frac{\partial o_j}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{ji}} \\
&= \sum_{k \in \text{Downstream}(j)} \frac{\partial E}{\partial z_k} \cdot \frac{\partial z_k}{\partial o_j} \cdot \frac{\partial o_j}{\partial z_j} \cdot x_{ji}
\end{aligned}$$

Again note that all the terms except x_{ji} in the above product are the same regardless of which input weight of unit j we are trying to update. Like before, we denote this common quantity

by δ_j . Also note that $\frac{\partial E}{\partial z_k} = \delta_k$, $\frac{\partial z_k}{\partial o_j} = w_{kj}$ and $\frac{\partial o_j}{\partial z_j} = o_j(1 - o_j)$.

$$\begin{aligned}
\delta_j &= \sum_{k \in \text{Downstream}(j)} \frac{\partial E}{\partial z_k} \cdot \frac{\partial z_k}{\partial o_j} \cdot \frac{\partial o_j}{\partial z_j} \\
&= \sum_{k \in \text{Downstream}(j)} \delta_k w_{kj} o_j (1 - o_j)
\end{aligned}$$

. Substituting,

Thus,

$$\delta_k = o_j(1 - o_j) \sum_{k \in \text{Downstream}(j)} \delta_k w_{kj} \quad (18)$$

We are now in a position to state the Backpropagation algorithm formally.

Formal statement of the algorithm:

For each training example:

- Input the instance \vec{x} and compute the output o_u of every unit.
- For each output unit k , calculate

$$\delta_k = o_k(1 - o_k)(t_k - o_k)$$

- For each hidden unit h , calculate

$$\delta_h = o_h(1 - o_h) \sum_{k \in \text{Downstream}(h)} w_{kh} \delta_k$$

- Update each network weight w_{ji} as follows:

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where $\Delta w_{ji} = \eta \delta_j x_{ji}$

Self Organizing Map(SOM):

Self Organizing Map(SOM) by **Teuvo Kohonen** provides a data visualization technique which helps to understand high dimensional data by reducing the dimensions of data to a map. SOM also represents clustering concept by grouping similar data together. Therefore it can be said that SOM **reduces data dimensions** and **displays similarities among data**.

With SOM, clustering is performed by having several units compete for the current object. Once the data have been entered into the system, the newtwork of artificial neurons is trained by providing information about inputs. The weight vector of the unit is closest to the current object becomes the winning or active unit. During the training stage, the values for the input variables are gradually adjusted in an attempt to preserve neighborhood relationships that exist within the input data set. As it gets closer to the input object, the weights of the winning unit are adjusted as well as its neighbors.

Teuvo Kohonen writes "The SOM is a new, effective software tool for the visualization of highdimensional data. It converts complex, nonlinear statistical relationsihps between highdimensional data items into simple geometric relationships on a low-dimensional display. As it thereby compresses information while preserving the most important topological and metric relationships of the primary data items on the display, it may also be thought to produce some kind of abstractions."

SOM Algorithm

Each data from data set recognizes themselves by competeting for representation. SOM mapping steps starts from initializing the weight vectors. From there a sample vector is selected randomly and the map of weight vectors is searched to find which weight best represents that sample. Each weight vector has neighboring weights that are close to it. The weight that is chosen is rewarded by being able to become more like that randomly selected sample vector. The neighbors of that weight are also rewarded by being able to become more like the chosen sample vector. From this step the number of neighbors and how much each weight can learn decreases over time. This whole process is repeated a large number of times, usually more than 1000 times.

In sum, learning occurs in several steps and over many iterations. :

1. Each node's weights are initialized.
2. A vector is chosen at random from the set of training data.

3. Every node is examined to calculate which one's weights are most like the input vector. The winning node is commonly known as the Best Matching Unit (BMU).
4. Then the neighbourhood of the BMU is calculated. The amount of neighbors decreases over time.
5. The winning weight is rewarded with becoming more like the sample vector. The neighbors also become more like the sample vector. The closer a node is to the BMU, the more its weights get altered and the farther away the neighbor is from the BMU, the less it learns.
6. Repeat step 2 for N iterations.

CONCLUSION

The back propagation algorithm is implemented and effect on different weights and bias values is observed. The algorithm uses one hidden layer only. The algorithm is situated for non linearity separate classes of problems.

SOM's that they are very easy to understand. They classify data well and then are easily evaluate for their own quality so you can actually calculated how good a map is and how strong the similarities between object same. Thus SOM's are successfully implemented.

Program Execution/formation/correction/ethical practices (07)	Documentation (02)	Timely Submission (03)	Viva Answer (03)	Experiment Marks (15)	Teacher Signature with date

Experiment No: 5A)

Back Propagation

Code:

```
from math import exp

hiddenlayer=[[0.10,0.25],[0.20,0.30]] outerlayer=[[0.4,0.45],[0.50,0.55]]

inp=[0.05,0.15]
b=[0.30,0.60] d=[0.01,0.99]

# forward pass def
forward(l, da): n=0
for i in range(len(l)):
    n+=da[i]*l[i]
    return n
o_net,o_out=[],[]
h_net,h_out=[],[] # Forward
pass for hidden layer for i in
hiddenlayer: da=inp
n=forward(i,da) n+=b[0]
o=1.0/(1.0+exp(-n))
h_out.append(o)
h_net.append(n) # Forward
pass for outer layer
da=h_out[:] for i in outerlayer:
n=forward(i,da) n+=b[1]
o=1.0/(1.0+exp(-n))
o_out.append(o)
o_net.append(n)
for i in range(len(h_net)):
print("Net h",i," ",h_net[i]) for i
in range(len(h_out)):
print("Out h",i," ",h_out[i]) for i
in range(len(o_net)):
print("Net O",i," ",o_net[i]) for i
in range(len(o_out)): print("Out
O",i," ",o_out[i])
# Error etotal=0 for i in
range(len(d)):
etotal+=0.5*(d[i]-o_out[i])**2
print("Ettotal: ",etotal)
```

```

# Backword pass def
backward(i):
    et_w=-(d[i]-o_out[i])*o_out[i]*(1-o_out[i])*h_out[i]
    return et_w def
h_backward(i,w,l):
e=0 for j in range(l):
    e+=(d[j]-o_out[j])*o_out[j]*(1-o_out[j])*w
    e=e*h_out[i]*(1-h_out[i])*inp[i]
    return(e)
# Backwardpass for outer layer
o_w,h_w=[],[] for i in
range(len(outerlayer)):
    for j in range(len(outerlayer)):
et_w=backward(i)
w=outerlayer[i][j]-0.5*et_w
o_w.append(w) #Backwardpass for
hidden layer for i in
range(len(hiddenlayer)):
    for j in range(len(hiddenlayer)):
        et_w=h_backward(i,outerlayer[i][i],len(hiddenlayer))
        w=hiddenlayer[i][j]-0.5*et_w h_w.append(w)
c=1 for i in
range(len(h_w)):
    print("W",c," ",h_w[i])
    c+=1 for i in
range(len(o_w)):
    print("W",c," ",o_w[i])
    c+=1

```

Output:

```

Net h 0 : 0.34249999999999997
Net h 1 : 0.355
Out h 0 : 0.584797674717737
Out h 1 : 0.5878295384825115
Net O 0 : 1.098442362204225
Net O 1 : 1.21570508352425
Out O 0 : 0.749968137434525
Out O 1 : 0.7713068408614581
Etotal: 0.2976897711361578
W 1 : 0.09975675368803144
W 2 : 0.24975675368803144
W 3 : 0.1989987717999043
W 4 : 0.29899877179990425
W 5 : 0.35942796140922084

```

W 6 : 0.40942796140922083
W 7 : 0.511338013381744
W 8 : 0.561338013381744
>

B) Self Organizing Maps

Code:

```
import math def som(weights,
sample, alpha ) :
    D0 = 0
    D1 = 0 for i in range( len(
sample ) ) :
        D0 = D0 + math.pow( ( weights[0][i] - sample[i]), 2 )
        D1 = D1 + math.pow( ( weights[1][i] - sample[i]), 2 )

    if D0 < D1 :
        J = 0 else :    J = 1 for i in range( len ( weights[J] ) ):
weights[J][i] = weights[J][i] + alpha * ( sample[i] - weights[J][i] )
    return weights

print("Enter S1, S2, S3, S4 : ")
S = [] for i in range(0,4):
S.append([]) for j in
range(0,4):
    t = int(input())
    S[i].append(t) w = [ [ 0.2, 0.6, 0.5, 0.9 ], [ 0.8,
0.4, 0.7, 0.3 ] ] alpha = float(input("Enter alpha : "))
epochs = int(input("Enter the number of epochs : "))
for i in range( epochs ) : print("Epoch ",i) for j in
range( len( S ) ) :
    sample = S[j]
    wg = som( w, sample, alpha )
    print("Weight updation of S%d"%j,wg)
    alpha = 0.5 * alpha
```

Output:

```
Enter S1, S2, S3, S4 :
1
2
4
```

1
5
1
1
2
3
1
5
6
8
7
9
1

Enter alpha : 0.5

Enter the number of epochs : 3

Epoch 0

Weight updation of S0 [[0.2, 0.6, 0.5, 0.9], [0.9, 1.2000000000000002,
2.3499999999999996, 0.6499999999999999]]

Weight updation of S1 [[0.2, 0.6, 0.5, 0.9], [2.9499999999999997, 1.1,
1.6749999999999998, 1.325]]

Weight updation of S2 [[0.2, 0.6, 0.5, 0.9], [2.9749999999999996, 1.05, 3.3375,
3.6624999999999996]]

