Vikash Polytechnic, Bargarh

Vikash Institute of Technology
Campus: Vikash Knowledge Hub, Barahaguda Canal Chowk, NH6
PO/DIST: Bargarh-768028, Odisha

Lecture Note on Artificial Intelligence & Machine Learning

Diploma 6th Semester

Submitted By:- Mrs. Lovely Rath

Unit-1

Introduction to AI

Artificial – man made

Intelligence – thinking power

AI is a branch of computer science by which we can create intelligent machine which can behave like human & think like human

E.g. Phone is pre-program machine(receiving calls or not & several activities) but AI don't need to preprogram because we have created intelligent machine. we need a machine that can sense the problem & react like a human as human reacts/does its work according to its situation.

Q. Why AI is needed?

AI has become increasingly important in the modern world due to its numerous applications & benefits & is quickly changing how we live our lives. In the real world man need smart machine as well as a machine who can think, judge, make decision, predict & do as human does according to different situation so that it can does the work whenever man is absent in that situation or man is unreachable at that point of time or it can save the time of man so man can do rest of its work smoothly.

Goals of AI

1. replicate human intelligence
2. solve knowledge-intensive tasks
3. connection of perception & action

Composition of AI

To create AI, first we should know how intelligence is composed, as intelligence is an intangible part of our brain which is a combination of reasoning, learning , problem solving, perception, language understanding etc.

1. reasoning (judgement, decision making & prediction)
2. learning(gaining knowledge)
3. perception(acquiring/interpreting, selecting/organizing sensors)
4. problem solving(working through the details of the problem to reach solution)
5. linguistic intelligence(comprehend speak, write verbal, written language

To achieve the above factors for a machine/software AI requires all the following disciplines:

1. mathematics
2. biology (scientific study of life)
3. psychology(study of mind & behavior)
4. sociology(human social relationship & institutions)
5. computer sc.(computation, automation & information)

6. neurology(nerve system)
7. statistics(presentation, interpretation, collection, analysis of data)

examples of AI

1. google assistant
2. AleXa, Siri, Cortana
3. chatbot(ask Disha 2.0 in IRCTC)
4. face detection & recognition

Evolution of AI

Ai isn't a new term & not a new technology for researchers. This technology is much older than you imagine. There are references of mechanical men in Greek & Egyptian mythology

In the year 1943, the first work which recognized as AI was done by Warren McCulloch & Walter Pits in 1943. They proposed a model of artificial neurons. Gradually evolution happened & some failure some success came in the path but as the time passes, in 2011 IBM supercomputer Watson won jeopardy, a quiz show where it had to solve complex question as well as riddles. Watson proved that it could understand natural language & can solve tricky questions quickly. Then the journey starts here we are seeing apps & features like google assistant, Siri, chatbot ,ADAS features in cars, TESLA cars, considered a big move towards AI

Application Of AI

1. astronomy(understanding the universe how it works, origin etc.)
2. healthcare(better & faster diagnostics than human)
3. gaming(strategic games like chess, mission games FAUJI)
4. finance(automation, trading, bitcoin)
5. social media(fb, Instagram)
6. data security(detect s/w bug, cyber-attack, phishing etc.)
7. travel & transport(travel arrangement hotels, best route)
8. automotive industry(virtual assistance, driverless car)
9. robotics(erica, Sophia)
10. entertainment(Netflix, amazon prime)
11. agriculture(agriculture monitoring, weather forecasting, predictive analysis)
12. e-commerce(helping shoppers to choose matching dress)
13. education(chatbot can communicate with students as teaching assistant)

Agent(machine/robots/anything depending upon the situation)

It is an independent program/entity that interact with its environment by perceiving its surroundings

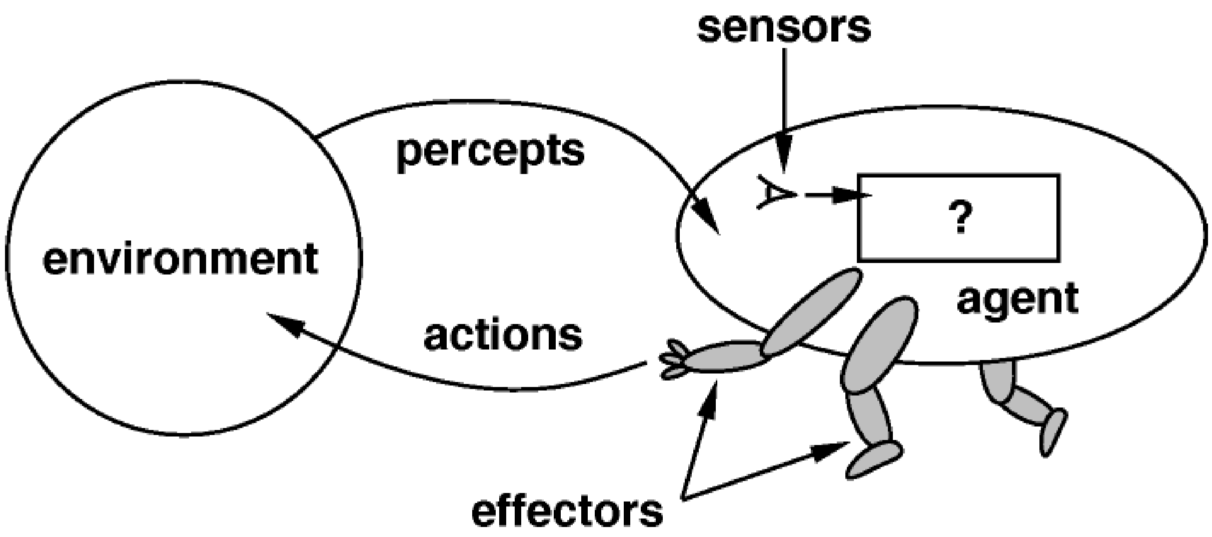
Intelligent agent

An agent who interacts with its environment by perceiving its surroundings via sensors then acting through actuators or effects.

An agent who sees the situation & after thinking & analyzing the situation & surrounding it reacts which is suitable to the situation as human does.

Imp. Terminology

1. sensors – device which detects the change in the environment & sends information to respective devices
2. actuators – component/machine that converts energy into motion. It is only responsible for moving & controlling machine.



Types of agent

1. human agent
   a. sensor – eye, ear, nose, tongue, skin & skin & another organ
   b. actuator – hand, leg, vocal tract
2. robotic agent –
   a. sensor – NLP(natural language processing – ability to understand text & spoken words as the same way human being can)
   b. camera
   c. IRFC –
3. s/w agent – Siri, Alexa, google assistance etc.

Rationality:

The ability to make decisions based on logical reasoning and optimize behaviour to achieve its goals, considering its perception of the environment and the performance measure.

It depends on PEAS

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

Rational agent:
It's a theoretical entity that considers realistic models of how people think, with preferences for advantageous outcomes and an ability to learn. In other words, it's what most people would call "you."

The rational agent is used in game theory and decision theory to help us apply artificial intelligence to various real-world scenarios.

We can use it to understand how we make decisions, allowing us to develop artificial intelligence that can mimic human behaviour to solve problems or make decisions.

Problems solving steps in AI:

The problem of AI is directly associated with the nature of humans and their activities. So, we need a number of finite steps to solve a problem which makes human easy works.

These are the following steps which require to solve a problem :

- **Problem definition:** Detailed specification of inputs and acceptable system solutions.
- **Problem analysis:** Analyse the problem thoroughly.
- **Knowledge Representation:** collect detailed information about the problem and define all possible techniques.
- **Problem-solving:** Selection of best techniques.

Components to formulate the associated problem:

- **Initial State:** This state requires an initial state for the problem which starts the AI agent towards a specified goal. In this state new methods also initialize problem domain solving by a specific class.
- **Action:** This stage of problem formulation works with function with a specific class taken from the initial state and all possible actions done in this stage.

- **Transition:** This stage of problem formulation integrates the actual action done by the previous action stage and collects the final stage to forward it to their next stage.
- **Goal test:** This stage determines that the specified goal achieved by the integrated transition model or not, whenever the goal achieves stop the action and forward into the next stage to determines the cost to achieve the goal.
- **Path costing:** This component of problem-solving numerical assigned what will be the cost to achieve the goal. It requires all hardware software and human working cost.

Learning Agent Architecture:

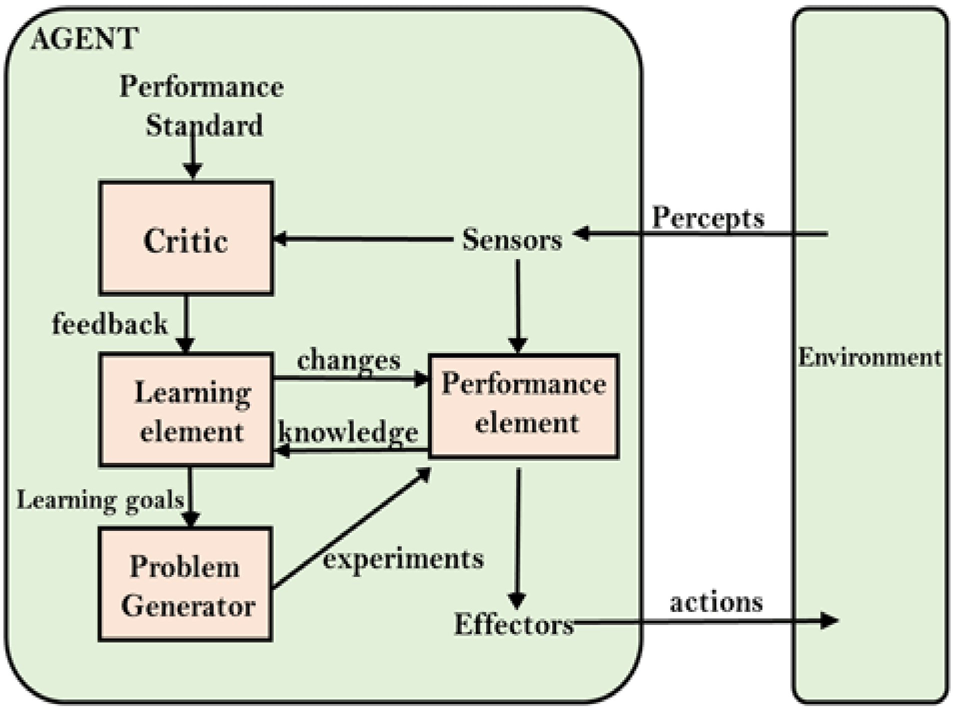
It allows agent to operate initially in unknown environment & to become more competent than its initial knowledge alone might allow

The learning element is responsible for making improvement

The performance element is responsible for making selecting external actions. The performance element takes in precepts & decide on actions.

The learning element uses feedback from the critic on how agent is doing & determine how performance element should be modified to be better in future.

The last component of the learning problem agent is problem generator. It is responsible for suggesting actions that will lead to new informative experience.



e.g.: class test before final exam

1. **Learning element:** This component is responsible to learn from the difference between performance standard & the feedback from the critics. According to the current percept it is supposed to understand the expected behaviour & enhance the standard.
2. **Critic:** It is the one who compares sensor's I/P specifying effects of agent's action on the environment with performance standard & generate feedback for learning element. The learning element takes feedback from critics which describes how well the agent is doing with respect to a fixed performance standard.
3. **Performance element:** based on the current percept received from sensors & the I/P obtained by the learning element, performance element is responsible to choose the actions to act upon the external environment.
   It is responsible for selecting external action.
4. **Problem Generator:** This component is responsible for suggesting actions that will lead to new and informative experiences.

Problem Solving Agent

Problem-solving in artificial intelligence is the process of finding a solution to a problem. There are many different types of problems that can be solved, and the methods used will depend on the specific problem. The most common type of problem is finding a solution to a maze or navigation puzzle.
Other types of problems include identifying patterns, predicting outcomes, and determining solutions to systems of equations. Each type of problem has its own set of techniques and tools that can be used to solve it.

There are three main steps in problem-solving in artificial intelligence:
1) understanding the problem: This step involves understanding the specifics of the problem and figuring out what needs to be done to solve it.
2) generating possible solutions: This step involves coming up with as many possible solutions as possible based on information about the problem and what you know about how computers work.
3) choosing a solution: This step involves deciding which solution is best based on what you know about the problem and your options for solving it.

Types of Problem-Solving Agents
Problem-solving agents are a type of artificial intelligence that helps automate problem-solving. They can be used to solve problems in natural language, algebra, calculus, statistics, and machine learning.
There are three types of problem-solving agents: propositional, predicate, and automata.
Propositional problem-solving agents can understand simple statements like "draw a line between A and B" or "find the maximum value of x."
Predicate problem-solving agents can understand more complex statements like "find the shortest path between two points" or "find all pairs of snakes in a jar."
Automata is the simplest form of problem-solving agent and can only understand sequences of symbols like "draw a square."

Classification of Problem-Solving Agents

Problem-solving agents can be classified as general problem solvers or domain-specific problem solvers. General problem solvers can solve a wide range of problems, while domain-specific problem solvers are better suited for solving specific types of problems. General problem solvers include AI programs that are designed to solve general artificial intelligence (AI) problems such as learning how to navigate a 3D environment or playing games. Domain-specific problem solvers include programs that have been specifically tailored to solve certain types of problems, such as photo editing or medical diagnosis. Both general and domain-specific problem-solving agents can be used in conjunction with other AI tools, including natural language processing (NLP) algorithms and machine learning models. By combining these tools, we can achieve more effective and efficient outcomes in our data analysis and machine learning processes.

State space search

State space search is a method used widely in artificial intelligence **and computer science to find a solution to a problem by searching through the set of possible states of the problem.** Furthermore, a state space search algorithm uses the state space to navigate from the initial state to the goal state. Additionally, it generates and explores possible successors of the current state until we find a solution.

The state space size can greatly affect a search algorithm's efficiency. Hence, it's important to choose an appropriate representation and search strategy to efficiently search the state space.

A common example of a state space search is the 8-puzzle problem. The 8-puzzle is a sliding puzzle that consists of 8 numbered tiles in a 3   3 grid and one blank space. The goal is to rearrange the tiles from a given initial state to a final goal state by sliding the tiles into the blank space.

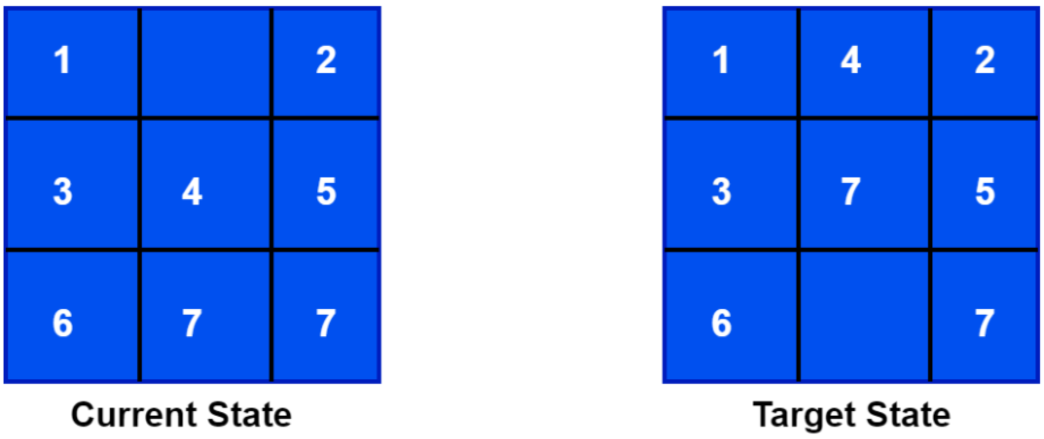
In this problem, we represent the state space by the 9 tiles in the puzzle and their positions in the grid. Additionally, we present each state in the state space by a 3   3 array with values from 1 to 8. Finally, we represent a blank space with an employ tile.

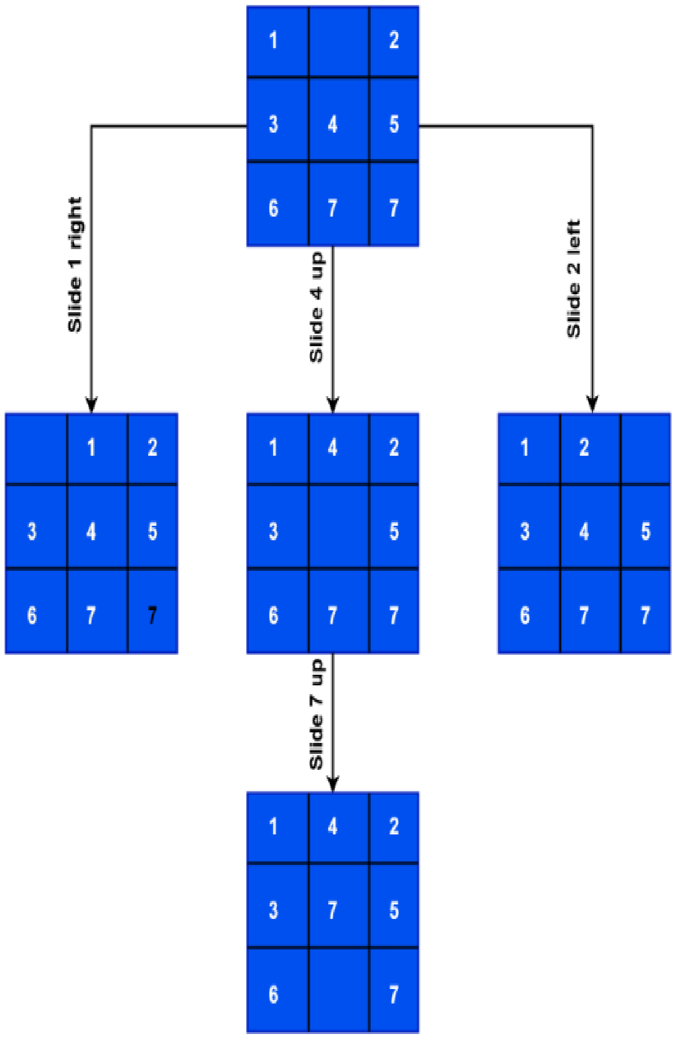
The initial state represents the starting configuration of the tiles. The goal state represents the desired configuration. Furthermore, the search algorithms use the state space to find a sequence of moves that transform the initial state into the goal state.

For example, we can use the breadth-first search to explore all possible states reachable from the initial state. It explores all the states one by one in a sequence. It ensures the solution but can become very slow for large state spaces. Other algorithms, such as A* search, use heuristics to guide the search more efficiently.

E.g.:

We're taking a practical example of an 8-puzzle problem. Let's pick current and target states for this example:





State space search algorithms have a wide range of applications in various fields, including artificial intelligence, robotics, game playing, computer networks, operations research, bioinformatics, cryptography, and supply chain management.

In artificial intelligence, we can use state space search algorithms to solve problems such as pathfinding, planning, and scheduling. Additionally, we can employ space search algorithms to plan the motion of robots, determining the best sequence of actions to achieve a goal. Furthermore, we also use these algorithms in games to determine the best move for a player, given a particular game state.

In computer networks, we can take advantage of state space search algorithms to optimize routing and resource allocation in computer networks. Additionally, we also use them in operations research to solve optimization problems, such as scheduling and resource allocation. We can also apply these algorithms to find patterns in biological data and predict protein structures in bioinformatics.

Searching Technique/algorithms

In AI whenever we are dealing with any problem like water jug, 8 queen, 8 puzzle, chess we can search/traverse from initial to final/goal state using 2 techniques.

1. Uninformed/Blind/Brute-Force Search
2. Informed/heuristics Search

Properties Of Searching algorithm:

1.completeness:it guarantees to return a solution at least 1

2.optimality: the solution found for an algorithm is guaranteed to be best

3.time complexity: it measures the time to complete a task

4.space complexity: max storage space required at any point during the search

Uninformed/Blind/Brute-Force Search

- **Uninformed searches**, also known as blind searches, are search algorithms that explore a problem space without using any specific knowledge or heuristics about the problem domain. They operate in a brute force, meaning they try out every part of search space blindly.
- As we are traversing/searching all the successor nodes/states starting from root node/initial state to reach final/goal state so it will take more time & space also, because we have to store that all possible traversed/searched nodes/states.
- This type of searching will give the optimal solution because we searching/traversing all the state which are possible from initial to goal state. In those possible states one path must be shortest in terms of time/cost/distance, that shortest solution is considered as the optimal solution.
- E.g.: BFS(Breadth First Search), DFS(Depth First Search)

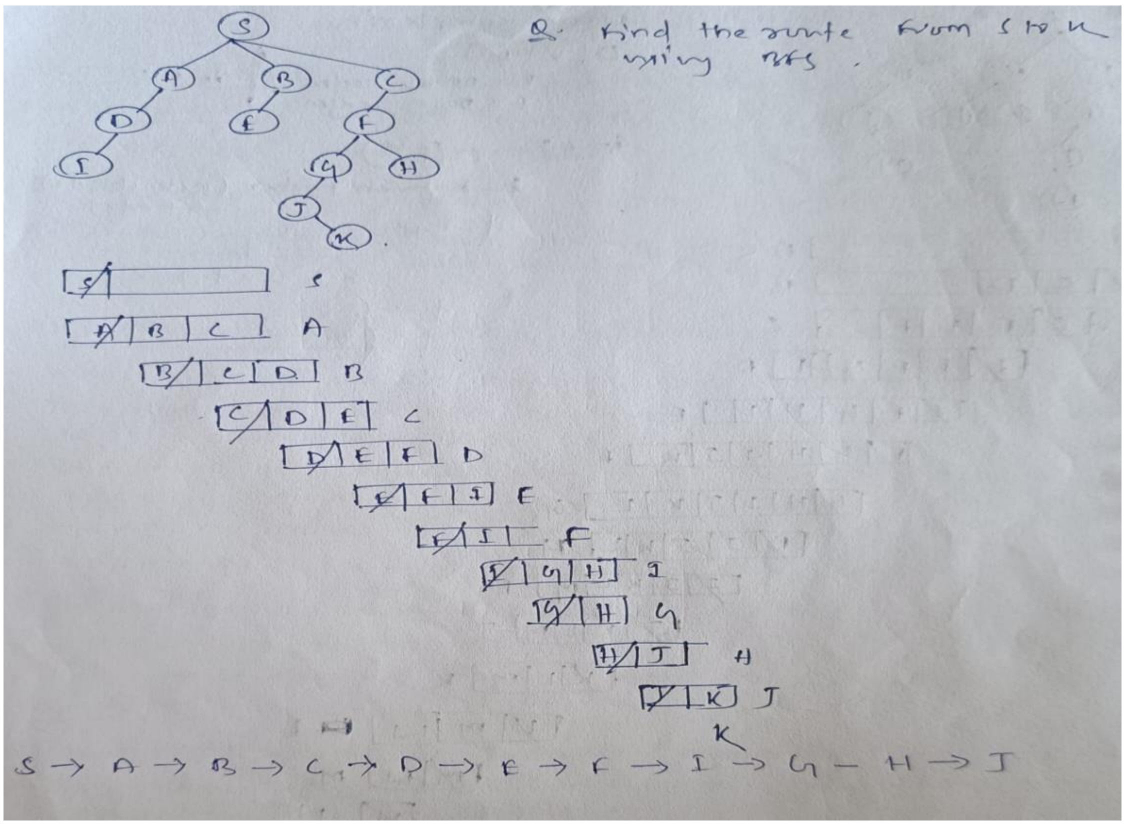
BFS(Breadth First Search)

- Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.
- BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before moving to nodes of next level.
- The breadth-first search algorithm is an example of a general-graph search algorithm.
- Breadth-first search implemented using FIFO queue data structure.
- It always deals with the shallowest node which comes under the level wise completion.

