SCHOOL OF COMPUTER SCIENCE,ENGINEERING AND APPLICATIONS ,

BHARATHIDASAN UNIVERSITY,

KHAJAMALAI CAMPUS,

TRICHY-620 023

ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

(SUB. CODE: MCA24302)

STUDY MATERIAL

FACULTY NAME : Mrs. R. RAMYA

DESIGNATION : GUEST LECTURER

SEMESTER : III

CLASS : MCA

ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

Unit – 2:

- Problem solving Methods
- Search Strategies
- Uninformed
- Informed
- Heuristics
- Local Search Algorithms and Optimization Problems
- Searching with Partial Observations
- Constraint Satisfaction Problems
- Constraint Propagation
- Backtracking Search
- Game Playing
- Optimal Decisions in Games
- Alpha - Beta Pruning
- Stochastic Games

PROBLEM SOLVING APPROACH TO TYPICAL AI PROBLEMS

- Artificial Intelligence, Search techniques are universal problem-solving methods.
- Rational agents or Problem-solving agents in AI mostly used these search strategies or algorithms to solve a specific problem and provide the best result.
- Problem- solving agents are the goal-based agents and use atomic representation.

Some of the most popularly used problem solving with the help of artificial intelligence are,

1. Chess.
2. Travelling Salesman Problem.
3. Tower of Hanoi Problem.
4. Water-Jug Problem.
5. N-Queen Problem.

Problem Searching

- In general, searching refers to as finding information one needs.
- Searching is the most commonly used technique of problem solving in artificial intelligence. The searching algorithm helps us to search for solution of particular problem.

Problem:

Problems are the issues which comes across any system. A solution is needed to solve that particular problem.

Steps : Solve Problem Using Artificial Intelligence

The process of solving a problem consists of five steps. These are:

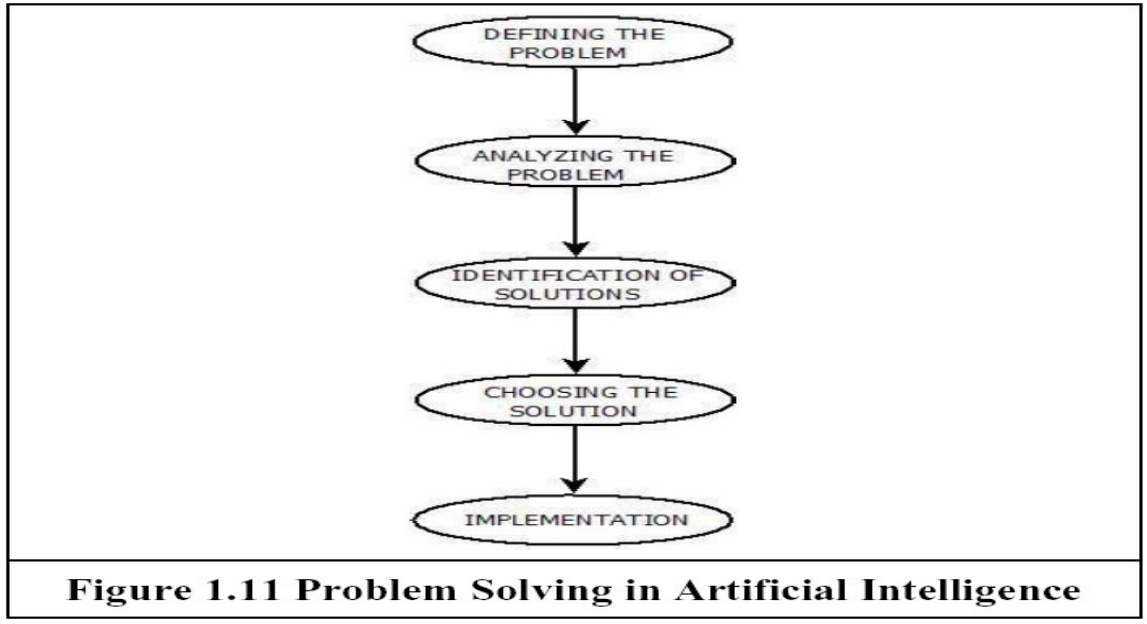


Figure 1.11 Problem Solving in Artificial Intelligence

1.Defining The Problem: The definition of the problem must be included precisely. It should contain the possible initial as well as final situations which should result in acceptable solution.

