Autumn End Semester Examination 2023
4th Semester B.Tech.
Computational Intelligence (CS 3013)
Evaluation Scheme

## Section A

1. Answer the following questions.

**(a) What are the constituents of Soft Computing?**

Ans: Soft Computing comprises three main constituents:

**Fuzzy Logic:** A mathematical framework that deals with uncertainty and imprecision, allowing for the representation of vague or ambiguous information.

**Neural Networks:** Computational models inspired by the structure and functioning of the human brain, capable of learning and making decisions from data.

**Evolutionary Algorithms:** Optimization algorithms inspired by the process of natural selection, involving mechanisms such as mutation, crossover, and selection to find optimal solutions to problems.

**(b) Briefly describe the Hebbian learning rules.**

Ans: Hebbian learning is a neurobiologically-inspired learning rule based on the principle "cells that fire together wire together." The basic idea is that when the activity of one neuron consistently triggers the firing of another neuron, the synaptic connection between them is strengthened. The Hebbian learning rule can be stated as follows:

"If neuron A repeatedly and persistently takes part in firing neuron B, then the synapse from A to B will be strengthened."

In mathematical terms, the Hebbian learning rule is often represented as:

$\Delta w = \eta * A * B$

where, $\Delta w$ is the change in synaptic weight, $\eta$ is the learning rate, A is the activity of the presynaptic neuron, B is the activity of the postsynaptic neuron.

**(c) How does the Radial Basis Function Neural Network improve the separability?**

Ans: The Radial Basis Function (RBF) Neural Network improves separability by transforming input data into a higher-dimensional space using radial basis functions. RBF networks use a nonlinear mapping technique to transform input data into a higher-dimensional feature space. The radial basis functions act as basis functions that map input data into this new space. In the higher-dimensional space, the radial basis functions create features that enhance the separability of different classes or categories in the data.

**(d) What is vanishing gradient problem? When does it occur?**

Ans: The vanishing gradient problem refers to diminishing the gradients of the loss function during the training of deep neural networks using gradient-based optimization algorithm like backpropagation. It is specifically occurred when activation functions like sigmoid or hyperbolic tangent are used in multiple layers networks, where the gradients can become very small for very less input values. To mitigate the vanishing gradient problem, alternative activation functions such as ReLU (Rectified Linear Unit) or variants is used in deep neural networks.

**(e) What are the different types of Fuzzy membership function?**

Ans: Triangular, Trapezoidal, Gaussian, Generalized Bell, and Sigmoid are commonly used membership functions.

**(f) Explain the convexity of Fuzzy Sets.**

Ans: The convexity of fuzzy sets refers to the property where the membership values within the set increase monotonically from the boundary towards the core.

A fuzzy set is considered convex if, for any pair of elements within the set, the membership value of linear combination of those elements falls within the same fuzzy set. Mathematically, if $x_1$and $x_2$ are elements in a fuzzy set A with membership degrees $\mu_A(x_1)$ and $\mu_A(x_2)$, then for any $0 \leq \lambda \leq 1$, the convex combination (linear combination) $t = \lambda x_1 + (1 - \lambda)x_2$ belongs to A, and $\mu_A(t) \geq \min(\mu_A(x_1), \mu_A(x_2))$.

**(g) What is max-product composition?**

Ans: Max-product composition is a mathematical operation used in fuzzy set theory to combine two fuzzy relations and derive a new fuzzy relation. Given two fuzzy relations $R_1$ and $R_2$ defined on the cartesian product of two sets, $A \times B$, the max-product composition is expressed as:

$$(R_1 o R_2)(a, c) = max_{b \epsilon B}(\min \{R_1(a, b), R_2(b, c)\}$$

**(h) Briefly describe the concentration and dilation of linguistic values with examples.**

Ans: Concentration and dilation are operations used in fuzzy logic to adjust the spread and intensity of linguistic values, allowing for more flexible and context-specific representation of fuzzy sets.

Concentration: Concentration, in the context of fuzzy logic refers to the process of narrowing or focusing the spread or range of linguistic values associated with a fuzzy set or linguistic variable. It involves making the membership values more intense or concentrated around a specific point or a narrower interval within the universe of discourse.

Dilation: Dilation, on the other hand, involves the process of expanding or spreading the range of linguistic values associated with a fuzzy set or linguistic variable. It aims to make the membership values less intense and distributed over a broader range within the universe of discourse.

If A is a linguistic value then operation concentration is defined by CON(A) = $A^2$, and dilation is defined by DIL(A) = $A^{0.5}$.

If age is a linguistic variable, then its term set is T(age) as follows.

T(age) = {young, not young, very young, not very young, …… middle aged, not middle aged, … old, not old, very old, more or less old, not very old, …not very young and not very old, …}.

For more or less old = DIL(old); extremely old = CON(CON(CON(old))).

**(i) What is the role of defuzzification in a fuzzy rule-based system?**

Ans: Defuzzification is the process of converting fuzzy output values, represented as fuzzy sets or linguistic terms, into a clear or crisp, which is often numerical value. This conversion is necessary in fuzzy logic systems

to obtain a specific and actionable result that can be used for decision-making, control, or other applications.

**(j) What is mutation in genetic algorithm? Why is it important in genetic algorithm?**

Ans: In genetic algorithms, mutation is an operator that introduces small random changes in an individual's genetic information. It plays a vital role in maintaining genetic diversity within the population. The primary purpose of mutation in a genetic algorithm is to prevent premature convergence, enhance exploration of the search space, and introduce new genetic material that might lead to better solutions.

## Section B

**2. (a) What is soft computing? How is it different from Hard computing? Provide an example illustrating when hard computing is typically more suitable than soft computing and vice versa. Justify your answer.**

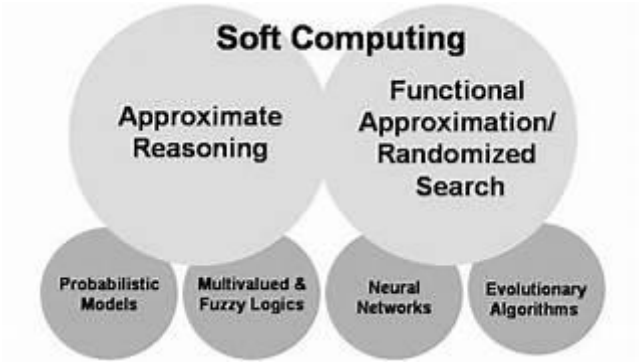
Ans: Soft computing:

Soft computing is a modern computing model that evolved to resolve non-linear problems that involve approximation, uncertainty, and imprecision. It can be associated with being liberal with inexactness, uncertainty, partial truth and approximation.

