REFERENCES

1. R. Pressman, Software engineering. New York: Mcgraw-Hill, 2014.
2. "What are the Characteristics of Software in Software Engineering?," InterviewBit, Feb. 28, 2022. https://www.interviewbit.com/blog/what-are-the-characteristics-of-software/
3. "Rapid Application Development Model | RAD Model - javatpoint," www.javatpoint.com. https://www.javatpoint.com/software-engineering-rapid-application-development-model
4. "SDLC - V-Model," www.tutorialspoint.com. https://www.tutorialspoint.com/sdlc/sdlc_v_model.htm#:~:text=V%2DModel%20%2D%20Design
5. Jignesh Kariya, "Unit 3," Apr. 18, 2018. https://www.slideshare.net/Jigneshkariya/unit-3-94164870
6. "Principles of Software Design," GeeksforGeeks, Jun. 13, 2020. https://www.geeksforgeeks.org/principles-of-software-design/
7. G. Pathak, "UNIT-6 SOFTWARE QUALITY ASSURANCE – ppt download," slideplayer.com. https://slideplayer.com/slide/15147825/
8. G. Booch, R. A. Maksimchuk, M. W. Engle, J. Conallen, B. Young, and K. Houston, Object-oriented Analysis and Design with Applications. Addison-Wesley Professional, 2007.
9. G. Booch, R. A. Maksimchuk, M. W. Engle, J. Conallen, B. Young, and K. Houston, Object-oriented Analysis and Design with Applications. Addison-Wesley Professional, 2007.
10. C. Larman, Applying UML and Patterns. Pearson, 2005.
11. "Software Engineering 1 Object-oriented Analysis and Design Applying UML and Patterns An Introduction to Object-oriented Analysis and Design and Iterative. - ppt download," slideplayer.com. https://slideplayer.com/slide/10853129/
12. "Object Oriented Analysis and Design Design Class Diagrams DESIGN CLASS DIAGRAMS."
13. Available: https://roshanchi.tripod.com/Documents/Study/OOAD/Notes/DCD.pdf
14. "Mapping Designs to Code. - ppt download," slideplayer.com.
15. https://slideplayer.com/slide/11389023

ARTIFICIAL INTELLIGENCE AND NEURAL NETWORK (ACtE09)

9.1 INTRODUCTION TO AI AND NEURAL NETWORKS

Artificial Intelligence:

AI refers to the simulation of intelligent behavior in machines. AI involves creating computer programs and algorithms that can perform tasks that usually require human intelligence, such as learning, problem-solving, pattern recognition, decision-making, and natural language processing.

Definitions of an AI:

Computers with the ability to mimic or duplicate the functions of the human brain.

Artificial Intelligence is the intelligence of machines and the branch of computer science which aims to create it.

"The branch of computer science that is concerned with the automation of intelligent behavior" (Luger and Stubblefield. 1993).

Intelligence:

Intelligence can be broadly defined as the ability to learn, understand, reason, and adapt to new situations. It involves the ability to acquire and apply knowledge, solve problems, make decisions, and learn from experience.

Concept of an AI:

AI, or Artificial Intelligence, is a field of computer science and engineering that focuses on the development of intelligent machines that can perform tasks that typically require human intelligence, such as perception, reasoning, learning, and decision-making.

The concept of AI is based on the idea of creating algorithms and systems that can mimic human cognitive functions, such as pattern recognition, language understanding, and problem-solving. AI systems can be designed to work in a variety of environments and can adapt to new situations through machine learning algorithms that enable them to learn from data and experience.

AI can be broadly divided into two categories: narrow or weak AI, which is designed to perform specific tasks such as image recognition, language translation, or speech recognition, and general or strong AI, which is the goal of creating machines that can perform any

intellectual task that a human can do. General AI is still a long-term goal and remains a topic of ongoing research.

AI has the potential to revolutionize many areas of human life, from healthcare and education to transportation and entertainment. However, it also raises ethical and social concerns, such as job displacement, bias, privacy, and the impact of autonomous systems on human decision-making. As AI continues to advance, it will be important to ensure that it is developed and used in a responsible and ethical manner.

Tasks of an AI:

- **Mundane Tasks:**
  - **Perception**
    - Vision
    - Speech
  - **Natural Language**
    - Generation
    - Understanding
    - Translation
  - Reasoning
  - Robot control
- **Formal Tasks**
  - **Playing games**
    - Chess
    - Tic-tac toe
    - Backgammon
    - Checkers
    - GO
  - **Mathematical Computation**
    - Integration
    - Logic
    - Differentiation
    - Geometry
  - **Theorem Proving**
- **Expert Tasks**
  - Engineering
    - Design
    - Manufacturing planning
    - Finding fault
    - Monitoring
  - Scientific Study
  - Financial Analysis
  - Medical Diagnostic

Important AI techniques:

- **Searching:** Searching is a fundamental AI technique that involves exploring a problem space to find a solution or optimal path to a goal. This technique can be used in a wide variety of applications, including game playing, route planning, and natural language processing. There are different types of search algorithms, including depth-first search, breadth-first search, heuristic search, and A* search, each with its own advantages and limitations.

- **Abstraction:** Abstraction is the process of simplifying complex systems or problems by focusing on the essential features or properties. Abstraction is a powerful technique in AI because it enables machines to reason about complex concepts and make decisions based on high-level concepts. For example, in a game of chess, a machine can use abstraction to represent the pieces and their moves in a way that allows it to search for the best move efficiently.
- **Use of Knowledge:** The use of knowledge involves incorporating human knowledge and expertise into an AI system. This can be done through expert systems, which use a set of rules or heuristics to make decisions based on a knowledge base, or through machine learning, where a system is trained on large amounts of data to learn patterns and rules. The use of knowledge is particularly useful in domains where there is a large amount of domain-specific knowledge, such as medicine, law, or finance. By incorporating human knowledge into an AI system, it can make better decisions, reduce errors, and improve performance.

AI Perspectives:

Views of AI fall into 4 different perspectives:

- **Acting humanly:** An AI system that acts humanly is designed to mimic human behavior, such as natural language understanding and generation, vision, speech recognition, and problem solving. This approach is sometimes called the Turing test approach, as it aims to create machines that can pass the Turing test, which is a test for a machine's ability to exhibit intelligent behavior equivalent to, or indistinguishable from, that of a human.
- **Thinking rationally:** An AI system that thinks rationally is designed to reason logically and follow formal rules of inference. This approach is sometimes called the logic-based approach, as it aims to create machines that can make logical deductions and reason about complex problems.
- **Thinking humanly:** An AI system that thinks humanly is designed to model the actual cognitive processes that humans use to solve problems, reason, and learn. This approach is sometimes called the cognitive modeling approach, as it aims to create machines that can mimic the cognitive processes of the human mind.
- **Acting rationally:** An AI system that acts rationally is designed to make optimal decisions based on its knowledge and goals. This approach is sometimes called the rational agent approach, as it aims to create machines that can act in the most effective and efficient way to achieve their objectives.

Goals of an AI:

1. Develop problem-solving ability,
2. Incorporate knowledge representation,
3. Facilitate planning,
4. Allow continuous learning,
5. Encourage social Intelligence,
6. Promote creativity,
7. Achieve general intelligence,
8. Promote synergy between humans and AI.

Applications of an AI:

1. AI in music,
2. AI in Medicine,

3. AI in Telecommunications,
4. Robotics,
5. AI in Gaming,
6. AI in Banking,
7. Credit Granting,
8. AI and Expert system embedded in products,
9. Help desk assistant,
10. Shipping,
11. Marketing,
12. Employee performance evaluation,
13. Satellite control,
14. Nuclear management,
15. Predicting stock market,
16. Machine translation,
17. Face detection,
18. Product recommendation,
19. Network development etc.

Challenges in AI:

1. Black box problem
2. AI Algorithm Bias
3. High computing power requirement
4. Complicated AI integration
5. Lack of understanding of implementation strategies
6. Legal concerns.

Process of AI categorization:

- **Sensing:** Sensing refers to the ability of an AI system to perceive and interpret its environment using sensors or other input devices. This can include input from cameras, microphones, or other sensors. The information obtained from the sensors is then processed by the AI system to derive insights about the environment. For example, a self-driving car uses sensors such as cameras, radar, and lidar to detect obstacles, traffic signals, and road markings.
- **Reasoning:** Reasoning refers to the ability of an AI system to process information and derive conclusions based on logical rules or probabilistic models. This involves using algorithms and models to analyze the information obtained from sensors or other input devices and make decisions or predictions based on that information. For example, an AI-powered medical diagnosis system uses reasoning to analyze patient data and generate a diagnosis.
- **Action:** Action refers to the ability of an AI system to act on its environment based on the conclusions drawn from its reasoning. This can involve physical actions, such as movement or manipulation of objects, or it can involve more abstract actions, such as generating a response to a user's request. For example, a robotic arm used in manufacturing can use the information obtained from sensors and the reasoning process to perform precise movements in order to assemble a product.

Myths about an AI:

AI is going to take over the world: This is a common misconception fueled by science fiction. The reality is that AI is a tool that can be used for both good and bad. It is ultimately humans who are responsible for how AI is used.

- **AI is infallible:** While AI can be very accurate and efficient, it is not perfect. AI systems can make mistakes or be biased, especially if they are not properly trained or designed.
- **AI is a magical solution:** AI is a powerful tool, but it is not a magical solution that can solve all problems. AI is most effective when used in conjunction with human expertise.
- **AI will replace human jobs:** While AI may automate some tasks, it is unlikely to completely replace humans in most jobs. In fact, AI can create new job opportunities and help humans to be more productive and efficient.
- **AI is only for tech companies:** AI can be used in many different industries, from healthcare to finance to transportation. Any business can benefit from using AI if it is properly designed and implemented.
- **AI is too complex to understand:** While the technical details of AI can be complex, the basic concepts can be understood by anyone with a basic understanding of technology. In fact, many AI tools are designed to be easy to use and require no coding or technical expertise.

Open-source AI software:

1. Mycroft.ai,
2. OpenCV,
3. OpenNN,
4. PyTorch,
5. RASA Open source,
6. TensorFlow,
7. Tesseract OCR

History of an AI:

- 1943: Warren McCulloch and Walter Pitts publish a paper on neural networks, which lay the groundwork for artificial neural networks.
- 1950: Alan Turing publishes a paper on the Turing Test, a benchmark for machine intelligence that tests a machine's ability to exhibit intelligent behavior equivalent to, or indistinguishable from, that of a human.
- 1956: John McCarthy, Marvin Minsky, Nathaniel Rochester, and Claude Shannon organize the Dartmouth Conference, which is widely considered the birth of AI as a field of study.
- 1957: Frank Rosenblatt introduces the Perceptron, a type of artificial neural network that can learn from data.
- 1965: Joseph Weizenbaum develops ELIZA, a computer program that simulates a conversation with a human and is considered one of the earliest examples of natural language processing.
- 1973: The Lighthill report is published, which criticizes the progress of AI research and leads to a period of reduced funding and interest in the field, known as the "AI winter."
- 1980s: Expert systems, which use knowledge representation and inference mechanisms to solve problems in specific domains, become popular and commercially successful.
- 1997: IBM's Deep Blue defeats world chess champion Garry Kasparov in a six-game match, marking a significant milestone in the development of AI for game-playing.

- 2011: IBM's Watson defeats human champions on the quiz show Jeopardy!, demonstrating the ability of AI to understand natural language and provide accurate answers to complex questions.
- 2012: The deep learning revolution begins, with the development of AlexNet, a deep convolutional neural network that significantly improves the state of the art in image classification.
- 2016: AlphaGo, a deep reinforcement learning system developed by Google DeepMind, defeats the world champion in the game of Go, demonstrating the ability of AI to learn complex strategies and make decisions in a highly complex environment.
- 2018: AI begins to be applied in a wide range of industries, from healthcare to finance to transportation, and becomes a key driver of digital transformation and innovation.

Foundation of AI:

The foundations of AI (artificial intelligence) are based on a range of disciplines, including computer science, mathematics, engineering, and cognitive psychology. Here are some of the key foundational concepts and ideas that underpin AI:

- **Logic and reasoning:** AI relies heavily on formal logic and reasoning to make decisions and solve problems. This involves the use of symbolic logic, such as propositional and predicate calculus, to represent knowledge and perform inference.
- **Probability and statistics:** Many AI algorithms and models are based on probabilistic reasoning, which involves reasoning under uncertainty. This includes techniques such as Bayesian networks, hidden Markov models, and Monte Carlo methods.
- **Machine learning:** AI relies on machine learning to enable systems to learn from data and improve over time. This involves a range of techniques, including supervised learning, unsupervised learning, and reinforcement learning.
- **Natural language processing:** AI involves the development of algorithms and models that enable computers to understand and generate human language. This involves a range of techniques, including syntax and semantic analysis, machine translation, and dialogue systems.
- **Robotics:** Robotics is a key area of AI that involves the development of intelligent systems that can sense and act upon the physical world. This involves a range of technologies, including sensors, actuators, and control systems.
- **Cognitive psychology:** AI is informed by research in cognitive psychology, which studies how humans think, learn, and reason. This involves the development of models of human cognition, such as decision-making and problem-solving.

Agent:

In AI, an agent is an entity that perceives its environment through sensors and acts upon that environment through actuators in order to achieve its goals. An agent can be a physical system, such as a robot, or a software program that runs on a computer.

The design of an agent is typically based on a problem-solving approach called the "agent paradigm." This approach involves identifying the objectives that the agent is trying to achieve, the actions that the agent can take, and the environmental factors that influence the agent's decisions.

Examples of agents in AI include autonomous vehicles, chatbots, personal assistants like Siri or Alexa, recommendation systems, and game-playing programs like AlphaGo.

Agent can be Human Agent and Robotic Agent.

- **Human agent:** eyes, ears, and other organs for sensors; hands, legs, mouth, and other body parts for actuators.
- **Robotic agent:** cameras and infrared range finders for sensors; various motors for actuators.

Example of Single Agent: Crossword Puzzle

Example of Multi- Agent: Playing Chess, Taxi Driving etc.

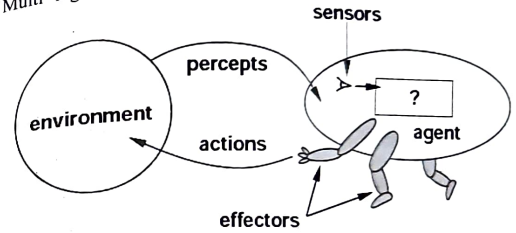


Figure: Agent and environments

Intelligent Agent's Structure:

Agent = Architecture + Agent Program

Architecture = the machinery that an agent executes on.

Agent Program = an implementation of an agent.

Properties of Intelligent Agent:

Intelligent agents in AI (artificial intelligence) possess a number of properties that enable them to operate effectively in a range of environments and applications. Some of the key properties of intelligent agents include:

- **Autonomy:** Intelligent agents are capable of operating independently, without human intervention or control. They are able to perceive their environment, reason about their goals, and take actions to achieve those goals.
- **Reactivity:** Intelligent agents are able to respond in real-time to changes in their environment, and to react to events and stimuli in order to achieve their goals.
- **Proactiveness:** Intelligent agents are able to take proactive measures to achieve their goals, rather than simply reacting to changes in their environment. They can generate plans and take actions to achieve long-term objectives.

- **Adaptivity:** Intelligent agents are able to learn and adapt to changes in their environment over time. They can modify their behavior and decision-making processes in response to new data or experiences.
- **Sociability:** Intelligent agents can interact and communicate with other agents, humans, and the environment. They can coordinate and cooperate with other agents in order to achieve common goals.
- **Veracity:** Intelligent agents are capable of representing and reasoning about uncertainty, incomplete information. They can reason probabilistically and make decisions under uncertainty.

Rational Agent:

In AI (artificial intelligence), a rational agent is an entity that acts to achieve the best possible outcome or outcome that is at least as good as any other available outcome. It is a framework for building intelligent agents that make decisions based on rationality, or the ability to maximize expected utility.

A rational agent can be designed to operate in a variety of environments, including deterministic, stochastic, fully observable, partially observable, single-agent, multi-agent, and adversarial environments. To act rationally, an agent must be able to perceive and reason about the current state of the environment, generate a set of possible actions, evaluate the expected outcomes of each action, and select the action that maximizes its expected utility.

Rational agents can be designed using a variety of AI techniques, including rule-based systems, decision trees, Bayesian networks, reinforcement learning, and evolutionary algorithms. They are used in a wide range of applications, including robotics, game playing, natural language processing, and financial forecasting, among others.

PEAS description of an Agent:

PEAS is an acronym that stands for Performance measure, Environment, Actuators, and Sensors. It is a framework used in artificial intelligence to describe the basic components of an intelligent agent. Here's how each component is defined:

- **Performance measure:** This component defines the criteria by which an agent's performance is measured. It is a way to determine how well an agent is performing in its environment. The performance measure can be objective (e.g., the number of chess games won) or subjective (e.g., customer satisfaction in a chatbot).
- **Environment:** The environment is the external context in which the agent operates. It is the part of the world that an agent can sense and interact with. The environment can be physical (e.g., a factory floor) or virtual (e.g., a computer simulation). It can also be deterministic or stochastic, observable or partially observable, and single-agent or multi-agent.
- **Actuators:** These are the physical or virtual devices that allow an agent to perform actions in the environment. Examples of actuators include motors, speakers, and output devices such as screens or printers.
- **Sensors:** Sensors allow an agent to perceive or sense the environment. Examples of sensors include cameras, microphones, temperature sensors, and other input devices.

P.E.A.S for Taxi Driver

- **Performance measure:** Safe, fast, legal, comfortable trip, maximize profits
- **Environment:** Roads, other traffic, pedestrians, customers
- **Actuators:** Steering wheel, accelerator, brake, signal, horn
- **Sensors:** Cameras, sonar, speedometer, GPS, odometer, engine sensors, keyboard

P.E.A.S for Medical Diagnosis System

- **Performance measure:** Healthy patient, minimize costs, lawsuits
- **Environment:** Patient, hospital, staff
- **Actuators:** Screen display (questions, tests, diagnoses, treatments, referrals)
- **Sensors:** Keyboard (entry of symptoms, findings, patient's answers)

P.E.A.S for Part-picking Robot

- **Performance measure:** Percentage of parts in correct bins.
- **Environment:** Conveyor belt with parts, bins Actuators: Jointed arm and hand
- **Sensors:** Camera, joint angle sensors.