Algorithm:

1. Enter starting node on queue
2. If QUEUE is empty then return FAIL & STOP
3. If FIRST element on QUEUE is GOAL then return SUCCESS & STOP
4. Else
5. Remove & expand FIRST element from QUEUE & insert children in QUEUE
6. Goto step 2

Example:


Q. Find the route from S to K using BFS.
S
A
B
C
D
E
F
I
G
H
J
K
S → A → B → C → D → E → F → I → G – H → J

Time Complexity: Time Complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node. Where the d= depth of shallowest solution and b is a node at every state.

T (b) = O (b^d)

Space Complexity: Space complexity of BFS algorithm is given by the Memory size of frontier which is O(b^d).

Advantages:

BFS will provide a solution if any solution exists.

If there are more than one solution for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

Disadvantages:

It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
BFS needs lots of time if the solution is far away from the root node.

DFS(depth-first search)

This searching technique comes under uninformed/blind search

It is a recursive algorithm to search all the vertices of a tree data structure or a graph. The depth-first search (DFS) algorithm starts with the initial node of graph G and goes deeper until we find the goal node or the node with no children.

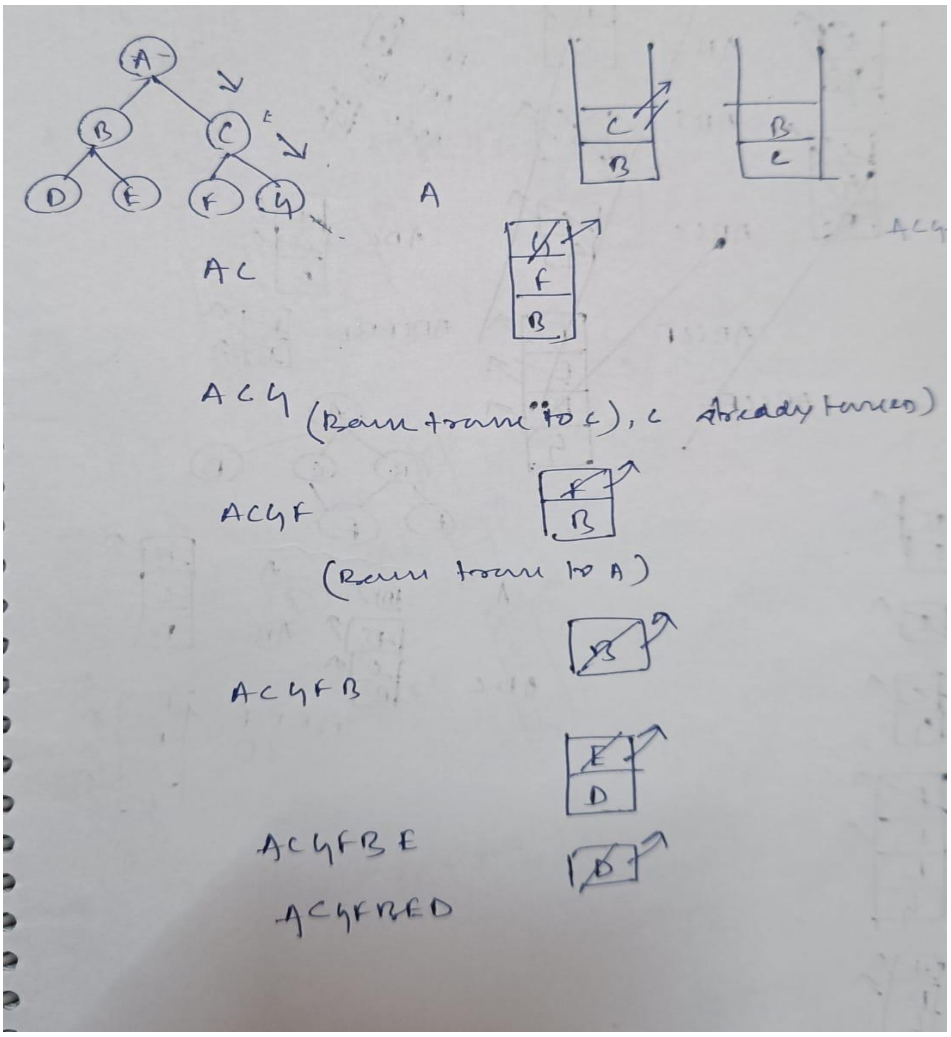
It uses stack to remember to get the next vertex to start a search which operate on LIFO(last in first out)

It involves both forward & backward searching

Algorithm:

```
1. Enter root node on stack
2. Do until stack isn't empty
    a. Remove node(POP)
        i. If node = GOAL then STOP
        ii. Else PUSH all children nodes in the stack
```

e.g.:

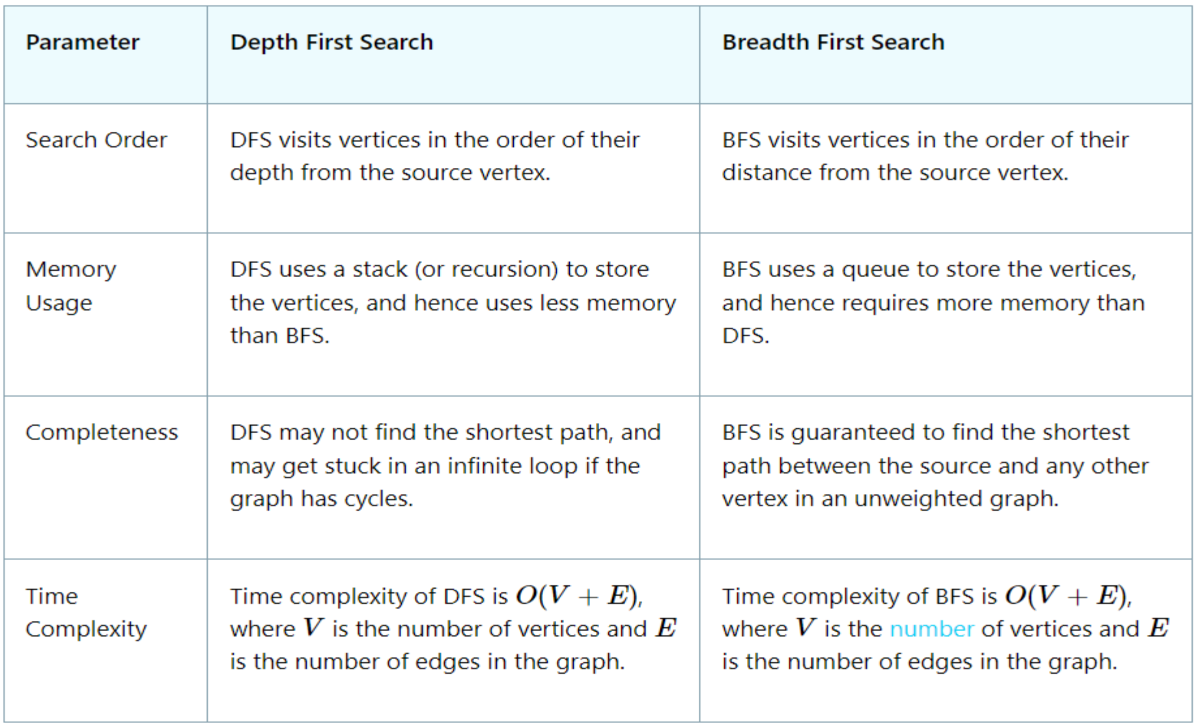


Advantage of DFS

- DFS utilizes less memory as it only stores the nodes on the current path. There is a possibility that DFS may encounter a solution without having to examine much of the search space.

Disadvantage of DFS

- DFS can be disadvantageous because it may become stuck traversing the left-most path indefinitely, even if the graph is finite, resulting in an infinite tree. A possible solution to this problem is setting a cutoff depth on the search, but determining the ideal depth, which is the solution depth, is often challenging. Selecting a depth lower than d would cause the algorithm to fail to find a solution, while choosing a higher depth than d would increase execution time significantly and may not yield the optimal solution.
- Depth-First Search is not guaranteed to find the solution and there is no guarantee to find a minimal solution, if more than one solution.



Parameter	Depth First Search	Breadth First Search
Search Order	DFS visits vertices in the order of their depth from the source vertex.	BFS visits vertices in the order of their distance from the source vertex.
Memory Usage	DFS uses a stack (or recursion) to store the vertices, and hence uses less memory than BFS.	BFS uses a queue to store the vertices, and hence requires more memory than DFS.
Completeness	DFS may not find the shortest path, and may get stuck in an infinite loop if the graph has cycles.	BFS is guaranteed to find the shortest path between the source and any other vertex in an unweighted graph.
Time Complexity	Time complexity of DFS is $O(V+E)$, where V is the number of vertices and E is the number of edges in the graph.	Time complexity of BFS is $O(V+E)$, where V is the number of vertices and E is the number of edges in the graph.

Informed/heuristic Search

What is Heuristic?

A heuristic is a technique that is used to solve a problem faster than the classic methods. These techniques are used to find the approximate solution of a problem when classical methods do not. Heuristics are said to be the problem-solving techniques that result in practical and quick solutions.

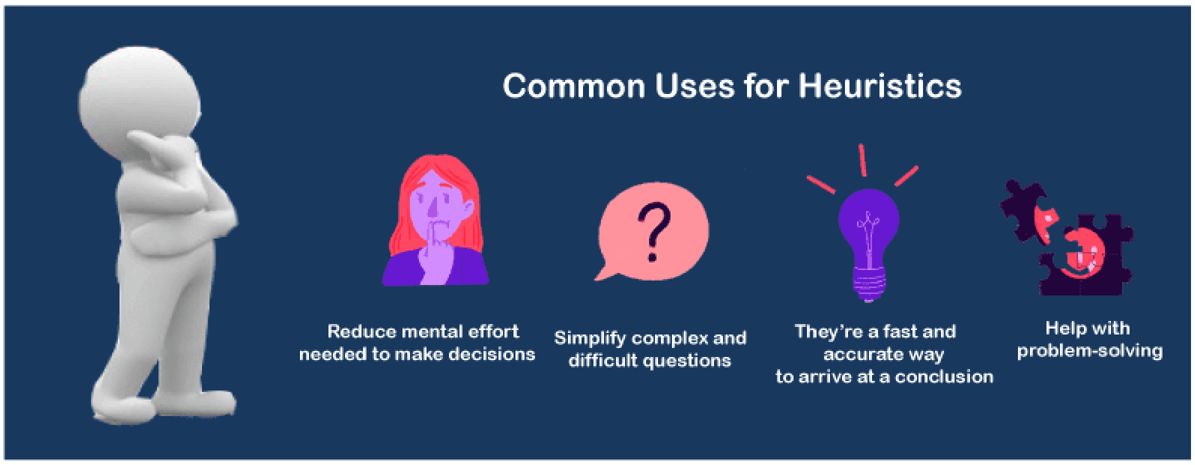
Heuristics are strategies that are derived from past experience with similar problems. Heuristics use practical methods and shortcuts used to produce the solutions that may or may not be optimal, but those solutions are sufficient in a given limited timeframe.

This type of searching uses heuristics(assumption) values to find the solution which is applicable to the problem. That heuristics value is that value which can be found from Euclidian distance, Manhattan distance, google map distance, local agent who can tell you a way from initial to GOAL.

Similarly, when we are dealing with the problems of Mathematics, we are assuming Selling Price, Cost Price as 100 , to make our solution procedure easier. With out that we can solve the problem also but that assuming procedure reduces our time & making us more comfortable towards the problem.

Why do we need heuristics?

Heuristics are used in situations in which there is the requirement of a short-term solution. On facing complex situations with limited resources and time, Heuristics can help the companies to make quick decisions by shortcuts and approximated calculations. Most of the heuristic methods involve mental shortcuts to make decisions on past experiences.


Common Uses for Heuristics
Reduce mental effort needed to make decisions
Simplify complex and difficult questions
They're a fast and accurate way to arrive at a conclusion
Help with problem-solving

The heuristic method might not always provide us the finest solution, but it is assured that it helps us find a good solution in a reasonable time.

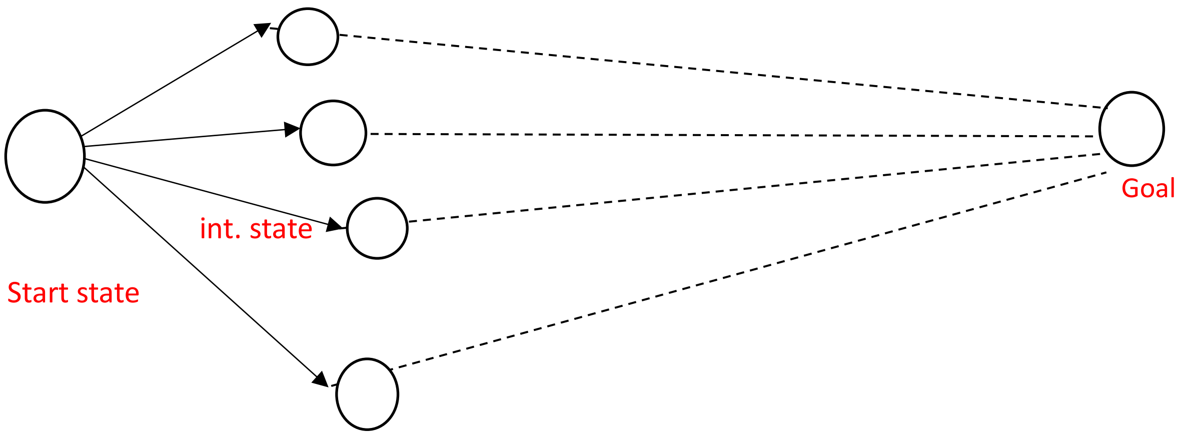
Based on context, there can be different heuristic methods that correlate with the problem's scope. The most common heuristic methods are - trial and error, guesswork, the process of elimination, historical data analysis. These methods involve simply available information that is not particular to the problem but is most appropriate. They can include representative, affect, and availability heuristics.

If we apply the uninformed/blind search then we have to reach all the states so the time complexity & space complexity will be more

In order to minimize the time complexity & space complexity & provide a quickest solution we use this approach.

How to calculate the Heuristics Value?

1.Eucledian Distance/Straight line Distance:



There are all possible states are given from initial to goal, by applying the Euclidian distance formula we know the which path suitable for the respective problem to reach the goal.

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

2.manhatten distance

Manhattan distance deals with vertical/horizontal distance

e.g.:

1	3	2
6	5	4
	8	7

Start state

1	2	3
4	5	6
7	8	

goal state

In this above problem we have to go from START to GOAL state by comparing START & GOAL

Let's compare the number position in START & GOAL

1 is in its position so distance = 0

2 isn't in its position so distance = 1(1 left move)

3 isn't in its position so distance = 1

4 isn't in its position so distance = 2

5 is in its position so distance = 0

6 isn't in its position so distance = 2

7 isn't in its position so distance = 2

8 is in its position so distance = 0

All possible moves/state = 0 + 1 + 1 + 2 + 0 + 2 + 2 + 0 = 8

Or

Count no of misplaced tiles

1 is in its position so misplaced = 0

2 isn't in its position so misplaced = 1

3 isn't in its position so misplaced = 1

4 isn't in its position so misplaced = 1

5 is in its position so misplaced = 0

6 isn’t in its position so misplaced = 1

7 isn’t in its position so misplaced = 1

8 is in its position so misplaced = 0

Total misplaced = 5

Means End Analysis

In AI we have studied many search strategies which traverse either in forward or backward but a mixture of these two is usually appropriate to solve a complex & large problem.

To solve the complex & large problem break the problem into small problem & solve the entire accordingly.

The mean end analysis is to find the difference between the current state & goal state & applying the operators to reduce the difference.

To solve mean end analysis, we need to apply MEA(means end analysis) recursively.

Working principle:

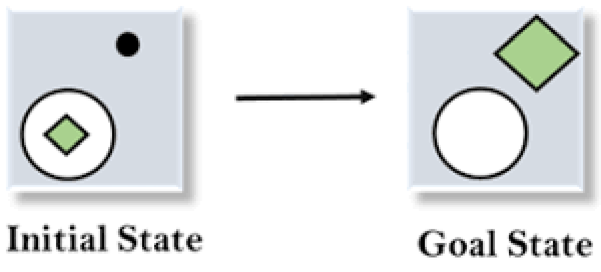
1.find the difference between initial state & goal state

2.from the available operator, select an operator which can be applied to current state to reduce the difference between current state & goal state

3.apply the selected operator

** If the operator can’t be applied to the current state, in the case we divide the current state into sub problem & then we apply an operator on sub-problem, such type of analysis is called operator sub-goaling.*

E.g.:



Operators we have

1.move

2.delete

3.expand

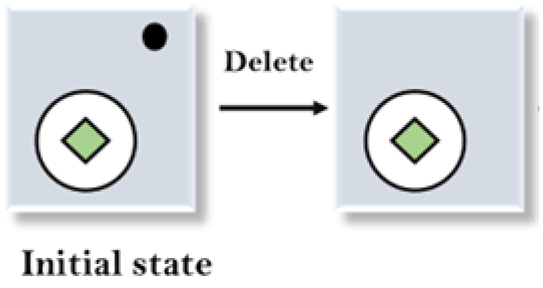
Solution:

1.evaluating the initial state

We will evaluate the initial state & will compare the initial state & goal state to find the difference

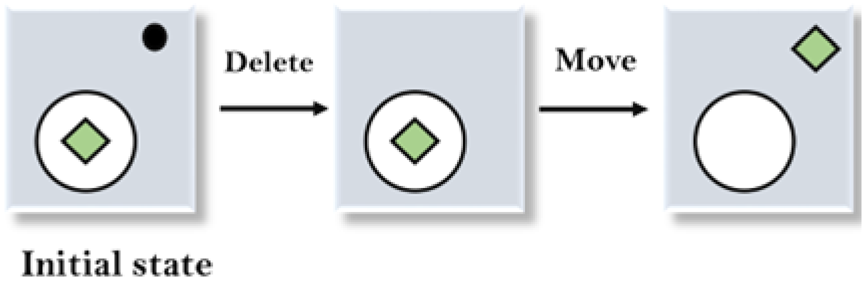
2.apply delete operator

There is no dot(.) symbol in goal state so first we have to delete that dot(.)



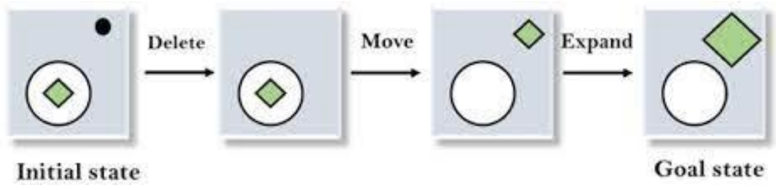
3.apply move operator

After applying delete operator the new state occurred , which will again compare with goal state. After comparing we get to know that the diamond is in outside of the circle in goal state. So, we will apply move operator.



4.apply expand operator

The size of the diamond is still occurred difference while comparing the initial & goal state. So, we will apply expand operator to increase the size of diamond.



CSP(Constraint Satisfaction Problem)

Finding a solution that meets a set of constraints is the goal of constraint satisfaction problems (CSPs), a type of AI issue. Finding values for a group of variables that fulfil a set of restrictions or rules is the aim of constraint satisfaction problems. For tasks including resource allocation, planning, scheduling, and decision-making, CSPs are frequently employed in AI.

There are mainly three basic components in the constraint satisfaction problem:

Variables: The things that need to be determined are variables. Variables in a CSP are the objects that must have values assigned to them in order to satisfy a particular set of constraints. Boolean, integer, and categorical variables are just a few examples of the various types of variables Variables, for instance, could stand in for the many puzzle cells that need to be filled with numbers in a sudoku puzzle.

Domains: The range of potential values that a variable can have been represented by domains. Depending on the issue, a domain may be finite or limitless. For instance, in Sudoku, the set of numbers from 1 to 9 can serve as the domain of a variable representing a problem cell.

