

**MAHARASHTRA COLLEGE OF ARTS,SCIENCE & COMMERCE
MUMBAI
400-008**

DEPARTMENT OF INFORMATION TECHNOLOGY

MSC I.T PART-1



CERTIFICATE

This is to certify that the practical done at MAHARASHTRA COLLEGE By
Ms/Mr. _____

(Seat No. _____) in partial fulfillment for M.Sc.Degree Examination has
been found satisfactory. This practical journal had not been submitted for any other
examination and does not form part of any other course undergone by candidate.

Signature
Lecture-In-Charge
Guided by

Signature
External Examiner
Examined By

Signature
Course Coordinator
Certified By

College Stamp

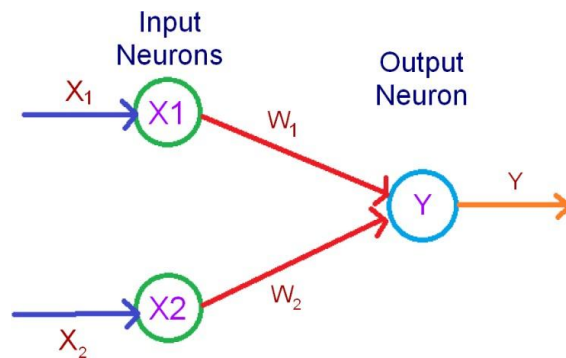
SOFT COMPUTING TECHNIQUES

INDEX

Practical No		Name of the Practical	Date	Sign.
1	A	Design a simple linear neural network model.		
	B	Calculate the output of neural net using both binary and bipolar sigmoidal function.		
2	A	Generate AND/NOT function using McCulloch-Pitts neural net.		
	B	Generate XOR function using McCulloch-Pitts neural net.		
3	A	Write a program to implement Hebb's rule.		
	B	Write a program to implement Delta rule.		
4	A	Write a program for Linear separation.		
5	A	Membership and Identity Operators is, is not		
	B	Membership and Identity Operators in, not in		
6	A	Find ratio using Fuzzy logic.		
	B	Solve Tipping problem using Fuzzy Logic.		
7	A	Implementation of Simple genetic algorithm.		

Practical : 01

Design a simple linear neural network model



The above diagram represents a simple neural network, having one input and one output layer, there are two neurons at the input layer and one at the output.

The input and the output layers are interconnected by weights, we are not considering the bias ($b = 0$)

The inputs are x_1 and x_2 to the neurons X1 and X2 respectively

The neuron X1 is connected by weight w_1 and X2 by weight w_2 to the output neuron respectively.

The threshold value for activation is 0 as represented by the activation function below

The net input to the output is

$$y_{in} = x_1 w_1 + x_2 w_2$$

In the present case we use the following activation for the output neuron

$$Y_{out} = \begin{cases} 0 & \text{if } y_{in} \leq 0 \\ 1 & \text{if } y_{in} > 0 \end{cases}$$

Practical : 01

Implement the following:

1a. Code : **Design a simple linear neural network model.**

```
print("Enter Value of X1 =")
x1 = int(input())

print("Enter Value of X2 =")
x2 = int(input())

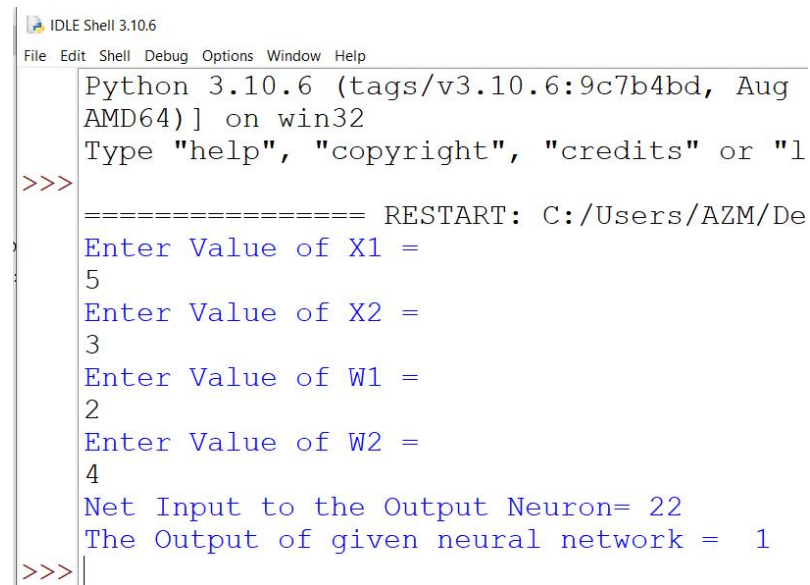
print("Enter Value of W1 =")
w1 = int(input())

print("Enter Value of W2 =")
w2 = int(input())

yin = x1*w1 + x2*w2
print("Net Input to the Output Neuron=",yin)

if yin <= 0:
    yout = 0
else :
    yout = 1
print("The Output of given neural network = ",yout)
```

Output :



```
IDLE Shell 3.10.6
File Edit Shell Debug Options Window Help
Python 3.10.6 (tags/v3.10.6:9c7b4bd, Aug
AMD64) on win32
Type "help", "copyright", "credits" or "1
>>> ===== RESTART: C:/Users/AZM/De
Enter Value of X1 =
5
Enter Value of X2 =
3
Enter Value of W1 =
2
Enter Value of W2 =
4
Net Input to the Output Neuron= 22
The Output of given neural network = 1
>>> |
```

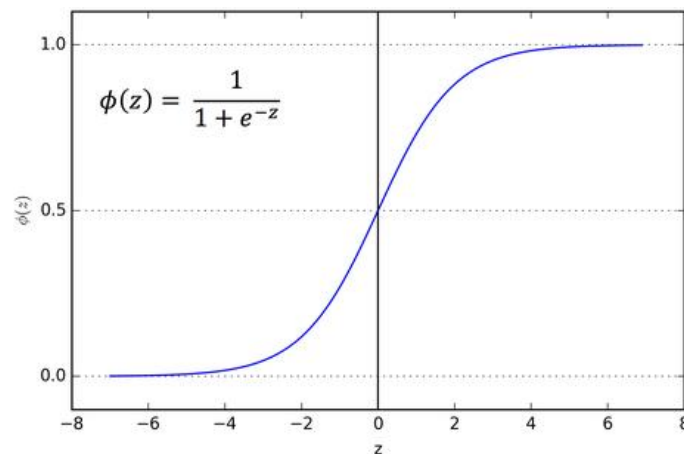
Practical : 1B

Neural net using both binary and bipolar sigmoidal function.

