THADOMAL SHAHANI ENGINEERING COLLEGE, BANDRA(W)
Computer Engineering Department

INDEX

EXPERIMENT 1A: Implementing Fuzzy Functions

Theory:

The membership function $\mu_A(x)$ of a fuzzy set A is a function $\mu_A(x) : X \longrightarrow [0,1]$ So, every element in x in X has a membership degree: $\mu_A(x) \in [0,1]$. A is completely determined by the set of tuples: $A = \{(x, \mu_A(x)), x \in X\}$

Operations on Fuzzy Sets:

Let $A(x)$ and $B(x)$ be two fuzzy sets, for $\forall x \in X$.
Definition has the form

$$Denotation = \{(x, membership\ \ function), x \in X\}$$

$A(x)$ = [(1, 0.87), (2, 0.0), (3, 0.49), (4, 1.0), (5, 0.85)]
$B(x)$ = [(1, 1.0), (2, 0.45), (3, 0.3), (4, 0.0), (5, 0.65)]

1. Union
   The union operation when applied on two fuzzy set (within the same universe of discourse) yields another fuzzy set with the membership values equal to the max amongst the input sets.

   Definition:
   $$(A \cup B)(x) = \{(x, max\{\mu_A(x), \mu_B(x)\}), x \in X\}$$
   Example: $(A \cup B)(x)$ = [(1, 1.0), (2, 0.45), (3, 0.49), (4, 1.0), (5, 0.85)]

2. Intersection
   The intersection operation when applied on two fuzzy sets (within the same universe of discourse) yields another fuzzy set with the membership values equal to them in amongst the input sets.

   Definition:
   $$(A \cap B)(x) = \{(x, min\{\mu_A(x), \mu_B(x)\}), x \in X\}$$
   Example: $(A \cap B)(x)$ = [(1, 0.87), (2, 0.0), (3, 0.3), (4, 0.0), (5, 0.65)]

3. Complement

The complement operation yields a fuzzy set with the membership values equal to one minus the original membership values.

Definition:

$$A^C(x) = \{(x, 1 - \mu_A(x)), x \in X\}$$

Example: $A^C(x)$ = [(1, 0.13), (2, 1.0), (3, 0.51), (4, 0.0), (5, 0.15)]

4. Set Difference

The set difference operation when applied on two fuzzy sets (within the same universe of discourse) yields another fuzzy set wherein the following operations are applied:

Definition:

$$(A - B)(x) = \{(x, min\{\mu_A(x), 1 - \mu_B(x)\}), x \in X\}$$

Example: $(A - B)(x)$ = [(1, 0.0), (2, 0.0), (3, 0.49), (4, 1.0), (5, 0.35)]

5. Algebraic Sum

Definition:

$$A(x) + B(x) = \{(x, \mu_A(x) + \mu_B(x) - \mu_A(x) \cdot \mu_B(x)\}), x \in X\}$$

Example: $A(x) + B(x)$ = [(1, 1.0), (2, 0.45), (3, 0.64), (4, 1.0), (5, 0.95)]

6. Algebraic Product

Definition:

$$A(x) \cdot B(x) = \{(x, \mu_A(x) \cdot \mu_B(x)\}), x \in X\}$$

Example: $A(x) \cdot B(x)$ = [(1, 0.87), (2, 0.0), (3, 0.15), (4, 0.0), (5, 0.55)]

7. Bounded Sum

Definition:

$$A(x) \oplus B(x) = \{(x, min\{1, \mu_A(x) + \mu_B(x)\}), x \in X\}$$

Example: $A(x) \oplus B(x)$ = [(1, 1), (2, 0.45), (3, 0.79), (4, 1), (5, 1)]

8. Bounded Difference

Definition:

$$A(x) \ominus B(x) = \{(x, max\{0, \mu_A(x) + \mu_B(x) - 1\}), x \in X\}$$

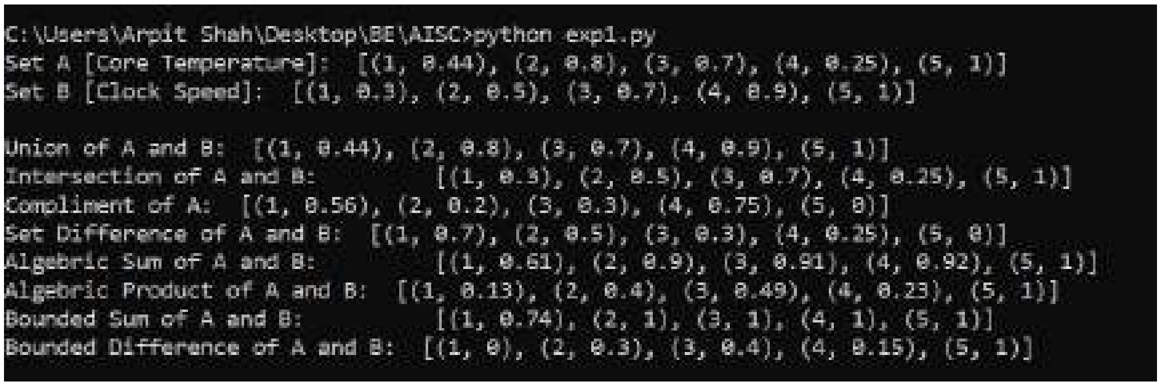
Example: $A(x) \ominus B(x)$ = [(1, 0.87), (2, 0), (3, 0), (4, 0),
(5, 0.5)]

Program:

```
A = {1:0.44, 2:0.8, 3:0.7, 4:0.25, 5:1}

B = {1:0.3, 2:0.5, 3:0.7, 4:0.9, 5:1}


U = dict()

I = dict()

C = dict()

SD = dict()

AS = dict()

AP = dict()

BS = dict()

BD = dict()



for A_key, B_key in zip(A, B):

  A_value = A[A_key]

  B_value = B[B_key] # Union

  U[A_key] = max(A_value,B_value) # Intersection

  I[A_key] = min(A_value,B_value) # Compliment

  C[A_key]= round(1-A[A_key],2) # Set Difference

  B_comp = 1 - B_value

  if A_value < B_value:

    SD[A_key] = round(A_value,2)

  else:

    SD[B_key] = round(B_comp,2)
```

```
 # Algebric Sum

 AS[A_key] = round(A_value + B_value - (A_value*B_value),2) # Algebric Product


 AP[A_key] = round(A_value*B_value,2) #Bounded Sum

 BS[A_key] = round(min(1,A_value+B_value),2) #Bounded Difference

 BD[A_key] = round(max(0,(A_value+B_value)-1),2)


print("Set A [Core Temperature]: ", end= " ")

print([(k, v) for k,v in A.items()])

print("Set B [Clock Speed]: ", end= " ")

print([(k, v) for k,v in B.items()])

print()

print("Union of A and B: ", end= " ")

print([(k, v) for k,v in U.items()])

print("Intersection of A and B:    ", end= " ")

print([(k, v) for k,v in I.items()])

print("Compliment of A: ", end= " ")

print([(k, v) for k,v in C.items()])

print("Set Difference of A and B: ", end= " ")

print([(k, v) for k,v in SD.items()])

print("Algebric Sum of A and B:  ", end= " ")

print([(k, v) for k,v in AS.items()])

print("Algebric Product of A and B: ", end= " ")

print([(k, v) for k,v in AP.items()])

print("Bounded Sum of A and B:              ", end= " ")

print([(k, v) for k,v in BS.items()])
print("Bounded Difference of A and B: ", end= " ")

print([(k, v) for k,v in BD.items()])
```

Output:



```
C:\Users\Arpit Shah\Desktop\BE\AISC>python exp1.py
Set A [Core Temperature]:  [(1, 0.44), (2, 0.8), (3, 0.7), (4, 0.25), (5, 1)]
Set B [Clock Speed]:  [(1, 0.3), (2, 0.5), (3, 0.7), (4, 0.9), (5, 1)]

Union of A and B:  [(1, 0.44), (2, 0.8), (3, 0.7), (4, 0.9), (5, 1)]
Intersection of A and B:         [(1, 0.3), (2, 0.5), (3, 0.7), (4, 0.25), (5, 1)]
Compliment of A:  [(1, 0.56), (2, 0.2), (3, 0.3), (4, 0.75), (5, 0)]
Set Difference of A and B:  [(1, 0.7), (2, 0.5), (3, 0.3), (4, 0.25), (5, 0)]
Algebric Sum of A and B:         [(1, 0.61), (2, 0.9), (3, 0.91), (4, 0.92), (5, 1)]
Algebric Product of A and B:  [(1, 0.13), (2, 0.4), (3, 0.49), (4, 0.23), (5, 1)]
Bounded Sum of A and B:          [(1, 0.74), (2, 1), (3, 1), (4, 1), (5, 1)]
Bounded Difference of A and B:  [(1, 0), (2, 0.3), (3, 0.4), (4, 0.15), (5, 1)]
```

EXPERIMENT 1B

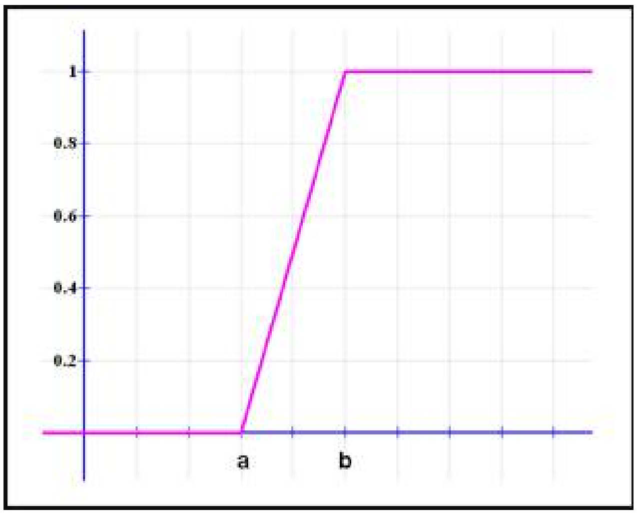
Aim: Implementing Fuzzy Membership Functions

Theory:

The membership function of a fuzzy set is a generalization of the indicator function for classical sets. In fuzzy logic, it represents the degree of truth as an extension of valuation. Degrees of truth are often confused with probabilities, although they are conceptually distinct, because fuzzy truth represents membership in vaguely defined sets, not likelihood of some event or condition.

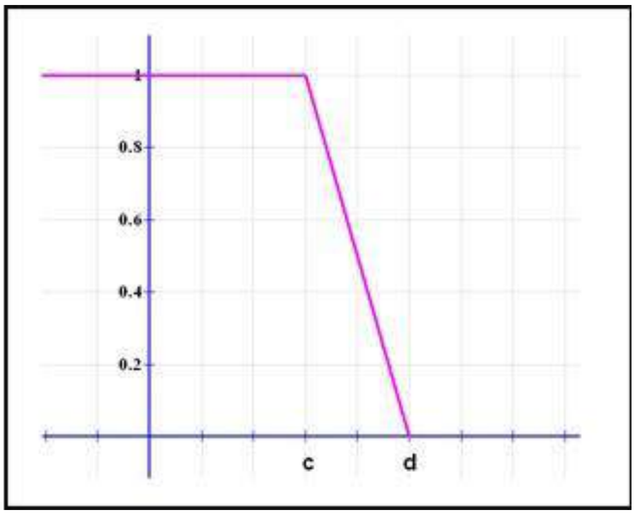
1. L-Shaped Membership Function:

$$\mu_A(x) = \begin{cases} 0, & x < a \\ \dfrac{x-a}{b-a}, & a \le x \le b \\ 1, & x > b \end{cases}$$



2. T-Shaped Membership Function:

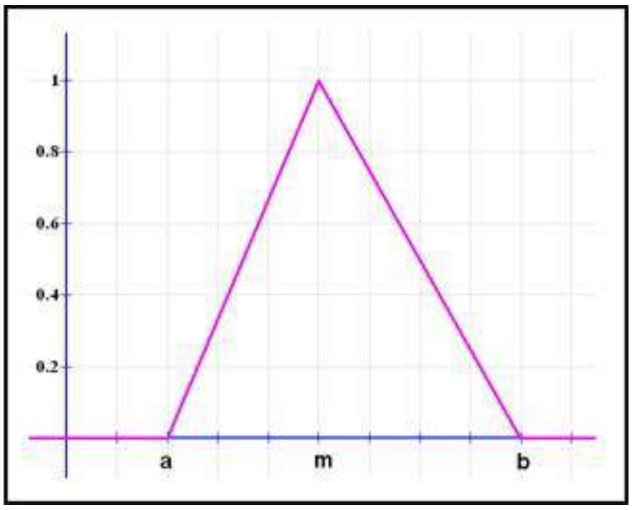
$$\mu_A(x) = \begin{cases} 0, & x > d \\ \dfrac{d-x}{d-c}, & c \le x \le d \\ 1, & x < c \end{cases}$$



3. Triangular Membership Function:

Defined by a lower limit **a**, an upper limit **b**, and a value **m**, where **a < m < b.**