Weight updation of S3 [[0.2, 0.6, 0.5, 0.9], [5.4875, 4.025, 6.168749999999999,
2.33125]]

Epoch 1

Weight updation of S0 [[0.4, 0.95, 1.375, 0.925], [5.4875, 4.025,
6.168749999999999, 2.33125]]

Weight updation of S1 [[1.5499999999999998, 0.9624999999999999, 1.28125,
1.19375], [5.4875, 4.025, 6.168749999999999, 2.33125]]

Weight updation of S2 [[1.5499999999999998, 0.9624999999999999, 1.28125,
1.19375], [4.865625, 3.2687500000000003, 5.8765624999999995,
3.2484374999999996]]

Weight updation of S3 [[1.5499999999999998, 0.9624999999999999, 1.28125,
1.19375], [5.649218749999999, 4.2015625000000005, 6.657421875,
2.6863281249999997]]

Epoch 2

Weight updation of S0 [[1.4812499999999997, 1.0921874999999999, 1.62109375,
1.1695312500000001], [5.649218749999999, 4.2015625000000005, 6.657421875,
2.6863281249999997]]

Weight updation of S1 [[1.9210937499999998, 1.0806640625, 1.54345703125,
1.27333984375], [5.649218749999999, 4.2015625000000005, 6.657421875,
2.6863281249999997]]

Weight updation of S2 [[1.9210937499999998, 1.0806640625, 1.54345703125,

1.27333984375], [5.318066406249999, 3.8013671875000004, 6.450244140625,
3.100537109375]]

Weight updation of S3 [[1.9210937499999998, 1.0806640625, 1.54345703125,
1.27333984375], [5.653308105468749, 4.2011962890625005, 6.768963623046875,
2.8379699707031247]]

>>>

EXPERIMENT NO-7

Aim:

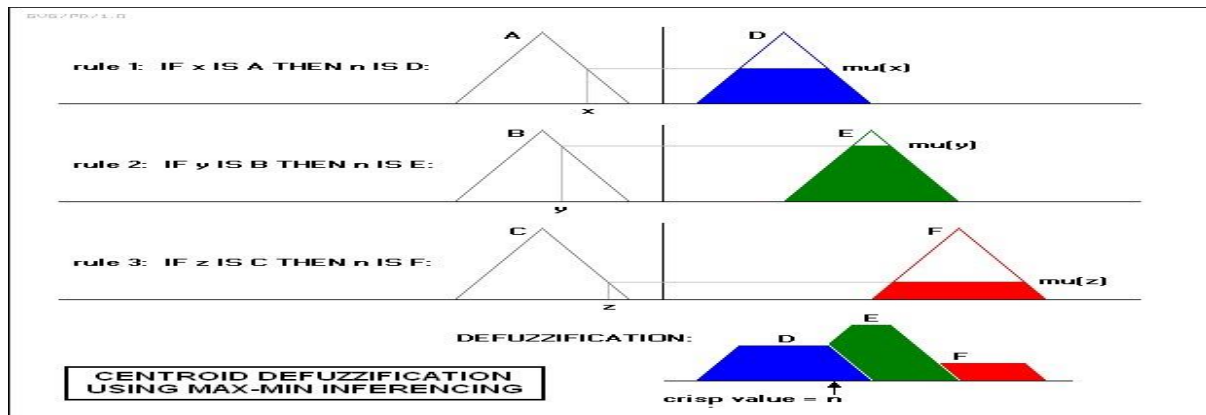
To implement Fuzzy Controller System.

Theory-

A fuzzy logic controller is a control system based on fuzzy logic—a mathematical system that analyzes analog input values in terms of logical variables that take on continuous values between 0 and 1, in contrast to classical or digital logic, which operates on discrete values of either 1 or 0 (true or false, respectively)

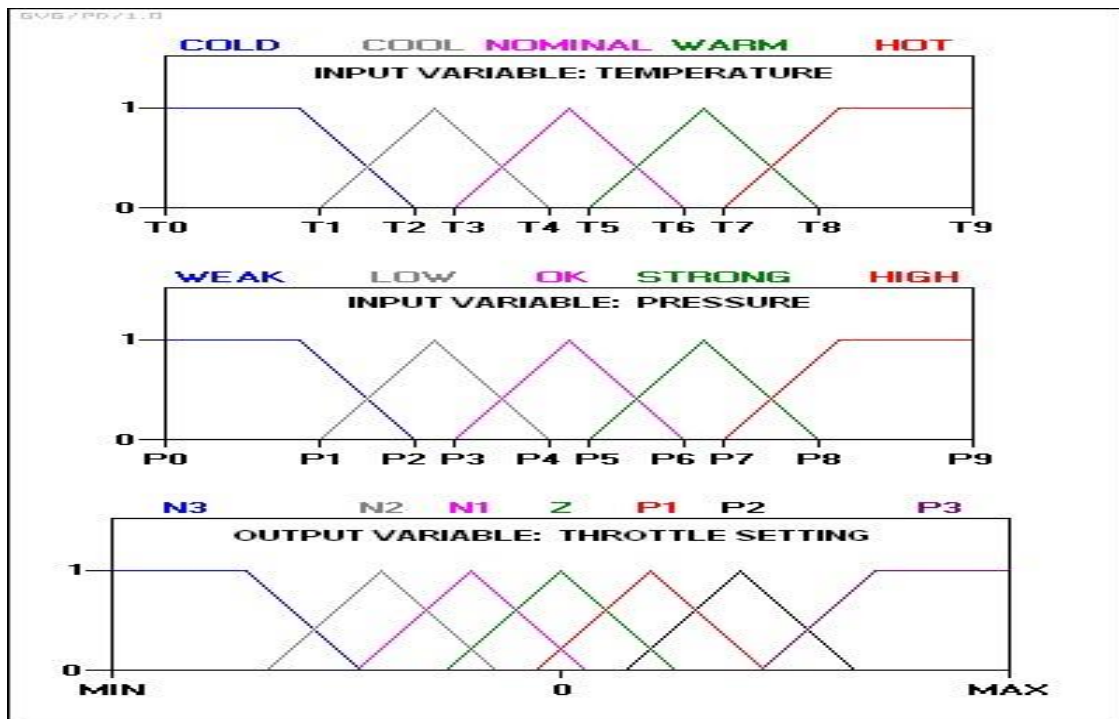
Fuzzy controllers are very simple conceptually. They consist of an input stage, a processing stage, and an output stage. The input stage maps sensor or other inputs, such as switches, thumbwheels, and so on, to the appropriate membership functions and truth values. The processing stage invokes each appropriate rule and generates a result for each, then combines the results of the rules. Finally, the output stage converts the combined result back into a specific control output value. The most common shape of membership functions is triangular, although trapezoidal and bell curves are also used, but the shape is generally less important than the number of curves and their placement. From three to seven curves are generally appropriate to cover the required range of an input value, or the "universe of discourse" in fuzzy jargon.

Consider a rule for a thermostat: IF (temperature is "cold") THEN (heater is "high") This rule uses the truth value of the "temperature" input, which is some truth value of "cold", to generate a result in the fuzzy set for the "heater" output, which is some value of "high". This result is used with the results of other rules to finally generate the crisp composite output. Obviously, the greater the truth value of "cold", the higher the truth value of "high", though this does not necessarily mean that the output itself will be set to "high" since this is only one rule among many. In some cases, the membership functions can be modified by "hedges" that are equivalent to adverbs. Common hedges include "about", "near", "close to", "approximately", "very", "slightly", "too", "extremely", and "somewhat". These operations may have precise definitions, though the definitions can vary considerably between different implementations. "Very", for one example, squares membership functions; since the membership values are always less than 1, this narrows the membership function. "Extremely" cubes the values to give greater narrowing, while "somewhat" broadens the function by taking the square root. In practice, the fuzzy rule sets usually have several antecedents that are combined using fuzzy operators, such as AND, OR, and NOT, though again the definitions tend to vary: AND, in one popular definition, simply uses the minimum weight of all the antecedents, while OR uses the maximum value. There is also a NOT operator that subtracts a membership function from 1 to give the "complementary" function. There are several ways to define the result of a rule, but one of the most common and simplest is the "maxmin" inference method, in which the output membership function is given the truth value generated by the premise.



each rule provides a result as a truth value of a particular membership function for the output variable. In centroid defuzzification the values are OR'd, that is, the maximum value is used and values are not added, and the results are then combined using a centroid calculation. As a general example, consider the design of a fuzzy controller for a steam turbine. The block diagram of this control system appears as follows:

The input and output variables map into the following fuzzy set:



—where:

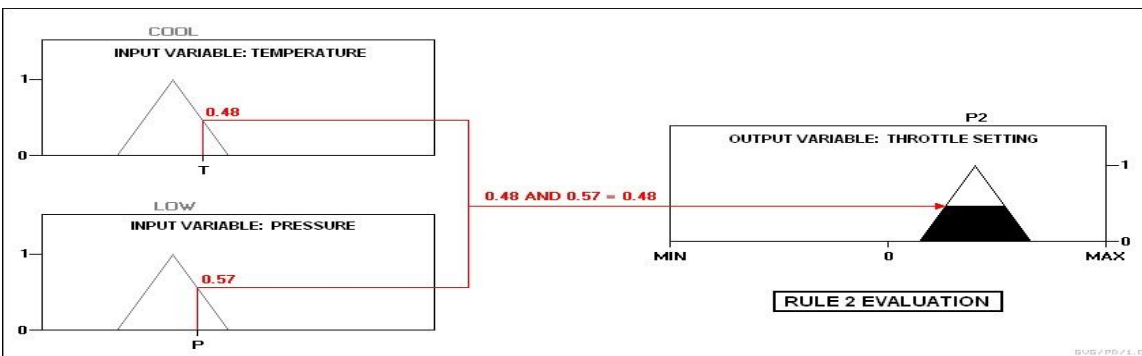
- N3: Large negative.
- N2: Medium negative.
- N1: Small negative.