Soft computing mainly depends on the formal logic and probabilistic reasoning.

It uses techniques such as fuzzy logic, neural networks, genetic algorithms, and other heuristic methods to solve problems.

Soft computing uses multivalued logics and thus has a sophisticated nature. It is mainly used to perform parallel computations. Soft computing produces approximate results.



**Hard Computing:**

Hard computing is a conventional approach used in computing and requires an accurately stated analytical model. Hard computing depends on the binary logic and crisp system.

Hard computing uses two-valued logic. Therefore, it has a deterministic nature. It produces precise and accurate results.

In hard computing, some definite control actions are defined using a mathematical model or algorithm.

The major drawback of hard computing is that it is incapable in solving the real world problems whose behaviour is imprecise and their information being changing continuously. Hard computing is mainly used to perform sequential computations.

**Hard computing vs. Soft computing**

The difference between soft computing and hard computing is that the hard computing is more precise and relies on mathematical models, while the soft computing is more flexible and relies on approximate solutions.

**Example:**

An example illustrating when hard computing is typically more suitable than soft computing is when we need to solve a problem that has a well-defined mathematical solution, such as finding the roots of a quadratic equation. Hard computing can provide an exact and deterministic answer using a simple formula, such as

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Where a, b, c are the coefficients of the equation. Soft computing, on the other hand, may not be able to find the exact roots, but only an approximation, using techniques such as genetic algorithms or neural networks. Moreover, soft computing may require more computational resources and time than hard computing for this problem.

Conter Examples:

An example illustrating when soft computing is typically more suitable than hard computing is when we need to solve a problem that is difficult or impossible to solve exactly, such as image recognition or natural language processing. Soft computing can provide a flexible and adaptable solution using techniques such as fuzzy logic or neural networks, which can handle ambiguity and noise in the data. Soft computing can also learn from the data and improve its performance over time. Hard computing, on the other hand, may not be able to handle the complexity and uncertainty of the problem, and may fail to provide a satisfactory solution. Moreover, hard computing may require a precise mathematical model or algorithm, which may not exist or be very difficult to formulate for this problem.

**b) Explain with a proper diagram how XOR classification problem can be solved in higher dimensions. Solve XOR classification by using RBF network with four radial basis functions.**

The XOR classification problem is a classic problem in the field of artificial neural networks, where the goal is to learn a function that can correctly classify four points in a two-dimensional space, such that two points have the same class (0 or 1) if and only if they have the same coordinates (0 or 1).

| Inputs | | Output |
|---|---|---|
| A | B | X |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

The problem is that these four points are not linearly separable, meaning that there is no straight line that can separate them into two classes. Therefore, a single-layer perceptron, which is a neural network that uses a linear activation function, cannot solve this problem.

One way to solve the XOR problem is to map the input points to a higher-dimensional space, where they become linearly separable. This can be done by using a nonlinear activation function, such as a radial basis function (RBF), in the hidden layer of a neural network.

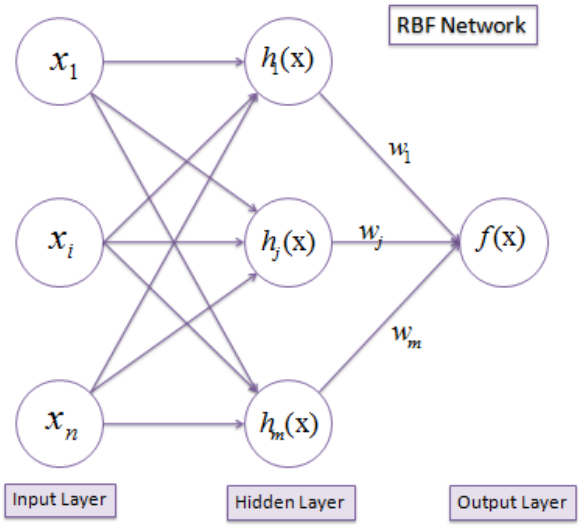
A radial basis function is a function that depends only on the distance from a center point, and has a bell-shaped curve. An example of a radial basis function is the Gaussian function, which is defined as

$$\phi(r) = \exp(-r^2/2\sigma^2)$$

where $\sigma > 0$.

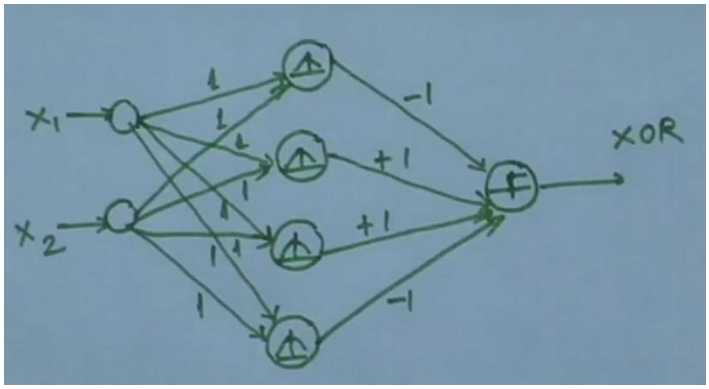
The architecture of an RBFN is shown in the following diagram:

To solve the XOR problem using an RBFN, we need to choose the number of hidden neurons, the centers and widths of the radial basis functions, and the weights of the output neuron.