Table: Examples of PEAS descriptions

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, reduced costs	Patient, hospital staff	Display of questions, tests, diagnoses, treatments, referrals	Keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system	Correct image categorization	Downlink from orbiting satellite	Display of scene categorization	Color pixel arrays
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, joint angle sensors
Refinery controller	Purity, yield, safety	Refinery, operators	Valves, pumps, heaters, displays	Temperature, pressure, chemical sensors
Interactive English tutor	Student's score on test	Set of students, testing agency	Display of exercises, suggestions, corrections	Keyboard entry

Types of Agents:

- **Table-driven agents:**
  - These agents rely on a fixed set of rules to determine their actions based on the current state of the environment.
  - perfect solution but intractable
  - Single perception look-up (HUGE table), perception sequence look-up (HUGE table).
  - example game: tic-tac-toe

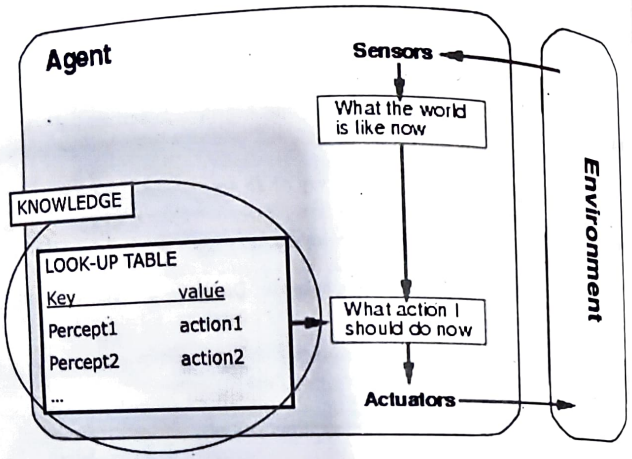


Figure: Table-driven agents

- **Simple reflex agents:** These agents select actions based solely on the current percept, or the current state of the environment, without considering the history of previous percepts or actions.
  - Based on current perception only. i.e., no instance variables in the agent object: no state
  - 'Condition-action' rules (if then else algorithm)
  - NO MEMORY, fails if environment is partially observable. E.g.: vacuum cleaner world.

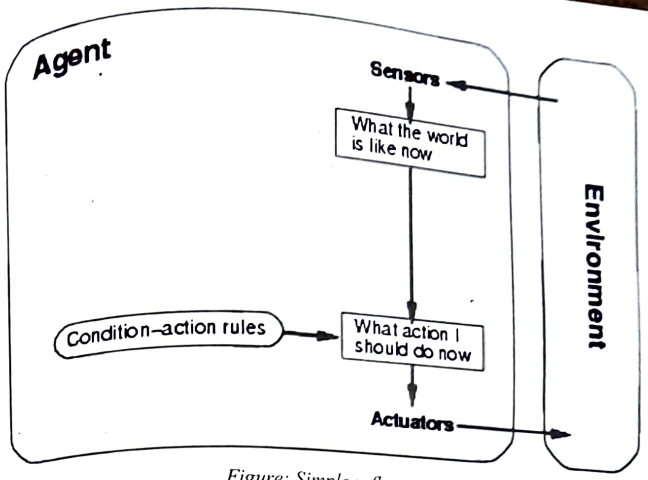


Figure: Simple reflex agents

- **Model-based reflex agents:** These agents build an internal model of the environment and use it to choose actions based on the current percept, past percepts, and the agent's internal state.

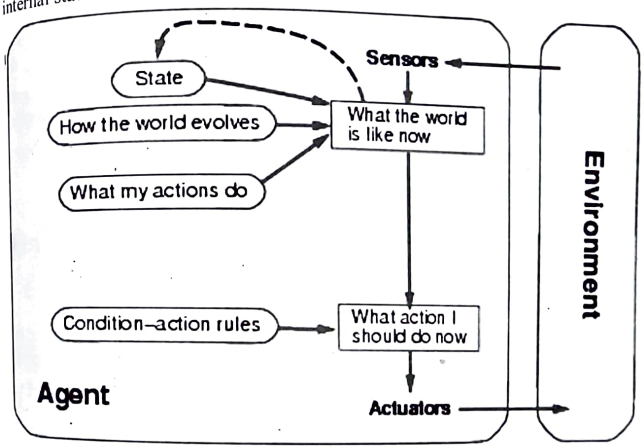


Figure: Model-based reflex agents

- **Goal based reflex agents:** These agents use a goal formulation process to determine their actions based on a desired end state or goal. They work to achieve the goal by selecting actions that move them closer to that goal.

- NOT REFLEX; 'tries' actions internally and tests results against goals.

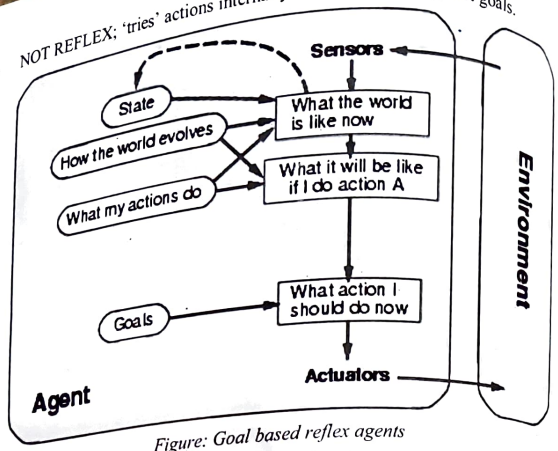


Figure: Goal based reflex agents

- **Utility based agents:** These agents use a utility function to determine the desirability of different states or outcomes, and select actions that maximize their expected utility.
  - Internal state representing environment as well as goals expressed in terms of environment and/or agent states with performance measure rationality.
  - 'tries' actions internally and tests results against goals AND performance measure.

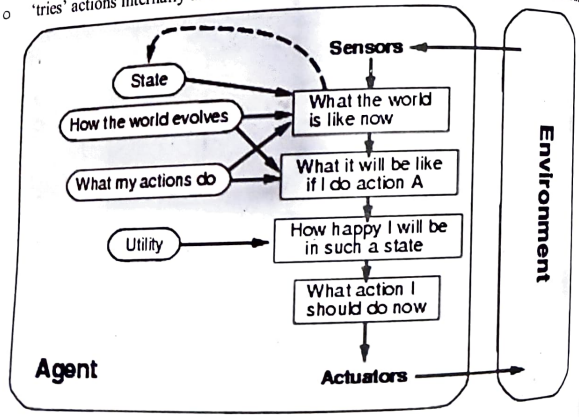


Figure: Utility based agents

- **Learning agents:** Extra component to evaluate performance and change program (if necessary) to act differently in same state.
  - E.g., Taxi skids on ice.

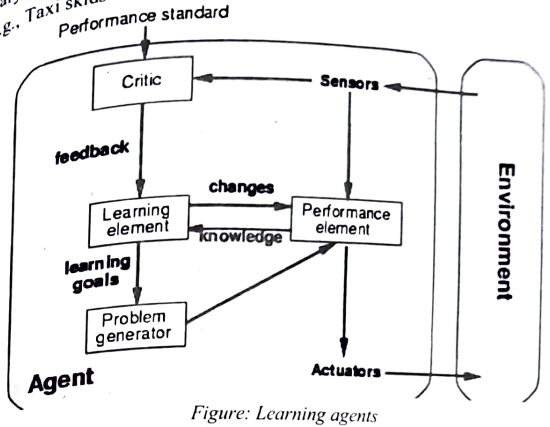


Figure: Learning agents

Agent Environment types with examples:

In artificial intelligence, agents can operate in different types of environments, each with its own characteristics and challenges. Here are some of the most common types of environments:

- **Deterministic:** In a deterministic environment, the next state of the environment is completely determined by the current state and the actions taken by the agent. There is no uncertainty involved. A chess game is an example of a deterministic environment. The rules of the game are fixed, and the outcome of any move can be predicted with certainty.
- **Stochastic:** In a stochastic environment, the next state of the environment is probabilistic, and depends on a combination of the current state and the actions taken by the agent. There is uncertainty involved. A weather forecasting system is an example of a stochastic environment. The behavior of the environment is affected by a large number of random variables that are difficult to predict with certainty.
- **Static:** In a static environment, the environment does not change while the agent is deliberating. The environment is not affected by the agent's actions. A crossword puzzle is an example of a static environment. The puzzle remains unchanged while the agent tries to solve it.
- **Dynamic:** In a dynamic environment, the environment changes while the agent is deliberating. The environment may be affected by the agent's actions or by other external factors. A traffic management system is an example of a dynamic environment. The traffic conditions change constantly, and the system needs to adjust to these changes in real-time.

- **Observable:** In an observable environment, the agent can directly or indirectly observe the complete state of the environment at each time step. A game of Tic Tac Toe is an example of an observable environment. The state of the game board is fully visible to both players at all times.
- **Partially observable:** In a partially observable environment, the agent cannot directly observe the complete state of the environment at each time step. It may have limited or noisy information about the state of the environment. A poker game is an example of a partially observable environment. The players have incomplete information about the cards held by their opponents, and must make decisions based on probability and strategy.
- **Single-agent environment:** In a single-agent environment, there is only one agent operating in the environment. The agent interacts with the environment to achieve its goals, but there are no other agents present that it needs to coordinate with or compete against. Single-agent environments are often simpler and easier to model than multi-agent environments. A robot navigating a maze is an example of a single-agent environment. The robot is the only agent in the environment, and its goal is to find the exit to the maze.
- **Multi-agent environment:** In a multi-agent environment, there are multiple agents operating in the environment. Each agent has its own goals and may need to coordinate or compete with other agents to achieve those goals. Multi-agent environments can be more complex and challenging to model than single-agent environments, as agents need to take into account the actions and goals of other agents. A game of soccer is an example of a multi-agent environment. There are two teams of players, each with their own individual goals, but also needing to coordinate and compete with each other to score goals and win the game.

9.2 PROBLEM SOLVING AND SEARCHING TECHNIQUES

Problem Solving:

In artificial intelligence, a problem is a situation where an agent needs to reach a goal, but the way to achieve that goal is not immediately apparent. Problem solving is the process of finding a sequence of actions that will lead to a desired goal state.

Problem solving is an essential task in AI, as it underlies many of the applications and systems that are developed in the field. The process of problem solving typically involves the following steps:

- **Formulating the problem:** This involves defining the problem in a way that can be solved by an agent. This step includes identifying the initial state of the problem, the goal state, and the actions that the agent can take to move from the initial state to the goal state.
- **Searching for a solution:** The agent searches through the space of possible actions and states to find a sequence of actions that will lead to the goal state. This search can be guided by heuristics or algorithms that help the agent find the most promising paths through the search space.

- ***Executing the solution:*** Once a solution is found, the agent executes the sequence of actions to achieve the goal.

problem space:

In artificial intelligence, the problem space is the set of all possible states, actions, and transitions that can occur when trying to solve a problem. It represents the universe of possible scenarios that the agent must consider when searching for a solution.

The problem space is typically defined by the initial state of the problem, the set of available actions, and the rules that govern how these actions can be applied to transition between states. The problem space may also include constraints, goals, and other factors that must be considered in order to find a solution. The size and complexity of the problem space can vary widely depending on the specific problem and the environment in which the agent is operating. In some cases, the problem space may be relatively small and easy to navigate, while in other cases it may be very large and require more sophisticated search or optimization techniques to find a solution.

Search and search strategies:

Search in problem space is a key technique used in artificial intelligence to find a sequence of actions that will lead to a goal state. The process of search involves exploring the problem space to identify a path from the initial state to the goal state that satisfies certain criteria, such as being the shortest, the cheapest, or the most efficient.

Search in problem space can be characterized by the following components:

- **The initial state:** The state of the problem at the start of the search.
- **The goal state:** The state of the problem that the agent is trying to reach.
- **The actions:** The set of available actions that the agent can take to move from one state to another.
- **The transition model:** The set of rules that govern how the actions can be applied to transition between states.
- **The search strategy:** The algorithm used to traverse the problem space and find a solution.

The search strategy can take many different forms, depending on the nature of the problem and the goals of the agent. Some common search strategies include:

- **Breadth-first search:** Explores all the nodes at the current depth before moving on to the next level of nodes.
- **Depth-first search:** Explores as far as possible along each branch before backtracking.
- **Uniform-cost search:** Expands the node with the lowest path cost.
- **A* search:** Evaluates nodes based on a combination of path cost and a heuristic estimate of the remaining cost to the goal.

Properties of searching techniques:

- **Completeness:** A searching technique is said to be complete if it is guaranteed to find a solution if one exists.

- **Time complexity:** It refers to the amount of time taken by a searching technique to find a solution.
- **Space complexity:** It refers to the amount of memory or space used by a searching technique to find a solution.
- **Optimality:** A searching technique is said to be optimal if it can find the best or optimal solution, i.e., the solution with the lowest cost among all possible solutions.

State and state space:

State: In artificial intelligence, a state is a configuration of the world or problem environment at a given point in time. A state can be thought of as a snapshot of the environment, describing the values of relevant variables, properties, and relationships at a particular moment. For example, in a game of chess, a state might represent the current positions of all the pieces on the board.

State Space: A state space, also known as a search space or problem space, is the collection of all possible states that can be reached from the initial state of a problem environment by applying a series of actions. The state space defines the universe of possible scenarios that the agent must consider when searching for a solution. The size and complexity of the state space can vary widely depending on the nature of the problem and the environment in which the agent is operating.

Structure of a state space:

The structure of a state space is determined by the problem environment and the set of actions that can be performed in that environment. In general, a state space can be represented as a graph or a tree, where each node in the graph or tree represents a state and each edge represents an action that can be taken to transition from one state to another. The structure of the state space can be visualized using a diagram, where the nodes and edges are shown as circles and arrows, respectively.

Problem solving agent:

A problem-solving agent is an intelligent agent that is designed to find solutions to problems in a given environment. The agent is equipped with a set of problem-solving strategies and algorithms that it can use to explore the environment and find a sequence of actions that lead to a desirable outcome.

A problem-solving agent typically operates in the following way:

- It observes the current state of the environment and identifies the problem that needs to be solved.
- It formulates the problem in a way that can be understood by the agent's problem-solving algorithms. This may involve representing the problem as a set of states and actions, or as a set of constraints and goals.
- It selects a problem-solving strategy or algorithm that is appropriate for the problem at hand. This may involve using a simple search algorithm, such as depth-first search or breadth-first search, or a more sophisticated algorithm, such as A* search or Monte Carlo tree search.

- It applies the selected algorithm to search the state space of the problem and find a sequence of actions that lead to a desirable outcome. The agent may use heuristics, domain knowledge, or other forms of intelligence to guide the search and improve its efficiency.
- Once a solution is found, the agent executes the sequence of actions and monitors the environment to ensure that the solution is valid and satisfactory.

Problem type:

In AI, problems can be classified into different categories based on their characteristics and the nature of the environment in which they arise. Some common types of problems in AI include:

- **Single-state problem:** A problem in which the goal state is known and the agent must find a sequence of actions that transforms the current state to the goal state. Examples of single-state problems include the 8-puzzle and the Tower of Hanoi.
- **Multi-stage problem:** A problem in which the agent must make a sequence of decisions in a dynamic environment to achieve a long-term goal. Examples of multi-stage problems include route planning, game-playing, and decision-making under uncertainty.
- **Contingency problem:** A problem in which the agent must deal with uncertainty in the environment and take actions that minimize the risk of negative outcomes. Examples of contingency problems include fault diagnosis and resource allocation.
- **Exploration problem:** A problem in which the agent must explore the environment to gather information that is necessary for making decisions. Examples of exploration problems include mapping, object recognition, and search and rescue.

Well-defined problem and its components:

A well-defined problem is a problem where the initial state, goal state, and the set of available actions are clearly defined. Well-defined problems have a clear solution, and the solution can be obtained by following a set of steps or an algorithm.

The components of a well-defined problem are:

- **Initial state:** The starting point of the problem, which represents the current situation.
- **Goal state:** The desired outcome of the problem, which represents the end goal.
- **Actions:** The set of available actions that can be taken by the agent to move from one state to another.
- **Transition model:** A description of the results of each action, which defines the new state that the agent will be in after taking a particular action.
- **Path cost:** The cost associated with each action or sequence of actions that the agent takes to reach the goal state.
- **Solution:** The set of actions that the agent must take to move from the initial state to the goal state.

In a well-defined problem, the solution is typically obtained by searching the state space, which involves exploring the different possible states and the actions that can be taken to move

from one state to another. Different search algorithms can be used to find the optimal or near-optimal solution to a well-defined problem.

Constraint satisfaction problem:

A constraint satisfaction problem (CSP) is a type of problem in artificial intelligence where a solution must be found that satisfies a set of constraints. It is a well-defined problem consisting of a set of variables, a domain for each variable, and a set of constraints that specify the allowed combinations of values for the variables.

The variables in a CSP represent the objects or variables that the agent must determine values for in order to satisfy the constraints. The domain of each variable is the set of possible values that can be assigned to it. The constraints specify the relationships between the variables and their allowed combinations of values.

The goal of a CSP is to find a complete and consistent assignment of values to all variables such that all constraints are satisfied. A solution to a CSP is a complete assignment of values to all variables that satisfies all constraints.

CSPs are useful in many different areas of artificial intelligence, including scheduling, planning, and configuration. There are various algorithms and techniques that can be used to solve CSPs, such as backtracking, constraint propagation, and local search. These algorithms can efficiently find solutions to large-scale CSPs with many variables and constraints.

Uninformed and informed search techniques:

Uninformed search techniques and informed search techniques are two broad categories of search algorithms used in artificial intelligence for solving problems in a search space.

Uninformed search techniques: Uninformed search techniques are also known as blind search algorithms. These algorithms do not use any additional knowledge about the problem beyond the problem definition, such as the cost of actions or the distance to the goal state. Uninformed search techniques explore the search space blindly, without using any heuristics to guide the search. Examples of uninformed search techniques include:

1. Breadth-first search
2. Depth-first search
3. Iterative deepening search
4. Uniform-cost search
5. Depth-limited search
6. Bidirectional search

Uninformed search techniques, also known as blind search algorithms, are search algorithms that do not use any additional information about the problem beyond the problem definition. Here are some examples of uninformed search techniques:

- **Depth-First Search (DFS):** DFS is a search algorithm that starts at the root node and explores as far as possible along each branch before backtracking. It is implemented using a stack data structure. DFS is often used to explore all the solutions in a problem space.
  - **Completeness:** DFS is not complete. If the search space is infinite or the goal state is very deep, DFS may get stuck in an infinite loop and never find the goal state.
  - **Time complexity:** The time complexity of DFS is $O(b^m)$, where b is the branching factor and m is the maximum depth of the search tree. This is because DFS may explore all paths of the search tree, including those that are not optimal, before finding the goal state.
  - **Space complexity:** The space complexity of DFS is O(bm), where b is the branching factor and m is the maximum depth of the search tree. This is because DFS only needs to store the path from the root to the current node, so the maximum number of nodes in memory is the maximum depth of the search tree.
  - **Optimality:** DFS is not guaranteed to find the optimal solution in a weighted search space. This is because DFS may explore a non-optimal path before finding the goal state. However, if the search space is not weighted or all paths have the same cost, then DFS will find the optimal solution.
- **Breadth-First Search (BFS):** BFS is a search algorithm that explores all the nodes at a given depth before moving on to the next depth level. It is implemented using a queue data structure. BFS is often used to find the shortest path to a solution.