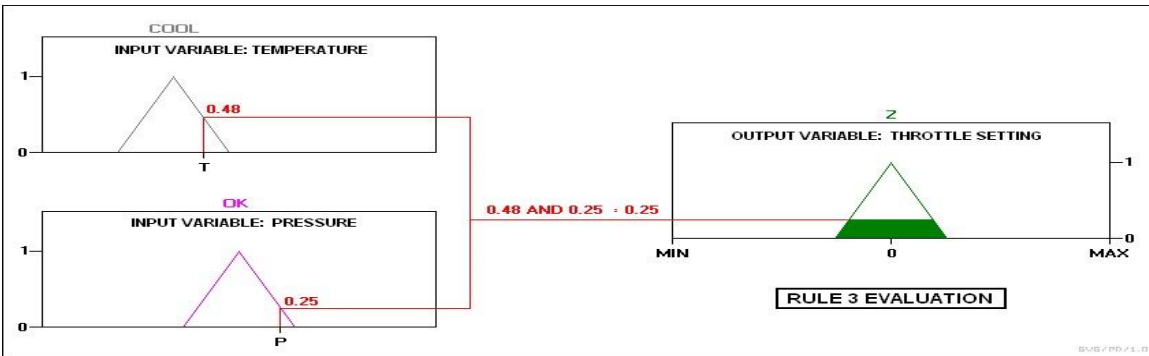
Z: Zero.
P1: Small positive.
P2: Medium positive.
P3: Large positive.

The rule set includes such rules as:

rule 1: IF temperature IS cool AND pressure IS weak,
THEN throttle is P3.
rule 2: IF temperature IS cool AND pressure IS low,
THEN throttle is P2.
rule 3: IF temperature IS cool AND pressure IS ok,
THEN throttle is Z.
rule 4: IF temperature IS cool AND pressure IS strong,
THEN throttle is N2.

In practice, the controller accepts the inputs and maps them into their membership functions and truth values. These mappings are then fed into the rules. If the rule specifies an AND relationship between the mappings of the two input variables, as the examples above do, the minimum of the two is used as the combined truth value; if an OR is specified, the maximum is used. The appropriate output state is selected and assigned a membership value at the truth level of the premise. The truth values are then defuzzified. For an example, assume the temperature is in the "cool" state, and the pressure is in the "low" and "ok" states. The pressure values ensure that only rules 2 and 3 fire:





The two outputs are then defuzzified through centroid defuzzification

Example:

FUZZY LOGIC CONTROLLER (FLC) FOR SMART WASHING MACHINE (SWM)

The two inputs are: 1. Degree of dirt 2. Type of dirt

The fuzzy controller takes two inputs (as stated for simplification), processes the information and outputs a wash time. How to get these two inputs can be left to the sensors (optical, electrical or any type). degree of dirt is determined by the transparency of the wash water. The dirtier the clothes, less transparent the water being analyzed by the sensors is. On the other hand, type of dirt is determined by the time of saturation, the time it takes to reach saturation. Saturation is a point, at which there is no more appreciable change in the color of the water. Degree of dirt determines how much dirty a cloth is. Where as Type of dirt determines the quality of dirt. Greasy cloths, for example, take longer for water transparency to reach transparency because grease is less soluble in water than other forms of dirt. Thus a fairly straight forward sensor system can provide us the necessary input for our fuzzy controller.

Rules:

1. If dirtiness_of_clothes is Large and type_of_dirt is Greasy then wash_time is VeryLong;
2. If dirtiness_of_clothes is Medium and type_of_dirt is Greasy then wash_time is Long;
3. If dirtiness_of_clothes is Small and type_of_dirt is Greasy then wash_time is Long;
4. If dirtiness_of_clothes is Large and type_of_dirt is Medium then wash_time is Long;
5. If dirtiness_of_clothes is Medium and type_of_dirt is Medium then wash_time is Medium;
6. If dirtiness_of_clothes is Small and type_of_dirt is Medium then wash_time is Medium;
7. If dirtiness_of_clothes is Large and type_of_dirt is NotGreasy then wash_time is Medium;
8. If dirtiness_of_clothes is Medium and type_of_dirt is NotGreasy then wash_time is Short;
9. If dirtiness_of_clothes is Small and type_of_dirt is NotGreasy then wash_time is VeryShort

The fuzzy sets should be like depicted in the following figure:

Rules:-

The sets of rules used here to derive the output are:

Rule Number	Linguistic Inputs			Linguistic output
	Type of Cloth	Type of Dirt	Dirtiness of Cloth	Wash time
1	Silk	Not greasy	Small	Very Short
2	Silk	Not greasy	Medium	Short
3	Silk	Not greasy	Large	Medium
4	Silk	Medium	Small	Medium
5	Silk	Medium	Medium	Long
6	Silk	Medium	Large	Long
7	Silk	Greasy	Small	Medium
8	Silk	Greasy	Medium	Long
9	Silk	Greasy	Large	Very long
10	Woolen	Not greasy	Small	Short
11	Woolen	Not greasy	Medium	Medium
12	Woolen	Not greasy	Large	Long
13	Woolen	Medium	Small	Medium
14	Woolen	Medium	Medium	Medium
15	Woolen	Medium	Large	Long
16	Woolen	Greasy	Small	Long
17	Woolen	Greasy	Medium	Long
18	Woolen	Greasy	Large	Very Long
19	Cotton	Not greasy	Small	Short
20	Cotton	Not greasy	Medium	Medium
21	Cotton	Not greasy	Large	Long

Rule Number	Linguistic Inputs			Linguistic output
	Type of Cloth	Type of Dirt	Dirtiness of Cloth	Wash time
22	Cotton	Medium	Small	Medium
23	Cotton	Medium	Medium	Long
24	Cotton	Medium	Large	Very Long
25	Cotton	Greasy	Small	Long
26	Cotton	Greasy	Medium	Long
27	Cotton	Greasy	Large	Very Long

Table: Rules for Fuzzy Wash Time Control

The rules obtained in Table can be read in terms of IF and THEN statements as shown in below.
Rule 1: IF (Type of Cloth is Silk) and (Type of Dirt is Not Greasy) and (Dirtiness of Cloth is Small) THEN (Wash time is Very Short)

Rule2: IF (Type of Cloth is Silk) and (Type of Dirt is Not Greasy) and (Dirtiness of Cloth is Medium) THEN (Wash time is Short)

.....