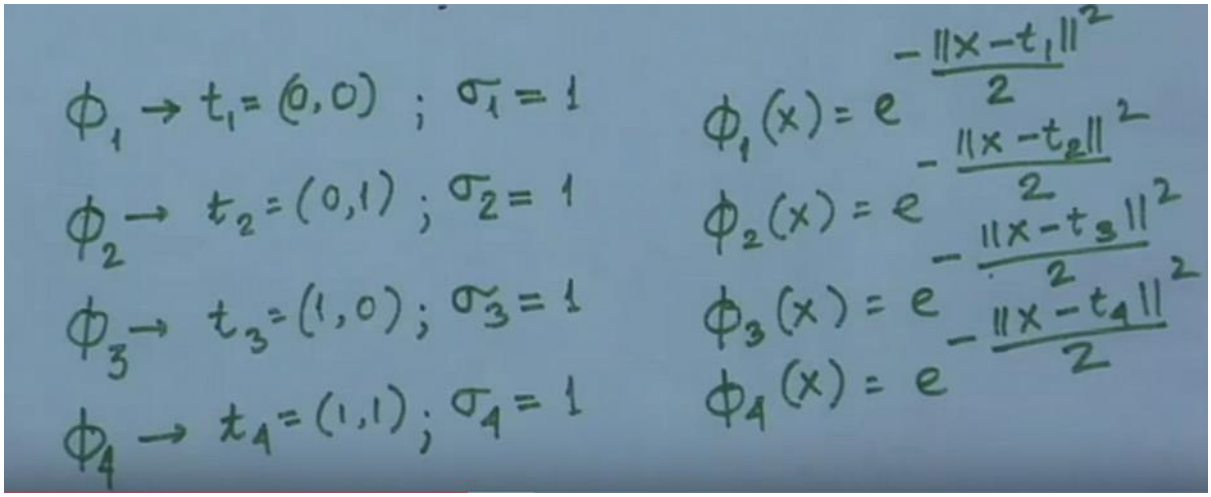

$$f(x) = \sum_{j=1}^{m} w_j h_j(x)$$

$$h(x) = \exp\left(-\frac{(x-c)^2}{r^2}\right)$$




(Architecture of XOR RBNN)

To solve the XOR problem using an RBFN, we need to choose the number of hidden neurons, the centers and widths of the radial basis functions, and the weights of the output neuron.

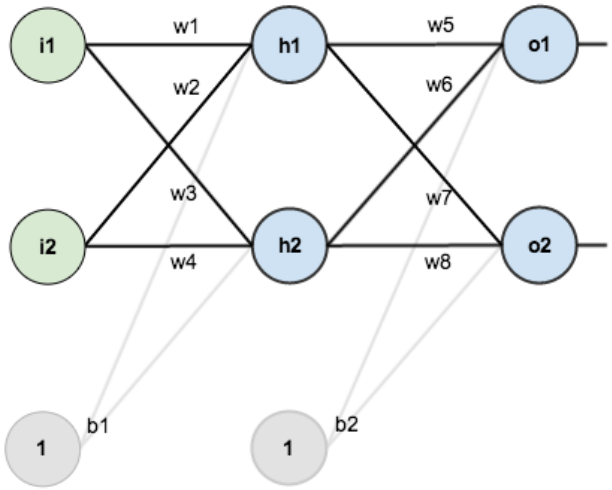



(Transformation function with receptors and variances.)

| Input | $\phi_1$ | $\phi_2$ | $\phi_3$ | $\phi_4$ | $\sum W_i \phi_i$ | Output |
|---|---|---|---|---|---|---|
| 0 0 | 1.0 | 0.6 | 0.6 | 0.4 | −0.2 | 0 |
| 0 1 | 0.6 | 1.0 | 0.4 | 0.6 | 0.2 | 1 |
| 1 0 | 0.6 | 0.4 | 1.0 | 0.6 | 0.2 | 1 |
| 1 1 | 0.4 | 0.6 | 0.6 | 1.0 | −0.2 | 0 |
| | −1 | +1 | +1 | −1 | | |

(linear combination of transformation function is tabulated.)

3. (a)




**Derive equations suitable to update the weights w2 and w6 in the above MLP model as per the back-propagation algorithm by using chain rule. Assume sigmoid activation function at the hidden and output layers.**

Ans: The goal of backpropagation is to optimize the weights so that the neural network can learn how to correctly map arbitrary inputs to outputs. Steps for updating the weights w1 and w5 are shown below. Similar process can be followed to update the weights of w2 and w6.

The Forward Pass

Here's how we calculate the total net input for:

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

We then squash it using the logistic function to get the output of $h_1$:

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}}$$

Carrying out the same process for

We repeat this process for the output layer neurons, using the output from the hidden layer neurons as inputs.

Here's the output for:

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

Calculating the Total Error

We can now calculate the error for each output neuron using the squared error function and sum them to get the total error:

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

The ½ is included so that exponent is cancelled when we differentiate later. The result is eventually multiplied by a learning rate anyway, so it doesn't matter that we introduce a constant here.

The Backwards Pass

Our goal with backpropagation is to update each of the weights in the network so that they cause the actual output to be closer the target output, thereby minimizing the error for each output neuron and the network as a whole.

Output Layer

By applying the chain rule we know that:

(b) Explain briefly how threshold logic works in a McCulloch Pitts neuron? Implement following boolean function using MP neuron

- AND
- OR
- NOR

- NOT

Threshold logic is a way of implementing boolean functions using McCulloch Pitts neurons.

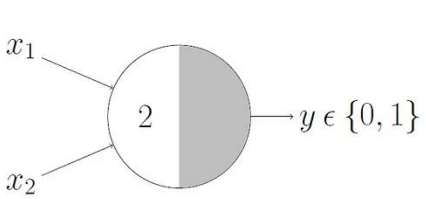
A McCulloch Pitts neuron is a simple model of an artificial neuron that has binary inputs and outputs, and a threshold value that determines its activation.

The neuron computes the weighted sum of its inputs, and compares it with the threshold. If the sum is greater than or equal to the threshold, the neuron outputs 1 (true); otherwise, it outputs 0 (false).

The weights and the threshold can be chosen to represent different boolean functions, such as AND, OR, NOR, and NOT.