2.Analyzing The Problem: Analyzing the problem and its requirement must be done as few features can have immense impact on the resulting solution.

3. **Identification Of Solutions:** This phase generates reasonable amount of solutions to the given problem in a particular range.

4.Choosing a Solution: From all the identified solutions, the best solution is chosen basis on the results produced by respective solutions.

5.Implementation: After choosing the best solution, its implementation is done.

Measuring problem-solving performance

- **Completeness:** Is the algorithm guaranteed to find a solution when there is one?
- **Optimality:** Does the strategy find the optimal solution?
- **Time complexity:** How long does it take to find a solution?
- **Space complexity:** How much memory is needed to perform the search?

Search Algorithm Terminologies:

Search: Searching is a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:

1. **Search Space:** Search space represents a set of possible solutions, which a system may have.
2. **Start State:** It is a state from where agent begins the search.
3. **Goal test:** It is a function which observe the current state and returns whether the goal state is achieved or not.

Search tree: A tree representation of search problem is called Search tree. The root of the search tree is the root node which is corresponding to the initial state.

Actions: It gives the description of all the available actions to the agent.

Transition model: A description of what each action do, can be represented as a transition model.

Path Cost: It is a function which assigns a numeric cost to each path.

Solution: It is an action sequence which leads from the start node to the goal node.

Optimal Solution: If a solution has the lowest cost among all solutions.

Problem-solving agents:

- A Problem solving agent is a goal-based agent. It decide what to do by finding sequence of actions that lead to desirable states. The agent can adopt a goal and aim at satisfying it.
- Goal formulation, based on the current situation and the agent's performance measure, is the first step in problem solving.
- The agent's task is to find out which sequence of actions will get to a goal state.
- Problem formulation is the process of deciding what actions and states to consider given a goal.

Example: Route finding problem

Referring to figure

On holiday in Romania : currently in Arad. Flight leaves tomorrow from Bucharest
Formulate goal: be in Bucharest

Formulate problem: **states**: various cities

actions: drive between cities

Find solution:

sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

Problem formulation

A **problem** is defined by four items:

initial state e.g., "at Arad"

successor function S(x) = set of action-state pairs e.g., S(Arad) = {[Arad ->Zerind;Zerind],….} **goal test**, can be

explicit, e.g., x = at Bucharest" implicit, e.g., NoDirt(x)

path cost (additive)

e.g., sum of distances, number of actions executed, etc. c(x; a; y) is the step cost, assumed to be >= 0

A **solution** is a sequence of actions leading from the initial state to a goal state.

Goal formulation and problem formulation

EXAMPLE PROBLEMS

- The problem solving approach has been applied to a vast array of task environments. Some best known problems are summarized below. They are distinguished as toy or real-world problems
- A **toy problem is intended to illustrate various problem solving methods.** It can be easily used by different researchers to compare the performance of algorithms.

For example, Vacuum World, 8-puzzle, 8-Queens problem**.**

- A **real world problem is one whose solutions people actually care about.**

For example, **Route-finding Problem, Airline Travel Problem, Touring Problems, The Travelling Salesperson Problem(TSP).**

TOY PROBLEMS(Vacuum World Example)

States: The agent is in one of two locations, each of which might or might not contain dirt. Thus there are $2 \times 2^2 = 8$ possible world states.

Initial state: Any state can be designated as initial state.

Successor function: This generates the legal states that results from trying the three actions (left, right, suck). The complete state space is shown in figure

Goal Test: This tests whether all the squares are clean.

Path test: Each step costs one, so that the path cost is the number of steps in the path.

