

Master of Computer Science

MCS-117(N)

Soft Computing

Block-1	ARTIFICIAL INTELLIGENCE & SOFT COMPUTING	5-47
UNIT-1	Introduction of Artificial Intelligence	5-18
UNIT-2	knowledge representation in artificial intelligence	19-35
UNIT-3	Introduction to reasoning and soft computing	36-47
Block-2	FUZZY SET THEORY	51-91
UNIT-1	Introduction to Fuzzy Sets	51-61
UNIT-2	Fuzzy Logic	62 -78
UNIT-3	Fuzzy System	79-91
Block-3	NEURAL NETWORK	95-137
UNIT-1	Introduction of neural networks	95-104
UNIT-2	ANN and Perceptron Model	105-125
UNIT-3	Feedforward neural networks	126-137

Course Design Committee

Prof. Ashutosh Gupta Director, School of Science, UPRTOU Allahabad	Chairman
Prof. Suneeta Agarwal Dept. of Computer Science & Engineering Motilal Nehru National Institute of Technology Allahabad	Member
Dr. Upendra Nath Tripathi Associate Professor DeenDayalUpadhyay Gorakhpur University, Gorakhpur	Member
Dr. Ashish Khare Associate Professor Dept. of Computer Science, University of Allahabad, Prayagraj	Member
Ms. Marisha Assistant Professor (Computer Science) School of Science, UPRTOU Allahabad	Member
Mr. Manoj Kumar Balwant Assistant Professor (Computer Science) School of Science, UPRTOU Allahabad	Member

Course Preparation Committee

Mr. Shivendu Mishra Assistant Professor Dept. of Information Technology Rajkiya Engineering College Ambedkar Nagar U.P. India-224122	Author (Block 1, Block 2: Unit 2, 3, Block 3, & 4)
Dr. Gunjan Singh Assistant Professor (Block 2: Unit 1) Faculty of management and computer application, RBS College, Khandari, Agra-282002.	Author
Prof. Manu Pratap Singh Professor, Department of Computer Science Engineering Dr. Bhimrao Ambedkar University, Agra	Editor
Mr. Manoj Kumar Balwant Assistant Professor (computer science), School of Sciences, UPRTOU Allahabad	Coordinator

© UPRTOU, Prayagraj. 2023

First Edition: July 2023

ISBN: 978-93-48270-40-5

All rights are reserved. No part of this work may be reproduced in any form, by mimeograph or any other means, without permission in writing from the Uttar Pradesh Rajarshi Tandon Open University.



<http://creativecommons.org/licenses/by-sa/4.0/>

Creative Commons Attribution-Share Alike 4.0 International License

Printed and Published by Col. Vinay Kumar, Registrar, Uttar Pradesh Rajarshi Tandon Open University, 2024.

Printed By: Cygnus Information Solution Pvt. Ltd, Lodha Supremus Saki Vihar Road
Andheri East, Mumbai.



MCS-117(N)

Soft Computing

Block-1	ARTIFICIAL INTELLIGENCE & SOFT COMPUTING	5-47
UNIT-1	Introduction of Artificial Intelligence	5-18
UNIT-2	knowledge representation in artificial intelligence	19-35
UNIT-3	Introduction to reasoning and soft computing	36-47

BLOCK-1 INTRODUCTION

Artificial Intelligence (AI) involves the ability of machines to replicate or enhance human cognitive functions, such as reasoning and learning from experience. Historically embedded in computer programs, AI has now expanded into a wide range of products and services. For example, modern digital cameras use AI to recognize objects in images, and the future promises even more innovative applications, such as AI-driven smart electric grids. AI leverages techniques from probability theory, economics, and algorithm design to solve real-world problems, drawing from diverse disciplines including computer science, mathematics, psychology, and linguistics. In this block, we explore how machines acquire knowledge through knowledge representation—a process essential for enabling machines to reason and interpret information. We also focus on First-Order Predicate Logic, a foundational model for capturing and manipulating knowledge in AI. Additionally, we delve into the core principles of reasoning within the frameworks of First-Order Logic (FOL) and Soft Computing. These concepts are crucial in AI and knowledge representation, playing a vital role in solving complex problems by combining logical precision with computational flexibility. The details of each unit in this block are described subsequently.

UNIT-I INTRODUCTION OF ARTIFICIAL INTELLIGENCE

- 1.1 Introduction
- 1.2 Objective
- 1.3 Definition
 - 1.3.1 Component of AI
 - 1.3.2 Applications of AI
- 1.4 Theoretical background of AI
- 1.5 AI problem domain
- 1.6 General AI techniques
- 1.7 Goal of AI
- 1.8 State space search
 - 1.8.1 State space
 - 1.8.2 State space representation
- 1.9 Water jug problem
- 1.10 8-puzzle problem
 - 1.11 Heuristic searching
 - 1.12 Summary
 - 1.13 Terminal Questions

1.1 INTRODUCTION

Artificial intelligence (AI) encompasses the capacity of machines to replicate or augment human cognitive abilities, like reasoning and learning from experience. While AI has had a presence in computer programs for quite some time, it has now extended its reach into a diverse array of products and services. For instance, cutting-edge digital cameras leverage AI software to identify objects within images. Furthermore, experts anticipate a wealth of ground breaking applications for AI in the future, including its integration into smart electric grids. AI harnesses a range of methodologies derived from probability theory, economics, and algorithm design to address real-world challenges. Furthermore, this field integrates insights from various disciplines including computer science, mathematics, psychology, and linguistics. Computer science equips AI researchers with the means to craft and implement algorithms, while mathematics furnishes the techniques for modelling and resolving intricate optimization and real-world problems that emerge as a result of these algorithms.

1.2 OBJECTIVES

After studying this unit, you should be able to:

- Explain what Artificial Intelligence is and what it does?
- Describe the evolution of Artificial Intelligence from early artificial neurons to deep learning.
- Describe State space representation, water jug problem, 8-puzzle problem, and heuristic searching.

1.3 DEFINITION

- Artificial Intelligence, often abbreviated as AI, derives its name from the fusion of two words: "Artificial," signifying the human-made aspect, and "Intelligence," representing ability to think. In essence, AI can be aptly defined as the discipline within computer science that empowers the creation of intelligent machines capable of emulating human behaviour, cognitive thought processes, and autonomous decision-making.
- Artificial Intelligence comes into play when a machine can exhibit human-like skills, including learning, reasoning, and problem solving. In the realm of Artificial Intelligence, there is no need to manually program a machine for every specific task. Instead, you can equip a machine with algorithms that enable it to operate with its own intelligence. This is where the true wonder of AI lies [1].
- Modern definition of AI:
 - In computer science, artificial intelligence (AI) is the field that creates machines and systems that can carry out operations that normally call for human cognitive capacities. Tasks like language comprehension, pattern recognition, decision-making, problem-solving, and experience-based learning are among them. Artificial Intelligence (AI) comprises a range of methodologies, including machine learning, deep learning, and neural networks. These techniques enable machines to mimic human intelligence and improve their capacity to process intricate data.
 - It is the development of intelligent systems capable of independent perception, analysis, and action in their surroundings. These systems are made to carry out activities like speech recognition, visual perception, language translation, and decision-making that call for intelligence comparable to that of a human. Large datasets, sophisticated algorithms, and computing power are all used in modern AI to create models that perform better over time, opening up new applications in sectors like finance, healthcare, and entertainment.

1.3.1 Component of AI

Artificial Intelligence extends beyond the boundaries of computer science, encompassing a wide range of interdisciplinary factors. To embark on the journey of creating AI, it is essential to understand the composition of human intelligence. Intelligence, a complex and intangible facet of the human mind, is an amalgamation of attributes such as reasoning, learning, planning, problem solving, perception, language comprehension, and more. These elements serve as the building blocks for creating AI systems. It consists with following techniques:

- Searching
- Knowledge
- Abstraction
- Inference

Artificial Intelligence requires the following discipline to achieve the aforementioned factors for a machine or software.

- Mathematics
- Biology
- Psychology
- Sociology
- Computer Science

- Neurons Study
- Statistics

1.3.2 Applications of AI:

Artificial Intelligence is widely employed across diverse sectors, playing an increasingly essential role in efficiently addressing complex challenges in various industries. It not only enhances the quality of life but also expedites daily operations. Here are some pivotal sectors where AI finds application [5]:

- **In Healthcare:**
 - AI aids in early and precise disease diagnosis.
 - AI speeds up drug discovery.
 - AI customizes treatment plans for individuals.
- **In Finance:**
 - AI-driven algorithms execute rapid financial decisions.
 - AI detects and prevents fraud in transactions.
- **In Education:**
 - AI personalizes educational content and assessments.
 - AI enhances language learning proficiency.
- **In Retail:**
 - AI recommends products based on customer preferences.
 - AI optimizes inventory and supply chains.
- **In Autonomous Systems:**
 - AI powers self-driving cars and autonomous decision-making.
 - AI controls drones for surveillance, agriculture, and logistics.
- **In Robotics:**
 - AI-driven robots handle manufacturing and surgery tasks.
 - Healthcare robots assist in medical procedures.
- **In Energy:**
 - AI optimizes energy distribution in smart grids.
 - AI improves energy efficiency in buildings
- **In Security:**
 - AI identifies individuals through facial recognition.
 - AI safeguards against cyber threats in real-time.
- **In Environmental Monitoring:**
 - AI manages environmental data, including air and water quality.
- **In Agriculture:**
 - AI optimizes crop yields and resource utilization in precision agriculture.
- **In Space Exploration:**

- AI aids in autonomous rover navigation and data analysis for space exploration.
- AI continuously transforms sectors, boosting efficiency, innovation, and quality of life. Evolving applications drive new opportunities and advancements in society.

1.4 Theoretical background of AI

The term "Artificial Intelligence" is neither a new concept nor a novel research technology. Its historical roots go back much further than most people realise. We find traces of mechanical men in the myths of Greek and Egyptian civilizations, hinting at the age-old fascination with creating intelligent beings. The following key moments in AI history illuminate the path from its inception to its current developments [1]:

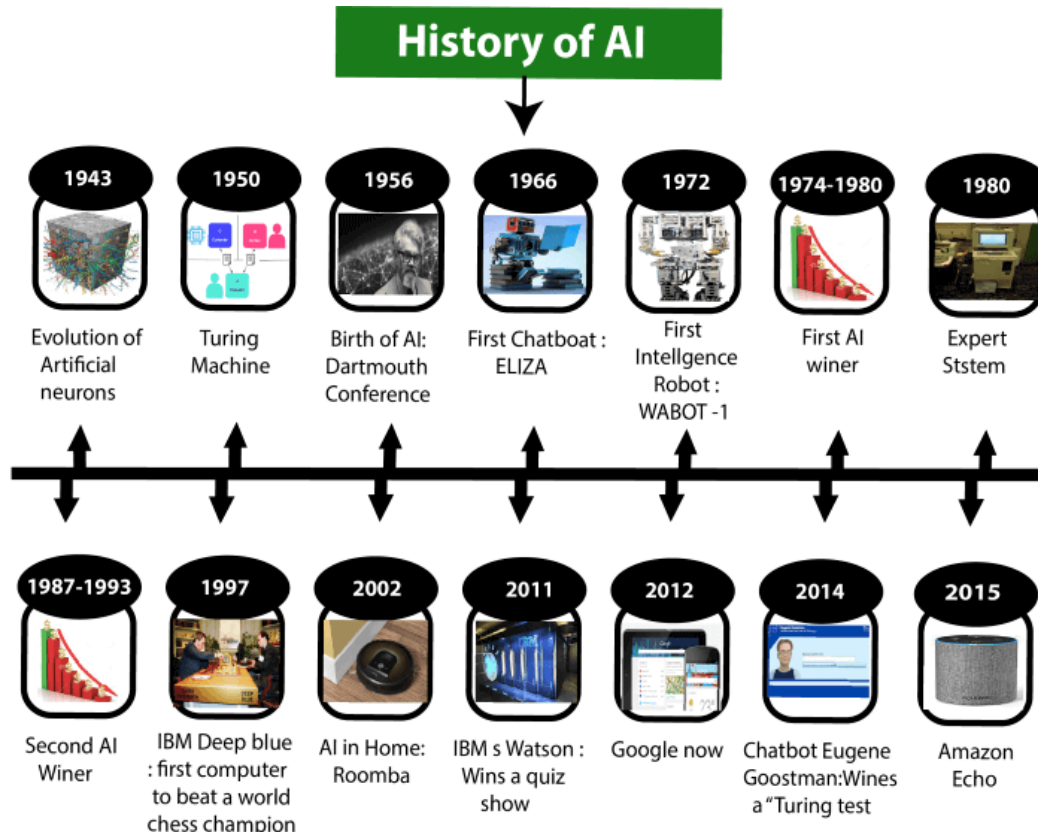


Fig. 1.1 History of Artificial Intelligence [7]

Maturation of Artificial Intelligence (1943-1952)

- **Year 1943** : Warren McCulloch and Walter Pitts' pioneering work in 1943 laid the groundwork for what we now call artificial intelligence. They developed a seminal model of artificial neurons, which marked an important step forward in the early development of artificial intelligence.
- **Year 1949** : Donald Hebb's ground-breaking contribution to the field of neural networks involved the demonstration of an updating rule for modifying the connection strength between neurons. This fundamental principle, now known as Hebbian learning, has played a pivotal role in shaping the way we understand and model synaptic plasticity and learning in artificial neural networks.
- **Year 1950** : Alan Turing, the renowned English mathematician, made significant contributions to the field of machine learning in the 1950s. In 1950, he published "Computing Machinery and

Intelligence," a seminal work where he introduced the concept of a test for assessing a machine's capacity to demonstrate intelligent behavior equivalent to human intelligence. This evaluation, famously known as the Turing test, has since become a cornerstone in the development and evaluation of artificial intelligence systems.

The birth of Artificial Intelligence (1952-1956):

- Year 1955: Allen Newell and Herbert A. Simon are credited with the creation of the "Logic Theorist," considered the first artificial intelligence program. This pioneering program made a significant impact by proving 38 out of 52 mathematics theorems, and in some cases, it even discovered new and more elegant proofs for these theorems. Their work marked a crucial milestone in the development of artificial intelligence and demonstrated the potential of computers to perform tasks that were traditionally thought to require human intelligence.
- Year 1956: The term "Artificial Intelligence" was officially coined by the American computer scientist John McCarthy during the Dartmouth Conference in 1956. This event marked the birth of AI as an academic field, and it was a time of great enthusiasm and innovation. Simultaneously, high-level computer programming languages like FORTRAN, LISP, and COBOL were being invented, further fuelling the optimism and possibilities for the field of AI.

The golden years-Early enthusiasm (1956-1974)

- **Year 1966:** During the early days of AI research, there was a strong emphasis on developing algorithms capable of solving mathematical problems. However, the field of AI began to diversify into various areas, including natural language processing. In 1966, Joseph Weizenbaum created the first ChatBot, named ELIZA. ELIZA is a notable milestone in AI history as it demonstrated the potential of computers to engage in human-like conversations and laid the foundation for the development of conversational AI and ChatBots.
- Year 1972: The first intelligent humanoid robot, known as WABOT-1, was developed in Japan. WABOT-1 was a significant milestone in the field of robotics and artificial intelligence.

- **The first AI winter (1974-1980):**

- The period from 1974 to 1980 is often referred to as the first "AI winter." An AI winter is a term used to describe a phase in the history of artificial intelligence research when there was a significant decline in government funding for AI projects. During these AI winters, there was a notable decrease in both public interest and publicity surrounding artificial intelligence. This lack of financial support and reduced interest had a cooling effect on AI research and development.

- **A boom of AI (1980-1987)**

- Year 1980: After the AI winter, AI experienced a revival with the introduction of Expert Systems in the 1980s. These systems emulated human expert decision-making.
- In 1980, the first national conference of the American Association of Artificial Intelligence was held at Stanford University, signifying a crucial moment in the field's development.

- **The second AI winter (1987-1993)**

- The second AI Winter, lasting from 1987 to 1993, saw a halt in funding from investors and governments. This was due to the perceived high cost and limited efficiency of AI research. While certain expert systems, like XCON, proved cost-effective, the AI field struggled to show its broader practicality and value.

- **The emergence of intelligent agents (1993-2011)**

- **Year 1997:** In 1997, IBM's Deep Blue made history by defeating the world chess champion, Gary Kasparov, becoming the first computer to achieve this feat.
- **Year 2002:** AI made its way into homes for the first time with the introduction of the Roomba, an AI-powered vacuum cleaner.
- **Year 2006:** AI made its entry into the business world before 2006, with companies like Facebook, Twitter, and Netflix incorporating AI into their operations.

Deep learning, big data and artificial general intelligence (2011-present)

- **Year 2011:** In 2011, IBM's Watson achieved a significant milestone by winning the quiz show Jeopardy. This demonstrated Watson's ability to understand natural language and quickly solve complex questions and riddles, showcasing its advanced capabilities in AI.
- **Year 2012:** Google introduced the "Google Now" feature in its Android app, which could provide users with predictive information.
- **Year 2014:** In the year 2014, ChatBot "Eugene Goostman" won a competition in the infamous "Turing test."
- **Year 2018:** IBM's "Project Debater" engaged in complex debates with two master debaters and delivered an exceptional performance.
- In the same year, Google's AI program "Duplex" displayed its capabilities by making a hairdresser appointment over the phone, and the person on the other end of the call was unable to discern that they were conversing with a machine.

Thus, Artificial Intelligence has indeed advanced to an impressive level. Concepts such as deep learning, big data, and data science are currently at the forefront of technological advancements. Leading companies like Google, Facebook, IBM, and Amazon are actively engaged in AI research and are responsible for creating remarkable devices and applications. The future of Artificial Intelligence is promising and holds the potential for even greater levels of intelligence and innovation.

1.5 AI problem domain

The AI problem domain encompasses the specific areas or industries where artificial intelligence techniques and technologies are utilized to tackle challenges and issues. These domains can range from natural language processing and computer vision to robotics, healthcare, finance, recommendation systems, and more. AI is applied within these domains to create intelligent solutions tailored to the unique demands of each field. In general, the AI domains consist of the following:

- Machine Learning
- Deep Learning
- Natural Language Processing
- Computer Vision
- Data Science

Another way to look at AI problem domain is by the types of tasks it can handle: expert tasks (like those you'd hire a professional for), formal tasks (involving logic and rules), and everyday tasks (common things you do daily).

Here are examples of expert tasks, formal tasks, and everyday tasks in AI:

- Expert tasks:
 - Design, engineering, graphics
 - Art, creativity
 - Music
 - Financial analysis
 - Consulting
- Formal tasks:
 - Board Game-Playing (chess, checkers, gobblet)
 - Logic
 - Calculus
 - Algebra
 - Verification, Theorem Proving
- Everyday tasks:
 - Vision
 - Speech
 - Natural Language Processing, Generation, Understanding
 - Reasoning

1.6 General AI techniques

Artificial intelligence techniques encompass a wide array of methods, algorithms, and data science approaches that empower computers to undertake tasks traditionally associated with human capabilities. These techniques play a crucial role in enabling AI systems to acquire knowledge, perform complex computations, recognize patterns, and provide forecasts for future events. Some of the most well-known AI techniques are [2]:

- Machine learning: Machine Learning (ML) stands as a foundational pillar of AI. This technique enables computers to learn from data and enhance their performance over time without the need for explicit programming. ML models have the capacity to make precise predictions and informed decisions using both supervised and unsupervised learning methods.
- Supervised learning: Supervised learning is a process where AI models are trained using labelled data, which is akin to presenting the AI with sets of questions and their corresponding known answers. This enables the AI to learn and make predictions or classifications based on the patterns and relationships it extracts from the labelled examples.
- Consider a company training an AI Chabot for customer support. They provide the AI with labelled question-response data. This supervised learning process helps the Chabot provide real-time assistance. In a similar vein, models for spam detection or classification tasks, like sentiment analysis, image recognition, or facial identification, are trained with labelled examples, making them capable of tasks like email filtering, diagnosing images, or recognizing objects and faces.
- Unsupervised learning: Unsupervised learning is a machine learning method that involves training with unlabelled data. Here, you supply a learning algorithm with relevant data for the intended task, but unlike supervised learning, there are no predefined outputs provided.
- The objective is to train AI to autonomously discover data patterns, relationships, and structures. This is particularly valuable for datasets where the patterns may not be known in advance. Anomaly detection is one such application. Consider a fraud

detection team examining financial transactions for signs of fraud. Instead of manual inspection for each transaction, AI can be trained on the financial data and alert financial institutions if it detects any irregularities that deviate from typical patterns.

- **Natural Language Processing:** Natural Language Processing (NLP) enables machines to understand, interpret, and generate human language, resulting in a new era of virtual assistants, Chatbots, and language translation tools. This advancement has significantly improved the fluidity of communication between humans and machines.
- **Computer Vision:** Computer Vision empowers machines to decipher visual data from the world, leading to transformative applications in healthcare, automotive, and robotics. This technology has made possible advancements like facial recognition, object detection, and autonomous driving.
- **Deep learning:** Deep Learning elevates machine learning by utilizing multi-layer neural networks to handle intricate data representations. This approach has been instrumental in AI accomplishments, including defeating human champions in games like 'chess' and 'Go', as well as advancing image and speech recognition systems. The deep learning has many application in medical science.
- **Reinforcement learning:** Reinforcement learning, a subset of, supervised type of learning without the explicit labelled output information. It enables machine to learn through trial and error. Unlike traditional data-driven approaches, reinforcement learning involves providing the model with rewards and penalties for its actions, fostering an iterative, feedback-driven process that allows the AI model to improve and optimize its behaviour over time. Companies apply reinforcement learning in various ways, like the development of Deep Blue, IBM's chess champion. Real-world usage includes training autonomous vehicles to make good decisions.

1.7 Goal of AI

The primary objectives of Artificial Intelligence include:

- The machine that thinks like human
- The machine that acts like human
- The machine that thinks relationally.
- The machine that acts relationally

Further, the objectives can be specified as:

1. One of the primary goals of Artificial Intelligence is to implement human-like intelligence in machines. This involves creating systems that can understand, think, learn, and behave in ways that resemble human intelligence.
2. Creating expert systems is a key application of Artificial Intelligence. These systems are designed to showcase intelligent behavior, learn, provide explanations, and offer advice to their users[2].

1.8 State space search

It is a fundamental problem-solving method employed in the field of Artificial Intelligence (AI). It is designed to navigate from an initial state to a desired goal state by systematically exploring and analyzing a range of intermediate states. This approach involves exhaustively examining all possible problem states, making it a cornerstone of AI and a versatile tool applied in diverse applications, spanning from game-playing algorithms to the intricacies of natural language

processing [1, 3].

1.8.1 State Space:

A state space is a mathematical framework for representing a problem that encompasses the entire range of possible problem states. It provides the framework in search algorithms because it allows for the representation of the problem's initial state, desired goal state, and current state under consideration. Each state is represented by a set of variables within this framework, allowing for a structured and comprehensive examination of the problem's various facets.

The size of the state space has a significant impact on the efficiency of a search algorithm. As a result, it is critical to choose an appropriate representation and search strategy with care to traverse the state space efficiently and effectively. Among the most renowned state space search algorithms, the A* algorithm stands out prominently. In addition to A*, other frequently employed state space search algorithms encompass breadth-first search (BFS), depth-first search (DFS), and hill climbing. These algorithms each offer distinct strategies for navigating the state space, tailored to the specific characteristics of the problem at hand.

1.8.2 State Space Representation:

- It involves pinpointing the INITIAL STATE (the starting point) and the GOAL STATE (the desired end). It then follows a predefined sequence of actions referred to as "States." There are two components to the representation of state spaces:
 - Static States
 - Transitions between States
- State Space Graphs: When the system's potential states are limited, they can be exhaustively captured, including the transitions connecting them, within a state space graph.
- State: AI problems can be represented as a set of well-defined states, which include an Initial State, a Goal State, and multiple intermediate states created by applying specific rules.
- Space: In an AI problem, "space" encompasses the comprehensive set of all imaginable states.
- Search: Search is a method that progresses from the initial state to the target state by adhering to valid rules within the realm of all potential states.
- Search tree: A search tree is a tree-like representation of the problem being explored. The root node of the tree corresponds to the initial state and serves as the starting point for the search. Additionally, it furnishes the agent with a comprehensive inventory of all feasible actions.
- Problem state: A problem space can equivalently be regarded as a search space since, in the process of problem solving; we navigate through this space to find the desired goal state.
- Transition model: A transition model specifies the effects of each action.

Features of State Space Search:

- State space search offers several attributes that render it a potent problem-solving technique in Artificial Intelligence. Some of these key features encompass:
 - **Exhaustiveness:**
In search of a solution, state space search investigates all possible problem states.
 - **Completeness:**
If a solution exists, state space search is designed to discover it.

- **Optimality:**

Searching through a state space can lead to the discovery of an optimal solution when the search strategy and problem representation are appropriately chosen.

Uninformed and Informed Search:

- In the realm of artificial intelligence, state space search can be categorized as uninformed when it doesn't utilize supplementary problem-related information. In contrast, informed search strategically leverages additional insights, like heuristics, to intelligently steer the search towards more promising paths.

Self- Evaluations

- What is AI problem domain? Explain it.
- What is the component of AI? Explain any five application of AI.
- Differentiate supervised and unsupervised machine learning.
- What are the features of State space search?

1.9 Water jug problem:

In the classic water jug problem, we are presented with two unmarked jugs: one capable of holding 3 gallons of water, and the other with a capacity of 4 gallons. No additional measuring tools are at our disposal. The challenge for the agent is to successfully fill the 4-gallon jug with precisely 2 gallons of water, using only these two jugs, and commencing with both completely empty. To tackle this problem, a specific set of rules has been formulated:

S.No.	Initial State	Condition	Final state	Description of action taken
1.	(x,y)	If $x < 4$	(4,y)	Fill the 4 gallon jug completely
2.	(x,y)	if $y < 3$	(x,3)	Fill the 3 gallon jug completely
3.	(x,y)	If $x > 0$	(x-d,y)	Pour some part from the 4 gallon jug
4.	(x,y)	If $y > 0$	(x,y-d)	Pour some part from the 3-gallon jug
5.	(x,y)	If $x > 0$	(0,y)	Empty the 4 gallon jug
6.	(x,y)	If $y > 0$	(x,0)	Empty the 3 gallon jug
7.	(x,y)	If $(x+y) < 7$	(4, y-[4-x])	Pour some water from the 3 gallon jug to fill the four gallon jug
8.	(x,y)	If $(x+y) < 7$	(x-[3-y],y)	Pour some water from the 4 gallon jug to fill the 3 gallon jug.
9.	(x,y)	If $(x+y) < 4$	(x+y,0)	Pour all water from 3 gallon jug to the 4 gallon jug
10.	(x,y)	if $(x+y) < 3$	(0, x+y)	Pour all water from the 4 gallon jug to the 3 gallon jug

Here, let x denote the 4-gallon jug and y denote the 3-gallon jug.

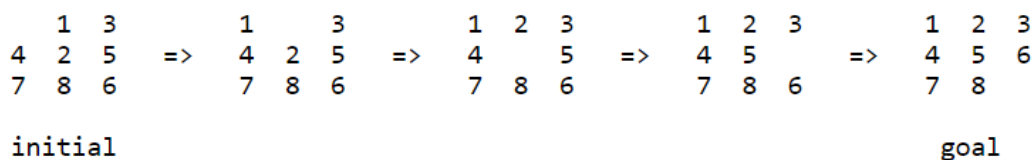
One of the Solution of water jug problem according to the production rules are:

S.No.	4-gallon jug contents	3-gallon jug contents	Rule followed
1.	0 gallon	0 gallon	Initial state
2.	0 gallon	3 gallons	Rule no.2
3.	3 gallons	0 gallon	Rule no. 9
4.	3 gallons	3 gallons	Rule no. 2
5.	4 gallons	2 gallons	Rule no. 7
6.	0 gallon	2 gallons	Rule no. 5
7.	2 gallons	0 gallon	Rule no. 9

After the seventh attempt, we finally explore the solution path in the large size of the search space and arrive at a state that aligns with our intended goal state. Now, our problem is successfully resolved.

1.10 8-puzzle problem [5]

A classic puzzle game played on a 3X3 board with 8 numbered tiles and one empty space. The objective is to slide the adjacent tiles into the space, ultimately arranging the numbered tiles to match a predefined final configuration. The permissible moves involve shifting tiles in the four adjacent directions: left, right, above, and below. A systematic demonstration of permissible moves is provided using below example, guiding you from the initial board configuration (on the left) to the wanted goal position (on the right).



- Rules:** To solve the 8-puzzle problem, the tiles are moved by swapping them with the empty space, which the user can visualize as moving the empty space itself rather than the tiles. The empty space can only move in one of four directions—**up, down, left, or right**—and **cannot move diagonally**. Each move involves swapping the empty space with an adjacent tile, allowing the puzzle to progress toward the goal state one-step at a time.
- State space:** The state space of the 8-puzzle consists of all possible configurations the puzzle can achieve, ranging from the initial state to the goal state. Each state within this space represents a distinct arrangement of the puzzle tiles. The following figure represents the state-space tree i.e., all states that may be reached from the beginning state (Indicated by root).

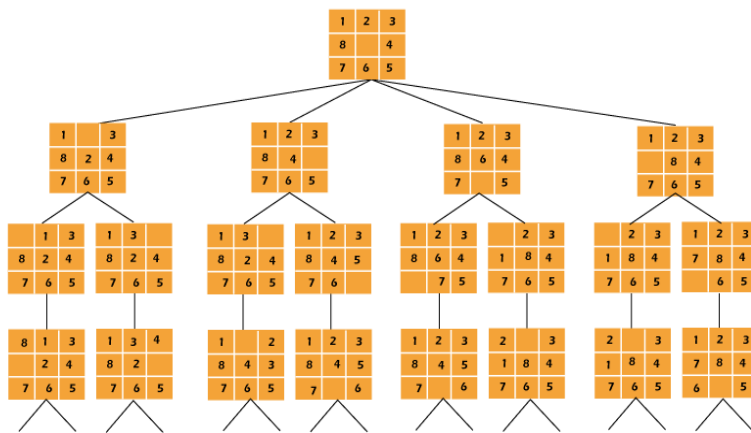


Fig.: 1.2 Puzzle's State Space Tree.

- **Number of state:** There are $9! \div 2 = 1,81,440$ possible states
- **Number of path:** The number of paths in the 8-puzzle depends on the specific initial and goal states, as well as their connections within the state space. Since multiple paths can lead to the goal state, and these paths can vary in length, the exact number of paths is not fixed. However, in the worst case, solving the 8-puzzle can require up to 31 moves from the most challenging initial state to the goal state.
- **Search techniques:**
 - Problem-solving algorithms, like Breadth-First Search and A*, systematically navigate the vast state space of the 8-puzzle, evaluating different configurations to find the optimal path from the initial to the goal state.
 - The complexity of the puzzle lies in the multitude of possible states, making efficient search strategies essential for navigating this space effectively.
 - **Breadth-First Search (BFS)** is an uninformed algorithm that explores states level by level, guaranteeing the shortest path to the goal but at the cost of high memory usage.
 - **Depth-First Search (DFS)**, another uninformed method, is more memory-efficient as it explores states deeply before backtracking, though it doesn't guarantee an optimal solution.
 - **The A*** algorithm, an informed search, efficiently finds optimal solutions by combining the cost to reach a state (g-value) with a heuristic estimate (h-value), making it ideal for problems like the 8-puzzle when using an admissible heuristic.

1.11 Heuristic searching

- **What is Heuristics?**
 - A heuristic is a problem-solving technique employed to expedite solutions in situations where traditional methods may prove inefficient. These methods are particularly valuable for approximating solutions when classical approaches are impractical. Heuristics are lauded for their ability to yield swift and pragmatic problem-solving outcomes [6].
 - Heuristics are problem-solving strategies that draw from experiences with analogous problems. These approaches rely on practical methods and shortcuts to generate solutions, which, while not necessarily optimal, prove adequate within the constraints of a specific time

frame.

- Why do we need heuristics?
 - Heuristics come into play when there's a pressing need for a short-term solution.
 - In the face of intricate situations marked by restricted resources and time, heuristics enable companies to swiftly arrive at decisions through the use of shortcuts and approximate calculations. Many heuristic methods hinge on mental shortcuts that leverage past experiences to guide decision-making.
 - While the heuristic approach may not always yield the optimal solution, it reliably guides us toward a satisfactory outcome within a reasonable timeframe. The choice of heuristic method may vary depending on the specific context and the complexity of the problem at hand. Some widely used heuristic techniques include trial and error, intuitive guesswork, the process of elimination, and historical data analysis. These methods rely on readily accessible information that may not be tailored to the exact problem but remains highly applicable.
- **Heuristic search techniques in AI (Artificial Intelligence)**

We can categorize heuristic techniques into two distinct groups [6]:

- Direct Heuristic Search techniques in AI
 - It encompasses Blind Search, Uninformed Search, and the Blind Control Strategy. However, these techniques can be resource-intensive, often demanding significant memory and time resources. They involve an exhaustive search of the entire solution space, making decisions based on an arbitrary sequencing of operations.
 - Examples of Direct Heuristic search techniques comprise well-known methods like Breadth-First Search (BFS) and Depth-First Search (DFS).
- Weak Heuristic Search techniques in AI
 - It encompasses Informed Search, Heuristic Search, and Heuristic Control Strategies. The effectiveness of these techniques becomes evident when they are thoughtfully applied to tasks that match their specific characteristics. Generally, they demand access to domain-specific knowledge to achieve their full potential.
 - Examples of Weak Heuristic search techniques encompass Best First Search (BFS), A*, AO*, and Hill Climbing.

1.12 SUMMARY

In summary,

- We have explored in this unit the concept of Artificial intelligence. AI refers to the simulation of human-like intelligence in machines, allowing them to perform tasks that typically require human intelligence, such as learning, reasoning, and problem solving. One of the key characteristics of AI is its ability to generalize from data and adapt to various tasks without the need for manual, task-specific programming.
- You have delved into the objectives of Artificial Intelligence (AI), the fundamental building blocks that constitute AI systems, and the ongoing process of advancing AI technology.
- In conclusion, you have also delved into the concepts of state space representation, the water jug problem, and the 8-queen problem.

1.13 **TERMINAL QUESTIONS**

1. What is artificial intelligence? Explain its application.
2. Explain the development of artificial intelligence.
3. Describes various general AI techniques.
4. What is Problem state? Describe a state space representation.
5. What is water jug problem? Explain it.
6. What is 8-puzzle problem? Explain it.
7. What is Heuristics? Why is it needed in AI?
8. Explain Heuristic search techniques in AI.

BIBLIOGRAPHY

1. Artificial Intelligence (AI): Recent Trends and Applications. (2021). United States: CRC Press.
2. Artificial intelligence. <https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/top-ai-techniques>. Accessed on 01-10-23.
3. State space search. <https://www.scaler.com/topics/state-space-search-in-artificial-intelligence/> Accessed on 04-10-23.
4. Water jug problem. <https://www.includehelp.com/ml-ai/water-jug-problem-in-artificial-intelligence.aspx>. Accessed on 07-10-23.
5. 8-puzzle problem. <https://www.javatpoint.com/8-puzzle-problem-in-python>. Accessed on 12-10-23.
6. Pearl, J. (1984). Heuristics: Intelligent Search Strategies for Computer Problem Solving. United Kingdom: Addison-Wesley Publishing Company.
7. History of AI. <https://www.javatpoint.com/history-of-artificial-intelligence>, Accessed on 6-10-23.

UNIT-II KNOWLEDGE REPRESENTATION IN AI

- 2.1 Introduction
- 2.2 Objective
- 2.3 Knowledge representation in AI
 - 2.3.1 Knowledge representation
 - 2.3.2 Different Types of Knowledge
 - 2.3.3 Cycle of Knowledge Representation in AI
 - 2.3.4 Knowledge representation model
 - 2.3.5 Techniques of Knowledge Representation in AI
 - 2.3.6 Representation Requirements
- 2.4 First-Order Logic in Artificial intelligence
 - 2.4.1 First-Order logic
 - 2.4.2 Syntax of First-Order logic
 - 2.4.3 Inference in First-Order Logic
 - 2.4.4 FOL inference rules for quantifier
 - 2.4.5 Resolution in FOL
- 2.8 Summary
- 2.9 Terminal Questions
- Bibliography

2.1 INTRODUCTION

Human beings possess remarkable abilities in comprehending, reasoning, and interpreting knowledge, enabling them to perform diverse tasks in the physical world. Machines, in contrast, achieve similar capabilities through a distinct approach. This unit has provided insights into the intriguing process of machines acquiring knowledge through knowledge representation. This representation serves not only as a foundational element but also as a driving force for machines to engage in the processes of reasoning and interpretation. Furthermore, we have examined in depth the well-established model of first-order predicate logic, which stands as a fundamental cornerstone for the capture and manipulation of knowledge in the realm of artificial intelligence.

2.2 OBJECTIVES

After studying this chapter, you should be able to:

- Familiar with the realm of knowledge representation in artificial intelligence.
- To explore knowledge representation enables machines to engage in reasoning and interpretation.
- To described the first order predicate logic, resolution, and monotonic reasoning.

2.3 Knowledge representation in AI

2.3.1 Knowledge Representation

- Knowledge Representation in AI pertains to the method used to depict information. It fundamentally examines how an intelligent agent's beliefs, intentions, and judgments can be effectively articulated for automated reasoning. A key objective of Knowledge Representation is to model intelligent behaviour in an agent [1].
- Knowledge Representation and Reasoning (KR, KRR) involve the process of capturing and structuring real-world information in a form that computers can comprehend and subsequently apply to tackle intricate real-life challenges, such as engaging in natural language communication with humans. Knowledge representation in AI goes beyond mere data storage; it enables machines to glean insights from that knowledge and exhibit intelligent behaviour akin to human capabilities.
- AI demands the representation of diverse categories of knowledge, comprising:
 - Objects
 - Events
 - Performance
 - Facts
 - Meta-Knowledge
 - Knowledge-base

2.3.2 Different Types of Knowledge

There are five types of Knowledge such as:

- **Declarative Knowledge** – This category encompasses concepts, facts, and objects, and it is typically conveyed through declarative sentences.
- **Structural Knowledge** – It constitutes fundamental problem-solving knowledge, elucidating the relationships between concepts and objects.
- **Procedural Knowledge** – This category pertains to knowing how to perform specific tasks and encompasses rules, strategies, procedures, and similar components.
- **Meta Knowledge** – Meta-knowledge characterizes knowledge about other categories or types of knowledge.
- **Heuristic Knowledge** – This represents expert knowledge within a specific field or subject.

2.3.3 Cycle of Knowledge Representation in AI

- Artificial Intelligent Systems are typically composed of a variety of components that collectively enable them to exhibit intelligent behaviour. Some of the core components in these systems include [2, 4]:
 - Perception
 - Learning
 - Knowledge Representation & Reasoning
 - Planning
 - Execution

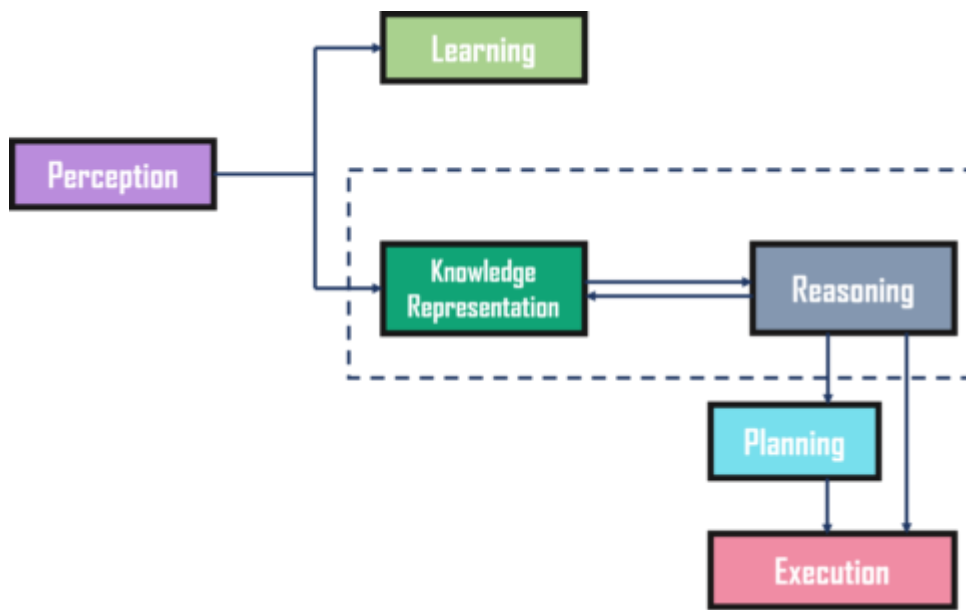


Fig. 2.1 Knowledge Representation Cycle [2].

The diagram above illustrates the interaction between an AI system and the physical world, depicting the integral components that contribute to the system's display of intelligence.

- The Perception component serves as the means to acquire data and information from the environment. It enables the system to gather data, identify the sources of sensory input, and assess if any damage has occurred to the AI. Furthermore, this component plays a crucial role in defining the appropriate responses when any sensory input is detected.
- The Learning Component is responsible for assimilating knowledge from the data captured by the Perception component. Its objective is to enable computers to learn rather than relying solely on traditional programming. Learning is centered on the concept of self-improvement. To acquire new knowledge, the system employs processes such as knowledge acquisition, inference, heuristics acquisition, and optimization of search methods, among others.
- The pivotal component in this cycle is Knowledge Representation and Reasoning, which underpins the manifestation of human-like intelligence in machines. Knowledge representation is the key to comprehending intelligence. Instead of attempting to construct or emulate brains from the ground up, its emphasis lies in comprehending and constructing intelligent behaviour from a top-down perspective. It centers on determining what information an agent must possess to exhibit intelligent behaviour. Furthermore, it establishes the methodologies for automated reasoning to make this knowledge readily accessible when required [1].
- The Planning and Execution components, informed by knowledge representation and reasoning, encompass initial state definition, preconditions and effects identification, action sequence planning to achieve a specific goal, followed by the execution of the devised plan.

2.3.4 Knowledge representation models:

The various knowledge representation models used in artificial intelligence are:

- Semantic Networks:
 - Semantic networks utilize nodes and links to convey knowledge, illustrating connections between concepts
 - Example: In a medical context, a semantic network can link "disease" to "symptoms" and "treatment" to represent medical knowledge relationships.

- Frames:
 - Frames create structured representations of objects, detailing their attributes and properties through templates.
 - Example: A "car" frame might include slots for "color," "model," "year," and other attributes to describe specific cars.
- Ontologies:
 - Ontologies establish hierarchical structures of concepts and their relationships, often used in the Semantic Web for machine understanding of data.
 - Example: The Gene Ontology categorizes genes and their functions, facilitating comprehensive integration of biological data.
- Rule-Based Systems:
 - Rule-based systems employ collections of rules to express knowledge, defining conditions and corresponding actions for logical reasoning.
 - Example: In a healthcare expert system, a rule might state, "If a patient has a fever and cough, then consider a respiratory infection."
- Description Logics:
 - Description logics, rooted in predicate logic, provide a range of formalisms for structured knowledge representation.
 - Example: Description logics are adept at capturing complex relationships between entities, especially in the Semantic Web and database domains.
- Fuzzy Logic:
 - Fuzzy logic accommodates uncertainty and vagueness by assigning degrees of truth to statements, enhancing the representation of imprecise information.
 - Example: Fuzzy logic can manage speed control in autonomous vehicles based on imprecise terms like "near," "far," or "slow."
- Bayesian Networks:
 - Bayesian networks model probabilistic links between variables, making them valuable for representing probabilistic knowledge.
 - Example: A Bayesian network can assess the likelihood of a patient having a specific disease based on symptoms, aiding medical diagnosis [4].
- Neural Networks:
 - While primarily employed in machine learning and pattern recognition, neural networks indirectly represent knowledge through learned patterns and associations.
 - Example: In image recognition, a neural network learns to recognize objects, effectively encoding knowledge about those objects.
- Commonsense Knowledge Bases:
 - Common sense knowledge bases store general information about the world, encompassing facts, relationships, and common-sense reasoning principles.
 - Open AI's Concept Net is an exemplar of a commonsense knowledge base, offering insights into everyday concepts and their associations.

These diverse knowledge representation models are chosen strategically based on the context, the type of knowledge to be conveyed, and the demands of the specific AI task at hand. The selection of a representation model significantly influences an AI system's functionality and adaptability[4].

2.3.5 Techniques of Knowledge Representation in AI

There are four techniques of representing knowledge such as[2]:

- Logical Representation
- Semantic network representation
- Production rules
- Frame representation

Logical Representation

- Logical representation is a structured language governed by explicit rules that pertain to propositions, ensuring unambiguous representation. It serves as a means to convey conclusions derived from various conditions and establishes vital communication guidelines. Moreover, it encompasses well-defined syntax and semantics that facilitate robust inference. Every sentence can be systematically translated into logical form using the rules of syntax and semantics.
- Logical representation serves as a foundation for logical reasoning and underpins the design of programming languages, enabling the execution of systematic and coherent logic-based operations.

Semantic Network Representation

- Semantic networks provide an alternative to predicate logic for knowledge representation. They enable the expression of knowledge through graphical networks, comprising nodes that represent objects and arcs that depict relationships between these objects. Additionally, semantic networks categorize objects into various forms and establish connections between them, offering a visual and structured means of representing information.
- This representation encompasses two fundamental types of relations: "is-a" relations, signifying hierarchical categorization, and "part-of" relations, indicating the inclusion or composition of one object within another. These relations play a pivotal role in shaping the structural framework and interconnections within the semantic network.

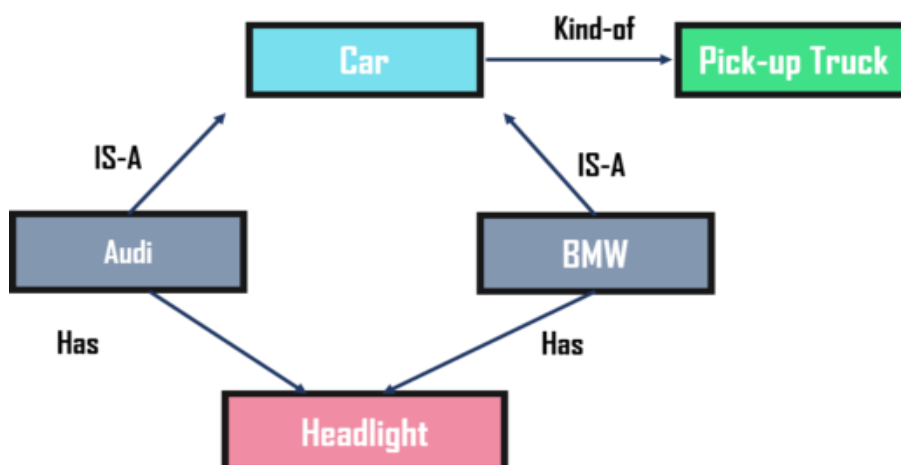


Fig. 2.2 Semantic network [2].

Frame Representation

- A frame is a structured record that comprises a set of attributes and their associated values, serving to describe an entity within the world. These data structures are employed in artificial intelligence to compartmentalize knowledge into substructures by representing typical scenarios or situations. Essentially, a frame consists of a collection of slots, each with names and values referred to as facets, which can vary in type and size.

Production Rules

- In production rules, an agent evaluates a set of conditions, and if these conditions are met, the production rule is triggered, leading to the execution of a corresponding action. The condition segment of the rule governs the selection of rules that can be applied to a specific problem, while the action segment directs the execution of problem-solving steps. This entire sequence of evaluating conditions and taking actions is commonly referred to as the "recognize-act cycle."
- The production rules system comprises three principal components:
 - The set of production rules
 - Working Memory
 - The recognize-act-cycle
- The production rules are expressed in natural language. The production rules are designed to be highly modular and can be readily added, removed, or modified to adapt to changing requirements or scenarios.

2.3.6 Representation Requirements

- A robust knowledge representation system should possess the following essential properties:
- Representational Accuracy: It should comprehensively represent all necessary knowledge.
- Inferential Adequacy: It should have the capability to generate new knowledge by manipulating existing representational structures.
- Inferential Efficiency: It should optimize inference by storing and utilizing appropriate guides to direct the knowledge mechanism effectively.
- Acquisitional efficiency: It should facilitate easy acquisition of new knowledge through automated methods.

2.4 First-Order Logic in Artificial intelligence

- In propositional logic, we can exclusively represent binary facts, classified as either true or false. This logic is inadequate for capturing complex sentences or natural language statements, as it possesses limited expressive capabilities. For instance, consider the following sentence that cannot be effectively represented using propositional logic [3].
 - "Some humans are intelligent", or
 - "Sachin likes cricket."
- To represent the above statements, propositional logic falls short, necessitating the use of more powerful logical frameworks, such as first-order logic [2].