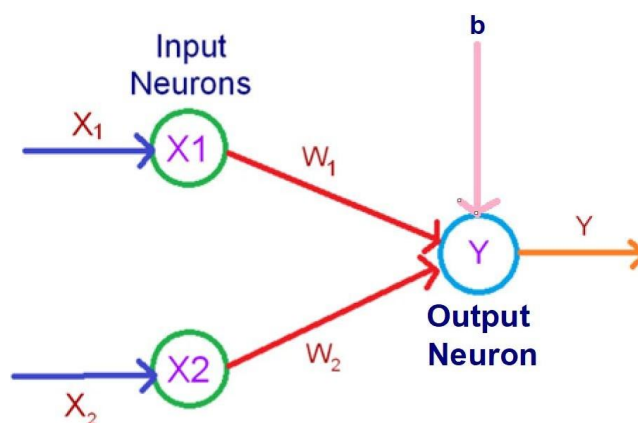
Part 1: Using Binary sigmoidal function

The function takes any real value as input and outputs values in the range 0 to 1. The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to 0.0.

The Sigmoid Function curve looks like a S-shape.



Consider the following Neural Network



1b(1) Code : Calculate the output of neural net using binary sigmoidal function.

```
import numpy as np

print("Enter Value of X1 =");
x1 = int(input());

print("Enter Value of X2 =");
x2 = int(input());

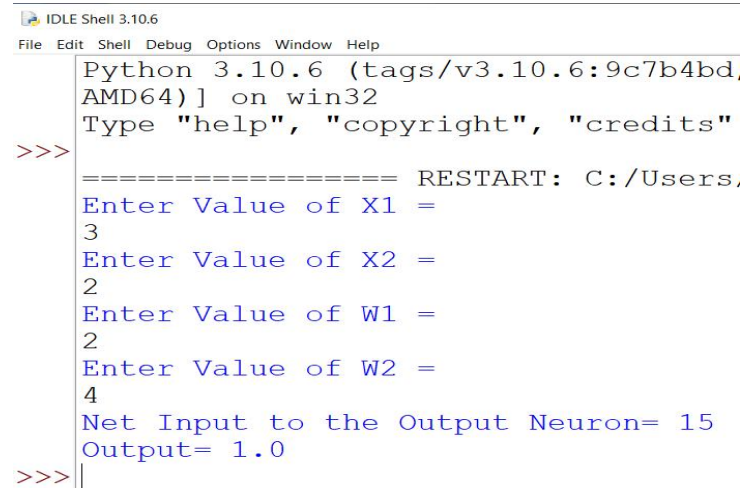
print("Enter Value of W1 =");
w1 = int(input());

print("Enter Value of W2 =");
w2 = int(input());
b = 1

yin = b + (x1*w1 + x2*w2);
print("Net Input to the Output Neuron=", yin);

output = 1/(1 + np.exp(-yin));
print("Output=", round(output, 4));
```

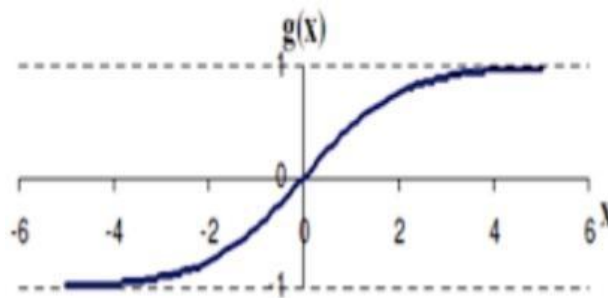
Output :



```
IDLE Shell 3.10.6
File Edit Shell Debug Options Window Help
Python 3.10.6 (tags/v3.10.6:9c7b4bd,
AMD64) on win32
Type "help", "copyright", "credits"
>>>
==== RESTART: C:/Users,
Enter Value of X1 =
3
Enter Value of X2 =
2
Enter Value of W1 =
2
Enter Value of W2 =
4
Net Input to the Output Neuron= 15
Output= 1.0
>>>|
```

Part 2: Using Bipolar sigmoidal function

The Bipolar sigmoidal function is represented in the following way



Mathematically it is given by

$$y = f(y_{in}) = \frac{2}{1 + e^{-y_{in}}} - 1$$

1b(2). Code : Calculate the output of neural net using bipolar sigmoidal function.

```
import numpy as np
print("Enter Value of X1 =");
x1 = int(input());

print("Enter Value of X2 =");
x2 = int(input());

print("Enter Value of W1 =");
w1 = int(input());

print("Enter Value of W2 =");
w2 = int(input());

b = 1;
yin = b + (x1*w1 + x2*w2);

print("Net Input to the Output Neuron=", yin);

output = -1 + (2/(1 + np.exp(-yin)))
print("Output=", round(output, 4));
```


Output :

```
IDLE Shell 3.10.6
File Edit Shell Debug Options Window Help
Python 3.10.6 (tags/v3.10.6:9c7b4bd,
AMD64) on win32
Type "help", "copyright", "credits"
>>>
===== RESTART: C:/Users/
Enter Value of X1 =
3
Enter Value of X2 =
4
Enter Value of W1 =
2
Enter Value of W2 =
6
Net Input to the Output Neuron= 31
Output= 1.0
>>> |
```

Practical : 02

Generate AND/NOT function using McCulloch-Pitts neural net.

The McCulloch–Pitt neural network is considered to be the first neural network. The neurons are connected by directed weighted paths. McCulloch–Pitt neuron allows binary activation (ON or OFF), i.e., it either fires with an activation 1 or does not fire with an activation of 0. If $w > 0$, then the connected path is said to be excitatory. If $w < 0$, then the connected path is said to be inhibitory. Excitatory connections have positive weights and inhibitory connections have negative weights. Each neuron has a fixed threshold for firing. That is, if the net input to the neuron is greater than the threshold, it fires.