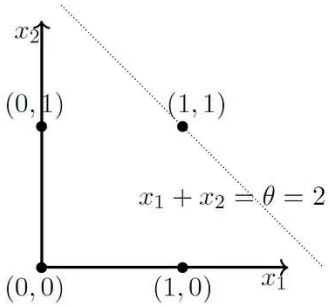
To implement the following boolean functions using MP neurons, we can use the following formulas:

- AND: The output is 1 if and only if all the inputs are 1. We can use a weight of 1 for each input, and a threshold of n, where n is the number of inputs. For example, for two inputs x and y, the output is

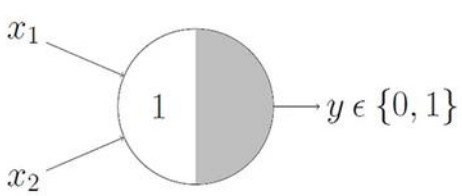



$AND\ function$

$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 2$$

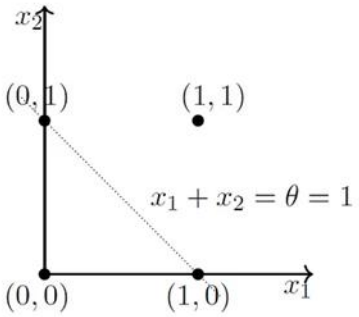



- OR: The output is 1 if at least one of the inputs is 1. We can use a weight of 1 for each input, and a threshold of 1. For example, for two inputs x and y, the output is




*OR function*

$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 1$$




- NOR: The output is 1 if and only if all the inputs are 0. We can use a weight of -1 for each input, and a threshold of -n + 1, where n is the number of inputs. For example, for two inputs x and y, the output is

$$z = \begin{cases} 1 & \text{if } -x - y \geq -1 \\ 0 & \text{otherwise} \end{cases}$$

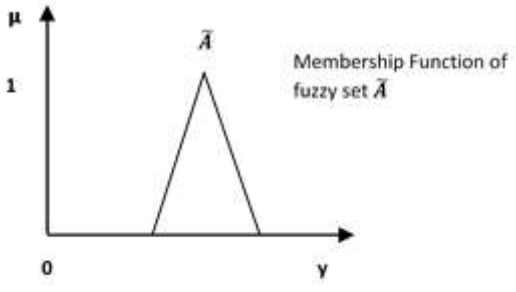
- NOT: The output is 1 if the input is 0, and 0 if the input is 1. We can use a weight of -1 for the input, and a threshold of 0. For example, for one input x, the output is

$$z = \begin{cases} 1 & \text{if } -x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

## Section C

**4. A)**

**Ans:** We already know that fuzzy logic is not logic that is fuzzy but logic that is used to describe fuzziness. This fuzziness is best characterized by its membership function. In other words, we can say that membership function represents the degree of truth in fuzzy logic.




Membership Function

Following are a few important points relating to the membership function −

Membership functions were first introduced in 1965 by Lofti A. Zadeh in his first research paper “fuzzy sets”. Membership functions characterize fuzziness (i.e., all the information in fuzzy set), whether the elements in fuzzy sets are discrete or continuous. Membership functions can be defined as a technique to solve practical problems by experience rather than knowledge. Membership functions are represented by graphical forms. Rules for defining fuzziness are fuzzy too.

Features of Membership Functions

We will now discuss the different features of Membership Functions.

## Core

For any fuzzy set $\widetilde{A}$, the core of a membership function is that region of universe that is characterize by full membership in the set. Hence, core consists of all those elements $y$ of the universe of information such that,

$$\mu_{\widetilde{A}}(y) = 1$$

## Support

For any fuzzy set $\widetilde{A}$, the support of a membership function is the region of universe that is characterize by a nonzero membership in the set. Hence core consists of all those elements $y$ of the universe of information such that,

$$\mu_{\widetilde{A}}(y) > 0$$

## Alpha cut

Let $\mu \in F(X)$ and $\alpha \in [0, 1]$.

Then the sets $[\mu]\alpha = \{x \in X \mid \mu(x) \geq \alpha\}$ , $[\mu]\alpha = \{x \in X \mid \mu(x) > \alpha\}$ are called the α-cut and strict α-cut of μ

## Singleton

In the fuzzy theory, fuzzy set A of universe X is defined by function μA(x) called the membership function of set A. We already discussed this point. μA(x): X → [0, 1], where μA(x) = 1 if x is totally in A; μA(x) = 0 if x is not in A; 0 < μA(x) < 1 if x is partly in A.

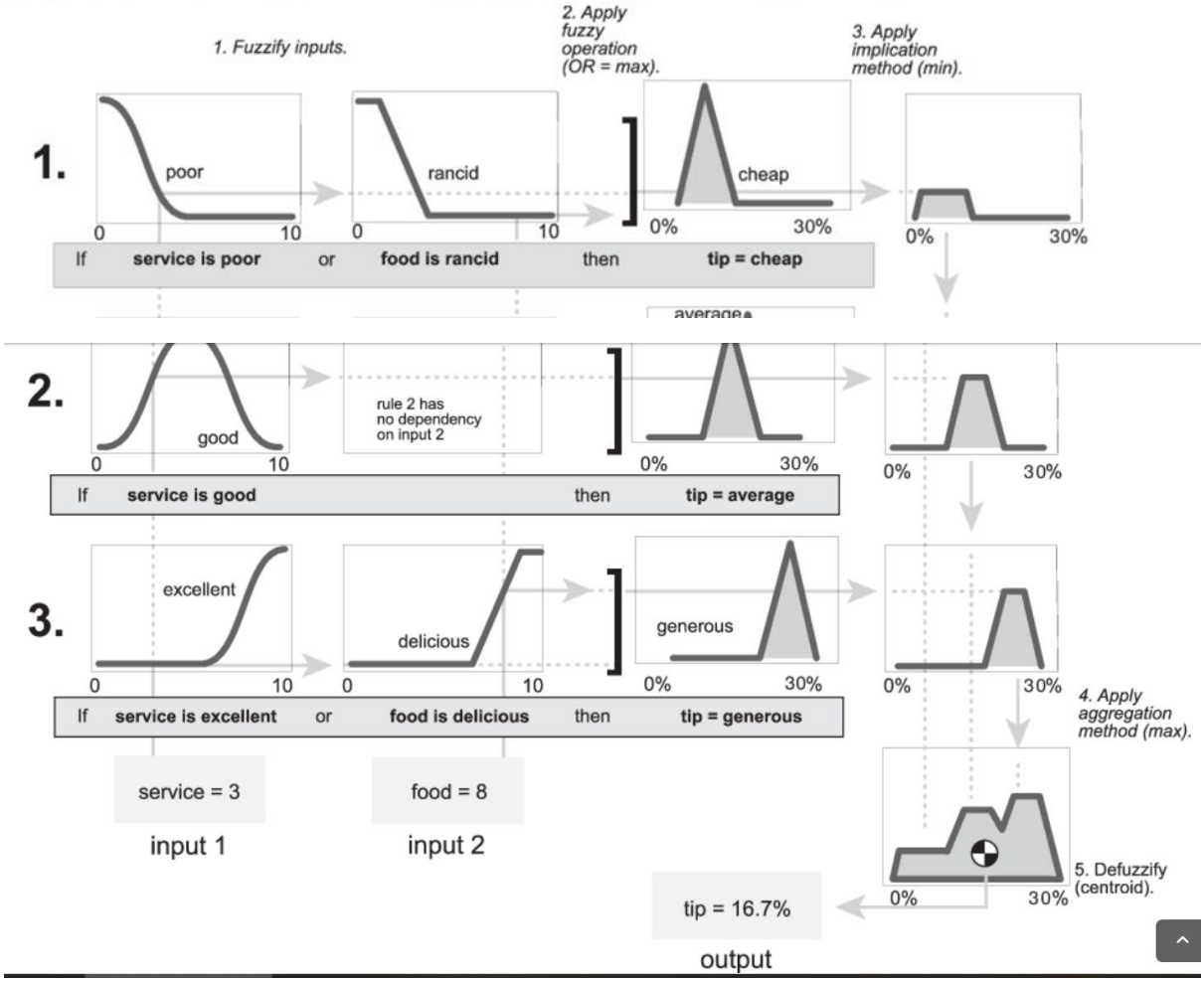
## B)