2.4.1 First-Order logic:

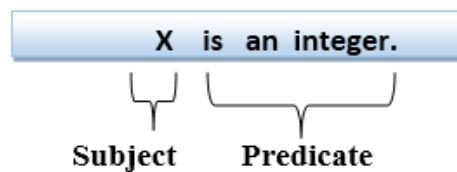
- First-order logic, often used in artificial intelligence for knowledge representation, serves as an extension to propositional logic, offering greater expressiveness to handle more complex relationships and statements.
- First-order logic (FOL) is expressive enough to succinctly represent natural language statements, making it a valuable tool for knowledge representation in artificial intelligence.
- First-order logic is also referred to as Predicate logic or First-order predicate logic. It is a powerful language that provides a more straightforward means of representing information about objects and expressing relationships between those objects.
- First-order logic, unlike propositional logic, does not just assume the existence of facts in the world. It also assumes the following aspects about the world[3]:
 - Objects: In first-order logic, it encompasses the existence and attributes of entities such as "A," "B," people, numbers, colors, etc.
 - Relations: In first-order logic, relations can be unary, such as "red" or "round," and n-ary, like "the sister of," "brother of," "has color," or "comes between." This versatility allows for the structured and precise representation of a diverse range of concepts and relationships.
 - Function: In first-order logic, functions such as "Father of," "best friend," "third inning of," and "end of" showcase the system's ability to represent a wide array of functional relationships and associations within the knowledge base.
- As a natural language, first-order logic also has two main parts:
 - Syntax
 - Semantics

2.4.2 Syntax of First-Order Logic:

- The syntax of first-order logic (FOL) specifies the set of symbols that constitute a valid logical expression. The fundamental syntactic elements in FOL are symbols, and statements are often written in shorthand notation to facilitate concise representation [3].

Constant	1, 2, A, John, Mumbai, cat,...
Variables	x, y, z, a, b,...
Predicates	Brother, Father, >,...
Function	sqrt, LeftLegOf,
Connectives	\wedge , \vee , \neg , \Rightarrow , \Leftrightarrow
Equality	$=$
Quantifier	\forall , \exists

- Basic Elements of First-order logic: Following are the basic elements of FOL syntax:
 - **Atomic sentences:**
 - Atomic sentences represent the fundamental building blocks of first-order logic. They are constructed by combining a predicate symbol with a set of parentheses containing a sequence of terms.
 - Atomic sentences in first-order logic can be represented in the form of "Predicate(term1, term2, ..., term n)," where the predicate symbol is followed by a list of terms enclosed in parentheses.
 - **Example:** Ravi and Ajay are brothers: \Rightarrow Brothers(Ravi, Ajay).
Chinky is a cat: \Rightarrow cat (Chinky).
 - **Complex Sentences:**
 - Complex sentences in first-order logic are created by combining atomic sentences through the use of logical connectives.
- **First-order logic statements consist of two essential components:**
 - Subject: The subject is the central component of the statement.
 - Predicate: A predicate can be defined as a relationship that connects or associates two atoms within a statement.
 - **Consider the statement: "x is an integer."** It comprises two elements: the initial part "x" serves as the subject of the statement, while the subsequent part, "is an integer," functions as the predicate.

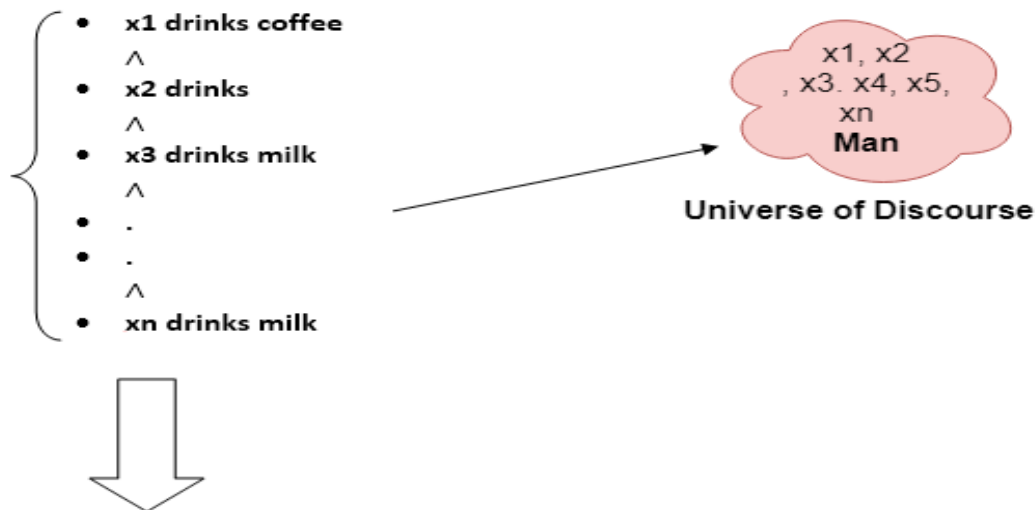


- **Quantifiers in First-order logic:**
 - A quantifier is a linguistic element that introduces quantification, and quantification defines the quantity of individuals within the universe of discourse.
 - These symbols play a crucial role in delineating and characterizing the range and extent of variables within a logical expression. Quantifiers come in two primary types:
 1. Universal Quantifier, (for all, everyone, everything)
 2. Existential quantifier, (for some, at least one).
- **Universal Quantifier:**
 - The Universal quantifier is a logical symbol that signifies that the statement it encompasses holds true for every instance or element within its specified range.
 - The Universal quantifier is denoted by the symbol \forall , which bears a resemblance to an inverted letter "A."

- If x is a variable, then $\forall x$ is read as "for all x " or "for every x .", For each x .
- Example:

All man drink coffee.

Let a variable x which refers to a man so all x can be represented in UOD as below:



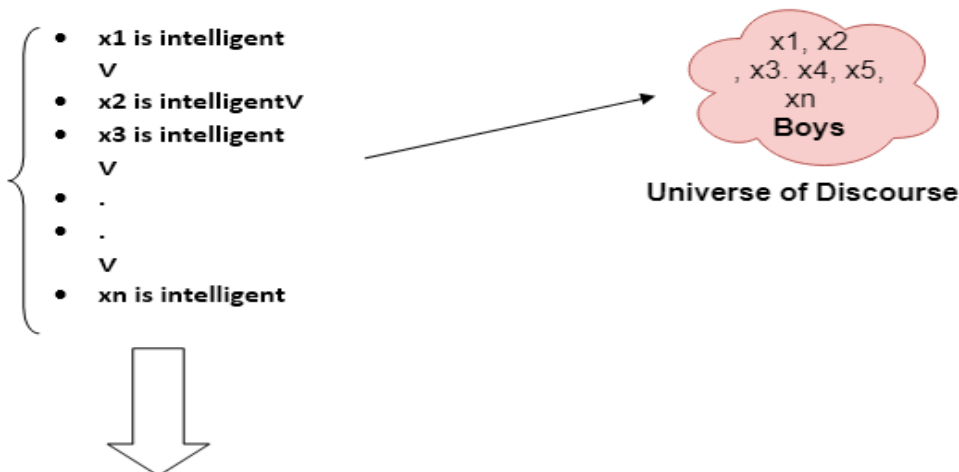
So in shorthand notation, we can write it as :

$\forall x \text{ man}(x) \rightarrow \text{drink}(x, \text{coffee}).$

It will be read as: There are all x where x is a man who drink coffee.

▪ **Existential Quantifier:**

- Existential quantifiers are a type of quantifier that indicates that the statement within their scope is true for at least one instance of something.
- It is represented by the logical operator \exists , which resembles an inverted letter "E." When used with a predicate variable, it is referred to as an existential quantifier.
- If x is a variable, then existential quantifier will be $\exists x$ or $\exists(x)$. And it will be read as:
- There exists a ' x .', For some ' x .', For at least one ' x .'
- Example:
 - Some boys are intelligent.



So in short-hand notation, we can write it as:

$\exists x: \text{boys}(x) \wedge \text{intelligent}(x)$

It will be read as: There are some x where x is a boy who is intelligent.

2.4.3 Inference in First-Order Logic

- In First-Order Logic (FOL), inference is a fundamental process used to derive new facts or sentences from existing sentences. To grasp FOL inference rules, it's essential to first become familiar with some basic terminology commonly employed in FOL [5].
- **Substitution:**
 - Substitution is a fundamental operation that applies to both terms and formulas in First-Order Logic (FOL). It is a crucial operation in all inference systems within FOL. Substitution becomes more intricate when quantifiers are present in FOL. When we write $F[a/x]$, it signifies the substitution of a constant "a" in place of the variable "x" within the formula F. This process allows us to replace occurrences of "x" with "a" in the formula, which is a common operation when working with quantified expressions.
- **Equality:**
 - First-order logic (FOL) extends its expressive power beyond just the use of predicates and terms to form atomic sentences; it also incorporates another essential concept: equality. In FOL, equality symbols play a pivotal role in denoting that two terms are, indeed, referring to the same object. This equality assertion enhances the logic's capacity to reason about the identity and equivalence of entities within a given domain.
 - **Example: Brother (John) = Smith.**
 - In the example provided earlier, we can observe that the object denoted by "Brother(John)" is identical to the object represented by "Smith." The equality symbol in first-order logic serves as a means to explicitly state this equivalence, allowing for precise and unambiguous representation of relationships between entities. Furthermore, in cases where two terms do not refer to the same objects, the equality symbol can be employed in conjunction with negation. This conveys the notion that the two terms are distinct and represent separate entities within the logical framework. **Example: $\neg(x=y)$ which is equivalent to $x \neq y$.**

2.4.4 FOL inference rules for quantifier:

- Just like propositional logic, first-order logic (FOL) employs a set of inference rules to facilitate logical reasoning. Here are some fundamental inference rules in FOL [3, 5, 6]:
 1. **Modus Ponens:**
 - When you have a statement in the form "P implies Q" ($P \rightarrow Q$), and you have confirmed the truth of "P," it logically follows that "Q" must also be true. This is a fundamental implication rule in logic, where the presence of P necessitates the presence of Q.
 - Example: Given: If it's raining (P), then I'll take an umbrella (Q). Statement 1: It's raining (P).
Conclusion: I'll take an umbrella (Q).
 2. **Modus Tollens:**
 - If you possess a statement "P implies Q," and you are aware that "Q" is indeed false ($\neg Q$), it is a valid deduction to conclude that "P" is also false ($\neg P$). This inference rule

is a fundamental aspect of logical reasoning, where the absence of Q implies the absence of P.

- Example: Given: If I study hard (P), then I'll pass the exam (Q).
 - Statement: I didn't pass the exam ($\neg Q$).
 - Conclusion: I didn't study hard ($\neg P$).

Self-Evaluations

- What are different types of knowledge? Explain cycles of knowledge representation in AI.
- What is semantic network? Explain it.
- Explain the properties of a knowledge representation system
- What is atomic sentence and quantifiers in first order logic?
- Explain inference in first order logic.

3. Universal Generalization:

- Universal generalization in first-order logic (FOL) allows us to conclude that if a statement $P(c)$ holds for any arbitrary element c in the domain, then it holds universally as $\forall x P(x)$. This rule extends specific truths to universal truths.

$$\frac{P(c)}{\forall x P(x)}$$

- It can be represented as: $\forall x P(x)$.
- This rule is applied to demonstrate that every element shares a common property, provided that x is not a free variable within it.
- Example 1: Let us represent, $P(c)$: "A byte contains 8 bits", so for $\forall x P(x)$ "All bytes contain 8 bits." it will also be true.
- Example 2: Statement: "Riya aced the Science test."
- Conclusion: "All students as good as Riya aced the Science test."

4. Universal Instantiation:

- Universal instantiation (UI), also known as universal elimination, is a valid inference rule that can be iteratively applied to introduce new sentences into the knowledge base. Each iteration maintains logical equivalence with the previous knowledge base. According to UI, we can deduce any sentence by substituting a ground term for the variable.
- The UI rule allows us to derive any sentence, $P(c)$, by replacing the universally quantified variable, x , with a specific ground term, c , which is a constant within the universe of discourse.

$$\frac{\forall x P(x)}{P(c)}$$

- It can be represented as: $P(c)$.

- **Example: 1.**

- IF "Every person like ice-cream" $\Rightarrow \forall x P(x)$ so we can infer that "John likes ice-cream" $\Rightarrow P(c)$

- **Example: 2.**

- "All kings who are greedy are Evil." So let our knowledge base contains this detail as in the form of FOL:
- $\forall x \text{ king}(x) \wedge \text{greedy}(x) \rightarrow \text{Evil}(x),$

5. Existential instantiation:

- Existential instantiation, also known as existential elimination, is a valid inference rule in first-order logic. It can be applied only once to replace an existential sentence. While the new knowledge base (KB) is not logically equivalent to the old KB, it remains satisfiable if the old KB was initially satisfiable.
- This rule allows us to deduce $P(c)$ from a formula presented in the form of $\exists x P(x)$, introducing a new constant symbol c for this purpose.
- The limitation of this rule is that the term c employed in the rule must be a new term for which the statement $P(c)$ holds true.

$$\frac{\exists x P(x)}{P(c)}$$

- It can be represented as:
- Example:1
- From the given sentence: $\exists x \text{ Crown}(x) \wedge \text{OnHead}(x, \text{John}),$

Concludes: $\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \rightarrow \text{Evil}(\text{John}),$

$\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \rightarrow \text{Evil}(\text{Richard}),$

So we can infer: $\text{Crown}(K) \wedge \text{OnHead}(K, \text{John}),$ as long as K does not appear in the knowledge base.

- **The above used K is a constant symbol, which is called Skolem constant.**
- The Existential instantiation is a special case of Skolemization process.
- Example2: Given Statement: "At least one student in the class speaks in Sanskrit."
- Inference: "Ram is a student in the class who speaks in Sanskrit."

6. Existential introduction

- Existential introduction, alternatively referred to as existential generalization, is a valid inference rule within first-order logic. This rule posits that when an element, denoted as c , exists within the universe of discourse and possesses a particular property P , we are entitled to conclude that there exists an element within the universe that shares the same property P . In essence, it allows us to extend our knowledge from specific instances to broader assertions about the existence of such elements within the defined universe.

- It can be represented as: $\frac{P(c)}{\exists xP(x)}$
- **Example:** "Priyanka got good marks in English." "Therefore, someone got good marks in English."

2.4.5 Resolution in FOL :

- Resolution is a powerful technique used when multiple statements are provided, and the goal is to prove a conclusion based on these statements.
- Unification, a crucial concept in resolution, plays a significant role in the process.
- Resolution serves as a singular and efficient inference rule, particularly effective when working with conjunctive normal form or clausal form representations of logical statements. It simplifies and streamlines the process of logical deduction.
- **Clause:** A disjunction of literals, which represents an atomic sentence, is referred to as a clause. Specifically, when a clause consists of a single literal, it is known as a unit clause.
- **Conjunctive Normal Form:** A sentence that is expressed as a conjunction of clauses is described as being in conjunctive normal form (CNF). In CNF, a sentence is formed by combining multiple clauses with the logical AND (conjunction) operator.
- **Steps for Resolution:**
 - Conversion of facts into first-order logic.
 - Convert FOL statements into CNF
 - Negate the statement which needs to prove (proof by contradiction)
 - Draw resolution graph (unification).
 - To gain a deeper understanding of the aforementioned steps, we will explore an example where we apply the resolution technique.
 - Example:
 1. John likes all kind of food.
 2. Apple and vegetable are food
 3. Anything anyone eats and not killed is food.
 4. Anil eats peanuts and still alive
 5. Harry eats everything that Anil eats. Prove by resolution that :
 6. John likes peanuts.

Step-1: Conversion of Facts into FOL

In the first step, we will convert all the given statements into its first order logic.

- $\forall x: \text{food}(x) \rightarrow \text{likes}(\text{John}, x)$
 - $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$
 - $\forall x \forall y: \text{eats}(x, y) \wedge \neg \text{killed}(x) \rightarrow \text{food}(y)$
 - $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil}).$
 - $\forall x : \text{eats}(\text{Anil}, x) \rightarrow \text{eats}(\text{Harry}, x)$
 - $\forall x: \neg \text{killed}(x) \rightarrow \text{alive}(x)$
 - $\forall x: \text{alive}(x) \rightarrow \neg \text{killed}(x)$
 - $\text{likes}(\text{John}, \text{Peanuts})$
- } **added predicates.**

Step 2: Conversion of FOL into CNF

In First order logic resolution, it is required to convert the FOL into CNF as CNF form makes easier for resolution proofs.

- **Eliminate all implication (\rightarrow) and rewrite**

1. $\forall x \neg \text{food}(x) \vee \text{likes}(\text{John}, x)$
2. $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$
3. $\forall x \forall y \neg [\text{eats}(x, y) \wedge \neg \text{killed}(x)] \vee \text{food}(y)$
4. $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$
5. $\forall x \neg \text{eats}(\text{Anil}, x) \vee \text{eats}(\text{Harry}, x)$
6. $\forall x \neg [\neg \text{killed}(x)] \vee \text{alive}(x)$
7. $\forall x \neg \text{alive}(x) \vee \neg \text{killed}(x)$
8. $\text{likes}(\text{John}, \text{Peanuts})$.

- **Move negation (\neg) inwards and rewrite**

1. $\forall x \neg \text{food}(x) \vee \text{likes}(\text{John}, x)$
2. $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$
3. $\forall x \forall y \neg \text{eats}(x, y) \vee \text{killed}(x) \vee \text{food}(y)$
4. $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$
5. $\forall x \neg \text{eats}(\text{Anil}, x) \vee \text{eats}(\text{Harry}, x)$
6. $\forall x \neg \text{killed}(x) \vee \text{alive}(x)$
7. $\forall x \neg \text{alive}(x) \vee \neg \text{killed}(x)$
8. $\text{likes}(\text{John}, \text{Peanuts})$.

- **Rename variables or standardize variables**

1. $\forall x \neg \text{food}(x) \vee \text{likes}(\text{John}, x)$
2. $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$
3. $\forall y \forall z \neg \text{eats}(y, z) \vee \text{killed}(y) \vee \text{food}(z)$
4. $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$
5. $\forall w \neg \text{eats}(\text{Anil}, w) \vee \text{eats}(\text{Harry}, w)$
6. $\forall g \neg \text{killed}(g) \vee \text{alive}(g)$
7. $\forall k \neg \text{alive}(k) \vee \neg \text{killed}(k)$
8. $\text{likes}(\text{John}, \text{Peanuts})$.

- **Eliminate existential instantiation quantifier by elimination. :**

In this step, we typically use Skolemization to eliminate existential quantifiers (\exists). However, since the example problem lacks such quantifiers, there are no changes to the statements in this step. Skolemization is specifically applied to handle existential quantifiers when present [3].

- **Drop Universal quantifiers:**

In this step, we can omit all universal quantifiers since the statements are not implicitly quantified, rendering them unnecessary.

1. $\neg \text{food}(x) \vee \text{likes}(\text{John}, x)$
2. $\text{food}(\text{Apple})$
3. $\text{food}(\text{vegetables})$
4. $\neg \text{eats}(y, z) \vee \text{killed}(y) \vee \text{food}(z)$
5. $\text{eats}(\text{Anil}, \text{Peanuts})$
6. $\text{alive}(\text{Anil})$
7. $\neg \text{eats}(\text{Anil}, w) \vee \text{eats}(\text{Harry}, w)$
8. $\text{killed}(g) \vee \text{alive}(g)$
9. $\neg \text{alive}(k) \vee \neg \text{killed}(k)$
10. $\text{likes}(\text{John}, \text{Peanuts})$.

- Statements " $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$ " and " $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$ " can be written in two separate statements.
- **Distribute conjunction \wedge over disjunction \vee .**

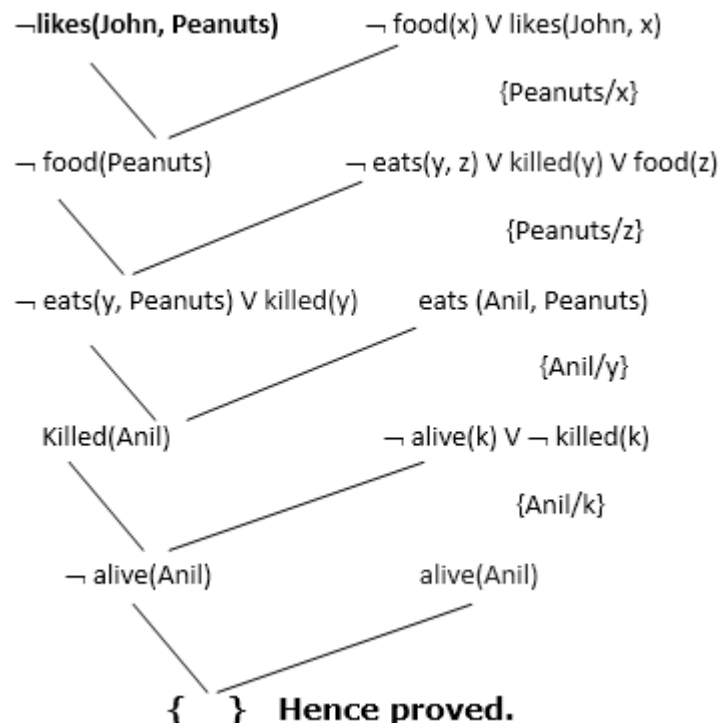
This step will not make any change in this problem.

Step-3: Negate the statement to be proved

In this statement, we will apply negation to the conclusion statements, which will be written as $\neg \text{likes}(\text{John}, \text{Peanuts})$

Step-4: Draw Resolution graph:

Now in this step, we will solve the problem by resolution tree using substitution. For the above problem, it will be given as follows:



Hence the negation of the conclusion has been proved as a complete contradiction with the given set of statements.

Explanation of Resolution graph:

- In the first step of resolution graph, $\neg \text{likes}(\text{John}, \text{Peanuts})$, and $\text{likes}(\text{John}, x)$ get resolved(canceled) by substitution of $\{\text{Peanuts}/x\}$, and we are left with $\neg \text{food}(\text{Peanuts})$
- In the second step of the resolution graph, $\neg \text{food}(\text{Peanuts})$, and $\text{food}(z)$ get resolved (cancelled) by substitution of $\{\text{Peanuts}/z\}$, and we are left with $\neg \text{eats}(y, \text{Peanuts}) \vee \text{killed}(y)$.
- In the third step of the resolution graph, $\neg \text{eats}(y, \text{Peanuts})$ and $\text{eats}(\text{Anil}, \text{Peanuts})$ get resolved by substitution $\{\text{Anil}/y\}$, and we are left with $\text{Killed}(\text{Anil})$.
- In the fourth step of the resolution graph, $\text{Killed}(\text{Anil})$ and $\neg \text{killed}(k)$ get resolve by substitution $\{\text{Anil}/k\}$, and we are left with $\neg \text{alive}(\text{Anil})$.
- In the last step of the resolution graph $\neg \text{alive}(\text{Anil})$ and $\text{alive}(\text{Anil})$ get resolved.

2.8 SUMMARY

In summary,

- We have explored in this unit the concept of knowledge representation in Artificial intelligence. Knowledge representation in AI is the process of designing structured models and formats to store and manage information in a manner comprehensible to machines. These representations serve as the foundation for AI systems to engage in reasoning, interpretation, and the manipulation of knowledge, empowering them to solve problems, make informed decisions, and exhibit intelligent behaviour.
- AI employs a range of techniques and formalisms, such as semantic networks, frames, and first-order predicate logic, to structure and organize information, making it amenable to computational processes and inference. Effective knowledge representation is crucial for developing AI applications that replicate human-like cognitive functions and support intelligent actions.
- In conclusion, you have also delved into the concepts of First-order logic, often referred to as first-order predicate logic (FOL), stands as a cornerstone in the realm of mathematical logic and artificial intelligence. This sophisticated formal system plays a pivotal role in the representation and logical inference of knowledge, facts, and relationships with precision and structure.

2.9 TERMINAL QUESTIONS

1. What is knowledge representation in artificial intelligence? Explain cycles of knowledge representation.
2. Explain the different types of knowledge and the requirements of knowledge representation in artificial intelligence.
3. Describes various is knowledge representation models in artificial intelligence.
4. What is First-Order logic? Describe inference in First-Order Logic.
5. Explain Resolution in First-Order Logic.

2.10 BIBLIOGRAPHY

1. MDATA: A New Knowledge Representation Model: Theory, Methods and Applications. (2021). Germany: Springer International Publishing.
2. Knowledge-representation. <https://www.edureka.co/blog/knowledge-representation-in-ai/>. Accessed on 13-10-23
3. First order logic. <https://www.javatpoint.com/first-order-logic-in-artificial-intelligence>. Accessed on 15-10-23.
4. Handbook of Knowledge Representation. (2008). Netherlands: Elsevier Science.
5. Smullyan, R. M., Smullyan, R. R. (2012). First-Order Logic. Germany: Springer Berlin Heidelberg.
6. First order logic. <https://www.codingninjas.com/studio/library/first-order-logic-in-artificial-intelligence>. Accessed on 14-10-23

UNIT-III INTRODUCTION TO REASONING AND SOFT COMPUTING

- 3.1 Introduction
- 3.2 Objective
- 3.3 Reasoning in AI
 - 3.3.1 Reasoning
 - 3.3.2 Types of Reasoning
- 3.4 Probabilistic reasoning in Artificial intelligence
 - 3.4.1 Probabilistic reasoning
 - 3.4.2 Need of probabilistic reasoning in AI
 - 3.4.3 Bayes' theorem
- 3.5 Bayesian Belief Network in artificial intelligence
- 3.6 Soft Computing
 - 3.6.1 What is Soft Computing?
 - 3.6.2 Characteristics of soft computing
 - 3.6.3 Soft computing paradigm
 - 3.6.4 Applications of soft computing
- 3.7 Pattern Recognition
 - 3.7.1 Pattern
 - 3.7.2 Tools for Machine Learning Pattern Recognition
 - 3.7.3 Application of pattern recognition
 - 3.7.4 Pattern association and mapping
- 3.8 Summary
- 3.9 Terminal Questions
- Bibliography

3.1 INTRODUCTION

In this module, we delve into the fundamental concepts of reasoning within the framework of First-Order Logic (FOL) and Soft Computing. These are key pillars in the domain of artificial intelligence and knowledge representation, and their combined understanding plays a pivotal role in solving complex problems with a blend of logical precision and computational flexibility.

3.2 OBJECTIVES

After studying this chapter, you should be able to:

- Familiar with the reasoning within the framework of First-Order Logic (FOL).
- Understand the concept of Bay's theorem, Bayesian network, and dependency network.
- Described the Soft Computing paradigm and pattern recognition

3.3 REASONING IN AI

3.3.1 Reasoning

- Reasoning is the cognitive process of drawing logical conclusions and making predictions based on existing knowledge, facts, and beliefs. In simpler terms, it can be described as a method for deducing facts from the information at hand.
- In artificial intelligence, reasoning plays a crucial role in enabling machines to emulate human-like rational thinking and perform tasks in a manner that mimics human cognition. This allows machines to approach problem-solving and decision-making in a way that is akin to human thought processes [1].

3.3.2 Types of Reasoning

- In artificial intelligence, reasoning can be categorized into several different types, each of which serves specific purposes and is used in various AI applications. Here are some common categories of reasoning in AI [1, 3]:
 - Deductive reasoning
 - Inductive reasoning
 - Abductive reasoning
 - Common Sense Reasoning
 - Monotonic Reasoning
 - Non-monotonic Reasoning
- **Deductive reasoning:**
 - Deductive reasoning derives new facts from logically connected existing information, ensuring the truth of the conclusion when the premises are true.
 - It's a key component of AI's propositional logic and is often called 'top-down reasoning' in contrast to inductive reasoning.
 - The truth of the premises guarantees the truth of the conclusion, typically moving from general premises to specific conclusions.
 - **Example:**
 - **Premise-1:** All the human eats veggies
 - **Premise-2:** Ramesh is human.
 - **Conclusion:** Ramesh eats veggies.
- **Inductive Reasoning:**
 - Inductive reasoning generalizes from limited data to form broad conclusions. It's a type of propositional logic known as cause-effect or bottom-up reasoning. Unlike deductive reasoning, inductive reasoning provides probable, not guaranteed, support for its conclusions.
 - **Example:**

- **Premise:** All of the pigeons we have seen in the zoo are white.
- **Conclusion:** Therefore, we can expect all the pigeons to be white.
- **Abductive reasoning:**
 - Abductive reasoning begins with one or more observations and aims to discover the most plausible explanation. It's an extension of deductive reasoning but doesn't ensure a guaranteed conclusion.
 - Example:
 - Implication:** Cricket ground is wet if it is raining
 - Axiom:** Cricket ground is wet.
 - Conclusion It is raining.
- **Common Sense Reasoning**
 - Common sense reasoning, informally based on personal experiences, emulates human presumptions about daily events. It depends on sound judgment rather than strict logic, operating with heuristic knowledge and rules.
 - **Example:**
 - One person can be at one place at a time.
 - If I put my hand in a fire, then it will burn.
 - These two statements illustrate common sense reasoning, which is easily grasped and assumed by the human mind.
- **Monotonic Reasoning:**
 - Monotonic reasoning keeps conclusions consistent when new information is added. It doesn't decrease the set of derivable propositions. While suitable for conventional reasoning systems, it's impractical for real-time systems where facts change. Theorems proving is a case of monotonic reasoning.
 - In monotonic reasoning, previous proofs always remain valid. Deductions made from existing facts are enduringly valid.
 - Monotonic reasoning falls short in representing real-world scenarios because it requires facts to be unchanging. It cannot handle hypothetical knowledge or incorporate new information from the dynamic real world, as it relies solely on existing proofs.
 - **Example:**
 - Earth revolves around the Sun.
 - It's an immutable fact, unaffected by the addition of other sentences to the knowledge base, such as "The moon revolves around the earth" or "Earth is not round."
- **Non-monotonic Reasoning:**
 - Non-monotonic reasoning allows for conclusions that can be invalidated by adding new information to the knowledge base.

- It deals with incomplete and uncertain models, making it suitable for representing "human perceptions for various things in daily life."
- **Example:** Letsuppose the knowledge base contains the following knowledge:
 - Birds can fly
 - Penguins cannot fly
 - Pitty is a bird

So from the above sentences, we can conclude that **Pitty can fly**.

However, if we add one another sentence into knowledge base "**Pitty is a penguin**", which concludes "**Pitty cannot fly**", so it invalidates the above conclusion.

- Non-monotonic reasoning is indeed valuable for real-world systems like robot navigation. It allows us to incorporate probabilistic facts and make informed assumptions, which is crucial in handling the uncertainties and dynamic nature of real-world environments.
- In non-monotonic reasoning, adding new sentences can invalidate old facts, making it unsuitable for theorem proving where facts remain consistent.

3.4 Probabilistic reasoning in Artificial intelligence

3.4.1 Probabilistic reasoning

- Probabilistic reasoning employs probability theory to model and manage uncertainty, allowing decisions to be made based on probabilities. Common tools for probabilistic reasoning in AI include Bayesian networks and Markov decision processes [2].
- We employ probability in probabilistic reasoning because it offers a systematic approach to managing uncertainty, which may arise from various sources, including incomplete information and human limitations such as laziness or ignorance.
- In the real world, uncertainty abounds in scenarios like "Today's weather," "individual behaviour in specific situations," or "sports match outcomes." These situations call for probabilistic reasoning, as they involve events that we can assume but not guarantee.
- Uncertainty: we've explored knowledge representation using first-order logic and propositional logic with certainty, where we were certain about the predicates. However, in situations where we're unsure about the truth of a predicate, we encounter uncertainty. To represent uncertain knowledge effectively, we require uncertain reasoning or probabilistic reasoning methods, which can handle and quantify this uncertainty.
- **Causes of uncertainty:**
 - Following are some leading causes of uncertainty to occur in the real world.
 1. Information occurred from unreliable sources.
 2. Experimental Errors
 3. Equipment fault
 4. Temperature variation
 5. Climate change.

3.4.2 Need of probabilistic reasoning in AI:

- In the presence of unpredictable results,.
- When the specifications or the number of possible predicates become overwhelming to manage.
- In cases of unexpected errors during an experiment.
- **Probabilistic reasoning** offers two approaches for addressing problems with uncertain knowledge [2,3]:
 - Bayes' rule
 - Bayesian Statistics

Since probabilistic reasoning relies on probability and associated concepts, it's important to first grasp some common terms to facilitate a better understanding of probabilistic reasoning.

- **Probability:** Probability can be defined as the likelihood of an uncertain event happening, represented numerically. It is a measure of the chance that an event will occur, with its value always ranging between 0 and 1, indicating the extent of uncertainty, where 0 signifies impossibility and 1 signifies certainty.
 1. $0 \leq P(X) \leq 1$, where $P(X)$ is the probability of an event X .
 2. $P(X) = 0$, indicates total uncertainty in an event X .
 3. $P(X) = 1$, indicates total certainty in an event X .

The probability of an uncertain event can be determined using the following formula.

$$\text{Probability of occurrence} = \frac{\text{Number of desired outcomes}}{\text{Total number of outcomes}}$$

- $P(\neg X)$ = probability of a not happening event X .
- $P(\neg X) + P(X) = 1$.
- **Event:** Each possible outcome of a variable is called an event.
- **Sample space:** The set comprising all possible events is termed the sample space.
- **Random variables:** Random variables are employed to represent events and objects in the real world.
- **Prior probability:** The prior probability of an event is the probability calculated before taking new information into account.
- **Posterior Probability:** The probability calculated after considering all available evidence or information is referred to as posterior probability. It represents a combination of the prior probability and new information.
- **Conditional probability:**
 - Conditional probability is the likelihood of an event occurring given that another event has already happened.
 - Suppose we aim to calculate the probability of event A occurring given that event B

has already taken place. This can be expressed as:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

- Where $P(A \cap B)$ = Joint probability of a and B
- $P(B)$ = Marginal probability of B.
- **Example :** In a class, there are 70% of the students who like English and 40% of the students who likes English and mathematics, and then what is the percent of students those who like English also like mathematics?
- **Solution:**
 - Let, A is an event that a student likes Mathematics
 - B is an event that a student likes English.

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{0.4}{0.7} = 57\%$$

-
- Hence, 57% are the students who like English also like Mathematics.

3.4.3 Bayes' theorem:

- Bayes' theorem, also known as Bayes' rule, Bayes' law, or Bayesian reasoning, is a fundamental concept in probability theory. It establishes a connection between the conditional probability of two random events and their marginal probabilities. It is a crucial tool for determining the probability of an event in the presence of uncertain knowledge.
- Bayes' theorem provides a method to calculate the value of $P(B|A)$ when we have knowledge of $P(A|B)$. This theorem enables the adjustment of probability predictions for an event based on the observation of new real-world information.
- Example: If cancer corresponds to one's age then by using Bayes' theorem, we can determine the probability of cancer more accurately with the help of age.
- Bayes' theorem can be derived by applying the product rule and utilizing conditional probability, specifically relating to event A given known event B.

We can express the product rule as follows:

$$P(A \cap B) = P(A|B) P(B) \text{ or } \text{----(1)}$$

Likewise, the probability of event B given the knowledge of event A:

$$P(A \cap B) = P(B|A) P(A) \text{ ----(2)}$$

Equating right hand side of both the equations, we will get:

$$P(A|B) = \frac{P(B|A) P(A)}{P(B)} \quad \text{....(a)}$$

- The equation (a) is commonly referred to as Bayes' rule or Bayes' theorem. It serves as a foundational concept for many modern AI systems, particularly in the context of probabilistic inference.

- Bayes' theorem demonstrates a straightforward connection between joint and conditional probabilities. In this context:
- $P(A|B)$ is known as the posterior, representing the probability of hypothesis A given that evidence B has occurred.
- - $P(B|A)$ is termed the likelihood, indicating the probability of the evidence given that the hypothesis is true.
- - $P(A)$ is the prior probability, signifying the probability of the hypothesis before considering any evidence.
- - $P(B)$ is the marginal probability, representing the pure probability of the evidence itself.

In the general case, we can express $P(B)$ as the sum of $P(A_i)$ multiplied by $P(B|A_i)$, making Bayes' rule take the form:

$$P(A_i|B) = \frac{P(A_i) \cdot P(B|A_i)}{\sum_{i=1}^k P(A_i) \cdot P(B|A_i)}$$

Where $A_1, A_2, A_3, \dots, A_n$ represents a set of events that are mutually exclusive and collectively exhaustive, covering all possible scenarios.

- Example-1:
- Question: From a standard deck of playing cards, a single card is drawn. The probability that the card is king is $4/52$, then calculate posterior probability $P(\text{King}|\text{Face})$, which means the drawn face card is a king card.
- Solution:

$$P(\text{king} | \text{face}) = \frac{P(\text{Face}|\text{king}) \cdot P(\text{King})}{P(\text{Face})} \dots\dots(i)$$

- $P(\text{king})$: probability that the card is King= $4/52 = 1/13$
- $P(\text{face})$: probability that a card is a face card= $3/13$
- $P(\text{Face}|\text{King})$: probability of face card when we assume it is a king = 1
- Putting all values in equation (i) we will get:

$$P(\text{king} | \text{face}) = \frac{1 * (\frac{1}{13})}{(\frac{3}{13})} = 1/3, \text{ it is a probability that a face card is a king card.}$$

- **Application of Bayes' theorem** in Artificial intelligence : Following are some applications of Bayes' theorem:
- It is used to calculate the next step of the robot when the already executed step is given.
- Bayes' theorem is helpful in weather forecasting.
- It can solve the Monty Hall problem.

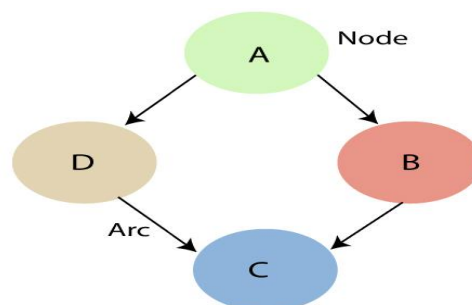
- In engineering and industrial applications, Bayes' theorem is a valuable tool for diagnosing faults and anomalies in machinery and complex systems by updating the likelihood of specific issues based on new data and observations.
- Traders utilize Bayesian methods for risk assessment and predicting stock market trends by incorporating probabilistic reasoning and updating their predictions based on the latest market information.

Self-Evaluations

- What is Bayes' theorem? Explain with suitable example?
- What are the leading causes of uncertainty to occur in the real world?
- Write short notes on common sense reasoning, monotonic reasoning, and probabilistic reasoning in AI.

3.5 Bayesian Belief Network in artificial intelligence

- A Bayesian belief network is a critical computer technology for managing probabilistic events and addressing problems involving uncertainty. We can define a Bayesian network as follows[1]:
- A Bayesian network is a probabilistic graphical model that represents a collection of variables and their conditional dependencies through a directed acyclic graph.
- It is also known as a Bayes network, belief network, decision network, or Bayesian model.
- Bayesian networks are probabilistic in nature because they are constructed based on probability distributions, and they leverage probability theory for tasks such as prediction and anomaly detection.
- Real-world applications often involve probabilistic elements, and Bayesian networks are valuable tools for representing relationships between multiple events. They find use in a wide range of tasks, including prediction, anomaly detection, diagnostics, automated insight generation, reasoning, time series prediction, and decision-making under conditions of uncertainty.
- Bayesian networks can be employed to construct models from both data and expert opinions. They typically consist of two main components:
 - **Directed Acyclic Graph**
 - **Table of conditional probabilities.**
- A Bayesian network graph is made up of nodes and Arcs (directed links), where:



- Each node in a Bayesian network corresponds to a random variable, which can be either continuous or discrete in nature.
- The arcs or directed arrows in a Bayesian network graph represent the causal relationships or conditional probabilities between random variables. These directed links or arrows connect pairs of nodes in the graph, indicating the dependencies between the associated variables.
- These links symbolize a direct influence from one node to another. When there is no directed link, it signifies that the nodes are mutually independent.
- In the diagram above, nodes A, B, C, and D correspond to random variables in the network graph.
- When examining node B, which is connected to node A via a directed arrow, we refer to node A as the parent of node B.
- And node C is independent of node A.

3.6 Soft Computing

3.6.1. What is Soft Computing?

1. The concept of soft computing was introduced in 1981 when Lotfi A. Zadeh published his seminal paper titled "What is Soft Computing" in the journal Soft Computing, which was later published by Springer-Verlag in 1997. Zadeh's definition of soft computing encompasses a multidisciplinary approach, which combines various fields, including Fuzzy Logic, Neuro-Computing, Evolutionary and Genetic Computing, and Probabilistic Computing[4].
2. Soft Computing represents the fusion of methodologies aimed at modelling and facilitating solutions for real-world problems that either cannot be accurately modelled mathematically or are too complex to do so. Its primary objective is to leverage the capacity to handle imprecision, uncertainty, approximate reasoning, and partial truth, allowing for decision-making processes that closely resemble human cognitive processes.

3.6.2 Characteristics of soft computing

- Soft computing offers an approximate yet precise solution for real-world problems.
- The algorithms of soft computing are adaptive, ensuring that the current process remains unaffected by changes in the environment.
- The concept of soft computing is grounded in learning from experimental data. This means that soft computing does not necessitate a mathematical model to address and solve problems.
- Soft computing is based on a combination of various techniques and methodologies, including fuzzy logic, genetic algorithms, machine learning, artificial neural networks (ANN), and expert systems.

3.6.3 Soft computing paradigm:

- According to Lotfi A. Zadeh, 1992 : “Soft Computing is an evolving computational approach that seeks to emulate the remarkable reasoning and learning abilities of the human mind within environments characterized by uncertainty and imprecision.”. Soft Computing comprises several computing paradigms, primarily [4]:
 - **Fuzzy Systems**, for knowledge representation via fuzzy If – Then rules
 - **Neural Networks**, for learning and adaptation

- **And Genetic Algorithms.** For evolutionary computation

These methodologies form the core of soft computing.

3.6.4 Applications of soft computing

There are numerous applications of soft computing in various domains. Here are some of them:

- Soft computing finds practical application in control systems, serving in various domains such as robotics, industrial process control, and adaptive control.
- Soft computing techniques are extensively utilized in gaming products, with notable applications in games such as Poker and Checkers.
- Indeed, soft computing techniques are also employed in kitchen appliances, including microwave ovens and rice cookers.
- soft computing is applied in a wide range of commonly used home appliances, including washing machines, heaters, refrigerators, and air conditioners.
- Image processing and data compression are popular and valuable applications of soft computing. These techniques help improve image quality, reduce data storage requirements, and enhance the efficiency of data transmission and processing.
- Soft computing techniques are also harnessed for handwriting recognition applications, where they aid in accurately converting handwritten text or characters into digital formats, facilitating various document-processing tasks.

3.7 Pattern Recognition

3.7.1 Patterns

- Patterns are omnipresent in the digital realm, whether perceived through physical observation or mathematical algorithms [5].
- **Example:** The colors on the clothes, speech pattern, etc. In computer science, a pattern is represented using vector feature values.
- Pattern recognition is the task of identifying patterns through machine learning algorithms. It involves categorizing data using existing knowledge or statistical insights obtained from patterns and their representations. Indeed, pattern recognition finds application in various domains. Examples include speech recognition, speaker identification, multimedia document recognition (MDR), and automatic medical diagnosis.
- In a standard pattern recognition application, raw data is pre-processed and transformed into a machine-readable format. Pattern recognition encompasses the tasks of classifying and clustering patterns.
 - In classification, a suitable class label is assigned to a pattern using abstractions derived from a training dataset or domain expertise.
 - Clustering creates a data partition that aids in decision-making for a specific task.

3.7.2 Tools for Machine Learning Pattern Recognition:

- Several tools are available for machine learning pattern recognition [5].
 - **Amazon Lex**, an open-source offering by Amazon, allows for the development of ChatBots and conversation agents that utilize text and speech recognition.

- **Google Cloud AutoML** leverages neural networks, including recurrent neural networks and reinforcement learning, to create high-quality machine learning models with minimal prerequisites.
- **R-Studio** serves as an integrated development environment, utilizing the R programming language for the development and testing of pattern recognition models.
- **IBM Watson Studio**, provided by IBM, facilitates data analysis and machine learning, empowering users to construct and deploy machine learning models.
- **Microsoft Azure Machine Learning Studio**, a Microsoft product, simplifies the process with a drag-and-drop graphical user interface for building and deploying machine learning models.

3.7.3 Application of pattern recognition:

It serves as diverse applications in various domains.

- It aids in trend analysis, allowing for the prediction of future trends based on historical data, like sales forecasting.
- Additionally, it plays a critical role in providing assistance, particularly in enhancing safety through obstacle prediction and alerts.
- E-commerce platforms employ it for visual search engines and personalized product recommendations.
- In computer vision, it enables the comparison of input images or videos with extensive databases to identify similar patterns, as seen in medical image analysis for cancer detection.
- Finally, biometric devices rely on face recognition and fingerprint detection technologies, driven by machine learning algorithms, to ensure secure authentication and safety.

3.7.4 Pattern association and mapping:

- It involve the identification and establishment of connections between different patterns or data points. These methods are commonly applied in fields like machine learning, data analysis, and cognitive science to reveal relationships, similarities, or correlations among datasets. Such analyses can yield valuable insights and support decision-making in diverse applications.

3.8 SUMMARY

In summary,

- We have explored in this unit the concept of reasoning in Artificial Intelligence. Reasoning is the cognitive process of deriving logical conclusions and predictions from existing knowledge and information. In artificial intelligence, it is vital for enabling machines to mimic human-like thinking and perform tasks resembling human cognition in problem solving and decision-making.
- We've also examined different types of reasoning, including probabilistic reasoning, which utilizes probability theory to model and handle uncertainty, enabling decision-making based on probabilities. Key tools for probabilistic reasoning in AI encompass Bayesian networks and Markov decision processes.
- In the final part of our exploration, we covered the concept of soft computing, which involves computational techniques designed to handle complex and uncertain information, and pattern recognition. Soft computing employs approaches like fuzzy logic, neural networks, and genetic

algorithms, suitable for tasks where precise mathematical modelling may not be applicable. Pattern recognition, a key component of artificial intelligence, teaches machines to identify and interpret patterns in data, making it valuable in applications ranging from image and speech recognition to fraud detection and medical diagnosis.

3.9 TERMINAL QUESTIONS

1. What is reasoning in AI? Discuss different types of reasoning used in AI.
2. Explain the concept of probabilistic reasoning in Artificial intelligence with suitable example.
3. Describe bay's theorem and its application.
4. Explain Bayesian Belief Network in artificial intelligence.
5. What is Soft Computing? Discuss its application.
6. Write short notes on pattern recognition and soft computing.

3.10 BIBLIOGRAPHY

1. Nakatsu, R. T. (2009). *Diagrammatic reasoning in AI*. John Wiley & Sons.
2. Probabilistic reasoning. <https://www.javatpoint.com/probabilistic-reasoning-in-artificial-intelligence>. Accessed on 20-10-23
3. Reasoning in AI. <https://www.geeksforgeeks.org/types-of-reasoning-in-artificial-intelligence>. Accessed on 20-10-23
4. Ibrahim, D. (2016). An overview of soft computing. *Procedia Computer Science*, 102, 34-38.
5. Theodoridis, S., & Koutroumbas, K. (2006). *Pattern recognition*. Elsevier.



Master of Computer Science

MCS-117(N) Soft Computing

Block-2	FUZZY SET THEORY	51-91
UNIT-1	Introduction to Fuzzy Sets	51-61
UNIT-2	Fuzzy Logic	62 -78
UNIT-3	Fuzzy System	79-91

Course Design Committee

Prof. Ashutosh Gupta Director, School of Science, UPRTOU Allahabad	Chairman
Prof. Suneeta Agarwal Dept. of Computer Science & Engineering Motilal Nehru National Institute of Technology Allahabad	Member
Dr. Upendra Nath Tripathi Associate Professor DeenDayalUpadhyay Gorakhpur University, Gorakhpur	Member
Dr. Ashish Khare Associate Professor Dept. of Computer Science, University of Allahabad, Prayagraj	Member
Ms. Marisha Assistant Professor (Computer Science) School of Science, UPRTOU Allahabad	Member
Mr. Manoj Kumar Balwant Assistant Professor (Computer Science) School of Science, UPRTOU Allahabad	Member

Course Preparation Committee

Mr. Shivendu Mishra Assistant Professor Dept. of Information Technology Rajkiya Engineering College Ambedkar Nagar U.P. India-224122	Author (Block 1, Block 2: Unit 2, 3, Block 3, & 4)
Dr. Gunjan Singh Assistant Professor (Block 2: Unit 1) Faculty of management and computer application, RBS College, Khandari, Agra-282002.	Author
Prof. Manu Pratap Singh Professor, Department of Computer Science Engineering Dr. Bhimrao Ambedkar University, Agra	Editor
Mr. Manoj Kumar Balwant Assistant Professor (computer science), School of Sciences, UPRTOU Allahabad	Coordinator

© UPRTOU, Prayagraj. 2023

First Edition: July 2023

ISBN: 978-93-48270-40-5

All rights are reserved. No part of this work may be reproduced in any form, by mimeograph or any other means, without permission in writing from the Uttar Pradesh Rajarshi Tandon Open University.