Vacuum World State Space

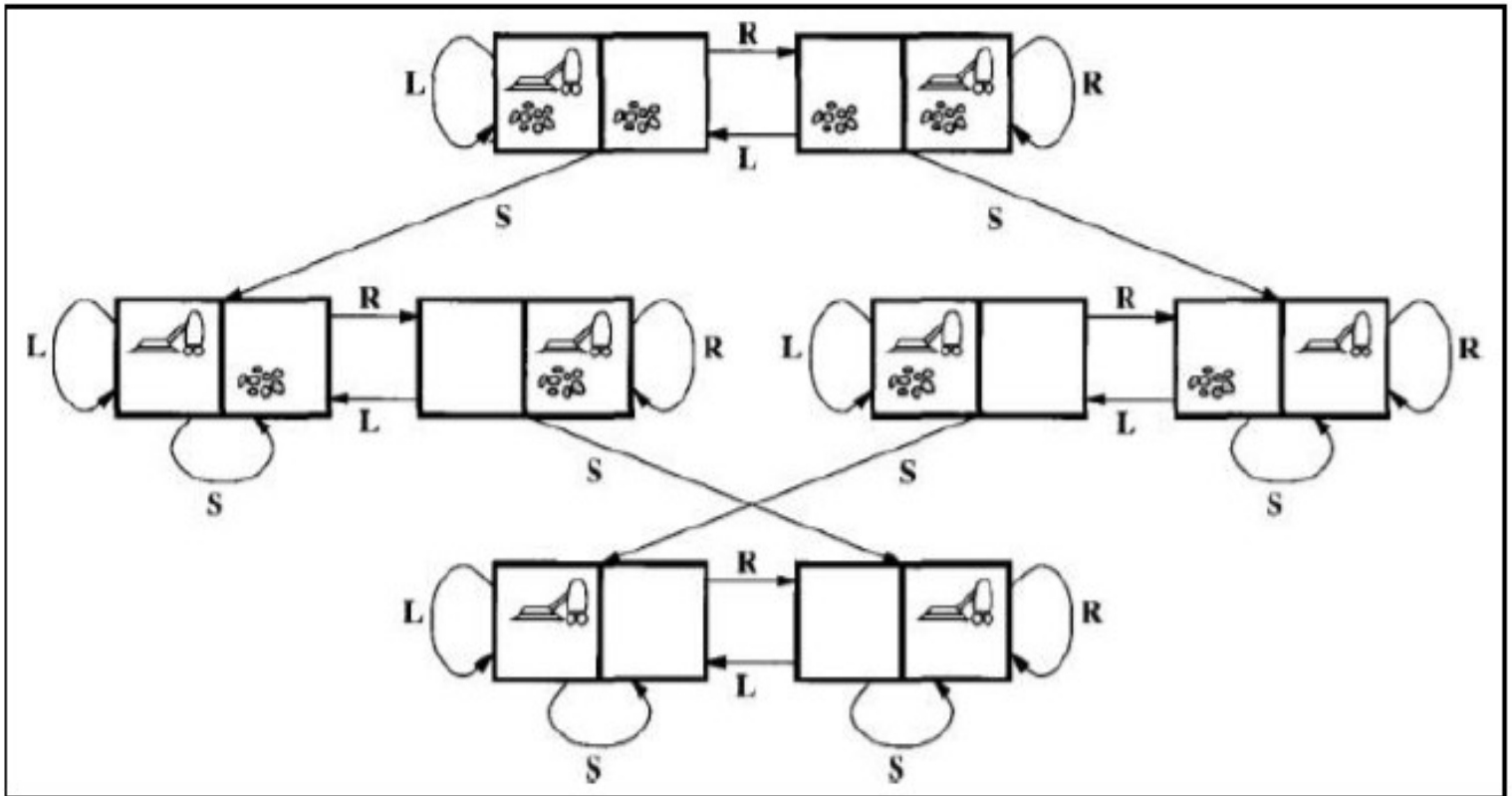


Figure 2.1 The state space for the vacuum world.

Arcs denote actions: L = Left, R = Right

AIRLINE TRAVEL PROBLEM

The airline travel problem is specifies as follows:

States: Each is represented by a location (e.g., an airport) and the current time.

Initial state: This is specified by the problem.

Successor function: This returns the states resulting from taking any scheduled flight(further specified by seat class and location),leaving later than the current time plus the within-airport transit time, from the current airport to another.

Goal Test: Are we at the destination by some prespecified time?

Path cost: This depends upon the monetary cost, waiting time, flight time, customs and immigration procedures, seat quality, time of date, type of air plane, frequent-flyer mileage awards, and so on.

Internet Searching

In recent years there has been increased demand for software robots that perform Internet searching, looking for answers to questions, for related information, or for shopping deals. The searching techniques consider internet as a graph of nodes(pages) connected by links.