Rule27: IF(Type of Cloth is Cotton) and (Type of Dirt is Greasy) and (Dirtine

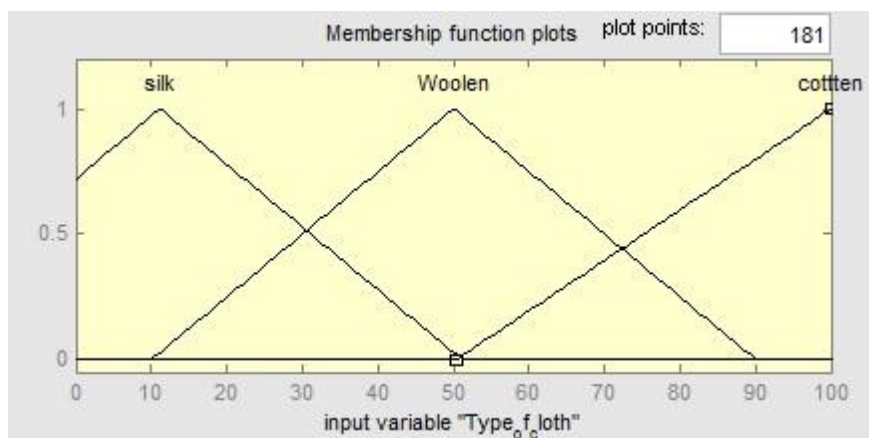


Figure: A membership for input variable Type of Cloth

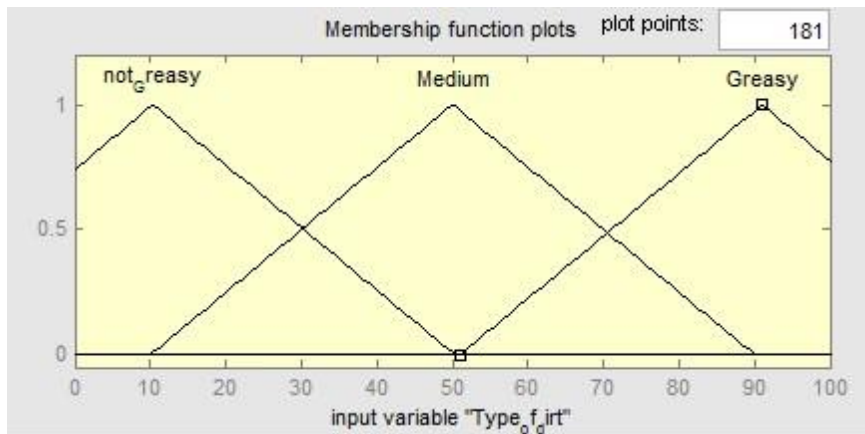


Figure: A membership for input variable Type of Dirt

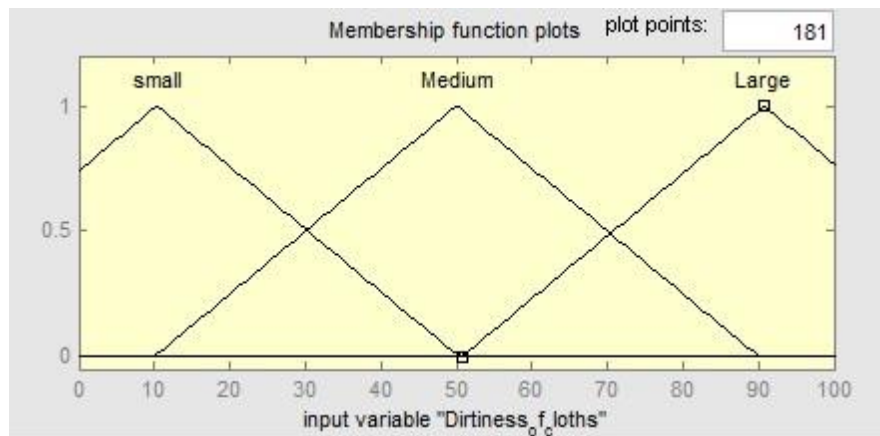


Figure: A membership for input variable Dirtiness of Cloths

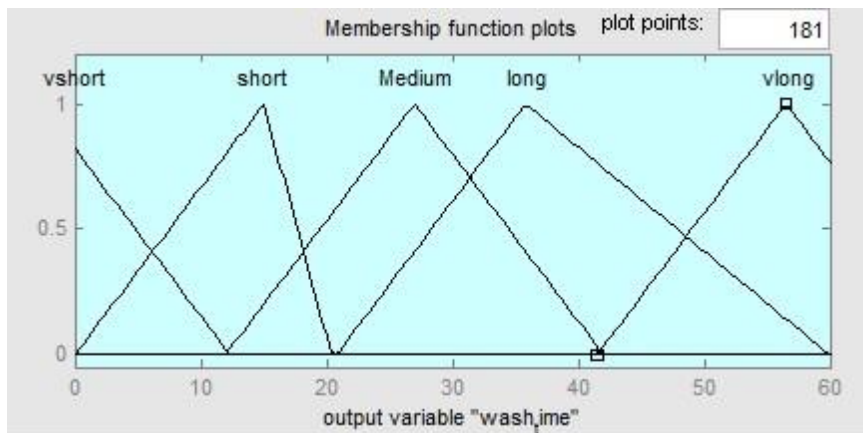


Figure: A membership for Output variable Wash Time

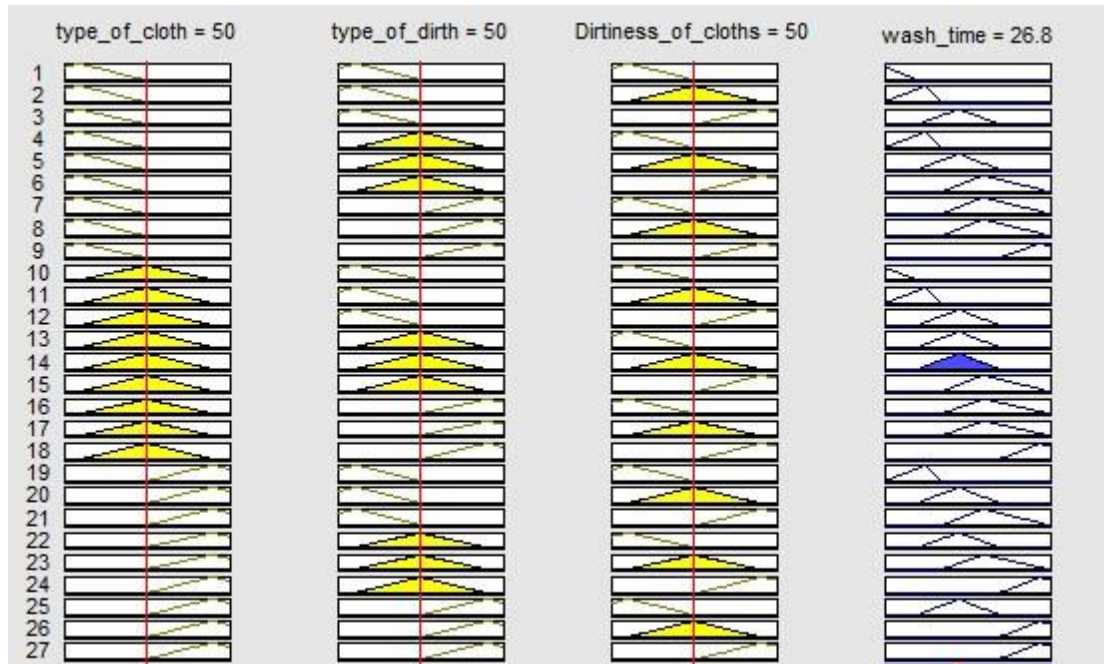


Figure: Rules of the System.

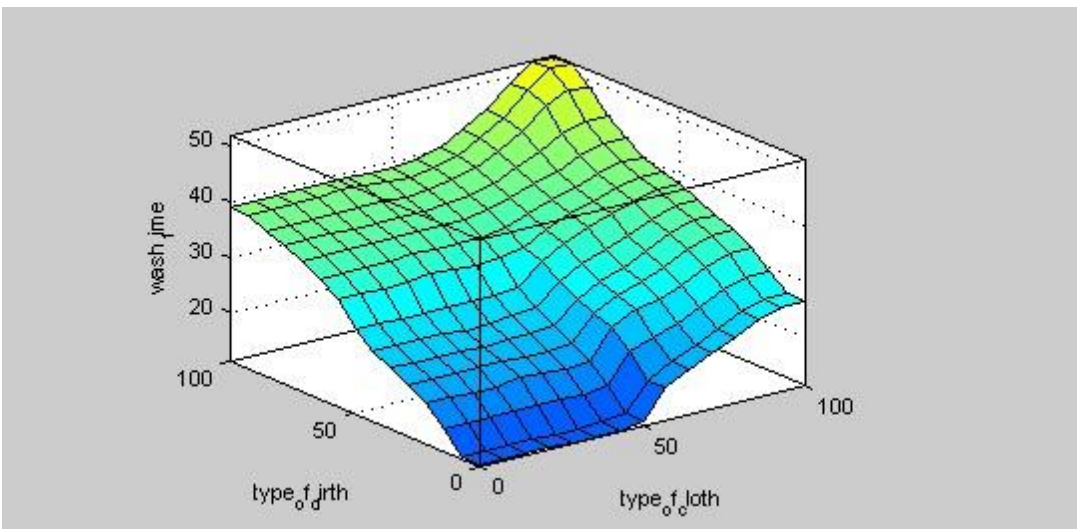


Figure: Response surface of the input output relations

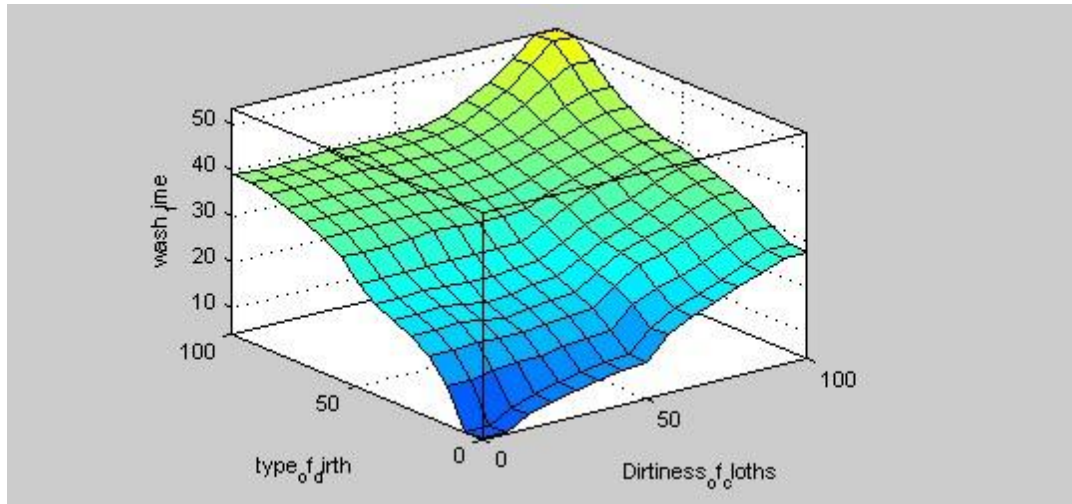


Figure: Response surface of the input output relations

CONCLUSION

By the use of fuzzy logic control we have been able to obtain a wash time for different type of dirt and different degree of dirt and different type of cloths. The conventional method required the human interruption to decide upon what should be the wash time for different cloths. In other words this situation analysis ability has been incorporated in the machine which makes the machine much more automatic and represents the decision taking power of the new arrangement. Here the sensors sense the input values and using the above Model the inputs are fuzzified and then by using simple if-else rules and other simple fuzzy set operations the output fuzzy function is obtained and using the criteria the output value for wash time is obtained.