<http://creativecommons.org/licenses/by-sa/4.0/>

Creative Commons Attribution-Share Alike 4.0 International License

Printed and Published by Col. Vinay Kumar, Registrar, Uttar Pradesh Rajarshi Tandon Open University, 2024.

Printed By: Cygnus Information Solution Pvt. Ltd, Lodha Supremus Saki Vihar Road Andheri East, Mumbai.

BLOCK-2 INTRODUCTION

This block introduces fuzzy logic, highlighting its necessity and advantages over traditional crisp logic in managing uncertainties and ambiguities in real-world data. The initial unit covers the foundational concepts of fuzzy logic, including basic terminology, properties, operations, and types of membership functions and fuzzy sets. The second unit of this block delves into the practical aspects of fuzzy logic, focusing on the creation and setup of fuzzy membership functions, the application of if-then rules for fuzzy reasoning, and the design of effective fuzzy rules. It also explores fuzzy relations, the fuzzy Cartesian product, nested fuzzy relations, and the extension principle. The final unit describes how fuzzy logic addresses uncertainty and imprecision in complex systems. It covers fuzzy rule base systems, fuzzy inference systems, and defuzzification methods that convert fuzzy results into crisp values. The unit further examines fuzzy control systems and their applications in fields such as industrial automation and consumer electronics, demonstrating the practical and versatile nature of fuzzy logic in real-world scenarios.

Unit I : Introduction to Fuzzy Sets

- 1.0 Introduction
- 1.1 Objectives
- 1.2 Fuzzy logic Concept
- 1.3 Crisp set theory
- 1.4 Fuzzy set theory
 - 1.4.1 Features of membership function
 - 1.4.2 Types of membership functions
- 1.5 Properties of Fuzzy sets
- 1.6 Fuzzy set operations
- 1.7 Types of Fuzzy set
- 1.8 Summary

1.0 INTRODUCTION

This unit is an introductory unit. It enables you to know the necessity and importance of fuzzy logic, as compared to the traditional crisp logic, to deal with uncertainties and ambiguities occur in real-world data. This unit present an overview of fuzzy logic including its basic terminology, properties and operations. It also presents various types of membership functions and fuzzy sets to develop a clear understanding of the concept of fuzzy logic.

1.1 OBJECTIVES

After the end of this unit, you should be able to:

1. Understand the concept of fuzzy logic
2. Compare fuzzy logic with crisp logic
3. Define fuzzy set
4. Know basic properties of fuzzy sets
5. Understand the idea and types of membership function
6. Apply various operations on fuzzy sets

1.2 FUZZY LOGIC CONCEPT

Human beings deal with uncertainty in the form of inexact, incomplete and missing information in day-to-day real world activities. Uncertainty occurs due to the ***presence of ambiguity and vagueness in data***, therefore information may be partial and flexible in meaning. The structure and functioning of human brain enables it to deal and analyze this uncertain, fuzzy and partial information into a structured way; but for a machine it is still difficult to handle such information. The emergence of fuzzy logic can be stated as a successful attempt to apply a more ***human like way of thinking and reasoning*** in the programming of computers.

The idea of fuzzy logic was given by ***Lofti A. Zadeh*** in 1965 as a method to solve and model real-world problems deal with uncertain data. Classical or ***crisp logic*** is too strict to deal with such type of data, as it is based on the two valued ***Boolean logic*** such as ***yes/no, true/false***, etc. Strict behavior or pattern of crisp logic requires to choose a particular ***threshold value*** to categorize data

under consideration. This threshold value set the boundary for the data such that data within the boundary is considered as true, whereas data outside the boundary is considered as false. This property of crisp logic makes it inadequate to handle the uncertainties with human thinking and reasoning process. To overcome this limitation of crisp logic, the concept of fuzzy logic was emerged as it provides a way to describe human knowledge involving vague, ambiguous or fuzzy uncertainties. Fuzzy uncertainty refers to the resistance and flexibility in the meaning of a concept.

1.3 CRISP SET THEORY

Crisp logic is two-valued logic with sharp boundaries between members and non-members of a set. A crisp set can be defined as “*a collection of distinct objects in a given domain*”. E.g. a set of persons with age less than 40 years, a set of students of class XII, a set of soft computing books, set of people vaccinated with two doses of vaccine, etc. A crisp set can be represented by using following two methods:

- a) **List method:** Using this method, a set is represented by listing all of its members. It is used to represent set with finite number of elements.

E.g. Let A is a set of all positive even numbers less than 10.

A will be represented as:

$$A = \{2, 4, 6, 8\}$$

- b) **Rule method:** Using this method, a set is represented by declaring a property owned by all of its members. Generally, it is used to represent set with infinite number of elements.

e.g. Set A given in above example is represented as:

$$A = \{x \mid x \text{ is even and } 0 < x < 10\}$$

An entity or object either belongs or does not belong to the given set, i.e. an object or entity can be a member or nonmember of the given set. Thus, membership value of an object is 1 if it is a member of the set and 0 if it is not a member of the set. Thus, for crisp set A, membership of an object x is expressed as:

$$\chi_A(x) = \begin{cases} 1, & \text{if } x \in A \\ 0, & \text{if } x \notin A \end{cases}$$

where A is a set in universe U, and $\chi_A(x)$ is the membership of element x in A.

E.g. Set A is defined over universal set U as shown in figure. Membership of elements p, q, r and s is 1, 1, 0 and 0 respectively.

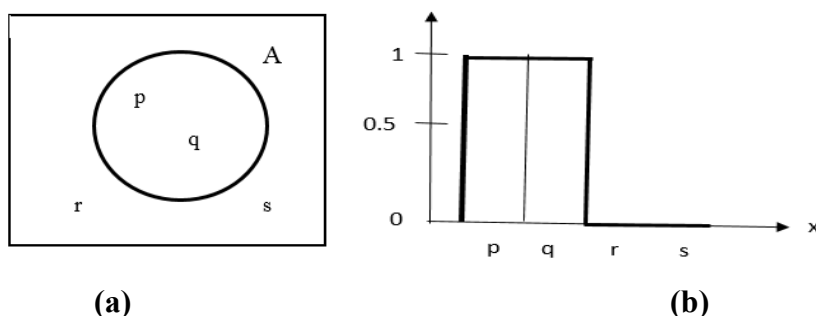


Figure 1.1: (a) Venn diagram of set A, (b) Graphical representation of membership of elements p, q, r and s.

1.4 FUZZY SET THEORY

Fuzzy logic can be thought of as an extension of crisp logic. In contrast to the two-valued crisp logic, fuzzy logic is a multi-valued logic which allows partial set membership of data by

defining the intermediate values between the sharp boundaries (membership or non-membership) of the set. In crisp logic results are returned in the form of 1(true) or 0 (false). In fuzzy logic 0 and 1 values are returned by operations when all fuzzy memberships are restricted to these values only. When there is not any restriction then fuzzy logic can also be based on partial truth or degrees of truth. Thus, fuzzy logic can also be termed as a superset of Crisp logic.

E.g. In Crisp logic, answer to the question - Is Priya beautiful? Is either yes (1) or no (0).

But in fuzzy logic, we may have more answers than yes or no such as Extremely beautiful, Very beautiful, Pretty, Simply beautiful, and so on.

Dr. Zadeh defined fuzzy sets as *“the sets on the universe U which can accommodate degrees of membership by extending the idea of crisp set, having sharp values 0 and 1 only and denoted by {0,1}, to the interval of real values, denoted by [0,1]”*. According to Zadeh the concept of fuzzy set allows partial membership i.e. fuzzy sets may have smooth boundaries. Membership of an element of fuzzy set is given by the degree of compatibility of the element with the description of the set. Membership is expressed between number 0 and 1, where 0 represents complete non-membership, 1 represents complete membership and value in-between 0 and 1 represents partial membership. e.g. Difference between representation of degree of membership for temperature using crisp logic and fuzzy logic is shown in following diagram:

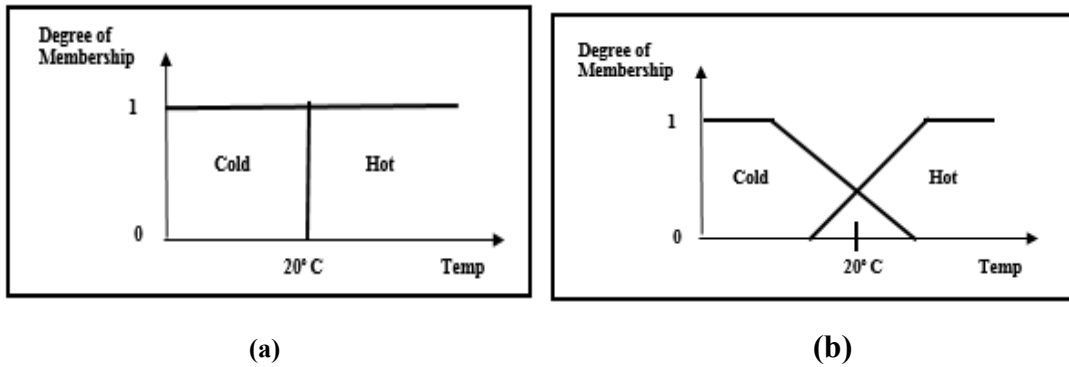


Figure 1.2: Representation of degree of membership of temperature using (a) crisp logic and (b) fuzzy logic

Membership in a fuzzy set need not be complete, member of one fuzzy set can also be a member of another set. A fuzzy set is described by its mapping from its universe of discourse into the interval [0,1]. Thus, a fuzzy set \tilde{A} can be defined *as a collection of ordered pairs such that*:

$$\tilde{A} = \{(x, \mu_{\tilde{A}}(x)) \mid x \in U\}$$

where U is universe of discourse and $\mu_{\tilde{A}}(x)$ is the degree of membership of a particular element x in \tilde{A} . $\mu_{\tilde{A}}(x)$ is also called **membership function** of \tilde{A} .

A membership function can be described for discrete as well as continuous values. When universe of discourse U is discrete and finite, fuzzy set is expressed as:

$$\tilde{A} = \left\{ \frac{\mu_{\tilde{A}}(x_1)}{x_1} + \frac{\mu_{\tilde{A}}(x_2)}{x_2} + \frac{\mu_{\tilde{A}}(x_3)}{x_3} + \dots \right\} = \left\{ \sum_{i=1}^n \frac{\mu_{\tilde{A}}(x_i)}{x_i} \right\}$$

E.g. For discrete values p, q, r, s, t as shown in following figure, membership of elements for

fuzzy set \tilde{A} can be represented is $\tilde{A} = \{(p, 1), (q, 1), (r, 0.5), (s, 0.5), (t, 0)\}$.

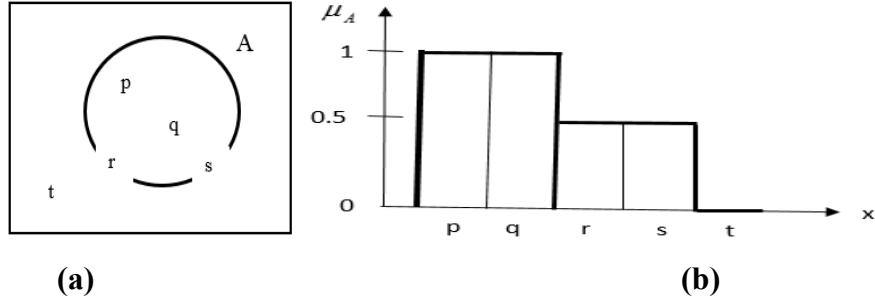


Figure 1.3: (a) Venn diagram representation of fuzzy set \tilde{A} , (b) Graphical representation of degree of membership of each discrete value of set \tilde{A}

When U is continuous and infinite, fuzzy set is expressed as:

$$\tilde{A} = \left\{ \int \frac{\mu_{\tilde{A}}(x)}{x} \right\}$$

E.g. Let \tilde{A} is a containing elements near to 0.

$$\tilde{A} = \int \frac{\mu_{\tilde{A}}(x)}{x} \quad \text{where} \quad \mu_{\tilde{A}}(x) = \frac{1}{1+x^2}$$

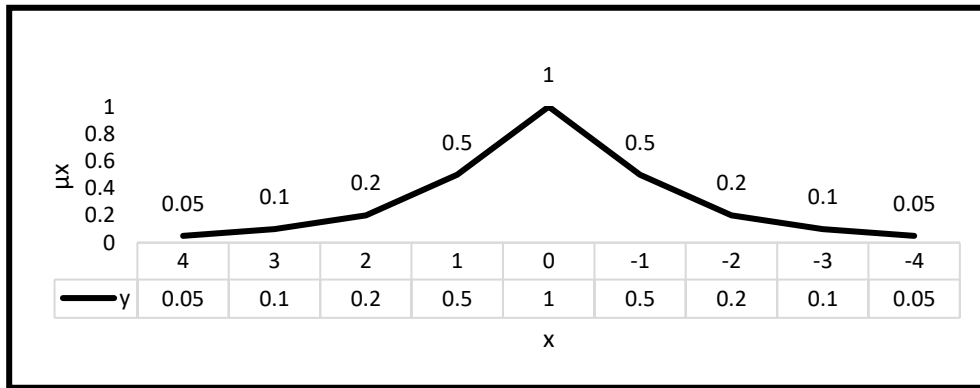


Figure 1.4: Graphical representation of degree of membership of continuous values of set \tilde{A} .

The maximum value of the membership function of a fuzzy set is called its height.

1.4.1 Features of Membership Function:

Three basic feature are:

- **Core:** It is the region described by the complete membership in the set \tilde{A} .

$$\text{Core}(\tilde{A}) = \{x \in U \mid \mu_{\tilde{A}}(x) = 1\}$$

- **Support:** It is the region described by a nonzero membership in the set \tilde{A} .

$$\text{Support}(\tilde{A}) = \{x \in U \mid \mu_{\tilde{A}}(x) > 0\}$$

- **Boundary:** It is the region described by nonzero but not complete membership in the set.

$$\text{Boundary}(\tilde{A}) = \{x \in U \mid 0 < \mu_{\tilde{A}}(x) < 1\}$$

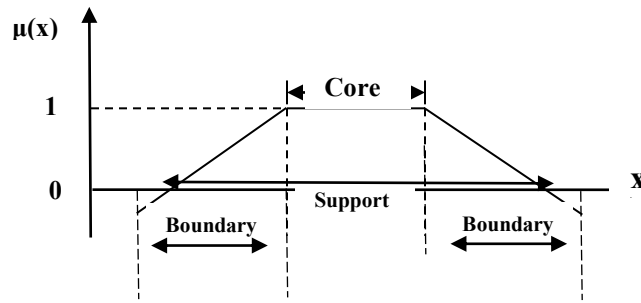
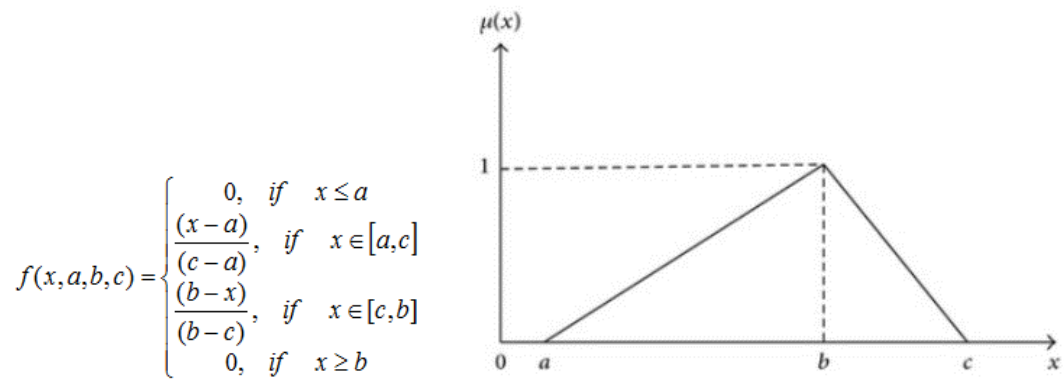


Figure 1.5: Features of membership function- core, support and boundary

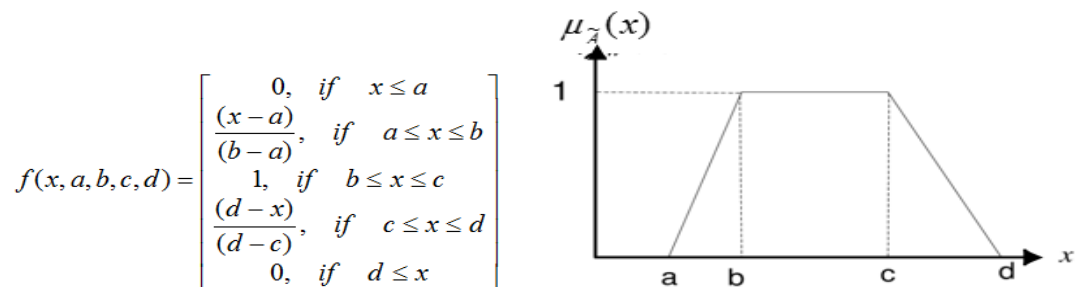
1.4.2 Types of Membership Functions

Most commonly used fuzzy membership functions are:

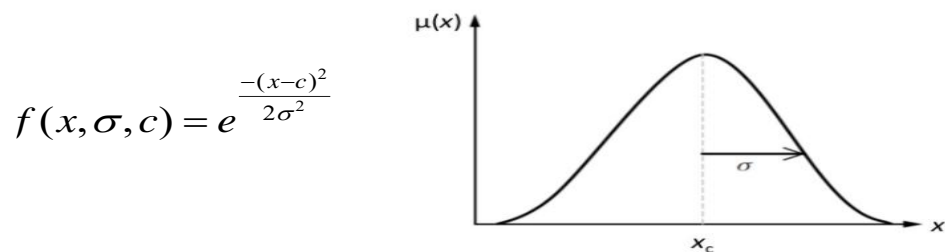
- a) **Triangular membership function:** It takes three scalar values a , b and c , where a , b , c represents the three points on x -axis of the triangular membership function such that $a < b < c$. Function is defined as follows.



- b) **Trapezoidal membership function:** It takes four scalar values a , b , c & d and is defined by the following expression.

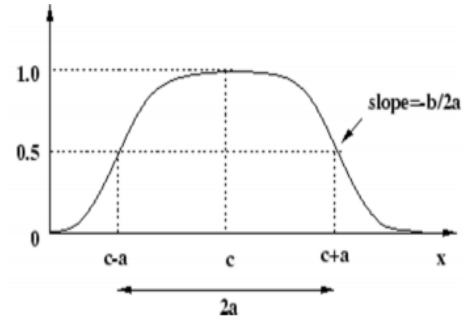


- c) **Gaussian membership function:** It takes two values σ and c , where σ represents width and c represents center of the membership function.



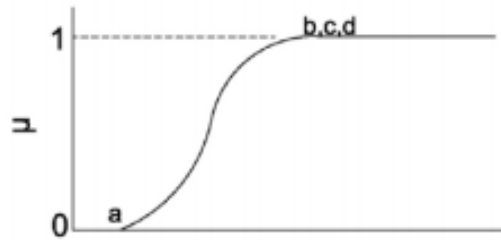
- d) **Generalized bell shaped membership function:** It takes three values a, b and c, where c is center of the curve and b is slope at the cusp or crossover points as shown in figure.

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



- e) **Sigmoid membership function:** It takes two parameters a and c, where a represents slope at cusp point c.

$$f(x, a, c) = \frac{1}{1 + e^{-a(x-c)}}$$



1.5 PROPERTIES OF FUZZY SETS:

Fuzzy sets possess same properties as crisp sets except for two properties – law of excluded middle and law of contradiction. Commonly used properties of fuzzy sets are as follows. Let \tilde{A} , \tilde{B} and \tilde{C} are fuzzy sets.

1. **Commutativity:**

$$\tilde{A} \cup \tilde{B} = \tilde{B} \cup \tilde{A}$$

$$\tilde{A} \cap \tilde{B} = \tilde{B} \cap \tilde{A}$$

2. **Associativity:**

$$\tilde{A} \cup (\tilde{B} \cap \tilde{C}) = (\tilde{A} \cup \tilde{B}) \cap \tilde{C}$$

$$\tilde{A} \cap (\tilde{B} \cup \tilde{C}) = (\tilde{A} \cap \tilde{B}) \cup \tilde{C}$$

3. **Distributivity :**

$$\tilde{A} \cup (\tilde{B} \cap \tilde{C}) = (\tilde{A} \cup \tilde{B}) \cap (\tilde{A} \cup \tilde{C})$$

$$\tilde{A} \cap (\tilde{B} \cup \tilde{C}) = (\tilde{A} \cap \tilde{B}) \cup (\tilde{A} \cap \tilde{C})$$

4. **Idempotence:**

$$\tilde{A} \cup \tilde{A} = \tilde{A}$$

$$\tilde{A} \cap \tilde{A} = \tilde{A}$$

5. **Identity:**

- (i) $\tilde{A} \cup \phi = \tilde{A}$, (ii) $\tilde{A} \cup \cup = \cup$
 (iii) $\tilde{A} \cap \phi = \phi$, (iv) $\tilde{A} \cap \cup = \tilde{A}$

(where \cup is universal set and is ϕ empty or null set.)

6. **Transitivity :**

If $\tilde{A} \subseteq \tilde{B} \subseteq \tilde{C}$, then $\tilde{A} \subseteq \tilde{C}$

7. **Involution:**

$$(\tilde{A}^c)^c = \tilde{A}$$

8. **Law of the Excluded Middle :**

$$\tilde{A} \cup \tilde{A}^c = \cup \quad (\text{where c represents contradiction of the set})$$

9. **Law of Contradiction :**

$$\tilde{A} \cap \tilde{A}^c = \phi$$

10. **De Morgan's law:**

$$(\tilde{A} \cup \tilde{B})^c = \tilde{A}^c \cap \tilde{B}^c$$

$$(\tilde{A} \cap \tilde{B})^c = \tilde{A}^c \cup \tilde{B}^c$$

1.6 FUZZY SET OPERATIONS

Let \tilde{A} and \tilde{B} are two fuzzy sets with membership functions $\mu_{\tilde{A}}(x)$ and $\mu_{\tilde{B}}(x)$ respectively in the universe of discourse \cup , where $x \in \cup$. Basic fuzzy set operations are defined as follows:

1. **Union:** Union of two fuzzy sets \tilde{A} and \tilde{B} is a fuzzy set $\tilde{A} \cup \tilde{B}$ on \cup . Membership of $\tilde{A} \cup \tilde{B}$ is defined as:

$$\mu_{\tilde{A} \cup \tilde{B}} = \max[\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)]$$

2. **Intersection:** Intersection of two fuzzy sets \tilde{A} and \tilde{B} is a fuzzy set $\tilde{A} \cap \tilde{B}$ on \cup . Membership function of $\tilde{A} \cap \tilde{B}$ is defined as:

$$\mu_{\tilde{A} \cap \tilde{B}} = \min[\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)]$$

3. **Complement:** Complement of fuzzy set \tilde{A} is a fuzzy set \tilde{A}^c on \cup with membership function defined as:

$$\mu_{\tilde{A}^c}(x) = 1 - \mu_{\tilde{A}}(x) \quad \text{for all } x \in \cup$$

4. **Algebraic sum:** Algebraic sum of two fuzzy sets \tilde{A} and \tilde{B} is a new fuzzy set $\tilde{A} + \tilde{B}$ on \cup . Membership function of $\tilde{A} + \tilde{B}$ is defined as:

$$\mu_{\tilde{A} + \tilde{B}} = \mu_{\tilde{A}}(x) + \mu_{\tilde{B}}(x) - \mu_{\tilde{A}}(x) \cdot \mu_{\tilde{B}}(x)$$

5. **Difference:** Difference of two fuzzy sets \tilde{A} and \tilde{B} is defined as:

$$\tilde{A} - \tilde{B} = (\tilde{A} \cap \tilde{B}^c)$$

- 6. Product:** Product of two fuzzy sets \tilde{A} and \tilde{B} is a fuzzy set $\tilde{A}.\tilde{B}$ on \cup with membership function is defined as:

$$\mu_{\tilde{A}.\tilde{B}} = \mu_{\tilde{A}}(x) \cdot \mu_{\tilde{B}}(x)$$

- 7. Product with a crisp number:** Product of fuzzy set with any crisp number, say a , is a fuzzy set $a.\tilde{A}$. membership of $a.\tilde{A}$ is defined as:

$$\mu_{a.\tilde{A}}(x) = a \cdot \mu_{\tilde{A}}(x)$$

- 8. Power of a fuzzy set:** Membership function of α power of fuzzy set \tilde{A} is defined as:

$$\mu_{\tilde{A}^\alpha}(x) = (\mu_{\tilde{A}}(x))^\alpha$$

- 9. Equality:** Two fuzzy sets \tilde{A} and \tilde{B} are said to be equal if :

$$\mu_{\tilde{A}}(x) = \mu_{\tilde{B}}(x)$$

- 10. Not equal :** Two fuzzy sets \tilde{A} and \tilde{B} are said to be not equal if :

$$\mu_{\tilde{A}}(x) \neq \mu_{\tilde{B}}(x) \quad \text{for at least one } x \in \cup$$

- 11. Bounded sum (or bold union):** The bounded sum of two fuzzy sets \tilde{A} and \tilde{B} is a fuzzy set $\tilde{A} \oplus \tilde{B}$ with membership function as:

$$\mu_{\tilde{A} \oplus \tilde{B}}(x) = \min\{1, \mu_{\tilde{A}}(x) + \mu_{\tilde{B}}(x)\}$$

- 12. Bounded difference (or bold intersection):** The bounded difference of two fuzzy sets \tilde{A} and \tilde{B} is a fuzzy set $\tilde{A} \circ \tilde{B}$. Membership function of $\tilde{A} \circ \tilde{B}$ is defined as:

$$\mu_{\tilde{A} \circ \tilde{B}}(x) = \max\{0, \mu_{\tilde{A}}(x) - \mu_{\tilde{B}}(x)\}$$

E.g. Let \tilde{A} and \tilde{B} are two fuzzy sets, where

$$\tilde{A} = \{(1,1), (3,0.2), (5,0.5), (7,0.9), (9,0.3)\}$$

$$\tilde{B} = \{(1,0.5), (3,0.3), (5,0.7), (7,0.2), (9,1)\}$$

Find Union, Intersection, Complement, Difference, Product, Bounded sum and Bounded difference of \tilde{A} and \tilde{B} .

Solution:

(i) **Union :** $\mu_{\tilde{A} \cup \tilde{B}} = \max[\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)]$

$$\mu_{\tilde{A} \cup \tilde{B}} = \max \left[\begin{array}{l} ((1,1), (1,0.5)), ((3,0.2), (3,0.3)), ((5,0.5), (5,0.7)), \\ ((7,0.9), (7,0.2)), ((9,0.3), (9,1)) \end{array} \right]$$

$$\mu_{\tilde{A} \cup \tilde{B}} = [(1,1), (3,0.3), (5,0.7), (7,0.9), (9,1)]$$

(ii) **Intersection :**

$$\mu_{\tilde{A} \cap \tilde{B}} = \min[\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)]$$

$$\mu_{\tilde{A} \cap \tilde{B}} = \min \left[\begin{array}{l} ((1,1), (1,0.5)), ((3,0.2), (3,0.3)), ((5,0.5), (5,0.7)), \\ ((7,0.9), (7,0.2)), ((9,0.3), (9,1)) \end{array} \right]$$

$$\mu_{\tilde{A} \cap \tilde{B}} = [(1,0.5), (3,0.2), (5,0.5), (7,0.2), (9,0.3)]$$

(iii) Complement:

Complement of \tilde{A} : $\mu_{\tilde{A}^c}(x) = 1 - \mu_{\tilde{A}}(x)$

$$\mu_{\tilde{A}^c}(x) = [(1,0), (3,0.8), (5,0.5), (7,0.1), (9,0.7)]$$

Complement of \tilde{B} : $\mu_{\tilde{B}^c}(x) = 1 - \mu_{\tilde{B}}(x)$

$$\mu_{\tilde{B}^c}(x) = [(1,0.5), (3,0.7), (5,0.3), (7,0.8), (9,0)]$$

(iv) Product:

$$\mu_{\tilde{A} \cdot \tilde{B}} = \mu_{\tilde{A}}(x) \cdot \mu_{\tilde{B}}(x)$$

$$\mu_{\tilde{A} \cdot \tilde{B}} = \left\{ \begin{array}{l} ((1,1) \cdot (1,0.5)), ((3,0.2) \cdot (3,0.3)), ((5,0.5) \cdot (5,0.7)), \\ ((7,0.9) \cdot (7,0.2)), ((9,0.3) \cdot (9,1)) \end{array} \right\}$$

$$\mu_{\tilde{A} \cdot \tilde{B}} = \{(1,0.5), (3,0.06), (5,0.35), (7,0.18), (9,0.3)\}$$

(v) Algebraic sum:

$$\mu_{\tilde{A} + \tilde{B}} = \mu_{\tilde{A}}(x) + \mu_{\tilde{B}}(x) - \mu_{\tilde{A}}(x) \cdot \mu_{\tilde{B}}(x)$$

$$\mu_{\tilde{A} + \tilde{B}} = [\{ ((1,1) + (1,0.5)), ((3,0.2) + (3,0.3)), ((5,0.5) + (5,0.7)), ((7,0.9) + (7,0.2)), \\ ((9,0.3) + (9,1)) \} - \{ ((1,1) \cdot (1,0.5)), ((3,0.2) \cdot (3,0.3)), ((5,0.5) \cdot (5,0.7)), \\ ((7,0.9) \cdot (7,0.2)), ((9,0.3) \cdot (9,1)) \}]$$

$$\mu_{\tilde{A} + \tilde{B}} = [\{ (1,1.5), (3,0.5), (5,1.2), (7,1.1), (9,1.3) \} - \{ (1,0.5), (3,0.06), (5,0.35), (7,0.18), (9,0.3) \}]$$

$$\mu_{\tilde{A} + \tilde{B}} = [(1,1), (3,0.44), (5,0.85), (7,0.92), (9,1)]$$

(vi) Bounded sum:

$$\mu_{\tilde{A} \oplus \tilde{B}}(x) = \min\{1, \mu_{\tilde{A}}(x) + \mu_{\tilde{B}}(x)\}$$

$$\mu_{\tilde{A} \oplus \tilde{B}}(x) = \min\{1, ((1,1) + (1,0.5)), ((3,0.2) + (3,0.3)), ((5,0.5) + (5,0.7)), \\ ((7,0.9) + (7,0.2)), ((9,0.3) + (9,1))\}$$

$$\mu_{\tilde{A} \oplus \tilde{B}}(x) = \min\{1, (1,1.5), (3,0.5), (5,1.2), (7,1.1), (9,1.3)\}$$

$$\mu_{\tilde{A} \oplus \tilde{B}}(x) = \{(1,1), (3,0.5), (5,1), (7,1), (9,1)\}$$

1.7 TYPES OF FUZZY SETS

- **Normal fuzzy set:** If the membership function of a fuzzy set has at least one element with membership value 1, then fuzzy set is called normal fuzzy set.
- **Subnormal fuzzy set:** If the membership function of a fuzzy set has no element with membership value 1, then fuzzy set is called subnormal fuzzy set.

- **Convex fuzzy set:** If values of membership function of a fuzzy set are either increasing or decreasing, or first increasing then decreasing for increasing values of elements of the set, then set is called convex fuzzy set.
- **Non-convex fuzzy set:** If values of membership function of a fuzzy set are not increasing or decreasing, or first increasing then decreasing for elements of the set, then set is called non-convex fuzzy set.

1.8 SUMMARY

- Classical or crisp logic is two-valued logic.
- Crisp logic partitioned data in members and non-members of the set.
- Crisp logic cannot deal with uncertainty.
- Fuzzy logic is multi-valued logic.
- Fuzzy set theory enables us to deal with uncertainty by handling imprecise and vague data in an effective way.
- Fuzzy set theory reduces the restrictions of classical logic theory by including the intermediate values of a given range.
- Fuzzy memberships functions are available for discrete as well as for continuous values of a given range.
- Core, Support and Boundary are the features of fuzzy membership function.
- There are five types of fuzzy membership functions: Triangular function, Trapezoidal function, Gaussian function, Generalized bell-shaped function and Sigmoid function.

Bibliography

- S. N. Sivanandam and S. N. Deepa, “Principles of Soft Computing”, Second Edition, Wiley.
- S. Rajasekran and G. A. V. Pai, “Neural Networks, Fuzzy Logic, and Genetic Algorithms-Synthesis and Applications, PHI, 2012
- S. Kaushik and S. Tiwari, “Soft Computing-Fundamentals, Techniques and Applications”, McGraw Hill Education.
- S. Chakraverty, D. M. Sahoo and N. R. Mahato, “Concepts of Soft Computing- Fuzzy and ANN with Programming”, Springer.

Self-Evaluation

- Q1. What do you mean by the term uncertainty?
- Q2. Why fuzzy logic is called multi-valued logic?
- Q3. Define crisp set and fuzzy set. Give two examples of each.
- Q4. State Commutative, Associative and Distributive properties of fuzzy sets.
- Q5. Briefly explain various types of membership function.
- Q6. What is the difference between convex and non-convex fuzzy sets?

- Q7. State bounded sum and bounded difference operations of fuzzy sets with example.
- Q8. Represent following crisp sets using list/rule method:
- Set of all natural numbers
 - Set of prime numbers up to 50
 - Set of vowels of English language
 - Set of combinations of alphabets a, b and c
- Q9. Given that $\tilde{A} = \{(1,0.2), (2,0.5), (3,0.6), (4,0.8), (5,1)\}$. What will be the value of \tilde{A}^c ?
- Q10. If $\tilde{A} = \{(10,0), (20,0.2), (30,0.6), (40,1), (50,0.8)\}$ and $\tilde{B} = \{(10,0.2), (20,0.4), (30,0.5), (40,0.9), (50,1)\}$
- Apply following operations on \tilde{A} and \tilde{B} :
- Algebraic sum
 - Intersection
 - Difference
 - Bounded sum
 - Bounded difference
- Q11. De Morgan's law is:
- $(\tilde{A} \cup \tilde{B})^c = \tilde{A}^c \cap \tilde{B}^c$
 - $(\tilde{A} \cup \tilde{B})^c = \tilde{A} \cap \tilde{B}^c$
 - $(\tilde{A} \cup \tilde{B})^c = \tilde{A}^c \cap \tilde{B}$
 - $(\tilde{A} \cup \tilde{B})^c = \tilde{A}^c \cup \tilde{B}^c$
- Q12. Membership function of $\tilde{A} \circ \tilde{B}$ is:
- $\mu_{\tilde{A} \circ \tilde{B}}(x) = \min\{0, \mu_{\tilde{A}}(x) - \mu_{\tilde{B}}(x)\}$
 - $\mu_{\tilde{A} \circ \tilde{B}}(x) = \max\{1, \mu_{\tilde{A}}(x) - \mu_{\tilde{B}}(x)\}$
 - $\mu_{\tilde{A} \circ \tilde{B}}(x) = \min\{1, \mu_{\tilde{A}}(x) - \mu_{\tilde{B}}(x)\}$
 - $\mu_{\tilde{A} \circ \tilde{B}}(x) = \max\{0, \mu_{\tilde{A}}(x) - \mu_{\tilde{B}}(x)\}$
- Q13. The area defined by the complete membership in fuzzy set \tilde{A} is called _____.
- Boundary
 - Support
 - Core
 - Height
- Q14. Fuzzy logic allows partial set membership by defining _____ values between the membership or non-membership values of the set.
- Q15. The maximum value of the membership function is called _____ of the fuzzy set.

UNIT-II Fuzzy Logic

- 2.1 Introduction
- 2.2 Objective
- 2.3 Fuzzy relations
- 2.4 Fuzzy Cartesian product
- 2.5 Fuzzy membership function formulation and parameterization
- 2.6 Fuzzy rules and reasoning
- 2.7 Formulation of fuzzy rules
- 2.8 Extension principle and nested fuzzy relations
- 2.9 Summary
- 2.10 Terminal Questions

2.1 INTRODUCTION

This chapter introduces fuzzy logic concepts, including how to create and set up fuzzy membership functions and use if-then rules for fuzzy reasoning. It focuses on exploring fuzzy relations, the fuzzy Cartesian product, and how to design effective fuzzy rules. We also cover the theory and practical uses of nested fuzzy relations and the extension principle.

2.2 OBJECTIVES

After studying this chapter, you should be able to:

- Describe the fuzzy Cartesian product and its role in forming fuzzy relations.
- Describe the types of membership functions, methods for their formulation, and parameterization techniques.
- Describe the principles for creating fuzzy rules, including linguistic variables and rule structures.
- Describe the extension principle, its theoretical foundation, and examples of nested fuzzy relations.

2.3 Fuzzy Relations :

- Fuzzy Relation R is a mapping from the Cartesian space $A \times B$ to interval $[0,1]$ where the strength of mapping is expressed by the membership function of the relation.

$$R \subseteq A \times B \quad \text{And} \quad \mu_R = A \times B \rightarrow [0, 1]$$

- A simple example of binary fuzzy relation on $U=\{1,2,3\}$, called “approximately equal” can be defined as

$$R(1,1) = R(2,2) = R(3,3) = 1$$

$$R(1,2) = R(2,1) = R(2,3) = R(3,2) = 0.8$$

$$R(1,3) = R(3,1) = 0.3$$

In other words, the membership function of R is given by

$$R(u,v) = \begin{cases} 1 & \text{if } u=v \\ 0.8 & \text{if } |u-v|=1 \\ 0.3 & \text{if } |u-v|=2 \end{cases}$$

- In matrix notation it can be represented as

$$R = \begin{pmatrix} 1 & 0.8 & 0.3 \\ 0.8 & 1 & 0.8 \\ 0.3 & 0.8 & 1 \end{pmatrix}$$

- **Graphical representation of a fuzzy relation** : A relation R also can be graphically represented. For example, suppose, a fuzzy relation R in $V \times W$, where $V = \{1, 2, 3\}$ and $W = \{1, 2, 3, 4\}$ has the following definition.

$$R = [\{\{1,1\},1\}, \{\{1,2\},.2\}, \{\{1,3\},.7\}, \{\{1,4\},0\},$$

$$\{\{2,1\},.7\}, \{\{2,2\},1\}, \{\{2,3\},.4\}, \{\{2,4\},.8\},$$

$$\{\{3,1\},0\}, \{\{3,2\},.6\}, \{\{3,3\},.3\}, \{\{3,4\},.5\},$$

This relation can be represented in the form of a graph as shown in Fig. 2.1.

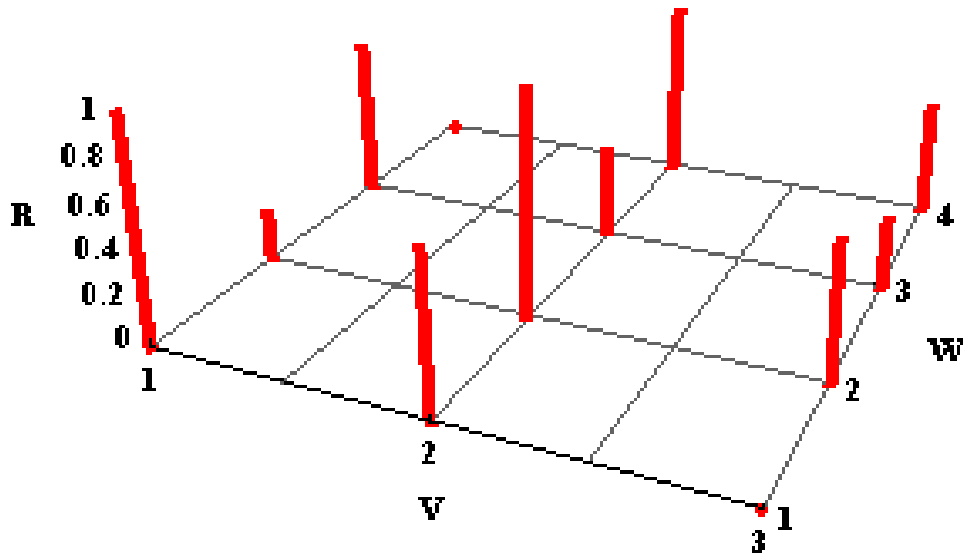


Figure 2.1: Graphical representation of relation R [4].

- **Operations on Fuzzy Relations:** Let R and S be two fuzzy relations on Cartesian space $X \times Y$. There are three major Fuzzy operations:

1. Union:

The union of R and S is defined by $(R \cup S)(u,v) = \max\{R(u,v), S(u,v)\}$

As we know, union of sets is to combine all the elements of those sets involved in that operation, in case of fuzzy relations (which are also sets), we not only combine their elements

but also calculate their membership function as:

$$\mu_{R \cup S}(x,y) = \text{Max}[\mu_R(x,y), \mu_S(y,z)]$$

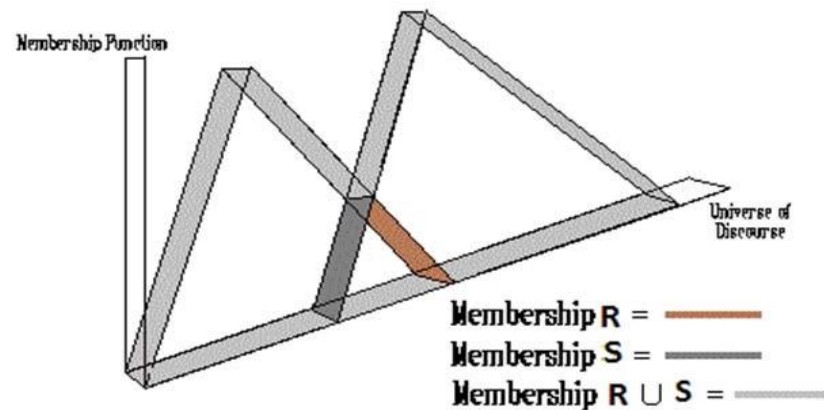


Figure 2.2: Graphical representation of union operation of two relations R and S [4].

Example:

Suppose, two relations R and S are given as follows.

R= “x is considerable larger than y”

$$\begin{matrix} & y1 & y2 & y3 & y4 \\ \begin{pmatrix} x1 & 0.8 & 0.1 & 0.1 & 0.7 \\ x2 & 0.0 & 0.8 & 0.0 & 0.0 \\ x3 & 0.9 & 1.0 & 0.7 & 0.8 \end{pmatrix} \end{matrix}$$

S= “x is very close toy”

$$\begin{matrix} & y1 & y2 & y3 & y4 \\ \begin{pmatrix} x1 & 0.4 & 0.0 & 0.9 & 0.6 \\ x2 & 0.9 & 0.4 & 0.5 & 0.7 \\ x3 & 0.3 & 0.0 & 0.8 & 0.5 \end{pmatrix} \end{matrix}$$

The union of R and S means that “x is considerable larger than y” **or** “x is very close to y”.

$$\begin{matrix} & y1 & y2 & y3 & y4 \\ (R \vee S)(x, y) = \begin{pmatrix} x1 & 0.8 & 0.1 & 0.9 & 0.7 \\ x2 & 0.9 & 0.8 & 0.5 & 0.7 \\ x3 & 0.3 & 0.1 & 0.8 & 0.8 \end{pmatrix} \end{matrix}$$

2. Intersection

The intersection of R and S is defined as

$$(R \wedge S)(u,v) = \min \{ R(u, v), S(u, v) \}.$$

Similarly in case intersection of fuzzy relations we find the membership function as:

$$\mu_{R \cap S} = \min [\mu_R(x,y), \mu_S(y,z)]$$

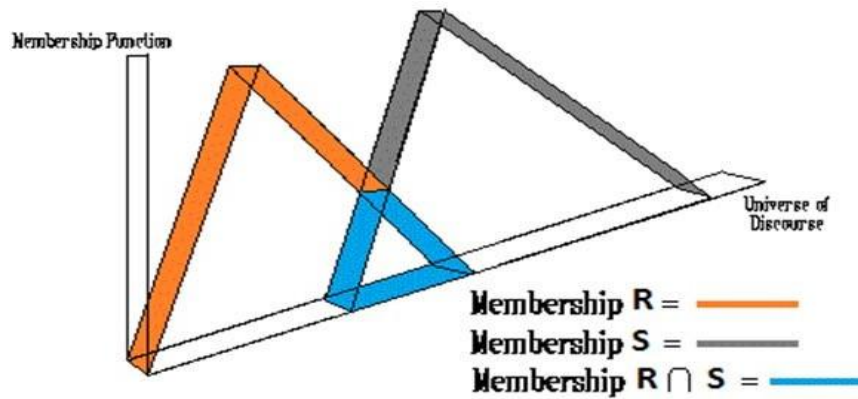


Figure 2.3: Graphical representation of intersection operation of two relations R and S [4].

With reference to the relations R and S as mentioned above, the intersection means that “x is considerable larger than y” and “x is very close toy”.

$$(R \wedge S)(x,y) = \begin{pmatrix} & y1 & y2 & y3 & y4 \\ x1 & 0.4 & 0.0 & 0.9 & 0.6 \\ x2 & 0.9 & 0.4 & 0.5 & 0.0 \\ x3 & 0.1 & 0.0 & 0.3 & 0.5 \end{pmatrix}$$

3. Complement

The Complement of R is defined R and can be stated as $R(u,v)=1-R(u,v)$.

Here to make a complement of afuzzy relation, we add all the elements of the primary relation and just make the membership function change as follows:

$$\mu_{R^c}(x,y) = 1 - \mu_R(x,y)$$

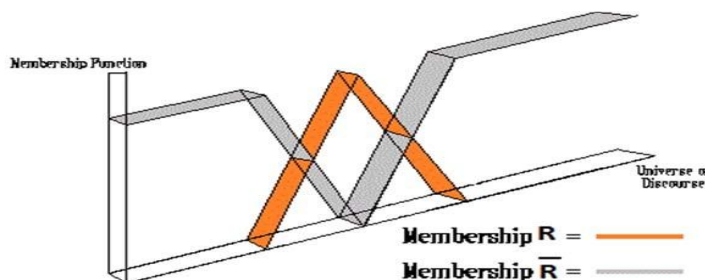


Figure 2.4: Graphical representation of intersection operation of two [4].

Relations R and S the Complement of R means that “x is consider ably smaller than y”.

$$\bar{R} = \begin{pmatrix} & y1 & y2 & y3 & y4 \\ x1 & 0.2 & 0.9 & 0.9 & 0.3 \\ x2 & 1.0 & 0.2 & 1.0 & 1.0 \end{pmatrix}$$

Crisp relation: Suppose, A and B are two (crisp) sets. Then Cartesian product denoted as $A \times B$ is a collection of order pairs, such that $A \times B = \{(a, b) | a \in A \text{ and } b \in B\}$

Note: (1) $A \times B \neq B \times A$

(2) $|A \times B| = |A| \times |B|$

(3) $A \times B$ provides a mapping from $a \in A$ to $b \in B$.

The mapping so mentioned is called a relation

Example 1: Consider the two crisp sets A and B as given below.

$A = \{1, 2, 3, 4\}$ $B = \{3, 5, 7\}$. Then, $A \times B = \{(1, 3), (1, 5), (1, 7), (2, 3), (2, 5), (2, 7), (3, 3), (3, 5), (3, 7), (4, 3), (4, 5), (4, 7)\}$.

Let us define a relation R as $R = \{(a, b) | b = a + 1, (a, b) \in A \times B\}$ Then, $R = \{(2, 3), (4, 5)\}$ in this case. We can represent the relation R in a matrix form as follows.

$$R = \begin{matrix} & \begin{matrix} 3 & 5 & 7 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \end{matrix}$$

• **Operations on crisp relations:**

Suppose, $R(x, y)$ and $S(x, y)$ are the two relations define over two crisp sets $x \in A$ and $y \in B$

Union: $R(x, y) \cup S(x, y) = \max(R(x, y), S(x, y))$;

Intersection: $R(x, y) \cap S(x, y) = \min(R(x, y), S(x, y))$;

Complement: $\overline{R(x, y)} = 1 - R(x, y)$

Self-Evaluations:

Example:

Suppose, $R(x, y)$ and $S(x, y)$ are the two relations define over two crisp sets $x \in A$ and $y \in B$

$$R = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \text{ and } S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

Find the following:

- $R \cup S$
- $R \cap S$
- Complement of R

- **Composition of two crisp relations:**

Given R is a relation on X, Y and S is another relation on Y, Z . Then $R \circ S$ is called a composition of relation on X and Z which is defined as follows.

$$R \circ S = \{(x, z) | (x, y) \in R \text{ and } (y, z) \in S \text{ and } \forall y \in Y\}$$

- **Max-Min Composition:** Given the two relation matrices R and S , the max-min composition is defined as $T = R \circ S$

$$T(x, z) = \max \{ \min \{ R(x, y), S(y, z) \} \text{ and } \forall y \in Y \}$$

For example 1:

$$X = (x_1, x_2, x_3); Y = (y_1, y_2); Z = (z_1, z_2, z_3);$$

$$R = \begin{matrix} & \begin{matrix} y_1 & y_2 \end{matrix} \\ \begin{matrix} x_1 \\ x_2 \\ x_3 \end{matrix} & \begin{bmatrix} 0.5 & 0.1 \\ 0.2 & 0.9 \\ 0.8 & 0.6 \end{bmatrix} \end{matrix}$$

$$S = \begin{matrix} & \begin{matrix} z_1 & z_2 & z_3 \end{matrix} \\ \begin{matrix} y_1 \\ y_2 \end{matrix} & \begin{bmatrix} 0.6 & 0.4 & 0.7 \\ 0.5 & 0.8 & 0.9 \end{bmatrix} \end{matrix}$$

$$R \circ S = \begin{matrix} & \begin{matrix} z_1 & z_2 & z_3 \end{matrix} \\ \begin{matrix} x_1 \\ x_2 \\ x_3 \end{matrix} & \begin{bmatrix} 0.5 & 0.4 & 0.5 \\ 0.5 & 0.8 & 0.9 \\ 0.6 & 0.6 & 0.7 \end{bmatrix} \end{matrix}$$

- $\mu_{R \circ S}(x_1, z_1) = \max \{ \min \{ (x_1, y_1), (y_1, z_1) \}, \min \{ (x_1, y_2), (y_2, z_1) \} \} = \max \{ \min(0.5, 0.6), \min(0.1, 0.5) \} = \max \{ 0.5, 0.1 \} = 0.5$.
- $\mu_{R \circ S}(x_1, z_2) = \max \{ \min \{ (x_1, y_1), (y_1, z_2) \}, \min \{ (x_1, y_2), (y_2, z_2) \} \} = \max \{ \min(0.5, 0.4), \min(0.1, 0.8) \} = \max \{ 0.4, 0.1 \} = 0.4$, similarly we compute other values.

