Unit 1: Fundamentals of Artificial Intelligence

1.1 Definition, history, motivation, and need for Artificial Intelligence (AI)

- Definition: AI is the field of creating intelligent machines that can perform tasks that normally require human intelligence, this includes abilities like:
    - Learning: Acquiring knowledge and skills from data.
    - Reasoning: Using logic to solve problems and draw conclusions.
    - Problem-solving: Finding solutions to complex situations.
    - Perception: Understanding and interpreting sensory information (like images and sound).
    - Language understanding: Comprehending and generating human language.
- History:
    - Early Days (1950s-1970s)
        - AI was born, with researchers exploring symbolic reasoning, logic, and early neural networks.
        - Initial progress was promising.
        - Limitations in computing power led to an “AI winter” with decreased funding and interest.
    - Expert Systems (1980s)
        - AI made a comeback with systems designed to mimic human experts.
        - These systems were rigid and hard to update.
    - Machine Learning (1990s-Present)
        - Computing power increased, and AI shifted to learning from data.
        - This led to significant advances in image recognition and language processing.
    - Deep Learning (2010s-Present)

        - Deep learning further revolutionized AI.
        - It enabled even more complex tasks like machine translation, speech recognition, and game playing.
- Motivation: AI aims to create machines that can assist humans in various ways, from automating mundane tasks to providing expert advice in complex domains.
- Need for AI: AI is needed to address complex problems that humans cannot solve alone, such as climate change, disease, and poverty. It also has the potential to improve our lives by making us more efficient and productive.

1.2 Goals and contributions of AI to modern technology

- Goals:
    - Create machines that can think and reason like humans.
    - Create machines that can learn and adapt to new situations.
    - Create machines that can perceive and understand the world around them.
    - Create machines that can communicate and interact with humans in natural language.
- Contributions:
    - AI has already made significant contributions to modern technology, including:
        - Expert systems: Used to diagnose diseases, provide financial advice, and troubleshoot complex problems.
        - Natural language processing: Used for machine translation, speech recognition, and sentiment analysis.
        - Robotics: Used for manufacturing, healthcare, and exploration.

- Computer vision: Used for facial recognition, object detection, and self-driving cars.

1.3 Production systems:

- Characteristics of production systems:
  - They consist of a set of rules, or productions, that define how to transform a state into a new state.
  - They use a working memory to keep track of the current state of the system.
  - They use a control mechanism to decide which production to apply next.
- Example: An expert system for diagnosing diseases might have a production system that contains rules like “If the patient has a fever and a cough, then they may have pneumonia.”

1.4 Search space and techniques:

- Search space: The set of all possible states that a problem can be in.
- Search techniques: Algorithms used to search through the search space for a solution to a problem.
  - Hill Climbing: A greedy algorithm that always chooses the action that leads to the greatest immediate improvement in the objective function.

- Best First Search: A greedy algorithm that chooses the action that leads to the state with the lowest estimated cost to the goal.
- Heuristic Search Algorithm: A search algorithm that uses a heuristic function to estimate the cost of reaching the goal from a given state.

1.4.1.1 Hill Climbing:

Hill climbing is a simple search algorithm that explores the search space by moving from one state to another, always choosing the best neighbor of the current state. The algorithm terminates when it reaches a state that is better than all of its neighbors. This state is called a local maximum.

Hill climbing is a good way to find a local maximum, but it is not guaranteed to find the best possible solution, which is called the global maximum. This is because the algorithm can get stuck in a local maximum that is not the global maximum.

Types of Hill Climbing

There are many different types of hill climbing algorithms. Some of the most common include:

- Steepest-ascent hill climbing: This algorithm always chooses the best neighbor of the current state, regardless of how much better it is.
- Stochastic hill climbing: This algorithm chooses a neighbor at random, with the probability of choosing a neighbor being proportional to how much better it is than the

current state.

- Random-restart hill climbing: This algorithm conducts a series of hill climbing searches from randomly generated initial states.
- Simulated annealing: This algorithm is a variation of stochastic hill climbing in which the probability of choosing a worse neighbor decreases over time.

1.4.1.2 Best First Search:

- Best-first search is a search algorithm that explores a graph by expanding the most promising node chosen according to a specified rule.
- It is an informed search algorithm, meaning that it uses domain-specific knowledge (heuristics) to guide the search, making it more efficient than uninformed search methods.
- The core of best-first search is the evaluation function, f(n), which estimates the cost from the start node to a goal node, constrained to go through node n.
- The node with the lowest f(n) value is always expanded first.