Constraints: The guidelines that control how variables relate to one another are known as constraints. Constraints in a CSP define the ranges of possible values for variables. Unary constraints, binary constraints, and higher-order constraints are only a few examples of the various sorts of constraints. For instance, in a sudoku problem, the restrictions might be that each row, column, and 3×3 box can only have one instance of each number from 1 to 9.

Constraint Satisfaction Problems (CSP) representation:

- The finite set of variables V_1, V_2, V_3 ……………V_n.
- Non-empty domain for every single variable D_1, D_2, D_3 …………D_n.
- The finite set of constraints C_1, C_2 ……..…, Cm.
    - where each constraint C_i restricts the possible values for variables,
        - e.g., $V_1 \neq V_2$
    - Each constraint C_i is a pair <scope, relation>
        - Example: <(V_1, V_2), V_1 not equal to V_2>
    - Scope = set of variables that participate in constraint.
    - Relation = list of valid variable value combinations.
        - There might be a clear list of permitted combinations. Perhaps a relation that is abstract and that allows for membership testing and listing.

Real-world Constraint Satisfaction Problems (CSP):

- **Scheduling:** A fundamental CSP problem is how to efficiently and effectively schedule resources like personnel, equipment, and facilities. The constraints in this domain specify the availability and capacity of each resource, whereas the variables indicate the time slots or resources.
- **Vehicle routing:** Another example of a CSP problem is the issue of minimizing travel time or distance by optimizing a fleet of vehicles' routes. In this domain, the constraints specify each vehicle's capacity, delivery locations, and time windows, while the variables indicate the routes taken by the vehicles.
- **Assignment:** Another typical CSP issue is how to optimally assign assignments or jobs to humans or machines. In this field, the variables stand in for the tasks, while the constraints specify the knowledge, capacity, and workload of each person or machine.
- **Sudoku:** The well-known puzzle game Sudoku can be modelled as a CSP problem, where the variables stand in for the grid's cells and the constraints specify the game's rules, such as prohibiting the repetition of the same number in a row, column, or area.

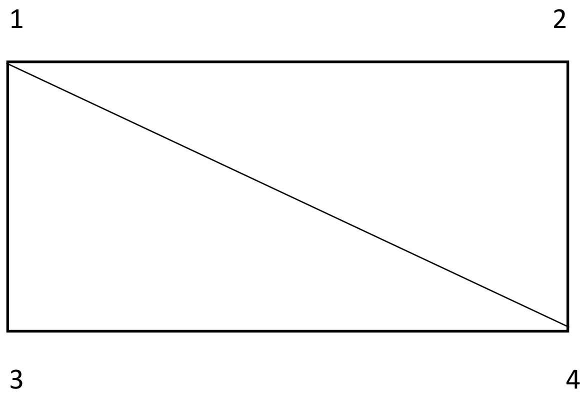
- **Constraint-based image segmentation:** The segmentation of an image into areas with various qualities (such as colour, texture, or shape) can be treated as a CSP issue in computer vision, where the variables represent the regions and the constraints specify how similar or unlike neighbouring regions are to one another.

Constraint Satisfaction Problems (CSP) benefits:

- conventional representation patterns
- generic successor and goal functions
- Standard heuristics (no domain-specific expertise).

E.g.:

A constraints graph is given. Color the graph & neighbour node must be of different colour



V = {1,2,3,4} set of variables/vertex of graph

D = {Red, Green, Blue} – domain

C = neighbour node must be of different colour

	1	2	3	4
Initial domain	R,G,B	R,G,B	R,G,B	R,G,B
1=R	R	GB	GB	GB
2=G	R	G	GB	B
3=B	R	G	B(let's take B)	B(empty)
3=G	R	G	G	B

In case of DFS, if no move arising, we are going to the first parent node/root node but in CSP we are backtracking to the conflict arising node. So, in comparison to DFS, CSP is easy to debug & time efficient.

UNIT-2

Adversarial search

Adversarial search is a search, where we examine the problem which arises when we try to plan ahead of the world and other agents are planning against us.

- In previous topics, we have studied the search strategies which are only associated with a single agent that aims to find the solution which often expressed in the form of a sequence of actions.
- But there might be some situations where more than one agent is searching for the solution in the same search space and this situation usually occurs in game playing.
- The environment with more than one agent is termed as **multi-agent environment**, in which each agent is an opponent of other agent and playing against each other. Each agent needs to consider the action of other agent and effect of that action on their performance.
- So, **Searches in which two or more players with conflicting goals are trying to explore the same search space for the solution, are called adversarial searches, often known as Games**.
- Games are modelled as a Search problem and heuristic evaluation function, and these are the two main factors which help to model and solve games in AI.

A game can be defined as a type of search in AI which can be formalized of the following elements:

- **Initial state:** It specifies how the game is set up at the start.
- **Player(s):** It specifies which player has moved in the state space.
- **Action(s):** It returns the set of legal moves in state space.
- **Result(s, a):** It is the transition model, which specifies the result of moves in the state space.
- **Terminal-Test(s):** Terminal test is true if the game is over, else it is false at any case. The state where the game ends is called terminal states.
- **Utility(s, p):** A utility function gives the final numeric value for a game that ends in terminal states s for player p. It is also called payoff function. For Chess, the outcomes are a win, loss, or draw and its payoff values are +1, 0, ½. And for tic-tac-toe, utility values are +1, -1, and 0.

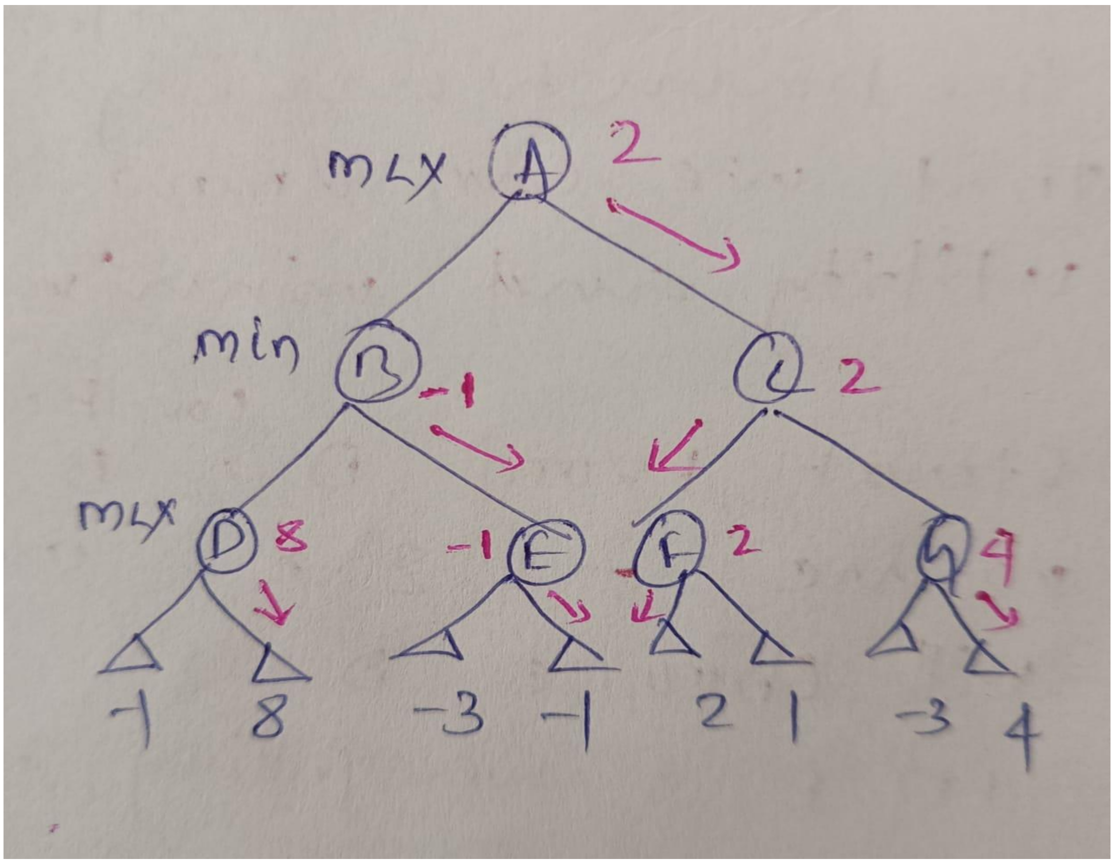
Minimax Algorithm:

- Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.
- Mini-Max algorithm uses recursion to search through the game-tree.
- Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state.
- In this algorithm two players play the game; one is called MAX and other is called MIN.
- Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.
- The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.
- The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

Working of Min-Max Algorithm:

- The working of the minimax algorithm can be easily described using an example. Below we have taken an example of game-tree which is representing the two-player game.
- In this example, there are two players one is called Maximiser and other is called Minimizer.
- Maximiser will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score.
- This algorithm applies DFS, so in this game-tree, we have to go all the way through the leaves to reach the terminal nodes.
- At the terminal node, the terminal values are given so we will compare those value and backtrack the tree until the initial state occurs. Following are the main steps involved in solving the two-player game tree:

E.g.



Properties of Mini-Max algorithm:

- **Complete-** Min-Max algorithm is Complete. It will definitely find a solution (if exist), in the finite search tree.
- **Optimal-** Min-Max algorithm is optimal if both opponents are playing optimally.
- **Time complexity-** As it performs DFS for the game-tree, so the time complexity of Min-Max algorithm is **$O(b^d)$**, where b is branching factor of the game-tree, and m is the maximum depth of the tree.
- **Space Complexity-** Space complexity of Mini-max algorithm is also similar to DFS which is **O(bm)**.

Limitation of the minimax Algorithm:

The main drawback of the minimax algorithm is that it gets really slow for complex games such as Chess, go, etc. This type of games has a huge branching factor, and the player has lots of choices to decide. This limitation of the minimax algorithm can be improved from **alpha-beta pruning**

alpha-beta pruning:

- Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.
- As we have seen in the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called **pruning**. This involves two threshold parameter Alpha and beta for future expansion, so it is called **alpha-beta pruning**. It is also called as **Alpha-Beta Algorithm**.
- Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prunes the tree leaves but also entire sub-tree.
- The two-parameter can be defined as:
    a. **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximiser. The initial value of alpha is **-∞**.
    b. **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is **+∞**.
- The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow. Hence by pruning these nodes, it makes the algorithm fast.

Key points about alpha-beta pruning:

- The Max player will only update the value of alpha.
- The Min player will only update the value of beta.
- While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.
- We will only pass the alpha, beta values to the child nodes.

Move Ordering in Alpha-Beta pruning:

The effectiveness of alpha-beta pruning is highly dependent on the order in which each node is examined. Move order is an important aspect of alpha-beta pruning.

It can be of two types:

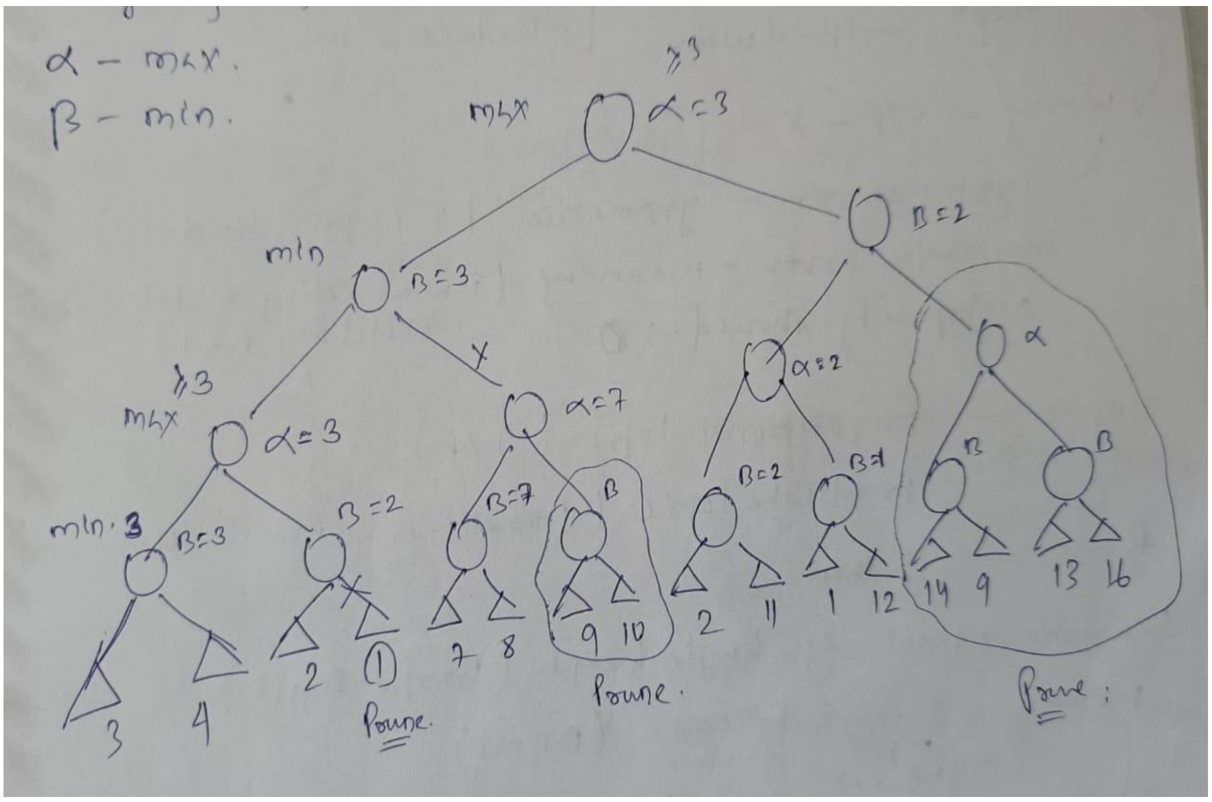
- **Worst ordering:** In some cases, alpha-beta pruning algorithm does not prune any of the leaves of the tree, and works exactly as minimax algorithm. In this case, it also consumes more time because of alpha-beta factors, such a move of pruning is called worst ordering. In this case, the best move occurs on the right side of the tree. The time complexity for such an order is O(b^m).
- **Ideal ordering:** The ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best moves occur at the left side of the tree. We apply DFS hence it first search left of the tree and go deep twice as minimax algorithm in the same amount of time. Complexity in ideal ordering is O($b^{m/2}$).

Rules to find good ordering:

Following are some rules to find good ordering in alpha-beta pruning:

- Occur the best move from the shallowest node.
- Order the nodes in the tree such that the best nodes are checked first.
- Use domain knowledge while finding the best move. Ex: for chess, try order: captures first, then threats, then forward moves, backward moves.
- We can bookkeep the states, as there is a possibility that states may repeat.

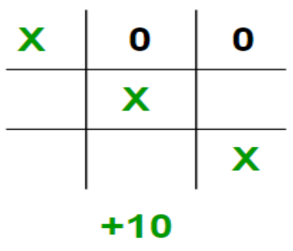
e.g.:



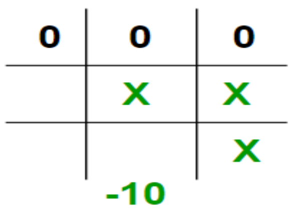
Evaluation function

As seen in the above article, each leaf node had a value associated with it. We had stored this value in an array. But in the real world when we are creating a program to play Tic-Tac-Toe, Chess, Backgammon, etc. we need to implement a function that calculates the value of the board depending on the placement of pieces on the board. This function is often known as Evaluation Function. It is sometimes also called a Heuristic Function.
The evaluation function is unique for every type of game. In this post, the evaluation function for the game Tic-Tac-Toe is discussed. The basic idea behind the evaluation function is to give a high value for a board if the **maximiser** turns or a low value for the board if the **minimizer** turns.
For this scenario let us consider **X** as the **maximiser** and **O** as the **minimizer**.
Let us build our evaluation function :

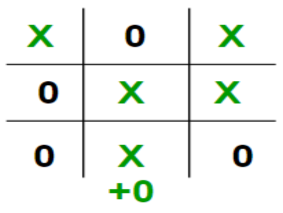
- If X wins on the board, we give it a positive value of +10.



- If O wins on the board, we give it a negative value of -10.



- If no one has won or the game results in a draw then we give a value of +0.



Knowledge-Based Agent

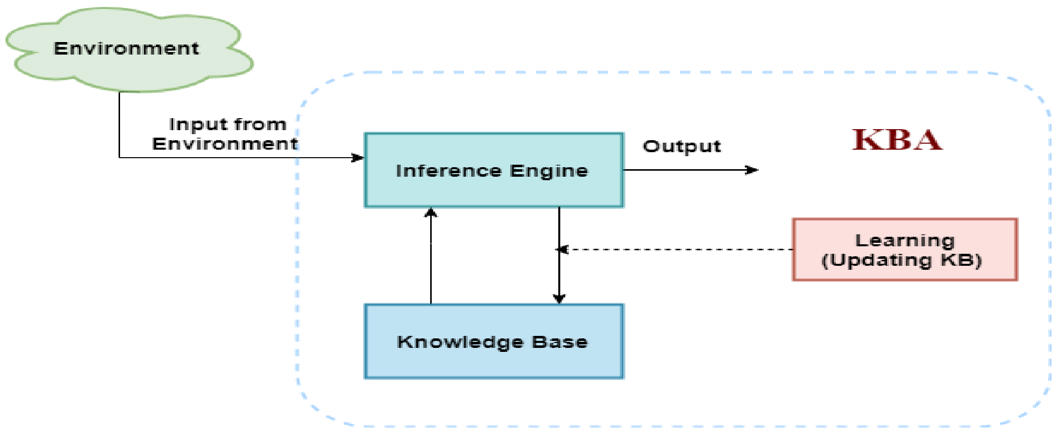
- An intelligent agent needs **knowledge** about the real world for taking decisions and **reasoning** to act efficiently.

- Knowledge-based agents are those agents who have the capability of **maintaining an internal state of knowledge, reason over that knowledge, update their knowledge after observations and take actions. These agents can represent the world with some formal representation and act intelligently**.
- Knowledge-based agents are composed of two main parts:
    - **Knowledge-base and**
    - **Inference system**.

A knowledge-based agent must able to do the following:

- An agent should be able to represent states, actions, etc.
- An agent Should be able to incorporate new percepts
- An agent can update the internal representation of the world
- An agent can deduce the internal representation of the world
- An agent can deduce appropriate actions.

The architecture of knowledge-based agent:



The above diagram is representing a generalized architecture for a knowledge-based agent. The knowledge-based agent (KBA) take input from the environment by perceiving the environment. The input is taken by the inference engine of the agent and which also communicate with KB to decide as per the knowledge store in KB. The learning element of KBA regularly updates the KB by learning new knowledge.

Knowledge base: Knowledge-base is a central component of a knowledge-based agent, it is also known as KB. It is a collection of sentences (here 'sentence' is a technical term and it is not identical to sentence in English). These sentences are expressed in a language which is called a knowledge representation language. The Knowledge-base of KBA stores fact about the world.

Why use a knowledge base?

Knowledge-base is required for updating knowledge for an agent to learn with experiences and take action as per the knowledge.

Inference system

Inference means deriving new sentences from old. Inference system allows us to add a new sentence to the knowledge base. A sentence is a proposition about the world. Inference system applies logical rules to the KB to deduce new information.

Inference system generates new facts so that an agent can update the KB. An inference system works mainly in two rules which are given as:

- Forward chaining
- Backward chaining

Approaches to designing a knowledge-based agent:

There are mainly two approaches to build a knowledge-based agent:

1. **1. Declarative approach:** We can create a knowledge-based agent by initializing with an empty knowledge base and telling the agent all the sentences with which we want to start with. This approach is called Declarative approach.
2. **2. Procedural approach:** In the procedural approach, we directly encode desired behaviour as a program code. Which means we just need to write a program that already encodes the desired behaviour or agent.

However, in the real world, a successful agent can be built by combining both declarative and procedural approaches, and declarative knowledge can often be compiled into more efficient procedural code.

Propositional logic

Propositional logic (PL) is the simplest form of logic where all the statements are made by propositions. A proposition is a declarative statement which is either true or false. It is a technique of knowledge representation in logical and mathematical form.

Proposition – sentence

logic – argument

Propositions can be either true or false, but it cannot be both.

Example:

1. It is Sunday.
2. The Sun rises from West (False proposition)
3. 3+3= 7(False proposition)
4. 5 is a prime number.
5. 1+1 = 2 (true proposition)
6. Some students are intelligent.(T/F) not a proposition because some aren't intelligent & some are intelligent ,here we confirm about which is true/false

Atomic Proposition: Atomic propositions are the simple propositions. It consists of a single proposition symbol. These are the sentences which must be either true or false.

Example:

1. a) 2+2 is 4, it is an atomic proposition as it is a **true** fact.
2. b) "The Sun is cold" is also a proposition as it is a **false** fact.

Compound proposition: Compound propositions are constructed by combining simpler or atomic propositions, using parenthesis and logical connectives.

Combining 2 or more sentences we are getting the complex proposition

We can take the help of logical connectors to make complex problems

Example:

1. a) "It is raining today, and street is wet."
2. b) "Ankit is a doctor, and his clinic is in Mumbai."

Logical Connectives:

Logical connectives are used to connect two simpler propositions or representing a sentence logically. We can create compound propositions with the help of logical connectives. There are mainly five connectives, which are given as follows:

1. **Negation:** A sentence such as ¬ P is called negation of P. A literal can be either Positive literal or negative literal.
2. **Conjunction:** A sentence which has ∧ connective such as, **P ∧ Q** is called a conjunction.
   Example: Rohan is intelligent and hardworking. It can be written as,

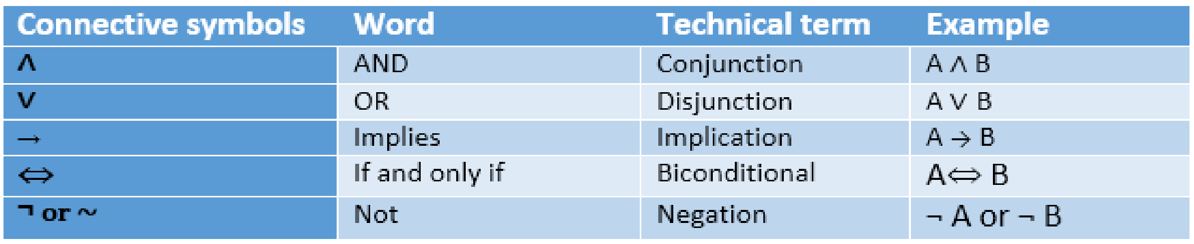
P = Rohan is intelligent,

Q= Rohan is hardworking.