Example 2:

Given

$$X = \{1, 3, 5\}; Y = \{1, 3, 5\}; R = \{(x, y) | y = x + 2\}; S = \{(x, y) | x < y\}$$

Here, R and S is on $X \times Y$.

Thus, we have

$$R = \{(1, 3), (3, 5)\}$$

$$S = \{(1, 3), (1, 5), (3, 5)\}$$

$$R = \begin{matrix} & \begin{matrix} 1 & 3 & 5 \end{matrix} \\ \begin{matrix} 1 \\ 3 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \end{matrix} \text{ and } S = \begin{matrix} & \begin{matrix} 1 & 3 & 5 \end{matrix} \\ \begin{matrix} 1 \\ 3 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \end{matrix}$$

$$\text{Using max-min composition } R \circ S = \begin{matrix} & \begin{matrix} 1 & 3 & 5 \end{matrix} \\ \begin{matrix} 1 \\ 3 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{matrix}$$

- **Reflexive Relation in Fuzzy Set Theory:**

A fuzzy relation R in $X \times X$ is called reflexive if:

$$\mu_R(x, x) = 1 \quad \forall x \in X$$

- **Example:**

Let $X = \{1, 2, 3, 4\}$

$$R = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 1 & 0.9 & 0.6 & 0.2 \\ 0.9 & 1 & 0.7 & 0.3 \\ 0.6 & 0.7 & 1 & 0.9 \\ 0.2 & 0.3 & 0.9 & 1 \end{bmatrix} \end{matrix}$$

is reflexive relation

- **Antireflexive relations**

A fuzzy relation $R \subseteq X \times X$ is called antireflexive if:

$$\mu_R(x, x) = 0 \quad \forall x \in X$$

- **Example:**

$$R_1 = \begin{matrix} & \begin{matrix} x_1 & x_2 & x_3 \end{matrix} \\ \begin{matrix} x_1 \\ x_2 \\ x_3 \end{matrix} & \begin{bmatrix} 0 & 0 & 0.6 \\ 0.3 & 0 & 0 \\ 0 & 0.3 & 0 \end{bmatrix} \end{matrix} \text{ is antireflexive relation}$$

- **Symmetric relations**

A fuzzy relation R is called symmetric if:

$$\mu_R(x, y) = \mu_R(y, x) \quad \forall x, y \in X$$

- **Example:**

$$R = \begin{matrix} & \begin{matrix} x_1 & x_2 & x_3 \end{matrix} \\ \begin{matrix} x_1 \\ x_2 \\ x_3 \end{matrix} & \begin{bmatrix} 0.8 & 0.1 & 0.7 \\ 0.1 & 1 & 0.6 \\ 0.7 & 0.6 & 0.5 \end{bmatrix} \end{matrix} \text{ is a symmetric relation.}$$

- **Antisymmetric Relation:**

A fuzzy relation $R \subseteq X \times X$ is antisymmetric if:

$$\text{if } \mu_R(x, y) > 0 \text{ then } \mu_R(y, x) = 0 \quad \forall x, y \in X, x \neq y$$

- **Example:**

$$R = \begin{matrix} & \begin{matrix} x_1 & x_2 & x_3 \end{matrix} \\ \begin{matrix} x_1 \\ x_2 \\ x_3 \end{matrix} & \begin{bmatrix} 0 & 0 & 0.7 \\ 0.2 & 0 & 0 \\ 0 & 0.2 & 0 \end{bmatrix} \end{matrix} \text{ is antisymmetric relation.}$$

- **Transitive Relation**

A fuzzy relation $R \subseteq X \times X$ is called transitive if:

$$\mu_R(x, z) \geq \min(\mu_R(x, y), \mu_R(y, z)) \quad \forall x, y, z \in X$$

- since $R^2 = R \circ R$

Then R is transitive if $R \circ R \subset R$

and $R^2 \subset R$ means that $\mu_{R^2}(x, y) \leq \mu_R(x, y)$.

Example:

Let $X = \{x_1, x_2\}$

$$\text{is } R = \begin{bmatrix} 0.4 & 0.2 \\ 0.7 & 0.3 \end{bmatrix} \text{ a transitive relation?}$$

Solution

$$R \circ R = \begin{bmatrix} 0.4 & 0.2 \\ 0.7 & 0.3 \end{bmatrix} \circ \begin{bmatrix} 0.4 & 0.2 \\ 0.7 & 0.3 \end{bmatrix}$$

using max-min composition

$$R^2 = \begin{bmatrix} \max(\min(0.4, 0.4), \min(0.2, 0.7)) & \max(\min(0.4, 0.2), \min(0.2, 0.3)) \\ \max(\min(0.7, 0.4), \min(0.3, 0.7)) & \max(\min(0.7, 0.2), \min(0.3, 0.3)) \end{bmatrix}$$

$$R^2 = \begin{bmatrix} \max(0.4, 0.2) & \max(0.2, 0.2) \\ \max(0.4, 0.3) & \max(0.2, 0.3) \end{bmatrix}$$

$$R^2 = \begin{bmatrix} 0.4 & 0.2 \\ 0.4 & 0.3 \end{bmatrix}$$

$\mu_{R^2}(x_i, x_j)$ is less than or equal to $\mu_R(x_i, x_j)$, so R is transitive.

Since $\mu_{R^2}(x_i, x_j)$ is not always less than or equal to $\mu_R(x_i, x_j)$, hence R is not transitive.

Self-Evaluation

Question:

Let $X = \{x_1, x_2, x_3\}$

is $R = \begin{bmatrix} 0.7 & 0.9 & 0.4 \\ 0.1 & 0.3 & 0.5 \\ 0.2 & 0.1 & 0 \end{bmatrix}$ a transitive relation?

2.4 Fuzzy Cartesian product:

Suppose

- A is a fuzzy set on the universe of discourse X with $\mu_A(x)|x \in X$
- B is a fuzzy set on the universe of discourse Y with $\mu_B(y)|y \in Y$

Then $R = A \times B \subset X \times Y$; where R has its membership function given by

$$\mu_{R(x, y)} = \mu_{A \times B}(x, y) = \min \{ \mu_A(x), \mu_B(y) \}$$

Example: $A = \{(a_1, 0.2), (a_2, 0.7), (a_3, 0.4)\}$ and $B = \{(b_1, 0.5), (b_2, 0.6)\}$

$$R = A \times B = \begin{matrix} & \begin{matrix} b_1 & b_2 \end{matrix} \\ \begin{matrix} a_1 \\ a_2 \\ a_3 \end{matrix} & \begin{bmatrix} 0.2 & 0.2 \\ 0.5 & 0.6 \\ 0.4 & 0.4 \end{bmatrix} \end{matrix}$$

Self-Evaluations

1. $A = \{(x_1, 0.5), (x_2, 1), (x_3, 0.6)\}$, $B = \{(y_1, 1), (y_2, 0.4)\}$

Find: $A \times B$

2.

Let, $R = x$ is relevant to y

and $S = y$ is relevant to z

be two fuzzy relations defined on $X \times Y$ and $Y \times Z$, respectively, where $X = \{1, 2, 3\}$, $Y = \{\alpha, \beta, \gamma, \delta\}$ and $Z = \{a, b\}$.

Assume that R and S can be expressed with the following relation matrices :

$$R = \begin{matrix} & \begin{matrix} \alpha & \beta & \gamma & \delta \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} & \begin{bmatrix} 0.1 & 0.3 & 0.5 & 0.7 \\ 0.4 & 0.2 & 0.8 & 0.9 \\ 0.6 & 0.8 & 0.3 & 0.2 \end{bmatrix} \end{matrix} \text{ and}$$

$$S = \begin{matrix} & \begin{matrix} a & b \end{matrix} \\ \begin{matrix} \alpha \\ \beta \\ \gamma \\ \delta \end{matrix} & \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.3 \\ 0.5 & 0.6 \\ 0.7 & 0.2 \end{bmatrix} \end{matrix}$$

- Find RoS .

2.5. Fuzzy membership function formulation and parameterization:

2.5.1 Fuzzy membership function:

- Fuzzy logic is used to describe fuzziness.
- Fuzziness (i.e., all the information in fuzzy set) is characterized by a membership function.
- The membership function represents the degree of truth in fuzzy logic.
- Membership functions solve practical problems through experience rather than knowledge [1, 5].
- **Formal Definition of membership function:** Consider a fuzzy set A defined as $A = \{(x, \mu_A(x)) | x \in X\}$
 - $\mu_A(x)$ is the membership function for the fuzzy set A.
 - X is the universe of discourse.
 - The membership function $\mu_A(x)$ maps each element $x \in X$ to a value in the interval [0, 1].
 - In fuzzy sets, each element is mapped to [0, 1]. by the membership function, $\mu_A : X \rightarrow [0, 1]$, where [0, 1] includes all real numbers between 0 and 1 (inclusive).
 - A fuzzy set has a "vague boundary" compared to a crisp set.
 - If X is discrete, the fuzzy set A can be alternatively denoted as $A = \sum \mu_A(x_i)/x_i$.
 - If X is continuous, the fuzzy set A can be alternatively denoted as $A = \int \mu_A(x)/x$

2.5.2 Features of membership function: Three basic feature of membership function are [1, 2]:

- **Core:** It is the region described by the complete membership in the set \tilde{A} .
 $\text{Core}(\tilde{A}) = \{x \in U | \mu_{\tilde{A}}(x) = 1\}$
- **Support:** It is the region described by a nonzero membership in the set \tilde{A} .
 $\text{Support}(\tilde{A}) = \{x \in U | \mu_{\tilde{A}}(x) > 0\}$
- **Boundary:** It is the region described by nonzero but not complete membership in the set.
 $\text{Boundary}(\tilde{A}) = \{x \in U | 0 < \mu_{\tilde{A}}(x) < 1\}$

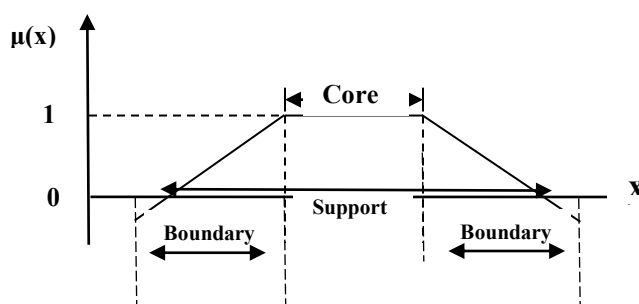


Figure 2.5: Features of membership function- core, support and boundary.

- **Other features:**

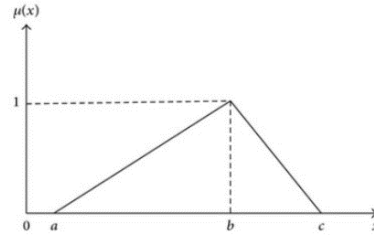
- **Crossover:** A crossover point of a fuzzy set A is a point $x \in X$ where $\mu_A(x)=0.5$.
i.e. $\text{Crossover}(A) = \{x \in X | \mu_A(x)=0.5\}$
- **Normality:** A fuzzy set A is normal if its core is non-empty. This means that $\text{core}(A) \neq \emptyset \implies A$ is a normal fuzzy set.
- **Fuzzy singleton:** A fuzzy set A is called a fuzzy singleton if its support is a single point x with $\mu_A(x)=1$.
- **α -cut :** An α -cut of a fuzzy set A is defined as:
$$A_\alpha = \{x \in X | \mu_A(x) \geq \alpha\}$$
- **Bandwidth:** For a normal and convex fuzzy set A, the bandwidth is the distance between two unique crossover points. Mathematically, the bandwidth is given by:
▪ **Bandwidth(A) = $|x_2 - x_1|$ where $\mu_A(x_1) = \mu_A(x_2) = 0.5$**

2.5.3 Membership functions: Parameterization and Formulation

Most commonly used fuzzy membership functions are:

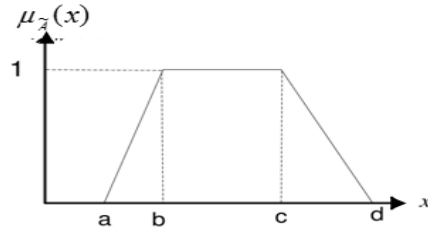
- a) **Triangular membership function:** It takes three scalar values a, b and c, where a, b, c represents the three points on x-axis of the triangular membership function such that $a < b < c$. Function is defined as follows.

$$f(x, a, b, c) = \begin{cases} 0, & \text{if } x \leq a \\ \frac{(x-a)}{(c-a)}, & \text{if } x \in [a, c] \\ \frac{(b-x)}{(b-c)}, & \text{if } x \in [c, b] \\ 0, & \text{if } x \geq b \end{cases}$$



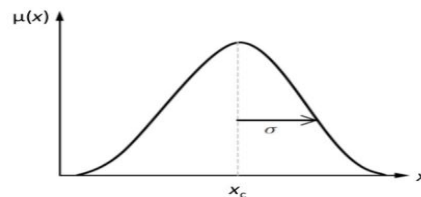
- b) **Trapezoidal membership function:** It takes four scalar values a, b, c & d and is defined by the following expression.

$$f(x, a, b, c, d) = \begin{cases} 0, & \text{if } x \leq a \\ \frac{(x-a)}{(b-a)}, & \text{if } a \leq x \leq b \\ 1, & \text{if } b \leq x \leq c \\ \frac{(d-x)}{(d-c)}, & \text{if } c \leq x \leq d \\ 0, & \text{if } d \leq x \end{cases}$$



- c) **Gaussian membership function:** It takes two values σ and c, where σ represents width and c represents center of the membership function.

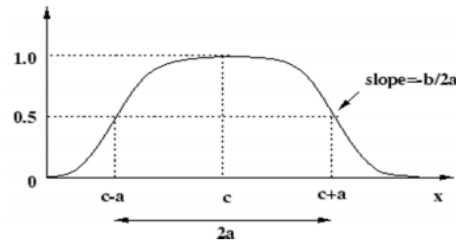
$$f(x, \sigma, c) = e^{\frac{-(x-c)^2}{2\sigma^2}}$$



- d) **Generalized bell shaped membership function:** It takes three values a, b and c, where c is

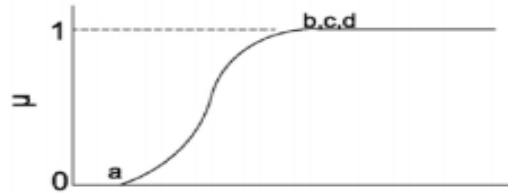
center of the curve and b is slope at the cusp or crossover points as shown in figure.

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



- e) **Sigmoid membership function:** It takes two parameters a and c, where a represents slope at cusp point c.

$$f(x, a, c) = \frac{1}{1 + e^{-a(x-c)}}$$



2.6 Fuzzy rules and reasoning :

2.6.1 Fuzzy rules

- A Fuzzy if then rule also known as fuzzy rule, fuzzy implication, or fuzzy conditional statement:

IF X is A THEN Y is B

- Fuzzy if-then rules: If X is A then Y is B, where A and B are linguistic values defined by fuzzy sets on universes of discourse X and Y.
- X is A is called the antecedent or premise.
- Y is B is called the consequence or conclusion.
- **Examples:**
 - If a tomato is red, then it is ripe
 - If the road is slippery, then driving is dangerous.
 - If pressure is high, then volume is small
 - If the speed is high, then apply the brake a little
- Without fuzzy if then rules there is no fuzzy system
- fuzzy if-then-rules “if X is A THEN Y is B” is also abbreviated by $A \rightarrow B$
- There are two ways to interpret $A \rightarrow B$:
 - A coupled with B
 - A entails B

if A is coupled with B then:

$$\mathbf{R} = \mathbf{A} \Rightarrow \mathbf{B} = \mathbf{A} * \mathbf{B} = \int_{\mathbf{x} * \mathbf{y}} \mu_{\mathbf{A}}(\mathbf{x}) \tilde{*} \mu_{\mathbf{B}}(\mathbf{y}) / (\mathbf{x}, \mathbf{y})$$

▪ where $\tilde{*}$ is a T - normoperator.

▪ Note:

T-norm operator

The most frequently used T-norm operators are:

Minimum : $T_{min}(a, b) = \min(a, b) = a \wedge b$

Algebraic product : $T_{ap}(a, b) = ab$

Bounded product : $T_{bp}(a, b) = 0 \vee (a + b - 1)$

Drastic product : $T_{dp} = \begin{cases} a & \text{if } b = 1 \\ b & \text{if } a = 1 \\ 0 & \text{if } a, b < 1 \end{cases}$

Here, $a = \mu_A(x)$ and $b = \mu_B(y)$. T_* is called the function of T-norm operator.

▪

If A entails B then:

$\mathbf{R} = \mathbf{A} \Rightarrow \mathbf{B} = \neg \mathbf{A} \cup \mathbf{B}$ (material implication)

$\mathbf{R} = \mathbf{A} \Rightarrow \mathbf{B} = \neg \mathbf{A} \cup (\mathbf{A} \cap \mathbf{B})$ (propositional calculus)

$\mathbf{R} = \mathbf{A} \Rightarrow \mathbf{B} = (\neg \mathbf{A} \cap \neg \mathbf{B}) \cup \mathbf{B}$ (extended propositional calculus)

$$\mu_{\mathbf{R}}(\mathbf{x}, \mathbf{y}) = \sup \left\{ \mathbf{c}; \mu_{\mathbf{A}}(\mathbf{x}) \tilde{*} \mathbf{c} \leq \mu_{\mathbf{B}}(\mathbf{y}), 0 \leq \mathbf{c} \leq 1 \right\}$$

- If then rules are of different types:
 - Single rule with single antecedent
 - Single rule with multiple antecedent
 - Multiple with multiple antecedent

2.6.2 Fuzzy Reasoning

- It is also Known as approximate reasoning
- It is an inference procedure derives conclusions from a set of fuzzy if-then rules and known facts [1, 2].
- **Inferring procedures in Fuzzy logic**
 - By propositional logic, we are aware of the following.
 - Modus Ponens : $P, P \Rightarrow Q, \Leftrightarrow Q$
 - Modus Tollens : $P \Rightarrow Q, \neg Q \Leftrightarrow, \neg P$

- Chain rule : $P \Rightarrow Q, Q \Rightarrow R \Leftrightarrow P \Rightarrow R$
- Example: Given
 - $C \vee D$
 - $\sim H \Rightarrow (A \wedge \sim B)$
 - $C \vee D \Rightarrow \sim H$
 - $(A \wedge \sim B) \Rightarrow (R \vee S)$
 - From the above can we infer $R \vee S$?
- **A similar idea is also used in fuzzy logic, also known as fuzzy rule base, to infer a fuzzy rule from a set of given fuzzy rules.**
- Fuzzy systems make use of two crucial inferring techniques:

Generalized Modus Ponens (GMP)

If x is A Then y is B

x is A'

y is B'

Generalized Modus Tollens (GMT)

If x is A Then y is B

y is B'

x is A'

- $A, B, A',$ and B' are fuzzy sets in this case. The max-min composition of the fuzzy sets B' and A' , respectively, with $R(x, y)$ (the known implication relation), is to be used to compute the membership functions A' and B' .

Thus,

$$B' = A' \circ R(x, y) \quad \mu_{B'}(y) = \max[\min(\mu_{A'}(x), \mu_R(x, y))]$$

$$A' = B' \circ R(x, y) \quad \mu_{A'}(x) = \max[\min(\mu_{B'}(y), \mu_R(x, y))]$$

- **Example: P : If x is A then y is B**

Let us consider two sets of variables x and y be

$X = \{x_1, x_2, x_3\}$ and $Y = \{y_1, y_2\}$, respectively. Also, let us consider the following. $A = \{(x_1, 0.5), (x_2, 1), (x_3, 0.6)\}$, $B = \{(y_1, 1), (y_2, 0.4)\}$. Then, given a fact expressed by the proposition x is A' , where $A' = \{(x_1, 0.6), (x_2, 0.9), (x_3, 0.7)\}$ derive a conclusion in the form y is B' (using generalized modus ponens (GMP)).

- **Solutions:**

If x is A Then y is B

x is A'

y is B'

We are to find $B' = A' \circ R(x, y)$ where $R(x, y) = \max\{A \times B, \bar{A} \times Y\}$

$$A \times B = \begin{matrix} & \begin{matrix} y_1 & y_2 \end{matrix} \\ \begin{matrix} x_1 \\ x_2 \\ x_3 \end{matrix} & \begin{bmatrix} 0.5 & 0.4 \\ 1 & 0.4 \\ 0.6 & 0.4 \end{bmatrix} \end{matrix} \text{ and } \bar{A} \times Y = \begin{matrix} & \begin{matrix} y_1 & y_2 \end{matrix} \\ \begin{matrix} x_1 \\ x_2 \\ x_3 \end{matrix} & \begin{bmatrix} 0.5 & 0.5 \\ 0 & 0 \\ 0.4 & 0.4 \end{bmatrix} \end{matrix}$$

Note: For $A \times B$, $\mu_{A \times B}(x, y) = \min(\mu_A(x), \mu_B(y))$

$$R(x, y) = (A \times B) \cup (\bar{A} \times Y) = \begin{matrix} & \begin{matrix} y_1 & y_2 \end{matrix} \\ \begin{matrix} x_1 \\ x_2 \\ x_3 \end{matrix} & \begin{bmatrix} 0.5 & 0.5 \\ 1 & 0.4 \\ 0.6 & 0.4 \end{bmatrix} \end{matrix}$$

Now, $A' = \{(x_1, 0.6), (x_2, 0.9), (x_3, 0.7)\}$

Therefore, $B' = A' \circ R(x, y) =$

$$[0.6 \quad 0.9 \quad 0.7] \circ \begin{bmatrix} 0.5 & 0.5 \\ 1 & 0.4 \\ 0.6 & 0.4 \end{bmatrix} = [0.9 \quad 0.5]$$

Thus we derive that y is B' where $B' = \{(y_1, 0.9), (y_2, 0.5)\}$

2.7 Formulation of Fuzzy Rules

- Creating a fuzzy rule in AI means making statements that can handle unclear or uncertain data. These rules are useful in decision-making, control systems, and pattern recognition.
- To create a good fuzzy rule, follow these steps:
 - **Identify Variables:** Determine relevant input and output variables for the problem.
 - **Define Fuzzy Sets:** Establish fuzzy sets for each variable to represent different linguistic values (e.g., "low," "medium," "high").
 - **Construct the Rule:** Use an "if-then" structure to formulate the rule. Example: "If temperature is high, then fan speed is maximum."
 - **Simplify the Rule:** Keep the rule straightforward to ensure clarity and ease of implementation. Avoid excessive conditions or exceptions.
 - **Test and Refine:** Apply the rule in real-world scenarios and refine it based on performance to enhance accuracy and reliability.
 - **Iterate:** Continuously update the rule with new data and feedback for ongoing improvement.
- By following these steps, you can create fuzzy rules that effectively manage imprecision and ambiguity in various applications.

2.8 Extension principle and nested fuzzy relations:

2.8.1 The extension principle

- To use fuzzy numbers and relations in any intelligent system, we must perform arithmetic operations with these fuzzy quantities.
- We use the extension principle to enable arithmetic operations on fuzzy numbers.
- Generally, the extension principle plays a fundamental role in extending point operations to operations involving fuzzy sets.
- The extension principle developed by Zadeh (1975) and later by Yager (1986) establishes how to extend the domain of a function on a fuzzy sets [6]
- **Extension Principle for Fuzzy Sets:**

\tilde{A} : fuzzy set defined on X

$y = f(x)$: functional transform or mapping

\tilde{B} : image of \tilde{A} on X under f .

\tilde{B} is a fuzzy set having universe of discourse Y .

$$\tilde{B} = f(\tilde{A})$$

$$\mu_{\tilde{B}}(y) = \bigvee_{f(x)=y} \mu_{\tilde{A}}(x)$$

Zadeh's Extension principle: Suppose f is a mapping from an n dimensional

Cartesian product space $X_1 \times X_2 \times \dots \times X_n$ to a one dimensional universe Y such that $y = f(x_1, x_2, \dots, x_n)$ and suppose $\tilde{A}_1, \tilde{A}_2, \dots, \tilde{A}_n$ are n fuzzy sets in x_1, x_2, \dots, x_n respectively. Then, the image of $\tilde{A}_1, \tilde{A}_2, \dots, \tilde{A}_n$ under f is given as:

$$\mu_{\tilde{B}}(y) = \max_{f(x_1, x_2, \dots, x_n) = y} \left\{ \min(\mu_{\tilde{A}_1}(x_1), \mu_{\tilde{A}_2}(x_2), \dots, \mu_{\tilde{A}_n}(x_n)) \right\}$$

Let

$$A = 0.1/-2 + 0.4/-1 + 0.8/0 + 0.9/1 + 0.3/2$$

and

$$f(x) = x^2 - 3.$$

Upon applying the extension principle, we have

$$\begin{aligned} B &= 0.1/1 + 0.4/-2 + 0.8/-3 + 0.9/-2 + 0.3/1 \\ &= 0.8/-3 + (0.4 \vee 0.9)/-2 + (0.1 \vee 0.3)/1 \\ &= 0.8/-3 + 0.9/-2 + 0.3/1, \end{aligned}$$

2.8.2 Nested fuzzy relations:

- Relations where fuzzy sets are contained within other fuzzy sets, creating a hierarchy or nested structure

- In order to model complex systems with hierarchical or multi-layered dependencies, nested fuzzy relations are used.
- With this method, complex relationships between variables can be captured at various abstraction levels.

2.9 SUMMARY

- Discussed the principles and applications of fuzzy relations and fuzzy Cartesian products.
- Examined various types of membership functions, including triangular, trapezoidal, and Gaussian, and their formulation techniques.
- Investigated methods of fuzzy reasoning and inference, focusing on Mamdani and Sugeno approaches.
- Studied the concept and hierarchical structure of nested fuzzy relations, including their applications in complex system modelling.

2.10 TERMINAL QUESTIONS

1. What is membership function? Explain various types of membership functions.
2. Define fuzzy relations and fuzzy Cartesian products with suitable examples.
3. Discuss in detail fuzzy rules.
4. Explain Crossover, support, bandwidth, alpha-cut, and core.
5. Explain fuzzy reasoning and fuzzy nested relations.

2.11 BIBLIOGRAPHY

1. J.S.R.Jang, C.T.Sun and E.Mizutani, “Neuro-Fuzzy and Soft Computing”, PHI, 2004, Pearson Education 2004.
2. Timothy J.Ross, “Fuzzy Logic with Engineering Applications”, McGraw-Hill, 1997
3. Fuzzy relations. <https://sunilwanjarisvpceet.wordpress.com/wp-content/uploads/2021/11/fl-relations.pdf>, accessed on 24-07-24.
4. Fuzzy relations <https://cse.iitkgp.ac.in/~dsamanta/courses/archive/sca/Archives/Chapter%20%20Fuzzy%20Relation.pdf>, accessed on 24-07-24.
5. Fuzzy Membership Function Formulation and Parameterization <https://cse.iitkgp.ac.in/~dsamanta/courses/archive/sca/Archives/Chapter%20%20Fuzzy%20Membership%20Functions.pdf>, Accessed on 26-07-24.
6. Fuzzy logic. https://www.tutorialspoint.com/fuzzy_logic/fuzzy_logic_membership_function.htm, Accessed on 27-07-24

UNIT-III Fuzzy rule base system

- 3.1 Introduction
- 3.2 Objective
- 3.3 Fuzzy rule base system
- 3.4 Fuzzy if-then rules
- 3.5 Fuzzy inference
- 3.6 Fuzzy inference system
- 3.7 Defuzzification methods
- 3.8 Fuzzy control systems
- 3.9 Applications of Fuzzy control systems.
- 3.10 Summary
- 3.11 Terminal Questions

3.1 INTRODUCTION

In this unit, we describe how fuzzy logic effectively handles uncertainty and imprecision in complex systems through the following topics: fuzzy rule base systems, which use fuzzy if-then rules and fuzzy inference in fuzzy inference systems. We will also cover defuzzification methods that convert fuzzy results into crisp values, making these systems practical. Additionally, we will explore fuzzy control systems and their applications in fields like industrial automation and consumer electronics, showcasing the versatility and effectiveness of fuzzy logic in real-world applications.

3.2 OBJECTIVES

After studying this chapter, you should be able to:

- Understand the structure and components of fuzzy rule base systems.
- Analyze the role of fuzzy if-then rules in decision-making processes.
- Understand the architecture and functioning of fuzzy inference systems.
- Understand how to convert fuzzy results into crisp, actionable values.
- Learn how fuzzy control systems are used to manage and regulate processes.
- Analyze real-world applications of fuzzy control systems in various fields.

3.3 FUZZY RULE BASE SYSTEM :

Fuzzy systems include [1, 2]:

- Fuzzy Logic:
- Fuzzy Set Theory
- Knowledge exists in two distinct forms:
- Objective knowledge:
 - Exists in mathematical form

- Used in engineering problems
- Subjective knowledge:
 - Exists in linguistic form
 - Usually impossible to quantify
- Fuzzy Logic: Coordinates objective and subjective knowledge logically
- Fuzzy Systems:
 - Any system that uses Fuzzy mathematics may be viewed as a Fuzzy system.
 - Handle numerical data and linguistic knowledge simultaneously
 - Provide opportunities for modelling conditions that are inherently imprecisely defined
 - Have been used to model, simulate, and replicate many real-world problems
 - Applications include:
 - Information retrieval systems
 - Navigation systems
 - Robot vision

Elements of Fuzzy system: the elements of fuzzy system are shown in the Figure 3.1

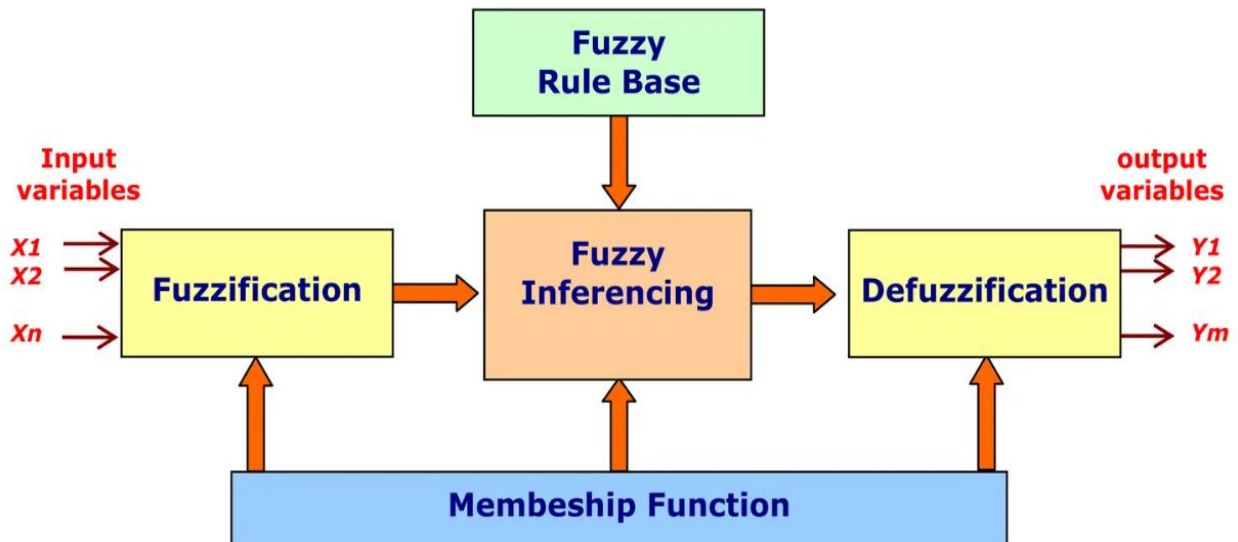


Figure 3.1 Elements of fuzzy system.

- **Input Vector (X):** $X = [x_1, x_2, \dots, x_n]^T$ are precise values turned into fuzzy sets in the fuzzification step.
- **Output Vector (Y):** $Y = [y_1, y_2, \dots, y_m]^T$ comes from the defuzzification step, changing a fuzzy set back into a precise value.
- **Fuzzification:** Converts precise values into fuzzy categories like "far", "near", and "small".

- **Fuzzy Rule Base:**
 - A set of rules using words instead of numbers.
 - Example: If (x is far) AND (y is near) THEN (z is small).
- **Membership Function:**
 - Measures how well something fits into a fuzzy category.
- **Fuzzy Inferencing:**
 - Uses the fuzzy data and rules to make decisions.
- **Defuzzification:**
 - Converts fuzzy results back into precise values.

3.4 FUZZY IF-THEN RULES :

A Fuzzy if then rule also known as fuzzy rule, fuzzy implication, or fuzzy conditional statement:

IF X is A THEN Y is B

- Fuzzy if-then rules: If X is A then Y is B, where A and B are linguistic values defined by fuzzy sets on universes of discourse X and Y.
- X is A is called the antecedent or premise.
- Y is B is called the consequence or conclusion.
- **Examples:**
 - If a tomato is red, then it is ripe
 - If the road is slippery, then driving is dangerous.
 - If pressure is high, then volume is small
 - If the speed is high, then apply the brake a little
- Without fuzzy if then rules there is no fuzzy system
- fuzzy if-then-rules “if X is A THEN Y is B” is also abbreviated by $A \rightarrow B$
- There are two ways to interpret $A \rightarrow B$:
 - A coupled with B
 - A entails B

if A is coupled with B then:

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

- where $*$ is a T - normoperator.

▪ **Note:**

T-norm operator

The most frequently used T-norm operators are:

Minimum : $T_{min}(a, b) = \min(a, b) = a \wedge b$

Algebraic product : $T_{ap}(a, b) = ab$

Bounded product : $T_{bp}(a, b) = 0 \vee (a + b - 1)$

Drastic product : $T_{dp} = \begin{cases} a & \text{if } b = 1 \\ b & \text{if } a = 1 \\ 0 & \text{if } a, b < 1 \end{cases}$

Here, $a = \mu_A(x)$ and $b = \mu_B(y)$. T_* is called the function of T-norm operator.

▪

If A entails B then:

$R = A \Rightarrow B = \neg A \cup B$ (material implication)

$R = A \Rightarrow B = \neg A \cup (A \cap B)$ (propositional calculus)

$R = A \Rightarrow B = (\neg A \cap \neg B) \cup B$ (extended propositional calculus)

$$\mu_R(x, y) = \sup \left\{ c; \mu_A(x) * c \leq \mu_B(y), 0 \leq c \leq 1 \right\}$$

- If then rules are of different types:
 - Single rule with single antecedent
 - Single rule with multiple antecedent
 - Multiple with multiple antecedent

Self-Evaluations

- What is Fuzzy if-then rules? Explain with suitable examples
- Explain membership functions and defuzzification.

3.5 FUZZY INFERENCE AND FUZZY INFERENCE INFERENCE SYSTEM :

A Fuzzy Inferencing System uses fuzzy reasoning to process inputs into outputs through three steps [1]:

1. **Fuzzification:**
 - a. It is the first step in the fuzzy inferencing process.
 - b. Converts exact sensor inputs (like temperature or pressure) into fuzzy values.
 - c. Each input has its own membership functions, which define how it is fuzzified.
2. **Rule Evaluation:** Applies fuzzy rules to the fuzzy inputs to generate fuzzy outputs.
3. **Defuzzification:** Converts the fuzzy outputs back into exact values.

- **Fuzzy inference system (FIS)** is illustrated in Figure 3.2.

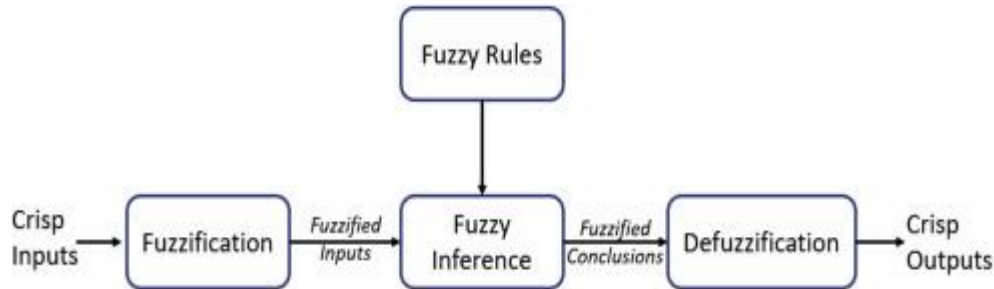


Figure 3.2 Fuzzy inference system

- **Methods of FIS**
 - Mamdani Fuzzy Inference System
 - Takagi-Sugeno Fuzzy Model (TS Method)
- **Mamdani Fuzzy Inference System:** The system, proposed by Ebhasim Mamdani in 1975, was designed to control a steam engine and boiler combination. It synthesized a set of fuzzy rules, which were obtained from individuals working on the system. To compute the output from this FIS, follow these steps:
 - **Determine the Set of Fuzzy Rules:** Identify the set of fuzzy rules to be used.
 - **Fuzzify the Inputs:** Convert the input values into fuzzy values using the input membership function.
 - **Establish Rule Strength:** Combine the fuzzified inputs according to the fuzzy rules to establish the rule strength.
 - **Determine the Consequent of the Rule:** Combine the rule strength with the output membership function to determine the rule's consequent.
 - **Combine the Consequents:** Merge all the consequents to obtain the output distribution.
 - **Defuzzify the Output Distribution:** Finally, convert the fuzzy output distribution into a crisp, defuzzified output.
- The Mamdani Fuzzy Interface System block diagram is shown below:

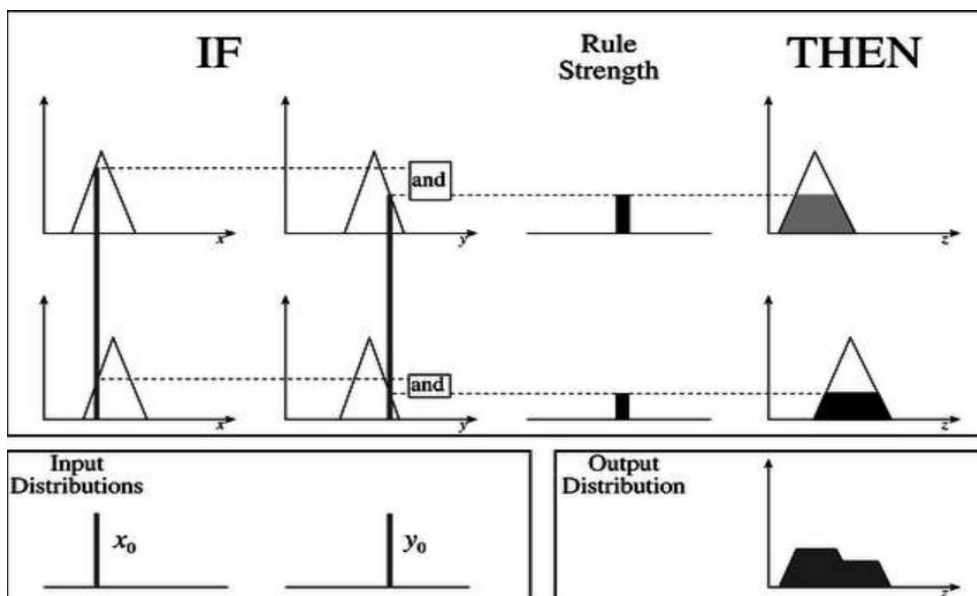


Figure 3.3 Mamdani Fuzzy Interface System [8]

- **Takagi-Sugeno Fuzzy Model (TS Method):** This method is proposed by: Takagi, Sugeno, and Kang in 1985.
 - **Rule Format:**
 - IF x is A and y is B THEN $Z = f(x,y)$
 - **Components:**
 - Antecedents: A and B are fuzzy sets.
 - Consequent: $z = f(x,y)$ is a crisp function.
 - The Takagi-Sugeno Fuzzy Model (TS Method) fuzzy inference procedure functions as follows:
 - **Fuzzifying the Inputs:** Provide fuzzy values for the system's inputs.
 - **Applying the Fuzzy Operator:** To get the output, apply fuzzy operators to the inputs that have been fuzzified.
 - Difference Between Mamdani and Sugeno Fuzzy Inference System:

Points	Mamdani FIS	Sugeno FIS
Output membership function	Present	Not Present
Output of surface	Discontinuous	Continuous
Distribution of output	Yes	No
Defuzzification	A clear outcome is achieved by defuzzifying the rules.	Here, there is no defuzzification. One can obtain a crisp result by utilising a weighted average of the subsequent rules.
Flexibility	less flexibility in the system design	More flexibility in the system design
In security evaluation block cipher algorithm	It has more accuracy	It has less accuracy
MISO (Multiple Input and Single Output) and MIMO (Multiple Input and Multiple Output) systems	Used	Using only MISO
Application:	Medical Diagnosis System	To monitor how an aircraft's performance varies with altitude

3.7 DEFUZZIFICATION METHODS :

- Fuzzy rule-based systems evaluate linguistic if-then rules using fuzzification, inference, and composition procedures.
- They produce fuzzy results that usually need to be converted into crisp output.
- Defuzzification transforms fuzzy results into a single crisp value.
- Defuzzification is the reverse process of fuzzification.
- The following are the known methods of defuzzification [3, 4]:
 - Center of Sums Method (COS)
 - Center of Gravity (COG) / Centroid of Area (COA) Method
 - Center of Area / Bisector of Area Method (BOA)
 - Weighted Average Method
 - Maxima Methods
 - First of Maxima Method (FOM)
 - Last of Maxima Method (LOM)
 - Mean of Maxima Method (MOM)

3.7.1 Center of Sums Method (COS)

- This is the most commonly used defuzzification technique, where the overlapping area is counted twice [5].

The defuzzified value x^* is defined as :

$$x^* = \frac{\sum_{i=1}^N x_i \cdot \sum_{k=1}^n \mu_{A_k}(x_i)}{\sum_{i=1}^N \sum_{k=1}^n \mu_{A_k}(x_i)}$$

Here, n represents the number of fuzzy sets, N denotes the number of fuzzy variables, and $\mu_k(x)$ is the membership function for the k-th fuzzy set.

Example:

The defuzzified value x^* is defined as :

$$x^* = \frac{\sum_{i=1}^k A_i \times \bar{x}_i}{\sum_{i=1}^k A_i} ,$$

Here, A_i represents the firing area of i^{th} rules and k is the total number of rules fired and \bar{x}_i represents the center of area.

The aggregated fuzzy set of two fuzzy sets C_1 and C_2 is shown in Figure 1. Let the area of this two fuzzy sets are A_1 and A_2 .

$$A_1 = \frac{1}{2} * [(8-1) + (7-3)] * 0.5 = \frac{1}{2} * 11 * 0.5 = 55/20 = 2.75$$

$$A_2 = \frac{1}{2} * [(9-3) + (8-4)] * 0.3 = \frac{1}{2} * 10 * 0.3 = 3/2 = 1.5$$

Now the center of area of the fuzzy set C_1 is let say $\bar{x}_1 = (7+3)/2 = 5$ and

the center of area of the fuzzy set C_2 is $\bar{x}_2 = (8+4)/2 = 6$.

$$\text{Now the defuzzified value } x^* = \frac{(A_1 \cdot \bar{x}_1 + A_2 \cdot \bar{x}_2)}{A_1 + A_2} = \frac{(2.75 * 5 + 1.5 * 6)}{(2.75 + 1.5)} = 22.75/4.25 = 5.35$$

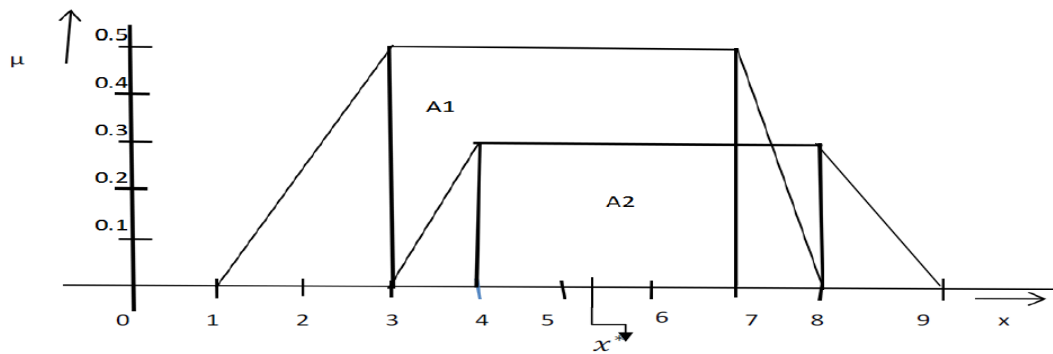


Figure 3.4 Fuzzy sets C1 and C2

3.7.2 Center of gravity (COG) / Centroid of Area (COA) Method:

- This method provides a precise value based on the center of gravity (centroid) of the fuzzy set.
- The total area of the membership function distribution, representing the combined control action, is divided into multiple sub-areas
- Calculation of Area and Centroid:
 - Calculate the area of each sub-area.
 - Determine the centroid of each sub-area.
- Summation for Defuzzified Value:
 - Sum the areas and centroids of all sub-areas.
 - Use this summation to find the defuzzified value for the discrete fuzzy set

For discrete membership function, the defuzzified value denoted as x^* using COG is defined as:

$$x^* = \frac{\sum_{i=1}^n x_i \cdot \mu(x_i)}{\sum_{i=1}^n \mu(x_i)}, \text{ Here } x_i \text{ indicates the sample element, } \mu(x_i) \text{ is}$$

the membership function, and n represents the number of elements in the sample.

For continuous membership function, x^* is defined as :

$$x^* = \frac{\int x \mu_A(x) dx}{\int \mu_A(x) dx}$$

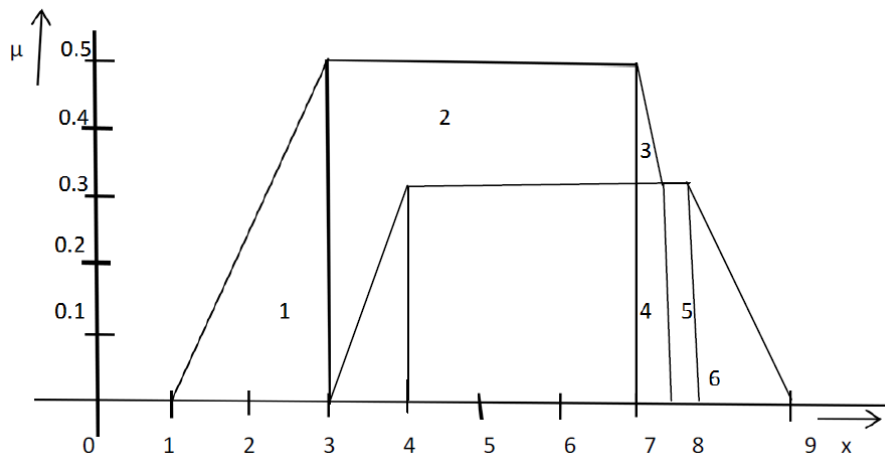


Figure 3.5: Fuzzy sets C1 and C2

Example:

The defuzzified value x^* using COG is defined as:

$$x^* = \frac{\sum_{i=1}^N A_i \times \bar{x}_i}{\sum_{i=1}^N A_i}, \text{ Here } N \text{ indicates the number of sub-areas, } A_i \text{ and}$$

\bar{x}_i represents the area and centroid of area, respectively, of i^{th} sub-area.

In the aggregated fuzzy set as shown in figure 2. , the total area is divided into six sub-areas. For COG method, we have to calculate the area and centroid of area of each sub-area.

These can be calculated as below.