In the present case we implement AND-NOT function using McCulloch-Pitts Neural Network.

Binary inputs are taken, and the weights are initialized to 0.

We have the following training sets and the target to implement the AND-NOT function

Training Sets		Target
X1	X2	t
0	0	0
0	1	0
1	0	1
1	1	0

Implement the following:

2a. Code : **Generate AND/NOT function using McCulloch-Pitts neural network**

```
w1 = 0
w2 = 0
print("For the ", 4, "inputs calculate the net input using  $y_{in} = x1.w1 + x2.w2$ ")

x1 = [0,0,1,1]
x2 = [0,1,0,1]
y = [0,0,1,0]
b = 0

print("Training sets\n")
print("x1",x1)
print("x2",x2)
print("Target (y): ", y)

for i in range(0,4):
    w1 = w1 + x1[i]*y[i]
```

```
w2 = w2 + x2[i]*y[i]
b = b+y[i]
```

```
print("The updated weights are :")
print("w1_new: ", w1)
print("w2_new: ", w2)
print("b_new: ", b)
```

```
print("The Final Weights are :")
print("w1_new: ", w1)
print("w2_new: ", w2)
print("b_new: ", b)
```

Output :

```
IDLE Shell 3.10.6
File Edit Shell Debug Options Window Help
Python 3.10.6 (tags/v3.10.6:9c7b4bd, Aug 1 2022, 21:53:49) [MSC v.1932 6
AMD64] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/AZM/Desktop/MSIT/sc_2a.py =====
For the 4 inputs calculate the net input using y_in = x1.w1 + x2.w2
Training sets
x1 [0, 0, 1, 1]
x2 [0, 1, 0, 1]
Target (y): [0, 0, 1, 0]
The updated weights are:
w1_new: 0
w2_new: 0
b_new: 0
The updated weights are:
w1_new: 0
w2_new: 0
b_new: 0
The updated weights are:
w1_new: 1
w2_new: 0
b_new: 1
The updated weights are:
w1_new: 1
w2_new: 0
b_new: 1
The Final Weights are :
w1_new: 1
w2_new: 0
b_new: 1
>>>
```

Practical 02:B

Generate XOR function using McCulloch-Pitts neural net.

We need to implement XOR function using McCulloch-Pitts Neural Network.

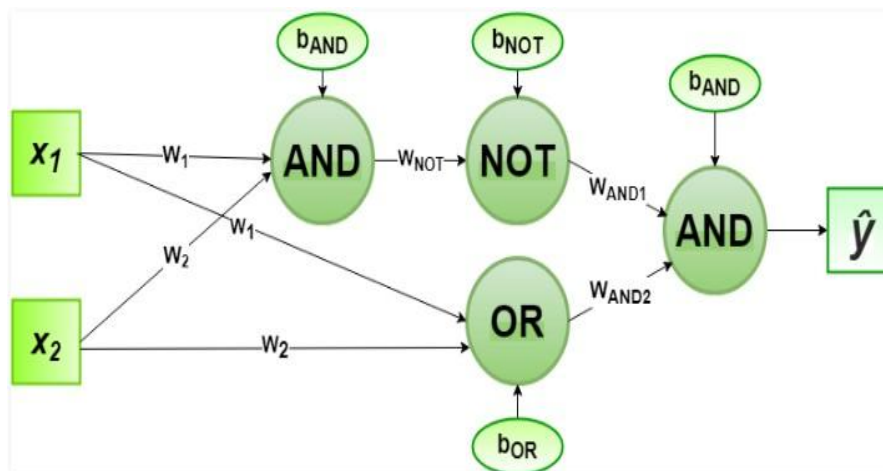
Binary inputs are taken, and the weights are initialized to suitable values.

We have the following training sets and the target to implement the EX-OR function

Training Sets		Target
X1	X2	t
0	0	0
0	1	1
1	0	1
1	1	0

As XOR function is not variable separable, we need some different strategy to implement this function.

Consider the diagram below



The weights are initialized as

$W_1 = 1, W_2 = 1,$

$W_{AND1} = 1, W_{AND2} = 1,$

$W_{NOT} = -1,$

$b_{AND} = -1.5, b_{OR} = -0.5,$

$b_{NOT} = 0.5$

2b. Code : **Generate XOR function using McCulloch-Pitts neural net.**

```
import numpy as np
def unit_step(v):
    if v >= 0:
        return 1
    else:
        return 0

def perceptron_model(x,w,b):
    v = np.dot(w,x)+b
    y = unit_step(v)
    return y

def NOT_logic(x):
    w_NOT = -1
    b_NOT = 0.5
    return perceptron_model(x,w_NOT,b_NOT)

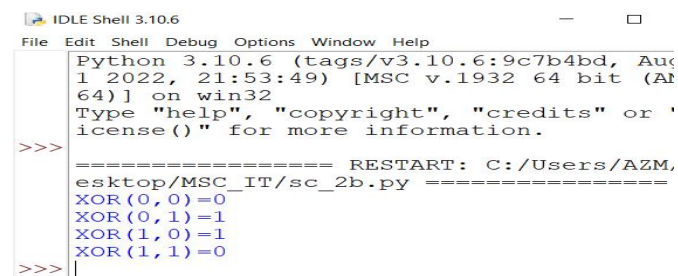
def AND_logic(x):
    w = np.array([1,1])
    b_AND = -1.5
    return perceptron_model(x,w,b_AND)

def OR_logic(x):
    w = np.array([1,1])
    b_OR = -0.5
    return perceptron_model(x,w,b_OR)

def XOR_logic(x):
    y1 = AND_logic(x)
    y2 = OR_logic(x)
    y3 = NOT_logic(y1)
    final_x = np.array([y2,y3])
    final_output = AND_logic(final_x)
    return final_output

test1 = np.array([0,0])
test2 = np.array([0,1])
test3 = np.array([1,0])
test4 = np.array([1,1])

print("XOR({},{})={}".format(0,0,XOR_logic(test1)))
print("XOR({},{})={}".format(0,1,XOR_logic(test2)))
print("XOR({},{})={}".format(1,0,XOR_logic(test3)))
print("XOR({},{})={}".format(1,1,XOR_logic(test4)))
```



```
IDLE Shell 3.10.6
File Edit Shell Debug Options Window Help
Python 3.10.6 (tags/v3.10.6:9c7b4bd, Aug 1 2022, 21:53:49) [MSC v.1932 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/AZM,
esktop/MSB_IT/sc_2b.py =====
XOR(0,0)=0
XOR(0,1)=1
XOR(1,0)=1
XOR(1,1)=0
>>>
```

Practical : 03

Write a program to implement Hebb's rule

Hebbian Learning Rule, also known as Hebb Learning Rule, was proposed by Donald O Hebb. It is one of the first and also easiest learning rules in the neural network. It is used for pattern classification. 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 Hebbian Rule works well as long as all the input patterns are orthogonal or uncorrelated. The requirement of orthogonality places serious limitations on the Hebbian Learning Rule.