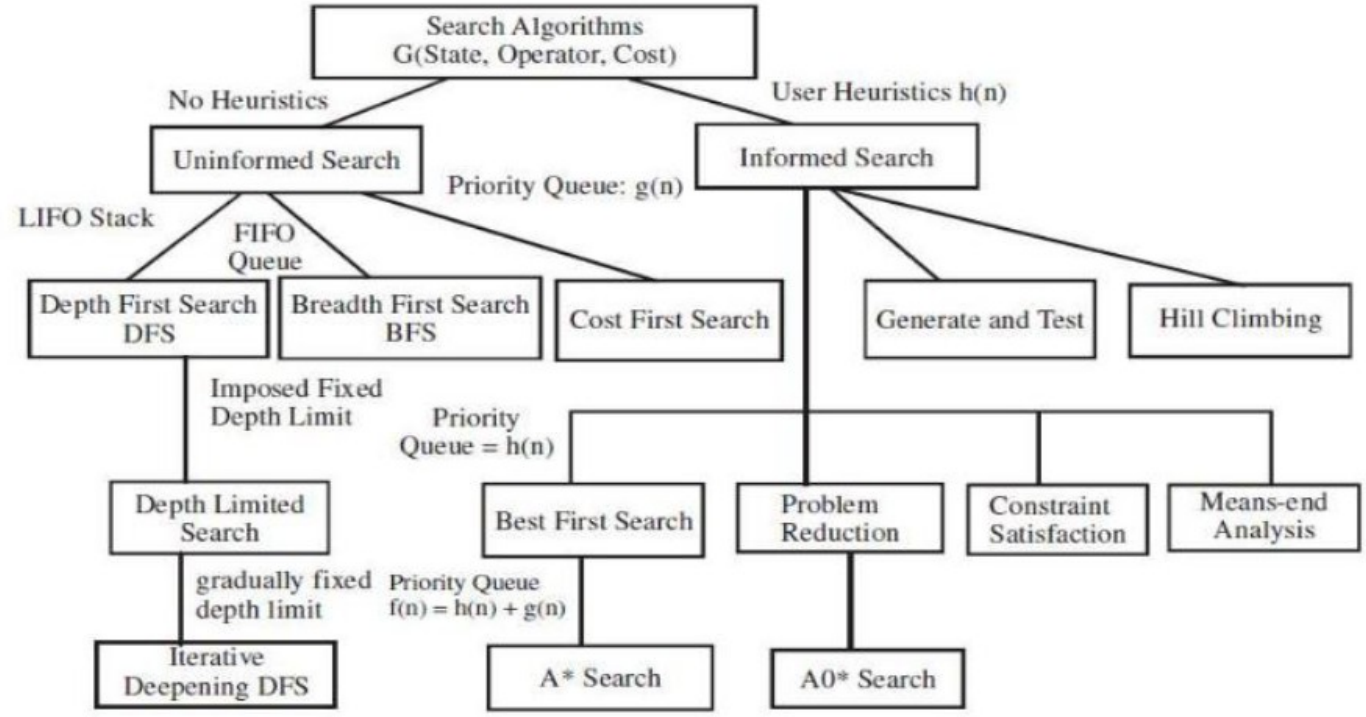


Figure 2.4 Different Search Algorithms

UNINFORMED SEARCH STRATGES

- Uninformed Search Strategies have no additional information about states beyond that provided in the problem definition.
- Strategies that know whether one non goal state is “more promising” than another are Informed search or heuristic search strategies.
- There are five uninformed search strategies as given below.
  - Breadth-first search
  - Uniform-cost search
  - Depth-first search
  - Depth-limited search
  - Iterative deepening search

Breadth-first search

- Breadth-first search is a simple strategy in which the root node is expanded first, then all successors of the root node are expanded next, then their successors, and so on.
- Breath-first-search is implemented by calling TREE-SEARCH with an empty fringe that is a first-in-first-out (FIFO) queue, assuring that the nodes that are visited first will

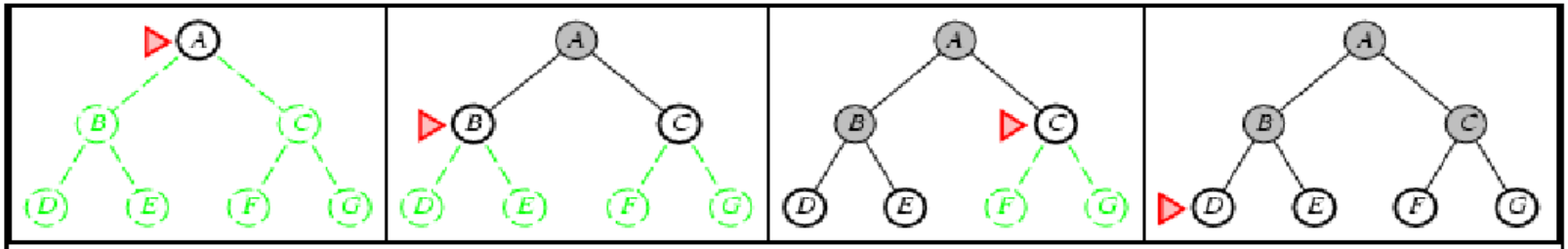


Figure 2.5 Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by a marker.

