**

***SOFT-COMPUTING LAB MANUAL***

**SUBMITTED TO:**

**MR. SANDIP KUMAR CHAURASIYA**

**SUBMITTED BY:**

**NAME: ROHIT CHAUHAN**

**SAP ID: 500110684**

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| 1. **No.** | **Objective of the Experiment** | **PAGE N0.** | **Remarks** |
| **1.** | **EXPERIMENT-1** | **3** |  |
| **2.** | **EXPERIMENT-2** | **7** |  |
| **3.** | **EXPERIMENT-3** | **12** |  |
| **4.** | **EXPERIMENT-4** | **15** |  |
| **5.** | **EXPERIMENT-5** | **18** |  |
| **6.** | **EXPERIMENT-6** | **25** |  |
| **7.** | **EXPERIMENT-7** | **30** |  |
| **8.** | **EXPERIMENT-8** | **35** |  |
| **9.** | **EXPERIMENT-9** | **37** |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**EXPERIMENT 1**

**Create a perceptron with appropriate no. of inputs and outputs. Train it using fixed increment learning algorithm until no change in weights is required. Output the final weights.**

**(MATLAB)**

**CODE-**

lr = 0.01;

w1 = 0.04;

w2 = 0.3;

x=[0 0 1 1];

y=[0 1 0 1];

choice = (input("\n1. And \n2. OR \n3. NAND \n4. NOR \n5. XOR \n\nEnter your choice :"));

switch choice

case 1

disp('choice 1');

label=[0 0 0 1];

case 2

label=[0 1 1 1];

case 3

label=[1 1 1 0];

case 4

label=[1 0 0 0];

case 5

label=[0 1 1 0];

otherwise

disp('Wrong choice');

end

for j=1:4

fprintf("\nEpoch - %d\n\n", j);

fprintf(" x1 \t\t x2 \t\t t \t\t yin \t\t y \t\t w1 \t\t w2\n");

output=[];

error=[];

for i=1:4

fprintf('\n');

fprintf("%4d \t\t",x1(i));

fprintf("%4d \t\t",x2(i));

fprintf("%4d \t\t",target(i));

temp=(w1\*x(i))+(w2\*y(i));

fprintf("%4d \t\t",temp);

if temp>0.5

temp = 1;

else

temp = 0;

endif

fprintf("%4d \t\t",temp);

output=[output,temp];

e=(label(i)-output(i))\*(label(i)-output(i));

error=[error, e];

d = sum(error);

w1=w1+lr\*d;

w2=w2+lr\*d;

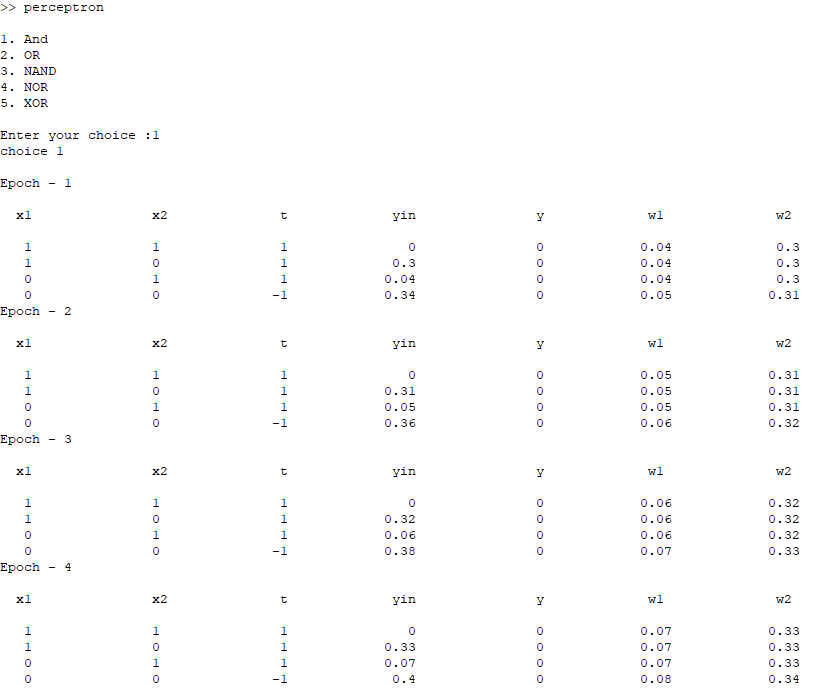
fprintf("%4d \t\t",w1);

fprintf("%4d \t\t",w2);

end;

end;

**OUTPUT-**



**EXPERIMENT 2**

**Que1- Nor Gate implementation with binary input and bipolar target using adaline. (MATLAB)**

**CODE-**

x1 = [1 1 -1 -1];

x2 = [1 -1 1 -1];

target = [-1 1 1 1];

lr = 0.1;

w1 = 0.1;

w2 = 0.1;

b = 0;

for j=1:5

fprintf("\nEpoch - %d\n\n", j);

fprintf(" x1 \t\t x2 \t\t t \t\t yin \t\t\t w1 \t\t\t w2 \t\t\t b \t\t\t Error\n");

output=[];

error=[];

d = 0;

for i=1:4

fprintf('\n');

fprintf("%4d \t\t",x1(i));

fprintf("%4d \t\t",x2(i));

fprintf("%4d \t\t",target(i));

temp = (w1\*x1(i))+(w2\*x2(i)) + b;

fprintf("%4d \t\t",temp);

if temp ~= target(i)

w1 = w1 + lr\*(target(i)-temp)\*x1(i);

w2 = w2 + lr\*(target(i)-temp)\*x2(i);

b = b + lr\*(target(i)-temp);

endif

fprintf("%4d \t\t",w1);

fprintf("%4d \t\t",w2);

fprintf("%4d \t\t",b);

e = (target(i) - temp) ^ 2;

fprintf("%4d \t\t",b);