### Mamdani Fuzzy Inference Systems

Mamdani fuzzy inference was first introduced as a method to create a control system by synthesizing a set of linguistic control rules obtained from experienced human operators [1]. In a Mamdani system, the output of each rule is a fuzzy set.

Since Mamdani systems have more intuitive and easier to understand rule bases, they are well-suited to expert system applications where the rules are created from human expert knowledge, such as medical diagnostics.

The inference process of a Mamdani system is described in Fuzzy Inference Process and summarized in the following figure.



**The output of each rule is a fuzzy set derived from the output membership function and the implication method of the FIS. These output fuzzy sets are combined into a single fuzzy set using the aggregation method of the FIS. Then, to compute a final crisp output value, the combined output fuzzy set is defuzzified using one of the methods described in Defuzzification Methods.**

## 5. A)

**Intersection.** A fuzzy set $A \cap B$ is said to be the intersection of the fuzzy sets $A$ and $B$ if

$$\mu_{A\cap B}x = \min \mu_A x, \mu_B x = \mu_A x \wedge \mu_B x, \forall x \in X. \quad (2$$

**Union.** A fuzzy set $A \cup B$ is said to be union of fuzzy sets $A$ and $B$ if

$$\mu_{A\cup B}x = \max \mu_A x, \mu_B x = \mu_A(x) \vee \mu_B(x), \forall x \in X. \quad (3$$

**Complement.** A fuzzy set $\bar{A}$ is said to be complement of a fuzzy set $A$ if

$$\mu_{\bar{A}} = 1 - \mu_A(x), \forall x \in X. \quad (4$$

The intersection of $A$ and $B$ corresponds to the connective "***and***" Thus $A \cap B = A$ ***and*** $B$.
The union of fuzzy sets $A$ and $B$ corresponds to the connective "***or***." Thus $A \cup B = A$***or***$B$. The operation of complementation corresponds to negation ***not***. Thus $\bar{A}$=***not*** $A$.

B)

**Fuzzy composition** can be defined just as it is for crisp (binary) relations. Suppose $\underline{R}$ is a fuzzy relation on $X \times Y$, $\underline{S}$ is a fuzzy relation on $Y \times Z$, and $\underline{T}$ is a fuzzy relation on $X \times Z$; then,

**Fuzzy Max–Min composition** is defined as:

$$\underline{T} = \underline{R} \circ \underline{S} = \mu_{\bar{T}}(x,z) = \bigvee_{y \in Y} (\mu_{\bar{R}}(x,y) \wedge \mu_{\bar{S}}(y,z))$$

$$= max_{y \in Y}\{min(\mu_{\bar{R}}(x,y), \mu_{\bar{S}}(y,z))\}$$

**6. A)**

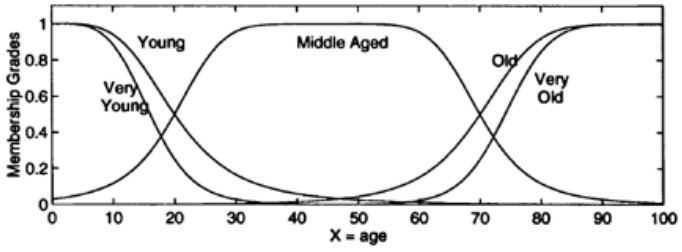
# Linguistic Variables

**Example 3.5** *Linguistic variables and linguistic values*

If *age* is interpreted as a linguistic variable, then its term set $T(age)$ could be

$$T(age) = \{ \textit{young, not young, very young, not very young, ...,}$$
$$\textit{middle aged, not middle aged, ...,}$$
$$\textit{old, not old, very old, more or less old, not very old, ...,}$$
$$\textit{not very young and not very old, ...} \},$$

where each term in $T(age)$ is characterized by a fuzzy set of a universe of discourse $X = [0, 100]$, as shown in Figure




*Typical membership functions of the term set $T(age)$.*

# Fuzzy If Then Rules

A **fuzzy if-then rule** (also known as **fuzzy rule**, **fuzzy implication**, or **fuzzy conditional statement**) assumes the form

$$\text{if } x \text{ is } A \text{ then } y \text{ is } B,$$

where $A$ and $B$ are linguistic values defined by fuzzy sets on universes of discourse $X$ and $Y$, respectively. Often "$x$ is $A$" is called the **antecedent** or **premise**, while "$y$ is $B$" is called the **consequence** or **conclusion**. Examples of fuzzy if-then rules are widespread in our daily linguistic expressions, such as the following:

- If pressure is high, then volume is small.
- If the road is slippery, then driving is dangerous.
- If a tomato is red, then it is ripe.
- If the speed is high, then apply the brake a little.