Properties of breadth-first-search

Complete?? Yes (if b is finite)

Time?? $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$, i.e., exp. in d

Space?? $O(b^{d+1})$ (keeps every node in memory)

Optimal?? No, unless step costs are constant

Space is the big problem; can easily generate nodes at 100MB/sec
so 24hrs = 8640GB.

Figure 2.6 Breadth-first-search properties

Uniform-cost search

- Instead of expanding the shallowest node, **uniform-cost search expands the node n** with the lowest path cost. Uniform-cost search does not care about the number of steps a path has, but only about their total cost.

Expand least-cost unexpanded node

Implementation:

$fringe$ = queue ordered by path cost, lowest first

Equivalent to breadth-first if step costs all equal

Complete?? Yes, if step cost $\geq \epsilon$

Time?? # of nodes with $g \leq$ cost of optimal solution, $O(b^{\lceil C^*/\epsilon \rceil})$
where C^* is the cost of the optimal solution

Space?? # of nodes with $g \leq$ cost of optimal solution, $O(b^{\lceil C^*/\epsilon \rceil})$

Optimal?? Yes—nodes expanded in increasing order of $g(n)$

Depth-first search

- Depth-first-search always expands the deepest node in the current fringe of the search tree.
- The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors.
- This strategy can be implemented by TREE-SEARCH with a last-in-first-out (LIFO) queue, also known as a stack.