  Properties:
  - **Completeness:** BFS is complete, meaning it will find a solution if one exists in a finite search space. This is because it exhaustively searches all nodes at a given depth level before moving on to the next level, ensuring that all possible solutions are considered.
  - **Time complexity:** The time complexity of BFS is $O(b^d)$, where b is the branching factor and d is the depth of the goal state. This is because BFS needs to explore all nodes at each depth level before moving on to the next level. In the worst case, BFS will explore the entire search space.
  - **Space complexity:** The space complexity of BFS is $O(b^d)$, where b is the branching factor and d is the depth of the goal state. This is because BFS needs to store all nodes at each depth level in memory.
  - **Optimality:** BFS is guaranteed to find the optimal solution in a weighted search space if the path cost is non-decreasing. This is because BFS explores all nodes at a given depth level before moving on to the next level, ensuring that the optimal solution is found before less optimal solutions are considered.
- **Depth-Limited Search (DLS):** DLS is a search algorithm that limits the depth of exploration in the search space. It is a modified version of DFS that sets a maximum depth for the search. DLS is often used when the search space is too large for BFS, but the goal state is not too deep.

  Properties:
  - **Completeness:** DLS is complete if the depth limit is greater than or equal to the depth of the shallowest goal state. Otherwise, it may not find the goal state.
  - **Time complexity:** The time complexity of DLS is $O(b^l)$, where b is the branching factor and l is the depth limit. This is because DLS may explore all paths up to the depth limit before finding the goal state.
  - **Space complexity:** The space complexity of DLS is O(bl), where b is the branching factor and l is the depth limit. This is because DLS only needs to store the path from the root to the current node, so the maximum number of nodes in memory is the depth limit.

- **Optimality:** DLS is not guaranteed to find the optimal solution in a weighted search space. This is because DLS may explore a non-optimal path before finding the goal state. However, if the depth limit is set to infinity, then DLS is equivalent to DFS and may find the optimal solution if one exists.

- **Iterative Deepening Search (IDS):** IDS is a combination of DLS and BFS. It repeatedly applies DLS with increasing depth limits until a solution is found. IDS is often used when the depth of the solution is unknown or when memory is limited.

Properties:

- **Completeness:** IDS is complete if the branching factor is finite and the depth limit is infinite. This is because it will eventually explore the entire search space, including the goal state.
- **Time complexity:** The time complexity of IDS is $O(b^d)$, where b is the branching factor and d is the depth of the shallowest goal state. This is because IDS explores each level of the search tree once, and the maximum depth of the search tree is d.
- **Space complexity:** The space complexity of IDS is O(bd), where b is the branching factor and d is the depth of the shallowest goal state. This is because IDS only needs to store the path from the root to the current node, and the maximum number of nodes in memory is the depth of the shallowest goal state.
- **Optimality:** IDS is optimal if the cost of the solution is a non-decreasing function of the depth limit. This means that the first solution found by IDS is guaranteed to be optimal, as each subsequent depth limit will only expand nodes that were not expanded in previous depth limits.

- **Bidirectional Search:** Bidirectional search is a search algorithm that simultaneously searches from the initial state and the goal state. It terminates when the two searches meet in the middle. Bidirectional search is often used when the search space is too large for unidirectional search or when the goal state is known.

Properties:

- **Completeness:** Bidirectional Search is complete if both the forward and backward searches are complete. This means that both searches can reach all states in the state space.
- **Time complexity:** The time complexity of Bidirectional Search is $O(b^{(d/2)})$, where b is the branching factor and d is the depth of the shallowest goal state. This is because the two searches meet in the middle of the search space, and each search has a depth of d/2.
- **Space complexity:** The space complexity of Bidirectional Search is $O(b^{(d/2)})$, where b is the branching factor and d is the depth of the shallowest goal state. This is because both searches only need to store the path from the start state or the goal state to the current node, and the maximum number of nodes in memory is the depth of the shallowest goal state.

- **Optimality:** Bidirectional Search is optimal if the cost of the solution is a non-decreasing function of the depth of the two searches. This means that the first solution found by Bidirectional Search is guaranteed to be optimal, as it is the shortest path that connects the start state and the goal state.

Informed search techniques: Informed search techniques use additional information about the problem to guide the search. This additional information is called a heuristic function and provides an estimate of the distance to the goal state. Informed search algorithms use this estimate to guide the search in a more efficient way. Examples of informed search techniques include:

1. Best-first search
2. A* search
3. Hill climbing
4. Simulated annealing

Informed search techniques are generally more efficient than uninformed search techniques because they use additional information to guide the search towards the goal state. However, the quality of the solution obtained by an informed search technique depends on the quality of the heuristic function used.

- **Greedy Best First Search:** This algorithm expands the node that appears to be closest to the goal according to the heuristic function. It does not consider the cost of the path to reach that node. Greedy Best First Search is not optimal, as it may get stuck in local optima.

Properties:

- **Completeness:** Greedy Best First Search is not guaranteed to find a solution even if one exists, and it can get stuck in loops.
- **Time complexity:** The time complexity of Greedy Best First Search depends on the quality of the heuristic function. In the worst case, it can take exponential time.
  - $O(b^m)$ but a good heuristic can give dramatic improvement.
- **Space complexity:** The space complexity of Greedy Best First Search can be quite large because it needs to store all the nodes that have been expanded but not yet explored.
  - $O(b^m)$ but keeps all nodes in memory
- **Optimality:** Greedy Best First Search is not guaranteed to find an optimal solution, as it may get stuck in a local minimum or maximum depending on the heuristic function.

- **A* Search:** This algorithm uses a heuristic function that combines the cost of the path to reach a node and the estimated cost of the remaining path to the goal. The algorithm expands the node that has the lowest cost function value. A* Search is complete, optimal, and admissible, meaning that it always finds the shortest path to the goal if one exists.

Properties:

- **Completeness:** A* search is complete if the heuristic function is admissible, meaning that it never overestimates the actual cost of reaching the goal state from any node.

- **Time complexity:** The time complexity of A* search depends on the quality of the heuristic function. In the worst case, it can take exponential time, but it is usually faster than uninformed search algorithms.
- **Space complexity:** The space complexity of A* search can be quite large because it needs to store all the nodes that have been expanded but not yet explored.
- **Optimality:** A* search is optimal if the heuristic function is admissible and consistent, meaning that the estimated cost of reaching the goal from any node is never less than the actual cost of reaching the goal from that node, and the estimated cost of reaching any successor of a node is never greater than the actual cost of reaching that successor plus the estimated cost of reaching the goal from that successor.

- **Hill Climbing:** This algorithm starts at an arbitrary state and repeatedly moves to a neighboring state with a higher heuristic value. It stops when it reaches a state with no better neighbors. Hill Climbing is not complete or optimal, as it may get stuck in local optima.

Properties:

- **Completeness:** Hill climbing search is not complete, meaning it may not find a solution even if one exists.
- **Time complexity:** The time complexity of hill climbing is typically much better than uninformed search algorithms like depth-first search, but it still may not find the optimal solution.
- **Space complexity:** The space complexity is relatively low, as it only needs to keep track of the current state and the best neighbor found so far.
- **Optimality:** Hill climbing search is not optimal, as it may get stuck at a local maximum or minimum instead of finding the global maximum or minimum.

- **Simulated Annealing:** This algorithm is similar to Hill Climbing, but it allows some moves that decrease the heuristic value. The probability of accepting a worse move is determined by a temperature parameter that decreases over time. Simulated Annealing is not complete or optimal, but it is more likely to find the global optimum than Hill Climbing.

Properties:

- **Completeness:** Simulated annealing search is complete, meaning it will eventually find a solution if one exists.
- **Time complexity:** The time complexity of simulated annealing can be high, especially if the annealing schedule is not well-tuned. However, it can still be more efficient than brute-force search techniques in many cases.
- **Space complexity:** The space complexity is relatively low, as it only needs to keep track of the current state and the best neighbor found so far.
- **Optimality:** Simulated annealing search is not guaranteed to find the optimal solution, but it can be tuned to increase the probability of finding it.

Game playing:

Game playing is the application of artificial intelligence (AI) to develop computer programs capable of playing games. Game-playing AI agents can be designed to play different kinds of games, including traditional board games like chess and checkers, video games, and even sports.

The development of game-playing AI agents is challenging because it requires the agent to make decisions based on incomplete information and to deal with uncertainty. Game-playing agents must be able to search through a large number of possible moves and evaluate the likely outcomes of those moves to make the best decision.

One approach to developing game-playing AI agents is to use a search algorithm to explore the game tree, which represents all possible moves in the game. The search algorithm evaluates the expected outcome of each possible move and selects the move that maximizes the agent's chances of winning. Other techniques, such as machine learning, can also be used to develop game-playing agents.

Game playing has many real-world applications, such as in the design of autonomous systems and in the development of decision-making algorithms for business and finance. Additionally, game playing is a popular area of research in AI, and the development of game-playing agents has led to significant advances in the field.

Adversarial search techniques:

Adversarial search techniques are used in game playing AI, where an agent needs to take actions in an environment where there are other agents with conflicting objectives. In this case, the environment is referred to as an adversarial environment.

The aim of an adversarial search algorithm is to make the best possible decision given the state of the environment, the objectives of the other agents, and the possible actions that can be taken by the agent.

Some of the most common adversarial search techniques include:

- **Minimax algorithm:** A decision-making algorithm used in two-player games to minimize the potential loss. It works by selecting moves that minimize the maximum loss.
- **Alpha-Beta pruning:** A technique used to reduce the number of nodes that are evaluated by the minimax algorithm, making the search more efficient.
- **Monte Carlo tree search (MCTS):** A heuristic search algorithm that is based on random sampling of the search space. It is commonly used in games that have a very large search space and a high branching factor.
- **Heuristic evaluation functions:** A function used to estimate the value of a state or a move in a game. It is used to speed up the search process by pruning nodes that are unlikely to lead to a good outcome.

Minimax search/ Algorithm:

The minimax algorithm is a decision-making algorithm that is used in two-player turn-based games, such as chess, checkers, and tic-tac-toe. The algorithm is designed to help a player determine the best move to make based on the current state of the game and the possible moves that their opponent could make in response.

The minimax algorithm works by recursively exploring the game tree, which represents all possible moves and counter-moves that could be made by both players. At each level of the tree, the algorithm alternates between maximizing the player's score and minimizing the opponent's score, assuming that the opponent will always make the best move for themselves. The algorithm then chooses the move that leads to the highest possible score for the player, assuming that the opponent will make the best possible counter-move.

The minimax algorithm is guaranteed to find the optimal move in a game tree, assuming that the tree is finite and both players play optimally. However, the algorithm can be computationally expensive for larger game trees, and more efficient algorithms, such as alpha-beta pruning, are often used in practice.

The properties of the minimax algorithm are as follows:

- **Complete:** The minimax algorithm can find a solution if one exists.
- **Optimal:** The minimax algorithm will always find the optimal solution for both players.
- **Time complexity:** The time complexity of the minimax algorithm is $O(b^m)$, where b is the branching factor of the game tree and m is the maximum depth of the tree.
- **Space complexity:** The space complexity of the minimax algorithm is O(bm), which is the maximum number of nodes that can be stored in memory at any point in time.
- **Deterministic:** The minimax algorithm assumes that the opponent will always make the best move, so it does not consider random or unpredictable moves.
- **Not suitable for all games:** The minimax algorithm is not suitable for all games, especially those with a large state space or complex rules.

Alpha beta pruning:

Alpha-beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree. It works by eliminating parts of the search tree that are guaranteed not to influence the final result.

The alpha-beta pruning algorithm maintains two values, alpha and beta, which represent the best choices found so far for the maximizer and minimizer, respectively. As the search progresses, alpha and beta are updated with the best values found so far.

When a branch of the search tree is being explored, the algorithm prunes the search if it determines that the branch can no longer improve the final result. This is done by comparing the alpha and beta values of the nodes in the branch. If the beta value is less than or equal to the alpha value, then the branch can be pruned, since the opposing player would never choose to go down that branch.

Alpha-beta pruning is an optimization of the minimax algorithm, and it can significantly reduce the number of nodes that need to be evaluated. The efficiency of the alpha-beta pruning algorithm depends on the ordering of the nodes in the search tree, and it is most effective when the nodes are ordered in a way that leads to early cutoffs.

The properties of Alpha-Beta pruning are as follows:

- **Completeness:** Alpha-beta pruning is a complete algorithm, meaning it will always produce a result.
- **Optimality:** Alpha-beta pruning is an optimal algorithm, meaning it will always produce the best result.
- **Time complexity:** The time complexity of the algorithm is $O(b^{m/2})$, where b is the branching factor, and m is the maximum depth of the tree.
- **Space complexity:** The space complexity of the algorithm is O(m), where m is the maximum depth of the tree.
- **Pruning:** Alpha-beta pruning effectively prunes parts of the search tree that cannot influence the final decision, which leads to significant time savings.
- **Adversarial:** Alpha-beta pruning is designed for adversarial games where the opponent is trying to prevent the algorithm from finding the best solution.
- **Heuristic evaluation function:** Alpha-beta pruning requires a heuristic evaluation function to evaluate the state of the game tree nodes. The quality of the heuristic function greatly affects the performance of the algorithm.

9.3 KNOWLEDGE REPRESENTATION

Knowledge representations and Mappings:

To tackle the complex challenges posed in artificial intelligence, a substantial amount of knowledge and tools for utilizing that knowledge to devise solutions are required. Various methods of representing facts have been employed in AI programs. However, before delving into each approach, it's crucial to acknowledge the distinction between two distinct entities:

- Facts: truths in some relevant world. These are the things we want to represent.
- Representations of facts in some chosen formalism. These are the things we will actually be able to manipulate.

One way to think of structuring these entities is as two levels:

- The knowledge level, at which facts (including each agent's behaviors and current goals) are described.
- The symbol level, at which representations of objects at the knowledge level are defined in terms of symbols that can be manipulated by programs.

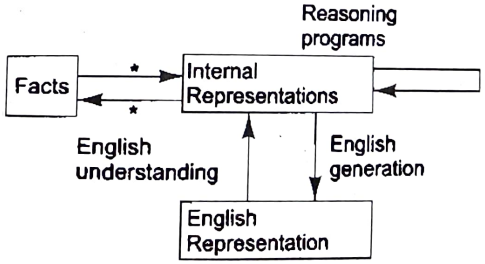


Figure: Mappings between facts/ knowledge and representations

Let's look at a simple example using mathematical logic as the representational formalism. Consider the English sentence:

Spot is a cat.

The fact represented by that English sentence can also be represented in logic as:

cat(Spot)

Suppose that we also have a logical representation of the fact that all cats have tails:

$\forall x: cat(x) \rightarrow hastail(x)$

Then, using the deductive mechanisms of logic, we may generate the new representation object:

hastail(Spot)

Using an appropriate backward mapping function, we could then generate the English sentence:

Spot has a tail.

Approaches to Knowledge Representation:

In the field of artificial intelligence, knowledge representation is a key aspect of building intelligent agents that can reason, learn, and make decisions. A good system which wants to represent a knowledge must possess following properties. There are several properties that are desirable for a knowledge representation system to have, including:

- **Representational adequacy:** This refers to the ability of a knowledge representation system to accurately represent the facts and concepts of a particular domain. A good knowledge representation system should be able to represent all of the relevant aspects of the domain in a clear and concise way.
- **Inferential adequacy:** This refers to the ability of a knowledge representation system to support reasoning and inference. A good knowledge representation system should be able to support a wide range of reasoning and inference tasks, including deduction, abduction, and induction.
- **Inferential efficiency:** This refers to the ability of a knowledge representation system to perform reasoning and inference tasks quickly and efficiently. A good knowledge representation system should be able to perform these tasks in a timely manner, without getting bogged down in unnecessary computations.
- **Acquisitional efficiency:** This refers to the ability of a knowledge representation system to learn from new information and incorporate it into its existing knowledge base. A good knowledge representation system should be able to learn quickly and efficiently, without requiring large amounts of data or manual intervention.

Simple Rational Knowledge: This is the knowledge that can be represented as a set of rules or facts that are true in a particular situation. It can be easily represented and is usually domain-specific.

Inheritable Knowledge: This is the knowledge that can be passed down from one agent to another. It is usually represented in a form that is easy to share and understand.

Inferential Knowledge: This is the knowledge that can be used to make inferences or draw conclusions about a particular situation. It is usually represented in a form that allows for reasoning and inference.

Procedural Knowledge: This is the knowledge that relates to the procedures or steps that need to be taken in order to solve a problem or complete a task. It is usually represented in a form that describes the steps that need to be taken in order to achieve a particular goal.

Issues in Knowledge Representation:

- Are any attributes of objects so basic that they occur in almost every problem domain? If there are, we need to make sure that they are handled appropriately in each of the mechanisms we propose. If such attributes exist, what are they?
- Are there any important relationships that exist among attributes of objects?
- At what level should knowledge be represented? Is there a good set of primitives into which all knowledge can be broken down? Is it helpful to use such primitives?
- How should sets of objects be represented?
- Given a large amount of knowledge stored in a database, how can relevant parts be accessed when they are needed?

Other knowledge representation issues:

- **Representation bias:** The choice of representation language or format can influence the kinds of inferences that can be made and the conclusions that can be drawn. This can lead to representation bias, where some aspects of the problem domain are overemphasized or underrepresented.
- **Scalability:** As the amount of knowledge increases, the size and complexity of the knowledge representation system can grow rapidly, making it difficult to maintain and use efficiently.
- **Uncertainty:** Real-world domains are often characterized by uncertainty, such as incomplete or ambiguous information. Representing and reasoning with uncertain information is challenging and requires specialized techniques.
- **Context dependence:** The interpretation of knowledge often depends on context, such as the time, location, or other circumstances in which the knowledge is being used. Representing and reasoning with context-dependent knowledge can be difficult.
- **Common sense reasoning:** Many real-world problems require common sense reasoning, which involves making inferences based on everyday experience and knowledge. However, common sense is often difficult to formalize and represent.
- **Knowledge acquisition:** Acquiring and integrating knowledge from various sources can be a time-consuming and challenging process. The knowledge representation system must also be able to adapt and evolve as new knowledge is acquired.

Knowledge Representation Techniques:

- Logical Representation
- Frame Representation
- Semantic Network Representation
- Production Rules

Semantic Network Representation:

Semantic network representation involves representing knowledge in the form of a network of interconnected nodes and edges. Nodes represent concepts, while edges represent relationships between concepts. For example, a semantic network could represent knowledge about animals, with nodes for different types of animals (e.g., cats, dogs, birds) and edges representing relationships between them (e.g., "is a" for hierarchical relationships like "cats are a type of animal", and "eats" for more general relationships like "cats eat mice"). Semantic networks can be used to model a wide variety of types of knowledge, from simple taxonomies to complex causal relationships. This representation consists of mainly two types of relations:

- IS-A relation (Inheritance)
- Kind-of-relation

Example: Representing the following statements in arcs and nodes:

Statement:

- Jerry is a cat.
- Jerry is owned by Priya.
- All Mammals are animal.
- Jerry is a mammal
- Jerry is brown colored.