The total area of the sub-area 1 is $\frac{1}{2} * 2 * 0.5 = 0.5$

The total area of the sub-area 2 is $(7-3) * 0.5 = 4 * 0.5 = 2$

The total area of the sub-area 3 is $\frac{1}{2} * (7.5-7) * 0.2 = 0.5 * 0.5 * 0.2 = .05$

The total area of the sub-area 4 is $0.5 * 0.3 = .15$

The total area of the sub-area 5 is $0.5 * 0.3 = .15$

The total area of the sub-area 6 is $\frac{1}{2} * 1 * 0.3 = .15$

Now the centroid or center of gravity of these sub-areas can be calculated as

3.7.3 Center of Area / Bisector of Area Method (BOA):

This method calculates the position under the curve where the areas on both sides are equal.

The BOA generates the action that partitions the area into two regions with the same area.

$$\int_{\alpha}^{x^*} \mu_A(x) dx = \int_{x^*}^{\beta} \mu_A(x) dx, \text{ where } \alpha = \min \{x | x \in X\} \text{ and } \beta = \max \{x | x \in X\}$$

3.7.4 Weighted Average Method:

- This method is valid for fuzzy sets with symmetrical output membership functions.
- It produces results very close to the Center of Area (COA) method.
- This method is less computationally intensive compared to other methods.
- Each membership function is weighted by its maximum membership value.
- The defuzzified value is defined as:

$$x^* = \frac{\sum \mu(x) \cdot x}{\sum \mu(x)}$$

Here \sum denotes the algebraic summation and x is the element with maximum membership function.

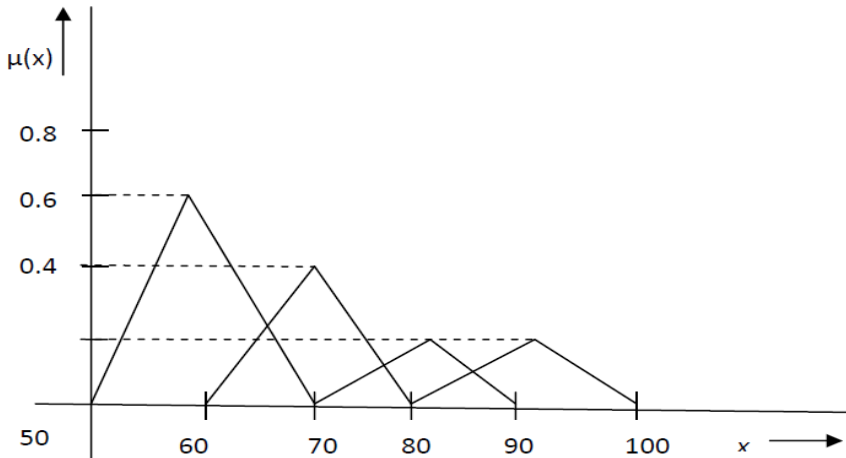


Figure 3.6: Fuzzy set A

Example:

- Let A be a fuzzy set representing a student's performance, as shown in the figure 3.5. The elements with their corresponding maximum membership values are given:

$$A = \{(P, 0.6), (F, 0.4), (G, 0.2), (VG, 0.2), (E, 0)\}$$

- Here, P: Pass student, F: Fair student, G: Good student, VG: Very Good student, EE: Excellent student,
- Now the defuzzified value x^* for set A will be

$$x^* = \frac{(60 \cdot 0.6 + 70 \cdot 0.4 + 80 \cdot 0.2 + 90 \cdot 0.2 + 100 \cdot 0)}{0.6 + 0.4 + 0.2 + 0.2 + 0}$$

$$= 98/1.4 = 70$$

- Using the weighted average method, the defuzzified value for the fuzzy set A represents a Fair student.

3.7.5 Maxima Methods:

- This approach takes values with maximum membership into account. Different maxima methods exist, each with a different approach to resolving conflicts when there are multiple maxima:
 - First of Maxima Method (FOM)
 - Last of Maxima Method (LOM)
 - Mean of Maxima Method (MOM)

First of Maxima Method (FOM)

- The smallest value of the domain with the highest membership value is found using this method.
 - Example:
- The defuzzified value x^* of the given fuzzy set will be $x^*=4$.

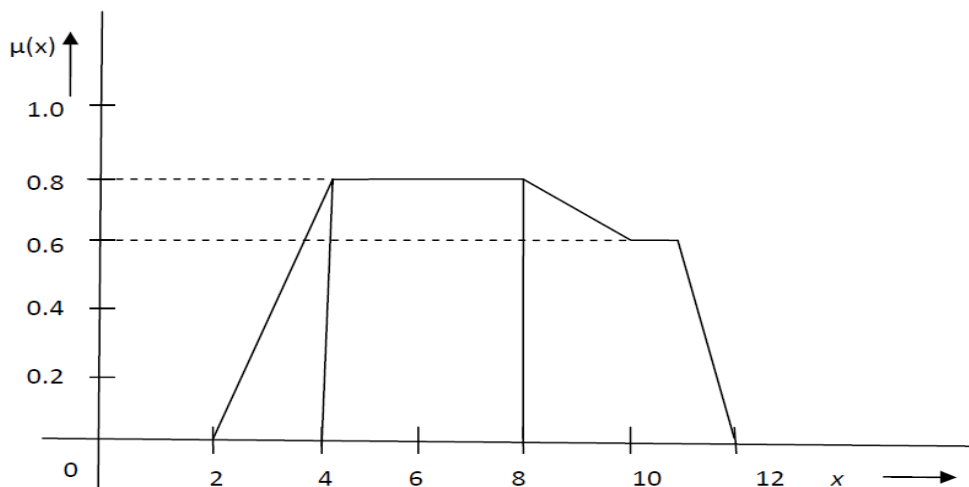


Figure 3.7: Example Fuzzy set

Last of Maxima Method (LOM)

- This method find the domain's largest value that has the highest membership value. For the FOM example, $x^* = 8$ will be the defuzzified value for the LOM method.

Mean of Maxima Method (MOM)

- The element with the highest membership value in this method is called the defuzzified value.
- In cases where multiple elements possess the highest membership value, the average of these maxima is determined.
- Let A be a fuzzy set with a membership function defined over X, where X is the universe of discourse
- The defuzzified value, denoted as x^* , of the fuzzy set A is defined as:

$$x^* = \frac{\sum_{x_i \in M} x_i}{|M|},$$

Here, $M = \{x_i \mid \mu_A(x_i) \text{ is equal to the height of the fuzzy set A}\}$ and $|M|$ is the cardinality of the set M.

Example

In the example as shown in Fig. , $x = 4, 6, 8$ have maximum membership values and hence $|M| = 3$

According to MOM method, $x^* = \frac{\sum_{x_i \in M} x_i}{|M|}$

Now the defuzzified value x^* will be $x^* = \frac{4+6+8}{3} = \frac{18}{3} = 6$.

Self-Evaluations

- Explain Fuzzy inference system with suitable diagram.
- What are defuzzyfication methods? Explain any one of them.

3.8 Fuzzy control systems, and Applications of Fuzzy control systems

3.8.1 Fuzzy control systems:

- A fuzzy control system is an AI technique that uses fuzzy logic for making decisions and controlling systems.
- Unlike traditional systems, which rely on precise models and algorithms, fuzzy control systems use approximate rules to handle imprecise or incomplete information.
- It is effective in scenarios where exact rules are hard to define, such as in complex or changing environments.
- It is Used in various fields, including robotics, automotive engineering, and medical diagnosis [6].

Working of fuzzy control system:

- A fuzzy control system uses fuzzy logic to analyze analog input values with continuous variables between 0 and 1, unlike classical logic which uses discrete values (0 or 1).
- **Principles:** Fuzzy control systems use imprecise reasoning and adaptive decision-making. Designed to handle uncertainty and imprecision in real-world scenarios.
- **Components:** The fuzzy control system consists of three main components:
 - **Fuzzification:** Converts crisp input values into fuzzy values using membership functions.
 - **Fuzzy Rule Base and Inference Engine:** Stores membership functions and fuzzy rules (IF-THEN rules), evaluated through approximate or interpolative reasoning.
 - **Defuzzification:** Converts fuzzy outputs back into crisp values for control actions.
- **Process:**
 - Inputs (e.g., sensors, switches) are mapped to membership functions in the input stage.
 - Fuzzy logic rules are applied to these inputs to derive a fuzzy output.
 - The output stage converts the fuzzy result into a crisp value for system control.

3.8.2 Applications of Fuzzy Control System:

- Fuzzy control systems have diverse applications across various domains [6, 7].
 - **In automotive systems**, they enhance performance by controlling automatic transmissions and anti-lock braking systems (ABS), adjusting gear shifts and braking force based on real-time conditions.
 - **In domestic appliances**, fuzzy control optimizes the operation of washing machines and air conditioners, improving wash cycles and maintaining comfort by adjusting temperature and fan speed.
 - **In industrial process control**, these systems regulate critical variables such as temperature, pressure, and flow rates, and manage robotic systems for efficient manufacturing processes.
 - **Medical diagnosis and treatment** benefit from fuzzy control by aiding in the diagnosis of medical conditions and customizing treatment plans based on patient data.
 - **In consumer electronics** like cameras and volume, controls use fuzzy logic to adjust focus and sound levels according to environmental factors.
 - **Environmental control applications** include managing climate systems in buildings and regulating emissions to maintain air quality.

3.9 SUMMARY

- Discussed fuzzy rule base systems, focusing on fuzzy if-then rules.
- Described the functioning of fuzzy inference systems.
- Explained defuzzification methods that convert fuzzy results into precise values.
- Explored fuzzy control systems, highlighting applications in industrial automation and consumer electronics.
- Demonstrated the practicality and versatility of fuzzy logic in real-world scenarios.

3.10 TERMINAL QUESTIONS

1. What is fuzzy rule based system? Explain the element of fuzzy system in details.
2. Write short notes on the fuzzy inference and fuzzy inference system.
3. What is Defuzzification? Explain various defuzzification methods.
4. What is fuzzy control system? Explain its working along with its application in real word.
5. Explain fuzzy inference system (FIS) with suitable diagram. Also discussed FIS methods.
6. Differentiate Mamdani and Sugeno Fuzzy Inference System.

3.11 BIBLIOGRAPHY

1. Soft Computing by D.K. Pratihari, Narosa Publication.
2. Belyadi, H., & Haghighat, A. (2021). Machine learning guide for oil and gas using Python: A step-by-step breakdown with data, algorithms, codes, and applications. Gulf Professional Publishing.
3. N. Mogharreban and L. F. DiLalla “Comparison of Defuzzification Techniques for Analysis of Noninterval Data”, IEEE, 06.
4. Jean J. Saade and Hassan B. Diab. “Defuzzification Methods and New Techniques for Fuzzy Controllers”, Iranian Journal of Electrical and Computer Engineering, 2004.
5. Aarthi Chandramohan, M. V. C. Rao and M. Senthil Arumugam: “Two new and useful defuzzification methods based on root mean square value”, Soft Computing, 2006.
6. Fuzzy control system. <https://klu.ai/glossary/fuzzy-control-system>, Accessed on 29-07-2024.
7. Applications of Fuzzy Control System. https://www.larksuite.com/en_us/topics/ai-glossary/fuzzy-control-system, Accessed on 30-07-2024.
8. Fuzzy Interface System. https://www.tutorialspoint.com/fuzzy_logic/fuzzy_logic_inference_system.htm, Accessed on 1-08-2024.



Uttar Pradesh Rajarshi Tandon
Open University

Master of Computer Science

MCS-117(N) Soft Computing

Block-3	NEURAL NETWORK	95-137
UNIT-1	Introduction of neural networks	95-104
UNIT-2	ANN and Perceptron Model	105-125
UNIT-3	Feedforward neural networks	126-137

Course Design Committee

Prof. Ashutosh Gupta Director, School of Science, UPRTOU Allahabad	Chairman
Prof. Suneeta Agarwal Dept. of Computer Science & Engineering Motilal Nehru National Institute of Technology Allahabad	Member
Dr. Upendra Nath Tripathi Associate Professor DeenDayalUpadhyay Gorakhpur University, Gorakhpur	Member
Dr. Ashish Khare Associate Professor Dept. of Computer Science, University of Allahabad, Prayagraj	Member
Ms. Marisha Assistant Professor (Computer Science) School of Science, UPRTOU Allahabad	Member
Mr. Manoj Kumar Balwant Assistant Professor (Computer Science) School of Science, UPRTOU Allahabad	Member

Course Preparation Committee

Mr. Shivendu Mishra Assistant Professor Dept. of Information Technology Rajkiya Engineering College Ambedkar Nagar U.P. India-224122	Author (Block 1, Block 2: Unit 2, 3, Block 3, & 4)
Dr. Gunjan Singh Assistant Professor (Block 2: Unit 1) Faculty of management and computer application, RBS College, Khandari, Agra-282002.	Author
Prof. Manu Pratap Singh Professor, Department of Computer Science Engineering Dr. Bhimrao Ambedkar University, Agra	Editor
Mr. Manoj Kumar Balwant Assistant Professor (computer science), School of Sciences, UPRTOU Allahabad	Coordinator

© UPRTOU, Prayagraj. 2023

First Edition: July 2023

ISBN: 978-93-48270-40-5

All rights are reserved. No part of this work may be reproduced in any form, by mimeograph or any other means, without permission in writing from the Uttar Pradesh Rajarshi Tandon Open University.



<http://creativecommons.org/licenses/by-sa/4.0/>

Creative Commons Attribution-Share Alike 4.0 International License

Printed and Published by Col. Vinay Kumar, Registrar, Uttar Pradesh Rajarshi Tandon Open University, 2024.

Printed By: Cygnus Information Solution Pvt. Ltd, Lodha Supremus Saki Vihar Road Andheri East, Mumbai.

BLOCK-3 INTRODUCTION

This block on neural networks begins with an exploration of their fundamental concepts, characteristics, and differences from traditional rule-based systems. Unit-1 of block introduces the structure of biological and artificial neurons, key neural network terminology, and the topology of artificial neural networks (ANNs), highlighting their diverse applications. Unit-2 delves into neural network dynamics, focusing on activation functions, learning laws, and various ANN architectures. Key topics include the perceptron model, ADALINE architecture, and multilayer perceptrons, addressing the strengths and limitations of these models. Finally, Unit-3 of blocks covers advanced neural network architectures and learning algorithms, including pattern mapping networks, multilayer feedforward networks, and the backpropagation learning algorithm. We also discuss enhancements to backpropagation, such as momentum and conjugate descent, along with radial basis networks, offering a comprehensive view of neural network advancements.

UNIT-I INTRODUCTION OF NEURAL NETWORKS

- 1.1 Introduction
- 1.2 Objectives
- 1.3 Structure of biological and artificial neural network
 - 1.3.1 Biological neural network
 - 1.3.2 Artificial neural network
- 1.4 Artificial neural network vs biological neural network
- 1.5 Characteristics of ANN and its applications
 - 1.5.1 Characteristics of ANN
 - 1.5.2 Applications of ANN
- 1.6 Artificial neural networks terminology
- 1.7 Topology of ANN
- 1.8 Rule-based systems
- 1.9 Summary
- 1.10 Terminal Questions

1.1 INTRODUCTION

A neural network, also known as an artificial neural network (ANN) or deep neural network, is a carefully designed machine learning (ML) model that attempts to replicate the intricate structure and functioning of the human brain. These networks comprise a sophisticated web of interconnected nodes or neurons that work together to solve complex problems. The primary goal of these networks is to create a system capable of performing a wide range of computational tasks at significantly faster speeds than conventional systems. The formation of a neural network is dependent on connecting different layers. Among the most basic is the single-layer neural network, also known as a perceptron, which is trained using machine-learning principles. These neural networks are classified into distinct types tailored to specific purposes. We begin this unit's study by carefully examining a neural network's fundamental structure, followed by a careful examination of its distinguishing features. We also define terms closely related to neural networks and their real-world uses.

1.2 OBJECTIVES

After studying this unit, you should be able to:

- Explain what neural networks is and what it does.
- Able to differentiate between artificial neural networks (ANN) and biological neural networks.
- Summarize the characteristics of ANN, ANN terminology, and applications of ANN.

1.3 STRUCTURE OF BIOLOGICAL AND ARTIFICIAL NEURAL NETWORK

1.3.1 BIOLOGICAL NEURAL NETWORK

Neural network is an artificial emulation of the human brain that attempts to replicate its learning mechanism. This artificial aspect entails the incorporation of neural networks into computer programmes, allowing them to perform the extensive calculations required throughout the learning process. The origin of neural networks can be traced back to biological neural networks, where a neuron is the basic functional unit of the nervous system, responsible for transmitting information to other cells. Comprising a cell body, an axon, and an abundance of dendrites, a neuron's primary function is to collect signals from its peers through an extensive array of fine structures known as dendrites, as illustrated in Figure 1.1 [1].

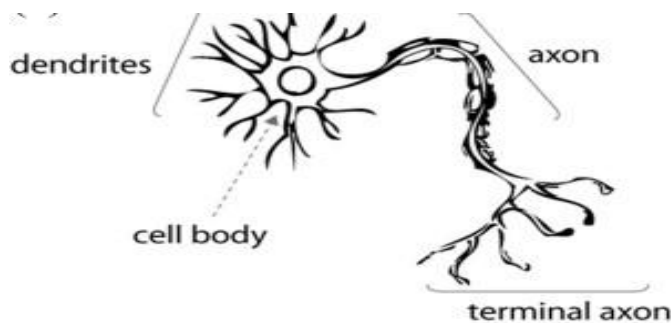


Fig. 1.1 – Biological neural network [1].

To convey electrical signals, neurons utilize their axons, which branch into an intricate network. It is important to highlight that neurons do not establish direct connections; rather, their communication relies on intricate junctions called synapses, as depicted in Figure 1.2.

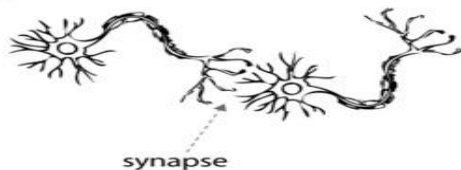


Fig. 1.2 Biological synapse [1].

These synapses serve as the connection points between neurons, where the signal must traverse a tiny space. As the signal traverses the synapse, it triggers the release of neurotransmitters, specialized chemical messengers responsible for conveying vital information from one neuron to another.

1.3.2 ARTIFICIAL NEURAL NETWORK

The inception of Artificial Neural Networks (ANNs) can be traced back to the pioneering work of Warren McCulloch and Walter Pitts in 1943. Artificial Neural Networks (ANNs) consist of numerous nodes, designed to mimic the behaviour of biological neurons in the human brain. These neurons are linked together, allowing them to interact with one another. Nodes have the ability to receive input data and execute basic operations on this data. The outcomes of these operations are then transmitted to other neurons, and the resulting value at each node is termed its activation or node value. Each link in the network is accompanied by a weight. ANNs possess the capacity for learning, a process achieved by modifying these weight values. The diagram below offers a simplified depiction of an ANN in action.

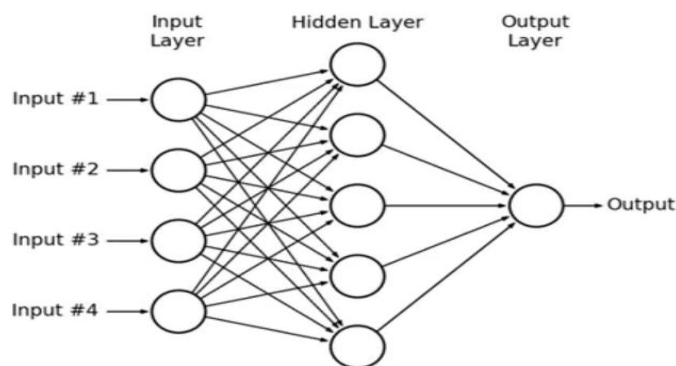


Fig. 1.3 Structure of ANN [2].

As illustrated in Figure 1.3, an ANN's architecture comprises three tiers: the input layer, the hidden layer, and the output layer. The input layer receives the initial data, and this data is subsequently processed with the aid of an intermediate layer known as the hidden layer. Weights serve as the crucial parameters facilitating the transformation of input data into the hidden layers. The hidden layer, in turn, connects to an output layer, which is responsible for predicting the ultimate output.

In summary, the connection between biological neural networks and artificial neural networks is illustrated in the table below:

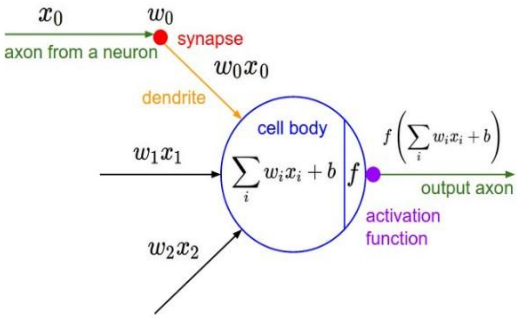
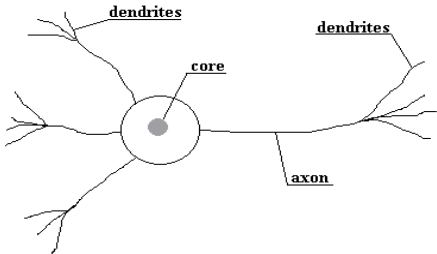
Table 1.1 Biological neural networks and artificial neural networks.

BNN	ANN
Cell nucleus	Nodes
Synapse	Weights
Dendrites	Inputs
Axon	Output

1.4 ARTIFICIAL NEURAL NETWORK VS BIOLOGICAL NEURAL NETWORK

There are several significant differences between Artificial Neural Networks (ANNs) and Biological Neural Networks (BNNs). Here are some of the major differences between the two:

S.No	Artificial Neural Networks (ANNs)	Biological Neural Networks (BNNs)
1.	Artificial Neural Networks (ANNs) serve as mathematical models predominantly influenced by the complex neuron system in the human brain.	A biological neural network comprises multiple processing units referred to as neurons, interlinked through synaptic connections.
2.	In ANNs, neurons are typically simplified, often featuring only a single output.	Neurons in Biological Neural Networks (BNNs) exhibit a greater degree of complexity and diversity.

3.	In ANNs processing occurred in a sequential and centralized manner.	BNNs processes information concurrently and in a distributed fashion.
4.	The operating environments of ANNs are typically well-defined and well-constrained.	The operating environments of BNNs are often poorly defined and less constrained compared to ANNs.
5.	ANNs is compact in size.	BNNs is substantial in size.
6.	ANNs Processes information with greater speed.	BNNs process information at a slower pace
7.	In ANNs, neural connections are typically static, with connection strength defined by a set of fixed weights	In BNNs, neural connections exhibit greater flexibility, and the strength of these connections can be adjusted by various factors, including learning processes and accumulated experiences.
8.	Within Artificial Neural Networks (ANNs), every neuron conducts a dot product operation with its inputs and corresponding weights, then adds the bias term, and subsequently applies a non-linear activation function, as illustrated in Figure 1.4.	Within Binary Neural Networks (BNNs), each neuron receives input signals through its dendrites. These dendrites transport the signals to the cell body, where they are aggregated. If the cumulative sum surpasses a specific threshold, the neuron initiates firing, generating output signals along its singular axon. Subsequently, the axon extends, forming connections through synapses with the dendrites of other neurons, as visually depicted in Figure 1.5.
8	 <p>Fig. 1.4 Structure of Neuron in ANNs [4].</p>	 <p>Fig. 1.5 Structure of Neuron in BNNs [3].</p>

Check your progress

1. What is the difference between Artificial Neural Networks (ANNs) and Biological Neural Networks (BNNs)?
2. What are the primary goals of Neural Networks?
3. What is the hidden layers in ANN?

1.5 Characteristics of ANN and its applications

1.5.1 Characteristics of ANN: Artificial Neural Networks (ANNs) exhibit several fundamental characteristics, which are outlined below.

- An Artificial Neural Network is a mathematical model that emulates the functioning of neurons in a biological neural network.
- An Artificial Neural Network comprises a vast number of interconnected processing elements known as neurons, which perform various operations.
- All of these processing elements are linked together by an enormous number of weighted connections.
- The connections among these elements facilitate the creation of a distributed representation of data.
- An Artificial Neural Network possesses the capability to learn, recall, and generalize from provided data by appropriately assigning and adjusting the weights of its connections.
- A learning process is employed to acquire knowledge from an Artificial Neural Network (ANN).

1.5.2 ANN Application:

ANNs are constantly discovering novel applications across diverse domains, as the field of artificial intelligence and machine learning evolves. Some notable examples include:

- ANNs are used for image classification, object detection, and the creation of facial recognition systems.
- Artificial Neural Networks (ANNs) play a critical role in autonomous vehicles, enabling them to sense their surroundings, make informed decisions, and navigate safely through their environments.
- ANNs are utilized for forecasting stock market trends, detecting fraudulent activities, and assessing risks in various financial contexts.
- Artificial Neural Networks (ANNs) are deployed in a variety of tasks, including sentiment analysis, machine translation, and the field of speech recognition.
- Artificial Neural Networks (ANNs) are instrumental in the realm of healthcare, where they are applied for tasks such as medical image analysis, disease prediction, drug discovery, and the advancement of personalized medicine.
- ANNs have made a name for themselves in various fields, including robotics, gaming, marketing and sales, speech and voice recognition, and many others, demonstrating their adaptability and flexibility.

1.6 ARTIFICIAL NEURAL NETWORKS TERMINOLOGY

The basic terminology of Artificial Neural Network is discussed as follows:

- a. **Artificial Neurons (Processing Elements):** Artificial Neural Networks (ANNs) serve as simplified computational models inspired by biological neural networks. ANNs are composed of basic processing units or elements, similar to the neurons found in the human brain. A processing unit consists of two essential components: a summing unit followed by an output unit (Activation or Transfer function). The role of the summing unit is to take input values from other neurons, assign weights to each input, and compute their weighted

sum, referred to as the Activation Value. The output unit (say sigmoid function) then generates its output based on the signal from this activation value.

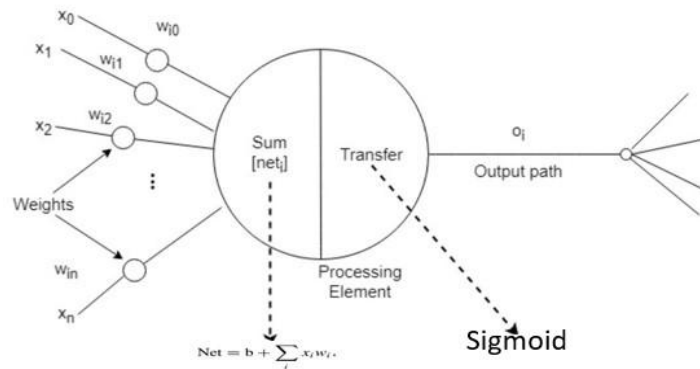


Fig. 1.6 ANN processing unit or Neurons [6].

- b. Weights:** A neural network comprises many processing elements known as neurons linked together by directed communication links known as weights. Weights act as information pipelines, transferring data from one neuron to another within the network.
- c. Bias:** The bias is a constant value that plays a significant role in the network's calculations, particularly in determining the net input. When bias is introduced, the net input is computed as follows:

$$\text{Net} = b + \sum_i x_i w_i.$$

The bias is a pivotal factor in shaping the network's output. It can take on positive or negative values, each with distinct effects. A positive bias contributes to elevating the net input of the network, while a negative bias contributes to reducing the net input of the network.

- d. Threshold (Θ):** The threshold is a predetermined value utilized within the activation function of an Artificial Neural Network (ANN). It plays a crucial role in defining the behaviour of the activation function and ultimately influences the network's output calculation.
- e. Learning Rate (α):** The learning rate serves as a crucial parameter for controlling the magnitude of weight adjustments during the training process. It falls within the range of 0 to 1 and directly governs the pace at which the network learns and adapts at each time step.
- f. Target value:** Target values, often simply referred to as "targets," represent the accurate or desired values for the output variable. These targets serve as the ground truth against which a model's predictions are compared, providing a measure of the model's performance and guiding the learning process.
- g. Error:** Error refers to the discrepancy between predicted output values and their corresponding target values, highlighting the inaccuracies in a model's predictions.

1.7 TOPOLOGY OF ANN

A topology encompasses the arrangement of a network, including its nodes and connecting lines. ANNs can be categorized into different types based on their topology, which determines their

structure and organization.

- **Feedforward Network:** This refers to a non-recurrent network characterized by layered processing units or nodes, where each node in a layer connects with nodes in the preceding layer. These connections bear varying weights. Importantly, there are no feedback loops, allowing signals to flow exclusively in a unidirectional path from input to output. This network can be further categorized into two distinct types.
- **Single layer feedforward network:** This type artificial neural network (ANN) is characterized by a single weighted layer i.e., here the input layer is fully connected to the output layer as depicted in Figure 1.7.

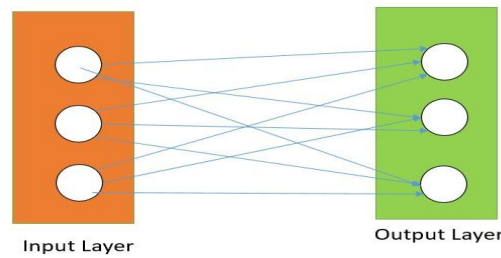


Fig. 1.7 Single layer feedforward network.

- **Multilayer feedforward network :** This type of feedforward ANN comprising multiple weighted layers. These intermediary layers positioned between the input and output layers are referred to as "hidden layers. In Figure 1.8, we provide a visual representation of multilayer feedforward networks. The left portion of the figure show cases a 2-layer Neural Network characterized by a single hidden layer comprising 4 neurons and an output layer consisting of 2 neurons, all of which are connected to three input nodes. On the right side, we present a 3-layer neural network that encompasses three input nodes, two hidden layers, each containing 4 neurons, and one output layer.

It's important to note the connectivity pattern in both cases. Neurons in these networks are interconnected across layers, forming synapses that facilitate information flow from inputs to outputs. However, there are no connections established between neurons within the same layer, emphasizing the layered architecture of these feedforward networks.

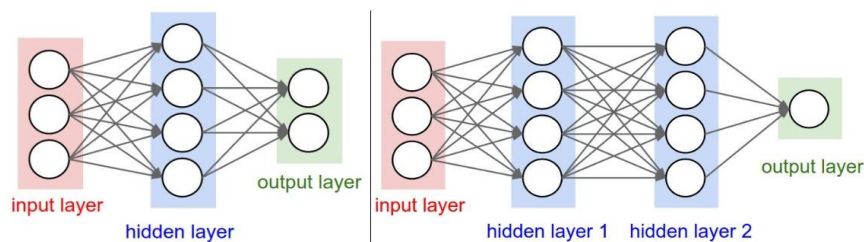


Fig. 1.8 Multi-layer feedforward network [5].

- **Feedback Network:** In contrast to traditional feedforward neural networks, this network architecture allows signals and information to travel forward and backward. They become significantly more powerful and complex as a result of this characteristic. Feedback neural networks behave dynamically as the network state changes over time until it reaches equilibrium. These networks stay in this equilibrium state as long as the input is constant. However, the dynamic process resumes when the input is altered until a new equilibrium is reached. The figure below illustrates a standard feedback neural network.

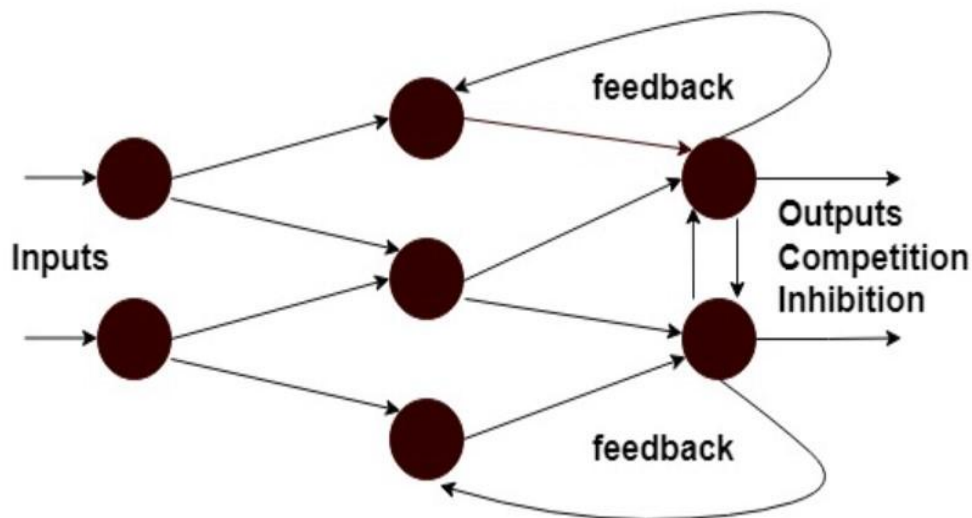


Fig. 1.9 Feedback network [6].

1.8 RULE-BASED SYSTEMS

Rule-based systems are computational constructs that leverage if-then conditional statements to guide decision-making and task execution. They exhibit notable advantages such as speed and ease of development; however, their reliance on predetermined rules and logical inference renders them less accurate beyond their specialized domains. Despite the continuous evolution of machine learning technologies, it's remarkable that rule-based systems still hold a significant role in decision-making processes. Rule-based systems' efficacy depends on the calibre and applicability of their predefined rules, which may not always be exact or precise. Contrarily, their enduring popularity is due to their exceptional efficiency and effectiveness, particularly when addressing specific, well-defined problems with a constrained set of potential solutions.

- **Limitations of rule-based systems:** Rule-based systems are renowned for their ease of use, interpretability, rapid deployment, flexibility, resilience, and seamless integration with machine learning and artificial intelligence. However, they do possess certain limitations, which include:
 - a. Rule-based systems struggle to handle a large number of variables, whereas machine learning easily incorporates them without adding complexity.
 - b. Rule-based systems encounter difficulties when handling multiple constraints within problem-solving, particularly when exceptions and special cases are involved. In contrast, machine learning excels in managing such complexities with a reduced error rate.
 - c. In rule-based systems, decision-making capacity is inherently constrained by what has been explicitly encoded.
 - d. Rule-based systems depend on human intervention for necessary adaptations, while machine learning systems autonomously derive insights from historical data and seamlessly adapt to new scenarios without external intervention.

- e. Resisting change and remaining static in response to alterations in data and the environment.
- f. Rule-based expert systems typically lack the capacity to learn from experience.
- g. Handling simpler problems.

1.9 SUMMARY

In summary,

- We have explored the concept of a neural network. Which is a carefully developed machine learning (ML) model intended to mimic the intricate structure and operation of the human brain. These networks comprise a sophisticated mesh of directly working nodes or neurons that work together to solve complex problems.
- We have also acquired a comprehensive understanding of both the intricate structure of biological neurons and the corresponding artificial neural networks. Additionally, we have grasped the essential characteristics and wide-ranging applications of artificial neural networks (ANNs). You observed that terminology and topology of ANN.
- In conclusion, we have also explored the world of rule-based systems, learning how they work and becoming familiar with their inherent drawbacks.

1.10 TERMINAL QUESTIONS

1. Discuss the structure of ANN and their characteristics.
2. What are the various components of a biological neural networks? Discuss how this is different with ANN.
3. Describes various terminology used in ANN.
4. Explains the ANN topology using a neat diagram.
5. What is feedforward and feedback neural network? Explain it.
6. What exactly is a rule-based system? Explain how it differs from a machine learning-based system.

BIBLIOGRAPHY

1. Suzuki, K. Artificial Neural Networks-Architectures and Applications; IntechOpen Limited: London, UK, 2013.
2. Artificial neural network, https://medium.com/@sakshisingh_43965/biological-artificial-neural-network-471722148217, accessed on 05-09-2023
3. Biological neural network, <https://www.nnwj.de/biological-model-human-brain.html>, accessed on 05-09-2023
4. Neural Network terminology, <https://www.cs.toronto.edu/~lc Zhang/360/lec/w02/terms.html>, accessed on 06-09-23.
5. Feed forward Neural Network, <https://cs231n.github.io/neural-networks-1/#nn>, accessed on 06-09-23.
6. ANN processing unit, <https://www.baeldung.com/cs/neural-networks-neurons>, Accessed on 7-09-23.

UNIT-II ANN AND PERCEPTRON MODEL

- 2.1 Introduction
- 2.2 Objectives
- 2.3 Artificial neural network
 - 2.3.1 Key components
 - 2.3.2 Multi-layer neural network
 - 2.3.3 ANN architecture
 - 2.3.4 Learning rules in ANN
- 2.4 Activation functions in neural networks
 - 2.4.1 What is activation function?
 - 2.4.2 Classification of an Activation function
 - 2.4.3 Most Common Non-linear Activation Functions
- 2.5 Perceptron
 - 2.5.1 Basic Components of Perceptron
 - 2.5.2 How Does Perceptron Work?
 - 2.5.3 Perceptron Learning Algorithm
 - 2.5.4 Perceptron Function
 - 2.5.5 Types of Perceptron models
 - 2.5.6 Characteristics of the Perceptron Model
 - 2.5.7 Limitations of the Perceptron model
- 2.6 Adaline (Adaptive Linear Neural)
- 2.7 LMS learning
- 2.8 Summary
- 2.9 Terminal Questions
- Bibliography

2.1 INTRODUCTION

A neural network, drawing inspiration from the human brain, comprises interconnected processing units organized into input, hidden, and output layers with weighted connections. The learning process entails weight adjustments based on known data exposure, enhancing accuracy, and enabling predictions for unfamiliar cases post-training. In this unit, we covered fundamental neural network concepts, learning rules, the versatile perceptron architecture, and the dynamics of activation functions.

2.2 OBJECTIVES

After studying this unit, you should be able to:

- Define artificial neural networks and understand how they operate.

- Explain the process of learning in artificial neural networks.
- Distinguish between the perceptron architecture and the ADALINE architecture.
- Provide a concise summary of the key characteristics of the Perceptron learning rule.

2.3 ARTIFICIAL NEURAL NETWORK

2.3.1 Key Components

- The Neural Network architecture is made up of individual units known as neurons that mimic the complex biological processes observed in the human brain. The following Figure 2.1 shown the various components that make up a neuron [1, 2].

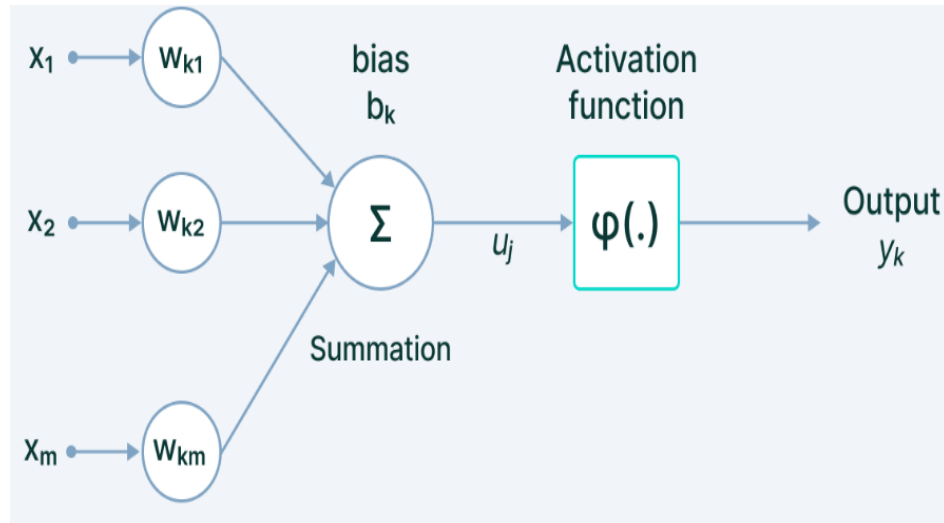


Fig. 2.1 Neuron architecture in Artificial Neural Network [2].

- **Input** - These are the features provided to the model to facilitate the learning process. In the context of object detection, for instance, the input could consist of an array of pixel values corresponding to an image.
- **Weight** - Its primary role is to assign significance to features that have a greater impact on the learning process. This is achieved through scalar multiplication between the input values and the weight matrix.
- **Transfer function** - The transfer function's main function is to combine different input signals into a single output value to make the activation function's use more convenient. This consolidation is accomplished by simply adding up all the input values inside the transfer function.
- **Activation Function**— It introduces essential non-linearity into the operation of perceptrons, accommodating the diversity in linearity that can arise from different input patterns. Without this non-linear transformation, the output would remain a mere linear combination of input values, incapable of introducing the crucial non-linearity required for the network's functionality.
- **Bias** - The bias plays a pivotal role in adjusting the output generated by the activation function. Its function is analogous to that of a constant in a linear function, serving to shift and fine-tune the overall output of the neural network.

2.3.2 Multi-layer Neural network

- When neurons are arranged sequentially in a connected fashion, they form a layer, and when

several such layers are stacked adjacent to one another, we create a multi-layer neural network. The key components of this intricate structure is described below [2]:

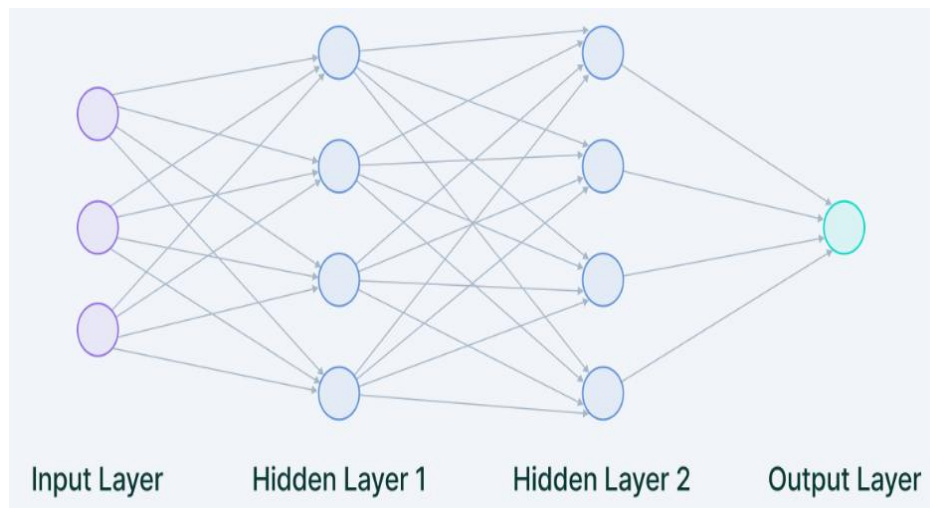


Fig. 2.2 Multi-layer Neural network [2].

- **Input Layer:** The information we provide to the model is introduced into the input layer through external sources such as a CSV file or a web service. This layer is the sole visible component in the entire Neural Network framework, serving as a conduit for transferring unprocessed data from the external environment without undergoing any computational transformations.
- **Hidden Layers:** Hidden layers are the driving force behind the advancements in deep learning. These intermediate layers are responsible for performing intricate computations and extracting essential features from the data. Multiple interconnected hidden layers play a pivotal role in uncovering various hidden characteristics within the data. In the context of image processing, the initial hidden layers are primarily tasked with recognizing fundamental features such as edges, shapes, or boundaries. In contrast, the subsequent hidden layers tackle more complex assignments, like discerning complete objects such as cars, buildings, or people.
- **Output Layer:** The output layer receives input from the preceding hidden layers and generates the ultimate prediction based on the model's acquired knowledge. This layer holds utmost significance as it provides the final outcome. In classification or regression models, the output layer typically consists of a single node. Nevertheless, its configuration is entirely contingent on the specific problem and the design of the model.

2.3.3 ANN architectures:

- Basic artificial neural network models:
 - The Perceptron stands as one of the most elementary and foundational architectures in the world of Neural Networks [1].
 - It is a type of neural network that processes a set of input values, subjecting them to specific mathematical transformations, and ultimately generates an output. These inputs are represented as a vector of real values, and for each attribute, a designated weight is applied, resulting in a linear combination.
 - The aggregated, weighted input is then consolidated into a single value, which subsequently undergoes an activation function.
 - These individual perceptron units can be interconnected to construct more intricate Artificial

Neural Network structures.

- Artificial neural network (ANN) architectures are the structural layouts and arrangements of neural networks used for a variety of machine learning tasks. With time, neural networks have developed, giving rise to a variety of architectures suited for various applications. The following are a few of the most popular ANN architectures [1]:
- **Feedforward Neural Network (FNN):** The simplest kind of neural network is a feedforward neural network (FNN), in which information travels from input layers to output layers in a single direction. An input layer, one or more hidden layers, and an output layer make up this structure.
- **Convolutional Neural Network (CNN):** CNNs are made for tasks involving images. They automatically recognise and extract features from input data using convolutional layers. They are extensively employed in image classification, object detection, and image creation.
- **Recurrent Neural Network (RNN):** RNNs have connections that loop back on themselves and are designed for sequential data. They are appropriate for jobs like time series analysis, speech recognition, and natural language processing.
- **Long Short-Term Memory (LSTM):** Modelling sequences with long-term dependencies is one application where LSTMs, a particular kind of RNN, perform well. LSTMs were developed to solve the vanishing gradient problem. Types of Learning Rules in ANN.
- **Gated Recurrent Unit (GRU):** Another RNN variant that addresses the vanishing gradient problem and has a simpler architecture than LSTMs is the GRU. They are frequently employed for identical sequence modelling tasks.
- **Autoencoders:** These networks are employed in dimensionality reduction and unsupervised learning. They are made up of an encoder network that compresses input data into a lower-dimensional representation and a decoder network that reconstructs the original data.
- **Radial Basis Function Network (RBFN):** RBFNs are frequently employed in tasks involving function approximation and pattern recognition. Radial basis functions are used by them as activation functions.
- **Generative Adversarial Network (GAN):** A generator and a discriminator network, which are both parts of GANs, are trained in together using game theory. They have uses in the creation of realistic data, including images, and can transfer styles.
- **Transformers:** The use of transformers in NLP tasks has grown in popularity. For tasks like machine translation and text generation, they are incredibly effective because they use attention mechanisms to capture relationships between words or tokens in a sequence.
- **Self-Organizing Maps (SOM):** SOMs are employed in data visualisation and unsupervised learning. They maintain the topological characteristics of the input data while mapping high-dimensional data onto a lower-dimensional grid.
- **Capsule Networks (CapsNets):** The hierarchical representation of features in CNNs is to be improved by CapsNets. Their goal is to address some of the drawbacks of conventional CNNs when performing tasks involving part-whole relationships.
- The above are merely a select number of ANN architecture examples; many more variants and hybrid models are available, each well-suited to particular applications and problem domains. The task and the type of data to be used determine the architecture to use. Researchers continue to create and improve neural network architectures to push the limits

of what ANNs can accomplish in machine learning and artificial intelligence.

2.3.4 LEARNING RULES IN ANN

- The application of a learning rule significantly boosts the performance of an Artificial Neural Network. By employing this rule, the network's weights and bias levels are updated when specific conditions are satisfied during the training process. This role in the development of the Neural Network is pivotal [2].
- In the course of training an ANN, learning rules govern the network's response to input data and guide the adjustment of its internal parameters to reduce errors and disparities between the observed and predicted results.
- The selection of a learning rule hinges on the particular task and the network's architectural design. Various learning rules for ANN are shown in Fig. 2.3 and described below:

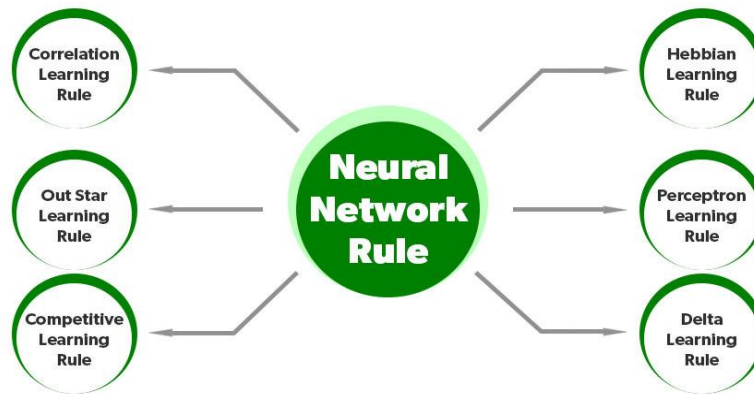


Fig. 2.3 Neural network learning rules [2].

1. Hebbian Learning Rule

- In 1949, Donald Hebb introduced an unsupervised learning algorithm within neural networks, known as Hebbian learning. This method is employed to enhance the connections' weights between nodes in a network. The essence of Hebbian learning lies in its ability to strengthen connections between nodes that fire simultaneously or are active in close succession, leading to a fundamental neural phenomenon:

The following are the core principles of Hebbian learning:

- When two adjacent neurons operate synchronously, their connection weight should increase.
- Neurons working in opposing phases should result in a reduction of the connection weight between them.
- If there is no signal correlation, the weight remains unaltered, with the sign of the weight contingent on the input sign between the nodes.
- When both nodes receive inputs of the same polarity (either positive or negative), it leads to the reinforcement of a positive weight.
- If one node receives positive input while the other receives negative input, a robust negative weight is established.