Program Execution/formation/correction/ethical practices (07)	Documentation (02)	Timely Submission (03)	Viva Answer (03)	Experiment Marks (15)	Teacher Signature with date

Experiment No: 7

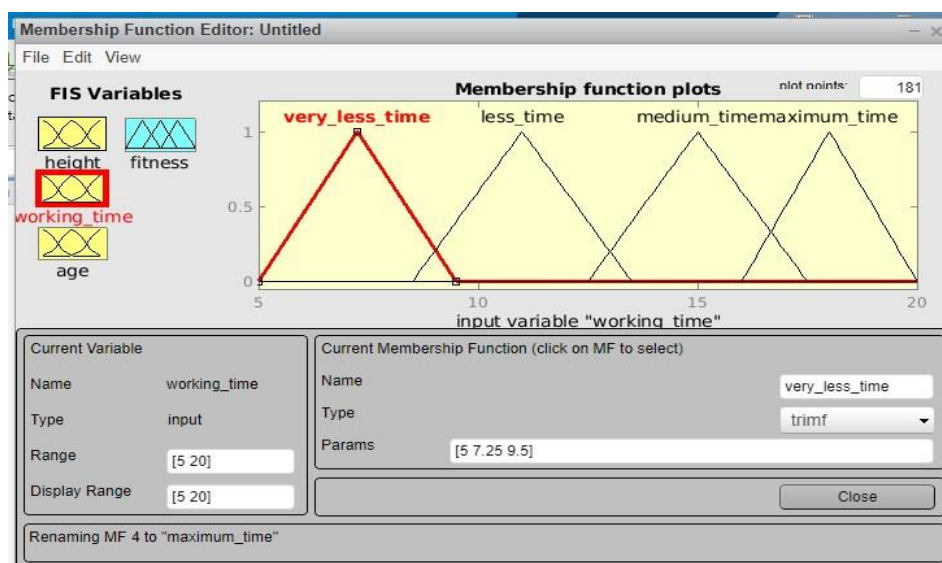
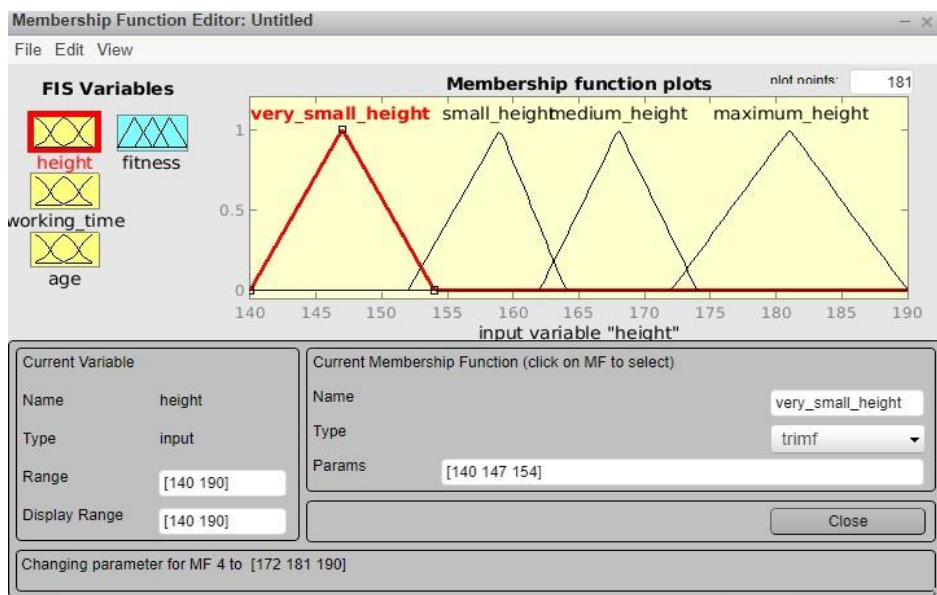
Fuzzy Controller

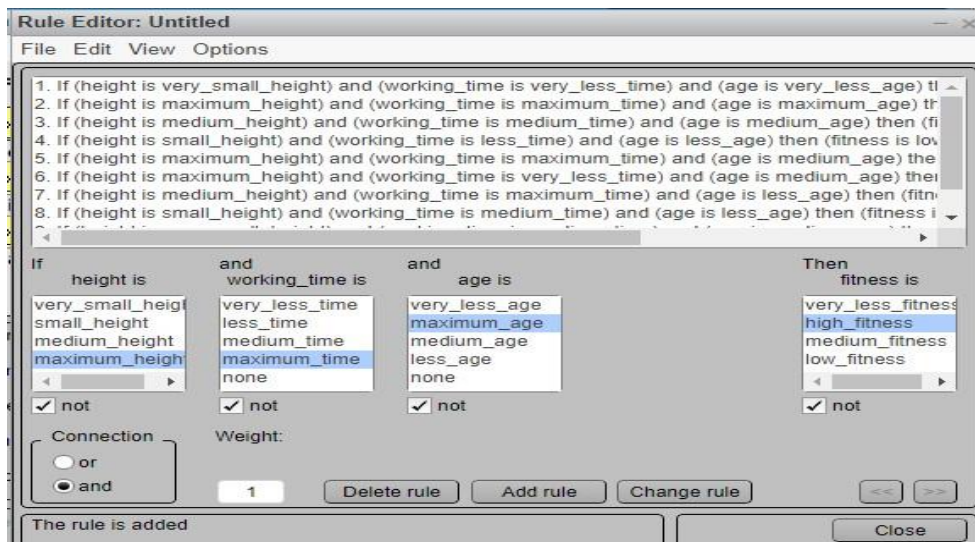
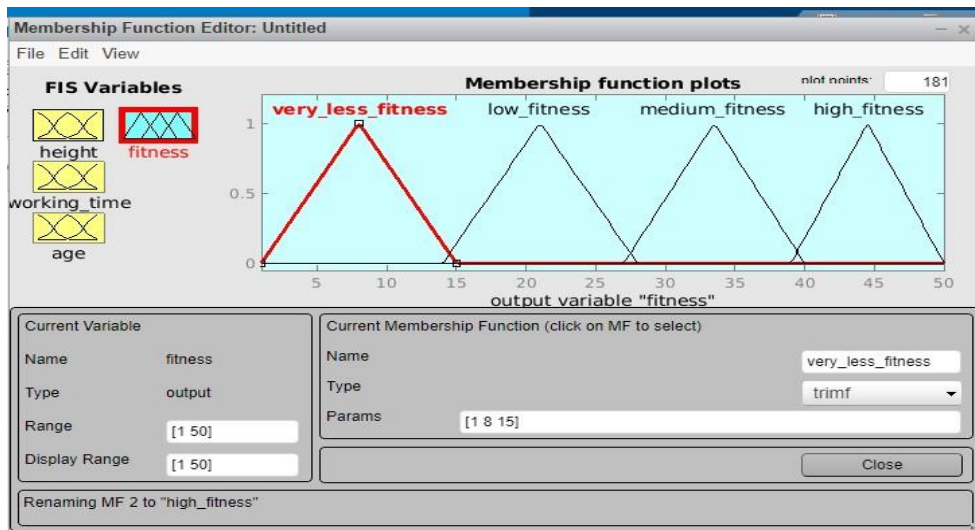
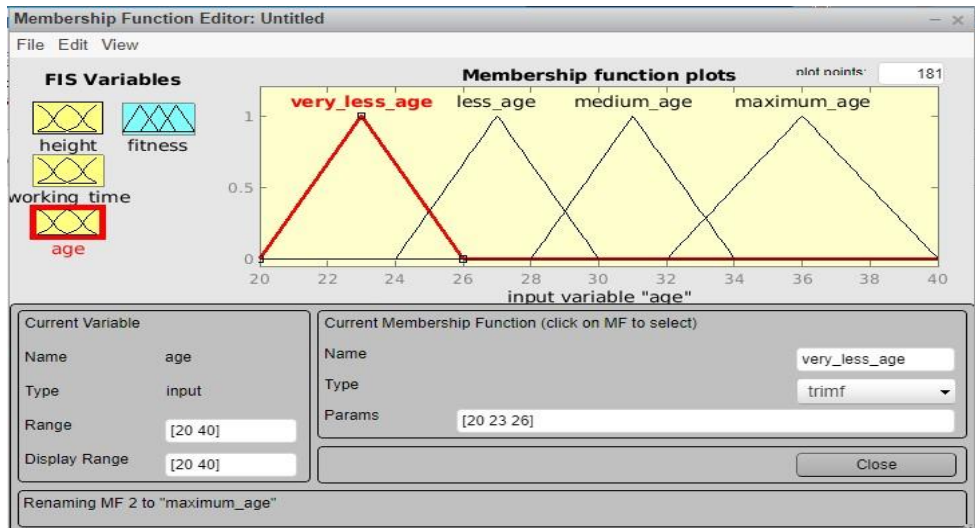
Problem statement :

Selection bridegroom for a bride. Input variables are height, working time and Age and output is fitness. Three profiles are available.