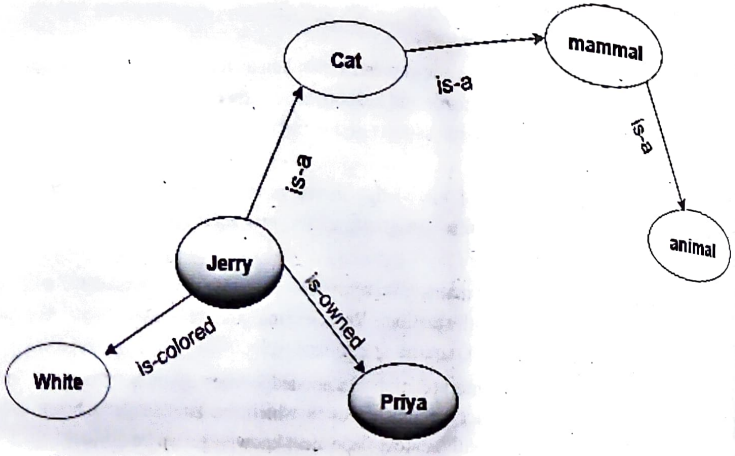


Figure: Different types of knowledge representation in forms of nodes and arcs.

Advantages:

- **Intuitive and easy to understand:** Semantic networks are easy to visualize and understand, making them a useful tool for representing knowledge for non-experts or for presenting knowledge to an audience.
- **Incremental development:** Semantic networks can be developed incrementally, allowing for easy modification and addition of new nodes and edges.
- **Supports inference:** Semantic networks can support reasoning and inference, allowing for the derivation of new knowledge from existing knowledge.

Disadvantages:

- **Limited expressiveness:** Semantic networks may not be able to represent some types of knowledge, such as complex temporal relationships or probabilistic relationships.
- **Scaling issues:** As the size and complexity of a semantic network grows, it can become difficult to manage and understand.
- **Ambiguity:** Semantic networks can be prone to ambiguity, as the same node or edge can represent different things depending on the context.
- **Not easily machine-readable:** While semantic networks are easy for humans to understand, they can be more difficult for machines to process and use in automated reasoning or decision-making systems.

Frame Representation:

Frame representation, on the other hand, involves representing knowledge in the form of a structured set of attributes and values, organized around a central concept or "frame." Frames are used to represent objects or concepts that have a complex structure, such as a car or a restaurant. Each frame has a set of attributes (e.g., "color", "price", "cuisine type") and values for those attributes (e.g., "red", "expensive", "Italian"). Frames can also have default values and inheritance relationships, allowing for more efficient representation of related concepts. For example, a "sedan" frame might inherit many of its attributes and values from a more general "car" frame. Frame representations can be used for a wide range of tasks, such as natural language understanding and image recognition.

Frame consists slots and slots values. Size and type of slots can be any kind. Slots contains names and values known as facets. A frame is also known as slot-filter knowledge representation in artificial intelligence.

Let's take an example of a frame for a book of Artificial Intelligence.

Slots	Filters
Title	Artificial Intelligence: A modern Approach
Genre	Computer Science and Engineering
Author	Peter Norvig
Edition	Second Edition
Year	1998
Chapters	12
Page	1144

Advantages:

- **Structured representation:** Frames provide a structured representation for knowledge, with clearly defined attributes and values, making it easier to organize and manipulate large amounts of data.
- **Natural language processing:** Frame representation can be used for natural language processing tasks, such as understanding the meaning of sentences in a text or responding to questions.

- **Supports inference:** Frame representations can support reasoning and inference, allowing for the derivation of new knowledge from existing knowledge.

Disadvantages:

- **Limited flexibility:** Frames can be inflexible and may not be able to capture certain types of knowledge or relationships, such as probabilistic relationships.
- **Complexity:** As the number of attributes and values in a frame increases, it can become difficult to manage and understand the frame, making it challenging to develop and maintain large knowledge bases.
- **Ambiguity:** Frame representations can be prone to ambiguity, as the same attribute can have different meanings in different frames or in different contexts.
- **Difficulty in learning:** Creating a frame representation for a particular domain or concept requires significant knowledge and expertise, which can be time-consuming and expensive.

Propositional Logic (PL) (Syntax, Semantics, Formal logic-connectives, tautology, validity, wellformed-formula, Inference using Resolution)

Propositional Logic: Propositional logic, also known as sentential logic or propositional calculus, is a branch of mathematical logic that deals with propositions, which are declarative statements that can be either true or false. Propositional logic is concerned with the formal manipulation of these propositions, without regard to their meaning or interpretation.

Advantages:

- **Simplicity:** Propositional logic is simple and easy to understand, as it deals with propositions that are either true or false.
- **Formalism:** Propositional logic has a well-defined syntax and semantics, which make it amenable to formalization and analysis.
- **Compactness:** Propositional logic allows for the representation of complex propositions using a small number of symbols, which makes it suitable for automated reasoning.
- **Modularity:** Propositional logic allows for the separation of knowledge into modules, which can be combined and reused to represent complex systems.
- **Soundness and completeness:** Propositional logic is sound and complete, which means that it is guaranteed to produce correct results and that all valid arguments can be proven using its inference rules.

Disadvantages:

- **Limited expressiveness:** Propositional logic has limited expressiveness, as it can only represent propositions that are either true or false, and cannot represent more complex relations between objects.
- **Lack of context:** Propositional logic is context-free, which means that it cannot take into account the context in which propositions are made.
- **Inability to reason about time:** Propositional logic cannot represent or reason about time, which makes it unsuitable for representing temporal relationships.

- **Inability to represent uncertainty:** Propositional logic cannot represent or reason about uncertainty, which makes it unsuitable for representing probabilistic relationships.
- **Inability to reason about negation:** Propositional logic has limited ability to reason about negation, as it can only negate whole propositions, and not parts of propositions. This can lead to problems when representing more complex relationships between objects.

Syntax:

Propositional logic has a simple syntax, consisting of a finite set of propositional variables (p, q, r, etc.) that can be combined using logical connectives. The logical connectives used in propositional logic are NOT, AND, OR, IMPLIES, and IF AND ONLY IF. These connectives are used to form compound propositions, or formulas, from simpler propositions.

The five basic logical connectives are:

- **Negation:** represented by the symbol ¬, it takes a single proposition and returns its negation. For example, ¬P represents the negation of proposition P.
- **Conjunction:** represented by the symbol ∧, it takes two propositions and returns their conjunction (i.e., their logical "and"). For example, P ∧ Q represents the proposition that both P and Q are true.
- **Disjunction:** represented by the symbol ∨, it takes two propositions and returns their disjunction (i.e., their logical "or"). For example, P ∨ Q represents the proposition that either P or Q (or both) is true.
- **Implication:** represented by the symbol →, it takes two propositions and returns their implication. For example, P → Q represents the proposition that if P is true, then Q must also be true.
- **Equivalence:** represented by the symbol ↔, it takes two propositions and returns their equivalence. For example, P ↔ Q represents the proposition that P and Q are either both true or both false.

Semantics:

The semantics of propositional logic defines the truth values of compound propositions in terms of the truth values of their constituent propositions. A truth table is a common method for defining the semantics of propositional logic, which lists all possible combinations of truth values for the propositional variables in a formula and the resulting truth value of the formula.

Formal logic-connectives:

- **NOT:** represented by the symbol ¬, is a unary operator that negates the truth value of a proposition. For example, if p is true, then ¬p is false.
- **AND:** represented by the symbol ∧, is a binary operator that is true only when both of its operands are true. For example, if p and q are both true, then p ∧ q is true.
- **OR:** represented by the symbol ∨, is a binary operator that is true when at least one of its operands is true. For example, if p is true and q is false, then p ∨ q is true.

- **IMPLIES:** represented by the symbol →, is a binary operator that represents logical implication. If p implies q, then p → q is true only when p is false or when both p and q are true.
- **IF AND ONLY IF:** represented by the symbol ↔, is a binary operator that is true when both operands have the same truth value. For example, if p and q are both true or both false, then p ↔ q is true.

Tautology:

A tautology is a compound proposition that is true regardless of the truth values of its constituent propositions. For example, p ∨ ¬p is a tautology, as it is always true, regardless of the truth value of p.

Validity:

A proposition is said to be valid if it is true in all possible interpretations. In other words, a proposition is valid if it is a tautology. For example, the proposition (p ∧ q) → p is valid, as it is true in all possible interpretations of p and q.

Well-formed formula:

A well-formed formula (WFF) is a formula in propositional logic that is constructed according to the syntax rules of the language. For example, (p ∧ q) ∨ ¬r is a WFF, but p ∨ ∧ q is not a WFF, as it violates the syntax rules of propositional logic.

Inference using Resolution:

Resolution is a commonly used inference rule in propositional logic that can be used to prove the validity or satisfiability of a propositional formula. The resolution rule involves creating a new clause that is the disjunction of the literals from two clauses, where the literals are complementary (i.e., one is the negation of the other). By applying the resolution rule repeatedly, one can derive new clauses that are logically equivalent to the original formula. If a contradiction is derived, then the original formula is unsatisfiable. If no contradiction is derived, then the formula is satisfiable.

Inference Rules:

- **Modus Ponens:** if P and P → Q are both true, we can infer that Q will be true as well.

$$\frac{P \quad P \rightarrow Q}{\therefore Q}$$

Example:

Statement-1: "If I am hungry then I go to restaurant" ==> P → Q

Statement-2: ""I am hungry" ==> P"

Conclusion: "I go to restaurant." ==> Q.

Hence, we can say that, if P → Q is true and P is true then Q will be true.

Modus Tollens: if P→ Q is true and ¬ Q is true, then ¬ P will also true.

$$\frac{\neg Q \quad P \rightarrow Q}{\therefore \neg P}$$

Example:

Statement-1: "If I am hungry then I go to restaurant" ==> P→ Q

Statement-2: "I do not go to the restaurant."==> ¬Q

Conclusion: Which infers that "I am not hungry" => ¬P

Hypothetical Syllogism: if P→R is true whenever P→Q is true, and Q→R is true.

$$\frac{P \rightarrow Q \quad Q \rightarrow R}{\therefore P \rightarrow R}$$

Example:

Statement-1: If you have my car key then you can unlock my car. P→Q

Statement-2: If you can unlock my car then you can take my ATM card. Q→R

Conclusion: If you have my car key then you can take my ATM card. P→R

Disjunctive Syllogism: if PVQ is true, and ¬P is true, then Q will be true.

$$\frac{P \vee Q \quad \neg P}{\therefore Q}$$

Example:

Statement-1: Today is Saturday or Sunday. ==> PVQ

Statement-2: Today is not Saturday. ==> ¬P

Conclusion: Today is Sunday. ==> Q

Addition: if P is true, then PVQ will be true.

$$\frac{P}{\therefore P \vee Q}$$

Example:

Statement-1: I have a kit kat. ==> P

Statement-2: I have a dairy milk. ==> Q

Conclusion: I have kit kat or dairy milk. ==> (PVQ)

Simplification: if P∧ Q is true, then Q or P will also be true.

$$\frac{P \wedge Q}{\therefore P}$$

- **Resolution:** if PVQ and ¬ PAR is true, then QVR will also be true.

$$\frac{\begin{array}{l} P \vee Q \\ \neg P \vee R \end{array}}{\therefore Q \vee R}$$

Table: Rule of Inference

Rule of Inference	Tautology	Name
$\frac{p \quad p \to q}{\therefore q}$	$(p \wedge (p \to q)) \to q$	Modus ponens
$\frac{\neg q \quad p \to q}{\therefore \neg p}$	$(\neg q \wedge (p \to q)) \to \neg p$	Modus tollens
$\frac{p \to q \quad q \to r}{\therefore p \to r}$	$((p \to q) \wedge (q \to r)) \to (p \to r)$	Hypothetical syllogism
$\frac{p \vee q \quad \neg p}{\therefore q}$	$((p \vee q) \wedge \neg p) \to q$	Disjunctive syllogism
$\frac{p}{\therefore p \vee q}$	$p \to (p \vee q)$	Addition
$\frac{p \wedge q}{\therefore p}$	$(p \wedge q) \to p$	Simplification
$\frac{p \quad q}{\therefore p \wedge q}$	$((p) \wedge (q)) \to (p \wedge q)$	Conjunction
$\frac{p \vee q \quad \neg p \vee r}{\therefore q \vee r}$	$((p \vee q) \wedge (\neg p \vee r)) \to (q \vee r)$	Resolution

Predicate Logic (FOPL, Syntax, Semantics, Quantification, Rules of inference, unification, resolution refutation system)

Predicate logic:

Predicate logic, also known as first-order logic (FOL) or first-order predicate calculus (FOPL), is an extension of propositional logic that allows for the representation of more complex statements, including statements about objects, relations, and properties. In predicate logic, statements are expressed using predicates, which are functions that take one or more arguments and return a truth value.

Advantages and Dis-advantages:

Advantages:

- **Increased expressiveness:** FOPL allows for the representation of complex relationships between objects, including quantification, variables, and functions, which makes it more expressive than propositional logic.
- **Improved reasoning ability:** FOPL provides more powerful reasoning mechanisms than propositional logic, such as resolution, unification, and the ability to handle negation, which makes it more suitable for handling complex knowledge and reasoning tasks.
- **Ability to represent complex relationships:** FOPL allows for the representation of complex relationships between objects, such as the relationships between different parts of a system, the properties of those parts, and the relationships between different properties.
- **Formal semantics:** FOPL has a formal semantics, which allows for precise and rigorous reasoning about the logical implications of propositions.
- **Suitable for automated reasoning:** FOPL is suitable for automated reasoning, which makes it a useful tool for artificial intelligence and other computer science applications.

Disadvantages:

- **Increased complexity:** FOPL is more complex than propositional logic, which can make it more difficult to understand and use.
- **Lack of context:** FOPL is context-free, which means that it cannot take into account the context in which propositions are made.
- **Difficulties with certain types of reasoning:** FOPL may have difficulties with certain types of reasoning tasks, such as reasoning about time, causality, or uncertainty, which may require more specialized logical languages.
- **Incomplete reasoning mechanisms:** FOPL's reasoning mechanisms, such as resolution and unification, may not always be complete, which means that some valid arguments may not be provable using these methods.
- **Limited scalability:** FOPL's expressiveness can make it difficult to reason about very large knowledge bases, which may require more specialized algorithms and techniques to handle.

Syntax:

The syntax of predicate logic includes two types of symbols: predicate symbols and variable symbols. Predicate symbols represent predicates or relations, and variable symbols represent objects or individuals. Quantifiers, such as "for all" and "there exists," are also used in predicate logic.

Syntax of FOPL:

- **Connectives:** There are five connectives: negation (~), conjunction (&), inclusive disjunction (v), implication (→), Equivalence (←→)
- **Quantifiers:** There are two quantifiers: Existential Quantification ($), Universal Quantification (")

- **Constants:** Constants are fixed value which are denoted by numbers, words, small letters. e.g., khusboo, 0, 1, copy etc.
- **Variables:** Variables are terms that can assume different values over a given domain.
- **Predicates:** A predicate is defined as a relation that binds two atoms together. E.g., Cat likes milks, is represented as:
  - LIKES (cat, milks).
  - Here LIKES is a predicate that links two atoms "cat" and "milks"
- **Functions:** Functions can be used as an argument. E.g., "Ganesh's father is Kuber's father is represented as:
  - FATHER (father (Ganesh), Kuber)
  - Here FATHER is a predicate and father(Ganesh) is a function to indicate kuber's father.

Semantics:

The semantics of predicate logic define the meaning of predicates and quantifiers in terms of objects and their properties. A domain of discourse is used to define the set of objects that can be referred to in a formula. Interpretations are used to define the truth values of formulas in terms of the objects and their properties.

Quantification:

Quantifiers are used to indicate the extent or scope of a statement that includes variables. Specifically, quantifiers indicate whether a statement applies to all or some of the elements in a particular domain of discourse. Predicate logic includes two types of quantifiers: the universal quantifier (∀) and the existential quantifier (∃). The universal quantifier is used to express that a formula is true for all objects in the domain of discourse, while the existential quantifier is used to express that there exists at least one object in the domain of discourse for which a formula is true.

- **Universal quantifier (∀):** The universal quantifier (∀) is used to express that a statement applies to all elements in a domain of discourse. For example, the statement "For all x, P(x)" is represented in predicate logic as "∀x P(x)," which means that "P" applies to every possible value of "x." The symbol "∀" is read as "for all" or "for every."
- **Existential quantifier (∃):** The existential quantifier (∃) is used to express that a statement applies to at least one element in a domain of discourse. For example, the statement "There exists an x such that P(x)" is represented in predicate logic as "∃x P(x)," which means that there is at least one value of "x" for which "P" is true. The symbol "∃" is read as "there exists" or "there is at least one."

Rules of inference:

The rules of inference in predicate logic are similar to those in propositional logic, but with the addition of rules for quantification. These rules include universal generalization (from a statement that is true for a specific object, infer that the statement is true for all objects), existential instantiation (from a statement that a formula is true for at least one object, infer that there exists an object for which the formula is true), and quantifier negation (from a statement that a formula is true for all objects, infer that the negation of the formula is true for no object).

Unification:

Unification is a process used in predicate logic to find substitutions that make two or more formulas equivalent. Unification involves finding a common substitution that can be applied to two or more formulas, resulting in equivalent formulas. For example, the formulas "P(x, y)" and "P(a, b)" can be unified by substituting "a" for "x" and "b" for "y".

Resolution refutation system:

A resolution refutation system is a method for proving the validity or satisfiability of a formula in predicate logic. The resolution refutation system involves converting the formula to negation normal form, and then applying the resolution rule, which involves creating a new clause that is the disjunction of the literals from two clauses, where the literals are complementary (i.e., one is the negation of the other). By applying the resolution rule repeatedly, one can derive new clauses that are logically equivalent to the original formula. If a contradiction is derived, then the original formula is unsatisfiable. If no contradiction is derived, then the formula is satisfiable.

Bayes Rule and its uses:

Bayes' rule is a powerful theorem in probability theory that has important applications in artificial intelligence (AI). It is named after the Reverend Thomas Bayes, an 18th-century mathematician and theologian.

Bayes' rule provides a way to calculate the probability of a hypothesis (H) given some observed evidence (E). The formula is:

$$P(H|E) = P(E|H) * P(H) / P(E)$$

where P(H|E) is the probability of the hypothesis H given the evidence E, P(E|H) is the probability of the evidence E given the hypothesis H, P(H) is the prior probability of the hypothesis, and P(E) is the probability of the evidence.