For the present case we consider the example of OR function, using Bipolar input we have the following relations between the training sets and the target

Training Sets		Target
X1	X2	t
-1	-1	-1
-1	1	1
1	-1	1
1	1	1

Implement the Following

3a. Code : Write a program to implement Hebb's rule.

```
w1 = 0
w2 = 0

x1 = [-1,-1,1,1]
x2 = [-1,1,-1,1]
y = [-1,1,1,1]
b = 0
print("Training Sets");
print("x1 = ",x1)
print("x2 = ",x2)
print("Target(y) :",y);

for i in range(0, 4):
    w1 = w1 + x1[i] * y[i];
    w2 = w2 + x2[i] * y[i];
    b = b + y[i];
    print("The Weights in epoch ",i+1);
    print("W1new=",w1);
```

```

    print("w2new=",w2);
    print("bnew=",b);

print("The Final Weights are :")
print("W1new=",w1);
print("w2new=",w2);
print("bnew=",b);

num_ip = int(input("Enter the number of inputs : "))
print("For the ", num_ip , " inputs calculate the net input using  $y_{in} = x_1w_1 + x_2w_2$  ")

x1 = []
x2 = []
for j in range(0, num_ip):
    ele1 = int(input("x1 = "))
    ele2 = int(input("x2 = "))
    x1.append(ele1)
    x2.append(ele2)
w1 = 0
w2 = 0
y = [-1,-1,1,-1]
b = 0

print("Training Sets");
print("x1 = ",x1)
print("x2 = ",x2)
print("Target(y) : ",y);

for i in range(0, 4):
    w1 = w1 + x1[i] * y[i];
    w2 = w2 + x2[i] * y[i];
    b = b + y[i];
    print("The Weights in epoch ",i+1);
    print("W1new=",w1);
    print("w2new=",w2);
    print("bnew=",b);

print("The Final Weights are :")
print("W1new=",w1);
print("w2new=",w2);
print("bnew=",b);

```

Output :

```
*IDLE Shell 3.10.6*
File Edit Shell Debug Options Window Help
Python 3.10.6 (tags/v3.10.6:9c7b4bd, Aug
AMD64)] on win32
Type "help", "copyright", "credits" or ">>>
===== RESTART: C:/Users/AZM/Desktop/
Training Sets
x1 = [-1, -1, 1, 1]
x2 = [-1, 1, -1, 1]
Target(y) : [-1, 1, 1, 1]
The Weights in epoch 1
W1new= 1
w2new= 1
bnew= -1
The Weights in epoch 2
W1new= 0
w2new= 2
bnew= 0
The Weights in epoch 3
W1new= 1
w2new= 1
bnew= 1
The Weights in epoch 4
W1new= 2
w2new= 2
bnew= 2
The Final Weights are :
W1new= 2
w2new= 2
bnew= 2
Enter the number of inputs : |

Enter the number of inputs : 4
For the 4 inputs calculate the net input using yin = x1w1 + x2w2
x1 = 1
x2 = 3
x1 = 4
x2 = 6
x1 = 3
x2 = 4
x1 = 3
x2 = 1
Training Sets
x1 = [1, 4, 3, 3]
x2 = [3, 6, 4, 1]
Target(y) : [-1, -1, 1, -1]
The Weights in epoch 1
W1new= -1
w2new= -3
bnew= -1
The Weights in epoch 2
W1new= -5
w2new= -9
bnew= -2
The Weights in epoch 3
W1new= -2
w2new= -5
bnew= -1
The Weights in epoch 4
W1new= -5
w2new= -6
bnew= -2
The Final Weights are :
W1new= -5
w2new= -6
bnew= -2
>
```


Practical : 03-B

Write a program to implement of delta rule.

The Delta rule in machine learning and neural network environments is a specific type of backpropagation that helps to refine connectionist ML/AI networks, making connections between inputs and outputs with layers of artificial neurons.

The Delta rule is also known as the Delta learning rule.

In general, backpropagation has to do with recalculating input weights for artificial neurons using a gradient method. Delta learning does this using the difference between a target activation and an actual obtained activation. Using a linear activation function, network connections are adjusted.

Another way to explain the Delta rule is that it uses an error function to perform gradient descent learning.

A tutorial on the Delta rule explains that essentially in comparing an actual output with a targeted output, the technology tries to find a match. If there is not a match, the program makes changes. The actual implementation of the Delta rule is going to vary according to the network and its composition, but by employing a linear activation function, the Delta rule can be useful in refining some types of neural network systems with particular flavour of backpropagation.