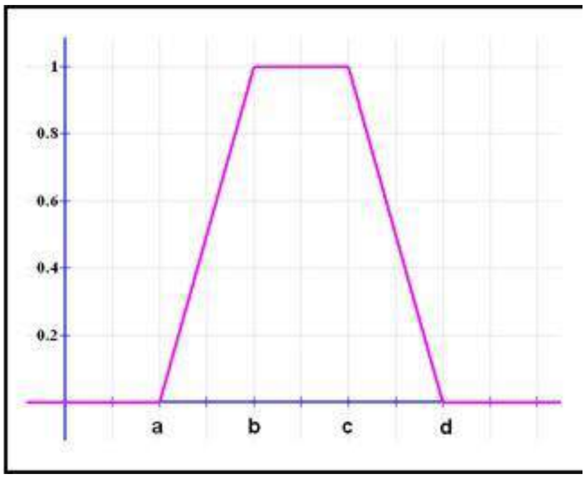
$$\mu_A(x) = \begin{cases} 0, & x \le a \\ \dfrac{x-a}{m-a}, & a<x\le m \\ \dfrac{b-x}{b-m}, & m<x<b \\ 0, & x\ge b \end{cases}$$



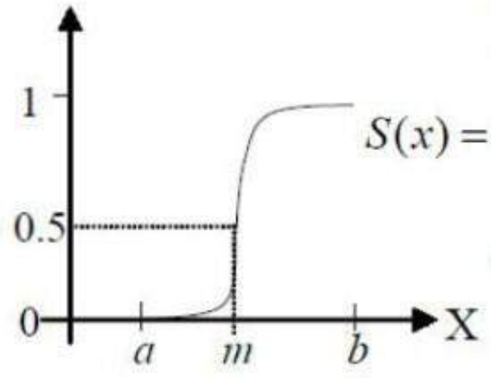
4. Trapezoidal Membership Function:

Defined by a lower limit a, an upper limit d, a lower support limit b, and an upper support limit c, where a < b < c < d.

$$\mu_A(x) = \begin{cases} 0, & (x < a) \text{ or } (x > d) \\ \dfrac{x-a}{b-a}, & a \le x \le b \\ 1, & b \le x \le c \\ \dfrac{d-x}{d-c}, & c \le x \le d \end{cases}$$



5. S-Shaped Membership Function:

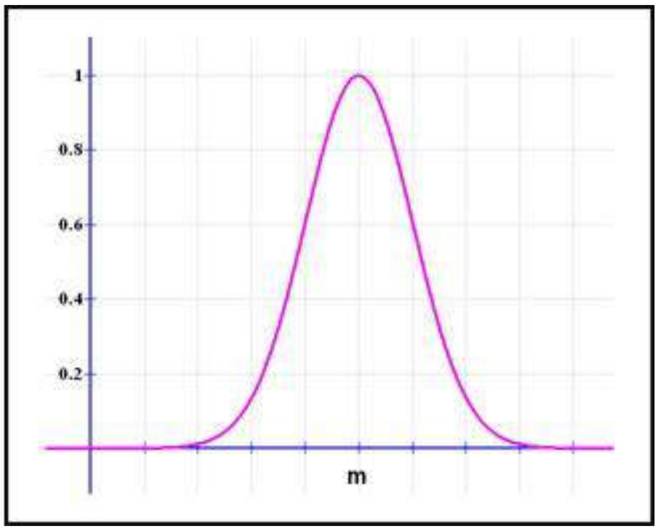


$$S(x) = \begin{cases} 0 & \text{if } x \le a \\ 2\{(x-a)/(b-a)\}^2 & \text{if } x \in (a, m] \\ 1 - 2\{(x-b)/(b-a)\}^2 & \text{if } x \in (m, b) \\ 1 & \text{if } x \ge b \end{cases}$$

6. Gaussian Membership Function:

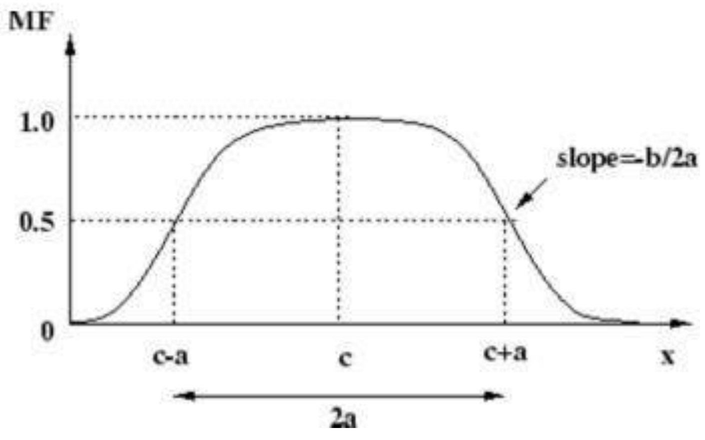
   Defined by a central value **m** and a standard deviation **k > 0**.
   The smaller k is, the narrower the "bell" is.

$$\mu_A(x) = e^{-\frac{(x-m)^2}{2k^2}}$$



7. Bell Membership Function:

$$gbellmf(x; a, b, c) = \frac{1}{1+\left|\frac{x-c}{b}\right|^{2b}}$$

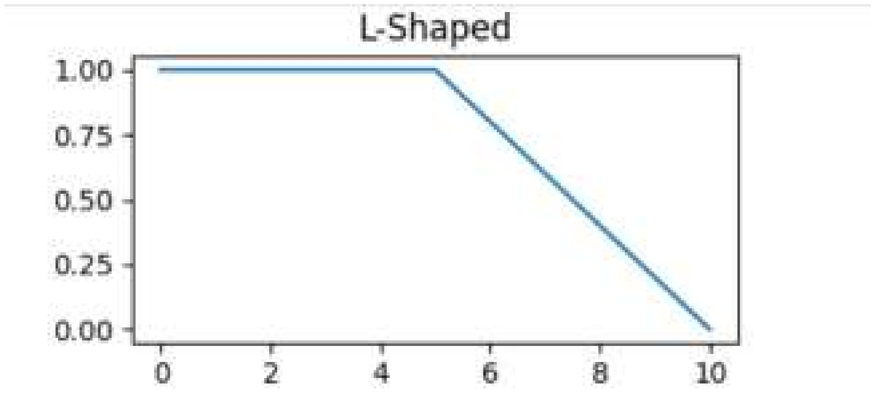


Program:

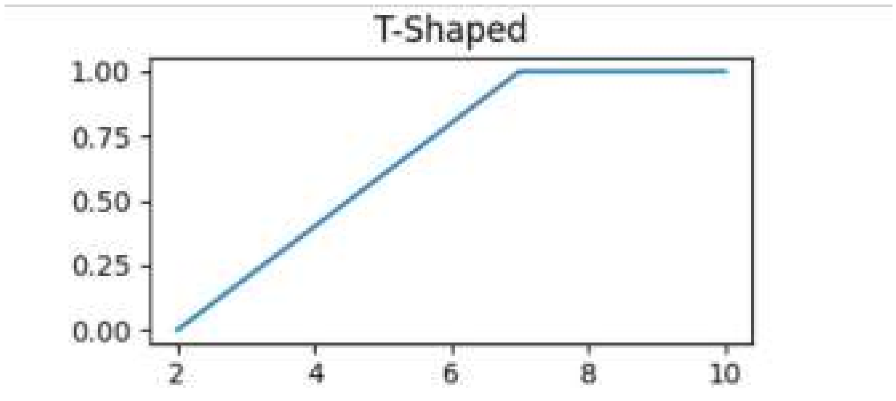
```
import matplotlib.pyplot as plt

import numpy as np

X = np.linspace(0, 100, 1000)

data = {

'L': [[0, 1], [5, 1], [10, 0]],

'T': [[2, 0], [7, 1], [10, 1]],

'TR': [[2.5, 0], [5, 1], [7.5, 0]],

'TZ': [[2, 0], [4, 1], [5, 1], [7, 0]]

}


functions = {

'S': lambda x, a = 20, b = 80, m = 50: np.concatenate([

  0 * x[x <= a],

  2 * np.power((x[(x > a) & (x <= m)] - a) / (b - a), 2),

  1 - 2 * np.power((x[(x > m) & (x < b)] - b) / (b - a), 2),

  0 * x[x >= b] + 1

]),

'G': lambda x: np.power(np.e, -0.5 * np.power((x - 50) / 20, 2)),

'B': lambda x: 1 / (1 + np.power((x - 50) / 20, 8))

}


figure, axes = plt.subplots(3, 3)

figure.tight_layout()

axes = axes.flatten()

figure.delaxes(axes[7])
figure.delaxes(axes[8])


axes[0].set_title('L-Shaped')

axes[1].set_title('T-Shaped')

axes[2].set_title('Triangle Shaped')

axes[3].set_title('Trapezoidal Shaped')

axes[4].set_title('S-Shaped')

axes[5].set_title('Gaussian')

axes[6].set_title('Bell')


for key, axs in zip(data, axes):

  axs.plot(*list(zip(*data[key])))


for key, axs in zip(functions, axes[4:]):

  axs.plot(X, functions[key](X))


plt.show()

figure.savefig('AISC_EXP_1B.png', dpi = 600)
```

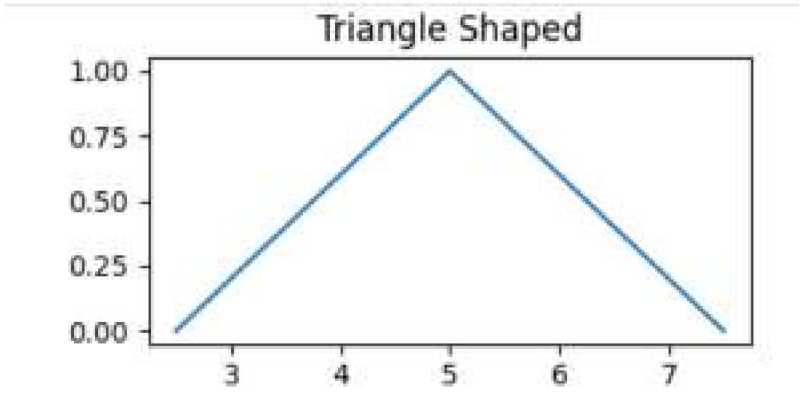
Output:

1)L-Shaped

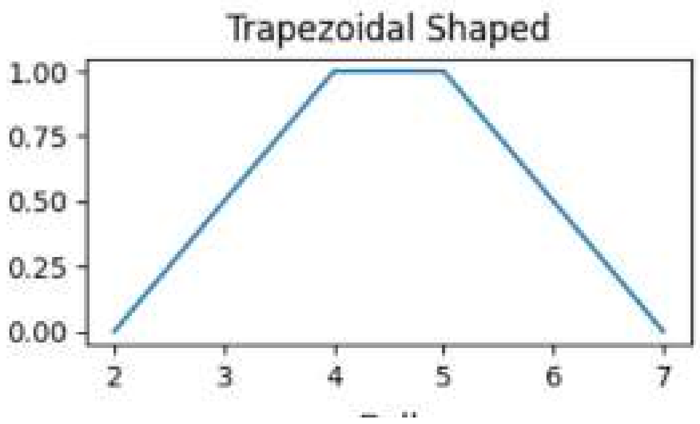


2)T-Shaped

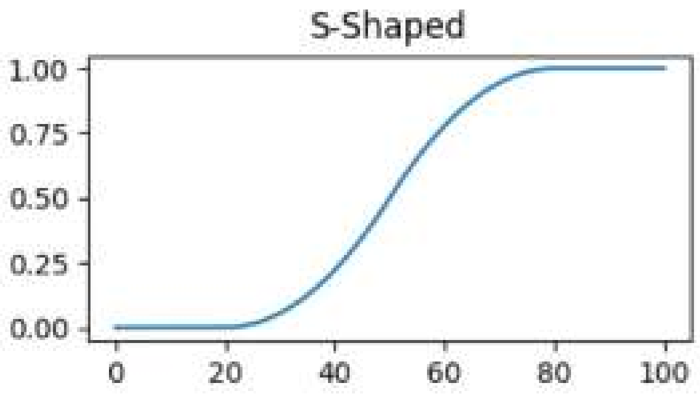




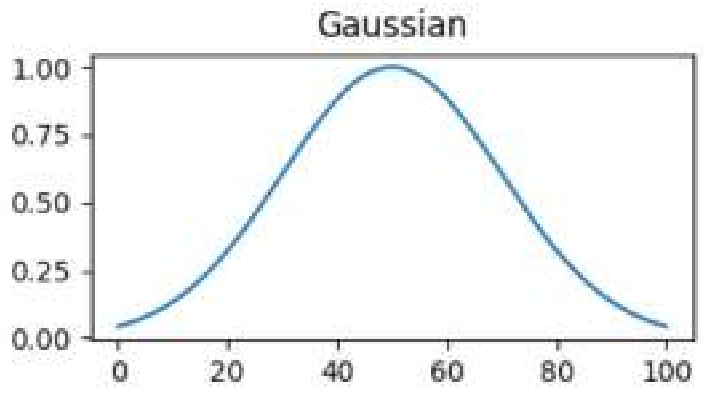
3)Trapezoidal Shaped



4)S-Shaped

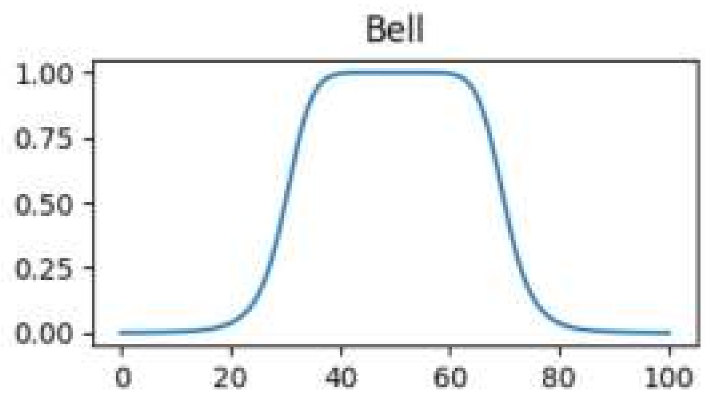


5)Gaussian



6)Bell

C22-85



Conclusion:

1)The fuzzy functions were understood properly

2)The fuzzy membership functions were understood efficiently.