Bayes' rule is particularly useful in AI for tasks such as:

- **Bayesian networks:** Bayesian networks are a probabilistic graphical model that uses Bayes' rule to model and reason about uncertainty in complex systems.
- **Machine learning:** Bayes' rule is used in machine learning algorithms such as Naive Bayes and Bayesian networks to model probability distributions and make predictions based on data.
- **Natural language processing:** Bayes' rule is used in natural language processing applications such as sentiment analysis and spam detection to classify text based on probabilistic models.
- **Decision making:** Bayes' rule is used in decision-making applications such as Bayesian decision theory to make optimal decisions based on uncertain information.

Bayesian Network:

Bayesian networks are graphical models that represent probabilistic relationships between a set of random variables. The nodes in the graph represent the variables and the directed edges represent the dependencies between them. Bayesian networks are used to make predictions,

decisions, or inferences about a particular variable of interest given observations of other variables in the network.

In a Bayesian network, each node is associated with a conditional probability distribution that describes the probability of the node given its parents in the network. These conditional probabilities can be learned from data or specified by an expert. By combining these conditional probabilities, the network can be used to compute the joint probability distribution over all the variables in the network.

One of the main advantages of Bayesian networks is that they allow for efficient probabilistic inference. In particular, the network can be used to compute the posterior distribution of a particular variable given observed evidence, using the Bayes' rule. This makes Bayesian networks a powerful tool for decision making, prediction, and inference in a wide range of domains, including medicine, finance, and engineering.

Bayesian networks are also useful for modeling uncertainty and dealing with missing data. They can be used to propagate uncertainties and make predictions under uncertainty. They also provide a framework for handling missing data and imputing missing values using probabilistic inference.

Areas of implementation of Bayesian Network:

- **Medical diagnosis:** A Bayesian network can be used to diagnose a disease based on observed symptoms and medical history. The network would include nodes representing symptoms and medical conditions, and the edges would represent the causal relationships between them.
- **Financial risk management:** A Bayesian network can be used to model financial risk by representing the relationships between different financial assets and economic factors. The network would include nodes representing asset prices, interest rates, inflation, and other economic indicators, and the edges would represent the causal relationships between them.
- **Image recognition:** A Bayesian network can be used to recognize objects in images by representing the relationships between image features and object classes. The network would include nodes representing image features such as color, texture, and shape, and nodes representing object classes such as cars, people, and animals.
- **Natural language processing:** A Bayesian network can be used to model the relationships between words and their meanings in natural language. The network would include nodes representing words and their meanings, and the edges would represent the semantic relationships between them.
- **Sensor networks:** A Bayesian network can be used to model the relationships between sensor measurements and environmental variables in sensor networks. The network would include nodes representing sensor measurements and environmental variables such as temperature, humidity, and air quality, and the edges would represent the causal relationships between them.

Reasoning in Belief Network:

Belief networks, also known as Bayesian networks, provide a graphical representation of probabilistic relationships between a set of random variables. They can be used to perform various forms of probabilistic reasoning, such as computing probabilities of events, making predictions, and making decisions under uncertainty. Here are some common forms of reasoning in belief networks:

Inference: Given a belief network and some evidence (i.e., observed values of some of the variables), one can use the network to infer the posterior probabilities of other variables in the network. This is done using Bayes' rule, which involves combining the prior probabilities specified by the network with the likelihoods of the evidence to compute the posterior probabilities of the other variables.

Diagnosis: Belief networks can be used for diagnosing a system or a disease given some observed symptoms. The network would include nodes representing the symptoms and nodes representing the possible causes of the symptoms. Given some observed symptoms, one can use the network to infer the probabilities of different causes, and thus diagnose the underlying condition.

Prediction: Belief networks can be used to make predictions about the future given some observed data. The network would include nodes representing the past data and nodes representing the possible outcomes. Given the observed data, one can use the network to infer the probabilities of the different outcomes, and thus make predictions about the future.

Decision-making: Belief networks can be used to make decisions under uncertainty. The network would include nodes representing the different actions that can be taken, nodes representing the different outcomes that can occur, and nodes representing the uncertain variables that affect the outcomes. By combining the probabilities specified by the network with some utility function that specifies the desirability of each outcome, one can use the network to choose the best action to take.

9.4 EXPERT SYSTEM AND NATURAL LANGUAGE PROCESSING

Expert System:

An expert system is an artificial intelligence (AI) system that is designed to emulate the decision-making abilities of a human expert in a particular field. It is a computer program that uses knowledge and reasoning techniques to solve complex problems and make decisions that would normally require human expertise. Expert systems are typically used in situations where there is a shortage of human experts or where the cost of consulting a human expert is too high.

Rule-based expert systems: These are the most common type of expert systems, and are based on a set of if-then rules. They use a rule engine to match the conditions of the rules to the input data, and then execute the actions associated with the matched rules. Rule-based systems are often used in diagnostic and decision-making applications.

Types of ES:

- **Frame-based expert systems:** These systems represent knowledge as a set of frames, which are similar to objects in object-oriented programming. Frames are used to represent concepts and relationships, and can contain attributes and methods. Frame-based systems are often used in natural language processing and information retrieval applications.
- **Fuzzy expert systems:** These systems are designed to handle uncertain or imprecise data. They use fuzzy logic to represent and reason about uncertain data, and are often used in applications where there is no clear-cut distinction between true and false or good and bad.
- **Neural expert systems:** These systems use artificial neural networks to learn and reason about data. Neural networks are modeled after the structure and function of the human brain, and can be trained to recognize patterns and make predictions. Neural expert systems are often used in image and speech recognition applications.
- **Neuro-fuzzy expert systems:** These systems combine fuzzy logic and neural networks to handle uncertain and imprecise data. They use fuzzy logic to represent uncertainty and neural networks to learn and reason about data. Neuro-fuzzy systems are often used in control and optimization applications.

Steps to develop an ES:

- **Identify the problem domain:** The first step in developing an expert system is to identify the problem domain, which is the area of expertise that the system will focus on. This involves analyzing the problem, gathering information about the domain, and identifying the knowledge sources that will be used.
- **Design the system:** The next step is to design the expert system, which involves defining the system requirements, selecting the knowledge representation and inference methods, and designing the user interface. This step also involves creating a knowledge base, which is a collection of domain-specific knowledge that the system will use to make decisions.
- **Develop the prototype:** Once the system design is complete, the next step is to develop a prototype of the system. This involves implementing the knowledge base, building the inference engine, and developing the user interface. The prototype should be tested and refined to ensure that it meets the system requirements.
- **Test and refine the prototype:** In this step, the prototype is tested and evaluated to ensure that it is functioning correctly and providing accurate results. Any errors or issues are identified and addressed, and the prototype is refined to improve its performance.
- **Develop and complete the expert system:** Once the prototype has been tested and refined, the final step is to develop and complete the expert system. This involves integrating the prototype into a production environment, adding any necessary features, and conducting final testing and evaluation.
- **Maintain the system:** The final step in the development process is to maintain the expert system, which involves monitoring its performance, updating the knowledge base as necessary, and making any necessary modifications or improvements to the system.

Benefits of an Expert System:

- **Increased accuracy and consistency:** Expert systems are designed to provide accurate and consistent results, as they use a predefined set of rules and knowledge to make decisions. Unlike humans, they do not make errors due to fatigue, emotions, or other factors.
- **Reduced reliance on human expertise:** Expert systems can replace or supplement human experts in situations where there is a shortage of experts or where consulting a human expert is too costly or time-consuming.
- **Improved decision-making:** Expert systems can provide fast and reliable decisions based on their knowledge and reasoning capabilities. They can also provide explanations for their decisions, which can help users understand the rationale behind the system's output.
- **Knowledge management:** Expert systems can store and manage knowledge and expertise in a structured and organized manner. This can help organizations retain and share knowledge, and also reduce the risk of knowledge loss due to employee turnover.
- **Increased productivity:** Expert systems can automate complex and time-consuming tasks, freeing up human experts to focus on higher-level tasks that require creativity and critical thinking.
- **Scalability:** Expert systems can be scaled up to handle large volumes of data and to support multiple users simultaneously, making them suitable for use in large organizations.
- **Adaptability:** Expert systems can be updated and modified easily to reflect changes in the domain knowledge or to address new problems. This makes them a flexible and adaptable solution for a variety of business problems.

Challenges of an Expert Systems:

- **Difficulty in acquiring and maintaining knowledge:** Expert systems require significant time and effort to acquire and maintain domain-specific knowledge and rules. This process can be costly and time-consuming, and may require ongoing updates and maintenance to keep the system up-to-date.
- **Limited domain expertise:** Expert systems are designed to provide advice and recommendations within a specific domain. They cannot handle tasks outside their domain, and their advice may be limited to the knowledge and rules stored in their knowledge base.
- **Lack of common sense reasoning:** Expert systems may not be able to reason in a common sense manner, and may not be able to handle tasks that require general knowledge or intuition.
- **Difficulty in explaining decisions:** Expert systems may not be able to explain their decisions in a way that is easily understandable to users, leading to a lack of trust in the system.

- **Limited user interaction:** Expert systems may not be able to handle tasks that require human interaction, such as tasks that require emotional intelligence or social skills.
- **Integration with existing systems:** Integrating an expert system with existing systems and processes can be a complex and challenging process, requiring significant planning and coordination.

Architecture of an Expert System:

- **Knowledge Base:** This is the part of the system that contains all the knowledge about a particular domain. It stores the rules, facts, and other relevant information that the expert system uses to make decisions.
- **Inference Engine:** This component of the expert system uses the knowledge stored in the knowledge base to make inferences and draw conclusions. It applies rules and reasoning techniques to the available data and generates recommendations.
- **User Interface:** This is the part of the expert system that interacts with the user. It presents questions to the user and receives answers in a user-friendly and intuitive manner.
- **Explanation Facility:** This component of the expert system provides an explanation to the user about how the system arrived at a particular recommendation. This helps the user to understand the system's reasoning and to build trust in the system.
- **Knowledge Acquisition System:** This is the part of the expert system that helps to acquire new knowledge and update the knowledge base. It may include tools for capturing knowledge from human experts or from other sources.
- **Knowledge Refinement System:** This component of the expert system is responsible for continuously refining and improving the knowledge base. It may involve updating rules or modifying the reasoning techniques used by the inference engine.

Knowledge Acquisition:

Knowledge acquisition is the process of capturing, organizing, and representing the knowledge of human experts in a particular domain in a form that can be used by an expert system. It is a crucial component of the development of an expert system, as it determines the accuracy and effectiveness of the system.

The knowledge acquisition process can be divided into several steps:

- **Identify domain experts:** The first step in the knowledge acquisition process is to identify individuals who have expertise in the domain that the expert system is intended to operate in. These individuals are typically subject matter experts (SMEs) or domain experts.
- **Gather knowledge:** Once domain experts have been identified, the knowledge acquisition process involves gathering knowledge from these experts. This can be done using various techniques such as interviews, surveys, or observation. The goal is to capture the expert's knowledge in a structured form that can be used by the expert system.

- **Organize knowledge:** The next step in the knowledge acquisition process is to organize the knowledge that has been gathered. This involves identifying patterns and relationships in the knowledge, and structuring it in a way that can be easily used by the expert system. Knowledge can be organized in a variety of ways, including decision trees, rule-based systems, or case-based reasoning.
- **Represent knowledge:** Once the knowledge has been organized, it needs to be represented in a way that can be used by the expert system. This involves creating a knowledge base that contains rules, facts, and other information that the system can use to make decisions and provide recommendations.
- **Validate and refine knowledge:** After the knowledge has been represented in the knowledge base, it needs to be validated and refined. This involves testing the expert system to ensure that it is making accurate and effective decisions, and making modifications to the knowledge base as needed to improve performance.

Natural Language Processing:

NLP is a branch of artificial intelligence that focuses on the interaction between human language and computers. NLP aims to develop computer programs that can understand, interpret, and generate human language.

NLP Terminologies:

- **Phonology:** The study of the sound patterns of language, including how sounds are produced, perceived, and combined.
- **Morphology:** The study of the structure of words and the rules for combining morphemes (the smallest units of meaning) to create words.
- **Morpheme:** The smallest unit of meaning in a language, such as a prefix, suffix, or root.
- **Syntax:** The study of the rules that govern the structure of sentences and how words are combined to create meaning.
- **Semantics:** The study of meaning in language, including how words and sentences convey meaning and how meaning is inferred from context.
- **Pragmatics:** The study of how language is used in context, including the social and cultural factors that influence communication.
- **Discourse:** The study of how language is used to create larger units of communication, such as conversations, narratives, and speeches.
- **Word knowledge:** The understanding of a word's meaning, including its denotation (dictionary definition) and connotation (emotional or cultural associations).

Advantages of NLP:

- **Increased efficiency:** NLP allows computers to process human language faster and more accurately than manual analysis, leading to increased efficiency in tasks such as data analysis and customer service.

- **Improved accuracy:** NLP algorithms can identify patterns and relationships in large volumes of text data that may not be immediately apparent to humans, leading to improved accuracy in tasks such as sentiment analysis and text classification.
- **Enhanced personalization:** NLP can be used to create personalized experiences for users by analyzing their language and tailoring recommendations or responses accordingly.
- **Facilitated communication:** NLP can be used to facilitate communication between speakers of different languages, or between humans and machines, leading to improved accessibility and ease of use.

Disadvantages of NLP:

- **Ambiguity:** Human language is often ambiguous, and NLP algorithms may struggle to accurately interpret the meaning of certain words or phrases, leading to errors or misinterpretations.
- **Complexity:** NLP is a complex field that requires a significant amount of computational resources and expertise to implement effectively, which may limit its accessibility to smaller organizations or individuals.
- **Bias:** NLP algorithms may reflect the biases of their creators or the data they are trained on, leading to potential issues with fairness and accuracy.
- **Lack of context:** NLP algorithms may struggle to accurately interpret language that is highly dependent on context, such as idiomatic expressions or jokes, leading to potential misunderstandings or misinterpretations.

NLP is difficult because:

- Imparting world knowledge is difficult: "The black mobile ate the pizza"
- Fictitious words: "People on Venus can fly"
- Poorly defined scopes: "People like iPhone". Does this mean all people likes iPhone?
- Language is changing and evolving
- Different ways of Parsing a sentence
- Word category ambiguity
- Word sense ambiguity
- Words can mean more than their sum of parts (The Times of India)

Natural Language Understanding and Natural Language Generation:

The components of NLP are NLU and NLG. Natural Language Understanding (NLU) and Natural Language Generation (NLG) are two important aspects of NLP.

NLU: Natural Language Understanding (NLU) is the process of analyzing and interpreting human language by computers. NLU involves breaking down language into smaller components, such as words and phrases, and then using algorithms to understand the meaning and context of those components. NLU is used in a variety of applications, such as virtual assistants, chatbots, and sentiment analysis, to enable computers to understand and respond to human language.

Steps in NLU:

Raw speech signal
↓ Speech recognition
Sequence of words spoken
↓ Syntactic analysis using knowledge of the grammar
Structure of the sentence
↓ Semantic analysis using info. about meaning of words
Partial representation of meaning of sentence
↓ Pragmatic analysis using info. about context
Final representation of meaning of sentence

NLG: Natural Language Generation (NLG) is the process of creating natural language text or speech output from computer-generated data. NLG involves taking structured data and generating text that is indistinguishable from text written by humans. NLG is used in a variety of applications, such as creating reports or summaries from large data sets, creating personalized messages for customers, and generating news articles.

NLU and NLG are complementary processes, with NLU allowing computers to understand human language, and NLG allowing computers to communicate with humans in a way that is natural and easy to understand. Together, NLU and NLG are important components of many NLP applications, and they are becoming increasingly sophisticated and accurate with advances in machine learning and artificial intelligence.

Steps of Natural Language Processing:

- **Lexical Analysis (Tokenization):** This step involves breaking down the text into individual words, phrases, and symbols, which are referred to as tokens. This process is called tokenization, and it involves identifying patterns in the text to split it into meaningful chunks that can be analyzed further. In this stage, the text is also analyzed for spelling, punctuation, and capitalization.
- **Syntactic Analysis (Parsing):** This step involves analyzing the grammatical structure of the text. This is done by parsing the text using various algorithms to identify the relationships between the words and phrases. The goal is to create a parse tree that shows how the text is structured according to the rules of the language.
- **Semantic Analysis:** This step involves understanding the meaning of the text. This is done by identifying the entities in the text and the relationships between them. This can involve techniques such as named entity recognition, word sense disambiguation, and semantic role labeling. The goal is to create a representation of the meaning of the text that can be used for further analysis.
- **Discourse Integration:** This step involves analyzing the text in its broader context. This includes understanding the relationships between sentences and paragraphs, and

identifying the overall theme or purpose of the text. This step can involve techniques such as co-reference resolution, anaphora resolution, and discourse analysis.

- **Pragmatic Analysis:** This step involves analyzing the text in terms of its intended meaning and its effect on the reader or listener. This can involve identifying the speaker's tone, intention, and attitude, and understanding the context in which the text is being used. Pragmatic analysis is important for tasks such as sentiment analysis, opinion mining, and natural language generation.

Applications of NLP:

- **Sentiment Analysis:** This involves analyzing the sentiment expressed in a text, such as positive or negative feelings about a product or service. It is used in marketing research and social media analysis.
- **Machine Translation:** NLP is used in machine translation to automatically translate text from one language to another. This is used in online translation services, language learning software, and international communication.
- **Chatbots and Virtual Assistants:** NLP is used in chatbots and virtual assistants to understand and respond to natural language queries. This is used in customer service, personal assistants, and other applications.
- **Text Summarization:** NLP is used to summarize long texts, such as news articles or scientific papers, into shorter, more manageable summaries. This is used in news aggregation services and research analysis.
- **Information Extraction:** NLP is used to automatically extract information from unstructured text, such as named entities, events, and relationships between entities. This is used in information retrieval, question answering systems, and intelligence analysis.
- **Speech Recognition:** NLP is used in speech recognition systems to transcribe spoken words into text. This is used in voice assistants, dictation software, and call center automation.
- **Natural Language Generation:** NLP is used in natural language generation to automatically generate text from structured data, such as weather forecasts or financial reports. This is used in content generation, data analysis, and report generation.

Challenges of NLP:

- **Ambiguity:** Natural language is often ambiguous, with multiple possible interpretations of a given sentence. This can make it difficult for NLP systems to accurately understand and generate natural language.
- **Complexity:** Natural language is also highly complex, with a large vocabulary and complex sentence structures. This can make it difficult for NLP systems to accurately analyze and generate text.
- **Context:** Understanding the context in which a sentence is used is critical for accurate NLP. However, context can be difficult to identify, and may vary depending on factors such as the speaker's tone and background knowledge.

Cultural and ... language varies greatly across different languages and cultures, which can make it difficult to develop NLP systems that work effectively across multiple languages and cultures.

Data availability and quality: NLP systems require large amounts of high-quality data to learn from. However, data may not always be available or of sufficient quality, which can limit the effectiveness of NLP systems.