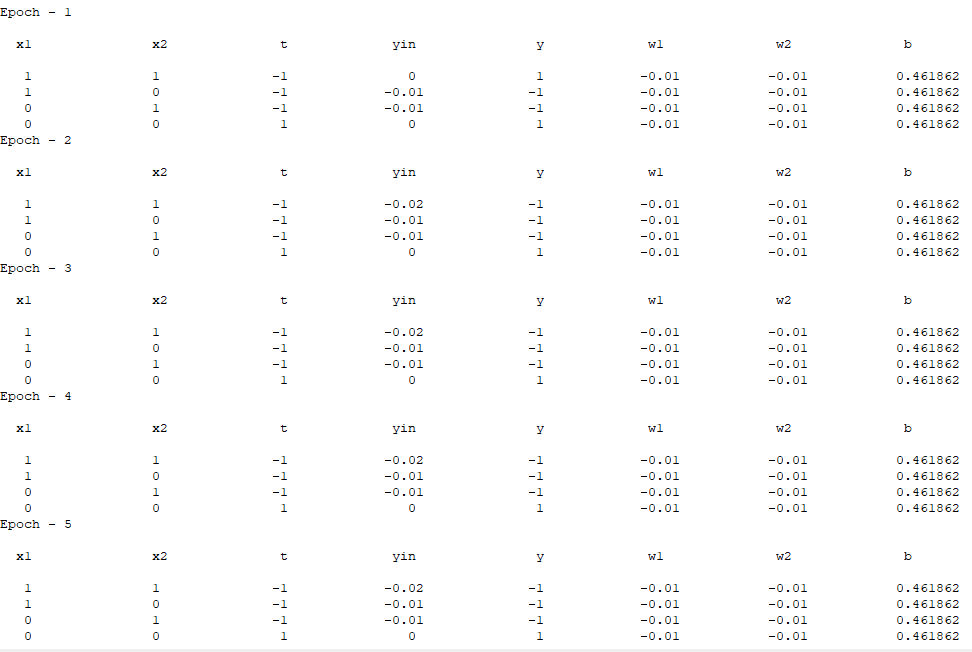
d = d + e;

endfor

%fprintf("Error = %d\n", d);

end;

**OUTPUT-**



**Que2- Nor Gate implementation with bipolar input and bipolar target using adaline.**

**CODE-**

x1 = [1 1 -1 -1];

x2 = [1 -1 1 -1];

target = [-1 1 1 1];

lr = 0.1;

w1 = 0.1;

w2 = 0.1;

b = 0;

for j=1:5

fprintf("\nEpoch - %d\n\n", j);

fprintf(" x1 \t\t x2 \t\t t \t\t yin \t\t\t w1 \t\t\t w2 \t\t\t b \t\t\t Error\n");

output=[];

error=[];

d = 0;

for i=1:4

fprintf('\n');

fprintf("%4d \t\t",x1(i));

fprintf("%4d \t\t",x2(i));

fprintf("%4d \t\t",target(i));

temp = (w1\*x1(i))+(w2\*x2(i)) + b;

fprintf("%4d \t\t",temp);

if temp ~= target(i)

w1 = w1 + lr\*(target(i)-temp)\*x1(i);

w2 = w2 + lr\*(target(i)-temp)\*x2(i);

b = b + lr\*(target(i)-temp);

endif

fprintf("%4d \t\t",w1);

fprintf("%4d \t\t",w2);

fprintf("%4d \t\t",b);

e = (target(i) - temp) ^ 2;

fprintf("%4d \t\t",b);

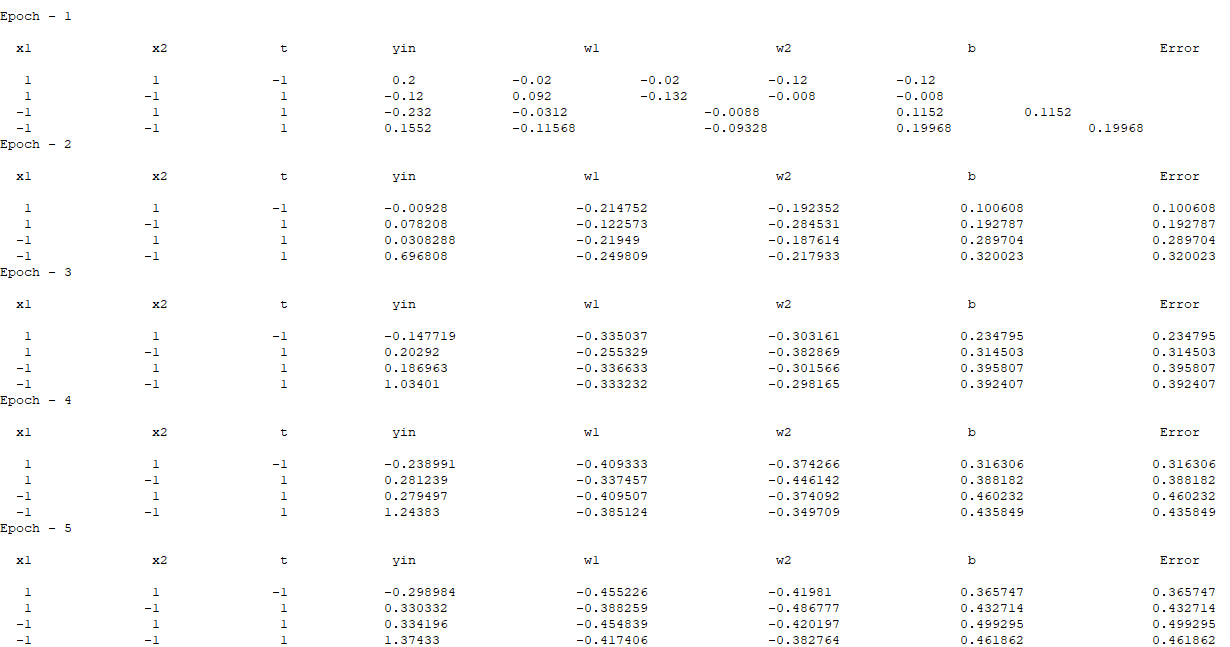
d = d + e;

endfor

%fprintf("Error = %d\n", d);

end;