For the present case we take the example of OR function using Bipolar inputs

Training Sets		Target
X1	X2	t
-1	-1	-1
-1	1	1
1	-1	1
1	1	1

3b. Code : **Write a program to implement of Delta rule.**

```
print("For the ", 4, " inputs calculate the net input using  $y_{in} = x_1w_1 + x_2w_2$  ")
print("The bipolar inputs (training sets)are :")
x1 = [1,1,-1,-1]
x2 = [1,-1,1,-1]
```

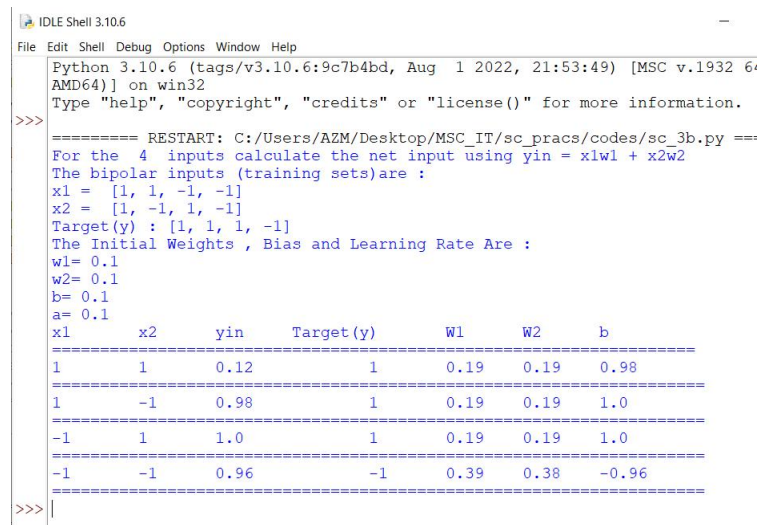
```

y = [1,1,1,-1]
print("x1 = ", x1)
print("x2 = ", x2)
print("Target(y) :", y)
print("The Initial Weights , Bias and Learning Rate Are :")
w1 = 0.1
w2 = 0.1
b = 0.1
a = 0.1
print("w1=", w1)
print("w2=", w2)
print("b=", b)
print("a=", a)
print("x1 \t x2 \t yin \t Target(y) \t W1 \t W2 \t b ");
print("=====")

for i in range(0, 4):
    yin = b + ((x1[i]*w1) + (x2[i]*w2))*a;
    dw1 = a * (y[i] - yin)*x1[i];
    dw2 = a * (y[i] - yin)*x2[i];
    db = (y[i] - yin);
    w1 = w1 + dw1;
    w2 = w2 + dw2
    b = b + db
    print(x1[i],"\t",x2[i],"\t",round(yin,2),"\t\t",round(y[i],2),"\t",round(w1,2),"\t",round(w2,2),"\t",round(b,2));
    print("=====")

```

Output :



```

IDLE Shell 3.10.6
File Edit Shell Debug Options Window Help
Python 3.10.6 (tags/v3.10.6:9c7b4bd, Aug 1 2022, 21:53:49) [MSC v.1932 64-bit AMD64] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> ===== RESTART: C:/Users/AZM/Desktop/MSC_IT/sc_pracs/codes/sc_3b.py ===
For the 4 inputs calculate the net input using yin = x1w1 + x2w2
The bipolar inputs (training sets)are :
x1 = [1, 1, -1, -1]
x2 = [1, -1, 1, -1]
Target(y) : [1, 1, 1, -1]
The Initial Weights , Bias and Learning Rate Are :
w1= 0.1
w2= 0.1
b= 0.1
a= 0.1
x1      x2      yin      Target(y)      W1      W2      b
=====
1        1        0.12      1              0.19    0.19    0.98
=====
1        -1        0.98      1              0.19    0.19    1.0
=====
-1        1        1.0       1              0.19    0.19    1.0
=====
-1        -1        0.96     -1              0.39    0.38   -0.96
=====
>>>

```

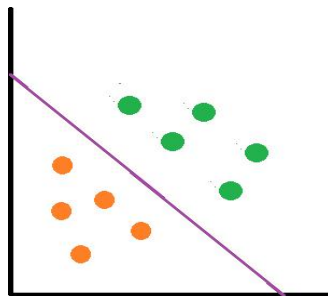
Practical : 04

Write a program for Linear separation

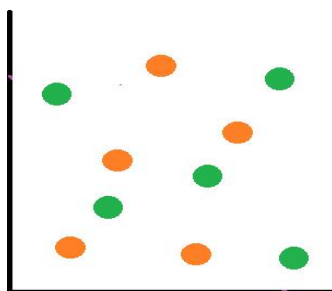
Linear separable means that there is a hyperplane, which splits the input data into two half-spaces such that all points of the first class should be in one half-space and other points of the second class should be in the other half-space.

In 2-D, it means that there is a line, which separates points of one class from points of the other class.

For example: In the following image, if orange circles represent points from one class and green circles represent points from the other class, then these points are linearly separable.



In the following image, the two classes represented by orange and green circles cannot be separated by a line



Linear separability is an important concept in neural networks. In ANN a decision line is drawn to separate positive and negative responses. The decision line may also be called as the decision-making Line or decision-support Line or linear-separable line. The necessity of the linear separability

concept was felt to clarify classify the patterns based upon their output responses

The Logic functions AND, OR etc are linearly separable, while XOR and EX-NOR are not linearly separable.

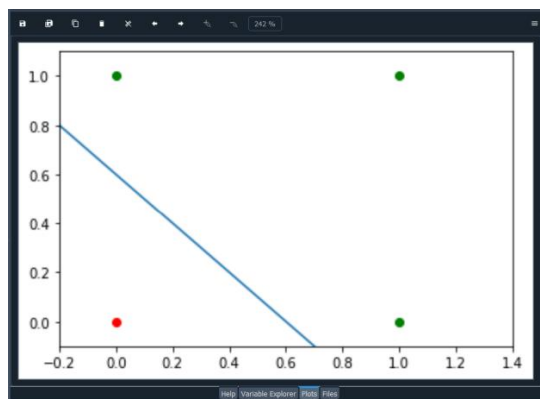
Practical 4 : **Write a program for Linear separation.**

Code :

```
import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots()
xmin, xmax = -0.2, 1.4
X = np.arange(xmin, xmax, 0.1)
ax.scatter(0, 0, color="r")
ax.scatter(0, 1, color="g")
ax.scatter(1, 0, color="g")
ax.scatter(1, 1, color="g")
ax.set_xlim([xmin, xmax])
ax.set_ylim([-0.1, 1.1])
m = -1
ax.plot(X, m * X + 0.6, label="decision boundary")
plt.show()
#plt.plot()
```

Output :