- **Mathematical Formulation:**

- $\delta_w = \alpha x_i y$
- In this equation, δ_w represents the change in weight, α is the learning rate, x_i is the input vector, and y is the output.

2. Perceptron Learning Rule

- The Perceptron Learning Rule, introduced by Rosenblatt, is an error-correction method designed for single-layer feedforward networks. This supervised learning rule calculates the discrepancy between the desired and actual outputs, and weight adjustments are made only if an output is generated.
- It is Computed as follows:
 - Assume $(x_1, x_2, x_3, \dots, x_n)$ are set of input vectors.
and $(w_1, w_2, w_3, \dots, w_n)$ are set of weights.
 - y = actual output
 - w_0 = initial weight
 - w_{new} = new weight
 - δ_w = change in weight
 - α = learning rate
 - actual output(y) = $w_i x_i$
 - learning signal(e_j) = $t_i - y$ (difference between desired and actual output)
 - $\delta_w = \alpha x_i e_j$
 - $w_{\text{new}} = w_0 + \delta_w$
- Now, the output can be calculated on the basis of the input and the activation function applied over the net input and can be expressed as:
 - $y = 1$, if net input $\geq \theta$
 - $y = 0$, if net input $< \theta$

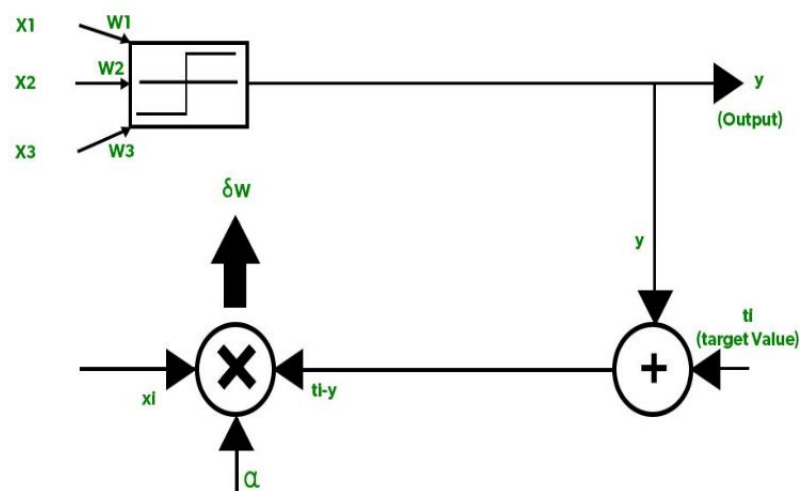


Fig. 2.4 Perceptron learning [2].

3. Delta Learning Rule

- Developed by Bernard Widrow and Marcian Hoff, the method relies on supervised learning and employs a continuous activation function. Commonly referred to as the Least Mean Square (LMS) method, it aims to minimize the error across all training patterns. Operating on a gradient descent approach, it continuously iterates to enhance performance. The product of the error, which is the disparity between the desired and actual output, and the input, determines the weight adjustment for a node.
- **It is Computed as follows:**
 - Assume $(x_1, x_2, x_3, \dots, x_n)$ are set of input vectors.
and $(w_1, w_2, w_3, \dots, w_n)$ are set of weights.
 - y = actual output
 - w_o = initial weight
 - w_{new} = new weight
 - δ_w = change in weight
 - Error = $t_i - y$
 - Learning signal $(e_j) = (t_i - y)y$
 - $y = f(\text{net input}) = \sum w_i x_i$
 - $\delta_w = \alpha x_i e_j = \alpha x_i (t_i - y)y$
 - $w_{new} = w_o + \delta_w$
 - The updating of weights can only be done if there is a difference between the target and actual output (i.e., error) present:
 - case I: when $t = y$
 - then there is no change in weight
 - case II: else
 - $w_{new} = w_o + \delta_w$

4. Correlation Learning Rule

- The correlation learning rule shares a similar principle with the Hebbian learning rule, where the weight between two neighboring neurons becomes more positive if they operate in the same phase at the same time, and more negative if they operate in opposite phases. However, unlike the Hebbian rule, the correlation rule is supervised in nature. It uses the desired or targeted response to calculate the change in weight, ensuring more precise adjustments based on the intended outcome.
- **Mathematical Formulation:**
 - $\delta_w = \alpha x_i y$
 - In this equation, δ_w represents the change in weight, α is the learning rate, x_i is the input vector, and y is the output.

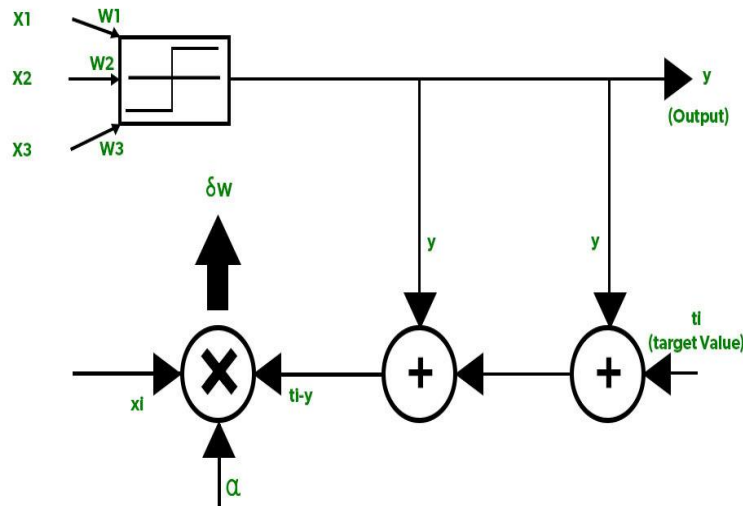


Fig. 2.5 Correlation learning [2].

5. Out Star Learning Rule

- The Outstar Learning Rule, introduced by Grossberg, is a supervised training method used in networks with a layered structure. In this rule, the weights connected to a specific node should match the target outputs of nodes connected through those same weights. The weight change is determined as $\delta w = \alpha(t - y)$, where α is the learning rate, y is the actual output, and t represents the desired output for the nodes in the layer. This rule helps adjust the weights to achieve the desired node activations in the network.

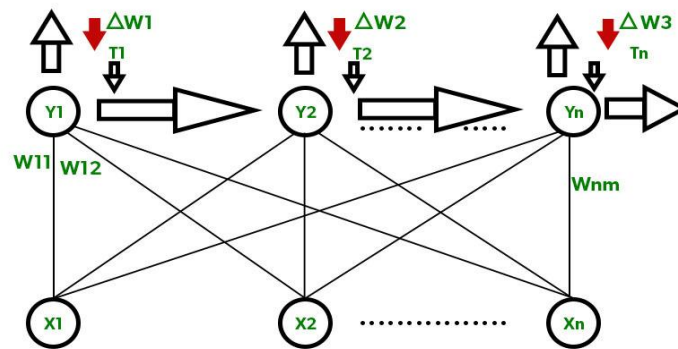


Fig. 2.6 Out star learning [2].

6. Competitive Learning Rule

- This rule is also recognized as the Winner-Takes-All (WTA) rule, and it operates in an unsupervised manner. In this process, all the output nodes engage in a competition to represent the input pattern, and the winner is determined by the node with the highest activation. The winning node is assigned an output of 1, while the others are set to 0.
- Within this framework, there's a set of neurons with initially random weight distributions, and an activation function is applied to a subset of these neurons. Only one neuron can be active at any given time, and it's the winner that undergoes weight updates,

while the weights of the other neurons remain unchanged. This mechanism promotes competition among the neurons, enabling the most suitable one to emerge as the victor.

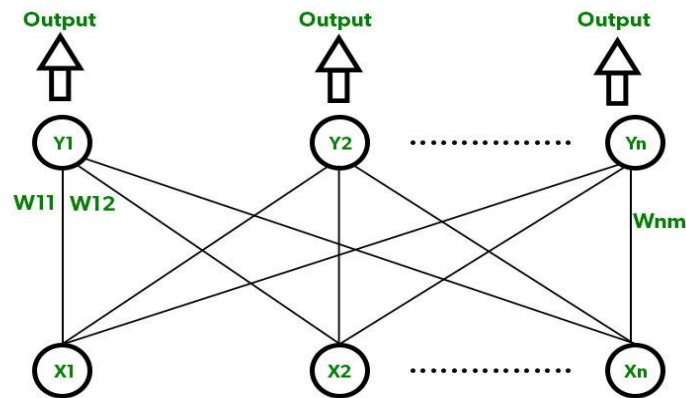


Fig. 2.7 Competitive learning [2].

2.4 ACTIVATION FUNCTIONS IN NEURAL NETWORKS

2.4.1 What is activation function?

- The performance of an Artificial Neural Network (ANN) is determined by a mathematical function known as an activation function [8, 10, 31].
- Activation function plays a pivotal role in determining whether a neuron should be activated or not, and it does so by calculating a weighted sum and then adding a bias. One of the primary objectives of the activation function is to introduce non-linearity into the neuron's output.
- Activation functions can take various forms, including step functions based on norms or thresholds. Activation functions may also serve the purpose of normalizing each neuron's output to fall within a specific range, such as $[0, 1]$ or $[-1, 1]$.
- Efficiency plays a pivotal role in the selection of activation functions, as they need to execute computations rapidly for every data point, especially considering their evaluation across thousands or even millions of neurons. Within the realm of training Artificial Neural Networks, the widely-used technique known as backpropagation adds an extra computational burden to both the activation function and its derivative. This underscores the paramount significance of optimizing their computational efficiency.

2.4.2 Classification of an Activation function

Activation functions can be categorized into three main groups: the binary step function, the linear activation function, and the nonlinear function [10, 11].

1. Binary Step Function:

- A binary step function relies on a threshold for activation. If the input value crosses a certain threshold, the neuron activates and transmits the same signal to the next layer. Mathematically it can be represented as:

$$\begin{aligned} &\text{Binary step} \\ f(x) &= \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases} \end{aligned}$$

- Limitation of the binary step function is that it doesn't support multiple output values, making it unsuitable for classifying inputs into multiple categories.

2. Linear Activation Function

- The linear activation function operates by taking inputs, multiplying them by the respective neuron weights, and producing an output signal proportional to the input. Unlike a step function, which typically provides binary outputs (yes or no), the linear function accommodates a broader range of outputs. However, a noteworthy drawback of the linear activation function is that its derivative remains constant and lacks any connection to the input, posing a significant limitation in certain contexts.

3. Non-Linear Activation Function

- Non-linear activation functions have become integral components in contemporary neural networks, enabling the crucial capability to establish mappings between input and output. This capability is essential for effectively learning complex data patterns. Non-linear activation functions address the limitations posed by the linear activation function, allowing neural networks to capture and represent intricate relationships in data, leading to more robust and versatile learning systems [11].

2.4.3 Most Common Non-linear Activation Functions:

1. **Sigmoid activation:** This function takes any real value as input and outputs values in the range of 0 to 1. The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to 0.0. Figure 1.2(a) shown a sigmoid activation function and the equation is:

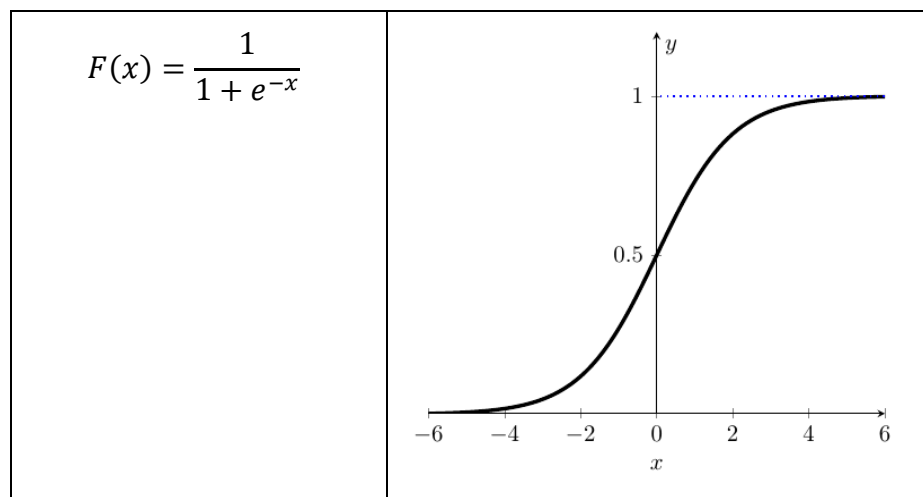


Fig. 2.8 Sigmoid function [4].

The sigmoid has a vanishing gradient problem, which may cause the network to fail to learn more predictions that are accurate.

Vanishing Gradient Problem [4]: When more layers using certain activation functions are added to neural networks, the gradients of the loss function approaches zero, making the network hard to train. For a shallow network with only a few layers that use these activations, this isn't a big problem. However, when more layers are used, it can cause the gradient to be too small for training to work effectively [11].

2. **TanH/Hyperbolic Tangent:** The Tanh function, short for Tangent Hyperbolic function, is frequently employed in the hidden layers of a neural network. Its output values are confined to the range of -1 to 1, resulting in a mean close to zero for the hidden layer activations. This property aids in centring the data by bringing its mean close to zero, which, in turn,

facilitates learning for subsequent layers. However, similar to the sigmoid function, Tanh also faces with the vanishing gradient problem. Figure 1.2(b) shows the graph of the Tanh activation function, and its mathematical expression is [4]:

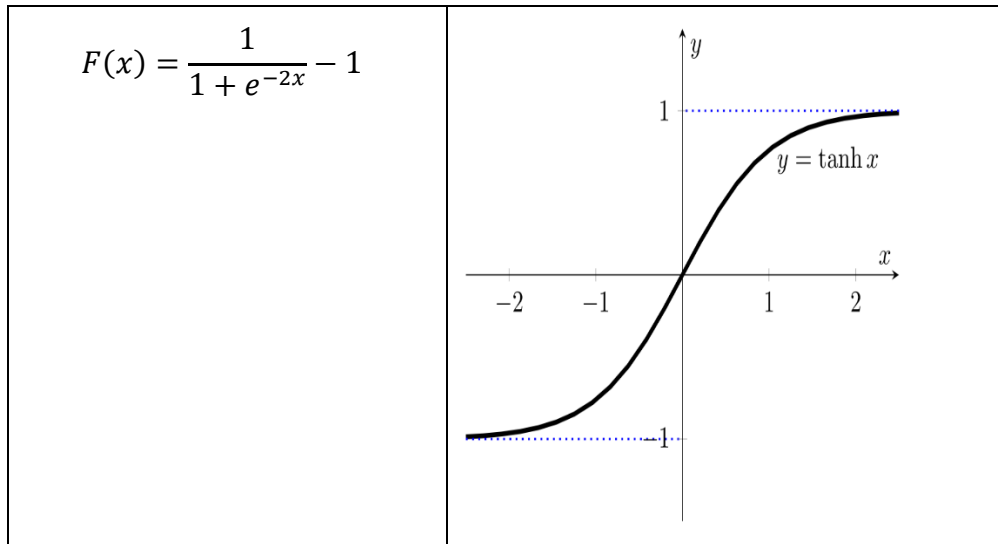


Fig. 2.9 Tanh function.

3. **ReLU (Rectified Linear Unit):** ReLU, which stands for Rectified Linear Unit, is known for its efficient computation, facilitating rapid convergence in neural networks. Despite its linear appearance, ReLU possesses a derivative function that supports effective backpropagation. However, a notable issue arises when inputs dip below or equal to zero, causing the function's gradient to become zero as well. This phenomenon results in the network's inability to back propagate and learn from such instances. You can visualize the graph of a ReLU activation function in Figure 1.2(c), and its mathematical equation is represented as follows:

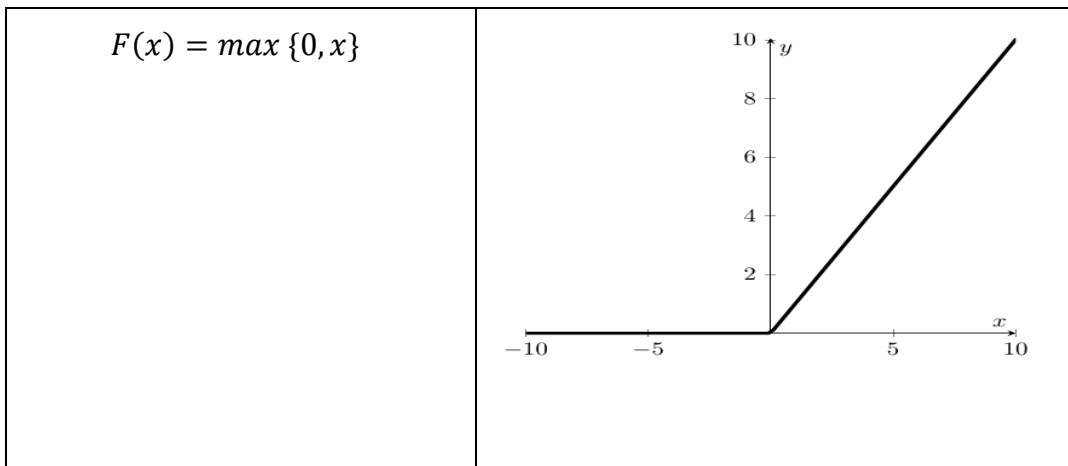


Fig. 2.10 ReLU function.

The challenges associated with this function are commonly referred to as the "Dying ReLU problem."

The Dying ReLU problem: The negative segment of the graph causes the gradient value to become zero. Consequently, during the backpropagation process, certain neurons' weights and biases remain unaltered. This circumstance can give rise to "dead neurons" that never become activated throughout the training process. This issue can lead to a reduction in the model's capacity

to effectively learn and fit to the data.

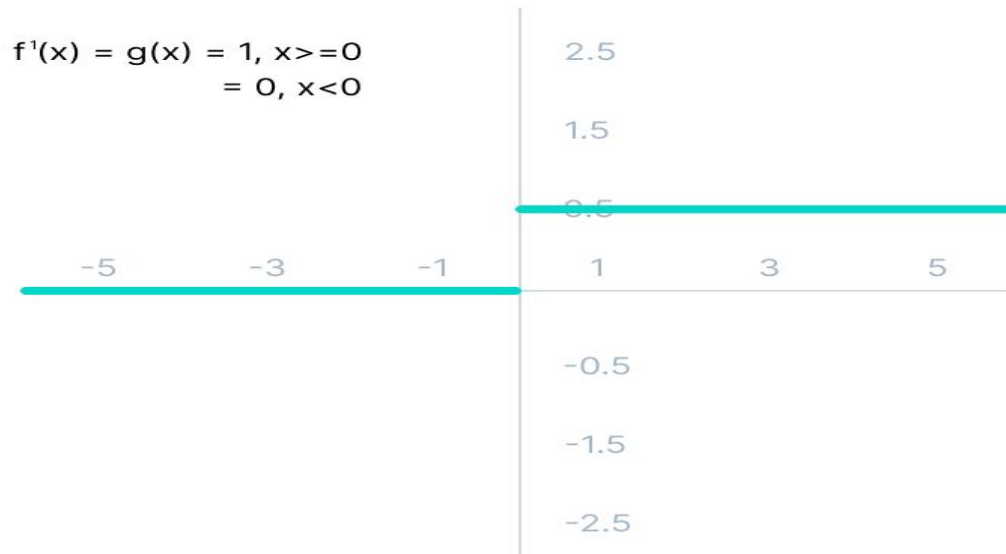


Fig. 2.11 The Dying ReLU problem [4].

4. **Leaky ReLU:** Leaky ReLU stands as an enhanced iteration of the ReLU function, specifically designed to address the Dying ReLU problem. It incorporates a small positive slope within the negative input range, enabling backpropagation for most negative values and avoiding the persistent "dead" neuron issue. It's worth noting that Leaky ReLU doesn't provide a consistent prediction for negative input values. You can visualize the Leaky ReLU activation function graph in Figure 1.2(d), and its mathematical equation is as follows:

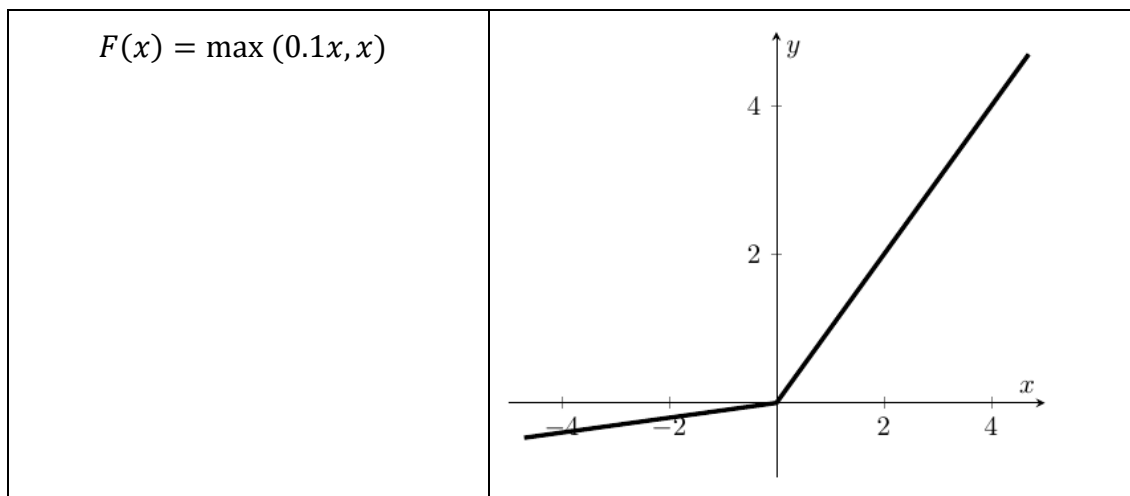


Fig. 2.12 Leaky ReLU [31].

5. **Parametric ReLU:** The concept of Leaky ReLU can be taken a step further by introducing a hyper-parameter that multiplies the input 'x.' This extension, known as Parametric ReLU (PReLU), has shown improved performance compared to standard Leaky ReLU. You can observe the graph of the Parametric ReLU activation function in Figure 1.2(e), and its mathematical equation is represented as follows:

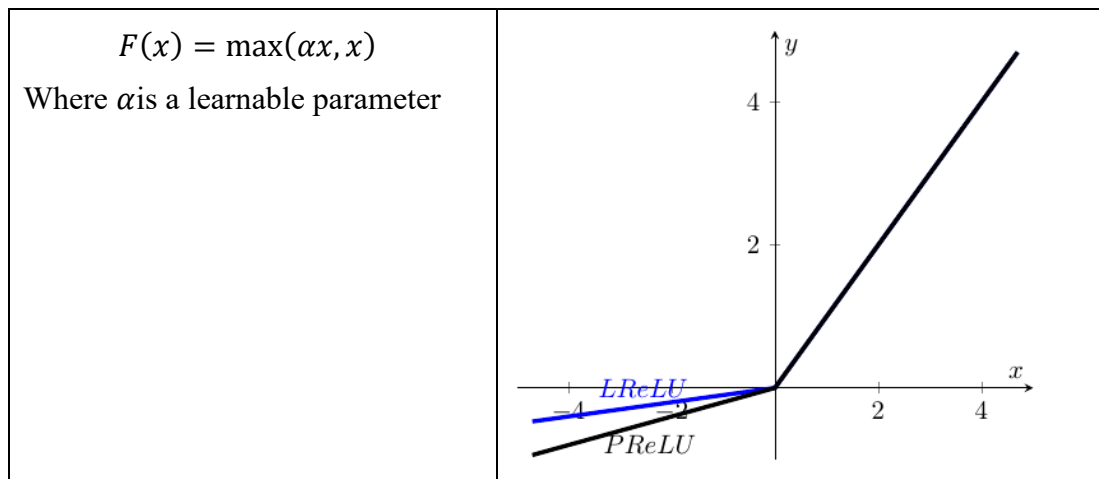


Fig. 2.13 Parametric ReLU.

6. **Softmax:** Softmax stands out by its ability to handle multiple classes, unlike many other activation functions designed for single-class scenarios. It transforms the raw scores for each class into probabilities by normalizing them, ensuring they fall within the range of 0 to 1. This normalization process is particularly valuable when dealing with output neurons, typically in neural networks tasked with categorizing inputs into distinct groups or classes. Visualizing the softmax activation function can be seen in Figure 1.2(f), while its mathematical expression is defined as follows [10, 11]:

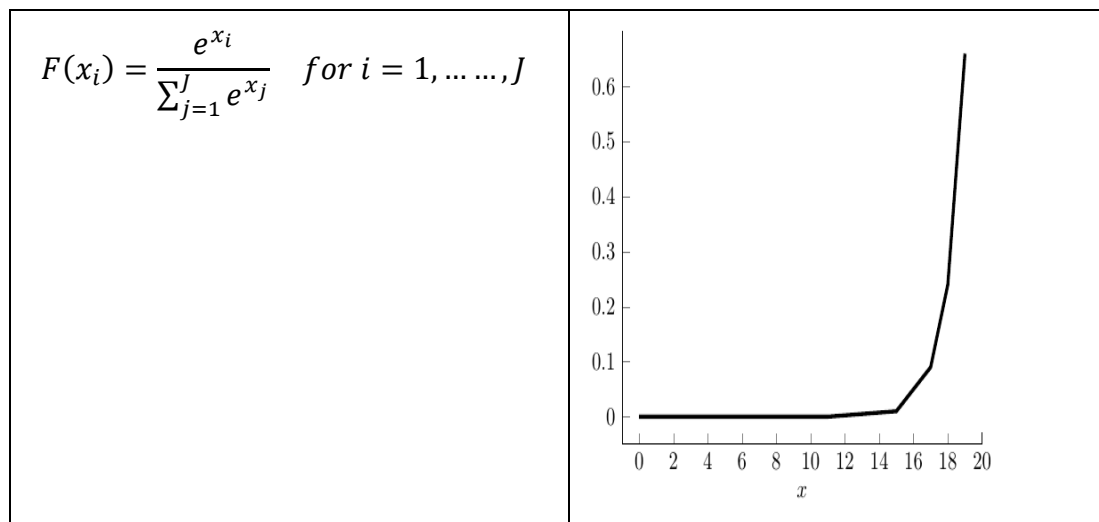


Fig. 2.14 Softmax [27].

- Softmax is most commonly used as an activation function for the last layer of the neural network in the case of multi-class classification.

7. Swish

Swish, a self-gating activation function introduced by researchers at Google, has demonstrated superior performance compared to the commonly used ReLU (Rectified Linear Unit). The following Figure 1.2(g) shows the mathematical expression for the Swish activation function along with its visualization.

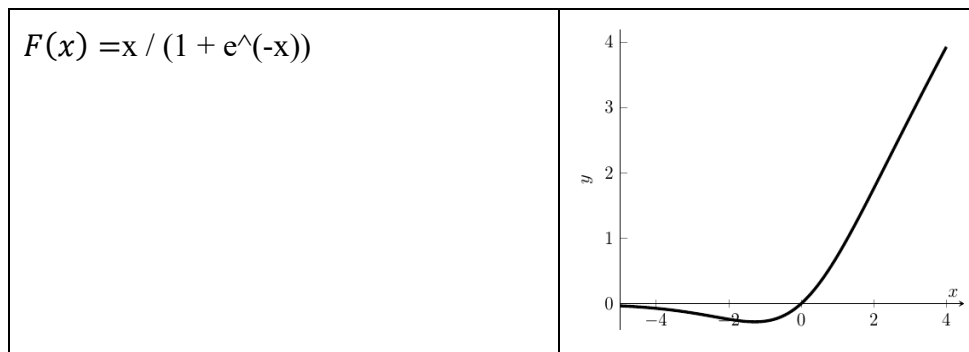


Fig. 2.15 Swish [26].

Self-Evaluations

- Explain Neuron architecture in Artificial Neural Network.
- Explain different ANN architecture.
- What is the principle of Hebbian learning rule?
- What is the activation function? Explain the most common nonlinear activation function?

2.5 PERCEPTRON

The perceptron, a core machine-learning algorithm, is primarily utilized for binary classification tasks in supervised learning and is often equated to an artificial neuron or neural network unit, essential for identifying specific patterns within input data in the realm of business intelligence. This simple yet powerful model operates as a single-layer neural network, hinging on four critical parameters: input values, weights, bias, net sum, and an activation function. Its effectiveness as a supervised learning tool makes it invaluable for making data-driven decisions with precision and efficiency.

2.5.1 Basic Components of Perceptron

The perceptron model serves as a binary classifier and comprises three core components, which are as follows [5]:

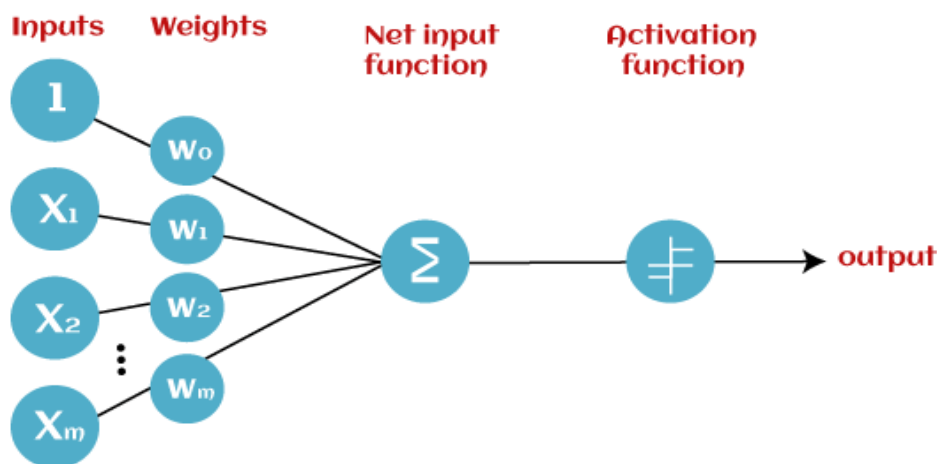


Fig. 2.16 Perceptron model components [5].

- **Input Nodes or Input Layer:** At the heart of the Perceptron model, the input nodes play a central role by receiving the initial data for subsequent processing. Each input node holds a real numerical value, making it the foundation for information intake and analysis.
- **Weights and Bias:** Each input neuron is linked to a weight, signifying the strength of the connection between the input neuron and the output neuron. Moreover, a bias term is introduced in the input layer to impart the perceptron with added flexibility, enabling it to model intricate patterns within the input data more effectively.
- **Activation Function:** The activation function plays a pivotal role in shaping the output of the perceptron, influenced by the weighted sum of inputs and the bias term. Frequently employed activation functions in perceptrons encompass the step function, sigmoid function, and ReLU function.
- **Output:** The perceptron yields a singular binary value, typically either 0 or 1, serving as an indicator of the class or category to which the input data is assigned.
- **Training Algorithm:** In the training process, the perceptron is typically fine-tuned through a supervised learning algorithm like the perceptron learning algorithm or backpropagation. This training methodology involves iteratively adjusting the weights and biases of the perceptron to minimize the disparity between the predicted output and the actual output for a specified set of training examples.

2.5.2 How Does Perceptron Work?

- The perceptron is regarded as a single-layer neural unit with four key parameters. The model initiates by taking the product of input values and their respective weights, followed by the summation of these products to yield the weighted sum. Subsequently, this weighted sum undergoes transformation by the activation function 'f' to produce the desired output. The activation function, often referred to as the step function, is symbolically represented as 'f.' [5]
- The step function, or activation function, plays a critical role in the process by ensuring that the output is appropriately mapped within a specified range, typically between (0, 1) or (-1, 1). It's important to note that the weight of an input signifies the strength of a node, and an input value has the capability to adjust the activation function curve vertically, either upwards or downwards, influencing the output range accordingly.
- The Perceptron model operates in two crucial steps, with the first step involving the multiplication of each input value by its corresponding weight and subsequent addition to compute the weighted sum. This process can be expressed mathematically as:

Weighted Sum ($\sum w_i * x_i$) = $x_1 * w_1 + x_2 * w_2 + \dots + x_n * w_n$

- To enhance the model's performance, a special term known as bias 'b' is introduced and added to the weighted sum:

Weighted Sum + bias ($\sum w_i * x_i + b$).

- In the second step, the weighted sum calculated in the first step is subjected to an activation function, resulting in the output, which can be in the form of a binary value or a continuous value. This relationship is represented as:

Output (Y) = Activation Function (f($\sum w_i * x_i + b$))

2.5.3 Perceptron Learning Algorithm:

The Perceptron Learning Algorithm, often referred to as an Artificial Neuron or a neural network unit, serves as a key component for detecting specific patterns within input data, particularly in the context of business intelligence. This algorithm is celebrated for its simplicity and is considered one of the most elementary forms of Artificial Neural Networks. It operates as a supervised learning method for binary classification tasks, functioning as a single-layer neural network with four primary parameters: input values, weights, bias, the net sum, and an activation function. This foundational model plays a crucial role in making data-driven decisions with precision and efficiency. There are four significant steps in a perceptron learning algorithm [8]:

- First, multiply all input values with corresponding weight values and then add them to determine the weighted sum. Mathematically, we can calculate the weighted sum as follows: $\sum w_i * x_i = x_1 * w_1 + x_2 * w_2 + \dots + w_n * x_n$. Add another essential term called bias 'b' to the weighted sum to improve the model performance $\sum w_i * x_i + b$.
- Next, an activation function is applied to this weighed sum, producing a binary or a continuous-valued output. $Y = f(\sum w_i * x_i + b)$
- Next, the difference between this output and the actual target value is computed to get the error term, E, generally in terms of mean squared error. The steps up to this form the forward propagation part of the algorithm. $E = (Y - Y_{\text{actual}})^2$
- We optimize this error (loss function) using an optimization algorithm. Generally, some form of gradient descent algorithm is used to find the optimal values of the hyper-parameters like learning rate, weight, Bias, etc. This step forms the backward propagation part of the algorithm as shown in the Fig. 2.17.

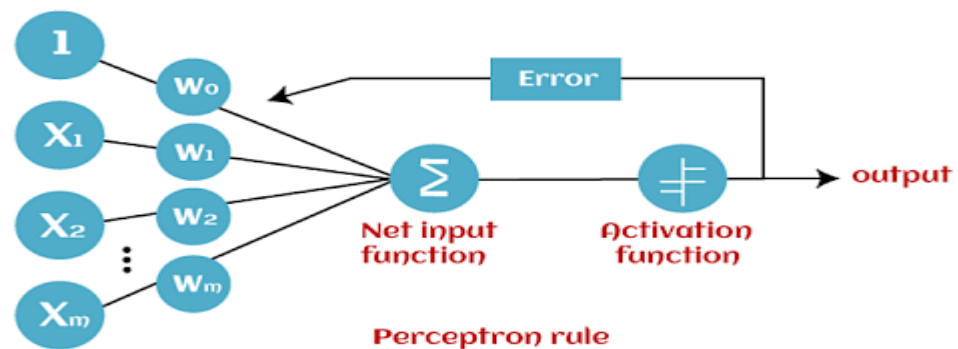


Fig. 2.17 Perceptron model learning [5].

2.54 Perceptron Function

- The Perceptron is a mathematical function that takes an input "x" and combines it with learned weight coefficients to produce an output value denoted as "f(x)."

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

In the equation provided:

- "w" represents a vector of real-valued weight coefficients.

- "b" is the bias, an additive element that shifts the decision boundary without being influenced by the input values.
- "x" denotes a vector of input values, and it plays a crucial role in the perceptrons computation.

$$\sum_{i=1}^m w_i x_i$$

- "m" refers to the number of inputs received by the Perceptron.
- The output of the Perceptron can be represented as binary values, typically "1" or "0," or alternatively as "-1" or "1," depending on the specific activation function used in the model.

2.5.5 Types of Perceptron models

- **The Single-Layer Perceptron model**, among the simplest types of Artificial Neural Networks (ANNs), features a feed-forward structure and incorporates a threshold transfer function. Its primary purpose is to analyze objects that are linearly separable, with binary outcomes. Single-layer perceptrons are limited to learning linearly separable patterns.
- In contrast, the **Multi-Layer Perceptron model** shares similarities with the single-layer perceptron but extends its capabilities by incorporating multiple hidden layers for more complex data processing and feature extraction.

The Multi-Layer Perceptron model, also referred to as the Backpropagation algorithm, operates in two distinct stages:

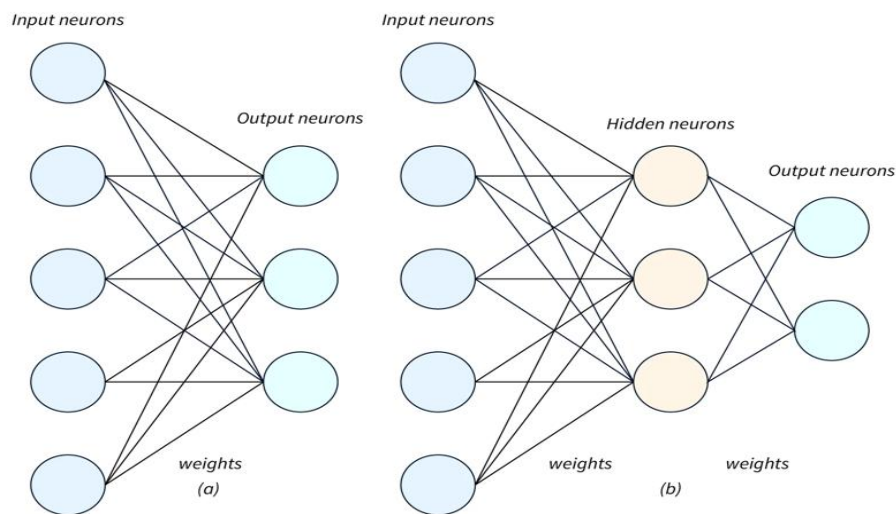


Fig. 2.18 Single layer (a) and Multi-layer model (b).

- **Forward Stage:** This stage commences at the input layer and proceeds through the neural network's hidden layers before reaching the output layer. Activation functions are applied at each layer to transform the input data and propagate it forward.
- **Backward Stage:** In the backward stage, the model adjusts the weight and bias values based on its requirements. Here, the error calculated between the actual output and the desired output is back-propagated from the output layer all the way to the input layer,

facilitating the modification of parameters to improve the model's performance.

- Therefore, a Multi-Layer Perceptron model is composed of multiple artificial neural networks with multiple layers, where the activation function is not limited to linearity, in contrast to a Single-Layer Perceptron model. Instead, various activation functions such as sigmoid, tanh, ReLU, and more can be employed.
- The Multi-Layer Perceptron possesses enhanced processing capabilities, enabling it to handle both linear and non-linear patterns effectively. Additionally, it can be harnessed to implement logic gates like AND, OR, XOR, NAND, NOT, XNOR, and NOR, showcasing its versatility in solving a wide array of computational tasks.

2.5.6 Characteristics of the Perceptron Model:

- **Supervised Binary Classification:** The Perceptron is a machine-learning algorithm designed for supervised learning tasks, particularly binary classification.
- **Automatic Weight Learning:** Perceptron automatically learns the weight coefficients associated with input features during the training process.
- **Neuron Activation:** It evaluates the weighted sum of input features and decides whether the neuron activates or not based on a predefined threshold.
- **Step Activation Function:** The activation function employs a step rule, determining whether the function's value exceeds zero, which influences the neuron's output.
- **Linear Decision Boundary:** The Perceptron establishes a linear decision boundary that separates two classes, typically denoted as +1 and -1, making it suitable for linearly separable data.
- **Threshold-Based Output:** If the cumulative sum of input values surpasses a specified threshold, the Perceptron produces an output signal; otherwise, no output is generated.

2.5.7 Limitations of the Perceptron model:

- **Binary Output:** The Perceptron model produces only binary outputs (0 or 1) due to its hard-edge transfer function, which restricts its ability to represent continuous or multi-class outputs.
- **Linear Separability:** The Perceptron can effectively classify data only when the input vectors are linearly separable. In cases where the data is not linearly separable, the Perceptron struggles to make accurate classifications, limiting its applicability to more complex and non-linear problems.

2.6 Adaline (Adaptive Linear Neural):

Adaline, or Adaptive Linear Neuron, is a single-layer neural network introduced by Widrow and Hoff in 1960. It employs a bipolar activation function, utilizes the delta rule for training to minimize Mean-Squared Error, and allows for adjustable weights and bias. This network is particularly useful for tasks involving linear regression, pattern recognition, and signal processing.

Architecture: Adaline's architecture closely resembles that of a perceptron, with the notable inclusion of a feedback loop designed to compare the actual output to the desired target output. This feedback loop assumes a pivotal role in the process of weight and bias updates within the training algorithm.

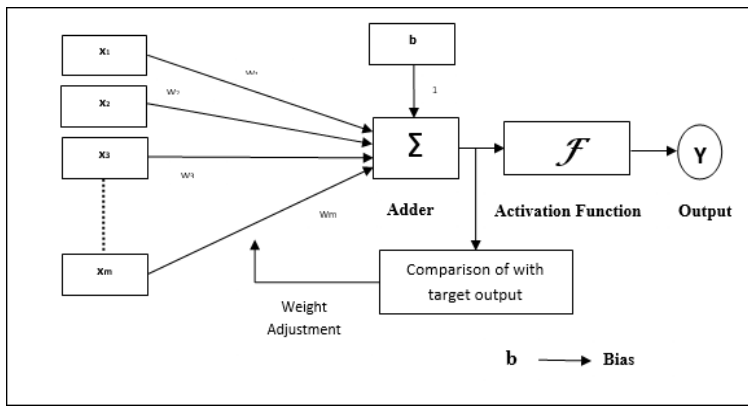


Fig. 2.19 Adaline's architecture [9].

- **Training Algorithm**
- **Step 1** - To initiate the training process, follow these initializations:
 - Initialize the weights.
 - Initialize the bias.
 - Set the learning rate (α) to facilitate ease of calculation and simplicity. Typically, weights and bias are initialized to 0, while the learning rate is set to 1.
- **Step 2** - Proceed to steps 3 through 8 as long as the stopping condition remains false.
- **Step 3** - For each bipolar training pair continue with steps 4 through 6 s.t.
- **Step 4** - Activate each input unit in the following manner –

$$x_i = s_i \quad (i = 1 \text{ to } n)$$

- **Step 5**- Calculate the net input using the following relationship –

$$y_{in} = b + \sum_i^n x_i w_i$$

Where 'b' represents the bias, and 'n' is the total number of input neurons

- **Step 6**- Apply the specified activation function to obtain the final output as follows -

$$f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} \geq 0 \\ -1 & \text{if } y_{in} < 0 \end{cases}$$

- **Step 7**- Update the weight and bias using the following adjustment process –
Case 1 – if $y \neq t$ then,

$$w_i(\text{new}) = w_i(\text{old}) + \alpha(t - y_{in})x_i$$

$$b(\text{new}) = b(\text{old}) + \alpha(t - y_{in})$$

Case 2 – if $y = t$ then,

$$w_i(new) = w_i(old)$$

$$b(new) = b(old)$$

Here 'y' represents the actual output, and 't' stands for the desired or target output and $(t - y_{in})$ is the computed error.

- **Step 8** - Evaluate the stopping condition, which is met when there is either no change in the weights or the most significant weight change during training is smaller than the defined tolerance.

2.6 LMS learning rule:

The LMS (Least Mean Squares) learning rule is a prominent algorithm applied in machine learning and signal processing. It plays a key role in adjusting model parameters, typically weights or coefficients, to minimize the mean squared error between predicted outputs and desired target values. This iterative algorithm finds extensive application in various domains, including adaptive filtering and supervised learning, with the primary goal of enhancing model accuracy by systematically reducing the squared discrepancies between predictions and target values.

The following are the key concepts of LMS learning rule:

- **Objective:** LMS aims to iteratively minimize the mean squared error between predicted and target outputs, typically in supervised learning.
- **Update process:** Each LMS iteration predicts the model's output for a given input, compares it to the target value, calculates the error as the difference, and leverages this error for parameter updates.
- **Weight Update:**
 - The LMS algorithm employs the following formula to update model weights based on the prediction error, using the learning rate α :
 - New Weight = Old Weight + α * Error * Input
 - New Weight: Updated weight.
 - Old Weight: Previous weight.
 - α (learning rate): Regulates the update step size.
 - Applications: Adaptive filtering, signal processing, and neural network training.
 - Challenges: Choosing the right learning rate is critical.

LMS is essential for model refinement, offering versatile applications and customizable parameters for optimization.

2.9 SUMMARY

In summary,

- When considering learning rules in Neural Networks, a standout attribute of Artificial Neural Networks is their capacity for learning. This learning capability mirrors the dynamic nature of the human brain, which modifies its neural structure by strengthening or weakening synaptic connections based on their activity. Information considered more

relevant is associated with stronger synaptic connections. As a result, there exist multiple algorithms for training artificial neural networks, each with its unique advantages and limitations.

- To conclude, we have also examined various learning rules for Artificial Neural Networks (ANN), the Perceptron model, which is a linear supervised machine-learning algorithm primarily designed for binary classification. Additionally, we've delved into the Adaline architecture.

2.10 TERMINAL QUESTIONS

1. Discuss the key component of ANN .
2. Describes various learning rules used in ANN.
3. What is perceptron? Explain its characteristics and limitations.
4. What is learning rules for perceptron model explain?
5. Explain different types of activation function.
6. Write short notes on Adaline (Adaptive Linear Neural) and LMS learning Algorithm

2.11 BIBLIOGRAPHY

1. Braspenning, P., Thuijsman, F., Weijters, A. (1995). Artificial Neural Networks: An Introduction to ANN Theory and Practice. Germany: Springer.
2. Neural-network-architectures. <https://www.v7labs.com/blog/neural-network-architectures-guide>, Accessed on 25-10-23.
3. Activation function. <https://www.geeksforgeeks.org/activation-functions-neural-networks/>, Accessed on 27-10-23
4. Activation function. <https://www.v7labs.com/blog/neural-networks-activation-functions>, Accessed on 27-10-23
5. Perceptron. <https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron#:~:text=Perceptron%20Learning%20Rule,-Perceptron%20Learning%20Rule&text=The%20Perceptron%20receives%20multiple%20input,the%20class%20of%20a%20sample>.
6. Perceptron. <https://www.javatpoint.com/perceptron-in-machine-learning>, accessed on 28-10-23.
7. Learning rules in ANN. <https://www.geeksforgeeks.org/types-of-learning-rules-in-ann/> accessed on 28-10-23.
8. Perceptron learning rule. <https://www.scaler.com/topics/machine-learning/perceptron-learning-algorithm/>, Accessed on 29-10-23.
9. ADLINE.https://www.tutorialspoint.com/artificial_neural_network/artificial_neural_network_supervised_learning.htm, Accessed on 31-10-23.
10. Datta, Leonid. A survey on activation functions and their relation with xavier and he normal initialization. arXiv preprint arXiv:2004.06632, 2020
11. Activation functions <https://medium.com/@snaily16/what-why-and-which-activation-functions-b2bf748c0441>

UNIT-III Feed Forward Neural Network

- 3.1 Introduction
- 3.2 Objectives
- 3.3 Neural network
 - 3.3.1 Feed Forward Neural Network
 - 3.3.2 Feed Forward Neural Network components
 - 3.3.3 Feed Forward Neural Network working
 - 3.3.4 Learning rules in ANN
- 3.4 Delta learning rule
- 3.5 Backpropagation learning rule
 - 3.5.1 Backpropagation
 - 3.5.2 Backpropagation learning algorithm
 - 3.5.3 Why We Need Backpropagation?
 - 3.5.4 Issues with Backpropagation
 - 3.5.5 Backpropagation limitations
 - 3.5.6 Improvement in backpropagation algorithms
 - 3.5.7 Backpropagation momentum algorithm?
- 3.6 Optimization algorithm
 - 3.6.1 Gradient Descent:
 - 3.6.2 Challenges with gradient descent
 - 3.6.3 Conjugate descent:
- 3.7 Radial basis network
 - 3.7.1 Architecture
 - 3.7.2 Working of RBF
 - 3.7.3 Radial Basis Function
 - 3.7.4 Training RBF networks
- 3.9 Summary
- 3.10 Terminal Questions
- Bibliography

3.1 INTRODUCTION

The feedforward neural network, inspired by the human brain, is a fundamental architecture with interconnected layers of processing units, including input, hidden, and output layers, connected by weighted links. This neural network excels at predictive tasks and classification. Learning algorithms are at the core of its operation, adjusting connection weights during training to enable the network to capture complex data patterns and enhance accuracy. Key learning techniques like backpropagation, coupled with optimization methods such as gradient descent, are pivotal in this process, fine-tuning the network's performance. Feedforward neural networks find application across various domains, from image and speech recognition to natural language processing, facilitating robust predictive capabilities and decision-making.

In this unit, we explored the fundamentals of feedforward neural networks, the delta and backpropagation learning rules, the optimization algorithm gradient descent, and radial basis

networks.