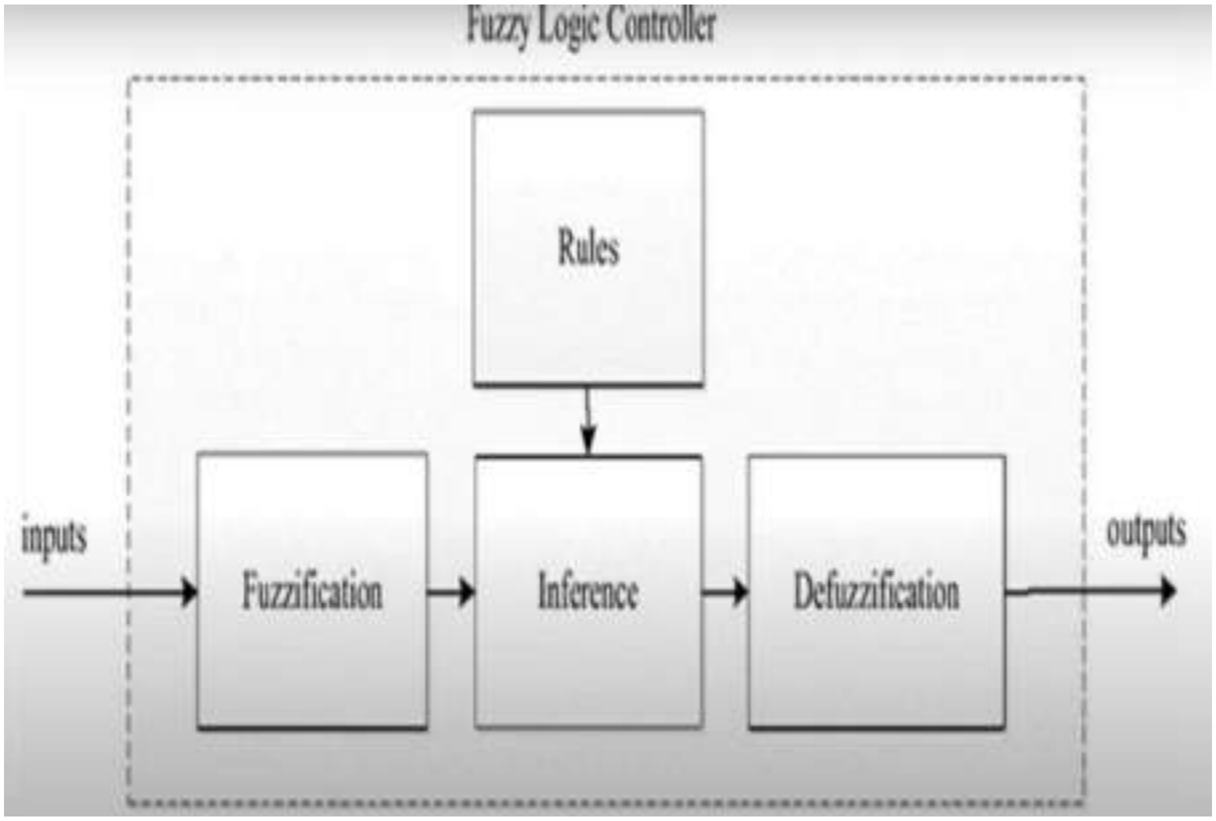
3)The designing of the different shapes of the membership functions were understood.

Exp No- 2

Aim: To Design and implement a Fuzzy Controller System.

Theory:

A fuzzy control system 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).



Smart Washing Machine

Consider an individual doing the laundry for the first time and is not familiar with the type of fabric or detergent to be used. In order to solve the laundry problem, he would seek input from professionals to save the effort. Consider a solution combining the inputs from detergent maker, fabric maker and professional laundry servicemen into an implementable model that would facilitate the unaware person doing laundry for the first time.

The primary concept in designing a Fuzzy Logic Controller (FLC) lies behind the information gathered from various sources of experience or experts (Laundry-Load Characteristics).

Inputs

The characteristics of the laundry load (inputs) include: saturation time and amount of dirt.

Output

The washing parameter (output) include: washing time

Practical Model

In the practical model implementation, consider, for simplicity, as shown in Figure 1, a washing machine with two inputs (Saturation Time and Dirtiness) and one output (Wash Time). For practical purposes type and amount of detergent has been kept manual.

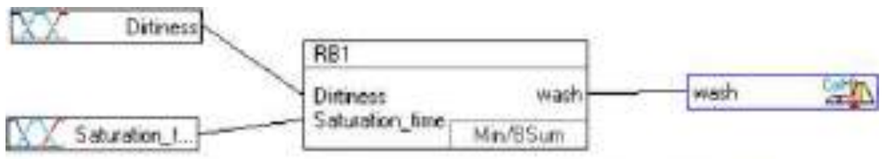


Figure 1. Fuzzy Logic Controller (FLC) of Smart Washing Machine (SWM)

Step 1: Identify input and output variables and decide descriptors for the same.
Inputs:
'dirt' : measured in units
'saturation time' : measured in mins.
Output:
'wash time' measured in minutes.
Descriptors(Terms/Linguistic Values):
We use three descriptors for each of the input variables.
Dirt: { S, M, H } Saturation time: {S, M, L} Washing Time: {VS,S,M,L,VL}
S - Small ST – small VL- Very LOW
M - Medium M - Medium S - SMALL
H - HIGH
L – Large
VH - Very HIGH

Step 2: Define the MFs for each input and output Variable. Fuzzy Subset configuration.
MF for Dirt:
The dirtiness is defined in the range from 0 to 30, by defined fuzzy subsets: Low, Medium and High as shown in Figure 2. For ideal demonstration, the subsets are kept triangular, however they can be adoptive in shapes and ranges in reality. At any given instance, all the three subsets will trigger a certain amount of scale. For example, at range 15(exactly middle), High is 0.0, Low is also 0.0 but

Medium subset is 0.99, which means all three subsets will execute in results extraction to 0.0, 0.99 and 0.0 levels.

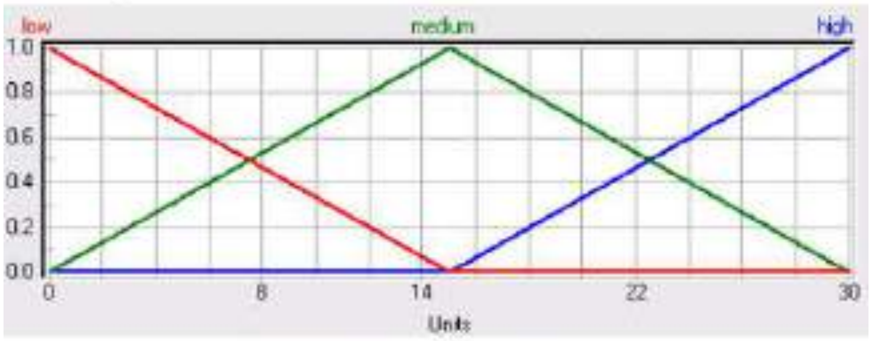


Figure 2. Variable Subset (Dirtiness)

MF for Saturation Time:

The Saturation time is defined in the range of 0 to 10 minutes and has been classified into three sub levels (Low, Medium and High). The same is shown in Figure 3.

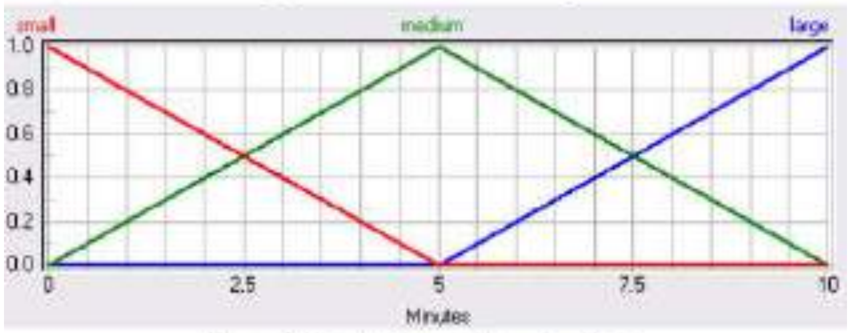


Figure 3. Variable Subset (Saturation Time)

MF for Wash Time:

The wash time is defined in the range from 0 to 15 minutes, based on the fuzzification of above mentioned inputs (saturation time and dirtiness). Further, scaling has been done into five sub levels. The output is defined by the Fuzzy subsets as shown in Figure 4

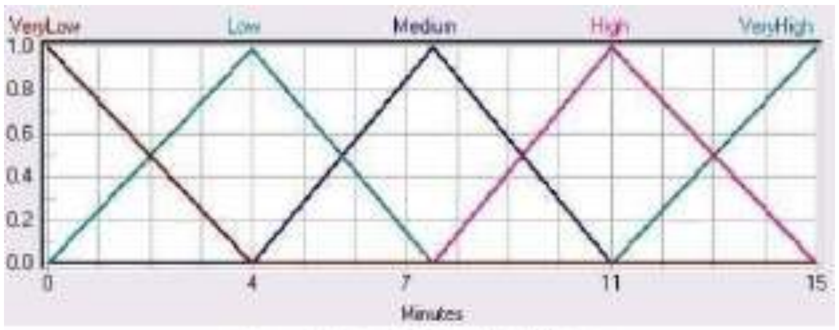


Figure 4. Output Subset (Wash-Time)

Rule Base

Saturation time/Dirt	L	M	H
S	VL	L	M
M	L	M	H
L	M	H	VH

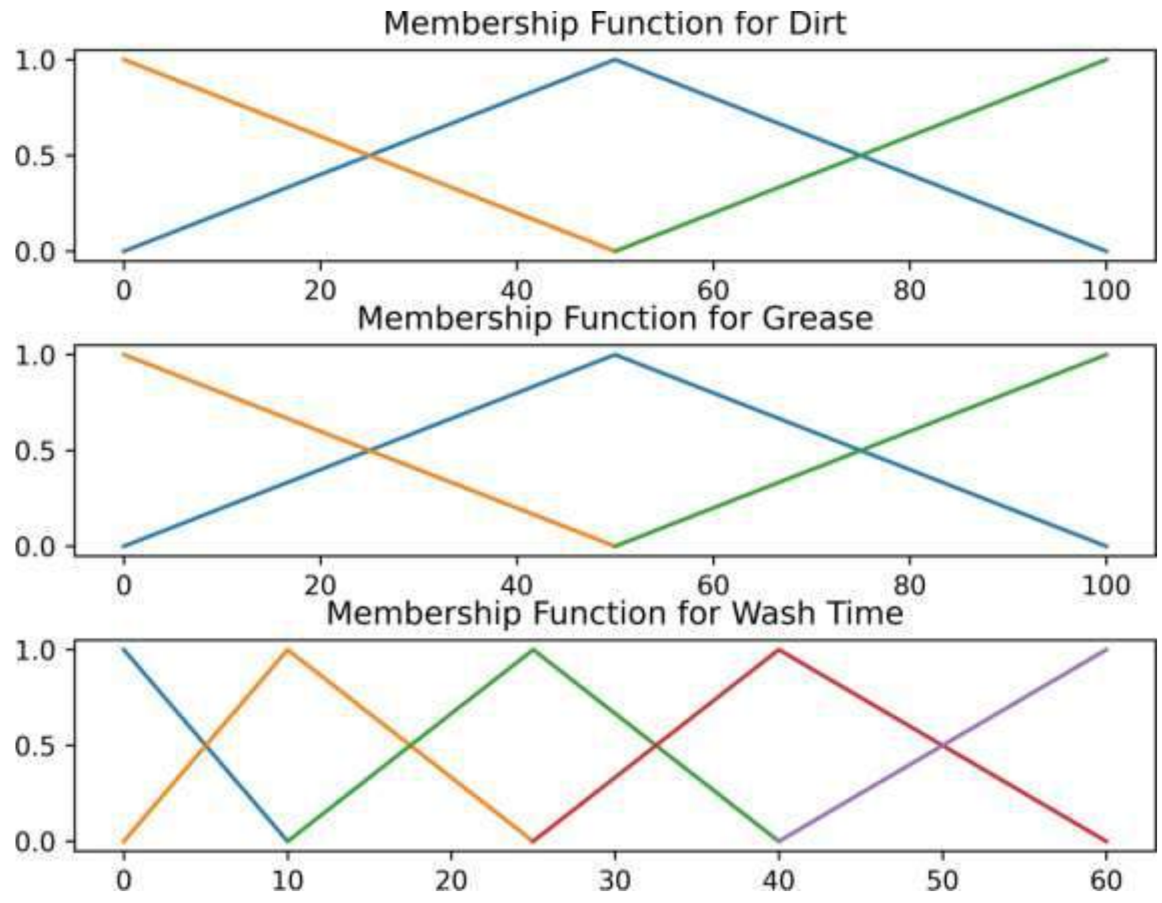
Program

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

import copy

X = np.array(list(range(100)))

dirtmf = [
    [[0, 0], [50, 1], [100, 0]],
    [[0, 1], [50, 0]],
    [[50, 0], [100, 1]]
]

greasemf = copy.deepcopy(dirtmf)

washmf = [
    [[0, 1], [10, 0]],
    [[0, 0], [10, 1], [25, 0]],
    [[10, 0], [25, 1], [40, 0]],
    [[25, 0], [40, 1], [60, 0]],
    [[40, 0], [60, 1]]
]

rules = [
    ['D\\G', 'SG', 'MG', 'HG'],
    ['SD', 'VS', 'M', 'L'],
    ['MD', 'S', 'M', 'L'],
    ['HD', 'M', 'L', 'VL']
]

a = list(map(int, input("Enter dirt & grease:\t").split()))
l = []
for i in range(2):
    l.append([])
    if 0 <= a[i] <= 50:
        l[i].append((50-a[i])/50)
        l[i].append(a[i]/50)
    if 50 <= a[i] <= 100:
        l[i].append((100-a[i])/50)
        l[i].append((a[i]-50)/50)

print('Rules:', l)
l1 = []
for i  in  range(2):
    for j in range(2):
        l1.append(min(l[0][i], l[1][j]))
print('Strength of each rule:', l1)
m = max(l1)
print('Max:', m)
d = (m * 15 + 10) + (40 - m * 15)
print('Defuzzified Value:', d)
print('Rule Table')
for r in rules:
    print(*r)
figure, axes = plt.subplots(3)
```