Practical : 05

Membership and Identity Operators is, is not

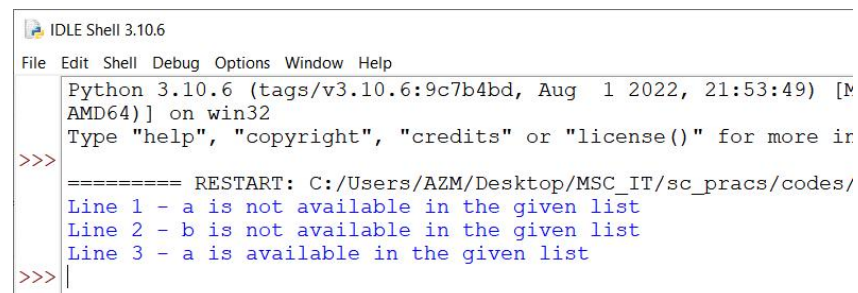
Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

Code : **Membership and Identity Operators | in, not in,**

```
a = 10
b = 20
list = [1, 2, 3, 4, 5];
if a in list:
    print ("Line 1 - a is available in the given list")
else:
    print ("Line 1 - a is not available in the given list")
if b not in list :
    print ("Line 2 - b is not available in the given list")
else:
    print ("Line 2 - b is available in the given list")

a = 2
if a in list :
    print ("Line 3 - a is available in the given list")
else:
    print ("Line 3 - a is not available in the given list")
```

Output :



```
IDLE Shell 3.10.6
File Edit Shell Debug Options Window Help
Python 3.10.6 (tags/v3.10.6:9c7b4bd, Aug 1 2022, 21:53:49) [N
AMD64] on win32
Type "help", "copyright", "credits" or "license()" for more ir
>>>
===== RESTART: C:/Users/AZM/Desktop/MSIT/sc_pracs/codes/
Line 1 - a is not available in the given list
Line 2 - b is not available in the given list
Line 3 - a is available in the given list
>>> |
```

Practical : 06

Find ratios using fuzzy logic

Ratio

We used the ratio function above to calculate the Levenshtein distance similarity ratio between the two strings (sequences).

Partial Ratio

FuzzyWuzzy also has more powerful functions to help with matching strings in more complex situations. This function allows us to perform substring matching. This works by taking the shortest string and matching it with all substrings that are of the same length.

Token Sort Ratio

FuzzyWuzzy also has token functions that tokenize the strings, change capitals to lowercase, and remove punctuation. This function sorts the strings alphabetically and then joins them together. Then, the `fuzz.ratio()` is calculated. This can come in handy when the strings we are comparing are the same in spelling but are not in the same order.

Token Set Ratio

This function is similar to the token sort ratio, except it takes out the common tokens before calculating the fuzz ration between the new strings. This function is the most helpful when applied to a set of strings with a significant difference in lengths.

Implement the Following

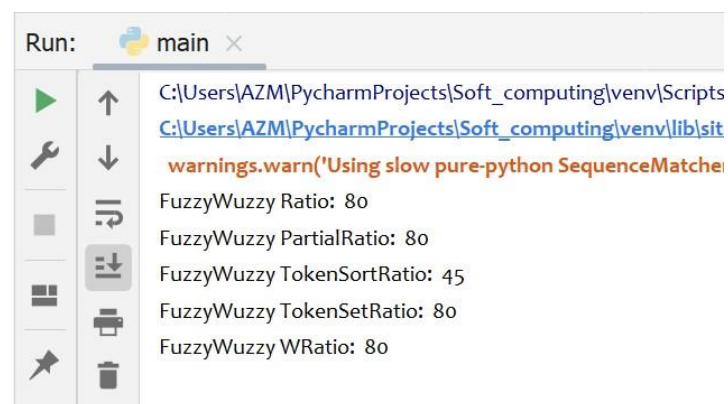
Note : pip install fuzzywuzzy

Code :

```
import fuzzywuzzy
from fuzzywuzzy import process

s1 = "I like softcomputing"
s2 = "I like hardcomputing"
print ("FuzzyWuzzy Ratio: ", fuzzywuzzy.fuzz.ratio(s1, s2))
print ("FuzzyWuzzy PartialRatio: ", fuzzywuzzy.fuzz.partial_ratio(s1, s2))
print ("FuzzyWuzzy TokenSortRatio: ", fuzzywuzzy.fuzz.token_sort_ratio(s1, s2))
print ("FuzzyWuzzy TokenSetRatio: ", fuzzywuzzy.fuzz.token_set_ratio(s1, s2))
print ("FuzzyWuzzy WRatio: ", fuzzywuzzy.fuzz.WRatio(s1, s2), '\n\n')
```

Output :

The screenshot shows a 'Run' window from a Python IDE. The title bar says 'Run: main'. On the left is a vertical toolbar with icons for running, stepping through, and other debugging actions. The main area displays the execution output. It starts with a warning message: 'warnings.warn("Using slow pure-python SequenceMatcher')'. Below this, the script's output is printed: 'FuzzyWuzzy Ratio: 80', 'FuzzyWuzzy PartialRatio: 80', 'FuzzyWuzzy TokenSortRatio: 45', 'FuzzyWuzzy TokenSetRatio: 80', and 'FuzzyWuzzy WRatio: 80'.

```
Run: main ×
C:\Users\AZM\PycharmProjects\Soft_computing\venv\Scripts
C:\Users\AZM\PycharmProjects\Soft_computing\venv\lib\sit
warnings.warn("Using slow pure-python SequenceMatcher
FuzzyWuzzy Ratio: 80
FuzzyWuzzy PartialRatio: 80
FuzzyWuzzy TokenSortRatio: 45
FuzzyWuzzy TokenSetRatio: 80
FuzzyWuzzy WRatio: 80
```

Practical : 06-B

The Tipping Problem

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

The Tipping Problem

We create a fuzzy control system which models how you might choose to tip at a restaurant. When tipping, you consider the service and food quality, rated between 0 and 10. You use this to leave a tip of between 0 and 25%.

We would formulate this problem as:

- 1) Antecedents (Inputs)
 - a) Service Universe (ie, crisp value range): How good was the service of the wait staff, on a scale of 0 to 10? : Fuzzy set (ie, fuzzy value range): poor, acceptable, amazing
 - b) Food quality Universe: How tasty was the food, on a scale of 0 to 10? Fuzzy set: bad, decent, great
- 2) Consequents (Outputs)
 - a) Tip Universe: How much should we tip, on a scale of 0% to 25% Fuzzy set: low, medium, high

Rules

- 1) IF the *service* was good *or* the *food quality* was good, THEN the tip will be high.
- 2) IF the *service* was average, THEN the tip will be medium.
- 3) IF the *service* was poor *and* the *food quality* was poor THEN the tip will be low.