P∧ Q.

3. **Disjunction:** A sentence which has ∨ connective, such as **P V Q**. is called disjunction, where P and Q are the propositions.
   Example: "Ritika is a doctor or Engineer",
   Here P= Ritika is Doctor. Q= Ritika is Doctor, so we can write it as **P V Q**.
4. **Implication:** A sentence such as P → Q, is called an implication. Implications are also known as if-then rules. It can be represented as
   If it is raining, then the street is wet.
   Let P= It is raining, and Q= Street is wet, so it is represented as P → Q
5. **Biconditional:** A sentence such as **P⇔ Q is a Biconditional sentence, example: If I am breathing, then I am alive**
   P= I am breathing, Q= I am alive, it can be represented as P ⇔ Q.

Following is the summarized table for Propositional Logic Connectives:



Connective symbols	Word	Technical term	Example
∧	AND	Conjunction	A ∧ B
∨	OR	Disjunction	A ∨ B
→	Implies	Implication	A → B
⇔	If and only if	Biconditional	A⇔ B
¬ or ~	Not	Negation	¬ A or ¬ B

Truth Table:

In propositional logic, we need to know the truth values of propositions in all possible scenarios. We can combine all the possible combination with logical connectives, and the representation of these combinations in a tabular format is called **Truth table**. Following are the truth table for all logical connectives:

For Negation:

P	¬ P
True	**False**
False	**True**

For Conjunction:

P	Q	P∧ Q
True	True	**True**
True	False	**False**
False	True	**False**
False	False	**False**

For disjunction:

P	Q	P V Q.
True	True	**True**
False	True	**True**
True	False	**True**
False	False	**False**

For Implication:

P	Q	P→ Q
True	True	**True**
True	False	**False**
False	True	**True**
False	False	**True**

For Biconditional:

P	Q	P⇔ Q
True	True	**True**
True	False	**False**
False	True	**False**
False	False	**True**

Truth table with three propositions:

We can build a proposition composing three propositions P, Q, and R. This truth table is made-up of 8n Tuples as we have taken three proposition symbols.

P	Q	R	¬R	Pv Q	PvQ→¬R
True	True	True	False	True	False
True	True	False	True	True	True
True	False	True	False	True	False
True	False	False	True	True	True
False	True	True	False	True	False
False	True	False	True	True	True
False	False	True	False	False	True
False	False	False	True	False	True

First order Logic

In the topic of Propositional logic, we have seen that how to represent statements using propositional logic. But unfortunately, in propositional logic, we can only represent the facts, which are either true or false. PL is not sufficient to represent the complex sentences or natural language statements. The propositional logic has very limited expressive power. Consider the following sentence, which we cannot represent using PL logic.

e.g:

Some humans are intelligent

All students like cricket

- First-order logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic.
- FOL is sufficiently expressive to represent the natural language statements in a concise way.
- First-order logic is also known as **Predicate logic or First-order predicate logic**. First-order logic is a powerful language that develops information about the objects in an easier way and can also express the relationship between those objects.
- First-order logic (like natural language) does not only assume that the world contains facts like propositional logic but also assumes the following things in the world:
  - **Objects:** A, B, people, numbers, colours, wars, theories, squares, pits, Wumpus, ...
  - **Relations: It can be unary relation such as:** red, round, is adjacent, **or n-any relation such as:** the sister of, brother of, has colour, comes between
  - **Function:** Father of, best friend, third inning of, end of, ...
- As a natural language, first-order logic also has two main parts:

a. **Syntax**

b. Semantics

Syntax of First-Order logic:

The syntax of FOL determines which collection of symbols is a logical expression in first-order logic. The basic syntactic elements of first-order logic are symbols. We write statements in short-hand notation in FOL.

Quantifiers in First-order logic:

- A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse.
- These are the symbols that permit to determine or identify the range and scope of the variable in the logical expression. There are two types of quantifiers:
    a. **Universal Quantifier, (for all, everyone, everything)**
    b. **Existential quantifier, (for some, at least one).**

Universal Quantifier:

Universal quantifier is a symbol of logical representation, which specifies that the statement within its range is true for everything or every instance of a particular thing.

The Universal quantifier is represented by a symbol ∀, which resembles an inverted A.

Note: In universal quantifier we use implication "→".

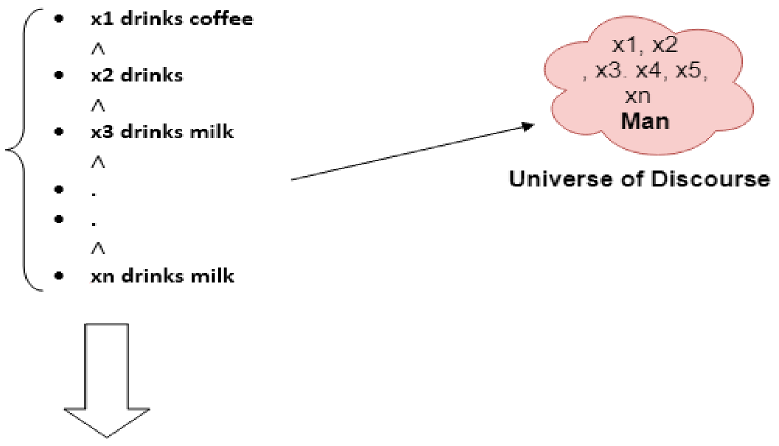
If x is a variable, then ∀x is read as:

- **For all x**
- **For each x**
- **For every x.**

Example:

All man drink coffee.

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



So in shorthand notation, we can write it as :

∀x man(x) → drink (x, coffee).

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

Existential Quantifier:

Existential quantifiers are the type of quantifiers, which express that the statement within its scope is true for at least one instance of something.

It is denoted by the logical operator ∃, which resembles as inverted E. When it is used with a predicate variable then it is called as an existential quantifier.

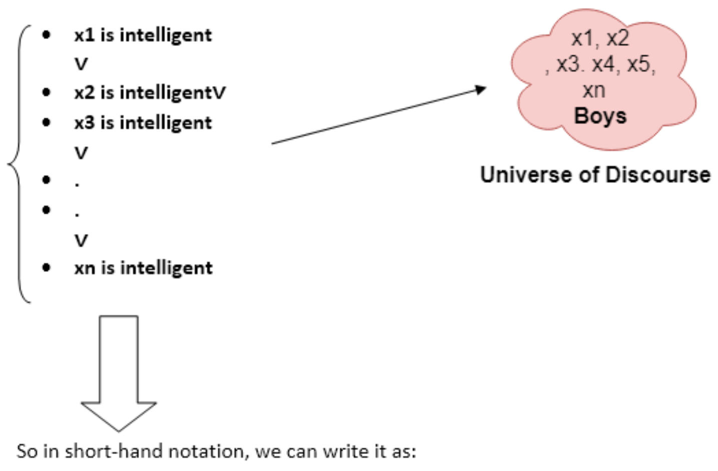
Note: In Existential quantifier we always use AND or Conjunction symbol (∧).

If x is a variable, then existential quantifier will be ∃x or ∃(x). And it will be read as:

- **There exists a 'x.'**
- **For some 'x.'**
- **For at least one 'x.'**

Example:

Some boys are intelligent.



∃x: boys(x) ∧ intelligent(x)

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

Properties of Quantifiers:

- In universal quantifier, ∀x∀y is similar to ∀y∀x.
- In Existential quantifier, ∃x∃y is similar to ∃y∃x.
- ∃x∀y is not similar to ∀y∃x.

Some Examples of FOL using quantifier:

1. All birds fly.
In this question the predicate is "**fly(bird)**."
And since there are all birds who fly so it will be represented as follows.
∀x bird(x) →fly(x).

2. Every man respects his parent.
In this question, the predicate is "**respect(x, y)," where x=man, and y= parent**.
Since there is every man so will use ∀, and it will be represented as follows:
∀x man(x) → respects (x, parent).

3. Some boys play cricket.
In this question, the predicate is "**play(x, y)**," where x= boys, and y= game. Since there are some boys so we will use **∃, and it will be represented as**:
∃x boys(x) → play(x, cricket).

4. Not all students like both Mathematics and Science.
In this question, the predicate is "**like(x, y)," where x= student, and y= subject**.
Since there are not all students, so we will use **∀ with negation, so** following representation for this:
¬∀ (x) [student(x) → like(x, Mathematics) ∧ like(x, Science)].

5. Only one student failed in Mathematics.
In this question, the predicate is "**failed(x, y)," where x= student, and y= subject**.
Since there is only one student who failed in Mathematics, so we will use following representation for this:

∃(x) [student(x) → failed (x, Mathematics) ∧∀ (y) [¬(x==y) ∧ student(y) → ¬failed (x, Mathematics)].

Inference in First Order Logic

Inference in First-Order Logic is used to deduce new facts or sentences from existing sentences. Before understanding the FOL inference rule, let's understand some basic terminologies used in FOL.

Substitution:

Substitution is a fundamental operation performed on terms and formulas. It occurs in all inference systems in first-order logic. The substitution is complex in the presence of quantifiers in FOL. If we write **F[a/x]**, so it refers to substitute a constant "**a**" in place of variable "**x**".

Note: First-order logic is capable of expressing facts about some or all objects in the universe.

Equality:

First-Order logic does not only use predicate and terms for making atomic sentences but also uses another way, which is equality in FOL. For this, we can use **equality symbols** which specify that the two terms refer to the same object.

Example: Brother (John) = Smith.

As in the above example, the object referred by the **brother (John)** is similar to the object referred by **Smith**. The equality symbol can also be used with negation to represent that two terms are not the same objects.

Example: ¬(x=y) which is equivalent to x ≠y.

Forward chaining vs Backward Chaining

S. No.	Forward Chaining	Backward Chaining
1.	Forward chaining starts from known facts and applies inference rule to extract more data unit it reaches to the goal.	Backward chaining starts from the goal and works backward through inference rules to find the required facts that support the goal.
2.	It is a bottom-up approach	It is a top-down approach
3.	Forward chaining is known as data-driven inference technique as we reach to the goal using the available data.	Backward chaining is known as goal-driven technique as we start from the goal and divide into sub-goal to extract the facts.
4.	Forward chaining reasoning applies a breadth-first search strategy.	Backward chaining reasoning applies a depth-first search strategy.
5.	Forward chaining tests for all the available rules	Backward chaining only tests for few required rules.
6.	Forward chaining is suitable for the planning, monitoring, control, and interpretation application.	Backward chaining is suitable for diagnostic, prescription, and debugging application.
7.	Forward chaining can generate an infinite number of possible conclusions.	Backward chaining generates a finite number of possible conclusions.
8.	It operates in the forward direction.	It operates in the backward direction.
9.	Forward chaining is aimed for any conclusion.	Backward chaining is only aimed for the required data.

Unification

- unification is a process of making two different logical atomic expressions identical by finding a substitution. Unification depends on the substitution process.
- It takes two literals as input and makes them identical using substitution.
- Let Ψ_1 and Ψ_2 be two atomic sentences and σ be a unifier such that, $\boldsymbol{\Psi_1\sigma = \Psi_2\sigma}$, then it can be expressed as **UNIFY(Ψ_1, Ψ_2)**.

- **Example: Find the MGU for Unify{King(x), King(John)}**

Let Ψ_1 = King(x), Ψ_2 = King(John),

Substitution θ = {John/x} is a unifier for these atoms and applying this substitution, and both expressions will be identical.

- The UNIFY algorithm is used for unification, which takes two atomic sentences and returns a unifier for those sentences (If any exist).
- Unification is a key component of all first-order inference algorithms.
- It returns fail if the expressions do not match with each other.
- The substitution variables are called Most General Unifier or MGU.

E.g. Let's say there are two different expressions, **P(x, y), and P(a, f(z))**.

In this example, we need to make both above statements identical to each other. For this, we will perform the substitution.

P(x, y)......... (i)
P(a, f(z))......... (ii)

- Substitute x with a, and y with f(z) in the first expression, and it will be represented as **a/x** and f(z)/y.
- With both the substitutions, the first expression will be identical to the second expression and the substitution set will be: **[a/x, f(z)/y]**.

Conditions for Unification:

- Predicate symbol must be same, atoms or expression with different predicate symbol can never be unified.
- Number of Arguments in both expressions must be identical.
- Unification will fail if there are two similar variables present in the same expression.

Find the MGU of {p(b, X, f(g(Z))) and p(Z, f(Y), f(Y))}

Here, Ψ_1 = p(b, X, f(g(Z))) , and Ψ_2 = p(Z, f(Y), f(Y))
S_0 => { p(b, X, f(g(Z))); p(Z, f(Y), f(Y))}
SUBST θ={b/Z}

S_1 => { p(b, X, f(g(b))); p(b, f(Y), f(Y))}
SUBST θ={f(Y) /X}

S_2 => { p(b, f(Y), f(g(b))); p(b, f(Y), f(Y))}
SUBST θ= {g(b) /Y}

S_2 => { p(b, f(g(b)), f(g(b)); p(b, f(g(b)), f(g(b))} **Unified Successfully.**

Resolution

Resolution is a theorem proving technique that proceeds by building refutation proofs, i.e., proofs by contradictions. It was invented by a Mathematician John Alan Robinson in the year 1965.

Resolution is used, if there are various statements are given, and we need to prove a conclusion of those statements. Unification is a key concept in proofs by resolutions. Resolution is a single inference rule which can efficiently operate on the **conjunctive normal form or clausal form**.

Clause: Disjunction of literals (an atomic sentence) is called a **clause**. It is also known as a unit clause.

Conjunctive Normal Form: A sentence represented as a conjunction of clauses is said to be **conjunctive normal form** or **CNF**.

Steps for Resolution:

1. Conversion of facts into first-order logic.
2. Convert FOL statements into CNF
3. Negate the statement which needs to prove (proof by contradiction)
4. Draw resolution graph (unification).

Example:

a. **john likes all kind of food.**

b. **Apple and vegetable are food**

c. **Anything anyone eats and not killed is food.**

d. **Anil eats peanuts and still alive**

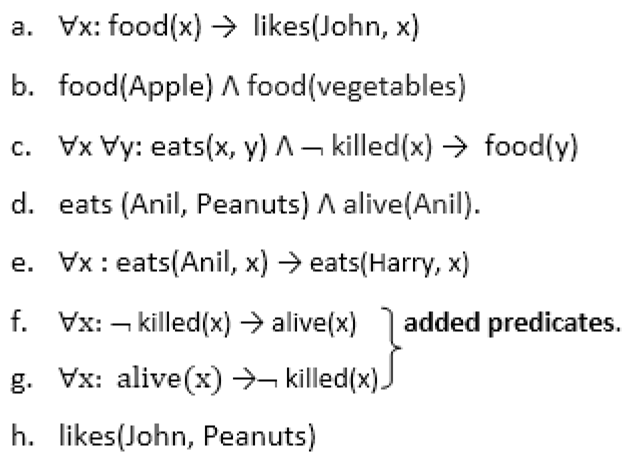
e. **Harry eats everything that Anil eats.**

Prove by resolution that:

f. **John likes peanuts.**

Step-1: Conversion of Facts into FOL

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



a. ∀x: food(x) → likes(John, x)
b. food(Apple) Λ food(vegetables)
c. ∀x ∀y: eats(x, y) Λ ¬ killed(x) → food(y)
d. eats (Anil, Peanuts) Λ alive(Anil).
e. ∀x : eats(Anil, x) → eats(Harry, x)
f. ∀x: ¬ killed(x) → alive(x) } added predicates.
g. ∀x: alive(x) →¬ killed(x)
h. likes(John, Peanuts)

Step-2: Conversion of FOL into CNF

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

- **Eliminate all implication (→) and rewrite**
  a. ∀x ¬ food(x) V likes(John, x)
  b. food(Apple) Λ food(vegetables)
  c. ∀x ∀y ¬ [eats(x, y) Λ ¬ killed(x)] V food(y)
  d. eats (Anil, Peanuts) Λ alive(Anil)
  e. ∀x ¬ eats(Anil, x) V eats(Harry, x)
  f. ∀x¬ [¬ killed(x)] V alive(x)
  g. ∀x ¬ alive(x) V ¬ killed(x)
  h. likes(John, Peanuts).
- **Move negation (¬)inwards and rewrite**
  a. ∀x ¬ food(x) V likes(John, x)
  b. food(Apple) Λ food(vegetables)
  c. ∀x ∀y ¬ eats(x, y) V killed(x) V food(y)
  d. eats (Anil, Peanuts) Λ alive(Anil)
  e. ∀x ¬ eats(Anil, x) V eats(Harry, x)
  f. ∀x ¬killed(x)] V alive(x)
  g. ∀x ¬ alive(x) V ¬ killed(x)

h. likes(John, Peanuts).

- **Rename variables or standardize variables**

  a. ∀x ¬ food(x) V likes(John, x)

  b. food(Apple) Λ food(vegetables)

  c. ∀y ∀z ¬ eats(y, z) V killed(y) V food(z)

  d. eats (Anil, Peanuts) Λ alive(Anil)

  e. ∀w¬ eats(Anil, w) V eats(Harry, w)

  f. ∀g ¬killed(g)] V alive(g)

  g. ∀k ¬ alive(k) V ¬ killed(k)

  h. likes(John, Peanuts).

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

  In this step, we will eliminate existential quantifier ∃, and this process is known as **Skolemization**. But in this example problem since there is no existential quantifier so all the statements will remain same in this step.

- **Drop Universal quantifiers.**

  In this step we will drop all universal quantifier since all the statements are not implicitly quantified so we don't need it.

  a. ¬ food(x) V likes(John, x)

  b. food(Apple)

  c. food(vegetables)

  d. ¬ eats(y, z) V killed(y) V food(z)

  e. eats (Anil, Peanuts)

  f. alive(Anil)

  g. ¬ eats(Anil, w) V eats(Harry, w)

  h. killed(g) V alive(g)

  i. ¬ alive(k) V ¬ killed(k)

  j. likes(John, Peanuts).

Note: Statements "food(Apple) Λ food(vegetables)" and "eats (Anil, Peanuts) Λ alive(Anil)" can be written in two separate statements.

- **Distribute conjunction Λ over disjunction ¬.**

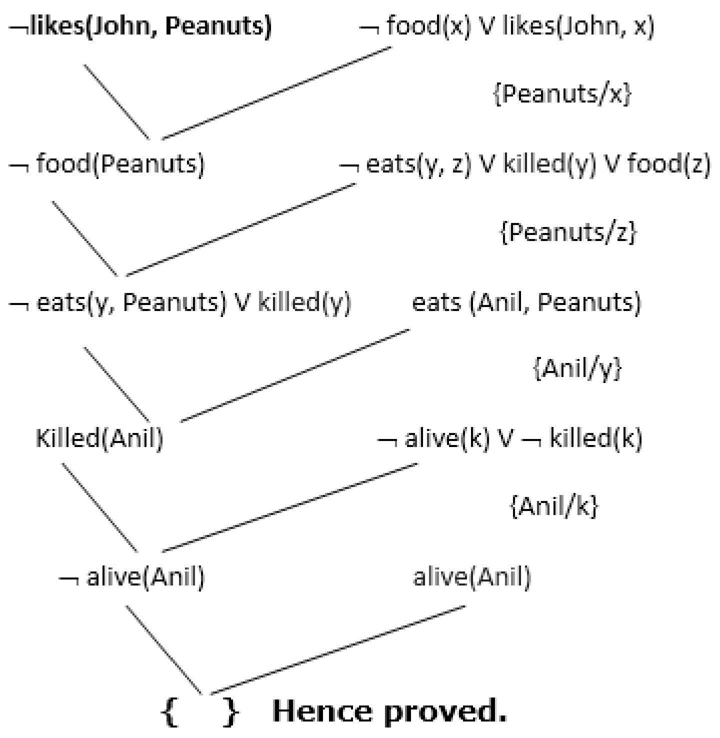
  This step will not make any change in this problem.

Step-3: Negate the statement to be proved

In this statement, we will apply negation to the conclusion statements, which will be written as ¬likes(John, Peanuts)

Step-4: Draw Resolution graph:

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



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

UNIT-3

Uncertainty:

Till now, we have learned knowledge representation using first-order logic and propositional logic with certainty, which means we were sure about the predicates. With this knowledge representation, we might write A→B, which means if A is true then B is true, but consider a situation where we are not sure about whether A is true or not then we cannot express this statement, this situation is called uncertainty.

Causes of uncertainty:

Following are some leading causes of uncertainty to occur in the real world.

1. Information occurred from unreliable sources.
2. Experimental Errors
3. Equipment fault
4. Temperature variation
5. Climate change.
6. Uncertain i/p
7. Uncertain knowledge

Probabilistic reasoning:

Probabilistic reasoning is a way of knowledge representation where we apply the concept of probability to indicate the uncertainty in knowledge. In probabilistic reasoning, we combine probability theory with logic to handle the uncertainty.

Need of probabilistic reasoning in AI:

- When there are unpredictable outcomes.
- When specifications or possibilities of predicates becomes too large to handle.
- When an unknown error occurs during an experiment.

In probabilistic reasoning, there are two ways to solve problems with uncertain knowledge:

- **Bayes' rule**
- **Bayesian Statistics**

As probabilistic reasoning uses probability and related terms, so before understanding probabilistic reasoning, let's understand some common terms:

Probability: Probability can be defined as a chance that an uncertain event will occur. It is the numerical measure of the likelihood that an event will occur. The value of probability always remains between 0 and 1 that represent ideal uncertainties.