3.2 OBJECTIVES

After studying this unit, you should be able to:

- Describe feed forward neural networks and understand how they operate.
- Explain the algorithms of learning in forward neural networks.
- Distinguish between the delta-learning rule and back propagation learning rules.
- Describe the key characteristics of Radial basis network

3.3 NEURAL NETWORK

3.3.1 Feed Forward Neural Network

- The most foundational type of neural network is the feed-forward neural network. In this architecture, input data flows unidirectional, progressing from input nodes through artificial neural nodes, and ultimately exiting through output nodes. Feed-forward neural networks consist of input and output layers, with the potential inclusion of hidden layers, making them versatile and adaptable. Depending on the presence of hidden layers, they can be categorized as single-layered or multi-layered feed-forward neural networks.
 - **Single-layered:** These networks have only one layer of neurons that directly connects input to output. They are simple and suitable for basic tasks but can't handle complex patterns.
 - **Multi-layered:** These networks have multiple layers, including input, hidden, and output layers, making a total of three or more layers. They are capable of learning complex patterns and are used for advanced tasks like image recognition and deep learning.
- The complexity of the function exhibits an inverse relationship with the number of layers in the neural network. The information flow is unidirectional, moving exclusively forward and not backward. In this context, the network's weights remain constant, and they are combined with the input data prior to being processed by an activation function.

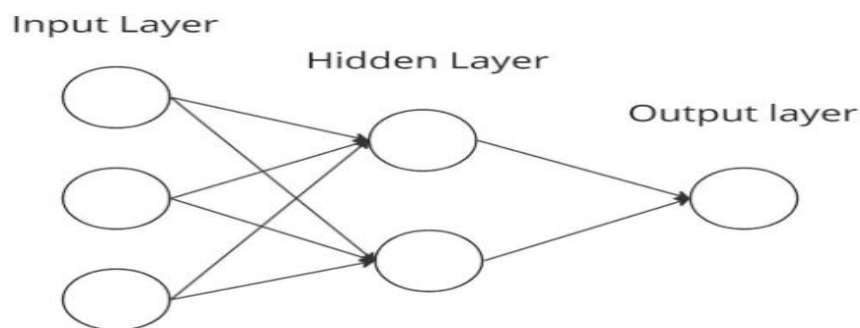


Fig. 3.1 Multi-layered feed-forward neural network [2].

3.3.2 Feed Forward Neural Network components

- **Neurons:** The core element of a neural network is the artificial neuron. This neuron operates in two main steps: it calculates the weighted sum of its inputs, and then it applies an activation function to standardize the result. Activation functions can be either linear or nonlinear. Each

input to a neuron is associated with a specific weight, and these weights need to be learned by the network during the training process. The following is a visual representation of a neuron:

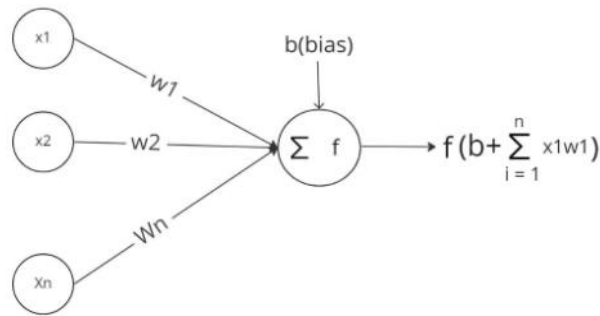


Fig. 3.2 Artificial Neuron [2]

- **Activation function:** The activation function in a neuron acts as the decision-maker for its output. It plays a crucial role in determining whether the neuron learns linear or non-linear decision boundaries. Furthermore, the activation function helps in maintaining a balanced output, preventing it from becoming excessively large as information passes through multiple layers. The three most widely used activation functions are as follows:
 - **Sigmoid:** Maps inputs to a range of 0 to 1, often used in binary classification.
 - **Tanh:** Transforms inputs to a range of -1 to 1, useful for cases with positive and negative values.
 - **ReLU:** Allows positive inputs to pass through unchanged, sets negative inputs to 0, and is commonly used in deep learning.
- **Input Layer:** Receives and relays data, with the number of neurons matching the dataset's features.
- **Output Layer:** Provides predictions, using the sigmoid function for binary classification, Softmax for multiclass and linear units for regression.
- **Hidden Layer:** Sits between input and output layers, processing and modifying data, with the number of layers depending on the model type. Weight adjustments improve prediction accuracy.

3.3.3 Feed Forward Neural Network working

- The feed-forward neural network, when simplified, resembles a single-layer perceptron. It multiplies inputs by weights and sums the results. Output is typically 1 if the sum exceeds a threshold and -1 if it falls below.
- Single-layer perceptrons are used for classification and can adapt through machine learning. They adjust their weights during training using the delta rule, comparing their outputs to expected values.
- Multi-layer perceptrons follow a similar weight adjustment process, known as back-propagation, where hidden layers are updated based on the final layer's output.

3.4 DELTA LEARNING RULE

- It is developed by Bernard Widrow and Marcian Hoff, the method relies on supervised learning and employs a continuous activation function. Commonly referred to as the Least Mean Square (LMS) method, it aims to minimize the error across all training patterns. Operating on a gradient descent approach, it continuously iterates to enhance performance. The product of the error, which is the disparity between the desired and actual output, and the input, determines the weight adjustment for a node.
- **It is Computed as follows:**
 - Assume $(x_1, x_2, x_3, \dots, x_n)$ are set of input vectors.
and $(w_1, w_2, w_3, \dots, w_n)$ are set of weights.
 - Y = actual output
 - w_o = initial weight
 - w_{new} = new weight
 - δ_w = change in weight
 - Error = $t_i - y$
 - Learning signal $(e_j) = (t_i - y)y'$
 - $Y = f(\text{net input}) = \int w_i x_i$
 - $\delta_w = \alpha x_i e_j = \alpha x_i (t_i - y)y'$
 - $w_{new} = w_o + \delta_w$
 - The updating of weights can only be done if there is a difference between the target and actual output (i.e., error) present:
 - case I : when $t = y$
 - then there is no change in weight
 - case II: else
 - $w_{new} = w_o + \delta_w$

3.5 BACKPROPAGATION LEARNING RULE

3.5.1 Backpropagation

- Backpropagation, or backward error propagation, checks for errors by tracing from output nodes back to input nodes in the network.
- Backpropagation is the core of neural network training, involving fine-tuning the network's weights based on the error from the previous iteration. Proper weight adjustments reduce errors, enhance model reliability, and improve generalization. It calculates the gradient of the loss function concerning all network weights, making it a standard method for training artificial neural networks [3].

3.5.2 Backpropagation learning algorithm

- The Backpropagation learning algorithm is a method for training neural networks by propagating errors backward from output to input layers, adjusting weights to minimize error.

- The Backpropagation algorithm in neural networks calculates the gradient of the loss function for an individual weight using the chain rule. It computes one layer at a time, rather than directly calculating everything at once, which makes it efficient. It determines the gradient but doesn't specify how it's utilized, providing a generalized approach rooted in the delta rule.
- To better understand, let's consider the following Backpropagation neural network example diagram:

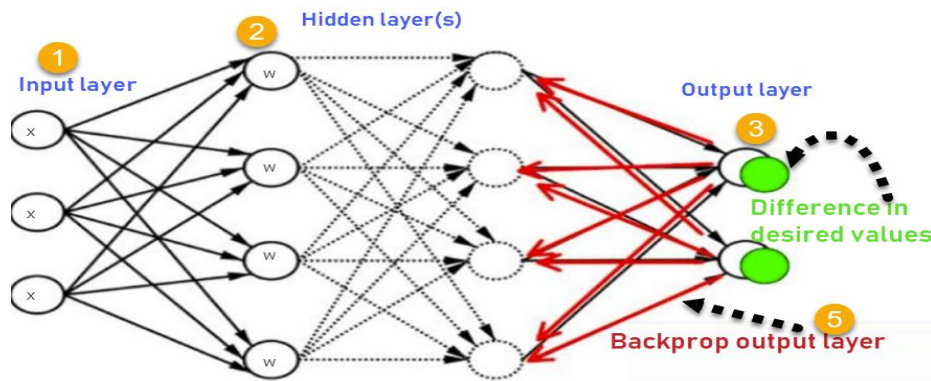


Fig. 3.3 working of backpropagation algorithm [3].

In the Backpropagation process:

1. Inputs X pass through preconnected pathways.
2. The inputs are modelled using real weights W , typically initialized randomly.
3. Calculate the output for each neuron, starting from the input layer, passing through hidden layers, and ending at the output layer.
4. Compute the error in the outputs, which is the difference between the actual output and the desired output.
5. Propagate backward from the output layer to the hidden layers, adjusting the weights to reduce the error.

Repeat this process iteratively until the desired output is achieved through weight adjustments.

3.5.3 Why We Need Backpropagation?

Backpropagation is essential because:

- It's fast, simple, and easy to program.
- Requires minimal parameter tuning.
- Adapts without prior knowledge about the network.
- A standard and reliable method.
- Doesn't need feature-specific information, making it versatile.

3.5.4 Issues with Backpropagation?

Backpropagation (BP) is associated with several challenges, including:

- **Gradient Problems:** Deep networks can suffer from vanishing or exploding gradients, hampering training.
- **Local Minima:** BP may get stuck in local minima during optimization, hindering convergence to the global minimum.

- **Computational Intensity:** Training large networks demands significant computational resources.
- **Data Demands:** Adequate datasets are needed to prevent overfitting, posing limitations when data is scarce.
- **Interpretability:** BP lacks built-in interpretability, making it challenging to understand model decisions.
- **Hyperparameter Sensitivity:** Careful tuning of hyperparameters is necessary, which can be time-consuming.
- **Supervised Learning Focus:** BP is primarily tailored for supervised learning tasks with labelled data.
- **Catastrophic Forgetting:** In some cases, retraining on new data can lead to forgetting previously learned information.

3.5.5 Backpropagation limitations

- Training is time- and resource-intensive
- Gradients issues in deep networks.
- Vulnerability to local minima.
- Requires substantial datasets.
- Lack of inherent interpretability.
- Sensitivity to hyperparameters.
- Mainly suited for supervised learning.
- Potential for catastrophic forgetting.

3.5.6 Improvement in backpropagation algorithms

- Enhancements in Backpropagation (BP) algorithms have addressed their limitations over time:
 - **Improved Weight Initialization:** Improved weight initialization techniques like Xavier/Glorot and He initialization address problems associated with vanishing and exploding gradients.
 - **Advanced Activation Functions:** ReLU and its variants help with training speed and vanishing gradient problems.
 - **Optimization Techniques:** Algorithms like Adam and RMSprop enhance convergence and avoid local minima.
 - **Regularization:** Techniques like dropout and L1/L2 regularization prevent overfitting.
 - **Batch Normalization:** Normalizing layer activations for stability and faster training.
 - **Complex Architectures:** Utilizing architectures like ResNets and CNNs for better performance.
 - **Learning Rate Schedulers:** Adaptive learning rate schedulers aid in quicker convergence.
 - **Ensemble Methods:** Combining networks using ensemble methods for improved accuracy.
 - **Custom Loss Functions:** Tailoring loss functions to specific tasks for efficient training.

- **Transfer Learning:** Leveraging pre-trained models and fine-tuning for task-specific improvements.

These advancements collectively enhance neural network training efficiency and performance across a variety of applications.

3.5.7 Backpropagation momentum algorithm?

- Neural network momentum is a straightforward technique that typically enhances both the speed and accuracy of training. When training a neural network, the objective is to adjust the weights and biases to ensure that, for a given input, the computed output closely matches the desired target values.
- The fundamental update rule for a single weight, without incorporating momentum, is depicted in the formula of Fig. 3.4. In simpler terms, it can be described as follows: "The change in weight (delta increment) connecting node i to node j is determined by multiplying a constant learning rate by the gradient linked to that weight."

$$\Delta w_{ij} = \left(\eta * \frac{\partial E}{\partial w_{ij}} \right)$$

weight increment learning rate weight gradient

Fig. 3.4 Update rule without momentum.

- Incorporating momentum, the adjusted update rule is represented in the following Fig. 3.5. In plain language, it can be explained as follows: "The change in weight (weight delta) is determined by taking the learning rate multiplied by the gradient and adding to it a momentum factor times the weight delta from the previous iteration."

$$\Delta w_{ij} = \left(\eta * \frac{\partial E}{\partial w_{ij}} \right) + (\gamma * \Delta w_{ij}^{t-1})$$

momentum factor weight increment, previous iteration

Fig. 3.5 Update rule with momentum.

- Thus, the backpropagation momentum algorithm is employed to accelerate the weight-tuning process in neural networks. Instead of relying solely on the current gradient to update weights, it considers the influence of past weight changes. This means that an accumulation of previous weight adjustments affects the direction of the current weight update.
- The momentum concept is applied to make the optimization process more efficient by smoothing out weight adjustments and speeding up convergence.

3.6 OPTIMIZATION ALGORITHM

Optimization is a crucial process in machine learning, characterized by iterative model training aimed at achieving the maximum or minimum values for a given function. This fundamental aspect plays a pivotal role in enhancing the performance of machine learning models, leading to improved outcomes. We strive to develop highly accurate models with minimal error

rates. To achieve this goal, there exist various methods and techniques for optimizing a model. Some of the, are discussed below:

3.6.1 Gradient Descent

- Gradient Descent stands out as one of the most prevalent optimization algorithms, essential for the training of machine learning models. Its primary purpose is to minimize the disparities between predicted and actual outcomes. Moreover, Gradient Descent plays a crucial role in the training of Neural Networks, aiding in the iterative refinement of model parameters to enhance predictive accuracy.
- Gradient Descent is a powerful minimization technique for identifying local minima in differentiable functions.
- The optimal way to characterize local minima or maxima of a function using gradient descent is as follows:
- When we follow the negative gradient or move in the opposite direction of the gradient at the current point, we approach a local minimum of the function. Conversely, by moving in the direction of the positive gradient or following the gradient at the current point, we approach a local maximum of the function.

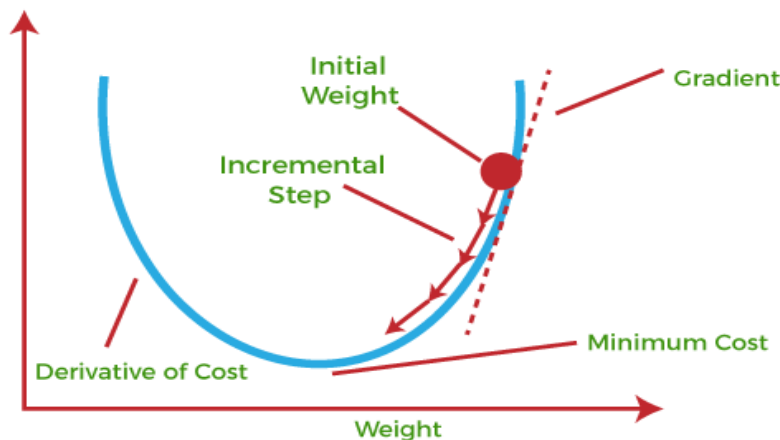


Fig. 3.6 Local minimum of the function[4]

- This entire process is commonly referred to as Descent, also known as steepest descent. The primary aim of gradient descent is to minimize the cost function, which represents the error between expected and actual values. To accomplish this, it carries out two iterative steps:
 - It calculates the first-order derivative of the function to determine the gradient or slope.
 - Then, it moves in the opposite direction of the gradient, increasing the slope from the current point by a factor of alpha, where Alpha represents the Learning Rate. This parameter plays a crucial role in determining the step length during the optimization process.
- Minimizing the cost function necessitates the use of two data points i.e., Direction & Learning Rate. These two data points are utilized in calculating the partial derivative for subsequent iterations, guiding the process towards convergence and the discovery of a local or global minimum.
- The learning rate is the magnitude of the step taken towards the minimum, often a small value adjusted based on the cost function's behavior. A high learning rate implies larger steps but

the risk of overshooting the minimum, while a low learning rate means smaller steps for increased precision at the expense of efficiency.

- Depending on the error calculation in different training models, the Gradient Descent learning algorithm can be categorized into three main types: Batch gradient descent, stochastic gradient descent, and mini-batch gradient descent.

3.6.2 Challenges with gradient descent

- While Gradient Descent is widely favored for optimization, it does come with its set of challenges. Some of these challenges include:
 - **Local Minima and Saddle Point:**
 - In convex problems, Gradient Descent can efficiently locate the global minimum, but in non-convex problems, it can be challenging to pinpoint the global minimum where machine-learning models achieve optimal results.
 - Saddle points exhibit a unique characteristic where the negative gradient is observed on one side of the point, leading to a local maximum on one side and a local minimum on the other.
 - **Vanishing gradients and exploding gradients.**
 - **Vanishing Gradients:** It refers to a situation where the gradient becomes smaller than expected during backpropagation. This causes a significant reduction in the learning rate for earlier layers compared to later layers in the network. As a result, weight parameters in those early layers update to the point of insignificance.
 - **Exploding Gradient:** Exploding gradient is just opposite to the vanishing gradient as it occurs when the Gradient is too large and creates a stable model. Further, in this scenario, model weight increases, and they will be represented as NaN. This problem can be solved using the dimensionality reduction technique, which helps to minimize complexity within the model.

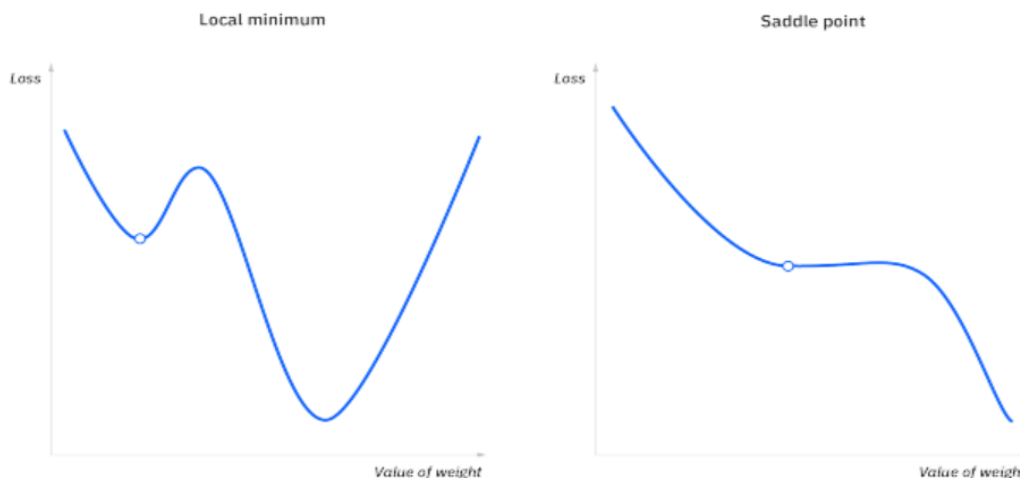


Fig. 3.7 Local minima and saddle point [4]

3.6.3 Conjugate descent:

- Conjugate Gradient is an extension of steepest gradient descent.

- Steepest gradient descent involves stepping in one direction per iteration, which can lead to a zig-zag pattern.
- Conjugate Gradient considers multiple directions simultaneously.
- It prevents retracing old directions by maintaining conjugacy between search directions.
- Conjugate Gradient was introduced in 1952 by Hestenes and Stiefel to simplify the multiple direction search in optimization.

Self-Evaluations

- What is Single-layered and multi-layered feed-forward neural networks.
- What is backpropagation? Explain backpropagation learning algorithm.
- what are the different challenges associated with Backpropagation
- Explain gradient descent methods.

3.7 Radial basis network

3.7.1 Architecture:

RBF are a unique class of feed-forward neural networks characterized by their structure, which includes three key layers: an input layer, a hidden layer, and an output layer. Unlike many other neural network architectures, which often consist of multiple layers and rely on the recursive application of non-linear activation functions, RBF networks exhibit distinctive characteristics. The standard architecture of a Radial Basis Function Neural Network comprises three key layers: the input layer, the hidden layer, and the summation or output layer. The three layers of RBF networks is described below [5]:

- **Input Layer:** The input layer receives the input data, serving as the initial stage of information processing.
- **Hidden Layer:** The hidden layer is the distinguishing feature of Radial Basis Functions Neural Networks. It conducts the core computations and is responsible for the network's unique behavior. It operates differently from most neural networks by using radial basis functions as activation functions.
- **Output Layer:** The output layer is where the final predictions are generated, making RBF networks suitable for various tasks such as classification or regression.
- RBF networks stand out due to their specific choice of activation functions in the hidden layer, which enables them to excel in certain applications where traditional neural networks may have limitations.

3.7.2 Working of RBF:

- Radial Basis Function (RBF) [5] Neural Networks share a conceptual similarity with K-Nearest Neighbor (k-NN) models, yet their practical implementation differs significantly. The core principle behind Radial Basis Functions is to predict a target value for an item based on its proximity to other items with similar predictor variable values. To operationalize this concept, an RBF Network positions one or multiple RBF neurons in a multi-dimensional space corresponding to the predictor variables.
- In this space, we compute the Euclidean distance between the point under evaluation and the

centers of each neuron. Each neuron's weight (influence) is determined by applying a Radial Basis Function (RBF), also referred to as a kernel function, to this distance. The term "Radial Basis" derives from the radius distance, which serves as the input to the function. Mathematically, the weight of a neuron is determined by the RBF applied to the distance: $\text{Weight} = \text{RBF}(\text{distance})$.

- Importantly, the greater the distance between a neuron and the point being evaluated, the weaker its influence or weight on the prediction, reflecting the principle that closer neighbors have a more significant impact on the outcome.

3.7.3 Radial Basis Function :

A Radial Basis Function (RBF) is a real-valued function that solely depends on the distance from the origin. While various types of RBFs exist, the Gaussian function is the most commonly employed and recognized among them.

3.7.4 Training RBF networks

Training a Radial Basis Function Network (RBFN) involves selecting key parameters, including prototypes (center points), beta coefficients (spread of radial basis functions), and output weights. Prototypes can be chosen from training data or through methods like clustering, while beta coefficients are often set to the average distance between points and the center. Output weights are trained using algorithms like gradient descent to minimize prediction errors. The choice of these parameters impacts the network's performance and complexity.

3.8 SUMMARY

In summary,

- The feedforward neural network, inspired by the human brain, forms a foundational architecture with interconnected layers—input, hidden, and output—where processing units are linked by weighted connections. Renowned for excelling in predictive tasks and classifications, this network undergoes learning algorithms that dynamically adjust connection weights during training, thereby enhancing accuracy. Its versatility spans diverse applications, positioning it as a cornerstone in fields such as image recognition and natural language processing.
- Backpropagation serves as a pivotal learning algorithm in neural networks, refining weights through the backward propagation of errors. Complementing this, gradient descent, an optimization algorithm, iteratively adjusts parameters, including neural network weights, with the aim of minimizing a specified function and achieving optimal values. In contrast, Radial Basis Function (RBF) networks present a distinct neural network architecture that leverages radial basis functions to model variable relationships, contributing to their effectiveness in various tasks.

3.9 TERMINAL QUESTIONS

1. What is Feed forward neural network? Discuss the key component of Feed forward neural network.
2. Explain delta-learning rules.
3. What is optimization algorithm? Explain any one-optimization algorithm.
4. What is backpropagation? Explain backpropagation learning algorithms.
5. What is the issues in the backpropagation? Explain improvements of backpropagation.
6. Explain various challenges in gradient descent algorithm.

7. Write short notes on Backpropagation momentum algorithm and Radial basis network.

BIBLIOGRAPHY

1. Backpropagation: Theory, Architectures, and Applications. (2013). United States: Taylor & Francis.
2. Feed-forward-neural-networks. <https://www.tutorialspoint.com/understanding-multi-layer-feed-forward-neural-networks-in-machine-learningActivation>, Accessed on 1-11-23.
3. Backpropagation. <https://www.guru99.com/backpropagation-neural-network.html>, Accessed on 1-11-23.
4. Gradient-descent. <https://www.ibm.com/topics/gradient-descent#:~:text=There%20are%20three%20types%20of,and%20mini%2Dbatch%20gradient%20descent>, Accessed on 2-11-23.
5. Radial-basis-functions .<https://www.simplilearn.com/tutorials/machine-learning-tutorial/what-are-radial-basis-functions-neural-networks>, Accessed on 3-11-23.



Uttar Pradesh Rajarshi Tandon
Open University

Master of Computer Science
MCS-117N Soft Computing

Block-4 GENETTIC ALGORITHM& SOFT COMPUTING 141-177		
UNIT-1	Introduction of Genetic algorithm	141-155
UNIT-2	Population representation, selection criteria and methods	156-168
UNIT-3	Problem solution and Genetic modelling	169-177

Course Design Committee

Prof. Ashutosh Gupta Director, School of Science, UPRTOU Allahabad	Chairman
Prof. Suneeta Agarwal Dept. of Computer Science & Engineering Motilal Nehru National Institute of Technology Allahabad	Member
Dr. Upendra Nath Tripathi Associate Professor DeenDayalUpadhyay Gorakhpur University, Gorakhpur	Member
Dr. Ashish Khare Associate Professor Dept. of Computer Science, University of Allahabad, Prayagraj	Member
Ms. Marisha Assistant Professor (Computer Science) School of Science, UPRTOU Allahabad	Member
Mr. Manoj Kumar Balwant Assistant Professor (Computer Science) School of Science, UPRTOU Allahabad	Member

Course Preparation Committee

Mr. Shivendu Mishra Assistant Professor Dept. of Information Technology Rajkiya Engineering College Ambedkar Nagar U.P. India-224122	Author (Block 1, Block 2: Unit 2, 3, Block 3, & 4)
Dr. Gunjan Singh Assistant Professor (Block 2: Unit 1) Faculty of management and computer application, RBS College, Khandari, Agra-282002.	Author
Prof. Manu Pratap Singh Professor, Department of Computer Science Engineering Dr. Bhimrao Ambedkar University, Agra	Editor
Mr. Manoj Kumar Balwant Assistant Professor (computer science), School of Sciences, UPRTOU Allahabad	Coordinator

© UPRTOU, Prayagraj. 2023

First Edition: July 2023 ISBN: 978-93-48270-40-5

All rights are reserved. No part of this work may be reproduced in any form, by mimeograph or any other means, without permission in writing from the Uttar Pradesh Rajarshi Tandon Open University.



<http://creativecommons.org/licenses/by-sa/4.0/>

Creative Commons Attribution-Share Alike 4.0 International License

Printed by : Chandrakala Universal Pvt.Ltd. 42/7 JLN Road, Prayagraj, 211002

BLOCK-4 INTRODUCTION

Genetic algorithms (GAs) are a powerful search technique inspired by Darwin's theory of evolution, designed specifically to address difficult machine learning optimization problems. By simulating natural selection, GAs efficiently solve problems that would otherwise be time-consuming and resource-intensive. These algorithms find applications in a variety of real-world tasks, including designing electronic circuits, breaking codes, processing images, and creating art.

This block consist of three units, In Unit 1, we will provide a comprehensive overview of genetic algorithms, introducing their fundamental principles, key terminology, and diverse applications. By the end of this unit, you should be well versed in the core concepts of GAs and able to explain their use in various fields. Unit 2 will focus on the differences between global and local optimization and the various encoding methods used to represent problems within genetic algorithms. This understanding is crucial for effectively applying GAs to complex optimization challenges. Finally, in Unit 3, we will delve into the mechanics of genetic algorithms, exploring how GAs maintain a population of potential solutions and refine them over time to discover the optimal solution. Key concepts such as selection, recombination, and mutation will be thoroughly examined. We will also cover additional important aspects, including the convergence of GAs and combinatorial optimization. Throughout this unit, you will learn how GAs model potential solutions as chromosomes, evolving them through genetic operations like crossover, inheritance, inversion, deletion, and mutation. Unlike traditional algorithms, GAs leverage a population-based approach and probabilistic rules, making them uniquely effective for solving complex optimization problems.

UNIT-I INTRODUCTION OF GENETIC ALGORITHM

- 1.1 Introduction
- 1.2 Objective
- 1.3 Genetic algorithm
- 1.4 Basic terminology
- 1.5 Fundamental and basic concepts of GAs
- 1.6 Applications, advantages' and Limitations of GA
- 1.7 Representation of chromosomes and gens
- 1.8 Population representation
- 1.9 Search space
- 1.10 Global vs local optimization
- 1.11 Encoding methods
- 1.12 Summary
- 1.13 Terminal Questions

1.1 INTRODUCTION

One search technique for difficult machine learning optimisation problems is genetic algorithms. Darwin's theory of evolution serves as its inspiration. By mimicking natural selection, it resolves problems that would otherwise take a long time to resolve. This algorithm is applied to many real-world tasks, including creating electronic circuits, decoding codes, processing images, and creating art. This chapter will provide an overview of genetic algorithms, covering their fundamentals, terminology, applications, and limitations.

1.2 OBJECTIVES

After studying this chapter, you should be able to:

- Familiar with the concepts of realm of Genetic algorithm.
- Able to describe the applications of genetic algorithms.
- Described the global vs local optimization and various encoding methods used to represent the problem in genetic algorithm.

1.3 GENETIC ALGORITHM

- **Theory of Natural Evolution:** Charles Darwin introduced the theory of natural evolution in his famous book "On the Origin of Species." He explained that living organisms change over time through natural selection, meaning that those best suited to their environment are more likely to survive and reproduce. Over many generations, this leads to species adapting to their surroundings by developing helpful traits. Consider examples from nature, like the efficient wing shapes of albatrosses and the similar body shapes of sharks and dolphins, shows the amazing results of evolution. These examples illustrate how random changes and natural selection can create complex and specialized features [8].
- **Competition and Selection in Nature:** In a population, individuals compete for resources like

food and shelter, as well as for chances to reproduce and pass on their genes. Natural selection benefits those with advantageous traits, allowing them to survive and reproduce. Over time, this leads to the gradual improvement of the species.

- **Development of Genetic Algorithms:** In 1975, John Holland introduced the idea of using principles of natural evolution to solve optimization problems in his book "Adaptation in Natural and Artificial Systems." This idea led to the development of Genetic Algorithms (GAs), which mimic natural selection to find solutions to complex optimization problems [1, 2].
- Genetic algorithms (GAs) are a computational optimization technique inspired by natural selection and genetics. They solve complex problems by mimicking evolution to improve a population of potential solutions over time. These algorithms work with candidate solutions represented as strings of binary digits or other data structures.
- At the core of a genetic algorithm is a population, which represents a group of potential solutions to a problem. Each individual in the population corresponds to a specific solution, defined by a set of parameters called genes. These genes encode the characteristics or features of the solution and can be represented as binary strings, real-valued numbers, or other data types.
- The genetic algorithm starts with an initial population of individuals, usually created randomly. It progresses through a series of iterations called generations or epochs. During each iteration, individuals undergo operations like selection, crossover, and mutation. These operations simulate natural selection, reproduction, and genetic variation seen in biological evolution [2, 3, 8].
 - During the selection phase of a genetic algorithm, individuals in the current population are assessed using a fitness function. This function measures how effectively each solution addresses the problem. Individuals with higher fitness scores are prioritized for subsequent processing, mirroring the principle of survival of the fittest in nature.
 - **Crossover (Recombination):** Crossover is a genetic process where two chosen individuals exchange genetic information to produce offspring. This mirrors sexual reproduction in nature, combining genetic material from both parents to create diverse offspring.
 - **Mutation:** Mutation introduces random changes in the genetic information of selected individuals. This maintains genetic diversity within the population, exploring different parts of the solution space.
 - **Population Replacement:** After applying genetic operators, a new population replaces the previous one. This cycle repeats for a set number of generations or until a termination condition, like achieving a desired fitness level or reaching a specified iteration limit.
- **Iterative Improvement:** Through successive generations, genetic algorithms explore the solution space, favouring solutions with higher fitness. By iteratively applying selection, crossover, and mutation, the algorithm converges towards an optimal or near-optimal solution.

Genetic algorithms have proven effective in solving diverse optimization problems such as parameter tuning, scheduling, routing, and machine learning. Their strength lies in navigating large solution spaces to discover globally optimal or nearly optimal solutions. This capability is especially valuable for tackling complex or non-linear problem landscapes where traditional optimization

approaches may falter.

How the genetic algorithm works.

1. As an example, let us look at how to optimise a typical task: figuring out the best way to get from home to work.
2. **Encoding the solutions:** Potential fixes in this situation could be represented as combinations of the cities or points along the commuter route. A series of city identifiers, like "A-B-C-D-E-F," where each letter denotes a location (such as a street, intersection, or landmark), could be used, for instance, to represent each possible route.
3. **Initialization:** Make a preliminary population of possible routes first. You have two options: start with an existing route and generate a set of routes at random.
4. **Evaluation:** Examine every route in the population, considering pertinent variables like travel time, distance, and traffic conditions. The evaluation function should measure each route's quality, with lower values denoting better options (such as shorter travel times or less time stuck in traffic).
5. **Selection:** Conduct a selection process to determine which routes will be a part of the next generation. In this case, routes with lower evaluation values are the targets of selection methods, which are designed to favour fitter people. Some popular selection methods are rank-based selection traffic, roulette wheel selection, and tournament selection.
6. **Crossover:** To create new routes, use crossover to combine genetic material from two parent routes. For example, you can choose two parent routes and switch around the route segments to make two new offspring routes.
7. **Mutation:** Through mutation, haphazard alterations are introduced to the routes. Doing so allows you to explore alternative options and stay out of local optima. A mutation operation might add a new city, swap two cities in a route at random, or shuffle the order of a few cities.
8. **New generation:** The offspring resulting from crossover and mutation, along with a select few most fit individuals from the preceding generation, comprise the new population for the subsequent iteration. This makes sure that effective solutions are maintained and continued.
9. **Termination:** After several generations or until a termination criterion is satisfied, the GA continues through selection, crossover, and mutation. A satisfactory solution (such as a route with a predetermined low evaluation value) or a maximum number of iterations can serve as termination criteria.
10. **Final solution:** The best option, usually with the lowest evaluation value, indicates the ideal or almost ideal route for your daily commute once the GA ends.

GAs assist in exploring and evolving the population of routes by iteratively applying selection, crossover, and mutation; this leads to a gradual convergence towards the shortest and most efficient route for your daily commute. It is noteworthy that proper parameter settings, including population size, selection strategy, crossover and mutation rates, and termination criteria, are necessary for GAs to balance exploration and exploitation.

1.4 BASIC TERMINOLOGY OF GENETIC ALGORITHM

The following are some common terms used in genetic algorithms:

- **Population:** It is a subset of all the possible (encoded) solutions to the given problem.

- **Chromosomes:** A chromosome is one such solution to the given problem.
- **Gene:** A gene is one element position of a chromosome.
- **Allele:** It is the value a gene takes for a particular chromosome.

The above is clearer with the Figure 1.1.

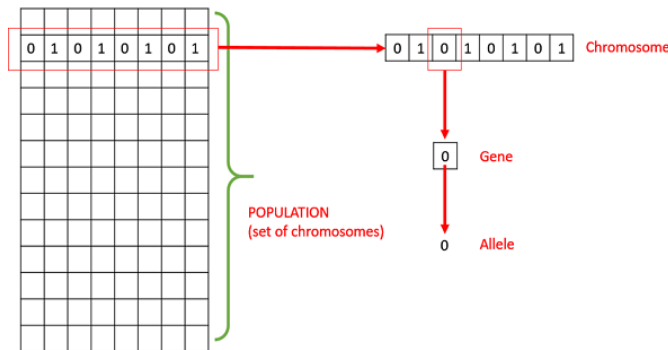


Fig. 1.1: Population, Chromosomes, Gene, and Allele [4].

- **Genotype:** The population in the computation space is called the genotype. Within the computational space, the solutions are expressed in a form that is easily comprehensible and manipulable with a computer system.
- **Phenotype:** The population in the real-world solution space, called a phenotype, comprises solutions represented the same way as in actual situations.
- **Decoding and Encoding:** The genotype and phenotype spaces are the same for simple problems. Most of the time, though, there are differences between the genotype and phenotype spaces. Encoding is changing from the phenotype space to the genotype space. In contrast, decoding is changing a solution from the genotype to the phenotype space. Since decoding occurs frequently in a GA when calculating fitness values, it should be quick.
 - Take the 0/1 Knapsack Problem, for instance. The solutions that only include the item numbers of the things that need to be chosen make up the Phenotype space. It can be represented, as a binary string of length n in the genotype space, where n is the number of items. When an item is selected at position x , a zero indicates it was chosen, and a one indicates the opposite (Fig. 1.2). In this instance, the spaces for genotype and phenotype are distinct.

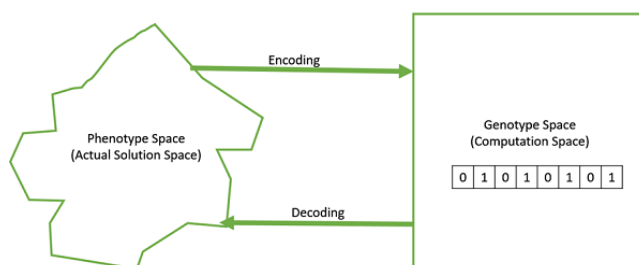


Fig. 1.2 Genotype and Phenotype [4]

- **Fitness Function:** In its most basic form, a fitness function is a function that accepts a solution as input and outputs the solution's appropriateness. Depending on the problem, the fitness and objective functions may differ in some situations or be the same in others.

- **Genetic Operators:** These operators modify the genetic composition of offspring in evolutionary algorithms. They include processes such as crossover, mutation, selection, and more.

1.5 FUNDAMENTAL AND BASIC CONCEPTS

The basic idea of a Genetic Algorithm (GA) is shown in Figure 4.3. We begin with a starting group of individuals (which can be randomly created or influenced by other methods). From this group, we choose parents to mate. We then use crossover and mutation techniques on the parents to create new offspring. These new offspring replace the old individuals in the group, and the cycle continues. In this way, genetic algorithms mimic human evolution to some extent.

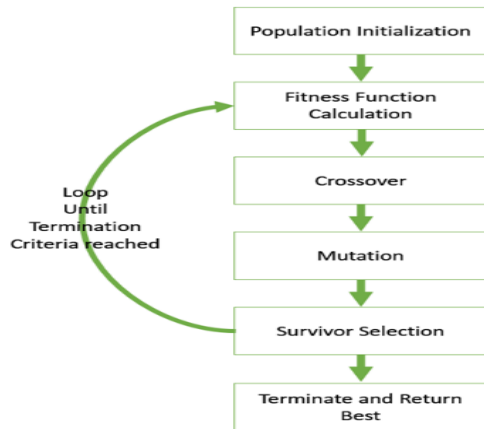


Fig. 1.3 GA basic concept [4].

The following program explains a generalized pseudo-code for a Genetic Algorithm (GA):

```

GA()
  Initialize population
  Find fitness of population
  While (termination criteria is not reached) do
    Parent selection
    Crossover with probability pc
    Mutation with probability pm
    Decode and fitness calculation
  Survivor selection
  Find best
  Return best
  
```

Example: 0-1 Knapsack problem: There are n items, each with a unique weight (w_i) and cost (c_i). There is a knapsack with a total capacity of W . taking as many items as possible without exceeding the knapsack's capacity is the challenge. The following is a more accurate description of this optimization problem.

Maximize :

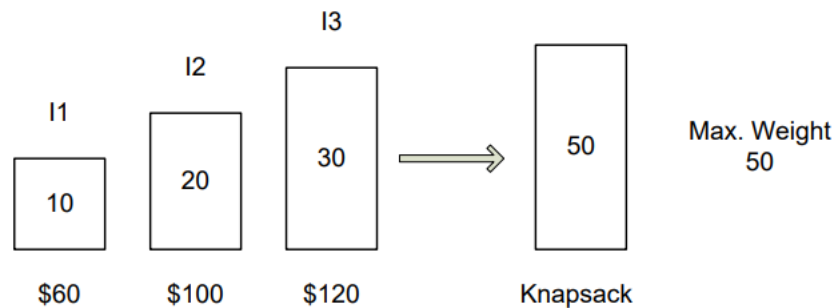
$$\sum_i c_i * w_i * x_i$$

Subject to

$$\sum x_i * w_i \leq W$$

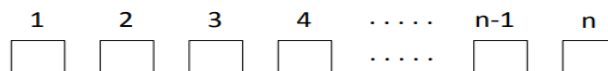
where $x_i \in [0 \dots 1]$

Consider the following, an instance of the 0-1 Knapsack problem

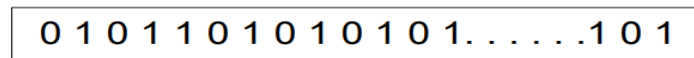


- The following is a statement of the brute force method used to solve the above problem: Pick out a minimum of one item [10], [20], [30], [10, 20], [10, 30], [20, 30], [10, 20, 30]. Therefore, there are $2^n - 1$ trials for n-items.
- [100], [010], [011], [110], [101], [011], [111] is one way to represent the encoding. In this case, a value of 1 denotes that the item is included, while a value of 0 denotes its exclusion.
- Thus for a set of n items, the encoding for the 0-1 Knapsack problem would generally look like this.

Genotype :



Phenotype :



A binary string of n-bits

1.6 Applications, advantages' and Limitations of GA

1.6.1 Applications of Genetic Algorithm

Genetic algorithms have been applied to various complex problems (such as NP-hard problems), machine learning, and the evolution of simple programs. They have also been used in art, evolving pictures and music. Some applications of Genetic Algorithms (GA) include:

1. Nonlinear Dynamical Systems:
 - Predicting and data analysis
 - Robot trajectory planning
2. Genetic Programming:
 - Evolving LISP programs

- Strategy planning
- 3. Biochemistry:
 - Finding the shape of protein molecules
 - Traveling Salesman Problem (TSP) and sequence scheduling
- 4. Art:
 - Functions for creating images
- 5. Control Systems:
 - Gas pipeline control
 - Pole balancing
 - Missile evasion
 - Pursuit
- 6. Design:
 - Semiconductor layout
 - Aircraft design
 - Keyboard configuration
 - Communication networks
- 7. Scheduling:
 - Manufacturing
 - Facility scheduling
 - Resource allocation
- 8. Machine Learning:
 - Designing neural networks (both architecture and weights)
 - Improving classification algorithms
 - Classifier systems
- 9. Signal Processing:
 - Filter design
- 10. Combinatorial Optimization:
 - Set covering
 - Traveling Salesman Problem (TSP)
 - Sequence scheduling
 - Routing
 - Bin packing
 - Graph colouring and partitioning

1.6.2 Advantages of Genetic Algorithm

The advantages of genetic algorithms include:

1. **Parallelism:** They can be easily parallelized.
2. **Reliability:** They provide consistent performance.
3. **Wider Solution Space:** They explore a broad range of solutions.
4. **Complex Fitness Landscape:** They can handle complex fitness landscapes.
5. **Global Optimum Discovery:** They can effectively find global optima.
6. **Multi-Objective Optimization:** They handle problems with multiple objectives.
7. **Function Evaluations Only:** They only require function evaluations, not derivatives.
8. **Adaptability:** They can be easily modified for different problems.
9. **Noise Tolerance:** They handle noisy functions well.
10. **Large Search Spaces:** They manage large, poorly understood search spaces easily.
11. **Multi-Modal Problems:** They perform well on multi-modal problems and return a suite of solutions.
12. **Robustness:** They are robust to difficulties in evaluating the objective function.
13. **No Gradient Information Needed:** They do not require knowledge of gradients.
14. **Discontinuity Handling:** Discontinuities in the response surface have little effect on performance.
15. **Avoid Local Optima:** They are resistant to becoming trapped in local optima.

1.6.3 Limitation of Genetic Algorithm

The limitation of genetic algorithm includes,

- a. The problem of identifying fitness function
- b. Definition of representation for the problem
- c. Premature convergence occurs
- d. The problem of choosing the various parameters like the size of the population, mutation rate, cross over rate, the selection method and its strength.
- e. Cannot use gradients.
- f. Cannot easily incorporate problem specific information
- g. Not good at identifying local optima
- h. No effective terminator.
- i. Not effective for smooth unimodal functions
- j. Needs to be coupled with a local search technique.
- k. Have trouble finding the exact global optimum
- l. Require large number of response (fitness) function evaluations
- m. Configuration is not straightforward

Self-Evaluations

- What is genetic Algorithm? Explain with algorithm the working of genetic algorithm.
- What is fitness function and genetic operators?
- Explain advantages and limitation of genetic algorithm.
- Write the application of genetic algorithm.

1.7 . Representation of chromosomes and gens

- **Chromosomes:** In order to solve the optimisation problem, chromosomes are encoded. The following formats are some of the ways they can be represented:
 - **Binary strings:** Sequences of 0s and 1s.
 - **Real numbers:** Arrays of floating-point numbers.
 - **Permutations:** Ordered lists, often used in combinatorial problems like the traveling salesman problem.
 - **Other structures:** Trees, lists, or custom data structures, depending on the problem.
- **Genes:**
 - The fundamental units of chromosomes are genes.
 - Each gene represents a single parameter or component of the solution.
 - A chromosome's entire gene pool encodes the whole answer.

1.8 Population Representation in Genetic Algorithms

- A population is a collection of chromosomes; at any given moment, it represents a wide range of possible solutions. The following Figure represent an instance of population.

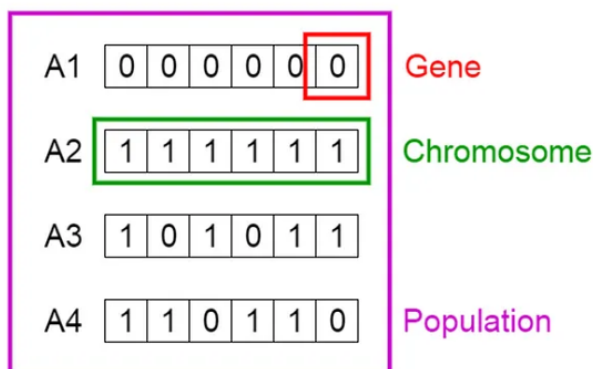


Fig. 1.4 An instance of Population.

- Although a larger population calls for more computational resources, it also increases genetic diversity.
- When interacting with the GA population, there are a few considerations to make.
 - Maintain population diversity to prevent premature convergence.

- Avoid very large populations to prevent GA slowdown.
- Avoid very small populations to ensure a good mating pool.
- Determine the optimal population size through trial and error.
- In a GA, there are two main ways to initialize a population. They are
 - Random Initialization: Assign entirely random solutions to the initial population.
 - Heuristic initialization: Use a well-known heuristic to populate the initial population of the problem.

1.9. Search Space

- When we solve a problem, we look for the best possible solution. The set of all possible solutions is called the search space (or state space). Each point in the search space represents one possible solution, and each solution has a value or fitness for the problem. We want to find the best solution, which is one of these points in the search space [5, 8].
- Finding a solution is like looking for the highest or lowest point in the search space. Sometimes we know the whole search space, but usually, we only know a few points and generate more as we continue looking for the solution.



Fig. 1.5 Example of a search space [8]

- The problem is that the search can be very complicated. It is hard to know where to start and where to look for the solution. There are many methods to find a suitable solution (not necessarily the best one), such as hill climbing, tabu search, simulated annealing, and genetic algorithms. The solutions found by these methods are often considered good because it is usually difficult to prove what the real best solution is?

1.10 Global vs local optimization

- **Local Optima:** A point in a function's domain where the function achieves the lowest (or highest) value in its immediate neighbourhood is referred to as a local optimum. A point x^* is mathematically referred to as a local minimum (maximum) if there is a neighbourhood surrounding it such that, for all x within that neighbourhood, $f(x^*) \leq f(x)$ (or $f(x^*) \geq f(x)$). Stated differently, a point where the function is lower (or higher) than its neighbouring points is referred to as a local optimum.
- **Global Optima:** A function's global optimum is the point in its domain where the function achieves its lowest (or maximum) value throughout. Stated differently, it represents the optimal choice for the optimisation problem as a whole.

- Local and Global Optimum is based on types of Objective Functions.
- An objective function in optimisation is a function that denotes the quantity that has to be maximised or minimised. We can formulate our objective function in various ways, depending on the nature of the problem [6].
- **Convex Objective Function:** A function is convex if:
 - For any two points within its domain, the line segment connecting them lies below or on the graph.
 - The graph is bowl-shaped and opens upward.
 - The slope increases or does not decrease from left to right.
- **Non-convex Objective Function:** A function is non-convex if:
 - It does not satisfy the property of convexity.
 - It can have various shapes, such as multiple peaks, valleys, or irregular patterns.
 - The line segment connecting any two points on the graph does not necessarily lie below or on the graph.
- Since the graph of a convex function is bowl-shaped, the local and global optimum values are equal. Since there is only one valley on the graph, determining the ideal value is simple.