Types of Best-First Search

- Greedy best-first search: Expands the node closest to the goal based purely on the heuristic, h(n). Can be fast but not always optimal.
- A* search: Combines the cost to reach the current node, g(n), and the heuristic, h(n). It is optimal if the heuristic is admissible (never overestimates).

1.4.1.3 Heuristic Search Algorithm

Heuristic Search Algorithm

- Heuristic search is a type of informed search that uses a heuristic function to estimate the cost of reaching the goal from a given state.
- Heuristic search algorithms are often more efficient than uninformed search algorithms because they can use the heuristic function to prune the search space.
- Some examples of heuristic search algorithms include A*, greedy best-first search, and hill climbing.

A* Search

- A* search is a type of heuristic search algorithm that is widely used in AI.
- A* search uses an evaluation function, f(n), to estimate the cost of reaching the goal from a given node.
- The evaluation function is defined as f(n) = g(n) + h(n), where g(n) is the cost of reaching the current node from the start node and h(n) is the heuristic function.
- A* search is optimal if the heuristic function is admissible, meaning that it never overestimates the cost of reaching the goal.

Greedy Best-First Search

- Greedy best-first search is a type of heuristic search algorithm that always expands the node that is closest to the goal according to the heuristic function.

- Greedy best-first search is not guaranteed to find the optimal solution, but it can be very efficient in practice.

Hill Climbing

- Hill climbing is a type of heuristic search algorithm that starts at an arbitrary point in the state space and repeatedly moves to a better neighbor until it reaches a peak.
- Hill climbing is not guaranteed to find the optimal solution, but it can be very effective at finding good solutions, especially when the state space is small.

1.4.2 A and AO Search Techniques:**

- *A:** A search algorithm that combines the best features of hill climbing and best-first search. It uses a heuristic function to estimate the cost to the goal, but it also takes into account the actual cost of the path so far.
- *AO:** A variant of A* that is designed for problems with uncertain costs. It uses a different heuristic function that takes into account the uncertainty in the cost estimates.

A* Search

- A* search is a type of informed search algorithm that is widely used in AI.
- A* search uses an evaluation function, f(n), to estimate the cost of reaching the goal from a given node.
- The evaluation function is defined as f(n) = g(n) + h(n), where g(n) is the cost of

reaching the current node from the start node and h(n) is the heuristic function.
- A* search is optimal if the heuristic function is admissible, meaning that it never overestimates the cost of reaching the goal.

AO* Search

- AO* search is a type of informed search algorithm that is used to solve problems with AND-OR graphs.
- An AND-OR graph is a graph where the nodes represent subproblems and the edges represent the relationship between the subproblems.
- AO* search uses an evaluation function, f(n), to estimate the cost of solving a subproblem.
- The evaluation function is defined as f(n) = g(n) + h(n), where g(n) is the cost of solving the current subproblem and h(n) is the heuristic function.
- AO* search is optimal if the heuristic function is admissible.

Feature	A* Search	AO* Search
Problem type	Solves problems with state-space graphs	Solves problems with AND-OR graphs
Evaluation function	f(n) = g(n) + h(n) where g(n) is the cost of reaching the current node from the start node	f(n) = g(n) + h(n) where g(n) is the cost of solving the current subproblem
Optimality	Optimal if the heuristic function is admissible	Optimal if the heuristic function is admissible

---

Unit 2: Knowledge Representation

2.1 Problems in representing knowledge

There are several challenges in representing knowledge in a way that can be used by AI systems. These include:

- The sheer volume of knowledge: The amount of knowledge that is needed to represent even a small domain can be vast.
- The difficulty of characterizing knowledge accurately: Knowledge is often complex and nuanced, making it difficult to represent in a formal way.
- The dynamic nature of knowledge: Knowledge is constantly changing, so AI systems need to be able to update their knowledge bases accordingly.
- The organization of knowledge: Knowledge needs to be organized in a way that corresponds to how it will be used by the AI system.

2.2 Knowledge representation methods

There are many different methods for representing knowledge in AI systems. Some of the most common include:

- Propositional logic: A simple logic that can represent facts and relationships between

facts.

- Predicate logic: A more expressive logic that can represent objects, properties, and relationships between objects.
- Semantic networks: A graphical representation of knowledge that uses nodes to represent concepts and links to represent relationships between concepts.
- Frames: A data structure that represents knowledge as a collection of slots and fillers.
- Rules: A way of representing knowledge as a set of if-then rules.

2.2.1 Propositional Logic and Predicate Logic

- Propositional logic is a simple logic that can represent facts and relationships between facts. It is based on the idea of propositions, which are statements that can be either true or false. Propositions can be combined using logical connectives such as AND, OR, and NOT to form more complex sentences.
- Predicate logic is a more expressive logic that can represent objects, properties, and relationships between objects. It is based on the idea of predicates, which are statements that describe properties of objects or relationships between objects. Predicates can be combined with quantifiers such as “for all” and “there exists” to form more complex sentences.