**OUTPUT-**



**EXPERIMENT 3**

**XOR Gate implementation with bipolar input and bipolar target using madaline. (Using Python)**

**CODE :**

inp=[[-1,-1],[-1,1],[1,-1],[1,1]]

target=[-1,1,1,-1]

w11=0.05

w12=0.1

w21=0.2

w22=0.2

v0=0.5

v1=0.5

lr=0.5

b1=0.3

b2=0.15

b3=0.5

w=[0.0,0.0,0.0,0.0]

b=[0.0,0.0,0.0]

print("x1\tx2\tt\tz1\tz2\ty\tdw11\tdw12\tdw21\tdw22\tdwb1\tdwb2\tdwb3")

for k in range(4):

for i in range(4):

y1=inp[i][0]\*w11+inp[i][1]\*w21+b1

if y1<0:

y1=-1

else:

y1=1

y2=inp[i][0 ]\*w12+inp[i][1]\*w22+b2

if y2<0:

y2=-1

else:

y2=1

y3=y1\*v0+y2\*v1+b3

if y3<0:

y3=-1

else:

y3=1

if y3!=target:

error=target[i]-y3

w[0]=lr\*error\*inp[i][0]

w[1]=lr\*error\*inp[i][0]

w[2]=lr\*error\*inp[i][1]

w[3]=lr\*error\*inp[i][1]

b[0]=lr\*error

b[1]=lr\*error

b[2]=lr\*error

w11=w11+w[0]

w12=w12+w[1]

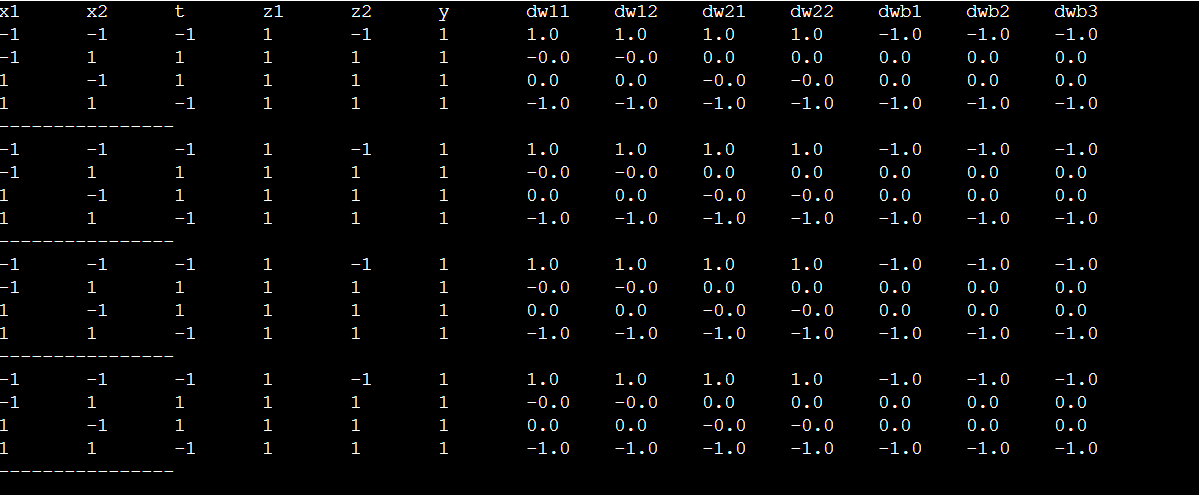
w21=w21+w[2]

w22=w22+w[2]

print(f"{inp[i][0]}\t{inp[i][1]}\t{target[i]}\t{y1}\t{y2}\t{y3}\t{w[0]}\t{w[1]}\t{w[2]}\t{w[3]}\t{b[0]}\t{b[1]}\t{b[2]}\t")

print("----------------")

**OUTPUT :**

****

**EXPERIMENT 4**

**Using back-propagation network, find the new weights. It is presented with the input pattern [0, 1] and the target output is 1. Use a learning rate α = 0.25 and binary sigmoidal activation function. (Using Python)**

**CODE :**

# using MADALINE N/W implement xor function with bbbpoler input and bbbpoler output,

bbb=[0.3,0.15,0.5]

www = [0.05,0.1,0.2,0.2]

w = [0.05,0.1,0.2,0.2]

v=[0.5,0.5]

lr =0.4

b=[1,1,1]

inp=[[-1,1,-1,1],[-1,-1,1,1]]

t=[1,-1,-1,-1]

dif=0

print("x1\tx2\tt\tz1\tz2\ty\tw0\tw1\tw2\tw3\tb1\tb2\tb3")

for i in range(4):

print(f"{inp[0][i]}\t{inp[1][i]}\t{t[i]}\tz1\tz2\ty\t{w[0]}\t{w[1]}\t{w[2]}\t{w[3]}\t{b[0]}\t{b[1]}\t{b[2]}")

for j in range(4):

print("============================================================================================================================================= ",dif)

for i in range(4):

z1=inp[0][i]\*w[0] + inp[1][i]\*w[1] +b[0]

z2=inp[0][i]\*w[2] + inp[1][i]\*w[3] +b[1]

# activation funciton on hidden layer

if z1<0:z1=-1

else: z1=1

if z2<0:z2=-1

else: z2=1

y= z1\*v[0] + z2\*v[1] +b[2]