1. $0 \leq P(A) \leq 1$, where P(A) is the probability of an event A.

1. $P(A) = 0$, indicates total uncertainty in an event A.

1. $P(A) = 1$, indicates total certainty in an event A.

We can find the probability of an uncertain event by using the below formula.

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

- $P(\neg A)$ = probability of a not happening event.
- $P(\neg A) + P(A) = 1$.

Event: Each possible outcome of a variable is called an event.

Sample space: The collection of all possible events is called sample space.

Random variables: Random variables are used to represent the events and objects in the real world.

Prior probability: The prior probability of an event is probability computed before observing new information.

Posterior Probability: The probability that is calculated after all evidence or information has taken into account. It is a combination of prior probability and new information.

Conditional probability:

Conditional probability is a probability of occurring an event when another event has already happened

Let's suppose, we want to calculate the event A when event B has already occurred, "the probability of A under the conditions of B", it can be written as:

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

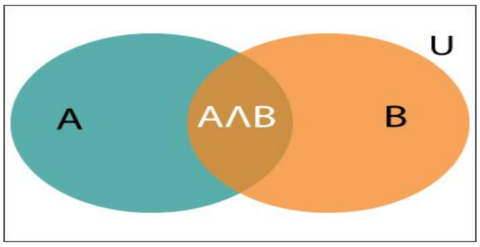
Where P(*A*⋀*B*)= Joint probability of a and B

P(B)= Marginal probability of B.

If the probability of A is given and we need to find the probability of B, then it will be given as:

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

It can be explained by using the below Venn diagram, where B is occurred event, so sample space will be reduced to set B, and now we can only calculate event A when event B is already occurred by dividing the probability of **P(A⋀*B*) by P(B)**.



Example:

In a class, there are 70% of the students who like English and 40% of the students who likes English and mathematics, and then what is the percent of students those who like English also like mathematics?

Solution:

Let, A is an event that a student likes Mathematics

B is an event that a student likes English.

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

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

Bayes' theorem:

Bayes' theorem is also known as **Bayes' rule, Bayes' law**, or **Bayesian reasoning**, which determines the probability of an event with uncertain knowledge.

In probability theory, it relates the conditional probability and marginal probabilities of two random events.

Bayes' theorem was named after the British mathematician **Thomas Bayes**. The **Bayesian inference** is an application of Bayes' theorem, which is fundamental to Bayesian statistics.

It is a way to calculate the value of P(B|A) with the knowledge of P(A|B).

Bayes' theorem allows updating the probability prediction of an event by observing new information of the real world.

Example: If cancer corresponds to one's age, then by using Bayes' theorem, we can determine the probability of cancer more accurately with the help of age.

Bayes' theorem can be derived using product rule and conditional probability of event A with known event B:

As from product rule we can write:

1. P(A ⋀ B)= P(A|B) P(B) or

Similarly, the probability of event B with known event A:

1. P(A ⋀ B)= P(B|A) P(A)

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

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

The above equation (a) is called as **Bayes' rule** or **Bayes' theorem**. This equation is basic of most modern AI systems for **probabilistic inference**.

It shows the simple relationship between joint and conditional probabilities. Here,

P(A|B) is known as **posterior**, which we need to calculate, and it will be read as Probability of hypothesis A when we have occurred an evidence B.

P(B|A) is called the likelihood, in which we consider that hypothesis is true, then we calculate the probability of evidence.

P(A) is called the **prior probability**, probability of hypothesis before considering the evidence

P(B) is called **marginal probability**, pure probability of an evidence.

In the equation (a), in general, we can write P (B) = P(A)*P(B|Ai), hence the Bayes' rule can be written as:

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

Where $A_1, A_2, A_3, \ldots\ldots, A_n$ is a set of mutually exclusive and exhaustive events.

Applying Bayes' rule:

Bayes' rule allows us to compute the single term P(B|A) in terms of P(A|B), P(*B*), and P(A). This is very useful in cases where we have a good probability of these three terms and want to determine the fourth one. Suppose we want to perceive the effect of some unknown cause, and want to compute that cause, then the Bayes' rule becomes:

$$\mathbf{P(cause|effect)} = \frac{P(effect|cause)\, P(cause)}{P(effect)}$$

Example-1:

Question: what is the probability that a patient has diseases meningitis with a stiff neck?

Given Data:

A doctor is aware that disease meningitis causes a patient to have a stiff neck, and it occurs 80% of the time. He is also aware of some more facts, which are given as follows:

- The Known probability that a patient has meningitis disease is 1/30,000.
- The Known probability that a patient has a stiff neck is 2%.

Let a be the proposition that patient has stiff neck and b be the proposition that patient has meningitis. , so we can calculate the following as:

P(a|b) = 0.8

P(b) = 1/30000

P(a)= .02

$$\mathbf{P(b|a)} = \frac{P(a|b)P(b)}{P(a)} = \frac{0.8*(\frac{1}{30000})}{0.02} = 0.001333333.$$

Hence, we can assume that 1 patient out of 750 patients has meningitis disease with a stiff neck.

Example-2:

Question: From a standard deck of playing cards, a single card is drawn. The probability that the card is king is 4/52, then calculate posterior probability P(King|Face), which means the drawn face card is a king card.

Solution:

$$\mathbf{P(king|face)} = \frac{P(Face|king) * P(King)}{P(Face)} \quad ....(i)$$

P(king): probability that the card is King= 4/52= 1/13

P(face): probability that a card is a face card= 3/13

P(Face|King): probability of face card when we assume it is a king = 1

Putting all values in equation (i) we will get:

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

Application of Bayes' theorem in Artificial intelligence:

- It is used to calculate the next step of the robot when the already executed step is given.
- Bayes' theorem is helpful in weather forecasting.
- It can solve the Monty Hall problem.

Bayesian Belief Network

Bayesian belief network is key computer technology for dealing with probabilistic events and to solve a problem which has uncertainty. We can define a Bayesian network as:

"A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph."

It is also called a **Bayes network, belief network, decision network**, or **Bayesian model**.

Bayesian networks are probabilistic, because these networks are built from a **probability distribution**, and also use probability theory for prediction and anomaly detection.

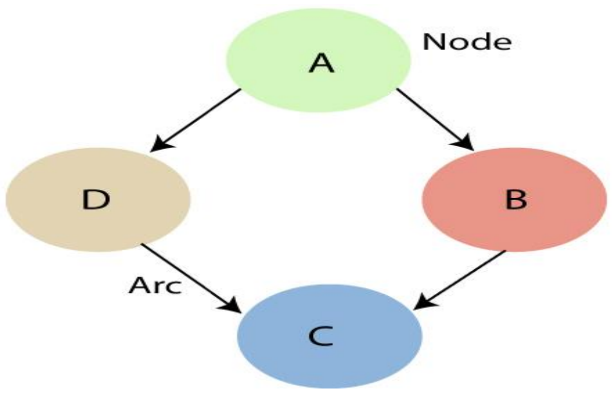
Real world applications are probabilistic in nature, and to represent the relationship between multiple events, we need a Bayesian network. It can also be used in various tasks including **prediction, anomaly detection, diagnostics, automated insight, reasoning, time series prediction**, and **decision making under uncertainty**.

Bayesian Network can be used for building models from data and experts opinions, and it consists of two parts:

- **Directed Acyclic Graph**
- **Table of conditional probabilities.**

The generalized form of Bayesian network that represents and solve decision problems under uncertain knowledge is known as an **Influence diagram**.

A Bayesian network graph is made up of nodes and Arcs (directed links), where:



- Each **node** corresponds to the random variables, and a variable can be **continuous** or **discrete**.
- **Arc or directed arrows** represent the causal relationship or conditional probabilities between random variables. These directed links or arrows connect the pair of nodes in the graph. These links represent that one node directly influence the other node, and if there is no directed link that means that nodes are independent with each other
  - **In the above diagram, A, B, C, and D are random variables represented by the nodes of the network graph.**
  - **If we are considering node B, which is connected with node A by a directed arrow, then node A is called the parent of Node B.**
  - **Node C is independent of node A.**

***Note: The Bayesian network graph does not contain any cyclic graph. Hence, it is known as a** directed acyclic graph or DAG.*

The Bayesian network has mainly two components:

- **Causal Component**
- **Actual numbers**

Each node in the Bayesian network has condition probability distribution **$P(X_i | Parent(X_i))$**, which determines the effect of the parent on that node.

Bayesian network is based on Joint probability distribution and conditional probability. So, let's first understand the joint probability distribution:

Joint probability distribution:

If we have variables x1, x2, x3...., xn, then the probabilities of a different combination of x1, x2, x3. xn, are known as Joint probability distribution.

$P[x_1, x_2, x_3,....., x_n]$, it can be written as the following way in terms of the joint probability distribution.

$= P[x_1 | x_2, x_3,....., x_n]P[x_2, x_3,....., x_n]$

$= P[x_1 | x_2, x_3,....., x_n]P[x_2 | x_3,....., x_n]....P[x_{n-1} | x_n]P[x_n]$.

In general for each variable Xi, we can write the equation as:

$P(X_i | X_{i-1},........., X_1) = P(X_i | Parents(X_i))$

Explanation of Bayesian network:

Let's understand the Bayesian network through an example by creating a directed acyclic graph:

Example: Harry installed a new burglar alarm at his home to detect burglary. The alarm reliably responds at detecting a burglary but also responds for minor earthquakes. Harry has two neighbours David and Sophia, who have taken a responsibility to inform Harry at work when they hear the alarm. David always calls Harry when he hears the alarm, but sometimes he got confused with the phone ringing and calls at that time too. On the other hand, Sophia likes to listen to high music, so sometimes she misses to hear the alarm. Here we would like to compute the probability of Burglary Alarm.

Problem:

Calculate the probability that alarm has sounded, but there is neither a burglary, nor an earthquake occurred, and David and Sophia both called the Harry.

Solution:

- The Bayesian network for the above problem is given below. The network structure is showing that burglary and earthquake is the parent node of the alarm and directly affecting the probability of alarm's going off, but David and Sophia's calls depend on alarm probability.
- The network is representing that our assumptions do not directly perceive the burglary and also do not notice the minor earthquake, and they also not confer before calling.
- The conditional distributions for each node are given as conditional probabilities table or CPT.
- Each row in the CPT must be sum to 1 because all the entries in the table represent an exhaustive set of cases for the variable.
- In CPT, a Boolean variable with k Boolean parents contains 2^K probabilities. Hence, if there are two parents, then CPT will contain 4 probability values

List of all events occurring in this network:

- **Burglary (B)**
- **Earthquake(E)**
- **Alarm(A)**
- **David Calls(D)**
- **Sophia calls(S)**

We can write the events of problem statement in the form of probability: **P[D, S, A, B, E]**, can rewrite the above probability statement using joint probability distribution:

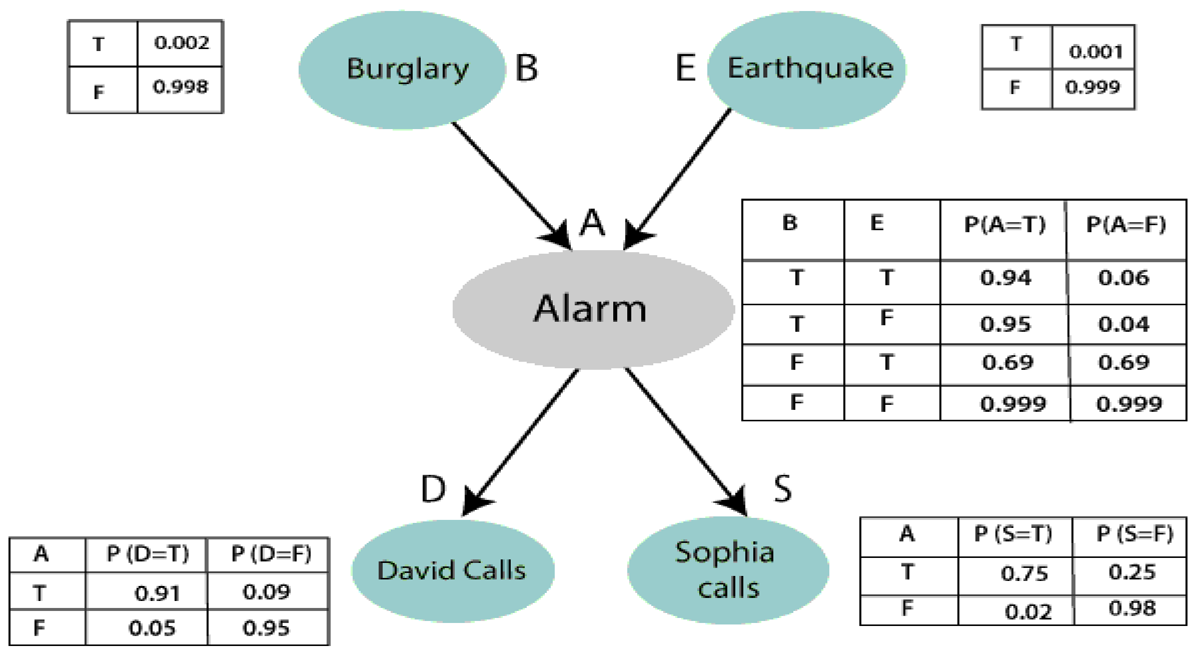
P[D, S, A, B, E]= P[D | S, A, B, E]. P[S, A, B, E]

=P[D | S, A, B, E]. P[S | A, B, E]. P[A, B, E]

= P [D| A]. P [S| A, B, E]. P[A, B, E]

= P[D | A]. P[S | A]. P[A| B, E]. P[B, E]

= P[D | A]. P[S | A]. P[A| B, E]. P[B |E]. P[E]



Let's take the observed probability for the Burglary and earthquake component:

P(B= True) = 0.002, which is the probability of burglary.

P(B= False)= 0.998, which is the probability of no burglary.

P(E= True)= 0.001, which is the probability of a minor earthquake

P(E= False)= 0.999, Which is the probability that an earthquake not occurred.

We can provide the conditional probabilities as per the below tables:

Conditional probability table for Alarm A:

The Conditional probability of Alarm A depends on Burglar and earthquake:

B	E	P(A= True)	P(A= False)
True	True	0.94	0.06
True	False	0.95	0.04
False	True	0.31	0.69
False	False	0.001	0.999

Conditional probability table for David Calls:

The Conditional probability of David that he will call depends on the probability of Alarm.

A	P(D= True)	P(D= False)
True	0.91	0.09
False	0.05	0.95

Conditional probability table for Sophia Calls:

The Conditional probability of Sophia that she calls is depending on its Parent Node "Alarm."

A	P(S= True)	P(S= False)
True	0.75	0.25
False	0.02	0.98

From the formula of joint distribution, we can write the problem statement in the form of probability distribution:

P(S, D, A, ¬B, ¬E) = P (S|A) *P (D|A)*P (A|¬B ^ ¬E) *P (¬B) *P (¬E).

= 0.75* 0.91* 0.001* 0.998*0.999

= 0.00068045.

Hence, a Bayesian network can answer any query about the domain by using Joint distribution.

The semantics of Bayesian Network:

There are two ways to understand the semantics of the Bayesian network, which is given below:

1. To understand the network as the representation of the Joint probability distribution.

It is helpful to understand how to construct the network.

2. To understand the network as an encoding of a collection of conditional independence statements.

Unit-4

Statistical Learning

It is based on the learning of uncertainty in real environment

The methods probability & decision theory are used to handle uncertainty by the agents

First agent must learn its probabilistic theories of the world from experience

A Bayesian view of learning is extremely powerful providing general solution to the problem of noise, overthinking & optimal predictions

Statistical Learning methods

Statistical learning is about inferences

The idea is general from the data & hypothesis & there are called as key terms of statistical learning

Data(sample & population) are evidences

Learning with Complete Data

Learning with complete data in AI typically refers to situations where the dataset used to train a model contains all the necessary and relevant information without missing values, ambiguities, or inconsistencies. In such cases, the focus shifts from handling incomplete or noisy data to optimizing the learning process for accuracy, generalization, and efficiency. Here's a breakdown of the concept and approaches involved:

Complete data means:

- **No Missing Values:** All features (columns) have values for every instance (row) in the dataset.
- **Label Availability (Supervised Learning):** For labeled datasets, all training examples have corresponding output labels.
- **Proper Representation:** The dataset represents the problem space adequately, covering various scenarios and edge cases.

Complete data allows AI algorithms to focus on learning patterns and relationships without being sidetracked by data imputation, augmentation, or noise handling.

Advantages of Learning with Complete Data

- **Higher Accuracy Potential:** Models trained on complete data are less prone to biases caused by missing information.
- **Faster Training Times:** No need for preprocessing steps like data cleaning or imputation.
- **Simpler Architectures:** Avoids complexities introduced by techniques like attention for handling noise or GANs for data generation.

Learning with Hidden Data

Learning with hidden data in AI refers to scenarios where parts of the dataset are unavailable, unobservable, or incomplete during training. Hidden data can pose challenges for training algorithms but also opens up opportunities to develop robust, adaptive, and efficient models. Here's a comprehensive overview:

Hidden data refers to:

- **Missing Values:** Some features or labels are absent for certain instances.
- **Latent Variables:** Underlying factors influencing the data are not explicitly observed (e.g., user intent).
- **Partial Observability:** In reinforcement learning or time-series tasks, the complete state information may not be available.
- **Privacy Constraints:** Some data is deliberately withheld for security or privacy reasons (e.g., anonymized datasets).

Strategies for Learning with Hidden Data

A. Data Imputation

Filling in missing data with plausible values:

- **Statistical Methods:** Mean, median, or mode substitution.
- **Machine Learning:** K-Nearest Neighbors (KNN), regression models.

- **Deep Learning:** Autoencoders or Variational Autoencoders (VAEs) for imputing complex datasets.

B. Probabilistic Models

Utilizing models that explicitly handle hidden variables:

- **Bayesian Networks:** Encode dependencies and estimate distributions.
- **Hidden Markov Models (HMM):** For time-series data with unobservable states.
- **Latent Dirichlet Allocation (LDA):** Extracting hidden topics from text data.

C. Semi-Supervised Learning

Using a mix of labeled and unlabeled data:

- **Self-Training:** Train a model on labeled data, then iteratively label and learn from unlabeled data.
- **Graph-Based Methods:** Leverage graph structures to infer hidden labels.

D. Generative Models

Generating synthetic data to fill gaps:

- **GANs (Generative Adversarial Networks):** Create realistic samples for missing or unobservable data.
- **VAEs:** Learn latent representations and reconstruct missing elements.

E. Reinforcement Learning

Address partial observability with:

- **Partially Observable Markov Decision Processes (POMDPs):** For environments where full states aren't available.
- **Recurrent Architectures:** Use memory (e.g., LSTMs or GRUs) to learn from sequences with hidden states.

1.What is Estimation

Estimate our expectation from machines then classify the data into some classes.

2.what is maximization

whatever age estimate should be maximized

3.convergence

Estimation algorithm

It is an iterative estimation algorithm that can drive the maximum likelihood estimation in the presence of missing data

e.g.:

let’s assume we have 2 coins c1 & c2

q1.probability of getting HEAD with C1

C1 – H T H	2(H)	1(T)
C2 – T H H	1(H)	2(H)
C1 – T H T	1(H)	2(T)

3/6

Learning by Taking Advice(learning from taking guidance)

Learning by taking advice in AI refers to integrating external guidance, rules, or expert knowledge into the learning process to improve efficiency, accuracy, or alignment with specific goals. This approach is particularly useful when:

- Training data is limited or noisy.
- Incorporating domain-specific expertise can enhance the model's performance.
- Ensuring alignment with ethical or regulatory standards is critical.

Types of Advice:

1. **Rule-Based Advice:** Explicit constraints (e.g., "if A, then B").
2. **Example-Based Advice:** Demonstrative guidance through specific cases or labeled examples.
3. **Reward-Based Advice:** In reinforcement learning, shaping rewards to guide behavior (e.g., penalties for undesired actions).
4. **Interactive Advice:** Real-time feedback from a human during training.

Benefits of Learning by Taking Advice

- **Faster Learning:** Incorporating guidance can reduce the data or time needed to train a model.
- **Better Generalization:** Advice can serve as a regularizer, steering the model toward meaningful solutions.
- **Domain Knowledge Integration:** Encodes expertise that may not be evident from data alone.
- **Improved Safety and Fairness:** Helps align AI behavior with human values and societal norms.

e.g:

teacher -> student

programmer -> machine

Learning by Problem Solving

We have to train/pre-program the machine by solving the problem as human does the solving. We have to make sure that the AI machine should have the right/proper approach that human apply towards a problem while solving a problem as AI machine is going to replace human labour.

First, we have to program those approaches & then put it in machine, then have to watch how the machine is performing , as directed by the I/P or diverting from those procedures. If our I/P, data ,program, procedures are best as we are expecting then the result will achieve 100% by the machine as we are expecting.

Learning from Examples

We have to explain/make understand the machine by giving the examples. As human teaches the student or to his child by giving examples for the better understanding towards a concept or a problem. We have to follow those approaches to out AI machine for better understandability & performance and to handle environment situation, means machine can perform what environment is expecting.

Learning from induction

Learning from induction in AI refers to the process of generalizing from specific instances or examples to derive broader rules, patterns, or principles. Inductive learning is foundational to many AI techniques, where the goal is to infer a general function, relationship, or model from observed data.

E.g.: different cats in different place having different colour

Characteristics of Inductive Learning

- **Data-Driven:** Requires a dataset of examples to infer patterns.
- **Probabilistic:** Inductive conclusions are uncertain and depend on the quality and representativeness of the data.
- **Iterative:** Often involves refining the model as new examples are added.