```
figure.tight_layout()

axes[0].set_title('Membership Function for Dirt')
for d in dirtmf:
    axes[0].plot(*list(zip(*d)))
axes[1].set_title('Membership Function for Grease')
for g in greasemf:
    axes[1].plot(*list(zip(*g)))
axes[2].set_title('Membership Function for Wash Time')
for w in washmf:
    axes[2].plot(*list(zip(*w)))
plt.show()
figure.savefig("AISC_EXP_2.png", dpi = 600)
```

Output

```
Enter dirt & grease:      60 70
Rules: [[0.8, 0.2], [0.6, 0.4]]
Strength of each rule: [0.6, 0.4, 0.2, 0.2]
Max: 0.6
Defuzzified Value: 25.0
Rule Table
D\G SG MG HG
SD VS  M  L
MD S M L
HD M L VL
```



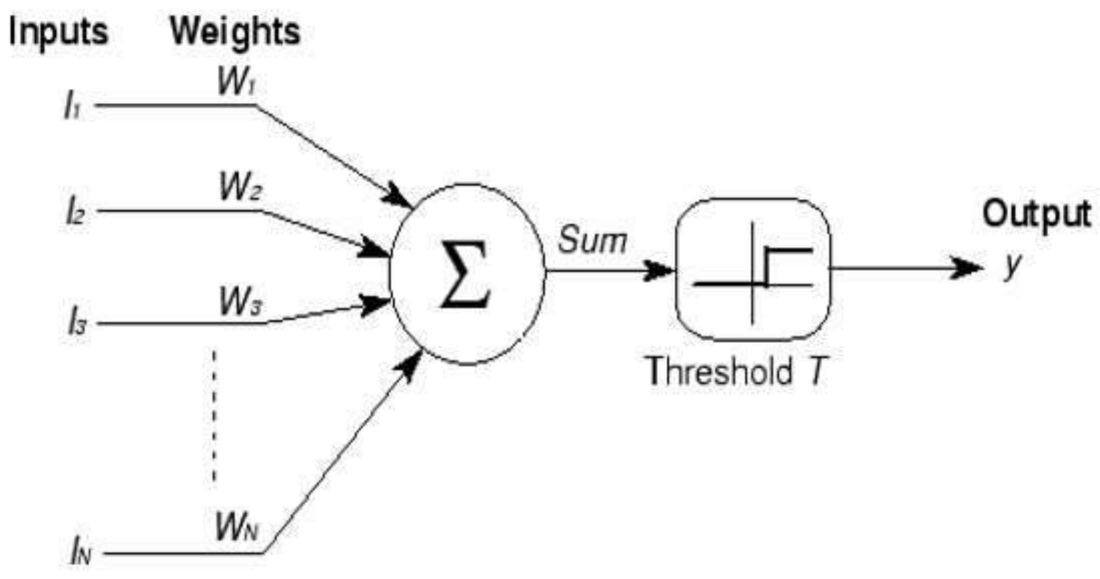
Conclusion:

1)The fuzzy controller system was understood.

2)The fuzzy controller system was created and effectively implemented for Washing Machine Controller System.

3)The fuzzy logic and its applications were understood.

EXP 3: Implementing Mccullon Pitts Model



Threshold=n w-P

N=total no of ips

W=+ve weights

P= -ve weights

The fundamental block of deep learning is artificial neuron i.e. it takes a weighted aggregate of inputs, applies a function and gives an output.

The very first step towards the artificial neuron was taken by Warren McCulloch and Walter Pitts in 1943 inspired by neurobiology, created a model known as McCulloch-Pitts Neuron.

The function (soma) is actually split into two parts: g — The aggregates the inputs to a single numeric value and the function f produces the output of this neuron by taking the output of the g as the input i,e.. a single value as its argument.

The function f will output the value 1 if the aggregation performed by the function g is greater than some threshold else it will return 0.

The inputs x1, x2, ….. xn for the MP Neuron can only take boolean values and the inputs can be inhibitory or excitatory. Inhibitory inputs can have maximum effect on the decision-making process of the model.

In some cases, inhibitory inputs can influence the final outcome of the model.

$$y = 0 \text{ if any } x_i \text{ is inhibitory, else}$$

$$g(x_1, x_2, \ldots x_n) = g(x) = \sum_{i=1}^{n} x_i$$

$$\begin{aligned} y = f(g(x)) &= 1 \text{ if } g(x) \geq b \\ &= 0 \text{ if } g(x) < b \end{aligned}$$

Program

```
from typing import List, Dict, Tuple, Callable

I1: List[List[int]] = [[0], [1]]
I2: List[List[int]] = [[0, 0], [0, 1], [1, 0], [1, 1]]

T: Dict[int, Callable[[int], int]] = {
   0: lambda y: 1 if y >= 1 else 0,
   1: lambda y: 1 if y < 1 else 0,
   2: lambda y: 1 if y >= 2 else 0,
   3: lambda y: 1 if y < 2 else 0
}
```

C22-85

```
O: Dict[int, Callable[[List[int]], int]] = {
    0: lambda i: i[0] + i[1],
    1: lambda x: x[0],
    2: lambda i: ((2 * i[0]) - i[1] > 1) + ((2 * i[1]) - i[0] > 1)
}

models: Dict[str, Tuple[Callable, Callable, List]] = {
    'AND' : (T[2], O[0], I2),
    'OR'  : (T[0], O[0], I2),
    'NAND': (T[3], O[0], I2),
    'NOR' : (T[1], O[0], I2),
    'NOT' : (T[1], O[1], I1),
    'XOR' : (T[0], O[2], I2)
}

for m in models:
    Y: List = list(map(models[m][0], map(models[m][1], models[m][2])))
    print(f'Model: {m}')
    print(f'Input: {models[m][2]}')
    print(f'Output: {Y}\n')
```

Output

```
Model: AND

Input: [[0, 0], [0, 1], [1, 0], [1, 1]]

Output: [0, 0, 0, 1]


Model: OR

Input: [[0, 0], [0, 1], [1, 0], [1, 1]]

Output: [0, 1, 1, 1]


Model: NAND

Input: [[0, 0], [0, 1], [1, 0], [1, 1]]

Output: [1, 1, 1, 0]


Model: NOR

Input: [[0, 0], [0, 1], [1, 0], [1, 1]]

Output: [1, 0, 0, 0]


Model: NOT

Input: [[0], [1]]

Output: [1, 0]


Model: XOR

Input: [[0, 0], [0, 1], [1, 0], [1, 1]]

Output: [0, 1, 1, 0]
```

Conclusion:

The Mc-Culloh Pitts Model was understood.
2)Its working with different gates like AND,OR etc was understood effectively.

Experiment No- 4

Aim: To design and implement a Neural Network using Perceptron learning rule.

Theory:

In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers. A binary classifier is a function which can decide whether or not an input, represented by a vector of numbers, belongs to some specific class. It is a type of linear classifier, i.e., a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector. the perceptron is an algorithm for learning a binary classifier called a Linear classifier: a function that maps its input **x** (a real-valued vector) to an output value $f(\mathbf{x})$ (a single Binary function value):

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}\cdot\mathbf{x} + b > 0, \\ 0 & \text{otherwise} \end{cases}$$

Where **w** is a vector of real-valued weights, **w.x** is the dot product $\sum_{i=1}^{m} w_i x_i$, where m is the number of inputs to the perceptron, and b is the "bias". The bias shifts the decision boundary away from the origin and does not depend on any input value.

The value of $f(\mathbf{x})$ (0 or 1) is used to classify as either a positive or a negative instance, in the case of a binary classification problem. If b is negative, then the weighted combination of inputs must produce a positive value greater than |b| in order to push the classifier neuron over the 0 threshold. Spatially, the bias alters the position (though not the orientation) of the decision boundary. The perceptron learning algorithm does not terminate if the learning set is not linearly separable. If the vectors are not linearly separable learning will never reach a point where all vectors are classified properly. The most famous example of the perceptron's inability to solve problems with linearly non separable vectors is the Boolean [[exclusive-or]] problem.

Program

```
import random

def gen_arr():
    arr = []
    for i in range(4):
        arr_i = random.randint(0, 1)
        arr.append(arr_i)
    return arr

def perceptron(X, Y, Z = 0, eta = 0.1, _iters = 50):
    weights = [0] * len(X[0])
    errors = [1] * len(Y)
    yhat = [1] * len(Y)
    J = []
    for _ in range(_iters):
        for i in range(len(X)):
            s = 0
            for j in range(len(weights)):
                s += X[i][j] * weights[j]
            if s > Z:
                _yhat = 1
            else:
                _yhat = 0
            yhat[i] = _yhat
            for j in range(len(weights)):
                weights[j] += round(eta * (Y[i] - _yhat) * X[i][j], 4)
        for i in range(len(Y)):
            errors[i] = pow((Y[i] - yhat[i]), 2)
        J.append(pow(sum(errors), 0.5))
    return [round(weight, 4) for weight in weights]

X = [[1, 0, 0],
     [1, 0, 1],
     [1, 1, 0],
     [1, 1, 1]]
Y = gen_arr()
print(f'Perceptron Rule for Target : {Y} is \n')
print(f'Input Vectors : {X}\n')
weights = perceptron(X, Y)
print(f'Weights : {weights}')
```

Output

```
Perceptron Rule for Target : [0, 0, 0, 1] is

Input Vectors : [[1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]]

Weights : [-0.2, 0.2, 0.1]
'''
```

C22-85

EXP 4B: Implementing Delta Learning Rule

The idea behind the gradient descent or the delta rule is that we search the hypothesis space of all possible weight vectors to find the best fit for our training samples. Gradient descent acts like a base for BackPropogation algorithms.

Delta Rule can be understood by looking it as training an unthresholded perceptron which is trained using gradient descent. The linear combination of weights and the inputs associated with them acts as an input to activation function.

O(x)= w.x

Before going into the activation function, we need to know how do we calculate the **training error** in the weights in case of misclassification. In Gradient Descent, we commonly use **half the squared difference between the ouput and obtained value** as the **training error** for our hypotheses.

$$E(\vec{w}) = \frac{1}{2} \sum_{d \epsilon D} (t_d - o_d)^2$$

Training error for Gradient Descent

- D is set of training samples.
- t is the target ouput for training example ‘d’.
- o is the output of linear unit for training example ‘d’.

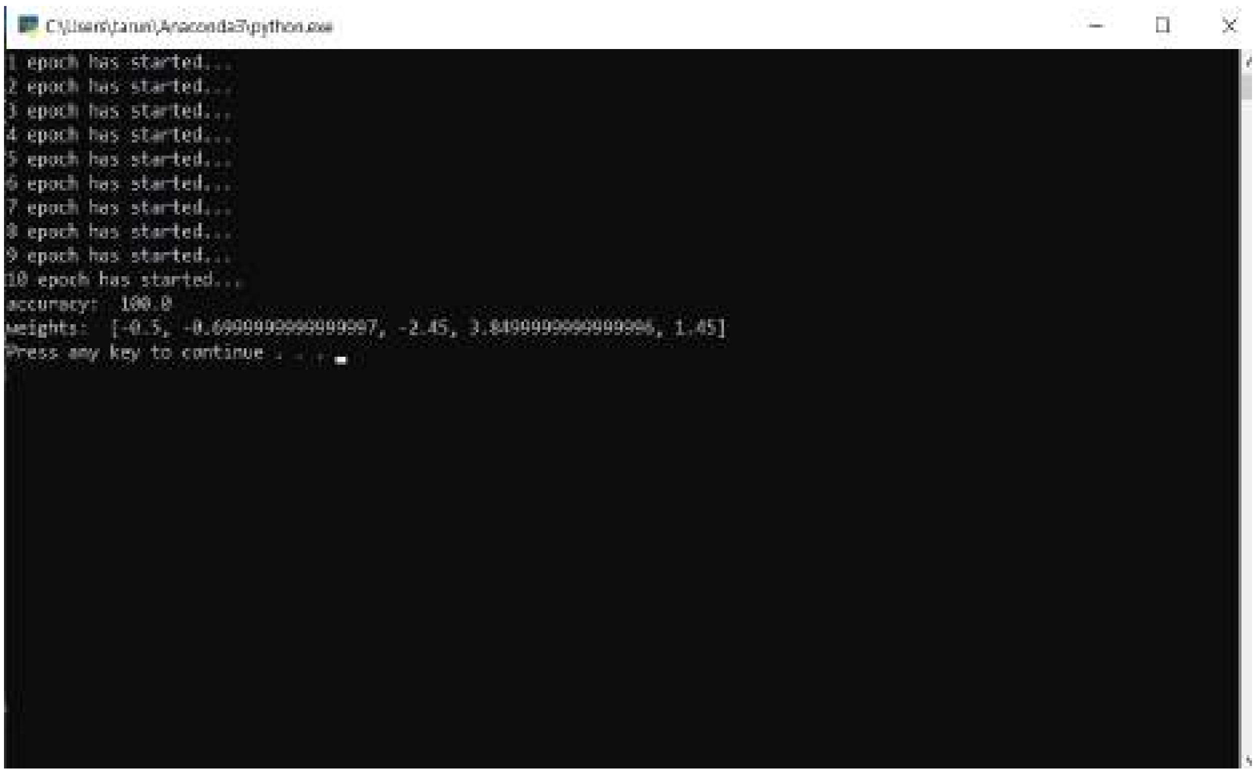
We have arrived at our final euqation on how to update our weights using delta rule. One more key difference is that, in perceptron rule we modify the weights after training all the samples, but in delta rule we update after every misclassification, making the chance of reaching global minima high.

Algorithm:

- Initialize the weight vector.
- For each training sample in dataset, apply the activation function and if any error occurs, update the weight according to the rule.

Repeat it for finite number of epochs, to make it more accurate.