# print(y)

# activation function in output layer

if y<0:

y =-1

else:

y =1

# print(f"{inp[0][i]}\t{inp[1][i]}\t{t[i]}\t{z1}\t{z2}\t{y}\t{w[0]}\t{w[1]}\t{w[2]}\t{w[3]}\t{b[0]}\t{b[1]}\t{b[2]}")

if y!=t[i]:

dif = t[i]-y

# print(dif)

w[0] += lr\*dif\*inp[0][i]

w[1] += lr\*dif\*inp[1][i]

w[2] += lr\*dif\*inp[0][i]

w[3] += lr\*dif\*inp[1][i]

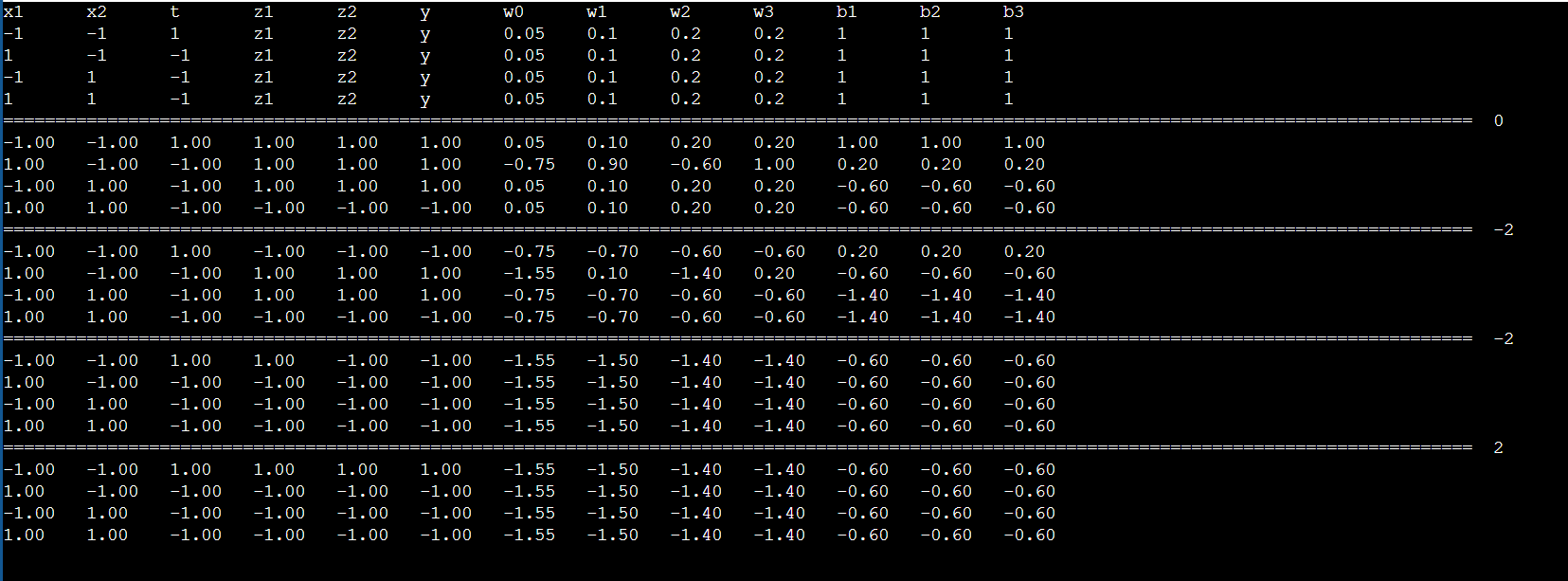
b[0] = b[0] + lr\*dif

b[1] = b[1] + lr\*dif

b[2] = b[2] + lr\*dif

print(f"{'%.2f' %inp[0][i]}\t{'%.2f' %inp[1][i]}\t{'%.2f' %t[i]}\t{'%.2f' %z1}\t{'%.2f' %z2}\t{'%.2f' %y}\t{'%.2f' %w[0]}\t{'%.2f' %w[1]}\t{'%.2f' %w[2]}\t{'%.2f' %w[3]}\t{'%.2f' %b[0]}\t{'%.2f' %b[1]}\t{'%.2f' %b[2]}")

**OUTPUT :**

****

**EXPERIMENT 5**

**FUZZY OPERATORS (Using Python)**

**CODE :**

A = dict()

B = dict()

Y = dict()

A = {"a": 0.5, "b": 0.7, "c": 0.3, "d": 0.4,"e":0.7,"f":0,"g":0.8,"h":0}

B = {"a": 0, "b": 0, "c": 0.9, "d": 0.1,"e":0,"f":0,"g":0.7,"h":0}

print('The First Fuzzy Set is :',"\n")

for A\_key in A:

print(A[A\_key],"/",A\_key," + ",end="")

print("\n")

print('The Second Fuzzy Set is :',"\n")

for B\_key in B:

print(B[B\_key],"/",B\_key," + ",end="")

print("\n")

# SET UNION

print('\n','FUZZY SET UNION IS : ')

for A\_key, B\_key in zip(A, B):

A\_value = A[A\_key]

B\_value = B[B\_key]

if A\_value > B\_value:

Y[A\_key] = A\_value

print('%.2f' %Y[A\_key],"/",A\_key," + ",end="")

else:

Y[B\_key] = B\_value

print('%.2f' %Y[B\_key],"/",B\_key," + ",end="")

print("\n")

# SET INTERSECTION

print('\n','FUZZY SET INTERSECTION IS : ')

for A\_key, B\_key in zip(A, B):

A\_value = A[A\_key]

B\_value = B[B\_key]

if A\_value < B\_value:

Y[A\_key] = A\_value

print('%.2f' %Y[A\_key],"/",A\_key," + ",end="")

else:

Y[B\_key] = B\_value

print('%.2f' %Y[B\_key],"/",B\_key," + ",end="")

print("\n")

# COMPLEMENT OF A

print("\n\n\n",'COMPLEMENT OF A IS :')

for A\_key in A:

Y[A\_key]= 1-A[A\_key]

print('%.2f' %Y[A\_key],"/",A\_key," + ",end="")

print("\n")

# COMPLEMENT OF B

print("\n\n\n",'COMPLEMENT OF B IS :')

for B\_key in B:

Y[B\_key]= 1-B[B\_key]

print('%.2f' %Y[B\_key],"/",B\_key," + ",end="")

print("\n")

# SET DIFFERENCE

print("\nFUZZY SET DIFFERENCE IS : ")

for A\_key, B\_key in zip(A, B):

A\_value = A[A\_key]

B\_value = B[B\_key]

B\_value = 1 - B\_value

if A\_value < B\_value:

Y[A\_key] = A\_value

print('%.2f' %Y[A\_key],"/",A\_key," + ",end="")

else:

Y[B\_key] = B\_value

print('%.2f' %Y[B\_key],"/",B\_key," + ",end="")

print("\n")

# SET ALGEBRAIC SUM

print("FUZZY SET ALGEBRAIC SUM IS : ")

Z = dict()

for A\_key,B\_key in zip(A,B):

A\_value=A[A\_key]

B\_value = B[B\_key]

Z[A\_key]= A\_value+B\_value-(A\_value\*B\_value)

print('%.2f' %Z[A\_key],"/",A\_key," + ",end="")

print("\n")

#SET ALGEBRAIC PRODUCT

print("\nFUZZY SET ALGEBRAIC PRODUCT IS :")

for A\_key,B\_key in zip(A,B):

A\_value=A[A\_key]

B\_value = B[B\_key]

Y[A\_key]= (A\_value\*B\_value)

print('%.2f' %Y[A\_key],"/",A\_key," + ",end="")

print("\n")

# BOUNDED SUM

print("\nBOUNDED SUM IS :")

for A\_key,B\_key in zip(A,B):

A\_value=A[A\_key]

B\_value = B[B\_key]

if(A\_value+B\_value < 1):

Y[A\_key]= A\_value+B\_value

print('%.2f' %Y[A\_key],"/",A\_key," + ",end="")

else:

Y[A\_key]= 1

print(Y[A\_key],"/",A\_key," + ",end="")

print("\n")

# BOUNDED DIFFERENCE

print("\nBOUNDED DIFFERENCE IS :")

for A\_key,B\_key in zip(A,B):

A\_value=A[A\_key]

B\_value = B[B\_key]

if(A\_value+B\_value > 0):

Y[A\_key]= A\_value+B\_value

print('%.2f' %Y[A\_key],"/",A\_key," + ",end="")

else:

Y[A\_key]= 0

print(Y[A\_key],"/",A\_key," + ",end="")

print("\n")

# CUBE OF A

print("\n\n\n",'FUZZY SET CUBE IS :')

for A\_key in A:

Y[A\_key]= pow(A[A\_key],3)

print('%.2f' %Y[A\_key],"/",A\_key," + ",end="")

print("\n")

#Height of A

a=0

for A\_key in A:

if(A[A\_key]>a):

a=A[A\_key]

print('HEIGHT OF A is :', '%.2f' %a,"\n\n\n")

#Height of B

b=0

for B\_key in B:

if(B[B\_key]>b):

b=B[B\_key]

print('HEIGHT OF B is :','%.2f' % b,"\n\n\n")

#CARDINALITY OF A

card=0

for A\_key in A:

card= card+ A[A\_key]

print('CARDINALITY of A is :','%.2f' % card,"\n\n\n")