# Fuzzy If Then Rules

A **fuzzy if-then rule** (also known as **fuzzy rule**, **fuzzy implication**, or **fuzzy conditional statement**) assumes the form

$$\text{if } x \text{ is } A \text{ then } y \text{ is } B,$$

where $A$ and $B$ are linguistic values defined by fuzzy sets on universes of discourse $X$ and $Y$, respectively. Often "$x$ is $A$" is called the **antecedent** or **premise**, while "$y$ is $B$" is called the **consequence** or **conclusion**. Examples of fuzzy if-then rules are widespread in our daily linguistic expressions, such as the following:

- If pressure is high, then volume is small.
- If the road is slippery, then driving is dangerous.
- If a tomato is red, then it is ripe.
- If the speed is high, then apply the brake a little.

between two variables $x$ and $y$; this suggests that a fuzzy if-then rule be defined as a binary fuzzy relation $R$ on the product space $X \times Y$. Generally speaking, there are two ways to interpret the fuzzy rule $A \to B$. If we interpret $A \to B$ as $A$ **coupled with** $B$, then

$$R = A \to B = A \times B = \int_{X \times Y} \mu_A(x) \tilde{*} \mu_B(y)/(x, y),$$

where $\tilde{*}$ is a T-norm operator and $A \to B$ is used again to represent the fuzzy relation $R$. On the other hand, if $A \to B$ is interpreted as $A$ **entails** $B$, then it can be written as four different formulas:

- Material implication:
$$R = A \to B = \neg A \cup B. \tag{3.19}$$

- Propositional calculus:
$$R = A \to B = \neg A \cup (A \cap B). \tag{3.20}$$

- Extended propositional calculus:
$$R = A \to B = (\neg A \cap \neg B) \cup B. \tag{3.21}$$

- Generalization of modus ponens:
$$\mu_R(x, y) = \sup\{c \mid \mu_A(x) \tilde{*} c \leq \mu_B(y) \text{ and } 0 \leq c \leq 1\}, \tag{3.22}$$
where $R = A \to B$ and $\tilde{*}$ is a T-norm operator.

B)

An attempt has been made by Hui et al. [132] to model Mamdani Approach of FLC using the structure of a feed-forward NN. Fig. 11.1 shows the schematic diagram of a Neuro-Fuzzy System (NFS). It consists of five layers, namely Layer 1 (input layer), Layer 2 (fuzzifica-

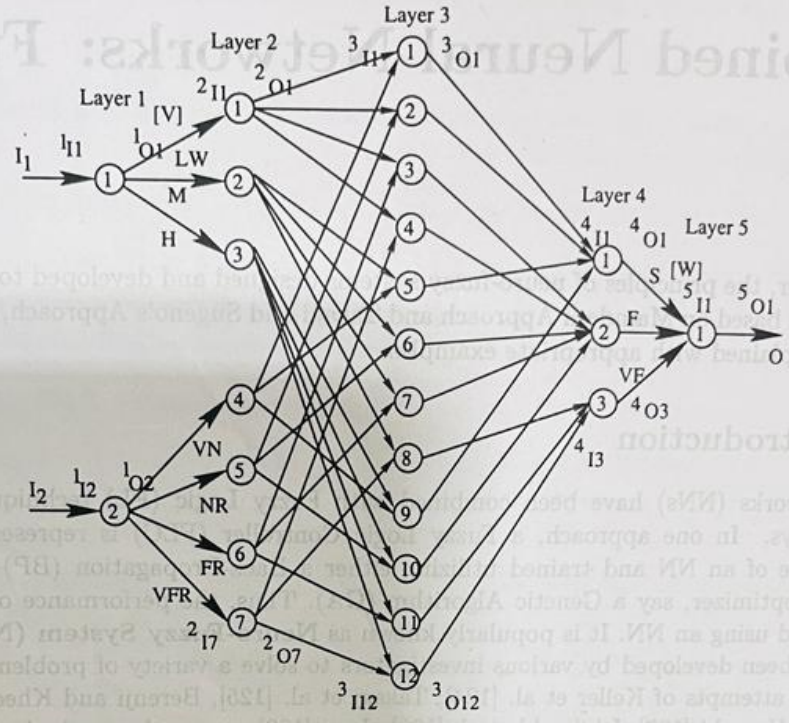
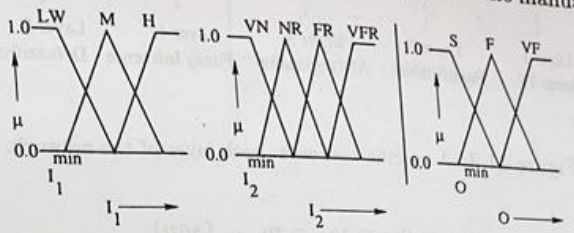



Figure 11.1: A schematic diagram of a neuro-fuzzy system – Mamdani Approach.

tion layer), Layer 3 (AND operation implementing layer), Layer 4 (fuzzy inference layer) and Layer 5 (defuzzification layer). The neurons of first layer are assumed to have linear transfer function. Thus, the outputs of the neurons lying on this layer are nothing but their corresponding input values. The second layer performs fuzzification through which the membership values of input variables are determined. The third layer represents all possible combinations of the input variables (also known as antecedents) and performs logical AND operation. The fourth layer identifies the fired rules corresponding to the set of input variables. The fifth layer performs defuzzification to convert the fuzzified output into its corresponding crisp value.

Let us assume that there are two inputs, such as $I_1$ and $I_2$ and one output $O$ of the process to be modeled using an FLC. Let us also consider that $I_1$, $I_2$ and $O$ are expressed

utilizing three (LW: Low, M: Medium, H: High), four (VN: Very Near, NR: Near, FR: Far, VFR: Very Far) and three (S: Slow, F: Fast, VF: Very Fast) linguistic terms, respectively. The membership function distributions of the inputs and output variables are assumed to be triangular in nature as shown in Fig. 11.2. Table 11.1 shows the manually designed Rule




LW: Low
M: Medium
H: High
VN: Very Near
NR: Near
FR: Far
VFR: Very Far
S: Slow
F: Fast
VF: Very Fast

Figure 11.2: Membership function distributions of the input and output variables.