```
class Perceptron:
    def __init__(self):
        self.weights=[]


#activation function
    def activation(self,data):
        #initializing with threshold value
        activation_val=self.weights[0]
        activation_val+=np.dot(self.weights[1:],data)
        return 1 if activation_val>=0 else 0

def fit(self,X,y,lrate,epochs):
        #initializing weight vector
        self.weights=[0.0 for i in range(len(X.columns)+1)]
        #no.of iterations to train the neural network
        for epoch in range(epochs):
            print(str(epoch+1),"epoch has started...")
            for index in range(len(X)):
                x=X.iloc[index]
                predicted=self.activation(x)
                #check for misclassification
                if(y.iloc[index]==predicted):
                    pass
                else:
                    #calculate the error value
                    error=y.iloc[index]-predicted
                    #updation of threshold
                    self.weights[0]=self.weights[0]+lrate*error
      #updation of associated self.weights acccording to Delta rule
                    for j in range(len(x)):

self.weights[j+1]=self.weights[j+1]+lrate*error*x[j]

def predict(self,x_test):
        predicted=[]
        for i in range(len(x_test)):
            #prediction for test set using obtained weights
            predicted.append(self.activation(x_test.iloc[i]))
        return predicted

def accuracy(self,predicted,original):
        correct=0
        lent=len(predicted)
        for i in range(lent):
            if(predicted[i]==original.iloc[i]):
                correct+=1
        return (correct/lent)*100

ef getweights(self):
        return self.weights

from Perceptron import Perceptron
import pandas as pd
from sklearn.model_selection import train_test_split
#read data from .csv file
data=pd.read_csv("iris.csv")
data.columns=["petal_length","petal_width","sepal_length","sepal_wi
dth","class"]
classes=data["class"]
data=data.drop(columns="class")
#splitting test and train data for iris
x_train,x_test,y_train,y_test=train_test_split(data,classes)
#training the percpetron
model=Perceptron()
model.fit(x_train,y_train,0.5,10)
pred=model.predict(x_test)
print("accuracy: ",model.accuracy(pred,y_test))
print("weights: ",model.getweights())
```



C:\Users\tarun\Anaconda3\python.exe

```
1 epoch has started...
2 epoch has started...
3 epoch has started...
4 epoch has started...
5 epoch has started...
6 epoch has started...
7 epoch has started...
8 epoch has started...
9 epoch has started...
10 epoch has started...
accuracy:  100.0
weights:  [-0.5, -0.6999999999999997, -2.45, 3.8499999999999996, 1.45]
Press any key to continue . . .
```

Conclusion:

1)The Perceptron Learning rule was understood.

2)The Delta running rule was understood clearly.

3)The difference and the efficiency between the Mc-culloh pitts model and perceptron model were understood clearly.

C22-85

Experiment – 5

Aim:

Select a relevant real-world problem. Prepare the task environment for the selected problem using PEAS description. Formulate the selected problem in terms of initial state, actions, transition model, goal test and path cost.

Theory:

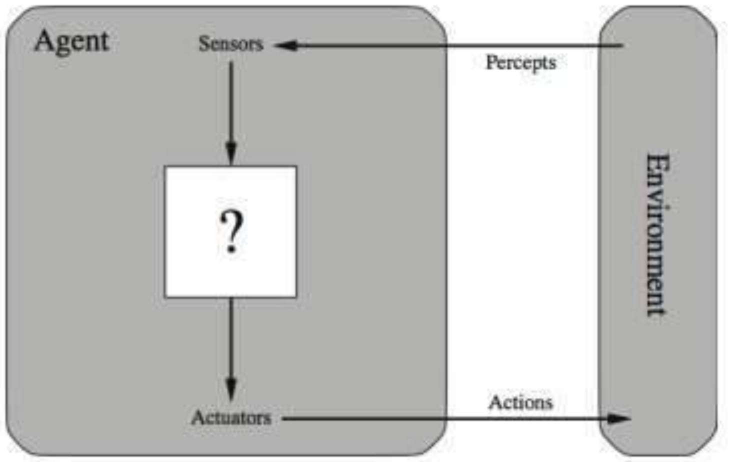
An AI system can be defined as the study of the rational agent and its environment. The agents sense the environment through sensors and act on their environment through actuators. An AI agent can have mental properties such as knowledge, belief, intention, etc.

What is an Agent?

An agent is anything that can be viewed as:
• perceiving its environment through sensors and
• acting upon that environment through actuators

An agent program runs in cycles of: 1. perceive, 2. think, and 3. act.

Agent = Architecture + Program



An agent can be:

• Human-Agent: A human agent has eyes, ears, and other organs which work for sensors and hands, legs, vocal tract work for actuators.
• Robotic Agent: A robotic agent can have cameras, infrared range finder, NLP for sensors and various motors for actuators.
• Software Agent: Software agent can have keystrokes, file contents as sensory input and act on those inputs and display output on the screen.

PEAS:

PEAS System is used to categorize similar agents together. The PEAS system delivers the performance measure with respect to the environment, actuators and sensors of the respective agent. Most of the highest performing agents are Rational Agents.

Rational Agent: The rational agent considers all possibilities and chooses to perform the highly efficient action. For example, it chooses the shortest path with low cost for high efficiency. PEAS stands for Performance measure, Environment, Actuator, Sensor.

• Performance Measure: Performance measure is the unit to define the success of an agent. Performance varies with agents based on their different precept.
• Environment: Environment is the surrounding of an agent at every instant. It keeps changing with time if the agent is set in motion. There are 5 major types of environments:
  o Fully Observable & Partially Observable
  o Episodic & Static
  o Static & Dynamic
  o Discrete & Continuous
  o Deterministic & Stochastic
• Actuator: Actuator is a part of the agent that delivers the output of an action to the environment.
• Sensor: Sensors are the receptive parts of an agent which takes in the input for the agent.

Topic:

Automated Vacuum Cleaners - An automated vacuum cleaner that can detect the dirt and clean and also has good coverage i.e. covering each and every corner of the place it needs to clean.

PEAS Description:

Performance: cleanness, efficiency: distance traveled to clean, battery life, security.

Environment: room, table, wood floor, carpet, different obstacles.

Actuators: wheels, different brushes, vacuum extractor.

Sensors: camera, dirt detection sensor, cliff sensor, bump sensors, infrared wall sensors.

Now, Let's write the algorithm for the Vacuum cleaner...

• Percepts: location and content (location sensor, dirt sensor).
• Actions: Left, Right, Clean, No Operation

Percept	Action
[A, clean]	Right
[A, dirty]	Clean
[B, clean]	Left
[B, dirty]	Clean

Environment Characteristics:

Partially Observable
Single Agent
Stochastic
Dynamic
Continuous
Sequential

What it means?

• Fully observable (vs. partially observable): An agent's sensors give it access to the complete state of the environment at each point in time.

• Deterministic (vs. stochastic): The next state of the environment is completely determined by the current state and the action executed by the agent. (If the environment is deterministic except for the actions of other agents, then the environment is strategic)

• Static (vs. dynamic): The environment is unchanged while an agent is deliberating. (The environment is semi-dynamic if the environment itself does not change with the passage of time but the agent's performance score does.)

• Discrete (vs. continuous): A limited number of distinct, clearly defined Percepts and actions. E.g., checkers is an example of a discrete environment, while self-driving car evolves in a continuous one.

• Single agent (vs. multi-agent): An agent operating by itself in an environment.

• Episodic (vs. sequential): The agent's experience is divided into atomic "episodes" (each episode consists of the agent perceiving and then performing a single action), and the choice of action in each episode depends only on the episode itself.

Problem Formulation:

Initial State: Vacuum cleaner should be in proper and working condition

Actions: Vacuum cleaner should be able to clean dirt if any found

Transition Model: The state of cleaning dirt properly

Goal State: All the dirt being collected from the given space into vacuum cleaner

Path Cost: This depends on components needed to make a vacuum cleaner

Conclusion:

1)The vaccum cleaner problem was understood clearly.
2)The PEAS description defintion was understood.
3)The PEAS description for 8 puzzle problem was generated effectively.

C22-85

Experiment No- 6

Aim: Knowledge representation and creating a knowledge base for the selected problem.

Selected Problem: Robot Navigation System .

Step 1: Define Problem Statement in English Sentences.

Abstract: We consider a warehouse robot that can move in 4 directions viz, left, right, forward and backward and we are building the knowledge base with prerequisite rules for the same. The robot at source can pick up a packet from a bin, identifies the packet size and moves to destination. The destination has 3 bins viz., bin1 for small sized packets, bin2 for medium sized packets and bin3 for large sized packets. The rules for handling the same will be given below. How the robot moves from start point to goal point is dependent on search goal based strategy.

Problem Statements:

A robot can move between several bins. The robot can pick up an object from a bin if the hand is above the bin and the hand is empty and the bin is not empty. The robot after picking will identify the object size either by volume or by weight. An object can be identified as either a small, medium or large object. The robot has identified the object size and it will move to the destination position to drop the object. If the object is small in size, the robot will drop the object in bin 1 at the destination. If the object is medium in size, the robot will drop the object in bin 2 at the destination. If the object is large in size, the robot will drop the object in bin 3 at the destination.

Step 2: Convert English statements to Propositional statements.

A -> A robot can move between several bins.
B -> Robot can pick up an object from a bin
C -> Robot’s hand is above the bin
D -> Robot’s hand is empty
S -> The bin is not empty
E -> The robot will identify the object size by volume
F -> The robot will identify the object size by weight
G -> An object can be identified as a small object
H -> An object can be identified as a medium object
I -> An object can be identified as a large object
J -> The robot has identified the object size
K -> The robot will move to the destination position
R -> The robot will drop the object
L -> The object is small in size
M -> The robot will drop the object in bin 1 at the destination
N -> The object is medium in size

O -> The robot will drop the object in bin 2 at the destination
P -> The object is large in size
Q -> The robot will drop the object in bin 3 at the destination

The propositional statements:

A
(C ∧ D ∧ ¬S) -> B
J -> (E ∨ F)
G ∨ H ∨ I
J ∨ K ∨ R
L -> M
N -> O
P -> Q

Step 3: Identify the variables and predicates.

The Variables (Objects) are:

• r : Instance of robot
• o : Instance of object
• b : Instance of bin (can be source bin or destination bin)

The Constants are:

BIN1 : First bin that stores small objects
BIN2 : First bin that stores medium objects
BIN3 : First bin that stores large objects

The Predicates (actions) are:

• drop(r, x, y) (robot r drops object x into bin y)
• move(r, y) (move robot r such that its hand is above the bin y)
• grab(r, x, y) (robot r picks up object x from bin y).
• holding(r, x) (the robot r is holding x)
• over(r, y) (the robot r is such that its hand is over bin y)
• empty(r) (the robot‘ s(r) hand is empty)
• empty(b) (the bin b is empty)
• determinesize(r,o) (The robot r identifies size)
• volume(r,o) (The robot r identifies object’s 0 size by volume)
• weight(r,o) (The robot r identifies object’s 0 size by weight)
• in(x, y) (object x is in the bin y).
• issmall(o) (size of object o is small)
• ismedium(o) (size of object o is medium)
• islarge(o) (size of object o is large)

Step 4: Identify the Quantifiers.

Quantifier in Statement(s): An object can be identified as either a small, medium or large object. Propositional Statement: $G \lor H \lor I$

Defining statement using Categories:

Object : Universal Category of Objects
SmallObject : Category containing small objects;
SmallObject ⊂ Object
MediumObject : Category containing medium objects;
MediumObject ⊂ Object
LargeObject : Category containing large objects;
LargeObject ⊂ Object

Knowledge Representation of statement is (using categories):

∀ o ∈ Object ,
(o ∈ SmallObject ∨ o ∈ MediumObject ∨ o ∈ LargeObject) Partition({SmallObject, MediumObject, LargeObject}, Object)

First Order Logic Statement:

∀ o (issmall(o) ∨ ismedium(o) ∨ islarge(o))

Step 5: Identify Implication relationship(s).

$(C \land D \land \neg S) \rightarrow B$
$J \rightarrow (E \lor F)$
$L \rightarrow M$
$N \rightarrow O$
$P \rightarrow Q$

Step 6: Rearrange the statements into first order logic.

A -> A robot can move between several bins.

First Order Logic Statement is: ∃r∀b (move(r,b))

B -> Robot can pick up an object from a bin

C -> Robot’s hand is above the bin

D -> Robot’s hand is empty

S -> The bin is not empty

(C ∧ D ∧ ¬S) -> B

First Order Logic Statement is: ∃r∃b∃o ((over(r,b) ∧ empty(r) ∧ ¬empty(b)) -> grab(r,o,b))

J -> The robot has identified the object size

E -> The robot will identify the object size by volume

F -> The robot will identify the object size by weight

J -> (E ∨ F)

First Order Logic Statement is: ∃r ∀o (determinesize(r,o) -> (volume(r,o) ∨ weight(r,o))

G -> An object can be identified as a small object

H -> An object can be identified as a medium object

I -> An object can be identified as a large object

G ∨ H ∨ I

First Order Logic Statement is: ∀o (issmall(o) ∨ ismedium(o) ∨ islarge(o))

J -> The robot has identified the object size

K -> The robot will move to the destination position

R -> The robot will drop the object

J ∨ K ∨ R

First Order Logic Statement is: ∃r∃b∃o (determinesize(r,o) ∧ move(r,b) ∧ drop(r,o,b))

L -> The object is small in size

M -> The robot will drop the object in bin 1 at the destination

L -> M

First Order Logic Statement is: ∃o∃r (issmall(o) -> drop(r,o,BIN1))

N -> The object is medium in size

O -> The robot will drop the object in bin 2 at the destination

N -> O

First Order Logic Statement is: ∃o∃r (ismedium(o) -> drop(r,o,BIN2))

P -> The object is large in size

Q -> The robot will drop the object in bin 3 at the destination

P -> Q

First Order Logic Statement is: ∃o∃r (islarge(o) -> drop(r,o,BIN3))

Conclusion:

1)The Knowledge for the Robot Navigation System problem was generated.
2)All the rules used to solve the Robot Navigation System problem effectively were understood clearly.

C22-85

Experiment No- 7

Aim: To implement appropriate search algorithm to reach goal state for the selected problem.

Theory:

Searching is the universal technique of problem solving in AI. There are some single-player games such as tile games, Sudoku, crossword, etc. The search algorithms help you to search for a particular position in such games.

Search Terminology

• **Problem Space** − It is the environment in which the search takes place. (A set of states and set of operators to change those states)
• **Problem Instance** − It is Initial state + Goal state.
• **Problem Space Graph** − It represents problem state. States are shown by nodes and operators are shown by edges.
• **Depth of a problem** − Length of a shortest path or shortest sequence of operators from Initial State to goal state.
• **Space Complexity** − The maximum number of nodes that are stored in memory.
• **Time Complexity** − The maximum number of nodes that are created.
• **Admissibility** − A property of an algorithm to always find an optimal solution.
• **Branching Factor** − The average number of child nodes in the problem space graph.
• **Depth** − Length of the shortest path from initial state to goal state.

Brute-Force Search Strategies

They are most simple, as they do not need any domain-specific knowledge. They work fine with small number of possible states.

Requirements –
• State description
• A set of valid operators
• Initial state
• Goal state description

Breadth-First Search

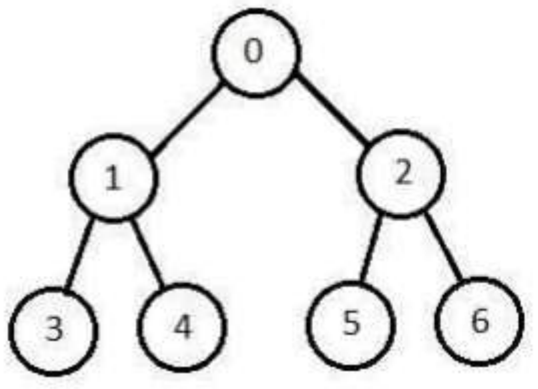
It starts from the root node, explores the neighbouring nodes first and moves towards the next level neighbours. It generates one tree at a time until the solution is found. It can be implemented using FIFO queue data structure. This method provides shortest path to the solution.

If branching factor (average number of child nodes for a given node) = b and depth = d, then number of nodes at level d = bd .

The total no of nodes created in worst case is b + b2 + b3 + … + bd .

Disadvantage − Since each level of nodes is saved for creating next one, it consumes a lot of memory space. Space requirement to store nodes is exponential.

Its complexity depends on the number of nodes. It can check duplicate nodes.



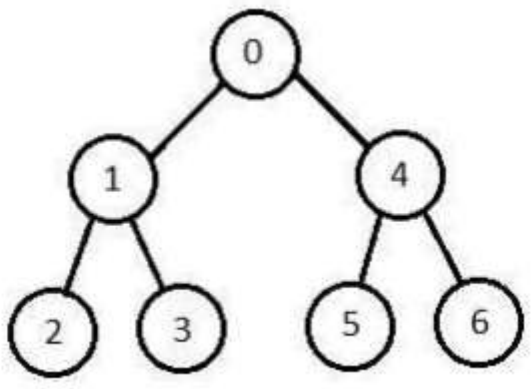
Depth-First Search

It is implemented in recursion with LIFO stack data structure. It creates the same set of nodes as Breadth-First method, only in the different order.

As the nodes on the single path are stored in each iteration from root to leaf node, the space requirement to store nodes is linear. With branching factor b and depth as m, the storage space is bm.

Disadvantage − This algorithm may not terminate and go on infinitely on one path. The solution to this issue is to choose a cut-off depth. If the ideal cut-off is d, and if chosen cut-off is lesser than d, then this algorithm may fail. If chosen cut-off is more than d, then execution time increases.

Its complexity depends on the number of paths. It cannot check duplicate nodes.



Informed (Heuristic) Search Strategies

To solve large problems with large number of possible states, problem-specific knowledge needs to be added to increase the efficiency of search algorithms.

Heuristic Evaluation Functions

They calculate the cost of optimal path between two states. A heuristic function for sliding-tiles games is computed by counting number of moves that each tile makes from its goal state and adding these number of moves for all tiles.

Pure Heuristic Search

It expands nodes in the order of their heuristic values. It creates two lists, a closed list for the already expanded nodes and an open list for the created but unexpanded nodes.

In each iteration, a node with a minimum heuristic value is expanded, all its child nodes are created and placed in the closed list. Then, the heuristic function is applied to the child nodes and they are placed in the open list according to their heuristic value. The shorter paths are saved and the longer ones are disposed.

A * Search

It is best-known form of Best First search. It avoids expanding paths that are already expensive, but expands most promising paths first.
f(n) = g(n) + h(n), where
• g(n) the cost (so far) to reach the node
• h(n) estimated cost to get from the node to the goal
• f(n) estimated total cost of path through n to goal.

It is implemented using priority queue by increasing f(n).

Greedy Best First Search

It expands the node that is estimated to be closest to goal. It expands nodes based on $f(n) = h(n)$. It is implemented using priority queue.
Disadvantage − It can get stuck in loops. It is not optimal

Source Code:

```
class Node:
    def__init_(self,data,level,fval):
        """ Initialize the node with the data, level of the node and the calculated fvalue
"""
        self.data = data
        self.level = level
        self.fval = fval

    def generate_child(self):
        """ Generate child nodes from the given node by moving the blank space
            either in the four directions {up,down,left,right} """
        x,y = self.find(self.data,'_')
        """ val_list contains position values for moving the blank space in either of
            the 4 directions [up,down,left,right] respectively. """
        val_list = [[x,y-1],[x,y+1],[x-1,y],[x+1,y]]
        children = []
        for i in val_list:
            child = self.shuffle(self.data,x,y,i[0],i[1])
            if child is not None:
                child_node = Node(child,self.level+1,0)
                children.append(child_node)
        return children

    def shuffle(self,puz,x1,y1,x2,y2):
        """ Move the blank space in the given direction and if the position value are out
            of limits the return None """
        if x2 >= 0 and x2 < len(self.data) and y2 >= 0 and y2 < len(self.data):
            temp_puz = []
            temp_puz = self.copy(puz)
            temp = temp_puz[x2][y2]
            temp_puz[x2][y2] = temp_puz[x1][y1]
            temp_puz[x1][y1] = temp
            return temp_puz
        else:
            return None



    def copy(self,root):
        """ Copy function to create a similar matrix of the given node"""
        temp = []
        for i in root:
            t = []
            for j in i:
                t.append(j)
            temp.append(t)
        return temp

    def find(self,puz,x):
        """ Specifically used to find the position of the blank space """
        for i in range(0,len(self.data)):
            for j in range(0,len(self.data)):
                if puz[i][j] == x:
                    return i,j



class Puzzle:
    def__init__(self,size):
        """ Initialize the puzzle size by the specified size,open and closed lists to empty
"""
        self.n = size
        self.open = []
        self.closed = []

    def accept(self):
        """ Accepts the puzzle from the user """
        puz = []
        for i in range(0,self.n):
            temp = input().split(" ")
            puz.append(temp)
        return puz

    def f(self,start,goal):
        """ Heuristic Function to calculate hueristic value f(x) = h(x) + g(x) """
        return self.h(start.data,goal)+start.level

    def h(self,start,goal):
        """ Calculates the different between the given puzzles """
        temp = 0
        for i in range(0,self.n):
            for j in range(0,self.n):
                if start[i][j] != goal[i][j] and start[i][j] != '_':
                    temp += 1
        return temp


    def process(self):
        """ Accept Start and Goal Puzzle state"""
        print("Enter the start state matrix \n")
        start = self.accept()
        print("Enter the goal state matrix \n")
        goal = self.accept()
        start = Node(start,0,0)
        start.fval = self.f(start,goal)
        """ Put the start node in the open list"""
        self.open.append(start)
        print("\n\n")
        while True:
            cur = self.open[0]
            print("")
            print("  | ")
            print("  | ")
            print(" \\\'/ \n")
            for i in cur.data:
                for j in i:
                    print(j,end=" ")
                print("")
            """ If the difference between current and goal node is 0 we have reached the goal node"""
            if(self.h(cur.data,goal) == 0):
                break
            for i in cur.generate_child():
                i.fval = self.f(i,goal)
                self.open.append(i)
            self.closed.append(cur)
            del self.open[0]

            """ sort the opne list based on f value """
            self.open.sort(key = lambda x:x.fval,reverse=False)


puz = Puzzle(3)
puz.process()
```

Output:

```
Enter the start state matrix

1 2 3
_ 4 6
7 5 8
Enter the goal state matrix

1 2 3
4 5 6
7 8 _




  |
  |
 \'/

1 2 3
_ 4 6
7 5 8

  |
  |
 \'/

1 2 3
4 _ 6
7 5 8

  |
  |
 \'/

1 2 3
4 5 6
7 _ 8

  |
  |
 \'/

1 2 3
4 5 6
7 8 _
```

Conclusion:

1)The A* algorithm was studied and understood clearly.

2)The use of A* algorithm to solve the 8-puzzle problem was understood and implement effectively.

Assignment -1

Aim: Case study on Hybrid Systems/Application.

Theory:

(1) What are hybrid systems? Types of Hybrid Systems. Explain ANIFS architecture with example.

Ans:

Hybrid systems: A Hybrid system is an intelligent system which is framed by combining atleast two intelligent technologies like Fuzzy Logic, Neural networks, Genetic algorithm, reinforcement Learning, etc. The combination of different techniques in one computational model make these systems possess an extended range of capabilities. These systems are capable of reasoning and learning in an uncertain and imprecise environment. These systems can provide human-like expertise like domain knowledge, adaptation in noisy environment etc.

Types of Hybrid Systems:

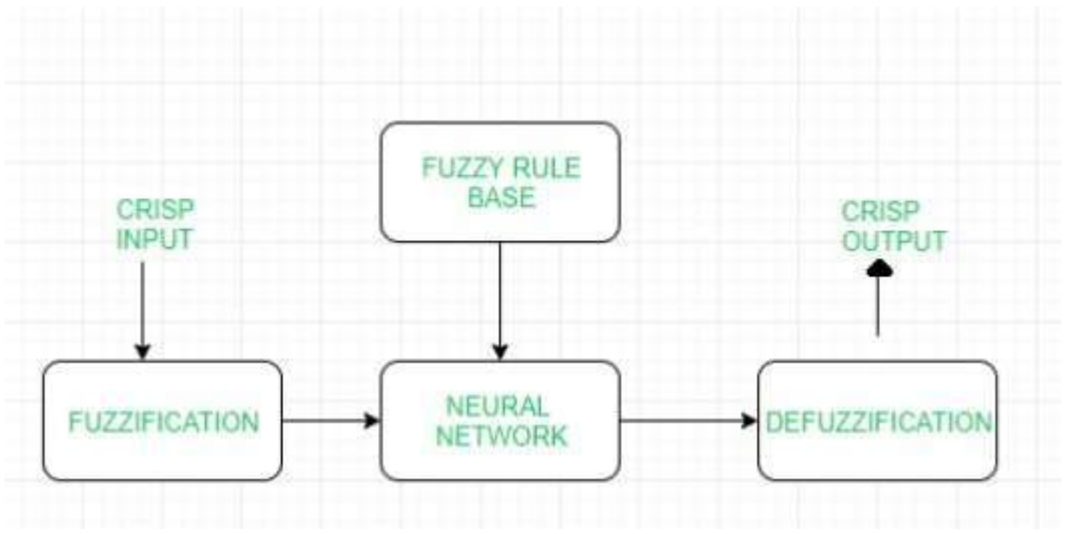
•Neuro Fuzzy Hybrid systems

•Neuro Genetic Hybrid systems

•Fuzzy Genetic Hybrid systems

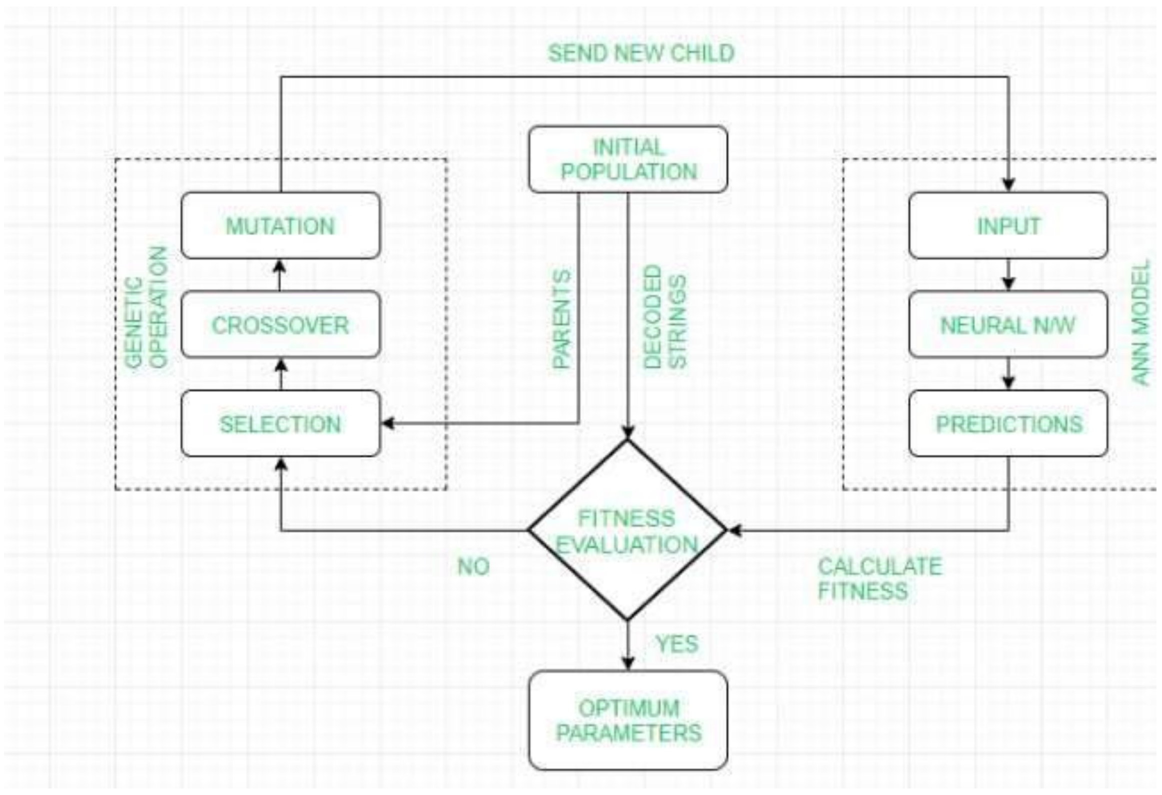
(A) Neuro Fuzzy Hybrid systems:

Neuro fuzzy system is based on which is trained on the basis of working of neural network theory. The learning process operates only on the local information and causes only local changes in the underlying fuzzy system. A neuro-fuzzy system can be seen as a 3-layer feedforward neural network. The first layer represents input variables, the middle (hidden) layer represents fuzzy rules and the third layer represents output variables. Fuzzy sets are encoded as connection weights within the layers of the network, which provides functionality in processing and training the model.



(B) Neuro Genetic Hybrid systems:

A Neuro Genetic hybrid system is a system that combines Neural networks: which are capable to learn various tasks from examples, classify objects and establish relation between them and Genetic algorithm: which serves important search and optimization techniques. Genetic algorithms can be used to improve the performance of Neural Networks and they can be used to decide the connection weights of the inputs. These algorithms can also be used for topology selection and training network.



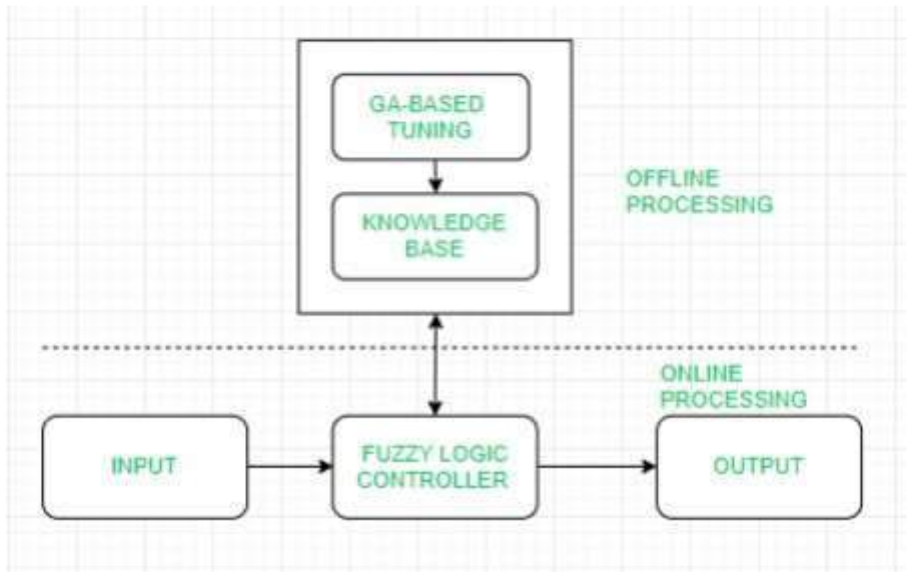
(C) Fuzzy Genetic Hybrid systems:

A Fuzzy Genetic Hybrid System is developed to use fuzzy logic based techniques for improving and modelling Genetic algorithms and viceversa. Genetic algorithm has proved to be a robust and efficient tool to perform tasks like generation of fuzzy rule base, generation of membership function etc. Three approaches that can be used to develop such system are:

•Michigan Approach

•Pittsburgh Approach

• IRL Approach



ANFIS Architecture Usage Example:

Problem Statement: An Adaptive Neuro-Fuzzy Inference System (ANFIS) for delivering adapted learning content to mobile learners.

Solution:

ANFIS Architecture

The Adaptive Neuro-Fuzzy Inference System technique was originally presented by Jang in 1993. ANFIS is a simple data learning technique that uses Fuzzy Logic to transform given inputs into a desired output through highly interconnected Neural Network processing elements and information connections, which are weighted to map the numerical inputs into an output.

ANFIS combines the benefits of the two machine learning techniques (Fuzzy Logic and Neural Network) into a single technique. An ANFIS works by applying Neural Network learning methods to tune the parameters of a Fuzzy Inference System (FIS). There are several features that enable ANFIS to achieve great success:

• It refines fuzzy IF-THEN rules to describe the behavior of a complex system;

• It does not require prior human expertise;

• It is easy to implement;

• It enables fast and accurate learning;

• It offers desired data set; greater choice of membership functions to use; strong generalization abilities; excellent explanation facilities through fuzzy rules; and

• It is easy to incorporate both linguistic and numeric knowledge for problem solving.

Different rules cannot share the same output membership function. The number of membership functions must be equal to the number of rules. To present the ANFIS architecture, two fuzzy IF-THEN rules based on a first order Sugeno model are considered:

Rule(1): IF x is A1 AND y is B1,THENf f1=p1x+q1y+r1. Rule(2): IF x is A2 AND y is B2,THEN f2=p2x+q2y+r2.

where:

• x and y are the inputs,

• A_i and B_i are the fuzzy sets,

• f_i are the outputs within the fuzzy region specified by the fuzzy rule, and

• p_i , q_i , and r_i are the design parameters that are determined during the training process.

Fig. 4 illustrates the reasoning mechanism for this Sugeno model, which is the basis of the ANFIS model.

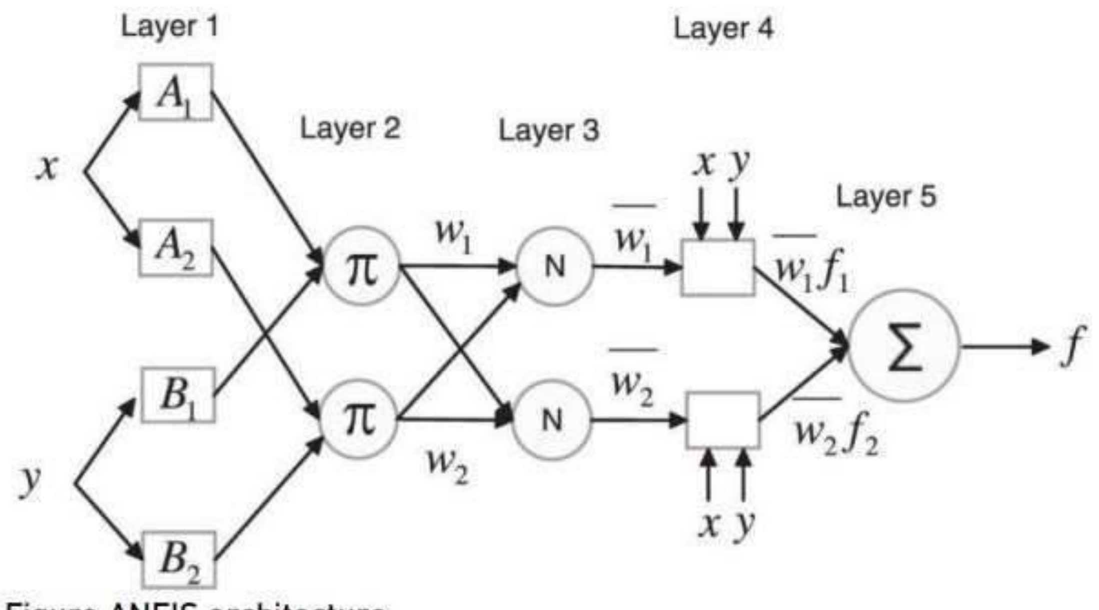


Figure ANFIS architecture.

The ANFIS architecture used to implement these two rules is shown in Fig. 4. In this figure, a circle indicates a fixed node, whereas a square indicates an adaptive node. ANFIS has a five- layer architecture. Each layer is explained in detail below.

In ${\rm Layer}_{(1)}$, all the nodes are adaptive nodes. The outputs of ${\rm Layer}_{(1)}$ are the fuzzy membership grade of the inputs, which are given by the following equations: O1,i=μAi(x),i=1,2,O1,i=μAi(x),i=1,2,

(1)

O1,i=μBi−2(y),i=3,4,O1,i=μBi−2(y),i=3,4,

(2)

where x and y are the inputs to node i , and A_i and B_i are the linguistic labels (high, low, etc.) associated with this node function. $\mu_{Ai}(x)$ and $\mu_{Bi-2}(y)$ can adopt any fuzzy membership function. For example, if the bell-shaped membership function is employed, $\mu_{Ai}(x)$ is given by μAi(x)=11+[(x−ciai)2]bi,i=1,2,μAi(x)=11+[(x−ciai)2]bi,i=1,2,

(3)

or the Gaussian membership function by

μAi(x)=exp[−(x−ciai)2],μAi(x)=exp[−(x−ciai)2],

(4)

where a_i , b_i , and c_i are the parameters of the membership function.

In ${\rm Layer}_{(2)}$, the nodes are fixed nodes. This layer involves fuzzy operators; it uses the AND operator to fuzzify the inputs. They are labeled with π , indicating that they perform as a simple multiplier. The output of this layer can be represented as O2,i=wi=μAi(x)∗μBi(y),i=1,2.O2,i=wi=μAi(x)∗μBi(y),i=1,2.

(5)

These are the so-called firing strengths of the rules.

In ${\rm Layer}_{(3)}$, the nodes are also fixed nodes labeled by N , to indicate that they play a normalization role to the firing strengths from the previous layer. The output of this layer can be represented as

O3,i=¯wi=wiw1+w2,i=1,2.O3

,i= w¯i=wiw1+w2,i=1,2.

(6)

Outputs of this layer are called normalized firing strengths.

In ${\rm Layer}_{(4)}$, the nodes are adaptive. The output of each node in this layer is simply the product of the normalized firing strength and a first order polynomial (for a first order Sugeno model). The output of this layer is given by

O4,i=¯wifi=¯wi(pix+qiy+ri),i=1,2,O4,i=w¯if

i=w¯i(pix+qiy+ri),i=1,2,

(7)

where $\bar{w}$ is the output of ${\rm Layer}_{(3)}$, and p_i , q_i , and r_i are the consequent parameters.

In ${\rm Layer}_{(5)}$, there is only one single fixed node labeled with $\sum$. This node performs the summation of all incoming signals. The overall output of the model is given by

O5,i=∑i¯wfi=∑iwifi∑iwi.O5,i=∑iw¯fi=∑iwifi∑iwi.

(8)

3.2Hybrid Learning Algorithm

The learning algorithm for ANFIS is a hybrid algorithm that is a combination of gradient descent and least squares methods. In the forward pass of the hybrid learning algorithm, node outputs go forward until ${\rm Layer}_{(4)}$ and the consequent parameters are determined by the least squares. In the backward pass, the error signals propagate backward and the premise parameters are updated using gradient descendent. The hybrid learning approach converges much faster by reducing search space dimensions of the original back propagation method [The overall output can be given by f=w1w1+w2f1+w2w1+w2f2,f=w1w1+ w2f1+w2w1+w2f2,

(9) f=¯w(p1x+q1y+r1)+¯w(p2x+q2y+r2),f=w¯(p1x+q1y+r1)+w¯(p2x+q2y+r2),

(10) f=(¯w1x)p1+(¯w1y)q1+(¯w1)r1+(¯w2x)p2+(¯w2y)q2+(¯w2)r2,f=(w1¯ x)p1+(w1¯y)q1+(w1¯)r1+(w2¯x)p2+(w2¯y)q2+(w2¯)r2,

(11) where p_1 , q_1 , r_1 , p_2 , q_2 , and r_2 are the linear consequent parameters.

The least squares method is used to identify the optimal values of these parameters. When the premise parameters are not fixed, the search space becomes larger and the convergence of the training becomes slower.

It should be noted that the ANFIS hybrid algorithm combines two methods, the least squares method and the gradient descent method, to solve the problem of search space. The hybrid algorithm is composed of a forward pass and a backward pass.

The least squares method (forward pass) is used to optimize the consequent parameters. The gradient descent method (backward pass) is used to optimize the premise parameters. The output of the ANFIS is calculated by employing the consequent parameters found in the forward pass. The output error is used to adapt the premise parameters by means of a standard backpropagation algorithm. It has been proven that this hybrid algorithm is highly efficient in training the ANFIS systems.

ANFIS Architecture for Adaptive Mobile Learning

ANFIS is an intelligent Neuro-Fuzzy technique used for the modelling and control of ill- defined and uncertain systems. ANFIS is based on the input/output data pairs of the system under consideration. ANFIS is selected to solve the problem of continuous changes in mobile learning environments, facilitating the delivery of adapted learning content. The proposed ANFIS model can be used for modelling the learner context. The steps required to apply ANFIS to learner modelling are: define input and output values; define fuzzy sets for input values; define fuzzy rules; and create and train the Neural Network.