Name	Height	Working time	Age
Raman	179	9	24
Rajesh	189	6	35
Ravi	155	18	21

Output :





Rule Editor: Untitled

File Edit View Options

3. If (height is medium_height) and (working_time is medium_time) and (age is medium_age) then (fitness is low_fitness)
 4. If (height is small_height) and (working_time is less_time) and (age is less_age) then (fitness is low_fitness)
 5. If (height is maximum_height) and (working_time is maximum_time) and (age is medium_age) then (fitness is low_fitness)
 6. If (height is maximum_height) and (working_time is very_less_time) and (age is medium_age) then (fitness is low_fitness)
 7. If (height is medium_height) and (working_time is maximum_time) and (age is less_age) then (fitness is low_fitness)
 8. If (height is small_height) and (working_time is medium_time) and (age is less_age) then (fitness is low_fitness)
 9. If (height is very_small_height) and (working_time is medium_time) and (age is medium_age) then (fitness is low_fitness)
 10. If (height is not maximum_height) and (working_time is not maximum_time) and (age is not maximum_age) then (fitness is high_fitness)

If	and	and	and	Then
height is	working_time is	age is		fitness is
very_small_height	very_less_time	very_less_age		very_less_fitness
small_height	less_time	maximum_age		high_fitness
medium_height	medium_time	medium_age		medium_fitness
maximum_height	maximum_time	less_age		low_fitness
	none	none		

☒ not ☒ not ☒ not ☒ not

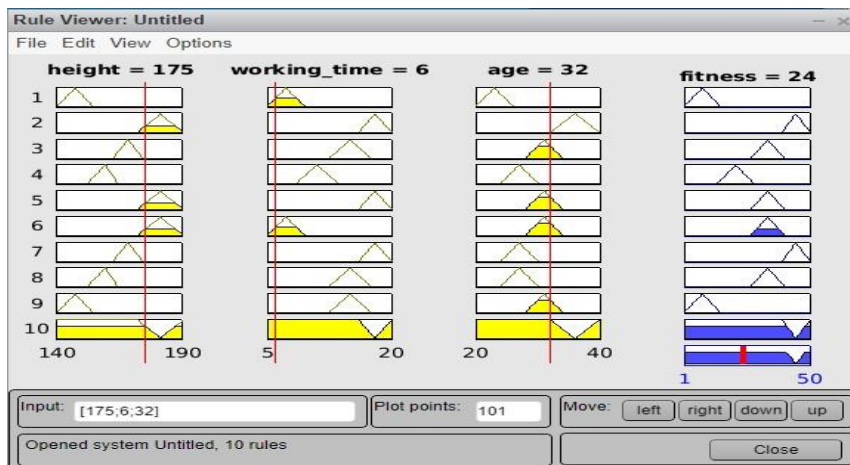
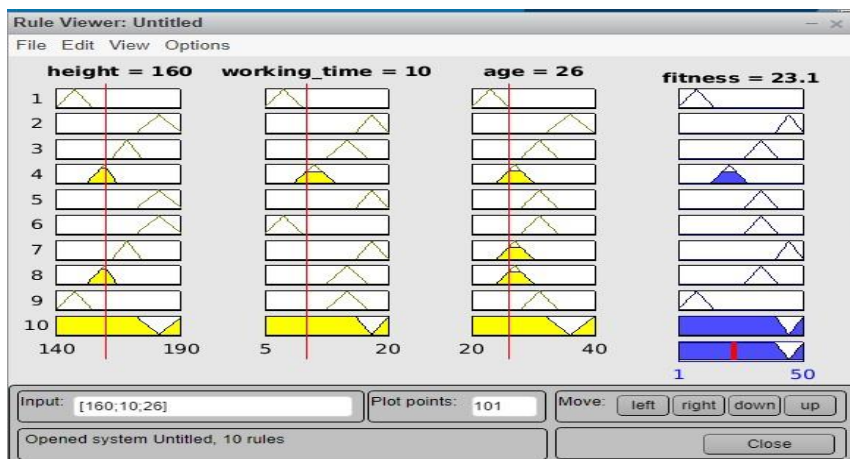
Connection: ☐ or ☒ and

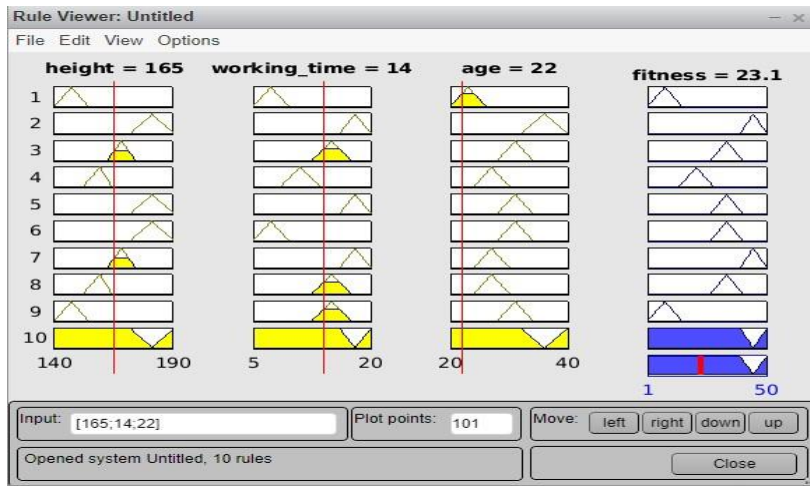
Weight: 1

Delete rule Add rule Change rule

The rule is added

Close





EXPERIMENT NO-8

Aim:

Develop an application using AI or SC techniques. **Theory:**

Neural network for recognizing handwritten digits using MNIST dataset.

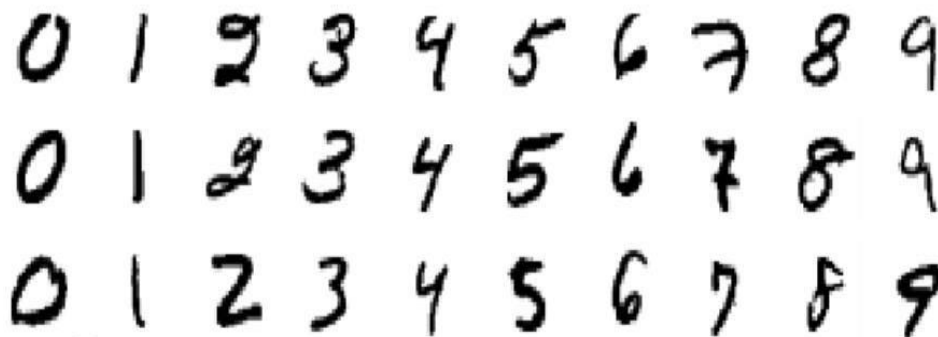
The MNIST problem is a dataset developed by Yann LeCun , it is a dataset consisting of handwritten digits. Following steps are followed for developing an application using neural network

Step1: Configure the environment required

Install the required dependencies and create a workspace to hold the files. Here tensorflow along with numpy ,pandas and matplotlib are installed.

Step2: Import the MNIST Dataset:

This dataset is made up of images of handwritten digits, 28x28 pixels in size. The dataset has been split into 55,000 images for training, 5000 for validation, and 10,000 for testing.



Step 3: Defining the Neural Network Architecture

The architecture of the neural network refers to elements such as the number of layers in the network, the number of units in each layer. Other elements of the neural network that need to be defined here are the hyperparameters like learning rate, batch size and number of epochs and dropout. The learning rate represents how much the parameters will adjust at each step of the learning process. Larger learning rates will overshoot the optimal values as they are updated. The epoch refers to number of iterations through training set and the batch size refers to how many training examples are used at each step. The dropout variable represents a threshold at which units are eliminated at random. This helps prevent overfitting.

Step 4 : Build and compile the model

The core concept of TensorFlow is the *tensor*, a data structure similar to an array or list. initialized, manipulated as they are passed through the model, and updated through the learning process. Two placeholders X and Y are defined. For X the shape of [None, 784] is used , where None represents any amount, that will be fed to neural network and 784(28*28 image is flattened into single vector of 784 pixel values) represents single 784-pixel images. The shape of Y is [None, 10] where 10 represents possible classes. The parameters that the network will update in the training process are the weight and bias thus weights and biases of network are initialized to random values.

The final step in building the graph is to define the loss function to optimize. A popular choice of loss function in TensorFlow programs is *cross-entropy*, also known as *log-loss*. For optimization adam optimizer is used.

Step 5: Training and Testing

The training process involves feeding the training dataset through the graph and optimizing the loss function. Every time the network iterates through a batch of more training images, it updates the parameters to reduce the loss in order to more accurately predict the digits shown. The testing process involves running our testing dataset through the trained graph, and keeping track of the number of images that are correctly predicted, so that we can calculate the accuracy.

Conclusion

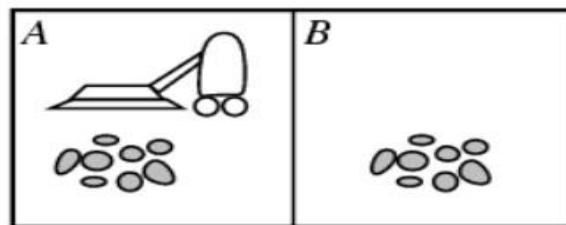
We have successfully trained a neural network to classify the MNIST dataset images. The training accuracy obtained is 98% and validation accuracy obtained is 92% .

AIM: To create an application using artificial intelligence where there are two rooms and one vacuum cleaner and dirt is there in both the rooms. Vacuum cleaners i.e agents will be present in any one room at a time. So the goal is to clean both the rooms.

Theory:

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

Agent here is a vacuum cleaner. Our agent can move left, right or it can clean the dirt. These actions are performed by our agent at a time.



- Percepts: location and contents, e.g., [A,Dirty]
- Actions: *Left, Right, Suck, NoOp*

Percept Sequence	Action
[A,Clean]	Right
[A,Dirty]	Suck
[B,Clean]	Left
[B,Dirty]	Suck

Different possible action for a Agent are:

Move left

Move right

Clean Dirt

Here the performance of our agent (vacuum cleaner) depends upon many factors such as time taken in cleaning, the path followed in cleaning, the number of moves the agent takes in total, etc.

Our agent works by checking the status of the room based on below algorithm if the room is dirty the action returned for an agent is to suck the dirt.If the dirt is present in adjacent rooms the action based on the algorithm can be to go right or to the left.