#CARDINALITY OF B

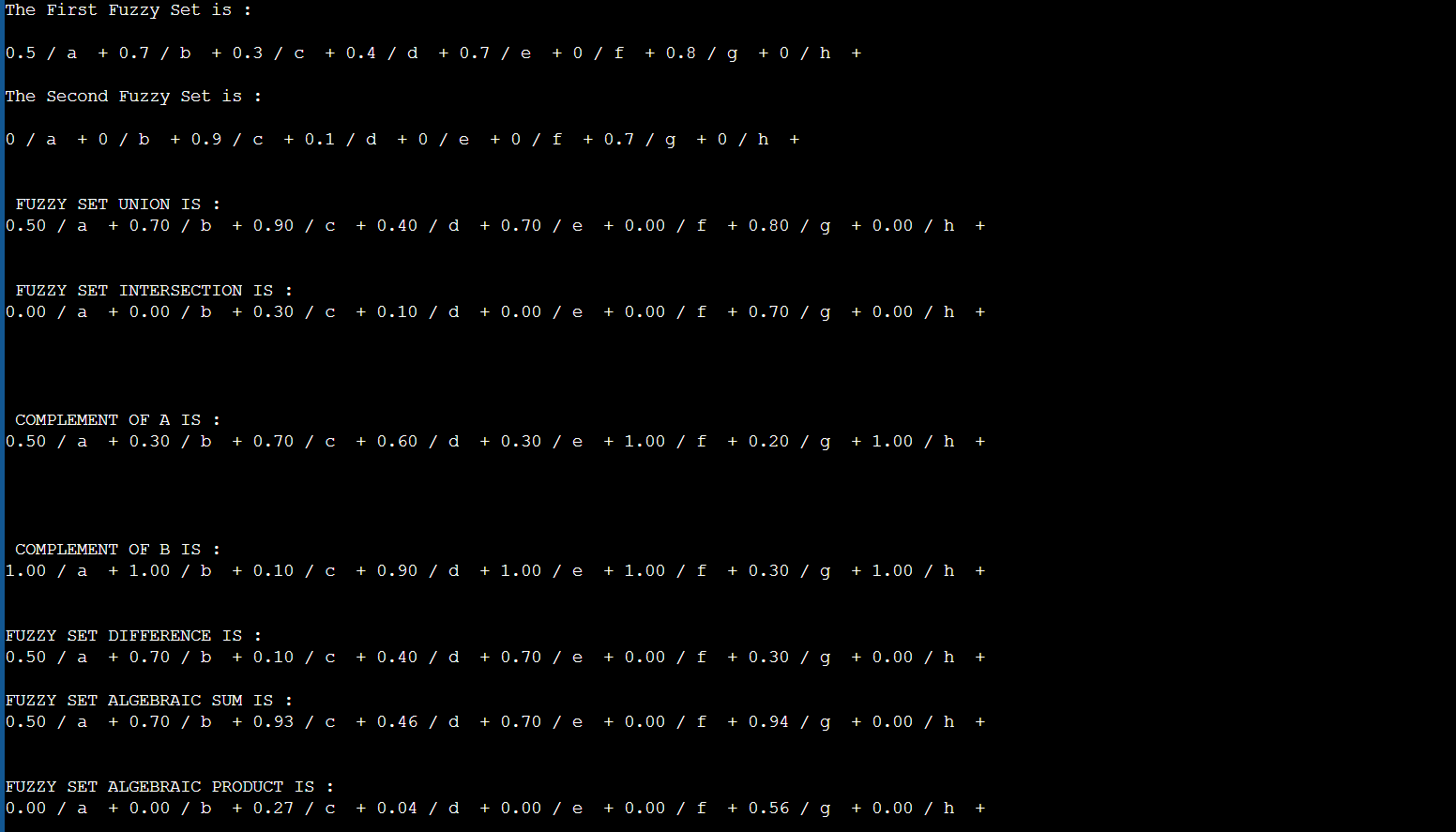
card1=0

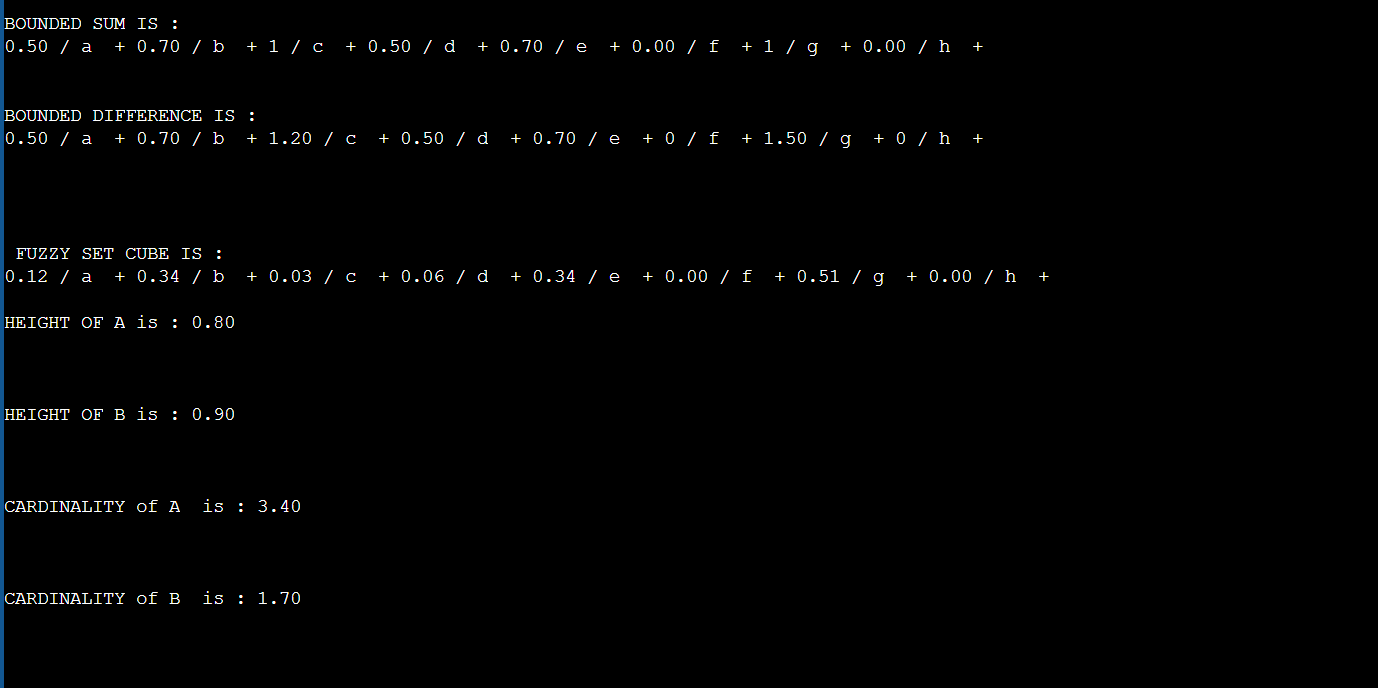
for B\_key in B:

card1= card1+ B[B\_key]

print('CARDINALITY of B is :', '%.2f' %card1,"\n\n\n")

**OUTPUT :**





**EXPERIMENT-6**

**Que1-Write programe that ask user to enter two fuzzy sets and computes the resultant fuzzy relation upto 10 greater than 6.**

**Octave Program-**

% Ask user to enter the two fuzzy sets

% Ask the user to input the first fuzzy set

disp("Enter the first fuzzy set elements between 6-10:");

A = input("A = ");

if length(A)>=6 && length(A)<=10

disp("Enter the second fuzzy set elements between 6-10::");

B = input("B = ");

if length(B)>=6 && length(B)<=10

R = zeros(length(A),length(B));

for i = 1:length(A)

for j = 1:length(B)

R(i,j) = min(A(i),B(j));

endfor

endfor

% Display the resultant fuzzy relation

disp("The resultant fuzzy relation is:")

disp(R)

else

disp("Please enter the setB elements within range of 6-10")

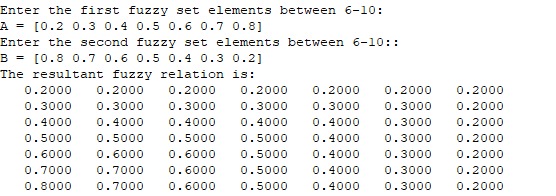
end

else

disp("Please enter the setA elements within range of 6-10")