To implement and test the proposed architecture, a development tool is required. MATLAB Fuzzy Logic Toolbox (FLT) from MathWorks was selected as the development tool. This tool provides an environment to build and evaluate fuzzy systems using a graphical userinterface. It consists of a FIS editor, the rule editor, a membership function editor, the fuzzy inference viewer, and the output surface viewer.

The FIS editor displays general information about a fuzzy inference system. The membership function editor is the tool that displays and edits the membership functions associated with all input and output variables. The rule editor allows the user to construct the rule statements automatically, by clicking on and selecting one item in each input variable box, one item in each output box, and one connection item. The rule viewer allows users to interpret the entire fuzzy inference process at once. The ANFIS editor GUI menu bar can be used to load a FIS training initialization, save the trained FIS, and open a new Sugeno system to interpret the trained FIS model.

(2) What are Expert Systems ? Types of Expert Systems. Explain the architecture of Expert Systems with example.

Ans:

Expert Systems:

An expert system is an AI software that uses knowledge stored in a knowledge base to solve problems that would usually require a human expert thus preserving a human expert’s knowledge in its knowledge base. They can advise users as well as provide explanations to them about how they reached a particular conclusion or advice.

Components of an expert system:

•**Knowledge base:** The knowledge base represents facts and rules. It consists of knowledge in a particular domain as well as rules to solve a problem, procedures and intrinsic data relevant to the domain.

• **Inference engine:** The function of the inference engine is to fetch the relevant knowledge from the knowledge base, interpret it and to find a solution relevant to the user’s problem. The inference engine acquires the rules from its knowledge base and applies them to the known facts to infer new facts. Inference engines can also include an explanation and debugging abilities.

•**Knowledge acquisition and learning module:** The function of this component is to allow the expert system to acquire more and more knowledge from various sources and store it in the knowledge base.

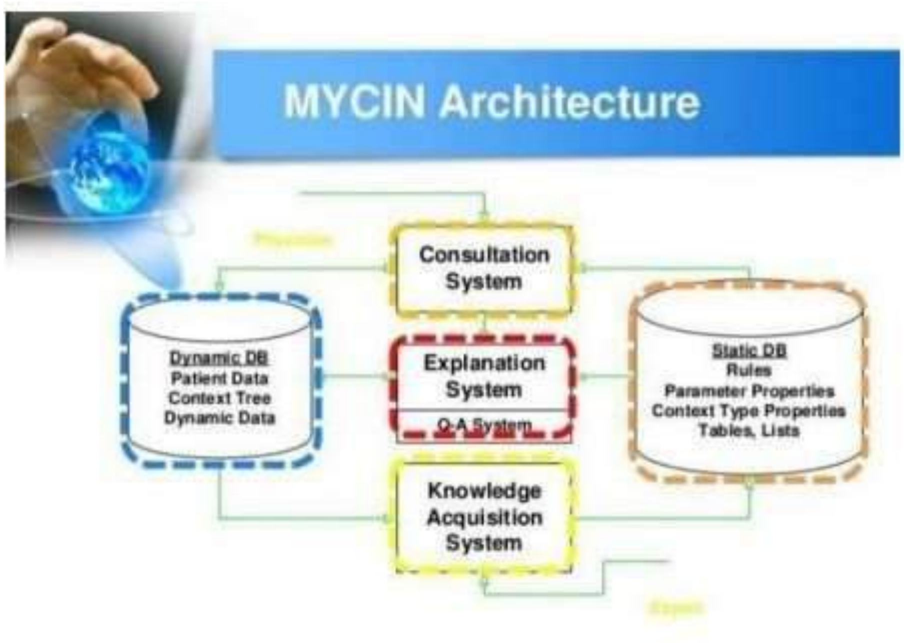
•**User interface:** This module makes it possible for a nonexpert user to interact with the expert system and find a solution to the problem.

•**Explanation module:** This module helps the expert system to give the user an explanation about how the expert system reached a particular conclusion.

Expert System Architecture Usage Example

A popular example of Expert System Architecture is MYCIN

MYCIN was an early that used to identify bacteria causing severe infections, such as and, and to recommend with the dosage adjusted for patient's body weight — the name derived from the antibiotics themselves, as many antibiotics have the suffix "-mycin".



The Structure of the MYCIN System

The MYCIN system comprises three major subprograms, as depicted in Fig. 3. The Consultation Program is the core of the system; it interacts with the physician to obtain information about the patient, generating diagnoses and therapy recommendations. The Explanation Program provides explanations and justifications for the program's actions. The Knowledge-Acquisition Program is used by experts to update the system's knowledge base.

1.Consultation Program:

It performs diagnosis and therapy selection, control structure reads static db and reads/writes to dynamic db, it has terminal interface to physician.

2.Explanation System:

Provides reasoning why a conclusion has been made or question has been asked. Record rule invocation and associate them with questions or rule involved.

3.Knowledge Acquisition System:

Extend static db via dialogue with experts, allows for incremental competence not an all or nothing model, requires minimal training for experts.

1804085

Assignment 1

Q.1). Diff b/w Hard & soft computing

Ans.

	Soft computing	Hard Computing
1)	Soft computing is liberal of Inexaction uncertainity partial & approximation	1) Hard computing needs a exactly state analytic model.
2).	Soft computing relies on formal logic and probab-listic reasoning	2) Hard computing relies on binary logic and crisp system.
3)	Soft computing learn the nature of approxima tion and dispositionality	3). Hard computing has the features exactitude and categoricity.
4)	Soft computing is stochast -ic in nature	4) hard computing is deterministic in nature.
5).	Soft computing works on ambiguous and noisy data	5). Hard computing works on exact data.
6)	Soft computing can perform parallel computation	6). Hard computing perform sequential computation.
7)	Soft computing produces approximate results	7) hard computing produces precise results
8).	Soft computing will emerge its own program	8) Hard computing requires programs to be written.
9)	Soft computing incorporat -es randomness	9) hard computing is settled
10.	Soft computing will use multivalued logic	10) Hard computing uses two valued logic.

Scanned with CamScanner

Q2) What is AI? and its components:

→ According to father of AI John McCarthy it is "the science of engineering of making intelligent machine especially intelligent computer programs"

• Artificial intelligence is a way of making a computer controller unit or a software intelligently in the similar manner the intelligent becomes human

- AI is accomplished by studying how human brain thinks, decides or a software and work while trying to solve problem and then using the outcomes of this study as a basic of developing intelligent user & system

1) Learning

one of the essential component of a in learning for AI includes the trial and error of method the solution keeps on solving problems until it comes across the right result. this way, the program keeps a note of all the moves that gave positive results of before in the database

2) Reasoning

the ability to differentiate makes reasoning one of the essential components of artificial intelligence to reason is to allow help platform to draw inferences that fit with provided situation

Scanned with CamScanner

3> Problem solving
- the ability to differentiate makes ~~the~~ considerable variety of problems being addressed in platform
- the different methods of problem solving cannot be essential a components that divide the queries into special and general purpose.

~~4> PS~~

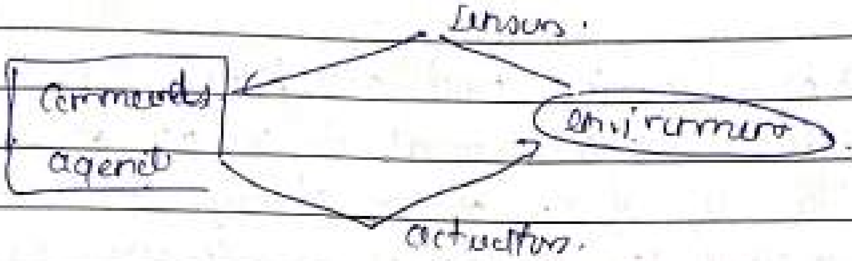
4> Perception
In using the perception components of artificial intelligence the element scans any given environment by using different sense organs, either artificial or real

5> Language understanding
the simpler terms, language can be defined as a set of different system signs that justify their means using convention occurring. In as the widely used artificial intelligence components language understanding uses distinctive types of language.

Q5) What is an agent? define intelligent agent explain structure of intelligent agent
- Artificial intelligent is defined as a study of rational agents. A rational agent could be anything which make decision as a person, firm machine or software.
- It carries out an action with the best outcomes after considering past and current percepts

Scanned with CamScanner

An AI system is composed of an agent and its environment. The agent act in their environment. The environment may contain other agents. An agent is anything that can be perceiving its environment through sensors and acting upon that environment through actuators.



- Intelligent Agent
An intelligent agent is an autonomous entity which acts upon an environment using sensors and actuators for achieving goal. An intelligent agent may learn from the environment to achieve their goals. A thermostat is an example of an intelligent agent.

Structure of intelligent agents

1) Architecture: this refers to machines or devices that consist of actuators and sensors. The intelligent agent execute on this machinery. Eg include a personal computer, a car or a camera.

2) Agent function
This is a function in which actions are mapped from a certain percepts because percept sequence refers to a history and what the intelligent agent has acquired.

Scanned with CamScanner

③ Agent program :- this is an implementation or executio of the agent function. the agent function is produced through the agent programs execution on the physical architecture.

4) What is rational agent ? give eg of rational agent.

→ In machine learning and artificial intelligence research. the rational agent is a concept that guides the use of a game theory and decision theory in applying artificial intelligence to various real world. the rational agent is theoretical entity based on a realistic model that has preferences for advantageous outcomes and will seek to achieve them in a learning scenario.

- Take an eg of some type of commercial artificial intelligence or machine learning project. suppose a business wants to understand how people will use a complex navigational space like a drive forward with the four lanes, or a complex statement layout with multiple tables and charts. the engineer and data scientists will construct profiles and properties. For rational acting which are modelled real life customers they will then run the customers. they will then run the machine learning programs with these rational actors in mind of [illegible] at output.

Scanned with CamScanner

Rational actors can be applied in all sorts of ways to artificial intelligence projects they help people to understand how theoretical humans might use technologies and how the technologies can learn about human behaviour to help other humans make decisions.

Scanned with CamScanner