We have used this approach for the implementation

REFLEX-VACUUM-AGENT([location,status]) returns an Action

if status = Dirty then return Suck else if location = A then return

Right else if location = B then return Left

NAME:- DUBEY KARAN SANJEEV

CLASS:- B.E - 4

ROLL NO:- 04

BATCH:- A

CODE:

```
from turtle import Turtle, Screen
from random import choice
from time import sleep
from queue import SimpleQueue
```

```
w: int w, h = (853,
480) wn = Screen()
wn.screensize(w, h)
wn.bgcolor("#d3d3d3")
```

```
Room_state = {"Clean": "#FFFFFF",
              "Dirty": "#b5651d"}
```

```
cleaned = 0
```

```
def filler(t, color, delay=0, vacclean = False):
    global cleaned
    t.fillcolor(color)
    t.penup() if color ==
Room_state['Clean']:
    sleep(delay) #To avoid instant cleaning
if vacclean: cleaned += 1
t.begin_fill()
t.circle(130)
t.end_fill()
```

```
def setup():
    A = Turtle() # Draws Circle in A
    B = Turtle() # Draws Circle in B
    X = Turtle() # Text Below A
    Y = Turtle() # Text Below B
```

```
A.ht()
B.ht()
X.ht()
Y.ht()
```

```
A.speed(100)
B.speed(100)
```

```

X.speed(100)
Y.speed(100)

A.penup()
B.penup()
X.penup()
Y.penup()

A.setpos(-w / 4, -120)
B.setpos(w / 4, -120)
X.setpos(-w / 4, -200)
Y.setpos(w / 4, -200)

A.pendown()
B.pendown()

filler(A, Room_state['Clean'], False)
filler(B, Room_state['Clean'], False)

# Creates rooms and boundary
t1 = Turtle()
t1.ht() t1.speed(20)
t1.penup()
t1.setposition(w / 2, h / 2)
t1.pendown()
t1.pensize(10) t1.right(90)
t1.forward(h) t1.right(90)
t1.forward(w) t1.right(90)
t1.forward(h) t1.right(90)
t1.forward(w)
t1.backward(w / 2)
t1.right(90) t1.pensize(5)
t1.forward(h - 90)
t1.penup() t1.setpos(-w /
4, h / 2 - 70)
t1.write("Room A", align="center", font=("Arial", 20, "normal"))
t1.setpos(w / 4, h / 2 - 70)
t1.write("Room B", align="center", font=("Arial", 20, "normal"))

return A, B, X, Y

A, B, X, Y = setup()

#Vaccum Cleaner
C = Turtle()
C.speed(8)
C.penup()

```

```

C.shape("circle")
C.setpos(A.xcor(), A.ycor() + 130)

count = 1 iter =
Turtle() cleanwriter =
Turtle()
iter.ht() cleanwriter.ht()
iter.penup()
cleanwriter.penup()
iter.setpos(0, -h / 2 + 50)
cleanwriter.setpos(0, -h / 2 + 20)

room_state = list(Room_state.keys())

state = SimpleQueue()
state.put_nowait(((choice(room_state)), choice(room_state)))

while count <= 10:
    iter.clear()
    cleanwriter.clear()
    iter.write("Iteration : " + str(count), align="center", font=("Arial", 16, "normal"))
    cleanwriter.write("Times Cleaned : " + str(cleaned), align="center", font=("Arial", 16, "normal"))

    condition = state.get_nowait()
    stateA = condition[0]
    stateB = condition[1]

    X.clear()
    Y.clear()

    nextA = choice(room_state)
    nextB = choice(room_state)

    state.put_nowait((nextA, nextB))

    filler(A, Room_state[stateA])
    filler(B, Room_state[stateB])

    X.write("Now : " + stateA + "\nNext : " + nextA, align="center", font=("Arial", 16, "normal"))
    Y.write("Now : " + stateB + "\nNext : " + nextB, align="center", font=("Arial", 16, "normal"))

    print("\nA : " + stateA, "\tB : " + stateB)

    if stateA == 'Dirty' and stateB == 'Dirty':
        if C.xcor() < 0:
            print("Both Dirty, Cleaned A going to B")
# noinspection PyTypeChecker    filler(A,

```



```

Room_state['Clean'], 0.5, True)      stateA =
'Clean'      C.setpos(B.xcor(), B.ycor() + 130)
# noinspection PyTypeChecker      filler(B,
Room_state['Clean'], 0.5, True)
      stateB = 'Clean'

      elif C.xcor() > 0:
          print("Both Dirty, Cleaned B going to A")
# noinspection PyTypeChecker      filler(B,
Room_state['Clean'], 0.5, True)      stateB =
'Clean'      C.setpos(A.xcor(), A.ycor() + 130)
# noinspection PyTypeChecker      filler(A,
Room_state['Clean'], 0.5, True)
      stateA = 'Clean'

      if stateA == 'Dirty':      print("Cleaned
A")      C.goto(A.xcor(), A.ycor() + 130)
# noinspection PyTypeChecker      filler(A,
Room_state['Clean'], 0.3, True)

      elif stateB == 'Dirty':
print("Cleaned B")
C.goto(B.xcor(), B.ycor() + 130)      #
noinspection PyTypeChecker
      filler(B, Room_state['Clean'], 0.3, True)

      count += 1
      sleep(0.5)

```

Output:

A : Clean B : Dirty
Cleaned B

A : Dirty B : Clean
Cleaned A

A : Dirty B : Clean
Cleaned A

A : Dirty B : Dirty
Both Dirty, Cleaned A going to B

A : Clean B : Dirty
Cleaned B

A : Dirty B : Clean
Cleaned A

A : Dirty B : Dirty
Both Dirty, Cleaned A going to B

A : Clean B : Dirty
Cleaned B

A : Dirty B : Clean
Cleaned A

A : Dirty B : Clean
Cleaned A

>>>

```
Both Dirty, Cleaned B going to A
>>>
==== RESTART: C:/Users/Kajal Padhiyar/Desktop/python programs/aiscexp8.py ====
A : Clean          B : Dirty
Cleaned B

A : Dirty          B : Clean
Cleaned A

A : Dirty          B : Clean
Cleaned A

A : Dirty          B : Dirty
Both Dirty, Cleaned A going to B

A : Clean          B : Dirty
Cleaned B

A : Dirty          B : Clean
Cleaned A

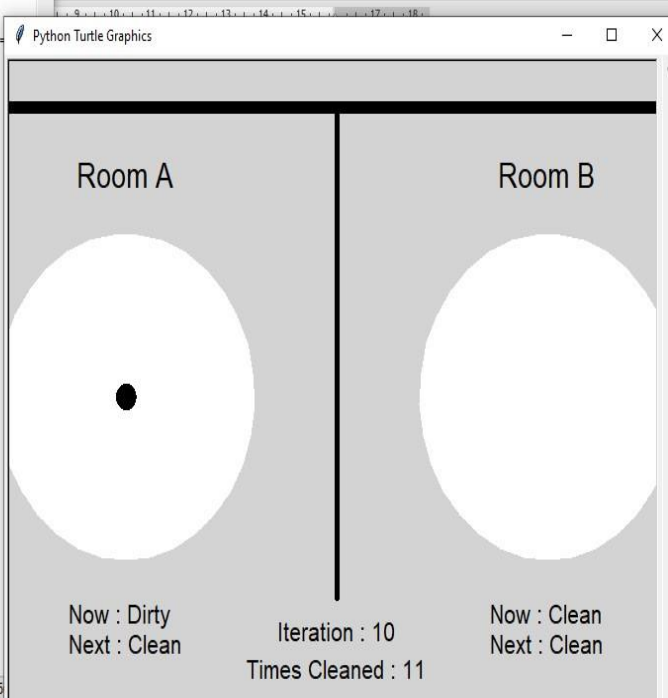
A : Dirty          B : Dirty
Both Dirty, Cleaned A going to B

A : Clean          B : Dirty
Cleaned B

A : Dirty          B : Clean
Cleaned A

A : Dirty          B : Clean
Cleaned A

>>>
```



Conclusion:

We have successfully implemented agent using artificial intelligence which reach the goal of cleaning dirt that resides in rooms.

EXPERIMENT NO-9

Aim: Implement Hebbian Learning Theory:

The Hebbian Learning Rule is a learning rule that specifies how much the weight of the connection between two units should be increased or decreased in proportion to the product of their activation. The rule builds on Hebb's 1949 learning rule which states that the connections between two neurons might be strengthened if the neurons fire simultaneously.. The requirement of orthogonality places serious limitations on the Hebbian Learning Rule.

Hebb's Law can be represented in the form of two rules:

1. If two neurons on either side of a connection are activated synchronously, then the weight of that connection is increased.
2. If two neurons on either side of a connection are activated asynchronously, then the weight of that connection is decreased.

Hebb's Law provides the 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 the weight w_{ij} at iteration p in the following form:

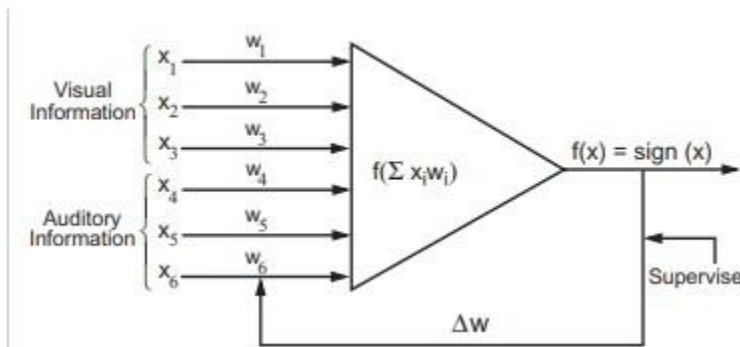
$$\Delta w = c * f(X, W) * X$$

Where c is learning rate, $f(X, W)$ is output of neuron and X is input vector.

$$F(X, W) = \text{sign}(W * X)$$

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 a non-linear **forgetting factor** into Hebb's Law: where j is the forgetting factor. Forgetting factor usually falls in the interval between 0 and 1, typically between 0.01 and 0.1, to allow only a little "forgetting" while limiting the weight growth.