Applications

A. Natural Language Processing (NLP)

- **Text Classification:** Generalizing from labeled text examples to classify new documents.
- **Language Models:** Learning grammar rules from sequences of text.

B. Computer Vision

- **Object Recognition:** Identifying objects in images based on labeled training examples.
- **Image Segmentation:** Generalizing pixel groupings from annotated images.

C. Predictive Analytics

- **Forecasting:** Predicting future trends based on historical data.
- **Fraud Detection:** Learning patterns from past fraudulent and legitimate transactions.

D. Robotics

- Inferring tasks or behaviors from demonstrations (e.g., learning a manipulation skill by observing humans).

Learning from observation & discovery

Learning from observation and discovery in AI refers to the process of understanding patterns, rules, or structures by passively observing data or interactions, without direct supervision or predefined goals. This approach enables models to infer relationships, make discoveries, or gain insights independently, making it foundational to unsupervised learning, self-supervised learning, and exploratory AI.

Learning from Observation

- **Passive Learning:** The AI system observes data, interactions, or environments to infer patterns or behaviors.
- **Real-World Examples:** AI learning from images, videos, or logs without explicit instructions.
- **Focus:** Extracting representations, clustering, detecting anomalies, or building associations.

Discovery

- **Uncovering Hidden Patterns:** Identifying structures, causal relationships, or novel insights from data.
- **Exploration:** Actively testing or simulating scenarios to learn about the environment.
- **Applications:** Scientific discovery, hypothesis generation, knowledge graphs.

Learning by analogy

 Definition: Learning by analogy involves using knowledge from a familiar source domain to solve problems or understand concepts in a target domain.

 Core Idea: Identify and apply correspondences between the two domains to reason about new situations.

 Human Inspiration: Humans frequently use analogies (e.g., comparing the flow of electricity to water flow) to understand and solve problems.

Key Components of Analogical Learning

1. **Source Domain:** The domain where knowledge is already established.
2. **Target Domain:** The new, unknown domain where the analogy is applied.
3. **Mapping:** The process of finding correspondences between the source and target domains.
4. **Transfer:** Using the mapping to apply knowledge, rules, or structures from the source domain to the target domain.

How AI Learns by Analogy

A. Case-Based Reasoning (CBR)

- AI systems solve new problems by adapting solutions from similar past problems.
- Example: Diagnosing diseases in patients by comparing symptoms to prior cases.

B. Structural Mapping Theory

- AI uses algorithms like the Structure-Mapping Engine (SME) to align and map relations between domains.
- Focuses on abstract relationships rather than surface features.

C. Analogical Transfer in Neural Networks

- Modern AI models, such as transformers, can implicitly learn analogies by identifying patterns in large datasets.
- Example: A language model might understand analogies like "King is to Queen as Man is to Woman" through word embeddings.

D. Rule-Based Systems

- Incorporate predefined analogical rules to solve problems or make predictions.
- Example: Comparing historical economic data to current trends to predict market behaviour.

Rote Learning

Rote learning is the process of memorizing specific new items as they are encountered. The basic idea is simple and easy to realize within a computer program: Each time a new and useful piece of information is encountered, it is stored away for future use.

For example, an AI system might be designed to recognize faces by extracting a variety of features (such as distance between the eyes) from an image and searching for a match within a database of 1000 stored feature sets. If it finds a match, it has recognized the person; if not, it reports “unknown person.” In this latter case, the person or some human operator can provide the necessary details of this unknown person, and the system can enter the details in its database so that next time this person is presented to the system he or she will be correctly recognized.

However, every rote learning event increases the size of the database, and speed of recognition (as well as accuracy if new faces are similar to old ones) will decrease as the size of the database grows.

Depending on the application of this system, certain new faces may be seen once and never again, in which case rote learning wastes time and space.
Alternatively, this new face may be commonly seen at first but may over time become rare and ultimately no longer seen. In the first case, rote learning is immediately detrimental to system performance, and in the second case it becomes detrimental over time.
Rote learning (and in general any sort of learning) implies some compensating mechanism of forgetting, which suggests that the apparent human weakness of imperfect recall may be a fundamental necessity to long-term intelligence. AI has largely been preoccupied with learning mechanisms usually without the compensating mechanism of forgetting.

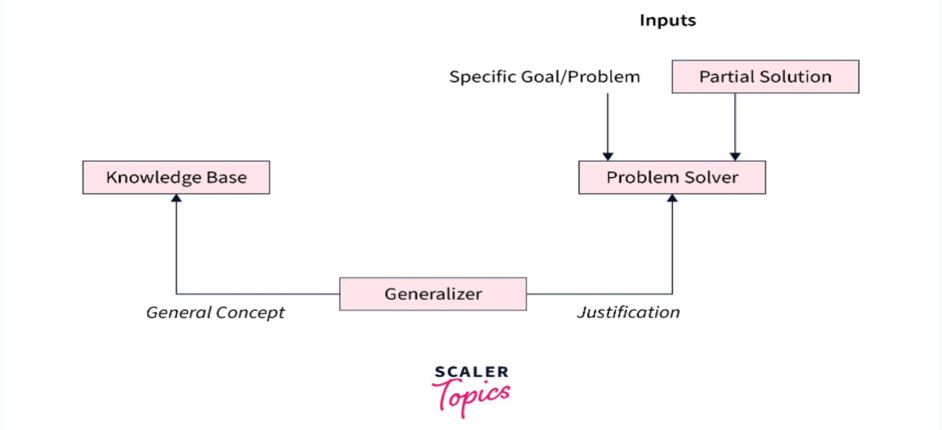
The second general point is that most situations change over time, so an AI system with any longevity will have to be able to track and react appropriately to the natural drifts in the problem domain. This is not an easy problem to solve. The fundamental difficulty is one of distinguishing between short-term variation and long-term drift when both short term and long term are ill-defined context-dependent quantities. If after appearing twice a day, a face is not seen for a week, should it be deleted? There is no correct answer and probably no more than guidelines even if the full details of the application of this face recognition system are known.

Therefore, a good rote learning system must exhibit flexibility and sophistication. It cannot simply store every new event and delete every one that is not used during some specified period of time. Human intelligence appears to embody some mechanisms of partial recall that, like forgetting, may not be weaknesses; for example, learned faces may gradually fade over time if not used but may be more quickly and easily reestablished if they reappear before they have been totally forgotten.

Explanation-Based Learning

Explanation-based learning in artificial intelligence is a problem-solving method that involves agent learning by analysing specific situations and connecting them to previously acquired information. Also, the agent applies what he has learned to solve similar issues. Rather than relying solely on statistical analysis, EBL algorithms incorporate logical reasoning and domain knowledge to make predictions and identify patterns.

Explanation-based learning architecture:



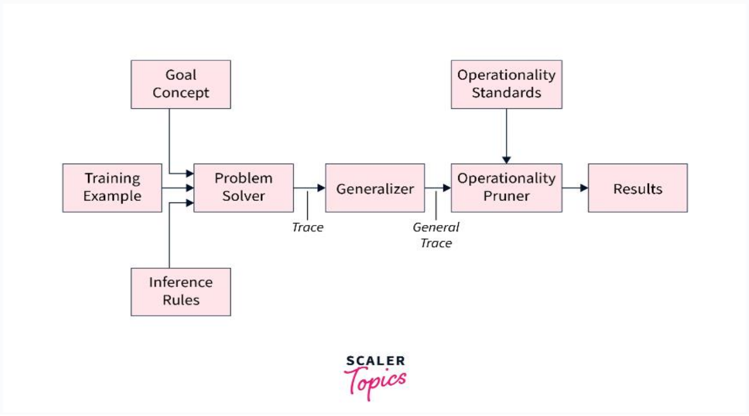
The environment provides two inputs to the EBL architecture:

1. **A specific goal**, and
2. **A partial solution**.

The problem solver analyses these sources and provides reasoning to the generalizer.

The generalizer uses general ideas from the knowledge base as input and compares them to the problem solver's reasoning to come up with an answer to the given problem.

Explanation-based learning System Representation:



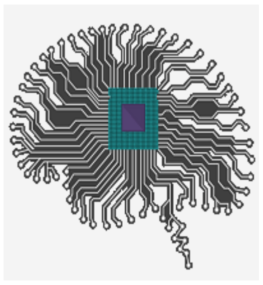
- **Problem Solver:** It takes 3 kinds of external inputs: The goal idea is a complex problem statement that the agent must learn. Training instances are facts that illustrate a specific instance of a target idea. Inference rules reflect facts and procedures that demonstrate what the learner already understands.
- **Generalizer:** The problem solver's output is fed into the generalizer, which compares the problem solver's explanation to the knowledge base and outputs to the operational pruner.
- **Operational pruner:** It takes two inputs, one from generalized and the other from operationally standard. The operational standard describes the final concept and defines the format in which the learned concept should be conveyed.

Examples of Explanation-Based Learning

- **Medical Diagnosis:** Explanation-based learning can be used in medical diagnosis to determine the underlying causes of a patient's symptoms. Explanation-based learning algorithms can find trends and produce more accurate diagnoses by analysing previously diagnosed instances.
- **Robot Navigation:** Explanation-based learning may be used to educate robots on how to navigate through complicated settings. Explanation-based learning algorithms can discover the rules and principles that were utilized to navigate those settings and apply them to new scenarios by analysing prior successful navigation efforts.
- **Fraud Detection:** Explanation-based learning may be utilized in fraud detection to discover patterns of fraudulent conduct. Explanation-based learning algorithms can

find the rules and principles that were utilized to detect prior cases of fraud and apply them to new cases by analysing previous incidents of fraud.

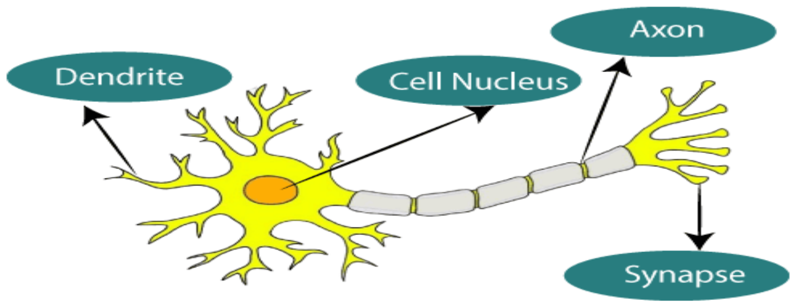
Artificial Neural Network



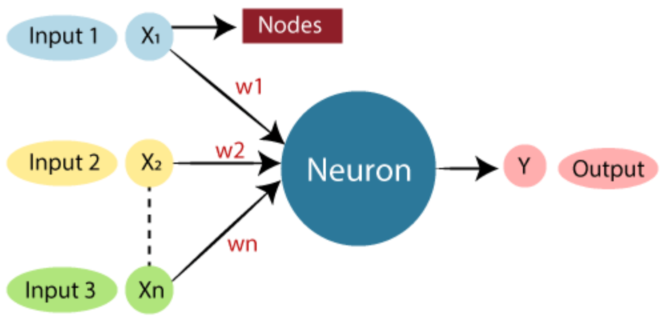
The term "Artificial neural network" refers to a biologically inspired sub-field of artificial intelligence modelled after the brain. An Artificial neural network is usually a computational network based on biological neural networks that construct the structure of the human brain. Similar to a human brain has neurons interconnected to each other, artificial neural networks also have neurons that are linked to each other in various layers of the networks. These neurons are known as nodes.

The term "**Artificial Neural Network**" is derived from Biological neural networks that develop the structure of a human brain. Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks. These neurons are known as nodes.

The given figure illustrates the typical diagram of Biological Neural Network.



The typical Artificial Neural Network looks something like the given figure.



Dendrites from Biological Neural Network represent inputs in Artificial Neural Networks, cell nucleus represents Nodes, synapse represents Weights, and Axon represents Output.

Relationship between Biological neural network and artificial neural network:

Biological Neural Network	Artificial Neural Network
Dendrites	Inputs
Cell nucleus	Nodes
Synapse	Weights
Axon	Output

An **Artificial Neural Network** in the field of **Artificial intelligence** where it attempts to mimic the network of neurons makes up a human brain so that computers will have an option to understand things and make decisions in a human-like manner. The artificial neural network is designed by programming computers to behave simply like interconnected brain cells.

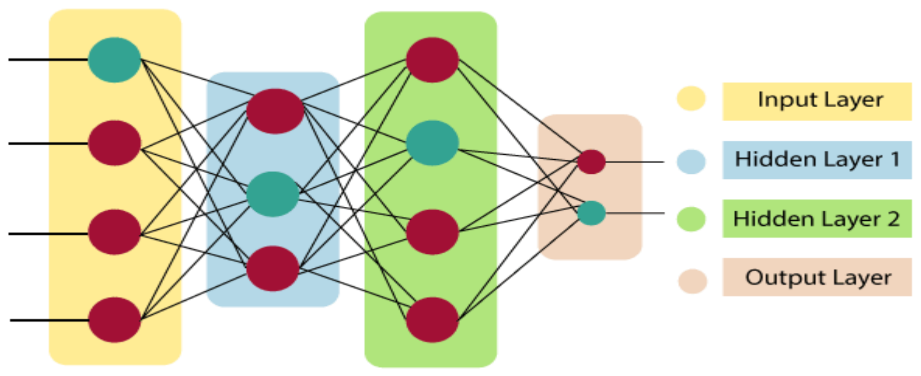
There are around 1000 billion neurons in the human brain. Each neuron has an association point somewhere in the range of 1,000 and 100,000. In the human brain, data is stored in such a manner as to be distributed, and we can extract more than one piece of this data, when necessary, from our memory parallelly. We can say that the human brain is made up of incredibly amazing parallel processors.

We can understand the artificial neural network with an example, consider an example of a digital logic gate that takes an input and gives an output. "OR" gate, which takes two inputs. If one or both the inputs are "On," then we get "On" in output. If both the inputs are "Off," then we get "Off" in output. Here the output depends upon input. Our brain does not perform the same task. The outputs to inputs relationship keep changing because of the neurons in our brain, which are "learning."

The architecture of an artificial neural network:

To understand the concept of the architecture of an artificial neural network, we have to understand what a neural network consists of. In order to define a neural network that consists of a large number of artificial neurons, which are termed units arranged in a sequence of layers. Let's us look at various types of layers available in an artificial neural network.

Artificial Neural Network primarily consists of **three** layers:



Input Layer:

As the name suggests, it accepts inputs in several different formats provided by the programmer.

Hidden Layer:

The hidden layer presents in-between input and output layers. It performs all the calculations to find hidden features and patterns.

Output Layer:

The input goes through a series of transformations using the hidden layer, which finally results in output that is conveyed using this layer.

The artificial neural network takes input and computes the weighted sum of the inputs and includes a bias. This computation is represented in the form of a transfer function.

$$\sum_{i=1}^{n} Wi * Xi + b$$

It determines weighted total is passed as an input to an activation function to produce the output. Activation functions choose whether a node should fire or not. Only those who are fired make it to the output layer. There are distinctive activation functions available that can be applied upon the sort of task we are performing.

Advantages of Artificial Neural Network (ANN)

Parallel processing capability:

Artificial neural networks have a numerical value that can perform more than one task simultaneously.

Storing data on the entire network:

Data that is used in traditional programming is stored on the whole network, not on a database. The disappearance of a couple of pieces of data in one place doesn't prevent the network from working.

Capability to work with incomplete knowledge:

After ANN training, the information may produce output even with inadequate data. The loss of performance here relies upon the significance of missing data.

Having a memory distribution:

For ANN is to be able to adapt, it is important to determine the examples and to encourage the network according to the desired output by demonstrating these examples to the network. The succession of the network is directly proportional to the chosen instances, and if the event can't appear to the network in all its aspects, it can produce false output.

Having fault tolerance:

Extortion of one or more cells of ANN does not prohibit it from generating output, and this feature makes the network fault-tolerance.

Disadvantages of Artificial Neural Network:

Assurance of proper network structure:

There is no particular guideline for determining the structure of artificial neural networks. The appropriate network structure is accomplished through experience, trial, and error.

Unrecognized behaviour of the network:

It is the most significant issue of ANN. When ANN produces a testing solution, it does not provide insight concerning why and how. It decreases trust in the network.

Hardware dependence:

Artificial neural networks need processors with parallel processing power, as per their structure. Therefore, the realization of the equipment is dependent.

Difficulty of showing the issue to the network:

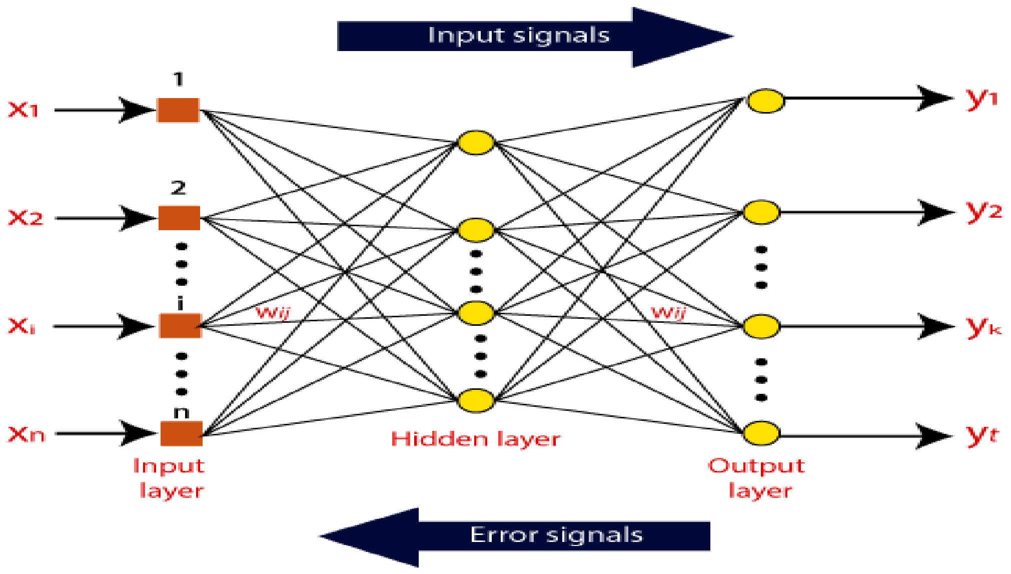
ANNs can work with numerical data. Problems must be converted into numerical values before being introduced to ANN. The presentation mechanism to be resolved here will directly impact the performance of the network. It relies on the user's abilities.

The duration of the network is unknown:

The network is reduced to a specific value of the error, and this value does not give us optimum results.

How do artificial neural networks work?

Artificial Neural Network can be best represented as a weighted directed graph, where the artificial neurons form the nodes. The association between the neurons outputs and neuron inputs can be viewed as the directed edges with weights. The Artificial Neural Network receives the input signal from the external source in the form of a pattern and image in the form of a vector. These inputs are then mathematically assigned by the notations x(n) for every n number of inputs.



Afterward, each of the input is multiplied by its corresponding weights (these weights are the details utilized by the artificial neural networks to solve a specific problem). In general terms, these weights normally represent the strength of the interconnection between neurons inside the artificial neural network. All the weighted inputs are summarized inside the computing unit.

If the weighted sum is equal to zero, then bias is added to make the output non-zero or something else to scale up to the system's response. Bias has the same input, and weight equals to 1. Here the total of weighted inputs can be in the range of 0 to positive infinity. Here, to keep the response in the limits of the desired value, a certain maximum value is benchmarked, and the total of weighted inputs is passed through the activation function.

The activation function refers to the set of transfer functions used to achieve the desired output. There is a different kind of the activation function, but primarily either linear or non-linear sets of functions. Some of the commonly used sets of activation functions are the Binary, linear, and Tan hyperbolic sigmoidal activation functions. Let us take a look at each of them in details:

Types of Artificial Neural Network:

There are various types of Artificial Neural Networks (ANN) depending upon the human brain neuron and network functions, an artificial neural network similarly performs tasks. The majority of the artificial neural networks will have some similarities with a more complex biological partner and are very effective at their expected tasks. For example, segmentation or classification.

Feedback ANN:

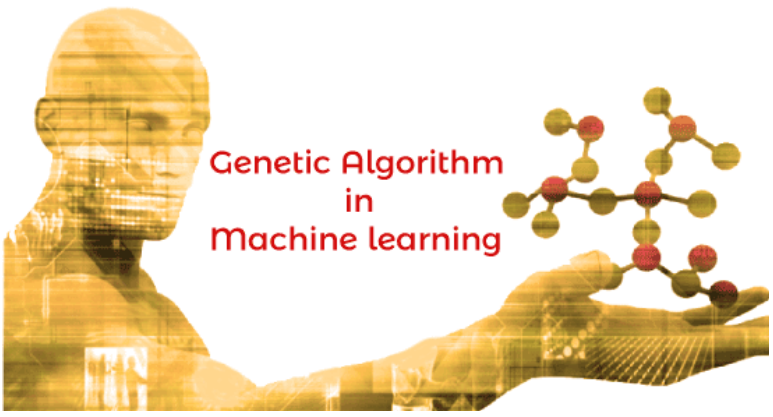
In this type of ANN, the output returns into the network to accomplish the best-evolved results internally. As per the **University of Massachusetts**, Lowell Centre for Atmospheric Research. The feedback networks feed information back into itself and are well suited to solve optimization issues. The Internal system error corrections utilize feedback ANNs.

Feed-Forward ANN:

A feed-forward network is a basic neural network comprising of an input layer, an output layer, and at least one layer of a neuron. Through assessment of its output by reviewing its input, the intensity of the network can be noticed based on group behaviour of the associated neurons, and the output is decided. The primary advantage of this network is that it figures out how to evaluate and recognize input patterns.

Genetic Learning

A genetic algorithm is an adaptive heuristic search algorithm inspired by "Darwin's theory of evolution in Nature." It is used to solve optimization problems in machine learning. It is one of the important algorithms as it helps solve complex problems that would take a long time to solve.