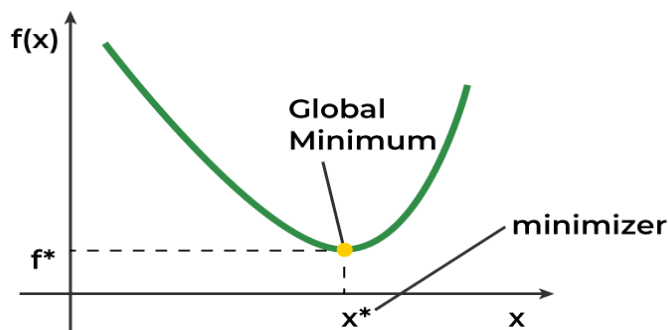


Fig. 1.6 Local (or global) minimum in convex function [6]

- Non-convex optimization is challenging due to multiple local optima.
- Various algorithms are used to explore the search space, including:
 - Genetic algorithms
 - Simulated annealing
 - Particle swarm optimization
- These methods do not guarantee finding the global optimum.
- Their effectiveness depends on the specific problem.
- In the graph below:

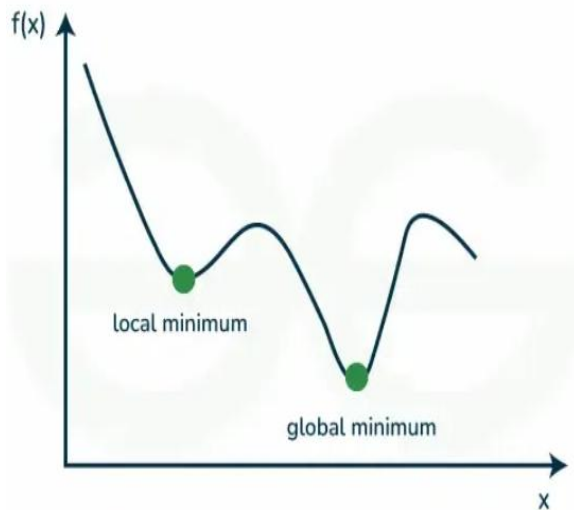


Fig. 1.7Local and global minimum in Non-convex function [6].

- The function is a univariate optimization problem.
- The x-axis shows different values of the decision variable.
- The y-axis shows the function values.
- There are two points where the function reaches a minimum. Accordingly, one is local minimum other is global minimum.

1.11 Encoding Methods in Genetic Algorithm:

The process of representing individual genes is called encoding. Bits, numbers, trees, arrays, lists, and other objects can all be used in the process. The solution to the problem determines the encoding primarily. One can directly encode real or integer numbers, for instance.

- **Binary Encoding:** Most popular encoding techniques. A chromosome is a sequence of ones and zeros, and each position in a chromosome denotes a distinct property of the solution. Say for instance [8]:

Chromosome A	10110010110011100101
Chromosome B	11111110000000011111

- **Octal Encoding:** This encoding uses string made up of octal numbers (0–7). For example [8]:
 - Chromosome1: 03467216
 - Chromosome2: 15723314
- **Hexadecimal Encoding:** This encoding uses string made up of hexadecimal numbers (0–9, A–F). For example [8]:
 - Chromosome1 9CE7
 - Chromosome2 3DBA

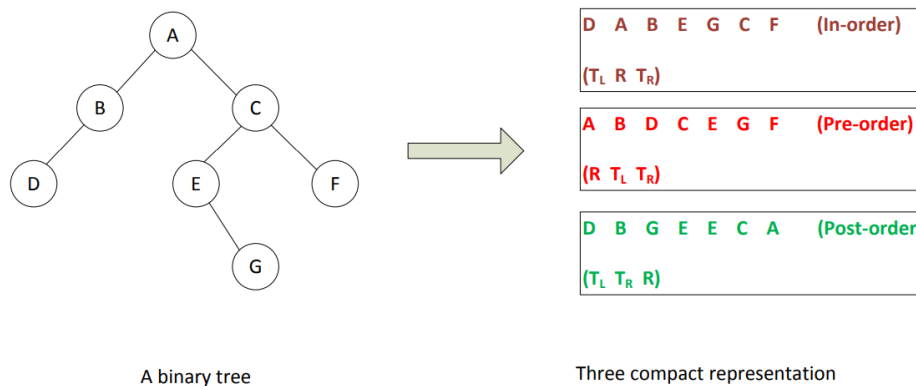
- **Permutation Encoding:** This is helpful for placing orders, like the Travelling Salesman Problem (TSP). Each chromosome in TSP represents a city that needs to be visited, represented by a string of numbers. For instance [8]:

Chromosome A	1 5 3 2 6 4 7 9 8
Chromosome B	8 5 6 7 2 3 1 4 9

- **Value Encoding:** This is used in situations where binary encoding is insufficient and complex values, like real numbers, are utilised. Excellent for certain issues, but frequently calls for the creation of particular crossover and mutation methods for these chromosomes. Say for instance [7,8]:

Chromosome A	1.235 5.323 0.454 2.321 2.454
Chromosome B	(left), (back), (left), (right), (forward)

- **Tree Encoding:**
 - Representing in the form of a tree of objects
 - In this encoding scheme, a solution is encoded in the form of a binary tree.



1.12 SUMMARY

- In summary, in this unit, we learned about genetic algorithms.
- **What is a Genetic Algorithm?** A genetic algorithm is a way to solve problems based on Charles Darwin's idea of natural evolution. It works by selecting the best solutions to create new ones, similar to how nature selects the fittest individuals to reproduce.
- **Key Requirements:**
 - **Solution Representation:** You need a way to describe a solution, like a string of bits, numbers, or characters (e.g., calculating total weight).

- Quality Measurement: You need a way to measure how good a solution is, using something called a fitness function.
- **Basic Principles:**
 - An individual has a set of parts called genes.
 - These genes form a string called a chromosome.
 - The chromosome represents the genotype.
 - The genotype has all the information to create an organism, known as the phenotype.
 - Reproduction is a simple process that happens at the genotype level.
 - Fitness is measured in the real world by how well the phenotype survives or performs.

In short, the following represent genetic algorithm steps:

1. Initialize the population;
2. Calculate Fitness Function;
3. Repeat the following 4to 7 until (Fitness Value != Optimal Value)
4. Selection //Natural Selection, survival of fittest
5. Crossover //Reproduction, propagate favourable characteristics
6. Mutation
7. Calculate Fitness Function

1.13 TERMINAL QUESTIONS

1. What is Genetic algorithm? Explain its application.
2. Describes various key terminology used in genetic algorithm.
3. What is search space? Describe with suitable example.
4. What is local optimization? Explain how it differ with global optimization.
5. What are the steps used to solve any optimization problem using genetic algorithm? Explain it.
6. Explain population, chromosome, gens, and allele.

BIBLIOGRAPHY

1. Melanie Mitchell, “An Introduction to Genetic Algorithms”, 1998.
2. David. E. Goldberg, “Genetic algorithms: In Search, optimization and Machine Learning”, 2006.
3. Genetic algorithm: <https://www.spiceworks.com/tech/artificial-intelligence/articles/what-are-genetic-algorithms/#:~:text=Genetic%20algorithms%20are%20defined%20as,population%20of%20potential%20solutions%20iteratively>. Accessed on 20-06-24.
4. Fundamental of genetic algorithm. www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_fundamentals.htm, Accessed on 27-06-24.
5. Search space. <https://www.obitko.com/tutorials/genetic-algorithms/search-space.php>, Accessed on 02-07-24.
6. Local vs global optimization. <https://www.geeksforgeeks.org/local-and-global-optimum-in-uni-variate-optimization/>, Accessed on 4-07-24.
7. Encoding Methods. <https://www.geeksforgeeks.org/encoding-methods-in-genetic-algorithm/> , Accessed on 5-07-24.
8. Sivanandam, S. N., Deepa, S. N., Sivanandam, S. N., &Deepa, S. N. (2008). Genetic algorithms (pp. 15-37). Springer Berlin Heidelberg.

UNIT-II Population representation, selection criteria and methods

- 2.1 Introduction
- 2.2 Objective
- 2.3 Population representation
- 2.4 Selection criteria and methods
- 2.5 Fitness function
- 2.6 Genetic operators
 - 2.6.1 Crossover
 - 2.6.2 Mutation
 - 2.6.3 Selection
- 2.7. Convergence of GA
- 2.8. Combinatorial optimization
- 2.9. Summary
- 2.10. Terminal Questions

2.1 INTRODUCTION

A genetic algorithm keeps a group of possible solutions and improves them over time to find the best one. It does this by using random processes like selection, recombination, and mutation. Selection means choosing the best solutions more often, based on how good they are. Recombination mixes parts of two different solutions to create new ones. Mutation randomly changes a solution a little bit. We will look in this unit the above concepts and other important parts of genetic algorithms like convergence of GA and combinatorial optimization

2.2 OBJECTIVES

After studying this chapter, you should be able to:

- Familiar with the Genetic algorithm operators like crossover, mutation and selection.
- Able to describe selection criteria, methods, and Fitness evaluation function used in the genetic algorithms.
- Described the convergence of GA and combinatorial optimization

2.3 Population representation

- A population is a collection of chromosomes; at any given moment, it represents a wide range of possible solutions. The following Figure represent an instance of population.

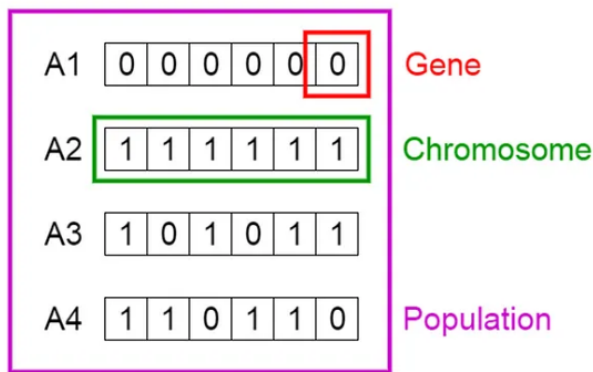


Fig. 2.1 An instance of Population.

- Although a larger population calls for more computational resources, it also increases genetic diversity.
- When interacting with the GA population, there are a few considerations to make.
 - Maintain population diversity to prevent premature convergence.
 - Avoid very large populations to prevent GA slowdown.
 - Avoid very small populations to ensure a good mating pool.
 - Determine the optimal population size through trial and error.
- In a GA, there are two main ways to initialize a population. They are
 - Random Initialization: Assign entirely random solutions to the initial population.
 - Heuristic initialization: Use a well-known heuristic to populate the initial population of the problem [1, 3].
- The entire population should not be initialized using a heuristic. Using only heuristic-based initialization can lead to similar solutions and low diversity in the population.
- Random solutions help drive the population towards optimal solutions.
- Heuristic initialization should be used to seed the population with a few good solutions. The rest of the population should be filled with random solutions.
- Heuristic initialization may only affect the initial fitness of the population.
- It has been observed that In the long run, it is the diversity of solutions that leads to optimal results [3].

Population Models

- There are two popular population models:
 - **Steady state:** In this, each iteration produces one or two offspring in a steady state GA, suppressing one or two population members. Incremental GA is another name for a steady state GA.
 - **Generational:** A generational model generates 'n' offspring, where n is the size of the population, and at the end of each iteration, the entire population is replaced by the new one.

2.4 Selection criteria and methods:

In genetic algorithms, selection is a crucial stage that establishes whether or not a given string will take part in reproduction. The reproduction operator is another name for the selection step. The selection pressure affects how quickly GA converges. Selection serves to concentrate search efforts in favourable areas of space and is motivated by Darwin's "survival of the fittest" theory. We will talk about potential selection techniques in this section [5].

- **Parent Selection:** This is the process of choosing parents to mate and create offspring for the next generation. Figure 1.5 shows the selection process.

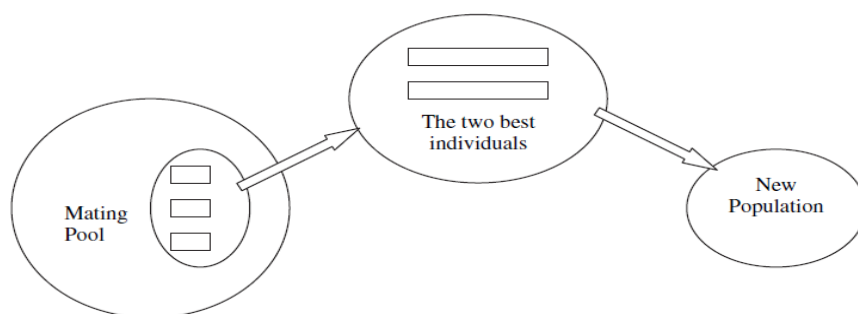


Fig. 2.2 Selection process [8]

- Good parent selection is crucial for the convergence rate of the genetic algorithm (GA) as it leads to better and fitter solutions.
- Avoid allowing one extremely fit solution to dominate the entire population, as this reduces diversity.
- When one solution takes over, leading to similar solutions and loss of diversity, which is undesirable.
- Maintaining good diversity in the population is essential for the success of a GA.

The classification of parent selection methods are as follows [6, 8]:

2.4.1 Fitness Proportionate Selection:

- A popular method of parent selection.
- In this method, each individual can become a parent with a probability proportional to its fitness.
- Fitter individuals have a higher chance of mating and passing their traits to the next generation.
- This strategy applies selection pressure to fitter individuals, leading to the evolution of better individuals over time.

Consider a circular wheel divided into n pies, where n is the number of individuals in the population, and each individual receives a portion of the circle proportional to its fitness level.

There are two possible ways to implement fitness proportionate selection.

- a **Roulette Wheel Selection:** In a roulette wheel selection, the wheel is divided into slices, with each slice's size based on how good the solution is (its fitness). As shown in Figure 2.3, a fixed point is marked on the wheel's edge. The wheel is spun, and the slice that stops in front of the fixed point is chosen as a parent. To select the second parent, the wheel is spun again, and the process is repeated.

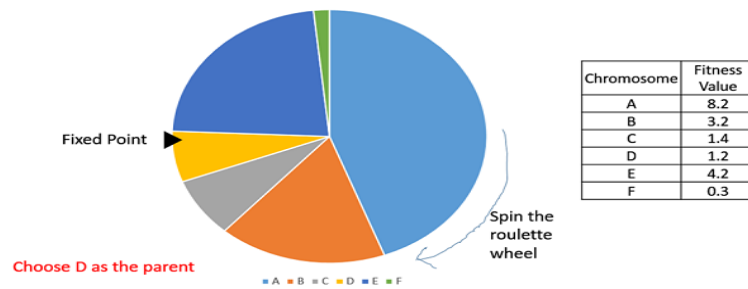


Fig. 2.3 Roulette Wheel Selection [6]

- It is clear that a fitter individual has a larger slice of the wheel, giving it a better chance of stopping in front of the fixed point when the wheel is spun. Therefore, the probability of choosing an individual is directly based on its fitness.

b Stochastic Universal Sampling (SUS):

A somewhat altered variant of the roulette wheel selection previously discussed is called stochastic universal sampling (SUS). The roulette wheel remains the same in terms of dimensions, but instead of employing a single selection point and spinning the wheel repeatedly until all required candidates have been chosen, we spin the wheel just once and make use of several selection points that are evenly distributed around it. In this manner, as shown in the diagram below, every person is chosen simultaneously:

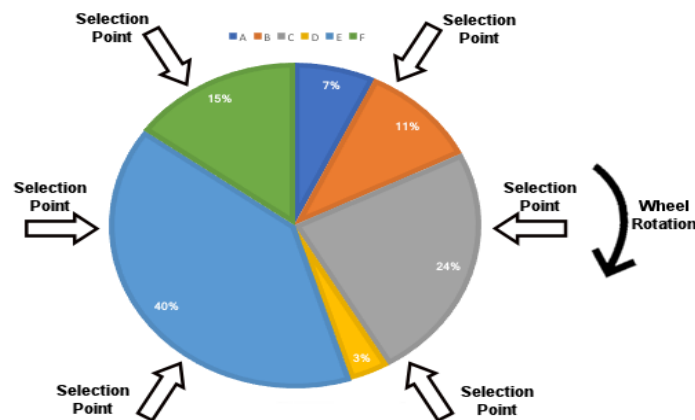


Fig. 2.4 Stochastic Universal Sampling Selection [6]

- It is to be noted that fitness proportionate selection methods don't work for cases where the fitness can take a negative value.

2.4.2 Tournament Selection

- In the K-Way tournament selection process, we randomly choose K people from the population, and then choose the best of them to become parents. The next parent is chosen by repeating the same procedure. Because tournament selection can even be used with negative fitness values, it is also very popular in literature.

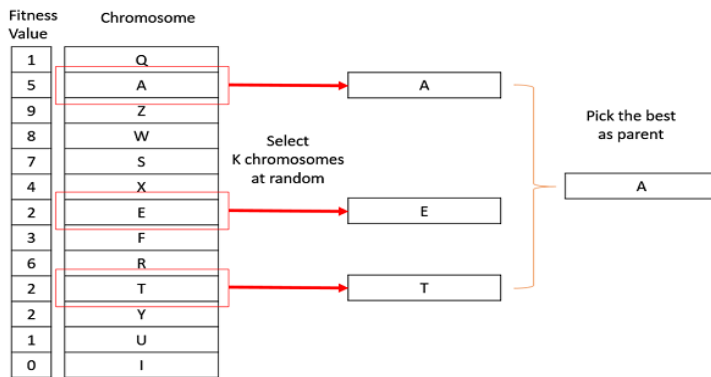


Fig. 2.5 Tournament Selection [6]

2.4.3 Rank Selection

- Rank Selection can handle populations where some individuals have negative fitness scores.
- It's often used when individuals have very similar fitness values, which typically occurs near the end of a genetic algorithm run.
- Rank Selection gives each individual an almost equal chance of being selected, similar to fitness proportionate selection.
- This equal probability reduces the emphasis on selecting fitter individuals, leading to poor parent selection.
- The reduced selection pressure can cause the genetic algorithm to perform poorly in identifying and selecting the best parents.

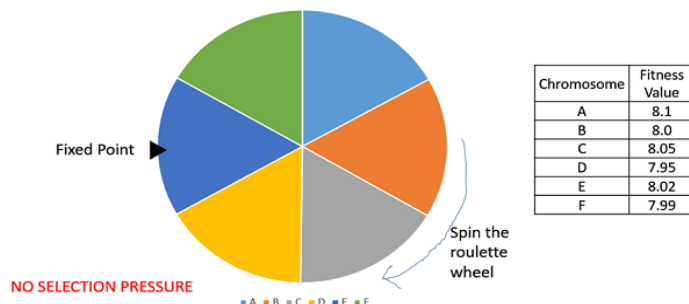


Fig. 2.6 Rank Selection [6]

- In this, each individual is ranked based on their fitness.
- Parents are selected according to their rank, not their fitness values.
- Higher-ranked individuals are given preference over lower-ranked ones in the selection process.

Chromosome	Fitness Value	Rank
A	8.1	1
B	8.0	4

C	8.05	2
D	7.95	6
E	8.02	3
F	7.99	5

2.4.4 Random Selection

- Parents are chosen randomly from the existing population.
- There is no preference for fitter individuals in the selection process.
- This strategy is generally avoided because it doesn't focus on selecting better individuals.

2.5 Fitness function:

- The fitness function can be defined as a function that, given a candidate solution to the problem, computes how "fit" or "good" the solution is in relation to the problem under consideration.
- Since the fitness value computation is done repeatedly in a GA, it should be sufficiently fast. A slow fitness value calculation can negatively affect the GA and cause it to operate abnormally slowly.
- Usually, the fitness function and the objective function are the same, aiming to maximize or minimize a given objective.
- For complex problems with multiple objectives and constraints, an algorithm designer might choose a different fitness function.

A fitness function should possess the following characteristics –

- The fitness function should be quick to compute.
- It must quantitatively measure how good a given solution is or how well individuals can be produced from that solution.

Sometimes, directly calculating the fitness function is not possible due to the problem's complexities. In such cases, fitness approximation is used to suit our needs.

Example:

0	1	2	3	4	5	6	Item Number
0	1	0	1	1	0	1	Chromosome
2	9	8	5	4	0	2	Profit Values
7	5	3	1	5	9	8	Weight Values

Knapsack capacity = 15
Total associated profit = 18
Last item not picked as it exceeds knapsack capacity

- The solution of the selected items (marked with a 1).
- The elements are scanned from left to right until the knapsack is full.

image shows the fitness calculation for a to the 0/1 Knapsack problem.

simple fitness function sums the profit values

Self-Evaluations

- Explain population methods used in genetic algorithm
- Explain different parent selection methods used in GA.

2.6 Genetic operators

2.6.1 Introduction to Crossover

The crossover operator in genetic algorithms mimics biological reproduction, combining genetic material from multiple parents to produce one or more offspring. This promotes genetic diversity and is usually applied with a high probability. We will talk about a few of the most often utilised crossover operators in this section.

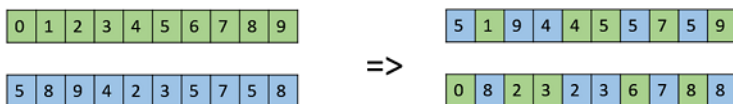
- **One Point Crossover:** In a one-point crossover, the tails of the two parents are switched at a random crossover point to produce new offspring.



- **Multi Point Crossover:** A multipoint crossover is an extension of a one-point crossover in which new offspring are produced by swapping alternating segments.



- **Uniform Crossover:** Each gene is treated independently rather than dividing the chromosome into segments in a uniform crossover. To determine whether or not each chromosome will be present in the offspring, we flip a coin for each one. Additionally, we can tip the odds in favour of one parent so that the child inherits more genetic material from that parent.

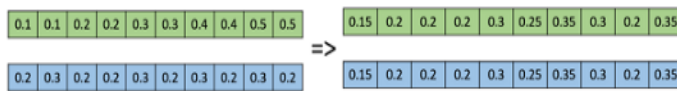


- **Whole Arithmetic Recombination:** It functions by calculating the weighted average of the two parents using the following formulas, and is frequently utilised for integer representations.

$$\text{Child1} = \alpha \cdot x + (1-\alpha) \cdot y$$

$$\text{Child2} = \alpha \cdot x + (1-\alpha) \cdot y$$

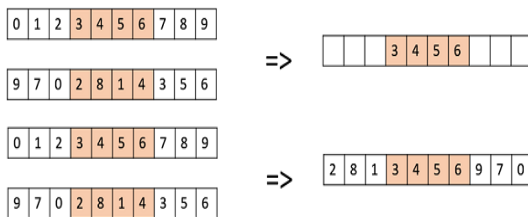
- Obviously, if $\alpha = 0.5$, then both the children will be identical as shown in the following image.



Davis' Order Crossover (OX1): OX1 is used for permutation-based crossovers to transmit relative ordering information to offspring.

The process works as follows:

1. Create two random crossover points in the first parent.
2. Copy the segment between these points from the first parent to the first offspring.
3. Starting from the second crossover point in the second parent, copy the remaining unused numbers to the first child, wrapping around the list.
4. Repeat the process for the second child, reversing the roles of the parents..



Repeat the same procedure to get the second child

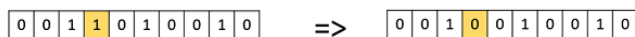
2.6.2 Introduction to Mutation

- Mutation is a small random tweak in the chromosome to get a new solution.
- It is used to maintain and introduce diversity in the genetic population.
- Mutation is usually applied with a low probability.
- If the probability is very high, the genetic algorithm (GA) becomes a random search.
- Mutation is related to the "exploration" of the search space in the GA.
- Mutation is essential for the convergence of the GA.
- Crossover is not essential for the convergence of the GA.

Mutation Operators

Some of the most popular mutation operators are covered in this section.

- **Bit Flip Mutation:** In bit flip mutation:
 - One or more random bits are selected.
 - The selected bits are flipped.
- This method is used for binary encoded genetic algorithms (GAs).



- **Random Resetting:** Random Resetting is an extension of the bit flip for integer representation. In this method:
 - A random gene is chosen.
 - A random value from the set of permissible values is assigned to the chosen gene.

- **Swap Mutation**

- In a swap mutation, we randomly choose two chromosomal locations and swap their values. This is typical of encodings based on permutations.

1 2 3 4 5 6 7 8 9 0 => 1 6 3 4 5 2 7 8 9 0

- **Scramble Mutation**

- With permutation representations, scramble mutation is also quite common. In this, a subset of genes is selected at random from the entire chromosome, and their values are shuffled or scrambled.

0 1 2 3 4 5 6 7 8 9 => 0 1 3 6 4 2 5 7 8 9

- **Inversion Mutation**

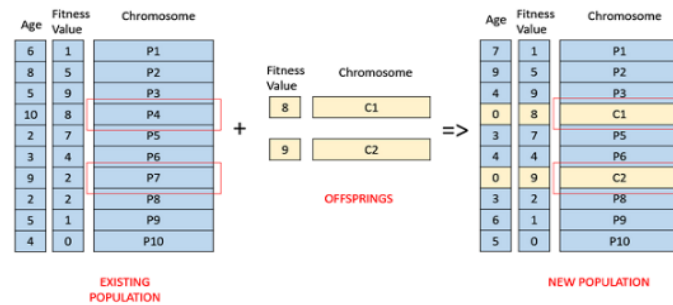
- Similar to scramble mutation, inversion mutation involves choosing a subset of genes, but instead of rearranging the subset, we simply invert every string within the subset.

0 1 2 3 4 5 6 7 8 9 => 0 1 6 5 4 3 2 7 8 9

2.6.3 Survivor Selection

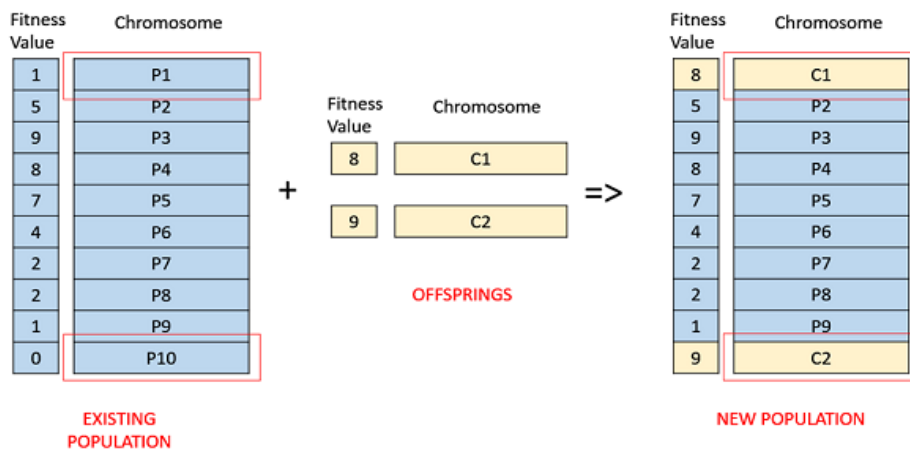
The Survivor Selection Policy determines which individuals are to be removed and which are to be retained in the next generation.

- This policy is crucial because it should:
 - Ensure that the fitter individuals are not removed from the population.
- Maintain diversity within the population.
 - Some Genetic Algorithms (GAs) employ Elitism, which:
 - Means the current fittest member of the population is always carried over to the next generation.
- Ensures that the fittest member of the current population cannot be replaced under any circumstance.
- The simplest policy is to remove random members from the population. However, this approach often leads to convergence issues. Therefore, the following strategies are widely used instead:
- **Age Based Selection** In Age-Based Selection:
 - There is no notion of fitness.
 - It is based on the premise that each individual is allowed in the population for a finite number of generations where it is allowed to reproduce.
 - After this period, the individual is removed from the population regardless of its fitness.
- For example:
 - The age of an individual is the number of generations it has been in the population.
 - The oldest members of the population (e.g., P4 and P7) are removed.
 - The ages of the remaining members are incremented by one.



Fitness Based Selection

- In fitness-based selection:
 - Children tend to replace the least fit individuals in the population.
 - The selection of the least fit individuals can be done using variations of previous selection policies, such as:
 - Tournament selection
 - Fitness proportionate selection
- Example:
 - In a given population, children replace the least fit individuals, e.g., P1 and P10.
 - If individuals like P1 and P9 have the same fitness value, the decision to remove one of them from the population is arbitrary.



2.7 Convergence Criteria

In short, the various stopping condition in genetic algorithms are listed as follows [7]:

- Maximum generations:** The genetic algorithm stops when a specified number of generations have evolved.
- Elapsed time:** The genetic process will end when a specified time has elapsed. If the maximum number of generations is reached before the specified time has elapsed, the process will end.

- **No change in fitness:** The genetic process will end if there is no change in the population's best fitness for a specified number of generations. If the maximum number of generations is reached before the specified number of generations with no changes has been reached, the process will end.
- **Stall generations:** The algorithm stops if there is no improvement in the objective function for a sequence of consecutive generations of length "Stall generations".
- **Stall time limit:** The algorithm stops if there is no improvement in the objective function during an interval of time in seconds equal to "Stall time limit".

Termination or Convergence Criterion

- The termination or convergence criterion brings the search to a halt.
- The following are a few methods of termination techniques [7]
- **Best Individual**
 - The search is halted once the minimum fitness in the population falls below the convergence value.
 - This accelerates the search process.
 - Guarantees that at least one good solution is found.
- **Worst individual:**
 - The search is halted when the fitness of the least fit individuals in the population falls below the convergence criteria.
 - Ensures that the entire population meets a minimum standard.
 - However, the best individual may not be significantly better than the worst.
 - If a stringent convergence value is set, it may never be met, causing the search to terminate after the maximum allowed iterations are exceeded.
- **Sum of Fitness:**
 - The search is considered satisfactorily converged when the sum of the fitness values of all individuals in the population is less than or equal to the convergence value.
 - Ensures that nearly all individuals in the population fall within a particular fitness range.
 - It is recommended to pair this criterion with a weakest gene replacement strategy to prevent a few unfit individuals from disproportionately affecting the fitness sum.
 - The population size must be taken into account when setting the convergence value.
- **Median Fitness:**
 - At least half of the individuals in the population will have a fitness value better than or equal to the convergence value.
 - Provides a good range of solutions to choose from

2.8 Combinatorial optimization:

- Finding an ideal object from a finite set of objects is the goal of combinatorial optimisation.

These potential items are referred to as feasible solutions, and the ideal is optimal.

- For example: we have a connected graph $G = (V, E)$ with nonnegative edge weights.
 - We want to find a spanning tree with the minimum total weight.
 - The set of all possible spanning trees is finite and a feasible solution.
 - However, our goal is to find the spanning tree with the smallest total weight (the optimal solution). This tree is called the minimum spanning tree.
- Therefore, this problem is a type of combinatorial optimization problem.

Here are some other simple examples of typical combinatorial optimization problems:

- **The Traveling Salesman Problem (TSP):**
 - Given: (x, y) positions of N different cities.
 - Task: Find the shortest possible path that visits each city exactly once and returns to the starting city.
- **Bin-Packing:**
 - Given a set of N objects, each with a specified size s_i .
 - Task: Fit these objects into the fewest number of bins, each of size B , as possible.
- Combinatorial optimization solves discrete optimization problems using combinatorial techniques. These problems aim to find the best solution from a finite set of possibilities. In computer science, combinatorial optimization enhances algorithms by:
 - Using mathematical methods to reduce the number of possible solutions.
 - Speeding up the search for the best solution.
- Combinatorial optimization has applications in various fields, including:
 - Artificial intelligence
 - Theoretical computer science
 - Applied mathematics
 - Machine learning
 - Software engineering and many other domains.

2.9 SUMMARY

- Discussed various terminologies and operators used in genetic algorithms in detail.
- Laid the foundation for understanding genetic algorithms.
- Explained how strings are coded to represent the underlying parameter set.
- Detailed the application of selection (reproduction), crossover, and mutation to string populations.
- Briefly discussed combinatorial optimization.

2.10 TERMINAL QUESTIONS

1. Differentiate between phenotype and genotype.
2. Define population and fitness. List a few search strategies.
3. Write note on the types of encoding techniques.
4. Discuss in detail about the selection process of genetic algorithm.
5. How is crossover operation performed? Give examples to illustrate various crossover techniques.
6. Mention the different types of mutation process.
7. Differentiate between Roulette wheel selection and tournament selection.
8. List few terminations search condition of genetic algorithm.

BIBLIOGRAPHY

1. Melanie Mitchell, “An Introduction to Genetic Algorithms”, 1998.
2. Davis E.Goldberg, “Genetic Algorithms: Search, Optimization and Machine Learning”, Addison Wesley, N.Y., 1989.
3. Population representation. https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_population.htm, Accessed on 7-07-24.
4. Fitness function. https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_fitness_function.htm, Accessed on 9-07-24.
5. Selection methods, <https://ebooks.inflibnet.ac.in/csp15/chapter/genetic-algorithms-ii/>, Accessed on 10-07-24.
6. Parent selection, https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_parent_selection.htm, Accessed on 10-07-24
7. Sivanandam, S. N., Deepa, S. N., Sivanandam, S. N., &Deepa, S. N. (2008). Genetic algorithms (pp. 15-37). Springer Berlin Heidelberg.
8. Parent Selection. <https://medium.com/@byanalytixlabs/a-complete-guide-to-genetic-algorithm-advantages-limitations-more-738e87427dbb>, Accessed on 10-07-24

UNIT-III Problem solution and Genetic modelling

- 3.1 Introduction
- 3.2 Objective
- 3.3 Problem solution and Genetic modelling,
- 3.4 Inheritance operator
- 3.5 Crossover operator and its various forms,
- 3.6 Inversion & deletion,
- 3.7 Mutation operator
- 3.8 Generation cycle
- 3.9 Differences & similarities between GA & other traditional method
- 3.10 Summary
- 3.11 Terminal Questions

3.1 INTRODUCTION

Genetic Algorithms (GAs) model potential solutions as chromosomes, evolving them towards an optimal solution through genetic operations. These operations include crossover, which combines portions of two chromosomes in various forms, and inheritance, inversion, deletion, and mutation, which introduce variations. The GA operates through a generational cycle, continually refining solutions. Unlike traditional algorithms, GAs leverage a population-based approach and probabilistic rules to explore a wide solution space, making them uniquely effective for complex optimization problems.

3.2 OBJECTIVES

After studying this chapter, you should be able to:

- Be familiar with the genetic algorithm operators: inheritance and crossover operators.
- Describe inversion, deletion, and mutation operators used in genetic algorithms.
- Explain the generational cycle of genetic algorithms.
- Identify the differences and similarities between genetic algorithms and traditional methods.

3.3 Problem solution and Genetic modelling:

Search heuristics like the genetic algorithm (GA) are frequently employed to provide practical answers to optimisation and search issues. It uses methods like inheritance, mutation, selection, and crossover—all of which are influenced by natural evolution—to produce solutions to optimisation problems. When it comes to solving an unknown problem, genetic algorithms are among the best methods available. They are an extremely broad algorithm; therefore function well in every search area [1, 2].

- GA handles a population of possible solutions. Each solution is represented through a chromosome. Coding all possible solutions into a chromosome is the first step but not the easiest part of a Genetic Algorithm.

- A set of reproduction operators needs to be determined. Reproduction operators are applied directly to the chromosomes. These operators perform mutations and recombinations over solutions.
- Appropriate representation and reproduction operators are crucial as they heavily influence the GA's behaviour.
- Selection compares each individual in the population using a fitness function.
- Each chromosome has a fitness value indicating the quality of the solution it represents. The optimal solution maximizes the fitness function.
- Genetic Algorithms aim to maximize the fitness function. Once reproduction and fitness functions are defined, the Genetic Algorithm follows a basic structure.
- It starts by generating an initial population of chromosomes with diverse genetic materials. A large gene pool is essential to cover the entire search space. The initial population is usually generated randomly. The genetic algorithm then iterates to evolve the population through the following steps [2, 3]:
- **SELECTION:**
 - Select individuals for reproduction.
 - Selection is random but biased by the relative fitness of individuals.
 - Fitter individuals are more likely to be chosen.
- **REPRODUCTION:**
 - The selected individuals breed offspring.
 - New chromosomes are generated using recombination and mutation.
- **EVALUATION:**
 - Evaluate the fitness of the new chromosomes.
- **REPLACEMENT:**
 - Replace individuals from the old population with the new ones.
 - The algorithm stops when the population converges toward the optimal solution.
- The Genetic algorithm process is discussed through the GA cycle in Fig. 3.1.

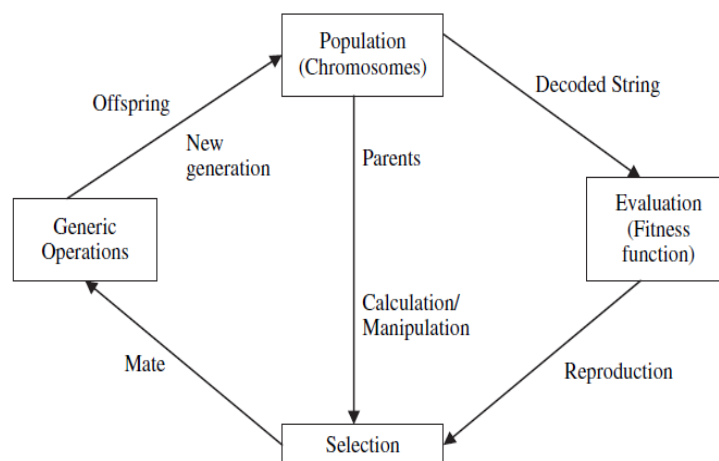


Fig. 3.1 GA cycle

Important criteria for a GA approach: The criteria for the GA approach are listed below [3, 4]

- **Completeness:** Every solution should have its encoding.
- **Non-redundancy:** Each code should correspond to one unique solution.
- **Soundness:** Every code produced by genetic operators should have a corresponding solution.
- **Characteristic perseverance:** Offspring should inherit useful characteristics from their parents.

Thus the basic four steps to solve a problem using Genetic Algorithm are:

1. Representing the problem.
2. Calculating fitness.
3. Controlling the algorithm with various variables and parameters.
4. Representing the result and determining the termination method.

3.4 Inheritance operator:

In inheritance operator in GA passes genetic information from parents to offspring. The purpose of inheritance operators are:

- Ensure offspring inherit useful characteristics from parents.
- Maintain genetic diversity for effective solution exploration..

The main inheritance operators are:

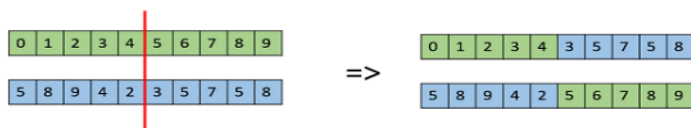
- **Crossover (Recombination):**
 - Combines parts of two parent chromosomes to create offspring.
- **Mutation:** Introduces random changes to a chromosome to maintain genetic diversity and explore new solution spaces.

The details of the above operators are discussed in the following sections.

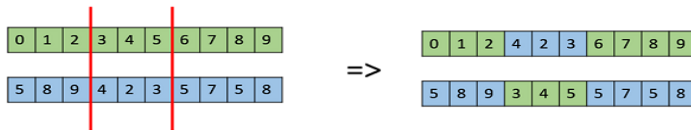
3.5. Crossover operator and its various forms

The crossover operator in genetic algorithms mimics biological reproduction, combining genetic material from multiple parents to produce one or more offspring. This promotes genetic diversity and is usually applied with a high probability. We will talk about a few of the most often utilised crossover operators forms in this section.

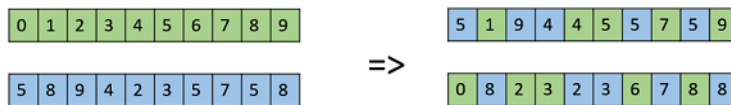
- **One Point Crossover:** In a one-point crossover, the tails of the two parents are switched at a random crossover point to produce new offspring.



- **Multi Point Crossover:** A multipoint crossover is an extension of a one-point crossover in which new offspring are produced by swapping alternating segments.



- **Uniform Crossover:** Each gene is treated independently rather than dividing the chromosome into segments in a uniform crossover. To determine whether or not each chromosome will be present in the offspring, we flip a coin for each one. Additionally, we can tip the odds in favour of one parent so that the child inherits more genetic material from that parent.

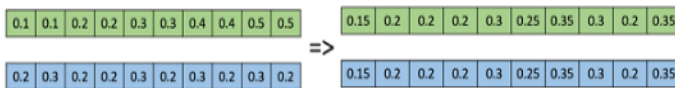


- **Whole Arithmetic Recombination:** It functions by calculating the weighted average of the two parents using the following formulas, and is frequently utilised for integer representations.

$$\text{Child1} = \alpha \cdot x + (1 - \alpha) \cdot y$$

$$\text{Child2} = \alpha \cdot x + (1 - \alpha) \cdot y$$

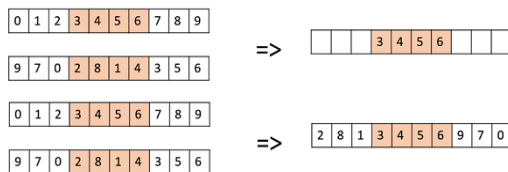
- Obviously, if $\alpha = 0.5$, then both the children will be identical as shown in the following image.



- **Davis' Order Crossover (OX1):** OX1 is used for permutation-based crossovers to transmit relative ordering information to offspring.

The process works as follows:

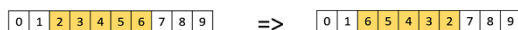
1. Create two random crossover points in the first parent.
2. Copy the segment between these points from the first parent to the first offspring.
3. Starting from the second crossover point in the second parent, copy the remaining unused numbers to the first child, wrapping around the list.
4. Repeat the process for the second child, reversing the roles of the parents..



Repeat the same procedure to get the second child

3.6 Inversion & Deletion:

- **Inversion:** Reverses the order of a segment of a chromosome, aiding in the exploration of new solution regions by altering gene sequences. For example:



- **Deletion:** Removes a segment of a chromosome to simplify or focus the search by eliminating less useful genetic material [4].
- **For Example:**
 - **0010010 (before deletion)→0010--0(At deletion)→0010100** (Duplication-previous bits are duplicated)
 - **0010010 (before deletion)→ 0010--0(At deletion)→0010110** (regeneration-deleted bits are generated randomly)

Self-Evaluations

- What are the important criteria for a GA approach?
- Explain selection and reproduction part of GA life cycle.

3.7 Mutation Operators:

- Mutation is a small random tweak in the chromosome to get a new solution.
- It is used to maintain and introduce diversity in the genetic population.
- Mutation is usually applied with a low probability.
- If the probability is very high, the genetic algorithm (GA) becomes a random search.
- Mutation is related to the "exploration" of the search space in the GA.
- Mutation is essential for the convergence of the GA.
- Crossover is not essential for the convergence of the GA.

Some of the most popular mutation operators are covered in this section.

- **Bit Flip Mutation:** In bit flip mutation:
 - One or more random bits are selected.
 - The selected bits are flipped.
- This method is used for binary encoded genetic algorithms (GAs).

0	0	1	1	0	1	0	0	1	0
---	---	---	---	---	---	---	---	---	---

 \Rightarrow

0	0	1	0	0	1	0	0	1	0
---	---	---	---	---	---	---	---	---	---

- **Random Resetting:** Random Resetting is an extension of the bit flip for integer representation. In this method:
 - A random gene is chosen.
 - A random value from the set of permissible values is assigned to the chosen gene.
- **Swap Mutation**
 - In a swap mutation, we randomly choose two chromosomal locations and swap their values. This is typical of encodings based on permutations.

1	2	3	4	5	6	7	8	9	0
---	---	---	---	---	---	---	---	---	---

 \Rightarrow

1	6	3	4	5	2	7	8	9	0
---	---	---	---	---	---	---	---	---	---

- **Scramble Mutation**

- With permutation representations, scramble mutation is also quite common. In this, a subset of genes is selected at random from the entire chromosome, and their values are shuffled or scrambled.

0 1 2 3 4 5 6 7 8 9 => 0 1 3 6 4 2 5 7 8 9

- **Inversion Mutation**

- Similar to scramble mutation, inversion mutation involves choosing a subset of genes, but instead of rearranging the subset, we simply invert every string within the subset.

0 1 2 3 4 5 6 7 8 9 => 0 1 6 5 4 3 2 7 8 9

3.8 Differences & similarities between GA & other traditional method

- The difference between GA and other traditional methods are given bellow [4]

Genetic Algorithm	Traditional Algorithm
The genetic algorithm is a type of algorithm that is based on the principle of genetics and natural selection to solve optimization problems.	The traditional algorithm is a step by step procedure to follow in order to solve a given problem.
The genetic algorithm helps to find the optimal solutions for difficult problems.	Traditional algorithm provides a step by step methodical procedure to solve a problem.
More Advanced	Less Advanced than genetic algorithm
Genetic Algorithm is used in fields such as research, Machine Learning and, Artificial Intelligence	Traditional algorithm is used in fields such as Programming, Mathematics, etc.
A search space is a set of all possible solutions to the problem. Genetic Algorithms use several sets in a search space.	Traditional Algorithms maintain only one set in a search space.
Genetic Algorithms just require one objective function to calculate the fitness of an individual	Traditional Algorithms require more information to perform a search
Genetic Algorithms can work in parallel (calculating the fitness of the individuals are independent)	Traditional Algorithms cannot work in parallel
In Genetic Algorithms multiple optimal solutions can be obtained from different generations	Traditional Algorithms can only produce one solution in the end

Genetic Algorithms are not guaranteed to produce global optimal solutions as well but are more likely to produce global optimal solutions because of the genetic operators like crossover and mutation	Traditional Algorithms are not more likely to produce global optimal solutions
Genetic algorithms are probabilistic and stochastic in nature.	Traditional algorithms are deterministic in nature
Genetic Algorithms, with the right parameter setting, can handle Real-world problems very well because of the large solution space.	Real-world problems are multi-modal (contains multiple locally optimal solutions), the traditional algorithms don't handle well these problems

- The similarities between genetic and traditional algorithms, are as follows:
- **Problem Solving:** Both help solve complex problems.
- **Searching for Solutions:** Both involve searching through many possible solutions to find the best one.
- **Optimization:** Both aim to find the best solution based on a specific goal.
- **Step-by-Step Improvement:** Both use steps to gradually improve solutions.
- **Set Procedures:** Both follow a set of steps to find solutions.
- **Evaluation:** Both judge potential solutions based on certain rules or goals.
- **Finding the Best Solution:** Both can eventually find the best or a very good solution.

3.9 Generation cycle of GA:

- Generation cycle is the diagrammatic representation of the genetic algorithm.
- It consist of all the required steps such as Selection, Crossover, and Mutation applied to a populations. The objective of GA is to search optimized solution for a problem from a set of possible solutions.
- The generation cycle is repeated until we reached on optimized solution
- The generation cycle or flow chart of genetic algorithm is shown in Fig. 3.2.

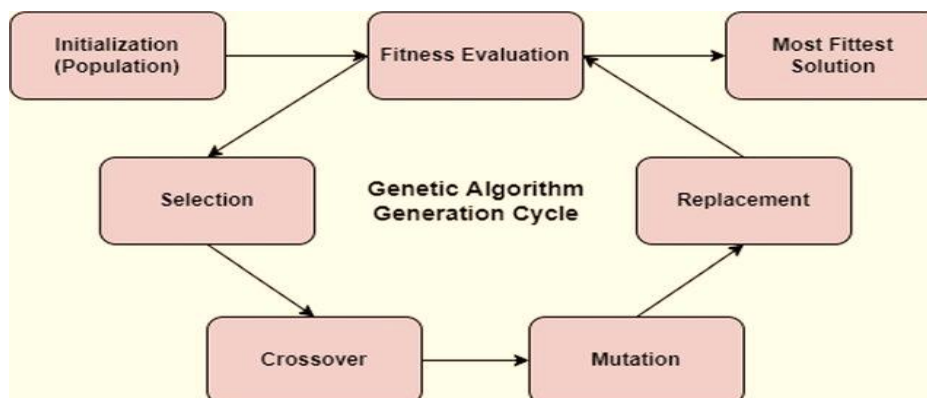


Fig. 3.2 Generation cycle of GA [5]

3.10 SUMMARY

In summary we discussed

- In brief how GAs model potential solutions as chromosomes, evolving them toward an optimal solution through genetic operations.
- Discussed crossover operator and its various forms, along with inheritance operators, inversion, deletion, and mutation operators.
- Discussed the generational cycle of GA.
- Discussed similarities and dissimilarities between GA and traditional algorithms.

3.11 TERMINAL QUESTIONS

1. Explain the working principle of genetic algorithm? What do you understand by fitness function?
2. What is reproduction? Give various method of selecting Chromosomes for parents to cross over?
3. Write short note on generational cycle of GA.
4. How is crossover operation performed? Give examples to illustrate various crossover techniques.
5. Mention the different types of mutation process.
6. Differentiate between Roulette wheel selection and tournament selection.
7. Differences between GAs and Traditional Methods.

3.12 BIBLIOGRAPHY

1. Melanie Mitchell, “An Introduction to Genetic Algorithms”, 1998.
2. Davis E.Goldberg, “Genetic Algorithms: Search, Optimization and Machine Learning”, Addison Wesley, N.Y., 1989.
3. Sivanandam, S. N., Deepa, S. N., Sivanandam, S. N., &Deepa, S. N. (2008). Genetic algorithms (pp. 15-37). Springer Berlin Heidelberg
4. S. Rajasekaran and G.A.VijayalakshmiPai.. Neural Networks Fuzzy Logic, and Genetic Algorithms, Prentice Hall of India, 2003
5. Suh, W.H., Oh, S. &Ahn, C.W. Metaheuristic-based time series clustering for anomaly detection in manufacturing industry. ApplIntell 53, 21723–21742 (2023). <https://doi.org/10.1007/s10489-023-04594-5>