Hebbian learning can be used to study Pavlov's experiments, where by simultaneously ringing a bell every time food was presented, a dog's salivation response to food was transferred to the bell. Here two layers neural network is used, an input layer with six nodes and an output layer with one node. The output layer returns either +1, signifying that the output neuron has fired, or a -1, signifying that it is quiescent.



We let the learning constant be the small positive real number 0.2. In this example the network is trained on the pattern $[1, -1, 1, -1, 1, -1]$ which is the concatenation of the two patterns, $[1, -1, 1]$ and $[-1, 1, -1]$. The pattern $[1, -1, 1]$ represents the unconditioned stimulus and $[-1, 1, -1]$ represents the new stimulus.

We let the learning constant be the small positive real number 0.2. In this example we train the network on the pattern $[1, -1, 1, -1, 1, -1]$ which is the concatenation of the two patterns, $[1, -1, 1]$ and $[-1, 1, -1]$. The pattern $[1, -1, 1]$ represents the unconditioned stimulus and $[-1, 1, -1]$ represents the new stimulus.

We assume that the network already responds positively to the unconditioned stimulus but is neutral with respect to the new stimulus. We simulate the positive response of the network to the unconditioned stimulus with the weight vector $[1, -1, 1]$, exactly matching the input pattern, while the neutral response of the network to the new stimulus is simulated by the weight vector $[0, 0, 0]$. The concatenation of these two weight vectors gives us the initial weight vector for the network, $[1, -1, 1, 0, 0, 0]$.

We now train the network on the input pattern, hoping to induce a configuration of weights which will produce a positive network response to the new stimulus. The first iteration of the network gives:

$$\begin{aligned} W * X &= (1 * 1) + (-1 * -1) + (1 * 1) + (0 * -1) + (0 * 1) + (0 * -1) \\ &= (1) + (1) + (1) = 3 \quad f(3) \\ &= \text{sign}(3) = 1. \end{aligned}$$

We now create the new weight W2:

$$\begin{aligned} W2 &= [1, -1, 1, 0, 0, 0] + .2 * (1) * [1, -1, 1, -1, 1, -1] \\ &= [1, -1, 1, 0, 0, 0] + [.2, -.2, .2, -.2, .2, -.2] \\ &= [1.2, -1.2, 1.2, -.2, .2, -.2] \end{aligned}$$

We expose the adjusted network to the original input pattern:

$$\begin{aligned} W * X &= (1.2 * 1) + (-1.2 * -1) + (1.2 * 1) + (-.2 * -1) + (.2 * 1) + (-.2 * -1) \\ &= (1.2) + (1.2) + (1.2) + (.2) + (.2) + (.2) = 4.2 \quad \text{and} \quad \text{sign}(4.2) \\ &= 1. \end{aligned}$$

We now create the new weight W3:

$$\begin{aligned} W3 &= [1.2, -1.2, 1.2, -.2, .2, -.2] + .2 * (1) * [1, -1, 1, -1, 1, -1] \\ &= [1.2, -1.2, 1.2, -.2, .2, -.2] + [.2, -.2, .2, -.2, .2, -.2] \\ &= [1.4, -1.4, 1.4, -.4, .4, -.4] \end{aligned}$$

It can now be seen that the vector product, $W * X$, will continue to grow in the positive direction, with the absolute value of each element of the weight vector increasing by .2 at each training cycle. After 10 more iterations of Hebbian training the weight vector will be:

$$W13 = [3.4, -3.4, 3.4, -2.4, 2.4, -2.4].$$

We now use this trained weight vector to test the network's response to the two partial patterns. We would like to see if the network continues to respond to the unconditioned stimulus positively and, more importantly, if the network has now acquired a positive response to the new, conditioned stimulus. We test the network first on the unconditioned stimulus $[1, -$

1, 1]. We fill out the last three arguments of the input vector with random 1, and -1 assignments. For example, we test the network on the vector [1, -1, 1, 1, 1, -1]:

$$\begin{aligned}\text{sign}(W*X) &= \text{sign}((3.4*1) + (-3.4*-1) + (3.4*1) \\ &+ (-2.4*1) + (2.4*1) + (-2.4*-1)) = \\ \text{sign}(3.4 + 3.4 + 3.4 - 2.4 + 2.4 + 2.4) &= \\ \text{sign}(12.6) &= +1.\end{aligned}$$

The network thus still responds positively to the original unconditioned stimulus. We now do a second test using the original unconditioned stimulus and a different random vector in the last three positions [1, -1, 1, 1, -1, -1]:

$$\begin{aligned}\text{sign}(W*X) &= \text{sign}((3.4*1) + (-3.4*-1) + (3.4*1) \\ &+ (-2.4*1) + (2.4*-1) + (-2.4*-1)) = \\ \text{sign}(3.4 + 3.4 + 3.4 - 2.4 - 2.4 + 2.4) &= \\ \text{sign}(7.8) &= +1.\end{aligned}$$

The second vector also produces a positive network response. In fact we note in these two examples that the network's sensitivity to the original stimulus, as measured by its raw activation, has been strengthened, due to repeated exposure to that stimulus. We now test the network's response to the new stimulus pattern, [-1, 1, -1], encoded in the last three positions of the input vector. We fill the first three vector positions with random assignments from the set {1, -1} and test the network on the vector [1, 1, 1, -1, 1, -1]:

$$\begin{aligned}\text{sign}(W*X) &= \text{sign}((3.4*1) + (-3.4*-1) + (3.4*1) \\ &+ (-2.4*1) + (2.4*1) + (-2.4*-1)) = \\ \text{sign}(3.4 - 3.4 + 3.4 + 2.4 + 2.4 + 2.4) &= \\ = \text{sign}(10.6) &= +1\end{aligned}$$

We do one final experiment, with the vector patterns slightly degraded. This could represent the stimulus situation where the input signals are slightly altered, perhaps because a new food and a different sounding bell are used. We test the network on the input vector [1, -1, -1, 1, 1, -1], where the first three parameters are one off the original unconditioned stimulus and the last three parameters are one off the conditioned stimulus:

$$\begin{aligned}\text{sign}(W*X) &= \text{sign}((3.4*1) + (-3.4*-1) + (3.4*1) \\ &+ (-2.4*1) + (2.4*1) + (-2.4*-1)) = \\ \text{sign}(3.4 + 3.4 - 3.4 - 2.4 + 2.4 + 2.4) &= \\ \text{sign}(5.8) &= +1.\end{aligned}$$

Even the partially degraded stimulus is recognized.

Conclusion:-Thus, an association is created between a new stimulus and an old response by repeatedly presenting the old and new stimulus together. The network learns to transfer its response to the new stimulus without any supervision using hebbian learning.

Program:

```

clc; np=input("Enter the number of
Patterns"); ne=input("Enter the elements in
Patterns"); for x=1:1:ne
X1=input('Enter the pattern');    for
j=1:1:ne    X(x,j)=X1(j);    end end
w=input('Enter the initial Weights'); for
q=1:1:np    T=X(q,1:ne);    net1=W*T';
if(net<0)    o1=-1;    else    o1=1
    end
    w=w+o1*T

end

```

Output:

Enter the number of pattern :4

Enter the number of pattern :2

Enter the pattern:[1 -2]

Enter the pattern:[0 1]

Enter the pattern:[2 3]

Enter the pattern:[1 -1]

Enter initial weights:[1 -1]

W=2 -3

W=2 -4

W=0 -7

W=1 -8

Final Weight=1 -8

Experiment No 9 Code:

```
x1=[1,1,-1,-1] x2=[1,-  
1,1,-1] y=[-1,1,-1,-1]  
b=[1,1,1,1]  
w1,w2,b=0,0,0  
  
for j in range(2):  
    print("b =",b,"\tW1 =",w1,"\tW2 =",w2)  
    w1,w2,b,nl=[],[],[],[]  
    for i in range(4):  
        w1=w1+x1[i]*y[i]  
        w2=w2+x2[i]*y[i]  
        b=b+y[i] w1.append(w1)  
        w2.append(w2)  
        bl.append(b)  
  
    for i in range(4):  
        net=w1*x1[i]+w2*x2[i]+b  
        nl.append(net)  
  
    print("X1\tX2\tY\tb\tW1\tW2\tnet") for i in range(4):  
    print(x1[i],"\t",x2[i],"\t",y[i],"\t",bl[i],"\t",w1[i],"\t",w2[i],"\t",nl[i])  
    print()
```

Output:

```
b = 0 W1 = 0          W2 = 0  
X1    X2    Y    b    W1    W2    net  
1      1    -1   -1    -1    -1    -2  
1     -1     1    0     0    -2     2  
-1 1 -1 -1 1 -3 -6  
  
-1 -1 -1 -2 2 -2 -2
```

```
b = -2 W1 = 2          W2 = -2  
X1    X2    Y    b    W1    W2    net  
1      1    -1   -3     1    -3    -4  
1     -1     1   -2     2    -4     4  
-1     1    -1   -3     3    -5   -12  
-1    -1    -1   -4     4    -4    -4
```