Genetic Algorithms are being widely used in different real-world applications, for example, **Designing electronic circuits, code-breaking, image processing, and artificial creativity.**

In this topic, we will explain Genetic algorithm in detail, including basic terminologies used in Genetic algorithm, how it works, advantages and limitations of genetic algorithm, etc.

What is a Genetic Algorithm?

Before understanding the Genetic algorithm, let's first understand basic terminologies to better understand this algorithm:

- **Population:** Population is the subset of all possible or probable solutions, which can solve the given problem.
- **Chromosomes:** A chromosome is one of the solutions in the population for the given problem, and the collection of gene generate a chromosome.
- **Gene:** A chromosome is divided into a different gene, or it is an element of the chromosome.
- **Allele:** Allele is the value provided to the gene within a particular chromosome.
- **Fitness Function:** The fitness function is used to determine the individual's fitness level in the population. It means the ability of an individual to compete with other individuals. In every iteration, individuals are evaluated based on their fitness function.
- **Genetic Operators:** In a genetic algorithm, the best individual mate to regenerate offspring better than parents. Here genetic operators play a role in changing the genetic composition of the next generation.
- **Selection**

After calculating the fitness of every existent in the population, a selection process is used to determine which of the individualities in the population will get to reproduce and produce the seed that will form the coming generation.

Types of selection styles available

- **Roulette wheel selection**
- **Event selection**
- **Rank- grounded selection**

So, now we can define a genetic algorithm as a heuristic search algorithm to solve optimization problems. It is a subset of evolutionary algorithms, which is used in computing. A genetic algorithm uses genetic and natural selection concepts to solve optimization problems.

How Genetic Algorithm Work?

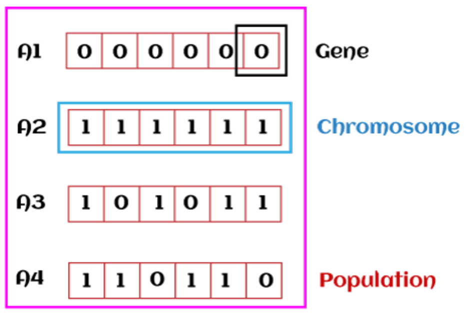
The genetic algorithm works on the evolutionary generational cycle to generate high-quality solutions. These algorithms use different operations that either enhance or replace the population to give an improved fit solution.

It basically involves five phases to solve the complex optimization problems, which are given as below:

- **Initialization**
- **Fitness Assignment**
- **Selection**
- **Reproduction**
- **Termination**

1. Initialization

The process of a genetic algorithm starts by generating the set of individuals, which is called population. Here each individual is the solution for the given problem. An individual contains or is characterized by a set of parameters called Genes. Genes are combined into a string and generate chromosomes, which is the solution to the problem. One of the most popular techniques for initialization is the use of random binary strings.



2. Fitness Assignment

Fitness function is used to determine how fit an individual is? It means the ability of an individual to compete with other individuals. In every iteration, individuals are evaluated based on their fitness function. The fitness function provides a fitness score to each individual. This score further determines the probability of being selected for reproduction. The high the fitness score, the more chances of getting selected for reproduction.

3. Selection

The selection phase involves the selection of individuals for the reproduction of offspring. All the selected individuals are then arranged in a pair of two to increase reproduction. Then these individuals transfer their genes to the next generation.

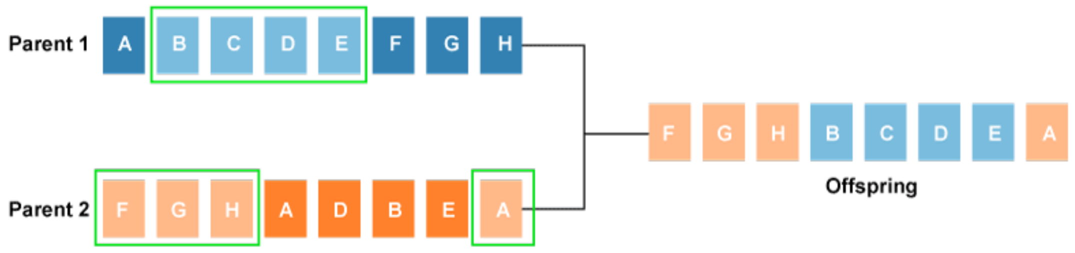
There are three types of Selection methods available, which are:

- Roulette wheel selection
- Tournament selection
- Rank-based selection

4. Reproduction

After the selection process, the creation of a child occurs in the reproduction step. In this step, the genetic algorithm uses two variation operators that are applied to the parent population. The two operators involved in the reproduction phase are given below:

- **Crossover:** The crossover plays a most significant role in the reproduction phase of the genetic algorithm. In this process, a crossover point is selected at random within the genes. Then the crossover operator swaps genetic information of two parents from the current generation to produce a new individual representing the offspring.

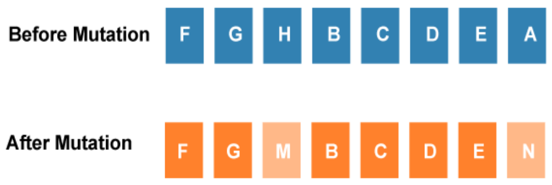
  

  The genes of parents are exchanged among themselves until the crossover point is met. These newly generated offspring are added to the population. This process is also called or crossover. Types of crossover styles available:

  - One point crossover
  - Two-point crossover
  - Livery crossover
  - Inheritable Algorithms crossover

- **Mutation**
  The mutation operator inserts random genes in the offspring (new child) to maintain the diversity in the population. It can be done by flipping some bits in the chromosomes.
  Mutation helps in solving the issue of premature convergence and enhances diversification. The below image shows the mutation process:
  Types of mutation styles available,

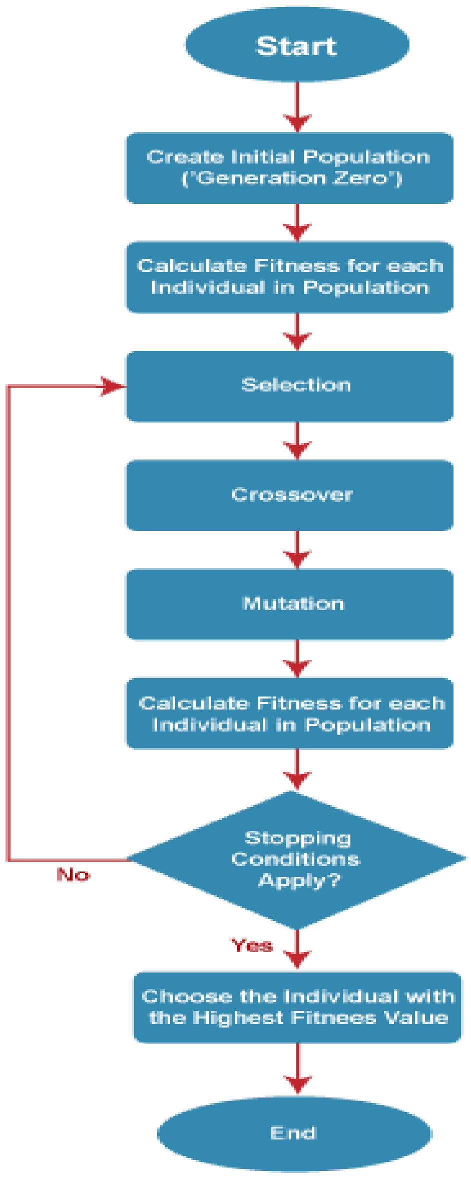
  - **Flip bit mutation**
  - **Gaussian mutation**
  - **Exchange/Swap mutation**



5. Termination

After the reproduction phase, a stopping criterion is applied as a base for termination. The algorithm terminates after the threshold fitness solution is reached. It will identify the final solution as the best solution in the population.

General Workflow of a Simple Genetic Algorithm


Start
Create Initial Population ('Generation Zero')
Calculate Fitness for each Individual in Population
Selection
Crossover
Mutation
Calculate Fitness for each Individual in Population
Stopping Conditions Apply?
No
Yes
Choose the Individual with the Highest Fitnees Value
End

Advantages of Genetic Algorithm

- The parallel capabilities of genetic algorithms are best.
- It helps in optimizing various problems such as discrete functions, multi-objective problems, and continuous functions.
- It provides a solution for a problem that improves over time.
- A genetic algorithm does not need derivative information.

Limitations of Genetic Algorithms

- Genetic algorithms are not efficient algorithms for solving simple problems.
- It does not guarantee the quality of the final solution to a problem.
- Repetitive calculation of fitness values may generate some computational challenges.

Difference between Genetic Algorithms and Traditional Algorithms

- A search space is the set of all possible solutions to the problem. In the traditional algorithm, only one set of solutions is maintained, whereas, in a genetic algorithm, several sets of solutions in search space can be used.
- Traditional algorithms need more information in order to perform a search, whereas genetic algorithms need only one objective function to calculate the fitness of an individual.
- Traditional Algorithms cannot work parallelly, whereas genetic Algorithms can work parallelly (calculating the fitness of the individualities are independent).
- One big difference in genetic Algorithms is that rather of operating directly on seeker results, inheritable algorithms operate on their representations (or rendering), frequently appertained to as chromosomes.
- One of the big differences between traditional algorithm and genetic algorithm is that it does not directly operate on candidate solutions.
- Traditional Algorithms can only generate one result in the end, whereas Genetic Algorithms can generate multiple optimal results from different generations.
- The traditional algorithm is not more likely to generate optimal results, whereas Genetic algorithms do not guarantee to generate optimal global results, but also there is a great possibility of getting the optimal result for a problem as it uses genetic operators such as Crossover and Mutation.

- Traditional algorithms are deterministic in nature, whereas Genetic algorithms are probabilistic and stochastic in nature.

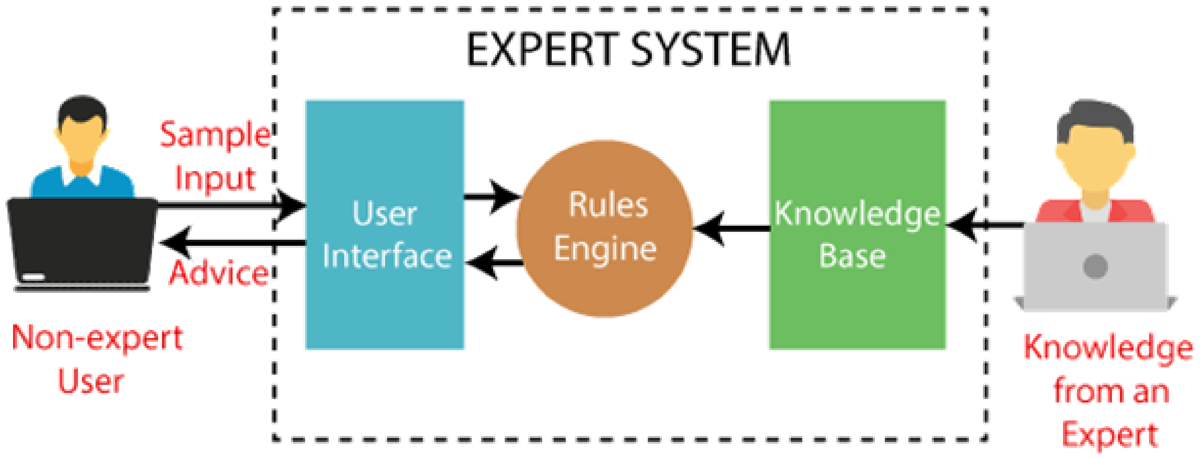
Expert System

An expert system is a computer program that is designed to solve complex problems and to provide decision-making ability like a human expert. It performs this by extracting knowledge from its knowledge base using the reasoning and inference rules according to the user queries.

The expert system is a part of AI, and the first ES was developed in the year 1970, which was the first successful approach of artificial intelligence. It solves the most complex issue as an expert by extracting the knowledge stored in its knowledge base. The system helps in decision making for complex problems using **both facts and heuristics like a human expert**. It is called so because it contains the expert knowledge of a specific domain and can solve any complex problem of that particular domain. These systems are designed for a specific domain, such as **medicine, science,** etc.

The performance of an expert system is based on the expert's knowledge stored in its knowledge base. The more knowledge stored in the KB, the more that system improves its performance. One of the common examples of an ES is a suggestion of spelling errors while typing in the Google search box.

Below is the block diagram that represents the working of an expert system:



Note: It is important to remember that an expert system is not used to replace the human experts; instead, it is used to assist the human in making a complex decision. These systems do not have human capabilities of thinking and work on the basis of the knowledge base of the particular domain.

Below are some popular examples of the Expert System:

- **DENDRAL:** It was an artificial intelligence project that was made as a chemical analysis expert system. It was used in organic chemistry to detect unknown organic molecules with the help of their mass spectra and knowledge base of chemistry.
- **MYCIN:** It was one of the earliest backward chaining expert systems that was designed to find the bacteria causing infections like bacteraemia and meningitis. It was also used for the recommendation of antibiotics and the diagnosis of blood clotting diseases.
- **PXDES:** It is an expert system that is used to determine the type and level of lung cancer. To determine the disease, it takes a picture from the upper body, which looks like the shadow. This shadow identifies the type and degree of harm.
- **CaDeT:** The CaDet expert system is a diagnostic support system that can detect cancer at early stages.

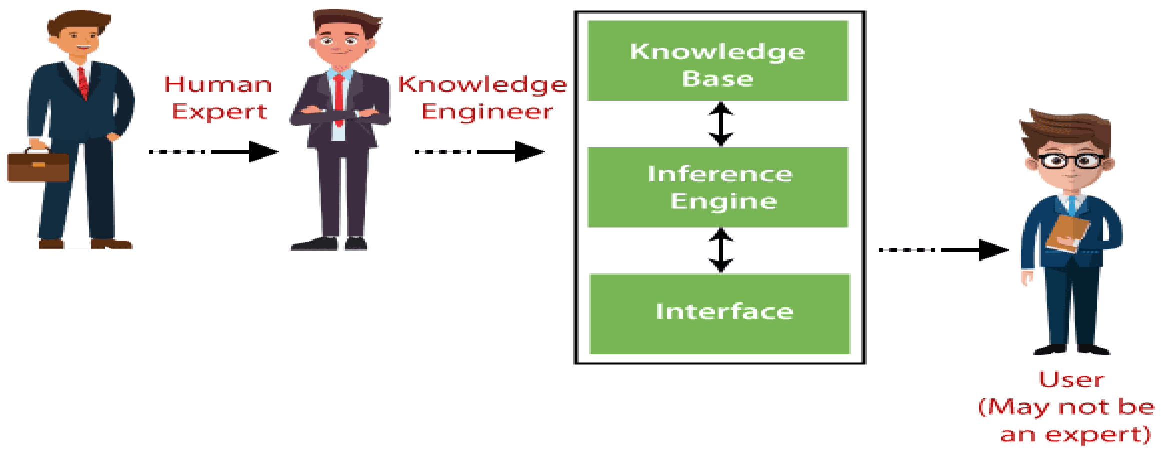
Characteristics of Expert System

- **High Performance:** The expert system provides high performance for solving any type of complex problem of a specific domain with high efficiency and accuracy.
- **Understandable:** It responds in a way that can be easily understandable by the user. It can take input in human language and provides the output in the same way.
- **Reliable:** It is much reliable for generating an efficient and accurate output.
- **Highly responsive:** ES provides the result for any complex query within a very short period of time.

Components of Expert System

An expert system mainly consists of three components:

- **User Interface**
- **Inference Engine**
- **Knowledge Base**



1. User Interface

With the help of a user interface, the expert system interacts with the user, takes queries as an input in a readable format, and passes it to the inference engine. After getting the response from the inference engine, it displays the output to the user. In other words, **it is an interface that helps a non-expert user to communicate with the expert system to find a solution**.

2. Inference Engine(Rules of Engine)

- The inference engine is known as the brain of the expert system as it is the main processing unit of the system. It applies inference rules to the knowledge base to derive a conclusion or deduce new information. It helps in deriving an error-free solution of queries asked by the user.
- With the help of an inference engine, the system extracts the knowledge from the knowledge base.
- There are two types of inference engine:
- **Deterministic Inference engine:** The conclusions drawn from this type of inference engine are assumed to be true. It is based on **facts** and **rules**.
- **Probabilistic Inference engine:** This type of inference engine contains uncertainty in conclusions, and based on the probability.

Inference engine uses the below modes to derive the solutions:

- **Forward Chaining:** It starts from the known facts and rules, and applies the inference rules to add their conclusion to the known facts.
- **Backward Chaining:** It is a backward reasoning method that starts from the goal and works backward to prove the known facts.

3. Knowledge Base

- The knowledgebase is a type of storage that stores knowledge acquired from the different experts of the particular domain. It is considered as big storage of knowledge. The more the knowledge base, the more precise will be the Expert System.
- It is similar to a database that contains information and rules of a particular domain or subject.
- One can also view the knowledge base as collections of objects and their attributes. Such as a Lion is an object and its attributes are it is a mammal, it is not a domestic animal, etc.

Components of Knowledge Base

- **Factual Knowledge:** The knowledge which is based on facts and accepted by knowledge engineers comes under factual knowledge.
- **Heuristic Knowledge:** This knowledge is based on practice, the ability to guess, evaluation, and experiences.

Knowledge Representation: It is used to formalize the knowledge stored in the knowledge base using the If-else rules.

Knowledge Acquisitions: It is the process of extracting, organizing, and structuring the domain knowledge, specifying the rules to acquire the knowledge from various experts, and store that knowledge into the knowledge base.

Capabilities of the Expert System

Below are some capabilities of an Expert System:

- **Advising:** It is capable of advising the human being for the query of any domain from the particular ES.
- **Provide decision-making capabilities:** It provides the capability of decision making in any domain, such as for making any financial decision, decisions in medical science, etc.
- **Demonstrate a device:** It is capable of demonstrating any new products such as its features, specifications, how to use that product, etc.
- **Problem-solving:** It has problem-solving capabilities.
- **Explaining a problem:** It is also capable of providing a detailed description of an input problem.
- **Interpreting the input:** It is capable of interpreting the input given by the user.

- **Predicting results:** It can be used for the prediction of a result.
- **Diagnosis:** An ES designed for the medical field is capable of diagnosing a disease without using multiple components as it already contains various inbuilt medical tools.

Advantages of Expert System

- These systems are highly reproducible.
- They can be used for risky places where the human presence is not safe.
- Error possibilities are less if the KB contains correct knowledge.
- The performance of these systems remains steady as it is not affected by emotions, tension, or fatigue.
- They provide a very high speed to respond to a particular query.

Limitations of Expert System

- The response of the expert system may get wrong if the knowledge base contains the wrong information.
- Like a human being, it cannot produce a creative output for different scenarios.
- Its maintenance and development costs are very high.
- Knowledge acquisition for designing is much difficult.
- For each domain, we require a specific ES, which is one of the big limitations.
- It cannot learn from itself and hence requires manual updates.

Applications of Expert System

- **In designing and manufacturing domain**
  It can be broadly used for designing and manufacturing physical devices such as camera lenses and automobiles.
- **In the knowledge domain**
  These systems are primarily used for publishing the relevant knowledge to the users. The two popular ES used for this domain is an advisor and a tax advisor.
- **In the finance domain**
  In the finance industries, it is used to detect any type of possible fraud, suspicious activity, and advise bankers that if they should provide loans for business or not.
- **In the diagnosis and troubleshooting of devices**
  In medical diagnosis, the ES system is used, and it was the first area where these systems were used.

- **Planning and Scheduling**
  The expert systems can also be used for planning and scheduling some particular tasks for achieving the goal of that task

Natural Language Processing

Natural language processing (NLP) is a subfield of Artificial Intelligence (AI). This is a widely used technology for personal assistants that are used in various business fields/areas. This technology works on the speech provided by the user breaks it down for proper understanding and processes it accordingly. This is a very recent and effective approach due to which it has a really high demand in today's market.

Natural Language Processing is an upcoming field where already many transitions such as compatibility with smart devices, and interactive talks with a human have been made possible. Knowledge representation, logical reasoning, and constraint satisfaction were the emphasis of AI applications in NLP.

Here first it was applied to semantics and later to grammar.

The goal of NLP is for computers to be able to interpret and generate human language. This not only improves the efficiency of work done by humans but also helps in interacting with the machine. NLP bridges the gap of interaction between humans and electronic devices.

Working of Natural Language Processing (NLP)

Working in natural language processing (NLP) typically involves using computational techniques to analyse and understand human language. This can include tasks such as language understanding, language generation, and language interaction.

I/P --> AUTO SPEECH RECOGNITION --> NLU --> NLG --> O/P

The field is divided into three different parts:

1. Speech Recognition — The translation of spoken language into text.
2. Natural Language Understanding (NLU) — The computer's ability to understand what we say.
3. Natural Language Generation (NLG) — The generation of natural language by a computer.

NLU and NLG are the key aspects depicting the working of NLP devices. These 2 aspects are very different from each other and are achieved using different methods.

Individuals working in NLP may have a background in computer science, linguistics, or a related field. They may also have experience with programming languages such as Python, and C++ and be familiar with various NLP libraries and frameworks such as NLTK, spaCy, and OpenNLP.

Speech Recognition:

- First, the computer must take natural language and convert it into machine-readable language. This is what speech recognition or speech-to-text does. This is the first step of NLU.
- Hidden Markov Models (HMMs) are used in the majority of voice recognition systems nowadays. These are statistical models that use mathematical calculations to determine what you said in order to convert your speech to text.
- HMMs do this by listening to your talk, breaking it down into small units (typically 10-20 milliseconds), and comparing it to pre-recorded speech to figure out which phoneme you uttered in each unit (a phoneme is the smallest unit of speech). The program then examines the sequence of phonemes and uses statistical analysis to determine the most likely words and sentences you were speaking.

Natural Language Understanding (NLU):

The next and hardest step of NLP is the understanding part.