Ethics and bias: NLP systems can also be subject to bias and ethical concerns, such as bias-based training data, discriminatory language models, and privacy concerns.

Machine Vison Concepts:

Machine vision is a branch of artificial intelligence and computer science that aims to enable computers to interpret and understand images and video content, similar to how humans do. Some of the key concepts in machine vision include:

Image processing: The manipulation and enhancement of digital images to extract useful information.

Computer vision: The use of algorithms to enable computers to interpret images and videos, and to recognize and identify objects within them.

Object recognition: The ability to identify and classify objects within an image or video stream.

Feature extraction: The process of identifying and isolating the relevant features of an image or video stream, such as edges or corners, in order to make it easier to recognize objects and patterns.

Pattern recognition: The process of identifying and classifying recurring patterns within an image or video stream, which can be used to make predictions and decisions.

Machine learning: The use of algorithms to enable computers to learn from large datasets of images and videos, and to improve their ability to recognize and classify objects.

Deep learning: A type of machine learning that uses neural networks with multiple layers to enable computers to learn from complex and diverse datasets.

Convolutional neural networks: A type of deep learning algorithm that is particularly well-suited to image and video recognition, and that uses a series of filters to extract relevant features from an image or video stream.

Semantic segmentation: The process of dividing an image or video stream into different regions, based on the objects or features present within each region.

Object tracking: The ability to follow and track the movement of objects within an image or video stream over time.

Machine Vision Stages:

Image acquisition: The first stage of machine vision involves capturing images or video using cameras or other types of sensors. This can involve setting up lighting, choosing the right camera settings, and capturing multiple images from different angles or in different lighting conditions.

- **Image Preprocessing:** Once the images have been captured, they may need to be preprocessed to remove noise, correct distortions, or enhance specific features. This stage can involve filtering, thresholding, and other types of image processing techniques.
- **Image Segmentation:** The next stage is to separate the image into its component parts, such as objects or regions of interest. This can involve techniques such as thresholding, edge detection, or clustering.
- **Image Analysis/ Feature extraction:** Once the objects or regions of interest have been segmented, the next stage is to extract relevant features from them. This can involve techniques such as texture analysis, shape analysis, or color analysis.
- **Pattern recognition:** The next stage is to recognize and classify the objects in the image. This can involve techniques such as template matching, machine learning, or deep learning.

Robotics:

Robotics is a field of study and engineering that deals with the design, construction, and operation of robots. A robot is an intelligent machine that is capable of carrying out complex tasks automatically, with or without human intervention. Robotics involves a variety of different technologies, including computer science, mechanical engineering, electrical engineering, and artificial intelligence.

The primary goals of robotics are to create machines that can perform tasks that are too dangerous, difficult, or repetitive for humans to do, as well as to improve efficiency and productivity in industries such as manufacturing, agriculture, and healthcare. Robotics is also used in research, exploration, and space exploration.

Robots can be classified into several categories based on their form and function. These include:

- **Industrial robots:** These robots are used in manufacturing and assembly lines to perform tasks such as welding, painting, and assembling.
- **Service robots:** These robots are designed to assist humans in various tasks, such as cleaning, security, and healthcare.
- **Medical robots:** These robots are used in healthcare to assist with surgery, patient care, and medical imaging.
- **Military robots:** These robots are used in the military for surveillance, reconnaissance, and bomb disposal.
- **Educational robots:** These robots are used in educational settings to teach programming and robotics concepts to students.

Robot control approaches:

- **Reactive Control:** Don't think, (re)act.
- **Deliberative (Planner-based) Control:** Think hard, act later.
- **Hybrid Control:** Think and act separately & concurrently.
- **Behavior-Based Control (BBC):** Think the way you act.

duction to Machine Learning:

ine learning is a field of computer science and artificial intelligence that involves the opment of algorithms and statistical models that enable computer systems to matically improve their performance on a specific task, without being explicitly rammed to do so. In other words, machine learning allows computers to learn from ence, by analyzing and recognizing patterns in data.

ne learning algorithms are designed to analyze large volumes of data, and to identify erns, trends, and relationships within that data. Once the algorithm has learned to recognize se patterns, it can then use that knowledge to make predictions or decisions based on new a. Machine learning is used in a wide range of applications, including computer vision, al language processing, speech recognition, and predictive analytics.

ations of ML:

Data quality: Machine learning models are only as good as the data they are trained on. If the data is incomplete, inconsistent, or biased, the resulting models will also be flawed.

Overfitting: Machine learning models can become too complex and "overfit" to the training data, meaning that they become very good at predicting the training data but perform poorly on new, unseen data.

Interpretability: Many machine learning models are complex and difficult to interpret, which can make it hard to understand why they are making certain decisions.

Generalization: Machine learning models are often designed to solve specific problems, and may not be able to generalize to new or different types of problems.

Scalability: Machine learning models can be computationally expensive, which can limit their scalability for large-scale applications.

Ethical concerns: Machine learning models can perpetuate and even amplify bias and discrimination, especially if they are trained on biased data or used to make decisions that affect people's lives.

Human expertise: While machine learning can be a powerful tool, it still requires human expertise to properly design, train, and evaluate the models.

tures of ML:

Automation: Machine learning automates the process of building models that can learn from data, reducing the need for manual programming and intervention.

Learning from data: Machine learning algorithms learn patterns and relationships in data and use that information to make predictions or take actions.

Adaptation: Machine learning models can adapt to new data and changing environments, improving their accuracy and usefulness over time.

Scalability: Machine learning algorithms can be applied to large and complex datasets, making it possible to process and analyze vast amounts of data.

- **Generalization:** Machine learning models can generalize beyond the specific examples they were trained on, making it possible to apply them to new situations and make accurate predictions.
- **Prediction and decision-making:** Machine learning models can make predictions and decisions based on data, helping automate decision-making processes.

Concept of Learning:

Learning refers to the process by which an algorithm is trained to identify patterns or relationships in data, and to use that information to make predictions or take actions on new data. The process of learning typically involves presenting the algorithm with a set of training data, along with the correct output or label for each example in the data. The algorithm then uses that information to adjust its parameters or structure to better match the data, so that it can make accurate predictions on new, unseen data.

Inductive Learning: Inductive learning, also known as inductive reasoning, is a bottom-up approach to machine learning. It involves analyzing specific examples and patterns in data to develop general rules or hypotheses. The goal of inductive learning is to find patterns in the data that can be used to make predictions or classifications on new, unseen data. Inductive learning is often used in supervised learning, where the algorithm learns from labeled data.

Deductive Learning: Deductive learning is a top-down approach to machine learning. It involves starting with general rules or hypotheses and using them to make predictions or classifications on specific examples. Deductive learning is often used in unsupervised learning, where the algorithm has to find patterns or structure in the data on its own, without any labeled examples.

Components in Learning Systems:

- **Environment:** This refers to the external context in which the learning system operates. It includes the inputs and outputs that the system interacts with, as well as any physical or social factors that may affect the learning process.
- **Performance element:** This is the part of the learning system that is responsible for taking actions in response to the inputs received from the environment. It uses the knowledge and rules provided by the knowledge base to make decisions and produce outputs.
- **Knowledge base:** This component stores the information and rules that the performance element uses to make decisions. It includes the data, models, and algorithms that the system needs to operate.
- **Learning element:** This is the part of the learning system that is responsible for improving the system's performance over time. It uses feedback from the environment to modify the knowledge and rules stored in the knowledge base, allowing the system to adapt and improve its performance.

Learning Situations:

- **Rote learning:** In this type of learning, the algorithm simply memorizes a set of inputs and outputs. It does not try to understand the underlying patterns in the data, and can only produce accurate outputs for the specific inputs that it has memorized.

Learning by being told: In this type of learning, the algorithm is given explicit instructions or rules that it can use to make predictions. The rules may be provided by a human expert or another source of knowledge.

Learning by example: In this type of learning, the algorithm is trained on a set of examples, with each example consisting of an input and an output. The algorithm learns to learn the underlying patterns in the data, so that it can make accurate predictions for new inputs that it has not seen before.

Learning by analogy: In this type of learning, the algorithm is trained on a set of examples that are similar to the problem it is trying to solve. It then uses this knowledge to make predictions for new inputs that are similar to the examples it has seen.

planation-based learning (EBL):

is a type of machine learning that uses human knowledge to guide the learning process. a form of inductive learning that involves generalizing from specific examples, but it also es use of domain-specific knowledge to help guide the learning process.

EBL, a learning algorithm is provided with a set of training examples and a set of background knowledge, which is used to generate a set of rules that can be used to make dictions on new data. When the algorithm encounters a new example, it first tries to apply background knowledge to generate a hypothesis about what the correct answer might be. hen tests this hypothesis against the training examples and modifies it based on the results.

L has been used in a variety of applications, including natural language processing, image cognition, and robotics. It is particularly useful in domains where domain-specific owledge can be used to guide the learning process and improve the accuracy of the learned dels.

tegories of ML techniques:

pervised, unsupervised, and reinforcement learning are three broad categories of machine arning techniques.

Supervised Learning: In supervised learning, the machine is provided with labeled data (input data and their corresponding output data) and it learns to make predictions by generalizing from that labeled data. The goal is to map inputs to outputs accurately. The machine tries to learn the underlying patterns or relationships between the input and output data. Examples of Supervised learning are: Linear Regression, Nearest Neighbor, Gaussian Naive Bayes, Decision Trees, Support Vector Machine (SVM), Random Forest. Two types of supervised learnings are:

- **Classification:** Classification is the task of assigning a label or category to a new input example based on its features. In other words, it involves predicting a discrete output value. Some common examples of classification tasks include image classification, spam detection, sentiment analysis, and disease diagnosis. For example, a model trained on a dataset of images of fruits and their corresponding labels used to classify a new image of a fruit as an apple, banana, or orange.

- **Regression:** Regression is the task of predicting a continuous output value based on a set of input features. In other words, it involves predicting a numerical value. Some common examples of regression tasks include predicting housing prices, stock prices, and temperature. For example, a model trained on a dataset of housing prices and their corresponding features (e.g., number of bedrooms, square footage, location) could be used to predict the price of a new house based on its features.

- **Unsupervised Learning:** In unsupervised learning, the machine is provided with unlabeled data and it tries to learn the underlying patterns or relationships within that data without any prior knowledge of the structure of that data. It is used for clustering and dimensionality reduction. Clustering is the grouping of similar objects together, while dimensionality reduction is the reduction of the number of variables in the data set. Examples of un-supervised learning are: K-Means Clustering, DBSCAN – Density-Based Spatial Clustering of Applications with Noise, BIRCH – Balanced Iterative Reducing and Clustering using Hierarchies, Hierarchical Clustering.

- **Reinforcement Learning:** Reinforcement learning is a type of machine learning in which an agent learns to behave in an environment by performing certain actions and observing the rewards or penalties it receives for those actions. The goal is to maximize the cumulative reward over a long period of time. Reinforcement learning has been used in game playing, robotics, and self-driving cars. Some reinforcement learning algorithms are: Temporal Difference (TD), Q-Learning, Deep Adversarial Networks etc.

Examples of Supervised Learning:

- **Linear Regression:** It is a supervised learning algorithm used to predict a continuous outcome variable based on one or more predictor variables.

- **Nearest Neighbor:** It is a simple classification algorithm that assigns a class label to an input data point based on the class labels of the k-nearest data points in the training set.

- **Gaussian Naive Bayes:** It is a classification algorithm that uses Bayes' theorem to predict the probability of each class given the input data.

- **Decision Trees:** It are a type of supervised learning algorithm used for both classification and regression tasks. They build a model in the form of a tree structure that predicts the class label or output value of an input data point based on its features.

- **Support Vector Machine (SVM):** It is a classification algorithm that finds the optimal hyperplane that separates the different classes in the input data. It is used for both linear and nonlinear classification tasks.

- **Random Forest:** It is an ensemble learning method that constructs multiple decision trees and combines their outputs to make a prediction. It is used for both classification and regression tasks.

Examples of Unsupervised Learning:

- **K-Means Clustering:** A popular clustering algorithm used to group data points into k clusters based on their similarity.

- **DBSCAN:** A density-based clustering algorithm that groups together data points that are close to each other in a high-density region.
- **BIRCH:** A hierarchical clustering algorithm that constructs a tree-like structure to represent the data.
- **Hierarchical Clustering:** A clustering algorithm that groups data points into nested clusters based on their similarity.
- **Principal Component Analysis (PCA):** A dimensionality reduction technique used to reduce the number of features in a dataset while retaining most of the information.
- **Anomaly Detection:** An unsupervised learning task that involves identifying unusual patterns or outliers in a dataset.
- **Association Rule Mining:** A data mining technique used to discover relationships or associations between different variables in a dataset.

Some reinforcement learning algorithms:

- **Q-Learning:** Q-Learning is a model-free reinforcement learning algorithm that is used to learn the optimal policy of an agent based on the values of state-action pairs.
- **Deep Q-Network (DQN):** DQN is an extension of Q-Learning that uses deep neural networks to estimate the Q-values of state-action pairs.
- **SARSA:** SARSA is an on-policy reinforcement learning algorithm that learns the optimal policy by updating the Q-values of state-action pairs based on the current policy.
- **Actor-Critic:** Actor-Critic is a model-based reinforcement learning algorithm that learns the optimal policy by maintaining both a policy function (actor) and a value function (critic).
- **Monte Carlo Tree Search (MCTS):** MCTS is a model-based reinforcement learning algorithm that uses a tree search to find the optimal policy by sampling possible actions and outcomes.

Inductive learning (Decision Tree):

Inductive learning is a type of machine learning in which the system learns to generalize rules from specific examples. Decision tree is a popular algorithm used for inductive learning.

A decision tree is a tree-like model that is used to classify data based on a set of rules. The tree is constructed by recursively partitioning the data into smaller subsets based on the most relevant attribute, and then applying the same procedure to each subset. The result is a tree structure in which each internal node represents a test on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label.

The decision tree algorithm is a supervised learning method, which means that it requires labeled training data to learn from. The algorithm works by selecting the attribute that provides the most information gain (i.e., the most useful attribute for classification) and using it to split the data into two subsets. This process is repeated recursively on each subset until the tree is fully grown. At this point, the tree can be used to classify new, unseen data. Algorithms used in decision tree are: ID3, Gini Index, Chi-Square, Reduction in Variance

Two types of decision tree:

- **Categorical variable decision trees:** This kind of tree are used when the dependent variable is a categorical variable, which means that it can take on a limited number of possible values. In this type of decision tree, the splitting of nodes is based on the frequency of the different categories of the dependent variable.
- **Continuous variable decision trees:** This kind of tree are used when the dependent variable is a continuous variable, which means that it can take on any value within a certain range. In this type of decision tree, the splitting of nodes is based on the values of the independent variables, and the goal is to find the best split that maximizes the separation between the classes of the dependent variable.

Terminologies of Decision Tree:

- **Root node:** The topmost node in a decision tree that represents the entire population or sample being analyzed.
- **Splitting:** The process of dividing a node into two or more sub-nodes based on a certain criterion.
- **Decision node:** A node in a decision tree that has one or more branches.
- **Terminal node (leaf node):** The end of a decision tree where the outcome of a decision is predicted.
- **Pruning:** The process of removing branches from a tree that do not provide any useful information.
- **Branch:** The lines connecting nodes that represent the decision made for a given attribute or feature.
- **Parent node:** A node in a decision tree that is split into two or more sub-nodes.
- **Child node:** A sub-node in a decision tree that is created as a result of splitting a parent node.

Algorithms used in Decision Tree:

Some common algorithms used in decision tree construction. Here is a brief description of each:

- **ID3 (Iterative Dichotomiser 3):** This is one of the oldest and most popular algorithms used for decision tree construction. It uses the concept of entropy to measure the randomness of the data.
- **Gini Index:** This algorithm uses the Gini impurity measure to determine the best split at each node. The Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it were randomly labeled according to the distribution of labels in the subset.
- **Chi-Square:** This algorithm uses the Chi-square test to determine the best split at each node. The Chi-square test measures the statistical significance of the association between the attribute and the class.

- **Reduction in Variance (CART):** This algorithm uses the reduction in variance as the splitting criterion. The idea is to choose the split that results in the greatest reduction in variance of the target variable.

Statistical-based learning (Naive Bayes):

Statistical-based learning is a type of machine learning that uses statistical methods to make predictions based on data. One example of a statistical-based learning model is the Naive Bayes model. The Naive Bayes model is a probabilistic model that uses Bayes' theorem to make predictions. It is commonly used in natural language processing and text classification tasks, such as spam filtering and sentiment analysis.

The Naive Bayes model assumes that the features used for classification are independent of each other, which is why it is called "naive." It calculates the probability of a particular class based on the probabilities of each feature given that class. The class with the highest probability is then chosen as the predicted class.

The Naive Bayes model works by first training on a dataset of labeled examples. During training, the model calculates the probability of each feature given each class. When a new, unlabeled example is presented to the model, it uses these probabilities to calculate the probability of each class given the features in the example. The class with the highest probability is then chosen as the predicted class.

The Naive Bayes model can be used with both discrete and continuous data, making it a versatile tool for many different types of problems. However, its assumption of independence between features can sometimes be a limitation, as it may not hold true in all cases.

Fuzzy learning:

Fuzzy learning is a type of machine learning that uses fuzzy logic to handle uncertain or imprecise data. Fuzzy logic is a mathematical framework for dealing with uncertainty, which is commonly used in control systems, expert systems, and other applications that require decision-making in uncertain environments.

In fuzzy learning, data is represented using fuzzy sets, which allow for partial membership and overlapping membership. This is in contrast to traditional binary logic, which assumes that data is either true or false. Fuzzy learning algorithms are commonly used for tasks such as classification, clustering, and prediction. They work by analyzing the relationships between the input variables and the output variable to identify patterns and make predictions.

One example of a fuzzy learning algorithm is the fuzzy c-means clustering algorithm, which is used for unsupervised clustering of data. Another example is the adaptive network-based fuzzy inference system (ANFIS), which is a type of neural network that uses fuzzy logic to make decisions. Fuzzy learning is a powerful tool for handling uncertain and imprecise data, and is particularly useful in applications such as image recognition, speech recognition, and natural language processing.

Fuzzy logic:

Fuzzy logic is a type of mathematical logic that allows for reasoning with degrees of truth instead of the usual true or false (1 or 0) values used in classical or boolean logic. In fuzzy

logic, the truth of a statement is represented as a value between 0 and 1, which represents the degree to which the statement is true. Fuzzy logic is particularly useful in situations where the boundaries between categories are not clear-cut or well-defined, and where there is a high degree of uncertainty or imprecision. Fuzzy logic has applications in a wide range of fields, including artificial intelligence, control systems, decision-making, and data analysis. Fuzzy logic is often represented using if-then rules. The rules are usually expressed in the form of "if condition A is true, then conclusion B is true to degree C," where A and B are linguistic variables and C is a degree of truth or membership. The rules can be combined using fuzzy inference to make decisions or draw conclusions based on uncertain or vague information.