Base (RB) (consisting of $3 \times 4 = 12$ rules) of the FLC. A particular rule (say the first rule

Table 11.1: Rule base of the fuzzy logic controller

| | | $I_2$ | | | |
|---|---|---|---|---|---|
| | | VN | NR | FR | VFR |
| $I_1$ | LW | S | S | F | F |
| | M | S | F | F | VF |
| | H | S | F | VF | VF |

of the above table) will look as follows:

**If** $I_1$ is $LW$ **AND** $I_2$ is $VN$ **THEN** $O$ is $S$.

To explain the principle of this NFS, let us consider $j$-th ($j = 1,2$), $k$-th ($k = 1,2,\ldots,7$), $l$-th ($l = 1,2,\ldots,12$), $m$-th ($m = 1,2,3$) and $n$-th ($n = 1$) neurons of the first, second, third, fourth and fifth layers, respectively (refer to Fig. 11.3). The following notations are used: $1_{Ij}$ and $1_{Oj}$ represent the input and output, respectively, of $j$-th neuron lying on the first layer; $v_{jk}$ indicates the connecting weight between $j$-th neuron of first layer and $k$-th neuron of second layer; $[W]$ denotes the connecting weight matrix between the neurons of fourth and fifth layers. The following assumptions are made to simplify the analysis (refer to Fig. 11.1):

$$v_{1,\,av} = \frac{v_{11}+v_{12}+v_{13}}{3},$$

7. (a) How does the neighborhood function influence the weight update process in a Kohonen Self-Organizing Map? [2 Marks]

Ans: In a Kohonen Self-Organizing Map (SOM), the neighborhood function plays a crucial role in determining how the weights of neurons are updated during the learning process. The SOM learning algorithm involves adjusting the weights of

neurons based on the input data and their spatial relationships within the map. The neighborhood function defines the influence of neighboring neurons on the weight update process. This neighborhood function is a monotonically decreasing function.

The weight update formula for a neuron i in a SOM is typically expressed as:

$$\Delta Wij = \alpha \cdot hij(t) \cdot (Xj - Wij)$$

Where

ΔWij is the change in weight for the connection between neuron i and the input feature j.

α is the learning rate, controlling the size of the weight adjustments

hij(t) is the neighborhood function, determining the influence of the neighboring neurons.

Xj is the input feature.

Wij is the weight of the connection between neuron i and input feature j.

The neighborhood function hij(t) defines the spatial influence of neuron i on its neighbors at a given time t. The function is usually defined as a Gaussian function centered around the winning neuron. The influence decreases with distance from the winning neuron, reflecting the topological organization of the map. The neighborhood function hij(t) is defined as

$$hij\,(t) = \exp(-dij2/2\sigma(t)2)$$

Where

dij is the Euclidean distance between the winning neuron and neuron i

σ(t) is the neighborhood width parameter, which typically decreases over time during the training process.

Question : Explain the competitive, cooperative, and adaptive process in the Kohonen Self-Organizing Map. [2 Marks]

Answer : The essential processes in the formation of self-organizing maps are Competition, Co-operation and Synaptic Adaptation. Details of each of the processes is given below.

Competition : For each input pattern, the neurons compute their respective values of a discriminant function which provides the basis for competition. The particular neuron with the smallest value of the discriminant function is declared the winner. If the input space is D dimensional (i.e. there are D input units) we can write the input patterns as x = {xi : i = 1, …, D} and the connection weights between the input units i and the neurons j in the computation layer can be written wj = {wji : j = 1, …, N; i = 1, …, D} where N is the total number of neurons. We can then define our discriminant function to be the squared Euclidean distance between the input vector x and the weight vector wj for each neuron j

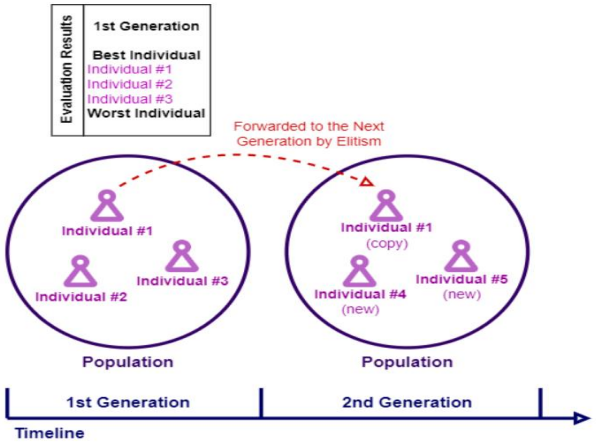
In other words, the neuron whose weight vector comes closest to the input vector (i.e. is most similar to it) is declared the winner.

Co-operation : The winning neuron determines the spatial location of a topological neighborhood of excited neurons, thereby providing the basis for cooperation among neighboring neurons. In neurobiological studies we find that there is lateral interaction within a set of excited neurons. When one neuron fires, its closest neighbors tend to get excited more than those further away. There is a topological neighborhood that decays with distance.

Synaptic Adaptation : Enables the excited neurons to increase their individual values of discriminant function in relation to the input pattern. Since we are adapting the weights, a learning process is associated with this process. The learning mechanism is called the Hebbian learning process. Moreover we have to use a modified Hebbian learning process by introducing the forgetting mechanism since in the Hebbian learning process the weights are continuously increasing leading to over learning.

7. (b) Question : Describe the concept of elitism in genetic algorithms and its purpose in maintaining the best solutions from one generation to the next. [2 Marks]

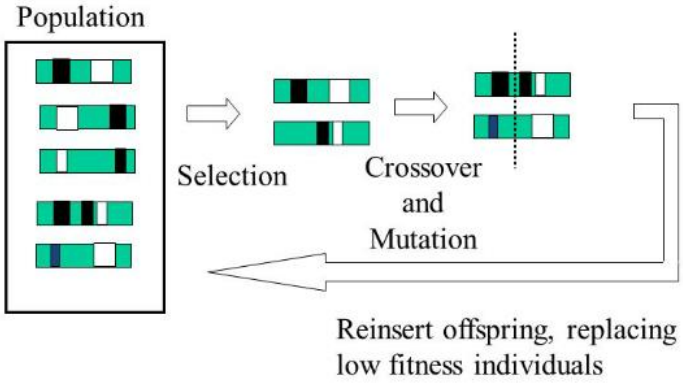
Answer :




Elitism is used in genetic algorithms to preserve some of the best solutions in each generation, allowing them to carry over to the next generation. This helps in maintaining the quality of solutions and prevents the algorithm from converging too quickly to a suboptimal solution. Elitism ensures that the best solutions found so far are not lost and provides a mechanism for exploration and exploitation of the search space. It helps in improving the convergence speed and the diversity of solutions in the population.