Usage

- 1) If I tell this controller that I rated:
the service as 9.8, and
the quality as 6.5,
it would recommend I leave : a 20.2% tip.

Note : pip install scikit-fuzzy

Code :

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import matplotlib as plt
```

```
# New Antecedent/Consequent objects hold universe variables and membership
```

```
# functions
```

```
quality = ctrl.Antecedent(np.arange(0, 11, 1), 'quality')
```

```
service = ctrl.Antecedent(np.arange(0, 11, 1), 'service')
```



```

tip = ctrl.Consequent(np.arange(0, 26, 1), 'tip')

# Auto-membership function population is possible with .automf(3, 5, or 7)
quality.automf(3)
service.automf(3)

# Custom membership functions can be built interactively with a familiar,
# Pythonic API
tip['low'] = fuzz.trimf(tip.universe, [0, 0, 13])
tip['medium'] = fuzz.trimf(tip.universe, [0, 13, 25])
tip['high'] = fuzz.trimf(tip.universe, [13, 25, 25])
quality['average'].view()
service.view()

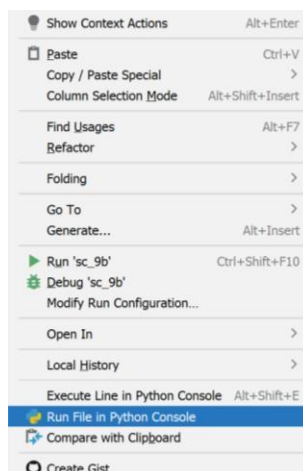
tip.view()
rule1 = ctrl.Rule(quality['poor'] | service['poor'], tip['low'])
rule2 = ctrl.Rule(service['average'], tip['medium'])
rule3 = ctrl.Rule(service['good'] | quality['good'], tip['high'])
rule1.view()

tipping_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
tipping = ctrl.ControlSystemSimulation(tipping_ctrl)

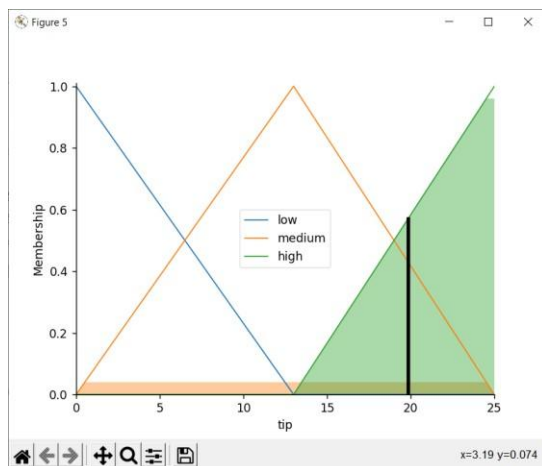
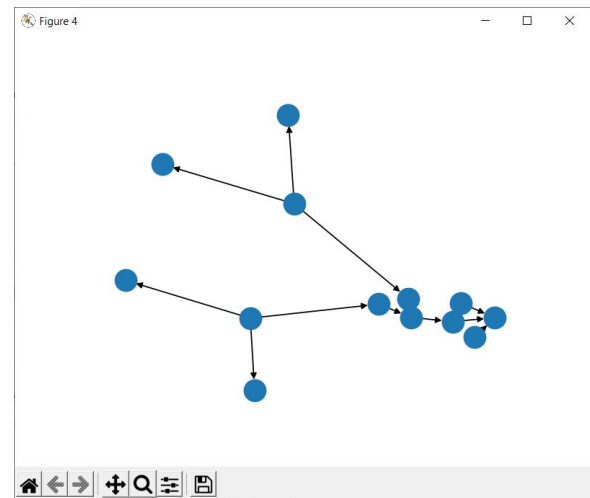
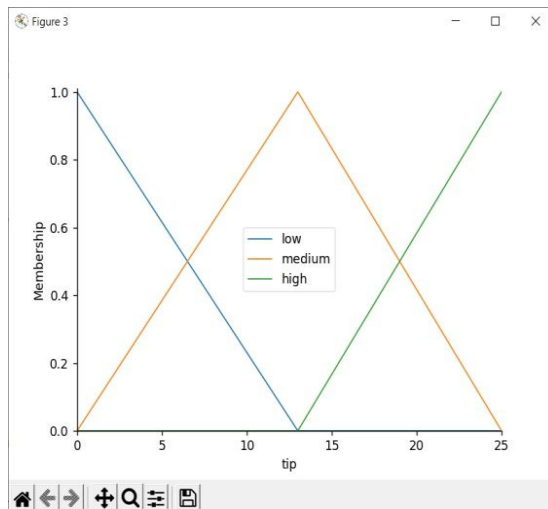
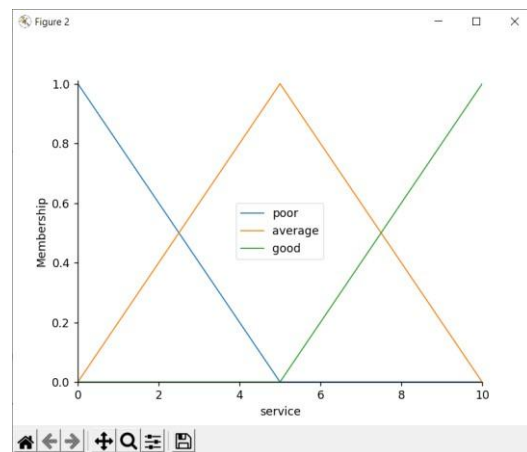
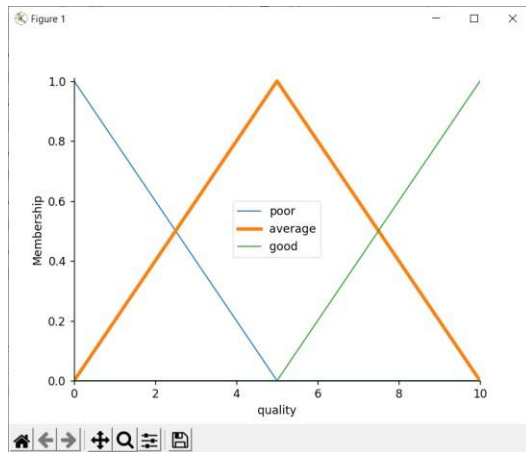
# Pass inputs to the ControlSystem using Antecedent labels with Pythonic API
# Note: if you like passing many inputs all at once, use .inputs(dict_of_data)
tipping.input['quality'] = 6.5
tipping.input['service'] = 9.8
# Crunch the numbers
tipping.compute()
print(tipping.output['tip'])
print("Executed Successfully!!!")

```

Note : Execute file in Pycharm by > right click on code > Run in Python Console> ok



Output :



Practical : 07

Implementation of Simple genetic algorithm

Genetic Algorithms (GAs) are search based algorithms based on the concepts of natural selection and genetics. GAs are a subset of a much larger branch of computation known as Evolutionary Computation. Genetic Algorithm (GA) is a search-based optimization technique based on the principles of Genetics and Natural Selection. It is frequently used to find optimal or near-optimal solutions to difficult problems which otherwise would take a lifetime to solve. It is frequently used to solve optimization problems, in research, and in machine learning. GAs were developed by John Holland and his students and colleagues.

In GAs, we have a pool or a population of possible solutions to the given problem. These solutions then undergo recombination and mutation (like in natural genetics), producing new children, and the process is repeated over various generations. Each individual (or candidate solution) is assigned a fitness value (based on its objective function value) and the fitter individuals are given a higher chance to mate and yield more “fitter” individuals. This is in line with the Darwinian Theory of “**Survival of the Fittest**”.

Code :