Fuzzy Inference System:

FIS is a type of artificial intelligence system that uses fuzzy logic to make decisions based on uncertain or ambiguous data. It is a type of expert system that can make decisions based on imprecise or incomplete information, which makes it particularly useful in situations where traditional binary logic may not be effective.

A FIS consists of four main components:

- **Fuzzifier:** This component converts crisp inputs into fuzzy inputs, which allows the system to deal with imprecise or uncertain data.
- **Inference engine:** This component uses the fuzzy rules to determine the appropriate output based on the fuzzy inputs.
- **Rule base:** This component contains the set of fuzzy rules that are used to make decisions based on the fuzzy inputs.
- **De-fuzzifier:** This component converts the fuzzy output back into a crisp output that can be used in the real world.

Fuzzy inference systems can be used in a wide range of applications, including control systems, decision-making systems, and pattern recognition systems.

Functional block of Fuzzy interface system:

- **Rule Base:** Contains fuzzy IF-THEN rules.
- **Database:** Shows the membership functions of fuzzy sets used in fuzzy rules.
- **Decision-making Unit:** Performs operation on rules.
- **Fuzzification Interface Unit:** Transforms the crisp quantities into fuzzy quantities.
- **Defuzzification Interface Unit:** Transforms the fuzzy quantities into crisp quantities.

Fuzzy inference methods:

Fuzzy inference methods are used in fuzzy logic-based systems to make decisions or to predict outcomes based on imprecise or uncertain data. Some common fuzzy inference methods are:

- **Mamdani method:** This is a widely used fuzzy inference method, in which fuzzy rules are defined based on linguistic variables and their membership functions. The output is generated by computing the weighted average of the rules.
- **Sugeno method:** This method is a generalization of the Mamdani method, in which the output is generated by a linear function of the inputs. This method is often used in cases where the output is a numerical value, rather than a linguistic value.

- **Tsukamoto method:** This method uses a different approach to generate the output, based on the degree of match between the input variables and the fuzzy rules. It is often used in cases where the inputs are continuous variables.
- **Larsen method:** This method is similar to the Mamdani method, but the output is generated by computing the product of the rule strength and the output value, rather than the weighted average. This method is often used in cases where the rules are mutually exclusive.

Genetic Algorithm:

Genetic Algorithm (GA) is a metaheuristic optimization algorithm based on the principles of natural selection and genetics. It is inspired by the process of evolution in biology, and it is commonly used to solve optimization problems.

The basic idea of a genetic algorithm is to generate a population of potential solutions to a problem, and then to use selection, crossover, and mutation operations to evolve the population over time. In each generation, the individuals in the population are evaluated according to a fitness function that measures how well they solve the problem. The fittest individuals are then selected to produce the next generation, and the process is repeated until a satisfactory solution is found.

The key components of a genetic algorithm are:

- **Population:** a set of potential solutions to the problem that is being solved.
- **Fitness function:** a measure of how well each individual in the population solves the problem.
- **Selection:** a process for selecting the fittest individuals in the population to reproduce and produce the next generation.
- **Crossover:** a process for combining the genetic material of two individuals to create new offspring.
- **Mutation:** a process for introducing random changes into the genetic material of an individual.

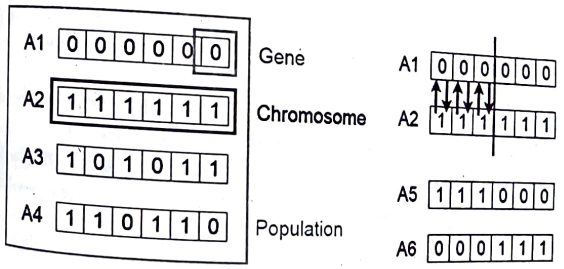


Figure: Gene, Chromosome, Population generation

Terminologies in GA:

- **Population:** A group of individuals that represent possible solutions to a problem.
- **Chromosome:** A single solution represented as a set of genes. In genetic algorithm, a chromosome is typically represented as a string of binary digits.
- **Gene:** A single element of a chromosome that represents a particular attribute of the solution.
- **Allele:** The value of a gene, representing a particular feature of a solution.
- **Fitness function:** A function that evaluates the fitness of an individual in the population. It is used to measure how well an individual solves the problem at hand.
- **Genetic operators:** These are operators used to modify the genetic material of the population, which includes selection, crossover and mutation.
- **Selection:** The process of selecting individuals from the population for breeding new offspring. Selection is typically based on the fitness of the individuals.

These terminologies are used in genetic algorithm to create a population of solutions, select the best solutions for reproduction, and then produce new offspring solutions by combining the genetic material of the selected individuals through crossover and mutation. The fitness function is used to evaluate the quality of each solution, and the genetic operators are used to generate new solutions that are potentially better than the original population. The process is repeated iteratively until a satisfactory solution is obtained.

Why to use Genetic Algorithm?

- **Global optimization:** GAs can find the global optimal solution instead of just a local optimal solution. This is because GAs can explore a large search space and find the best solution.
- **Robustness:** GAs are robust to noisy and incomplete data. They can handle missing or noisy data and still find a good solution.
- **Parallel processing:** GAs can be easily parallelized, allowing for faster optimization.
- **No assumptions:** GAs do not make any assumptions about the underlying data or model. This makes them suitable for a wide range of problems.
- **Versatility:** GAs can be applied to a wide range of problems, including optimization, search, and machine learning.

Genetic Algorithm (Genetic Algorithm Operators, Genetic Algorithm Encoding, Selection Algorithms, Fitness function, and Genetic Algorithm Parameters)

Genetic Algorithm Operators:

GA uses three main operators - selection, crossover, and mutation - to create new solutions from the current population. Selection chooses the best individuals from the current population to be used in the next generation. Crossover creates new solutions by combining the characteristics of two or more solutions from the current population. Mutation adds random changes to the solutions to explore new areas of the search space.

Genetic Algorithm Encoding:

GA encodes potential solutions as strings of binary digits or other data types. The encoding method is important because it determines how solutions are represented and manipulated by the genetic algorithm operators.

Selection Algorithms:

Selection algorithms choose the best individuals from the current population to be used in the next generation. Common selection algorithms include tournament selection, roulette wheel selection, and rank-based selection Some of the commonly used selection algorithms in genetic algorithms are:

- **Tournament Selection:** In tournament selection, a subset of individuals is selected from the population at random, and the fittest individual among them is chosen for reproduction. This process is repeated until the required number of individuals for the next generation is selected.
- **Roulette Wheel Selection:** In roulette wheel selection, each individual in the population is assigned a probability of selection proportional to its fitness value. The individuals with higher fitness have a higher chance of being selected for reproduction.
- **Rank-Based Selection:** In rank-based selection, the individuals in the population are sorted based on their fitness values, and each individual is assigned a selection probability based on its rank. The fittest individual has the highest probability of selection, and the probability decreases with decreasing fitness ranks.

Fitness Function:

In genetic algorithms, the fitness function is a function that measures the "goodness" of a particular solution or candidate. It's used to evaluate how well each individual in the population solves the problem at hand. The fitness function assigns a fitness score to each individual in the population, which is used to determine which individuals will be selected for reproduction and which will be discarded.

The goal of the genetic algorithm is to find the individual or set of individuals with the highest fitness score. The fitness function is problem-specific, meaning that it needs to be designed according to the requirements of the problem being solved. It can be as simple or as complex as required, depending on the complexity of the problem.

For example, in a optimization problem where the goal is to minimize a certain cost, the fitness function could be defined as the inverse of the cost, so that individuals with lower costs have higher fitness scores. In a classification problem, the fitness function could be based on the accuracy of the classification.

Genetic Algorithm Parameters:

GA requires several parameters to be specified before running the algorithm, including population size, number of generations, selection rate, crossover rate, mutation rate, and others. These parameters are important because they can have a significant impact on the performance and effectiveness of the algorithm. Choosing the right values for these parameters can be a challenging task, and it often requires trial and error.

7.6 NEURAL NETWORKS

Neural Networks:

Neural networks are a type of machine learning algorithm that are modeled after the structure and function of the human brain. They consist of interconnected processing nodes or "neurons" that work together to learn patterns and relationships from data. Each neuron receives input signals, processes them, and generates output signals that are transmitted to other neurons in the network. Neural networks are trained using labeled data and adjust the strength of the connections between neurons to improve their ability to classify or predict new data. They can be used for a wide range of applications, including image and speech recognition, natural language processing, and predictive analytics. Types of NN:

- **Feed-forward Neural Networks (FFNN):** These are the most basic type of neural networks, in which information flows in one direction, from the input layer, through the hidden layers, and to the output layer. Two types:
  - **Single layer feedforward neural networks:** It is also known as perceptrons, have a single layer of artificial neurons that take input and produce output based on the provided input. These networks have a simple architecture that can be trained using a variety of supervised learning algorithms.

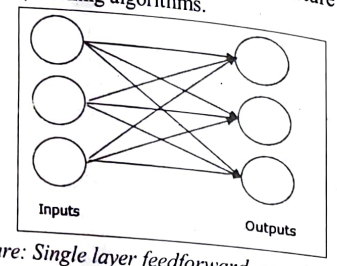


Figure: Single layer feedforward neural network

  - **Multi-layer feedforward neural networks:** It has one or more hidden layers between the input and output layers. Each neuron in a hidden layer receives input from the previous layer and produces output that serves as input for the next layer. These networks are capable of learning more complex functions than single layer networks, but they can be more difficult to train and require more computational resources.

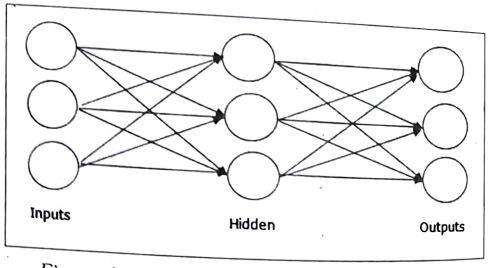


Figure: Multi-layer feedforward neural networks

- **Recurrent Neural Networks (RNN):** In RNNs, the output from one time step is used as input for the next time step. This allows RNNs to handle sequential data such as time series, text, and speech. Two types:

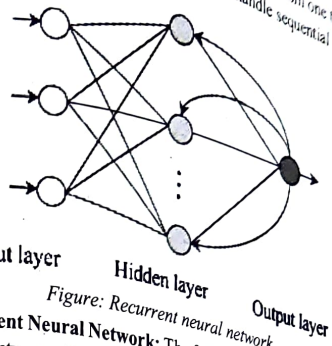


Figure: Recurrent neural network

  - **Fully Recurrent Neural Network:** The fully recurrent neural network has feedback connections between all neurons.

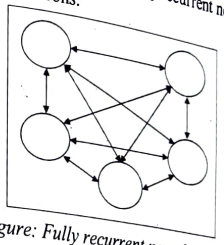


Figure: Fully recurrent neural network

  - **Jordan Network:** Jordan Network is a type of recurrent neural network with output units receiving feedback from their own output as well as the hidden layer.

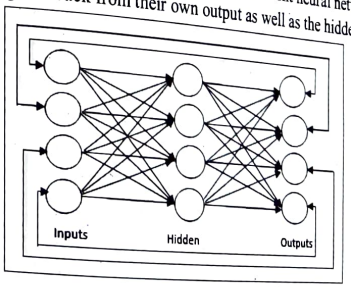


Figure: Jordan Network

- **Convolutional Neural Networks (CNN):** CNNs are commonly used for image recognition tasks. They have a specialized architecture that makes them particularly effective at identifying patterns in images.

- **Self-Organizing Maps (SOM):** SOMs are used for clustering and visualization tasks. They use unsupervised learning to identify patterns in the input data and map them onto a low-dimensional space.
- **Deep Belief Networks (DBN):** DBNs are composed of multiple layers of Restricted Boltzmann Machines (RBMs), which are unsupervised learning algorithms. They are used for tasks such as image and speech recognition.
- **Autoencoders:** Autoencoders are neural networks that are trained to reconstruct their input data. They are used for tasks such as data compression, feature extraction, and denoising.
- **Generative Adversarial Networks (GAN):** GANs are composed of two neural networks that work together to generate new data that is similar to the training data. They are used for tasks such as image and text generation.

Features of NN:

Some of the key features of neural networks include:

- **Ability to learn from data:** Neural networks can learn and improve their performance over time by adjusting the weights of their connections based on the input data.
- **Non-linearity:** Neural networks can model complex, non-linear relationships between inputs and outputs, making them suitable for a wide range of applications.
- **Parallel processing:** Neural networks can perform many computations in parallel, which can make them faster than other types of algorithms for certain tasks.
- **Robustness:** Neural networks can be highly robust to noisy or incomplete data, and can often continue to provide useful output even when presented with incomplete or degraded inputs.
- **Adaptability:** Neural networks can adapt to changes in the input data or the task at hand, which can make them more flexible than traditional, rule-based algorithms.
- **Generalization:** Neural networks can often generalize well to new, unseen data, which can make them useful in a wide range of real-world applications.

Biological Neural Network:

BNN stands for Biological Neural Network, which is a network of interconnected neurons that form the biological basis for the functioning of the nervous system. The BNN is a complex network of neurons that work together to process and transmit information in the form of electrical and chemical signals. The structure of the BNN is highly organized, with different regions of the brain responsible for specific functions. The BNN is capable of learning, adapting, and changing based on experience, which makes it a powerful tool for processing complex information and making decisions. Artificial neural networks (ANNs) are modeled on the structure and function of the BNN to perform various tasks in machine learning and artificial intelligence.

Artificial Neural Network:

Artificial Neural Network (ANN) is a type of machine learning algorithm that is modeled after the structure and function of biological neural networks. It is an interconnected group of nodes, similar to the vast network of neurons in a brain, that is capable of modeling and processing complex patterns in data.

Artificial Neural Networks (ANNs) consist of three main types of layers: input layer, hidden layer, and output layer.

- The input layer is where the data is initially fed into the neural network. The input layer is typically made up of a set of input neurons, with each neuron representing a specific feature or attribute of the input data.
- The hidden layer is where most of the processing in the neural network occurs. It is called "hidden" because its output is not directly observable. A hidden layer can consist of one or more layers of neurons, and the number of hidden layers and the number of neurons in each layer can vary depending on the complexity of the problem and the size of the input data.
- The output layer is where the final output of the neural network is generated. It typically consists of one or more output neurons, with each neuron representing a specific class or category.

Application areas of ANN:

- **Image and speech recognition:** ANNs are used to identify objects in images and recognize speech patterns.
- **Natural language processing:** ANNs are used to analyze and generate human language.
- **Robotics:** ANNs can be used to control robot movements and behavior.
- **Health care:** ANNs can be used to analyze medical data and assist in diagnosis and treatment decisions.
- **Gaming:** ANNs can be used to create intelligent game-playing agents that can learn and adapt to new challenges.
- **Fraud detection:** ANNs can be used to detect fraudulent activity in financial transactions and other areas.

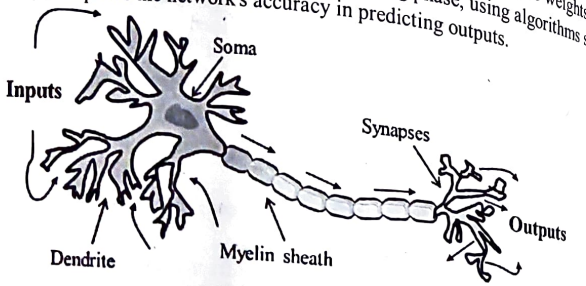
BNN vs ANN

Biological Neural Networks (BNN)	Artificial Neural Networks (ANN)
Modeled after the human brain	Modeled after BNN, but not a direct copy.
Consists of neurons, dendrites, axons, and synapses	Consists of input, output, and hidden layers of nodes or neurons
Operates in parallel and is capable of self-organization	Operates sequentially and requires programming or configuration
Performs tasks that are hard for computers, such as pattern recognition, real-time processing, and decision making	Performs tasks such as prediction, classification, and data analysis

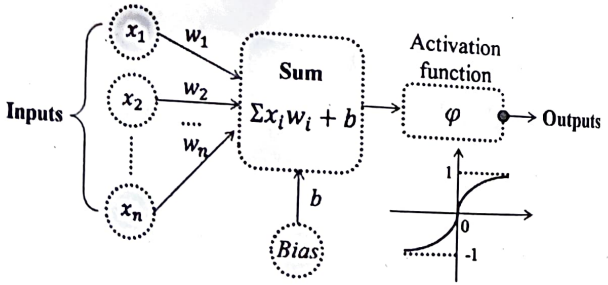
Biological Neural Networks (BNN)	Artificial Neural Networks (ANN)
Limited computational power	High computational power
Not scalable	Scalable
Can self-repair and self-replicate	Cannot self-repair or self-replicate
Can operate in noisy or uncertain environments	Can operate in noisy or uncertain environments, but not as well as BNN
Can learn from few examples	Requires a large amount of labeled data for training
Can work with imprecise or incomplete data	Requires precise and complete data
Not fully understood	Well-understood and widely used in machine learning

Mathematical model of ANN:

Artificial neural networks (ANNs) are based on mathematical models that represent a simplified version of biological neurons and synapses. The mathematical model of an artificial neuron takes input values and combines them with weights to produce an output, which is passed through an activation function to produce the neuron's final output. The weights of the connections between neurons are adjusted during the training phase, using algorithms such as backpropagation, to improve the network's accuracy in predicting outputs.



(a) Biological neuron



(b) Artificial neuron

Figure: Biological neuron vs Artificial neuron

ANN Vs ANN terminology comparison

BNN		ANN
Soma	⟶	Node
Dendrites	⟶	Input
Synapse	⟶	Weights or interconnections
Axon	⟶	Output

McCulloch-Pitts Neuron:

The McCulloch-Pitts neuron is a simple model of a neuron in the brain proposed by Warren McCulloch and Walter Pitts in 1943. It is a binary threshold unit that takes one or more binary inputs and produces a binary output. The output of the neuron is based on whether the weighted sum of the inputs is greater than or equal to a certain threshold value. It has been an important concept in the development of artificial neural networks.

The McCulloch-Pitts neuron can be represented as:

$x1$	$x2$	$x3$	...	xn
$w1$	$w2$	$w3$	...	wn

$$y = f(w1x1 + w2x2 + w3x3 + ... + wnxn)$$

where xi is the i-th input, wi is the weight associated with that input, n is the total number of inputs, y is the output of the neuron, and f() is the activation function which is a threshold function.

The McCulloch-Pitts neuron is a binary threshold unit which means that its output can be only one of two states: 0 or 1. The activation function is a threshold function that compares the weighted sum of the inputs to a threshold value. If the weighted sum is greater than or equal to the threshold, the output of the neuron is 1, otherwise it is 0.