2.2.2 Comparison of Propositional and Predicate Logic

Feature	Propositional Logic	Predicate Logic
Expressiveness	Limited to representing simple facts and their combinations.	Can represent objects, properties, and relationships, enabling more complex knowledge representation.
Complexity	Simpler syntax and semantics.	More complex syntax and semantics due to quantifiers, variables, and predicates.
Reasoning	Inference is typically done through truth tables or syntactic manipulations.	Inference often involves complex procedures like unification and resolution.
Efficiency	Generally more efficient for computation due to its simplicity.	Can be computationally more expensive, especially with large knowledge bases.
Applications	Suitable for simpler problems like solving puzzles or basic circuit design.	Better suited for problems that require nuanced understanding of relationships and properties, such as natural language processing or complex knowledge-based systems.

2.3 Resolution and refutation

- Resolution is a sound and complete inference rule that can be used to prove sentences in propositional and predicate logic. It works by resolving two clauses that contain complementary literals to produce a new clause.
- Refutation is a proof technique that can be used to prove a sentence by showing that its negation leads to a contradiction.

2.4 Deduction, theorem proving, and inferencing

- Deduction is the process of deriving new sentences from old ones using inference rules.
- Theorem proving is the process of finding a proof for a given sentence.
- Inferencing is a more general term that refers to any process of drawing conclusions from evidence.

2.5 Monotonic and non-monotonic reasoning

- Monotonic reasoning is a type of reasoning in which the set of entailed sentences can only increase as information is added to the knowledge base.
- Non-monotonic reasoning is a type of reasoning in which the set of entailed sentences can both increase and decrease as information is added to the knowledge base.

Unit 3: Probabilistic Reasoning and Knowledge Structures

3.1 Probabilistic reasoning and Bayes' Theorem

- Probabilistic reasoning is a method of reasoning that uses probability theory to represent and reason with uncertain knowledge.
- Bayes' Theorem is a mathematical formula that is used to update the probability of an event based on new evidence.

3.2 Semantic networks, scripts, schemas, and frames

- Semantic networks are a graphical way of representing knowledge that uses nodes to represent concepts and links to represent relationships between concepts.
- Scripts are a way of representing knowledge about stereotypical events, such as going to a restaurant or taking a plane trip.
- Schemas are a more general way of representing knowledge about the world, including knowledge about objects, events, and situations.
- Frames are a data structure that represents knowledge as a collection of slots and fillers.

3.3 Conceptual dependency representation

- Conceptual dependency representation is a way of representing the meaning of sentences in terms of a small number of primitive actions and concepts.

3.4 Forward and backward reasoning

- Forward reasoning is a type of reasoning that starts with known facts and uses inference rules to derive new facts.
- Backward reasoning is a type of reasoning that starts with a goal and uses inference rules to find facts that support the goal.

Unit 4: Game Playing and Natural Language Processing (NLP)

4.1 Game-playing techniques

Game playing has been a core domain in artificial intelligence since the early days. It is easy to represent a game as a search problem, and the solution to that search problem is a sequence of moves that leads to a win, or at least a draw.

4.1.1 Minimax procedure

The minimax procedure is a decision rule used in decision theory, game theory, statistics, and philosophy for minimizing the possible loss for a worst-case (maximum loss)[1] scenario. The minimax procedure is a method in AI for making decisions when the opponent is also perfectly rational. It is a recursive algorithm that proceeds all the way down to the leaves of the tree and then backs up the minimax values through the tree as the recursion unwinds.[2]

4.1.2 Alpha-beta cut-offs

Alpha-beta pruning is a way of making the minimax procedure more efficient by eliminating large parts of the search tree that do not affect the final decision. It is often possible to prune entire subtrees rather than just leaves. The general principle is this: consider a node *n* somewhere in the[3] search tree, such that the player has a choice of moving to that node. If the player has a better choice either at the parent node of *n* or at any choice point further up, then that player will never move to node *n*. So once we have found out enough about *n* (by

examining some of its descendants) to reach this conclusion, we can prune it.[4]

4.2 Planning

Planning is a higher-level cognitive process that involves reasoning about possible courses of actions and their outcomes. It is used to achieve a goal or to solve a problem. Planning is an important part of AI because it allows agents to act rationally in complex environments.