- First, the computer must comprehend the meaning of each word. It tries to figure out whether the word is a noun or a verb, whether it's in the past or present tense, and so on. This is called Part-of-Speech tagging (POS).
- A lexicon (a vocabulary) and a set of grammatical rules are also built into NLP systems. The most difficult part of NLP is understanding.
- The machine should be able to grasp what you said by the conclusion of the process. There are several challenges in accomplishing this when considering problems such as words having several meanings (polysemy) or different words having similar meanings (synonymy), but developers encode rules into their NLU systems and train them to learn to apply the rules correctly.

Natural Language Generation (NLG):

NLG is much simpler to accomplish. NLG converts a computer's machine-readable language into text and can also convert that text into audible speech using text-to-speech technology.

- First, the NLP system identifies what data should be converted to text. If you asked the computer a question about the weather, it most likely did an online search to find your answer, and from there it decides that the temperature, wind, and humidity are the factors that should be read aloud to you.
- Then, it organizes the structure of how it's going to say it. This is similar to NLU except backward. NLG system can construct full sentences using a lexicon and a set of grammar rules.
- Finally, text-to-speech takes over. The text-to-speech engine uses a prosody model to evaluate the text and identify breaks, duration, and pitch. The engine then combines all the recorded phonemes into one cohesive string of speech using a speech database.

Applications of NLP

1.Question Answering

2.Spam Detection IN email & email & another online platform

3.Sentiment Analysis in online

4.Machine Translation

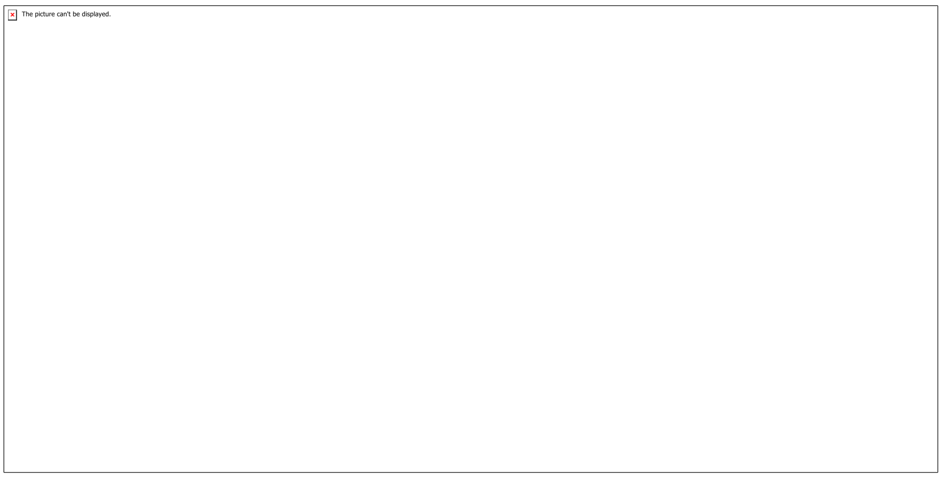
5.Spelling Correction

6.Speech Recognition in voice assistant

7.Information Extraction

Text Classification

Text classification in NLP involves categorizing and assigning predefined labels or categories to text documents, sentences, or phrases based on their content. Text classification aims to automatically determine the class or category to which a piece of text belongs. It's a fundamental task in NLP with numerous practical applications, including sentiment analysis, spam detection, topic labelling, language identification, and more. Text classification algorithms analyse the features and patterns within the text to make accurate predictions about its category, enabling machines to organize, filter, and understand large volumes of textual data.



Why text classification is important

With text classification, businesses can make the most out of unstructured data. Text classification tools allow organizations to efficiently and cost-effectively arrange all types of texts, e-mails, legal papers, ads, databases, and other documents. This enables them to save time and make informed decisions based on relevant data.

For example, you could collect app crash reports and categorize them based on the problem. Categories for this could be:

- Loading time
- App not responsive
- Screen freezes

Types

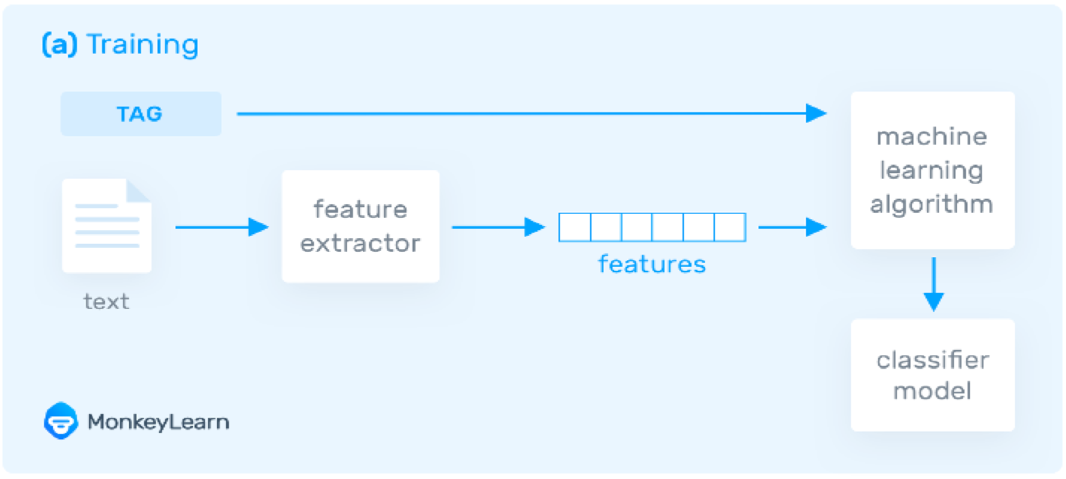
Text clarification is the process of categorizing the text into a group of words. By using NLP, text classification can automatically analyse text and then assign a set of predefined tags or categories based on its context. NLP is used for sentiment analysis, topic detection, and language detection. There is mainly three text classification approach:

- Rule-based System
- Machine System
- Hybrid System

In the rule-based approach, texts are separated into an organized group using a set of handicraft linguistic rules. Those handicraft linguistic rules contain users to define a list of words that are characterized by groups. For example, words like Donald Trump and Boris Johnson would be categorized into politics. People like LeBron James and Ronaldo would be categorized into sports.

Machine-based classifier learns to make a classification based on past observation from the data sets. User data is prelabelled as tarin and test data. It collects the classification strategy from the previous inputs and learns continuously. Machine-based classifier usage a bag of a word for feature extension.

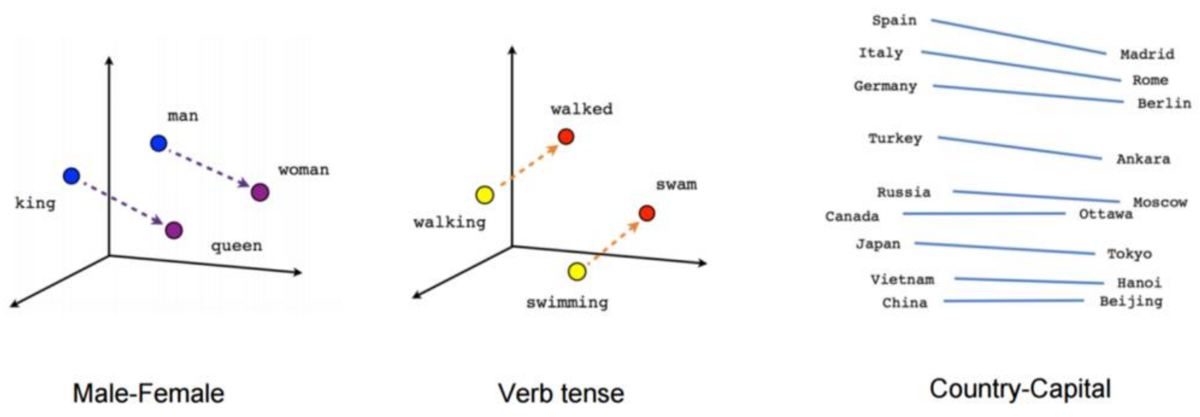
In a bag of words, a vector represents the frequency of words in a predefined dictionary of a word list. We can perform NLP using the following machine learning algorithms: Naïve Bayer, SVM, and Deep Learning.



The hybrid approach to text classification is the Hybrid Approach. Hybrid approach usage combines a rule-based and machine Based approach. Hybrid based approach usage of the rule-based system to create a tag and use machine learning to train the system and create a rule. Then the machine-based rule list is compared with the rule-based rule list. If something does not match on the tags, humans improve the list manually. It is the best method to implement text classification

Vector Semantic

Vector Semantic is another way of word and sequence analysis. It defines semantic and interprets words meaning to explain features such as similar words and opposite words. The main idea behind vector semantic is two words are alike if they have used in a similar context. Vector semantic divide the words in a multi-dimensional vector space. Vector semantic is useful in sentiment analysis.

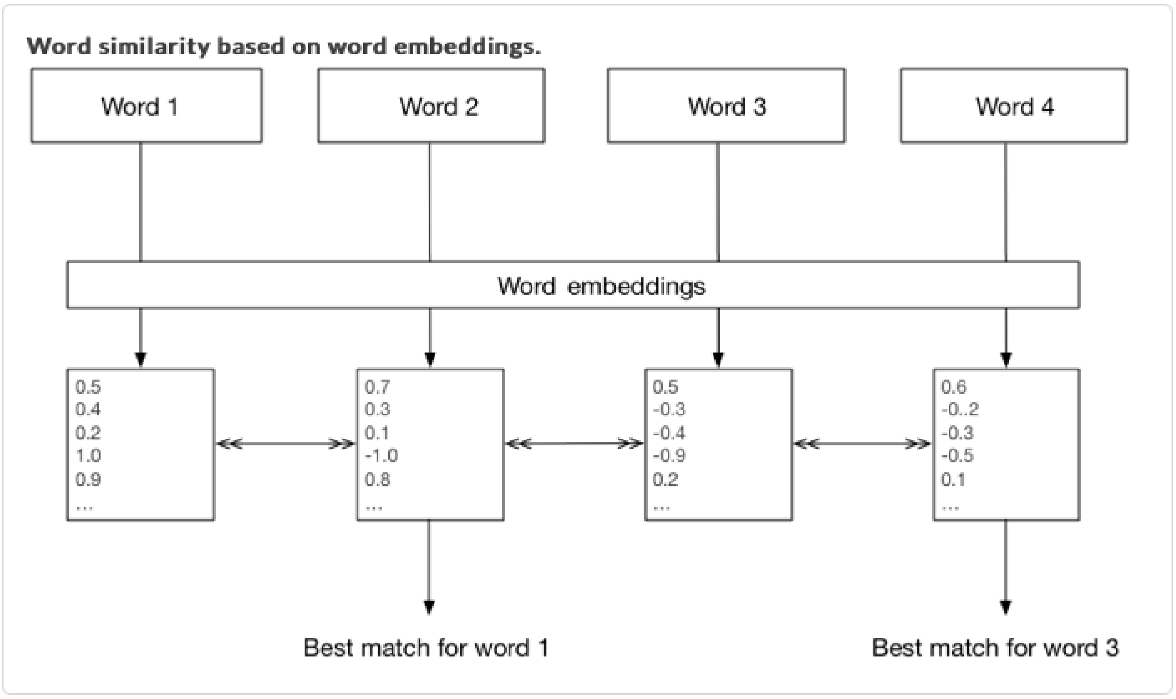


Word Embedding

Word embedding is another method of word and sequence analysis. Embedding translates spares vectors into a low-dimensional space that preserves semantic relationships. Word embedding is a type of word representation that allows words with similar meaning to have a similar representation. There are two types of word embedding-

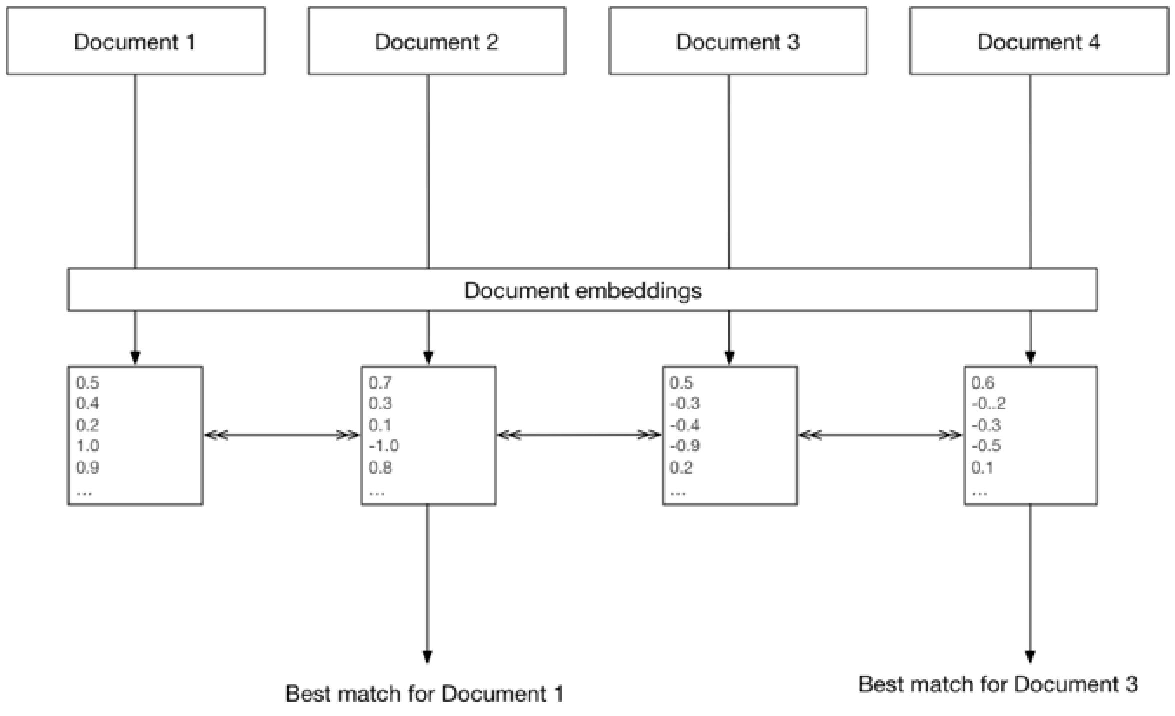
- Word2vec
- Doc2Vec.

Word2Vec is a statistical method for effectively learning a standalone word embedding from a text corpus.



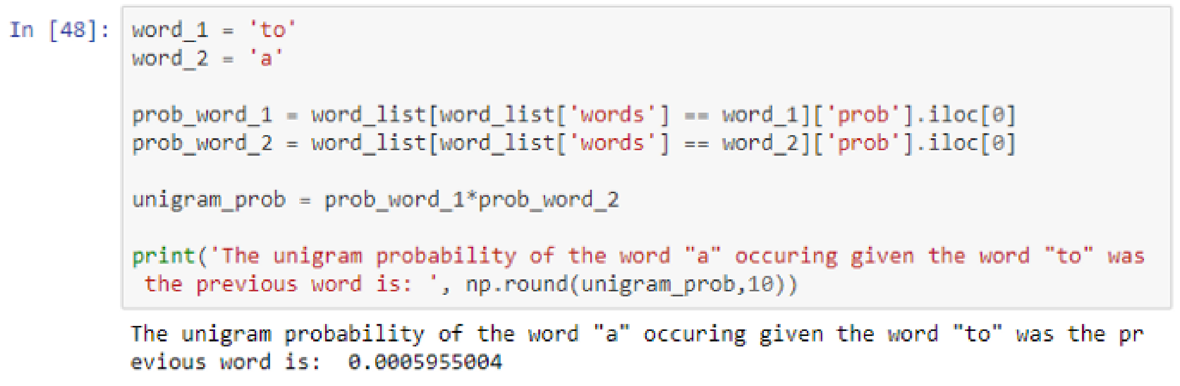
Doc2Vec is similar to Doc2Vec, but it analyses a group of text like pages.

Document similarity based on document embeddings.



Probabilistic Language Model

Another approach to word and sequence analysis is the probabilistic language model. The goal of the probabilistic language model is to calculate the probability of a sentence of a sequence of words. For example, the probability of the word "a" occurring in a given word "to" is 0.00013131 percent.

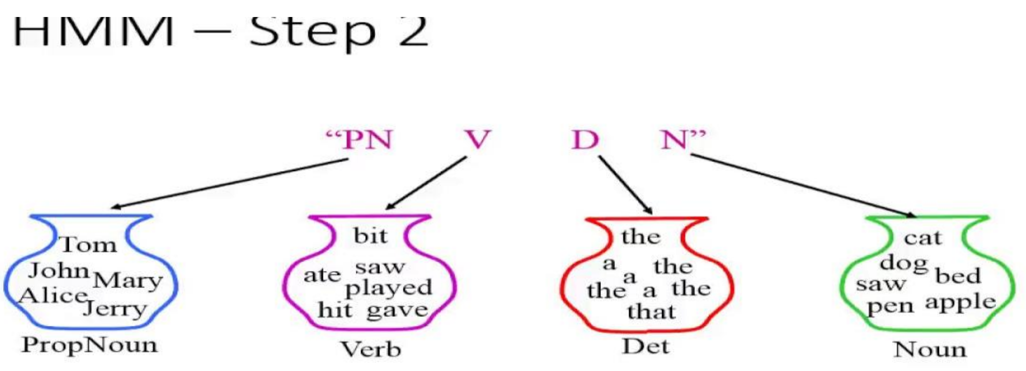


```
In [48]: word_1 = 'to'
         word_2 = 'a'

         prob_word_1 = word_list[word_list['words'] == word_1]['prob'].iloc[0]
         prob_word_2 = word_list[word_list['words'] == word_2]['prob'].iloc[0]

         unigram_prob = prob_word_1*prob_word_2

         print('The unigram probability of the word "a" occuring given the word "to" was
          the previous word is: ', np.round(unigram_prob,10))

         The unigram probability of the word "a" occuring given the word "to" was the pr
         evious word is:  0.0005955004
```

Sequence Labelling

Sequence labelling is a typical NLP task that assigns a class or label to each token in a given input sequence. If someone says "play the movie by tom hanks". In sequence, labelling will be [play, movie, tom hanks]. Play determines an action. Movies are an instance of action. Tom Hanks goes for a search entity. It divides the input into multiple tokens and uses LSTM to analyse it. There are two forms of sequence labelling. They are token labelling and span labelling.

Parsing is a phase of NLP where the parser determines the syntactic structure of a text by analysing its constituent words based on an underlying grammar. For example, “tom ate an apple” will be divided into proper noun ◊ tom, verb ◊ ate, determiner ◊ , noun ◊ apple. The best example is Amazon Alexa.

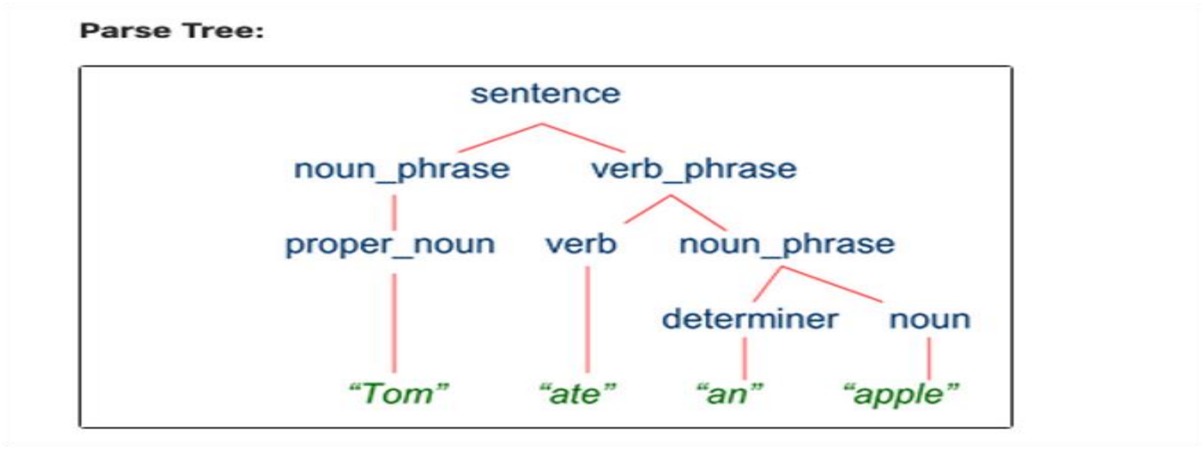

HMM – Step 2
“PN V D N”
Tom John Mary Alice Jerry
PropNoun
bit ate saw played hit gave
Verb
the a the a the a the that
Det
cat dog saw bed pen apple
Noun

We discuss how text is classified and how to divide the word and sequence so that the algorithm can understand and categorize it. In this project, we are going to discover a sentiment analysis of fifty thousand IMDB movie reviewer. Our goal is to identify whether the review posted on the IMDB site by its user is positive or negative.

This project covers text mining techniques like Text Embedding, Bags of Words, word context, and other things. We will also cover the introduction of a bidirectional LSTM sentiment classifier. We will also look at how to import a labelled dataset from TensorFlow automatically. This project also covers steps like data cleaning, text processing, data balance through sampling, and train and test a deep learning model to classify text.

Parsing

Parser determines the syntactic structure of a text by analysing its constituent words based on an underlying grammar. It divides group words into component parts and separates words.



Semantic

Text is at the heart of how we communicate. What is really difficult is understanding what is being said in written or spoken conversation? Understanding lengthy articles and books are even more difficult. Semantic is a process that seeks to understand linguistic meaning by constructing a model of the principle that the speaker uses to convey meaning. It's has been used in customer feedback analysis, article analysis, fake news detection, Semantic analysis, etc.

sSSSSSSSS

Applications of Text Classification

1.content filtering

2.Spam Detection IN email & email & another online platform

3.Sentiment Analysis in online

4.Machine Translation

5.Spelling Correction

6.Speech Recognition in voice assistant

7.Information Extraction

8.language detection in google translate

9.age/gender identification of anonymous user

10.Ads media analysis