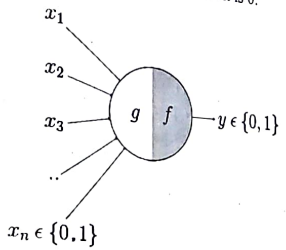


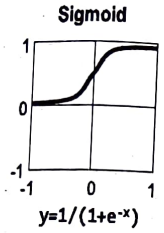
Figure: McCulloch-Pitts neuron

Activation functions and its types:

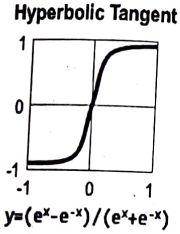
In a neural network, the activation function is a mathematical function that is applied to the weighted sum of the input values plus the bias of the neuron, and produces an output signal that is sent to the next layer in the network. The activation function introduces nonlinearity into the network and helps the network to learn complex relationships between the input and

output variables. Activation function is also known as transfer function. It is used to determine the output of neural network like yes or no. It maps the resulting values in between 0 to 1 or -1 to 1.

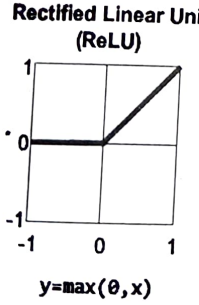
- **Linear Activation function:** A linear activation function produces an output that is proportional to the input.
- **Non-linear Activation function:** A non-linear activation function produces an output that is not proportional to the input. Some of Non-linear Activation functions are:
  - **Sigmoid:** produces an output that ranges from 0 to 1, which is useful for binary classification tasks.



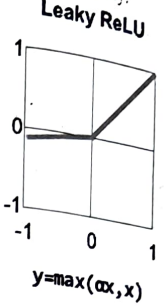
  - **Tanh:** produces an output that ranges from -1 to 1, which is useful for tasks that require negative values.



  - **ReLU (Rectified Linear Unit):** produces an output of 0 for negative inputs and a proportional/ infinity output for positive inputs, which is widely used due to its simplicity and computational efficiency.



  - **Leaky ReLU:** similar to ReLU, but allows for small negative outputs to avoid "dead" neurons. Range is -infinity to infinity.



Perceptron:

Perceptron is a type of artificial neural network consisting of a single layer of input nodes connected directly to a single layer of output nodes, with each output node corresponding to a class label.

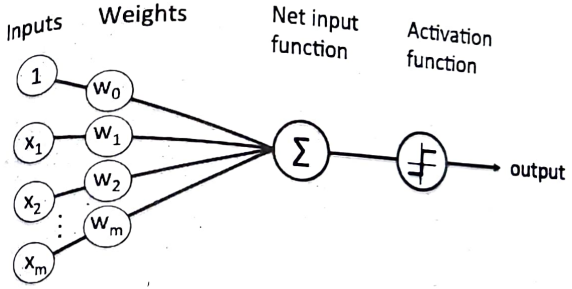


Figure: Perceptron

A perceptron consists:

- **Input nodes:** These are the nodes that receive input values from the external world or from other layers of a neural network.
- **Weights:** Each input node is associated with a weight, which represents the strength or importance of the connection between the input node and the perceptron.
- **Bias:** The bias is a constant value that is added to the weighted sum of the inputs to the perceptron, allowing the perceptron to adjust the decision boundary.
- **Activation function:** The activation function determines the output of the perceptron based on the weighted sum of the inputs and the bias. It introduces non-linearity to the output and helps to model complex relationships between the inputs and the output. Some common activation functions include the sigmoid function, ReLU function, and tanh function.

There are two types of perceptron:

- **Single-layer perceptron:** a type of feedforward neural network that can only represent linearly separable functions.

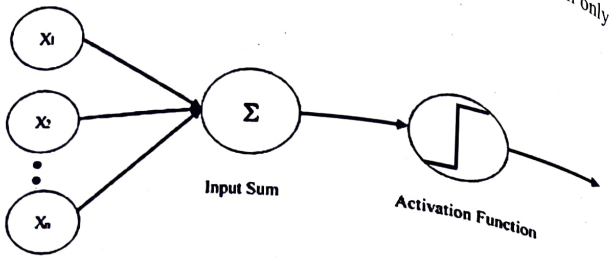


Figure: Single-layer perceptron

- **Multi-layer perceptron:** a type of feedforward neural network that can represent complex non-linear functions, with one or more hidden layers of nodes between the input and output layers.

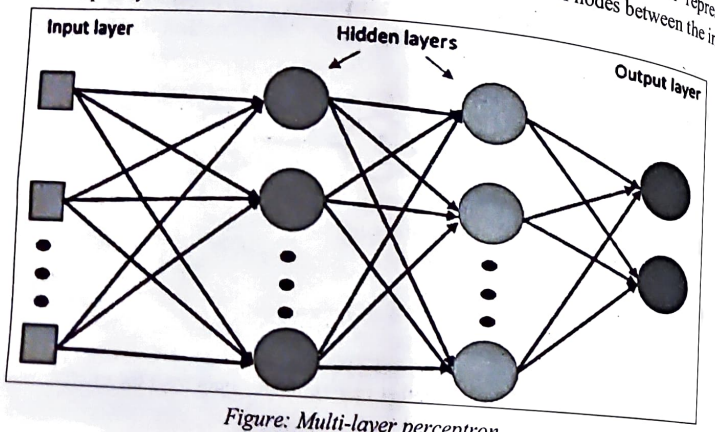


Figure: Multi-layer perceptron

Characteristics of Perceptron:

- Perceptron is a type of neural network that is trained to classify input data by learning the optimal set of weights and biases.
- It has a single layer of input nodes, where each node corresponds to a feature of the input data.
- Perceptron applies a set of weights and biases to the input data and passes the result through an activation function to produce an output.
- The activation function used in a perceptron can be linear or non-linear, and determines the type of decision boundary that the network can learn.

- Perceptron can learn to classify linearly separable data, but is limited in its ability to learn more complex decision boundaries.

Perceptron learning rule:

The perceptron learning rule is an algorithm that updates the weights of the input nodes of a perceptron based on the error of the output. The steps of the algorithm are:

- Initialize the weights and bias to small random values.
- For each training example:
  - Calculate the output of the perceptron.
  - Update the weights and bias based on the error.
- Repeat step 2 until the error is below a certain threshold or a maximum number of iterations is reached.

The weight update formula is:

$\Delta w = \alpha(y - \hat{y})x$

where Δw is the change in weight, α is the learning rate, y is the target output, $\hat{y}$ is the actual output, and x is the input. The bias update formula is similar, with x being replaced by 1. The learning rate determines the step size of the weight update and is typically set to a small value, such as 0.1.

Gradient Descent:

- A technique used to minimize the error between the predicted output of a neural network and the actual output.
- Adjusts the weights of the network in the opposite direction of the gradient of the error function.
- Can be batch or stochastic (updating weights after each training example).

The Delta Rule:

- A learning rule used to adjust the weights of a single-layer perceptron.
- Compares the output of the perceptron with the actual output to calculate the error, and then adjusts the weights based on the error.
- Uses the derivative of the activation function to determine how much to adjust the weights.

Hebbian Learning:

- A type of unsupervised learning used to strengthen connections between neurons that are active at the same time.
- Based on the idea that "neurons that fire together, wire together."
- Can be used to detect patterns or correlations in input data without explicit labeling or supervision.

Adaline:

Adaline (Adaptive Linear Neuron) is a type of neural network that uses a linear activation function and a weight update rule that is based on the least mean square error (LMSE) criterion.

It was introduced by Bernard Widrow and his colleagues in 1960 as an improvement over the perceptron algorithm. The Adaline network has the following properties:

- It is a single-layer feedforward neural network with a linear activation function.
- It uses the LMSE criterion to adjust the weights of the input features.
- It can be used for both classification and regression tasks.
- It can handle both binary and continuous input and output values.
- It is a supervised learning algorithm that requires labeled training data.
- It can converge to a solution even if the input data is not linearly separable.
- It has a single output unit that produces a continuous output value.
- It can be trained using batch or online learning.
- It can be extended to handle multiclass classification problems using a one-vs-all approach.

Madaline:

Madaline (Multiple Adaline) is an artificial neural network that consists of multiple Adaline neurons organized in layers, where each layer is connected to the previous and next layer. Unlike Adaline, which has only one output node, Madaline can have multiple output nodes, which enables it to perform multi-class classification. The weights in Madaline are updated using a modified version of the delta rule, which allows it to converge to the optimal solution more quickly. Here are some properties of Madaline:

- Madaline stands for "Multiple ADALINE".
- It is a multilayer neural network, which consists of multiple Adaline networks.
- Madaline can learn and classify linearly separable patterns as well as some nonlinear patterns.
- In Madaline, the outputs of all Adaline networks are combined using a decision logic to produce a final output.
- Madaline can be trained using the Delta rule or the backpropagation algorithm.

Multilayer Perceptron Neural Networks:

- Also known as feedforward neural networks.
- Consists of multiple layers of perceptron, including input, hidden, and output layers.
- Can be used for both classification and regression problems.
- Activation functions can be linear or non-linear, such as the sigmoid or ReLU functions.
- Trained using backpropagation algorithm.

Backpropagation Algorithm:

- A supervised learning algorithm used for training feedforward neural networks.
- Used to adjust the weights of the connections between the neurons in the network.
- Works by propagating the errors backwards from the output layer to the input layer, and adjusting the weights accordingly.

- Helps to minimize the difference between the predicted outputs and the actual outputs.

Hopfield Neural Network:

- A type of recurrent neural network.
- Consists of a single layer of neurons that are fully connected to each other.
- Can be used for pattern recognition and associative memory tasks.
- Uses the Hebbian learning rule to adjust the weights between neurons based on the correlation between their activities.
- Attractor states are stable states that the network tends to converge to over time, depending on the initial inputs.
- Two types:
  - **Continuous Hopfield network:**
    - Nodes are connected to all other nodes except itself, and the connections are symmetric.
    - Nodes use continuous activation functions such as sigmoid or hyperbolic tangent.
    - The network is designed to find a stable equilibrium state or energy minimum, where the output of the network is the same as the input.
    - The network is trained using the Hebbian learning rule, where the weights between two nodes are strengthened if they activate together and weakened if they do not.
  - **Discrete Hopfield network:**
    - Nodes are also connected to all other nodes except itself, and the connections are symmetric.
    - Nodes use binary activation functions, such as step functions.
    - The network is designed to store and retrieve patterns, where the output of the network converges to the stored pattern when given a noisy or incomplete input.
    - The network is trained using the Hebbian learning rule, where the weights between two nodes are set to the product of the input and the output of the two nodes.

Q. 208

ANSWER: Negation

The truth value of ¬P is opposite of the truth value of P.

Example of the Negation:

- *P: Today is Saturday*
- *¬P: Today is not Saturday*
- *¬P will be true on all other days than Saturday*

Q. 209

ANSWER: Conjunction

The proposition P∧Q is true when both P and Q are true and false otherwise.

Example of the Conjunction:

P: Today is Saturday

- *Q: Today is Rainy*
- *P∧Q: Today is Rainy and Saturday*
- *P∧Q will be true on Rainy Saturdays and false on non-Saturdays or non-Rainy days*

Q. 210

ANSWER: Disjunction

The proposition P∨Q is true when either P or Q is true and false when both P and Q are false.

Example of Disjunction:

P: Today is Saturday

- *Q: Today is Rainy*
- *P∨Q: Today is Rainy or Saturday*
- *P∨Q will be true on any Saturday or any day that is Rainy*

Q. 212

ANSWER: Exclusive OR

The proposition P⊕Q is true when exactly one of P and Q is true otherwise false.

Example of Exclusive OR:

P: Today is Saturday

- *Q: Today is Rainy*
- *P⊕Q will be true on any Saturday which is not rainy or any rainy day that is not Saturday.*

Q. 213

ANSWER: Conditional Statement

Example of conditional statement

P: You get 50% on the NEC Licensing Exam

Q: You get a C

P→Q: If you get 50% on the NEC Licensing exam, then you will get a C

- *If you do not get 50%, you may get a C Or you may not get a C, which depends on other factors.*
- *There won't be problem if you get 50% and a C*
- *If 50% is scored but you do not get a C then you may feel disappointed.*

Q. 214

ANSWER: All of above

Some more are:

"q if p",	*"q when p",*	*"a necessary condition for p is q",*
"q unless ¬p",		
"p implies q",	*"p only if q",*	*"a sufficient condition for q is p",*
"q whenever p",	*"q is necessary for p",*	*"q follows from p" etc. [10]*

Q. 218

ANSWER: Bi-conditional statement

The bi-conditional statement P↔Q is true when P and Q have same truth values and is false otherwise

Example of bi-conditional statement:

P: You can take the ride

Q: You buy a trouser

P↔Q: You can take the ride if and only if you buy a trouser

Q. 223

ANSWER: Object, predicate

The Object is what (or whom) the sentence is about and the Predicate tells something about the Object. Example: A sentence "Ram {runs}". The Object is Ram and the predicate is runs.

Predicate is a verb phrase template that describes a property of objects, or a relation among objects represented by the variables.

Q. 224

ANSWER: The predicate is "is red"

- *Predicate is "is red" because it describes property.*
- *Predicates are given names; Let 'B' is name for predicate "is red".*
- *Sentence is represented as "B(x)", read as "x is red";*
- *Symbol "x" represents an arbitrary Object.*

Q. 225

ANSWER: Predicate

A predicate becomes a proposition when specific values are assigned to the variables.

Example: She lives in the city.

Predicate: P (x, y): x lives in y.

P (Khusboo, New York) is a proposition: Khusboo lives in New York.

Q. 229

ANSWER: Existential

Existential quantifier: read for "some" statement that is true or false.

Q. 230

ANSWER: Universal

Universal quantifier: read for "all", "each", "every".

Q. 232

ANSWER: Both A and B

Other inference rules are:

Hypothetical syllogism, Disjunctive syllogism, Addition, Simplification, Resolution

Q. 233

ANSWER: Modus ponens

Notation:

$$\frac{P \quad P \rightarrow Q}{\therefore Q}$$

Example:

Statement-1: "If I am hungry then I go to restaurant" ==> $P \rightarrow Q$

Statement-2: ""I am hungry" ==> P"

Conclusion: "I go to restaurant." ==> Q.

Hence, we can say that, if $P \rightarrow Q$ is true and P is true then Q will be true.

Q. 234

ANSWER: Modus tollens

Notation:

$$\frac{\neg Q \quad P \rightarrow Q}{\therefore \neg P}$$

Example:

Statement-1: "If I am hungry then I go to restaurant" ==> $P \rightarrow Q$

Statement-2: "I do not go to the restaurant."==> $\sim Q$

Conclusion: Which infers that "I am not hungry" => $\sim P$

Q. 235

ANSWER: Hypothetical syllogism

Notation:

$$\frac{P \rightarrow Q \quad Q \rightarrow R}{\therefore P \rightarrow R}$$

Example:

Statement-1: If you have my car key then you can unlock my car. $P \rightarrow Q$

Statement-2: If you can unlock my car then you can take my ATM card. $Q \rightarrow R$

Conclusion: If you have my car key then you can take my ATM card. $P \rightarrow R$

Q. 236

ANSWER: Disjunctive syllogism

Notation:

$$\frac{P \vee Q \quad \neg P}{\therefore Q}$$

Example:

Statement-1: Today is Saturday or Sunday. ==>$P \vee Q$

Statement-2: Today is not Saturday. ==> $\neg P$

Conclusion: Today is Sunday. ==> Q

Q. 237

ANSWER: Addition

Notation:

$$\frac{P}{\therefore P \vee Q}$$

Example:

Statement-1: I have a kit kat. ==> P

Statement-2: I have a dairy milk. ==> Q

Conclusion: I have kit kat or dairy milk. ==> $(P \vee Q)$

Q. 299

ANSWER: Handling Ambiguity of Sentences

Lexical ambiguity: Treating the word "board" as noun or verb?

Syntactical ambiguity: "He lifted the beetle with red cap"→Did he use cap to lift the beetle or he lifted a beetle that had red cap?

Referential ambiguity: Rima went to Gauri. She said, "I am tired."→ Exactly who is tired?

Q. 335

ANSWER: Classification

Classification is a Supervised Learning task where output is having defined labels (discrete value). In above figure Output – "Purchased" has defined labels i.e., 0 or 1; 1 means the customer will purchase and 0 means that customer won't purchase. The goal here is to predict discrete values belonging to a particular class and evaluate on the basis of accuracy.

It can be either binary or multi class classification. In binary classification, model predicts either 0 or 1; yes or no but in case of multi class classification, model predicts more than one class.

Q. 336

ANSWER: Regression

Regression is a Supervised Learning task where output is having continuous value.

In above figure, Output – "Wind Speed" is not having any discrete value but is continuous in the particular range. The goal here is to predict a value as much closer to actual output value as our model can and then evaluation is done by calculating error value. The smaller the error the greater the accuracy of our regression model.

MULTIPLE CHOICE QUESTIONS

1. **Intelligence is defined as: _____**
   A. The capacity to acquire and apply knowledge.
   B. The faculty of thought and reason.
   C. Superior powers of mind.
   D. All of mentioned above
2. **Artificial Intelligence (AI) is the simulation of human intelligence by machines. AI has ability to _____**
   A. Solve Problems
   B. Act Rationally
   C. Act like Humans
   D. All of mentioned above
3. **The central principle of AI includes _____**
   A. Reasoning, knowledge, planning, learning and communication
   B. Perception and the ability to move and manipulate objects.
   C. It is the science and engineering of making intelligent machines, especially intelligent computer programs
   D. All of the mentioned above
4. **_____ is about AI**
   A. Making a machine Intelligent
   B. Putting your intelligence in Machine
   C. Programming on Machine with your Own Intelligence
   D. Playing a game on Computer
5. **What is Artificial Intelligence?**
   A. Artificial Intelligence is a field that aims to make humans more intelligent
   B. Artificial Intelligence is a field that aims to collect and mine the data
   C. Artificial Intelligence is a field that aims to develop intelligent machines
   D. Artificial Intelligence is a field that aims to improve the privacy and security
6. **_____ is the father of Artificial Intelligence**
   A. Alan Turing
   B. John McCarthy
   C. Lady ADA
   D. Charles Babbage
7. **If a machine can change its course of action based on the external environment on its own, the machine is called _____**
   A. Ideal
   B. Intelligent
   C. Both A and B
   D. Mobile
8. **_____ is the branch of Artificial Intelligence.**
   A. Network Architecture
   B. Full Stack Developer
   C. Machine Learning
   D. None of above
9. **_____ is the goal of an AI**
   A. To extract scientific causes
   B. To solve artificial problems
   C. To solve real-world problems
   D. To explain various sorts of intelligence
10. **_____ is an application of AI**
    A. It helps to exploit vulnerabilities to secure the firm
    B. Easy to create a website
    C. It helps to deploy applications on the cloud
    D. Language understanding and problem-solving (Text analytics and NLP)