End

**Output-**

****

**Que2-Create two matrix of the dimension two matrix of dimension defined by user containing random number(0-1) as there elements perform the union intersection and complement operation on these two matrices treating them to be fuzzy relation.**

**Octave Program-**

% user to input the dimensions of the matrices

rows = input('Enter the number of rows for the matrices: ');

cols = input('Enter the number of columns for the matrices: ');

% two matrices with random numbers between 0 and 1

matrix1 = rand(rows, cols);

matrix2 = rand(rows, cols);

% union operation

union\_matrix = max(matrix1, matrix2);

% intersection operation

intersection\_matrix = min(matrix1, matrix2);

% complement operation

complement\_matrix1 = 1 - matrix1;

complement\_matrix2 = 1 - matrix2;

% original matrices and the resulting matrices

disp('Matrix 1:')

disp(matrix1)

disp('Matrix 2:')

disp(matrix2)

disp('Union:')

disp(union\_matrix)

disp('Intersection:')

disp(intersection\_matrix)

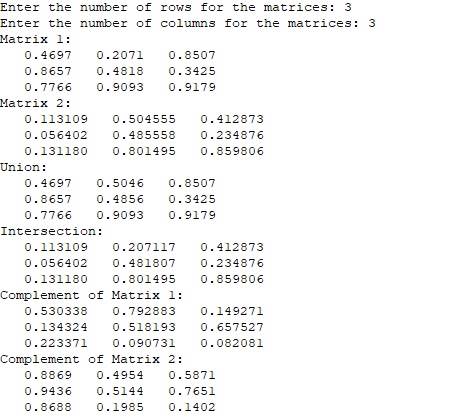
disp('Complement of Matrix 1:')

disp(complement\_matrix1)

disp('Complement of Matrix 2:')

disp(complement\_matrix2)

**Output-**

****

**Que3-create two matrices of the dimension 3x3 and 3x4 respectively which contain random number as there elements. Compute composition of these two fuzzy relation using both max-min and max-product composition.**

**Octave Program-**

m1 = 3;

n1 = 3;

m2 = 3;

n2 = 4;

% Create the two matrices with random numbers

A = rand(m1, n1);

B = rand(m2, n2);

% Compute the max-min composition

R\_mm = zeros(m1, n2);

for i = 1:m1

for j = 1:n2

max\_min = -Inf;

for k = 1:n1

max\_min = max(max\_min, min(A(i,k), B(k,j)));

endfor

R\_mm(i,j) = max\_min;

endfor

endfor

% Compute the max-product composition

R\_mp = zeros(m1, n2);

for i = 1:m1

for j = 1:n2

max\_prod = -Inf;

for k = 1:n1

max\_prod = max(max\_prod, A(i,k)\*B(k,j));

endfor

R\_mp(i,j) = max\_prod;

endfor

endfor

% Display the results

disp("Matrix A:")

disp(A)

disp("Matrix B:")

disp(B)

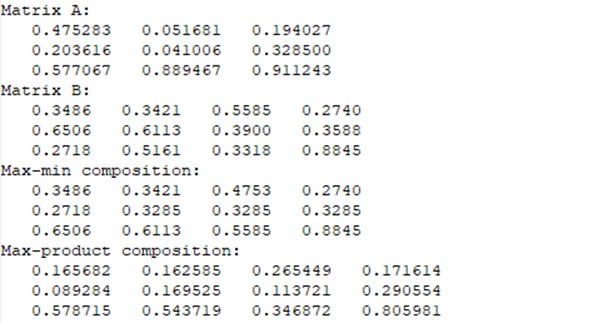
disp("Max-min composition:")

disp(R\_mm)

disp("Max-product composition:")

disp(R\_mp)

**Output-**

****

**EXPERIMENT-7**

**Que-write a program that creates two random fuzzy sets of the dimensions say n and m (to be defined by the user) Complete the fuzzy relation indexed by Cartesian product of the sets.**

**Octave Program-**

for i=1:3

A{1}(i) = input("Enter value for set A : ");

B{1}(i) = input("Enter value for set B : ");

end

%Display fuzzy set A

fprintf("A = { ");

for i=1:3

fprintf("%d, ", A{1}(i));

end

fprintf("}\n");

%Display fuzzy set B

fprintf("B = { ");

for i=1:3

fprintf("%d, ", B{1}(i));

end

R = {[],[]};

%Display cartersian relation

fprintf("}\n\nRelation : \n\n")

for i=1:3

for j=1:3

R{i}(j) = min(A{1}(i), B{1}(j));

fprintf("%d\t",R{i}(j));

endfor

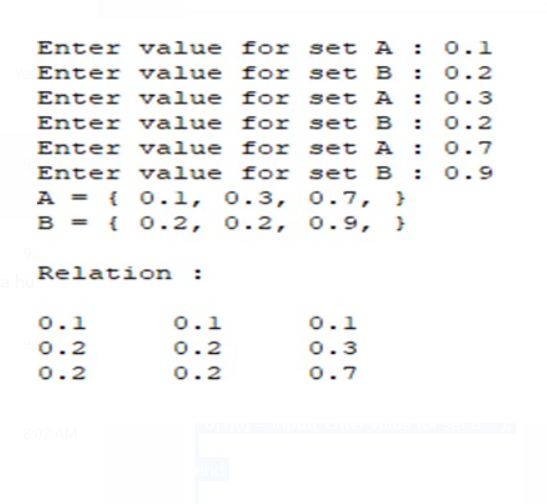
fprintf("\n");

end

fprintf("\n");

end

**Output-**

****

**Ques - consider a set P={p1, p2, p3, p4, p5} of five verities of plants. set D= {D1, D2, D3, D4, D5} of the various diseases affecting the plants and S={s1, s2, s3, s4, s5} be the common symptoms. let R=P X D & Q = D X S**