```
import random

# Number of individuals in each generation
POPULATION_SIZE = 100

# Valid genes
GENES = ""abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890, .-:;_!"#%&/()=?@${}""

# Target string to be generated
TARGET = "My name is smile"

class Individual(object):

    def __init__(self, chromosome):
        self.chromosome = chromosome
        self.fitness = self.cal_fitness()

    @classmethod
    def mutated_genes(self):

        global GENES
        gene = random.choice(GENES)
        return gene

    @classmethod
```

```

def create_gnome(self):

    global TARGET
    gnome_len = len(TARGET)
    return [self.mutated_genes() for _ in range(gnome_len)]

def mate(self, par2):

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

        prob = random.random()

        if prob < 0.45:
            child_chromosome.append(gp1)

        elif prob < 0.90:
            child_chromosome.append(gp2)

        else:
            child_chromosome.append(self.mutated_genes())

    return Individual(child_chromosome)

def cal_fitness(self):

    global TARGET
    fitness = 0
    for gs, gt in zip(self.chromosome, TARGET):
        if gs != gt: fitness += 1
    return fitness

# Driver code
def main():
    global POPULATION_SIZE

    generation = 1

    found = False
    population = []

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

    while not found:

        population = sorted(population, key=lambda x: x.fitness)

        if population[0].fitness <= 0:
            found = True

```

```

        break

    new_generation = []

    s = int((10 * POPULATION_SIZE) / 100)
    new_generation.extend(population[:s])

    s = int((90 * POPULATION_SIZE) / 100)
    for _ in range(s):
        parent1 = random.choice(population[:50])
        parent2 = random.choice(population[:50])
        child = parent1.mate(parent2)
        new_generation.append(child)

    population = new_generation

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

    generation += 1

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

if __name__ == '__main__':
    main()

```

Output :

```
IDLE Shell 3.10.6
File Edit Shell Debug Options Window Help
Python 3.10.6 (tags/v3.10.6:9c7b4bd, Aug 1 2022, 21:53:49) [MSC v.1932 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/AZM/Desktop/MSC_IT/sc_pracs/codes/sc_10a.py =====
Generation: 1 String: Id 00at))wt
auke Fitness: 14
Generation: 2 String: I; w8at(?kz
m6Ve Fitness: 13
Generation: 3 String: I; w8at(?kz
m6Ve Fitness: 13
Generation: 4 String: I; w8at(?kz
m6Ve Fitness: 13
Generation: 5 String: M; HJj#Gdw3sRi(u Fitness: 12
Generation: 6 String: M( wajp)oh/
muVe Fitness: 11
Generation: 7 String: Md =ajr)2htQmiUe Fitness: 10
Generation: 8 String: M; Ha=RjdhtsmiUe Fitness: 9
Generation: 9 String: M; Ha=RjdhtsmiUe Fitness: 9
Generation: 10 String: M; Ha=RjdhtsmiUe Fitness: 9
Generation: 11 String: M; Ha=RjdhtsmiUe Fitness: 9
Generation: 12 String: M( namBjdhtsmiAe Fitness: 7
Generation: 13 String: M( namBjdhtsmiAe Fitness: 7
Generation: 14 String: M( namBjdhtsmiAe Fitness: 7
Generation: 15 String: M; namdo9szsmi[e Fitness: 6
Generation: 16 String: M; namdo9szsmi[e Fitness: 6
Generation: 17 String: M; namdo9szsmi[e Fitness: 6
Generation: 18 String: M; namdo9szsmi[e Fitness: 6
Generation: 19 String: M; namdo9szsmi[e Fitness: 6
Generation: 20 String: M; namdo9szsmi[e Fitness: 6
Generation: 21 String: M; namV-%sjsmile Fitness: 5
Generation: 22 String: M; namV-%sjsmile Fitness: 5
Generation: 23 String: M; namV-%sjsmile Fitness: 5
Generation: 24 String: M; namV-%sjsmile Fitness: 5
Generation: 25 String: M) nam2 9stsmile Fitness: 4
Generation: 26 String: M) nam2 9stsmile Fitness: 4
Generation: 27 String: M) nam2 9stsmile Fitness: 4
Generation: 28 String: M) nam2 9stsmile Fitness: 4
Generation: 29 String: M) nam2 9stsmile Fitness: 4
Generation: 30 String: M) nam2 9stsmile Fitness: 4
Ln: 82 Col: 0
```