4.2.1 Study of the block world problem in robotics

The blocks world problem is a classic planning problem in AI. It involves a robot arm that can pick up and move blocks around a table. The goal is to arrange the blocks in a particular configuration. The blocks world problem is a good example of a planning problem because it is simple to state but difficult to solve. There are many different possible ways to arrange the blocks, and the robot arm must be able to find a sequence of actions that will achieve the goal. The blocks world problem has been used to study a variety of different planning algorithms, including STRIPS, GraphPlan, and SATPlan.

4.3 Introduction to understanding and Natural Language Processing (NLP)

Understanding is the process of extracting meaning from information. In AI, understanding is often used in conjunction with natural language processing (NLP) to allow agents to understand human language.

4.3.1 Components of NLP

NLP has many components, including:

- Syntactic analysis: The process of analyzing the grammatical structure of a sentence.
- Semantic interpretation: The process of extracting the meaning of a sentence.
- Pragmatic interpretation: The process of taking into account the context in which a sentence is uttered.

4.3.2 Applications of NLP in designing expert systems

NLP can help design expert systems by allowing the system to interact with human experts in natural language. This can make it easier to acquire knowledge from experts and to build and maintain expert systems.

Unit 5: Expert Systems

5.1 Definition and characteristics of Expert Systems (ES)

An expert system is a computer program that is designed to mimic the problem-solving behavior of a human expert.

Expert systems are typically used to solve complex problems that require specialized knowledge. They are often used in the following domains:

- Medical diagnosis
- Financial analysis
- Engineering design
- Scientific discovery

Some of the key characteristics of expert systems include:

- They are able to reason with uncertain or incomplete information.
- They are able to explain their reasoning.
- They are able to learn from experience.

5.2 Requirements, components, and capabilities of Expert Systems

Requirements

- Expertise: The system must have access to a substantial body of expert knowledge.
- Problem-solving ability: The system must be able to use its knowledge to solve complex problems.
- Explanation capability: The system must be able to explain its reasoning to the user.
- Learning ability: The system should be able to learn from experience.

Components

- Knowledge base: A repository of expert knowledge.
- Inference engine: A program that uses the knowledge base to solve problems.
- User interface: A way for the user to interact with the system.

Capabilities

- Problem-solving: Expert systems are able to solve complex problems that require specialized knowledge.
- Explanation: Expert systems are able to explain their reasoning to the user.
- Learning: Expert systems are able to learn from experience.

5.3 Inference Engine

The inference engine is the program that uses the knowledge base to solve problems. It does this by using a process of reasoning, such as forward or backward chaining.

5.3.1 Forward and backward chaining

- Forward chaining is a type of reasoning that starts with known facts and uses inference rules to derive new facts.
- Backward chaining is a type of reasoning that starts with a goal and uses inference rules to find facts that support the goal.

5.4 Limitations of Expert Systems

- Limited knowledge: Expert systems are only as good as the knowledge that is encoded in their knowledge bases.
- Difficulty in acquiring knowledge: It can be difficult to acquire the knowledge needed to build an expert system.
- Brittleness: Expert systems can be brittle, meaning that they can break down when faced with unexpected inputs.
- Lack of common sense: Expert systems often lack common sense, meaning that they can make mistakes that a human expert would never make.

5.5 Expert System Development Environment and technology

There are a number of different tools and technologies that can be used to develop expert systems. These include:

- Expert system shells: These are software packages that provide a framework for building expert systems.
- Knowledge representation languages: These are formal languages that are used to represent knowledge in expert systems.
- Inference engines: These are programs that use the knowledge base to solve problems.

5.6 Benefits of Expert Systems

- Expertise: Expert systems can provide expertise in a specific domain, even when human experts are not available.
- Efficiency: Expert systems can often solve problems more efficiently than humans.
- Consistency: Expert systems can provide consistent results, even when faced with complex or ambiguous problems.
- Learning: Expert systems can learn from experience, meaning that they can become more accurate and efficient over time.

Related posts:

1. Artificial Intelligence Intelligence Tutorial for Beginners
2. Difference between Supervised vs Unsupervised vs Reinforcement learning
3. What is training data in Machine learning
4. What other technologies do I need to master AI?
5. How Artificial Intelligence (AI) Impacts Your Daily Life ?
6. Like machine learning, what are other approaches in AI ?
7. Best First Search in AI
8. Heuristic Search Algorithm
9. Hill Climbing in AI

10. A* and AO* Search Algorithm
11. Knowledge Representation in AI
12. Propositional Logic and Predicate Logic
13. Resolution and refutation in AI
14. Deduction, theorem proving and inferencing in AI
15. Monotonic and non-monotonic reasoning in AI
16. Probabilistic reasoning in AI
17. Bayes' Theorem
18. Transformer Architecture in LLM
19. Input Embedding in Transformers
20. Positional Encoding in Transformers
21. Multi-Head Attention in Transformers