**such as R=[{0.6, 0.6, 0.4, 0.9},**

**{0.1, 0.2, 0.2, 0.9},**

**{0.9, 0.3, 0.1, 1},**

**{0.9, 0.3, 0.6, 0},**

**{0.7, 0.7, 0.7, 0.2}]**

**Q=[{0.4, 0.2, 1, 0.5, 0.6},**

**{0.3, 0.3, 1, 0.4, 0.6},**

**{0.2, 0, 0.9, 0.3, 0.6},**

**{0.5, 0, 0.8, 0, 0.7},**

**{0.6, 1, 0.7, 0, 0.5}]**

**obtain the association of plants with different symptoms of diseases using max-min and max-product.**

**Octave Program-**

% Define the matrices R and Q

R = [0.6, 0.6, 0.4, 0.9,0.7;

0.1, 0.2, 0.2, 0.9,0.6;

0.9, 0.3, 0.1, 1,0.2;

0.9, 0.3, 0.6, 0,0.3;

0.7, 0.7, 0.7, 0.2,0.4];

Q = [0.4, 0.2, 1, 0.5, 0.6;

0.3, 0.3, 1, 0.4, 0.6;

0.2, 0, 0.9, 0.3, 0.6;

0.5, 0, 0.8, 0, 0.7;

0.6, 1, 0.7, 0, 0.5];

% Convert matrices R and Q to fuzzy sets

R\_fuzzy = num2cell(R, 1);

Q\_fuzzy = num2cell(Q, 1);

% Obtain association using max-min

association\_maxmin = zeros(size(R, 1), size(Q, 1)); % Initialize the association matrix with zeros

for i = 1:size(R, 1)

for j = 1:size(Q, 1)

association\_maxmin(i, j) = max(min(R\_fuzzy{i}, Q\_fuzzy{j})); % Max-min operator for association

end

end

% Display the association matrix obtained using max-min

disp("Association matrix obtained using max-min:");

disp(association\_maxmin);

% Obtain association using max-product

association\_maxprod = zeros(size(R, 1), size(Q, 1)); % Initialize the association matrix with zeros

for i = 1:size(R, 1)

for j = 1:size(Q, 1)

association\_maxprod(i, j) = max(R\_fuzzy{i}.\*Q\_fuzzy{j}); % Max-product operator for association

end

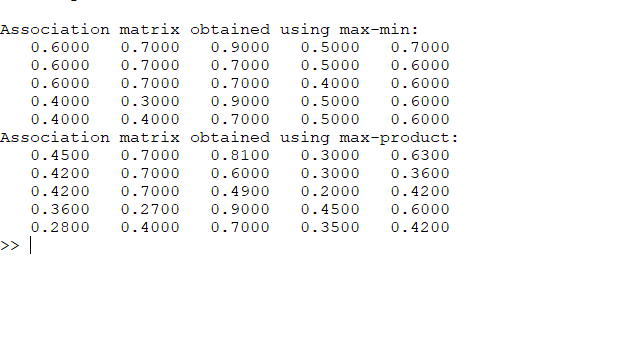
end

% Display the association matrix obtained using max-product

disp("Association matrix obtained using max-product:");

disp(association\_maxprod);

**Output-**



**EXPERIMENT-8**

**Que-Train an autocorrelator network for the pattern [1,-1,1,1] and also test the new weight for one missing and one mistake entry in the test vector respectively.**

**Octave Program-**

%input pattern%

x=[ 1 -1 1 1];

disp('inputs are -');

disp(x);

%target value%

y=[ 1 -1 1 1];

disp('target value-');

disp(y);

%transpose of the x %

xt=x';

disp('transpose of the x');

disp(xt);

w=zeros(length(xt),length(y));

for i=1:length(xt)

for j=1:length(y)

w(i,j)=xt(i)\*y(j);

endfor

endfor

disp('weight are');

disp(w);

disp('yin -');

yin=x\*w;

disp(yin);

for i=1:length(yin)

if (yin(i)>=0)

yin(i)=1;

elseif(yin(i)<0)

yin(i)=-1;

endif

endfor

if yin=y

fprintf("sucessful tested");

else

fprint("Error");

endif

**Output-**



**EXPERIMENT-9**

**Que-Train the autocorrelator by given patterns: A1=(-1,1,-1,1), A2=(1,1,1,-1),**

**A3=(-1, -1, - 1, 1). Test it using patterns: Ax=(-1,1,-1,1), Ay=(1,1,1,1), Az=(-1,-1,-1,-1).**

**Octave Code-**

A1 = [-1 1 -1 1];

A2 = [1 1 1 -1];

A3 = [-1 -1 -1 1];

tA1 = [-1 1 -1 1];

tA2 = [1 1 1 1];

tA3 = [-1 -1 -1 -1];

w1 = A1'\*A1;

w2 = A2'\*A2;

w3 = A3'\*A3;

w = w1 + w2 + w3;

disp(w);

yin1 = tA1\*w';

yin2 = tA2\*w';

yin3 = tA3\*w';

y1 = [];

y2 = [];

y3 = [];

for i = yin1

if i < 0

y1 = [y1 -1];

else

y1 = [y1 1];

end

end

for i = yin2

if i < 0

y2 = [y2 -1];

else

y2 = [y2 1];

end

end

for i = yin3

if i < 0

y3 = [y3 -1];

else

y3 = [y3 1];

end

end

disp("Train Set : ")

disp(A1)

disp(A2)

disp(A3)

disp("Train Output : ")

disp(y1)

disp(y2)

disp(y3)

disp("Testing Set: ")

disp(tA1)

disp(tA2)

disp(tA3)

disp("Testing Output : ")

disp(y1)

disp(y2)

disp(y3)

**Output-**