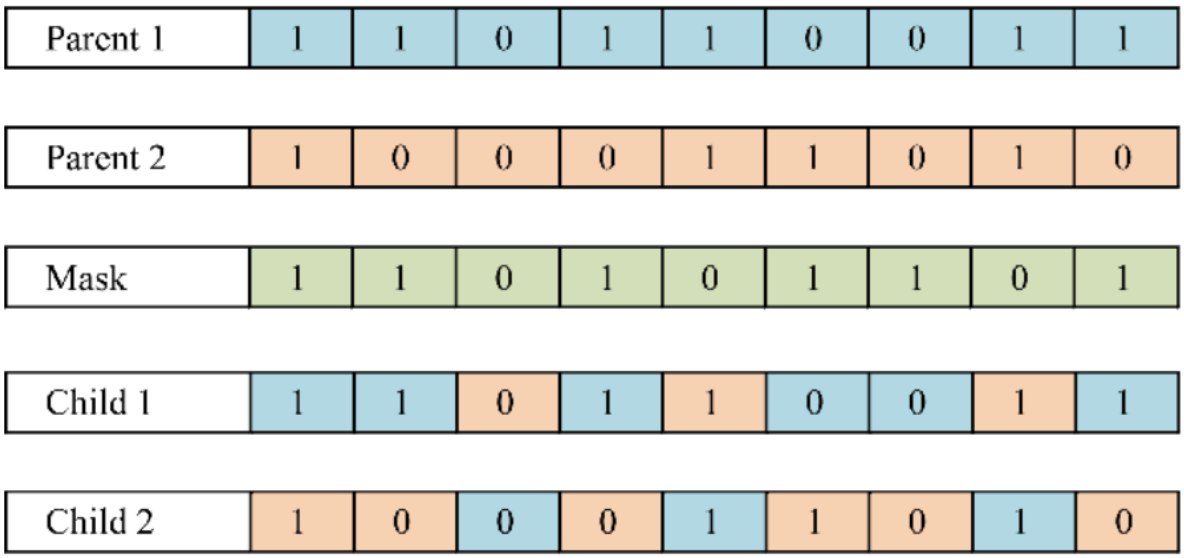
7. (b) Explain the Steady State Selection technique in Genetic Algorithm.

Ans :




The main idea of the steady state selection technique is that the bigger part of the chromosome should survive to the next generation. Here, GA works in the following way. In every generation, a few good (individuals with high fitness) chromosomes are selected for creating new offspring. Then some bad chromosomes with low fitness are removed and new offsprings are placed in that place. The rest of the population survives a new generation.

8. (a) Question : Explain the Uniform crossover with mask. [ 2 Marks]




Ans: In a uniform crossover with a mask, the gene in the offspring is created by copying the corresponding gene from one or the other parent chosen according to a randomly generated crossover mask. When there is 1 in the mask, the gene is copied from the first parent, and when there is 0, the gene is copied from the second parent. The process is repeated with the parents exchanged to produce the second offspring.

**What would happen if we choose parents deterministically in uniform crossover?** [ 2 Marks]

Ans: In the context of genetic algorithms, the uniform crossover is a type of crossover operator that creates offspring by inheriting genetic material from both parents. The choice of each bit of genetic material from either parent is determined with a probability of 0.5. In uniform crossover, if we choose parents deterministically, it means that the selection of genetic material from each parent is fixed and does not change over time. This has several implications. It does not introduce variability in the crossover process across generations. This lack of variability can limit the exploration of the solution space, potentially leading to premature convergence to a suboptimal solution. Without the stochasticity introduced by a random selection of genetic material from parents, the algorithm might have a higher probability of converging to local optima. The lack of diversity in the crossover process can hinder the algorithm's ability to escape local optima and explore the full solution space. Genetic diversity is crucial in genetic algorithms to maintain a diverse population. Deterministic uniform crossover reduces the diversity introduced during crossover, which can result in a population with limited genetic variation. The deterministic nature of

uniform crossover means that the initial population's characteristics heavily influence the subsequent generations.

**8. (b) Compare and contrast between Simulated Annealing and Genetic Algorithm. [3 Marks]**

Ans: Genetic Algorithms (GA) and Simulated Annealing (SA) are both optimization techniques used in solving complex problems, but they differ in their approaches and underlying principles. Bellow is the comparison and contrast between Genetic Algorithms (GA) and Simulated Annealing (SA):

| Attribute | Genetic Algorithm | Simulated Annealing |
|---|---|---|
| **Inspiration** | GAs are inspired by the process of natural selection and genetics. They involve the evolution of a population of candidate solutions using genetic operators such as selection, crossover, and mutation. | SA is inspired by the annealing process in metallurgy. It mimics the annealing of materials, where a material is heated and then slowly cooled to remove defects and optimize its structure. |
| **Representation of Solutions** | Solutions are represented as chromosomes, typically in the form of binary strings, and undergo genetic operations like crossover and mutation. | Solutions are represented in a state space, and neighboring states are explored by making small changes to the current solution. |
| **Search Strategy** | Use a population-based approach. Multiple solutions (individuals) coexist in the population, and the algorithm evolves this population over generations. | Operates with a single solution at a time, exploring the solution space by iteratively accepting probabilistically worse solutions. |
| **Exploration vs Exploitation** | Emphasize exploration. The genetic operators provide a mechanism to explore a wide range of the solution space. | Balances exploration and exploitation by accepting worse solutions with a decreasing probability, allowing the algorithm to escape local optima. |
| **Parameter Sensitivity** | Sensitivity to parameter settings, such as population size, crossover rate, and mutation rate | Sensitivity to the initial temperature and cooling schedule |
| **Application** | Well-suited for combinatorial optimization problems, where the search space is discrete. | Effective for continuous optimization problems with a smooth and continuous solution space. |

**Which one is better at hill climbing? Justify your answer. [1 Marks]**

Ans: Among Genetic Algorithms (GAs) and simulated Annealing (SA), simulated Annealing is more well-suited for hill climbing problems. In fact, simulated-annealing is believed to be a modification or an advanced version of hill-climbing methods. This is due to certain similarity between hill climbing and simulated-annealing technique. For example, SA typically operates with a single solution at a time and explores the neighboring solutions and moves towards the one with better fitness. Moreover, SA's key feature is its ability to accept worse solutions probabilistically. This is crucial for hill climbing, as it allows the algorithm to overcome local optima by occasionally accepting moves that lead to worse solutions.