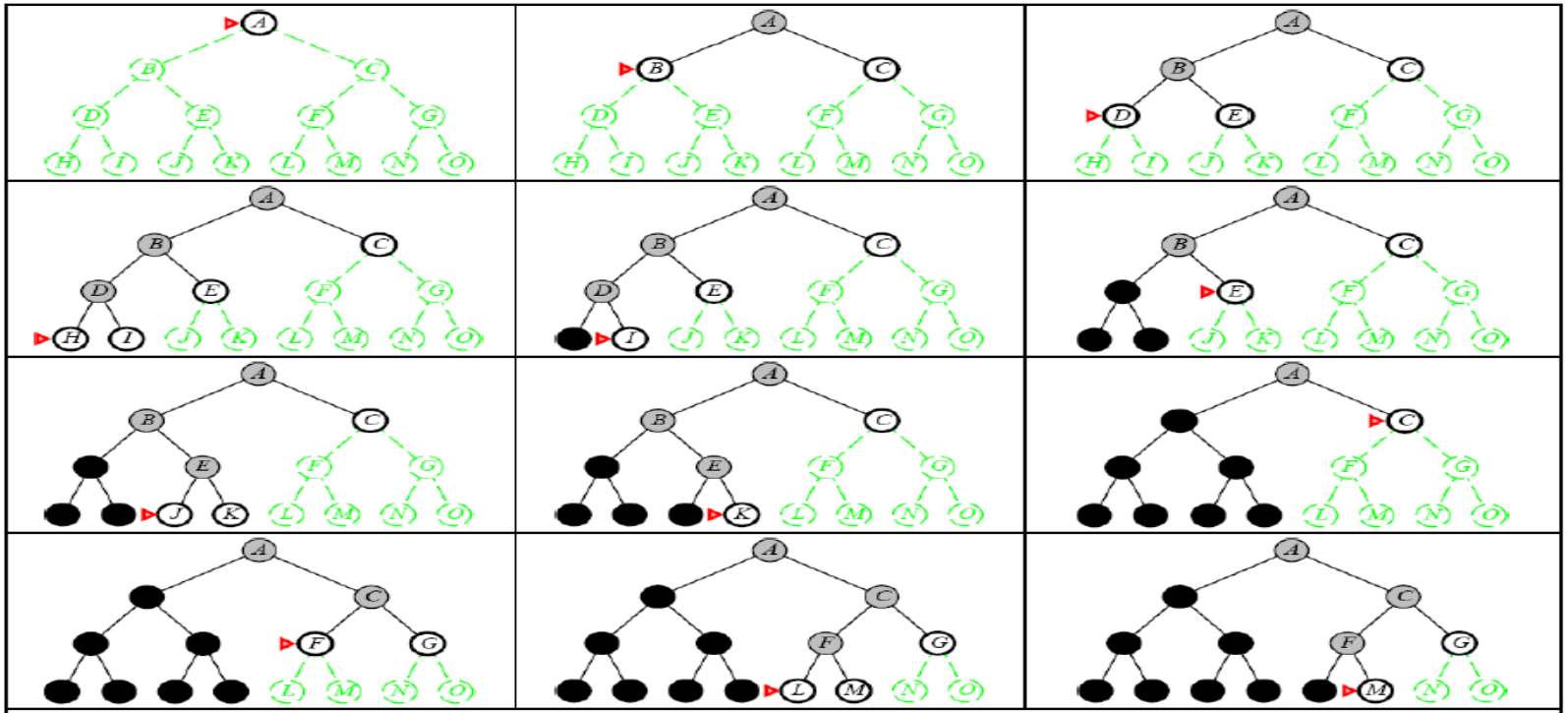


Figure 2.7 Depth-first-search on a binary tree. Nodes that have been expanded and have node scendants in the fringe can be removed from the memory; these are shown in black. Nodes at depth 3 are assumed to have no successors and M is the only goal node.

BACKTRACKING SEARCH

- A variant of depth-first search called backtracking search uses less memory and only one successor is generated at a time rather than all successors.; Only O(m) memory is needed rather than O(bm)

Depth-limited search

- The problem of unbounded trees can be alleviated by supplying depth-first-search with a pre- determined depth limit l. That is, nodes at depth l are treated as if they have no successors. This approach is called **depth-limited-search. The depth limit solves the infinite path problem.**

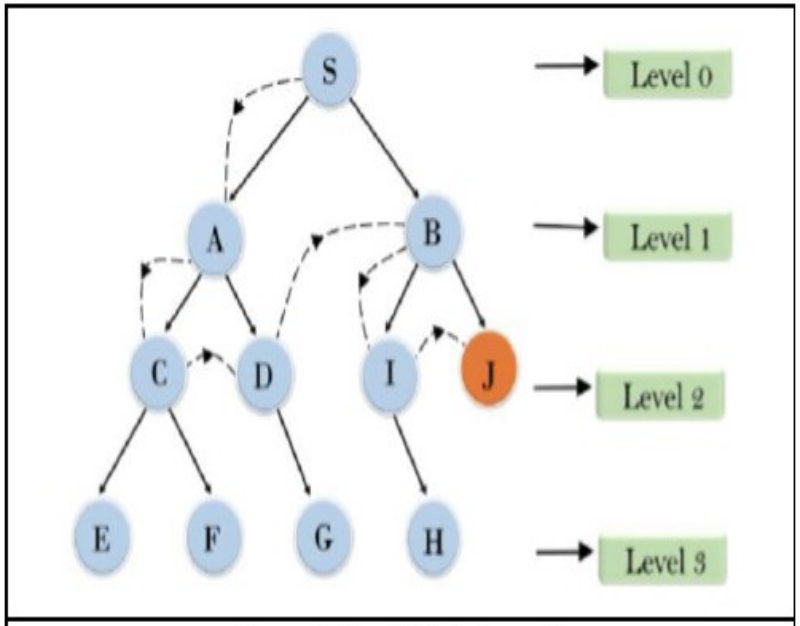


Figure 2.8 Depth-limited-search

```
function Depth-Limited-Search( problem, limit) returns a solution/fail/cutoff return
Recursive-DLS(Make-Node(Initial-State[problem]), problem, limit) function Recursive-
DLS(node, problem, limit) returns solution/fail/cutoff cutoff-occurred?      false
if Goal-Test(problem,State[node]) then return Solution(node)
else if Depth[node] = limit then return cutoff
else for each successor in Expand(node, problem) do result
Recursive-DLS(successor, problem, limit)  if result = cutoff then cutoff_occurred?true
else if result not = failure then return result
ifcutoff_occurred? then return cutoff else return failure
```

Figure 2.9 Recursive implementation of Depth-limited-search

Iterative deepening search

- Iterative deepening combines the benefits of depth-first and breadth-first-search Like depth-first-search, its memory requirements are modest; O(bd) to be precise.

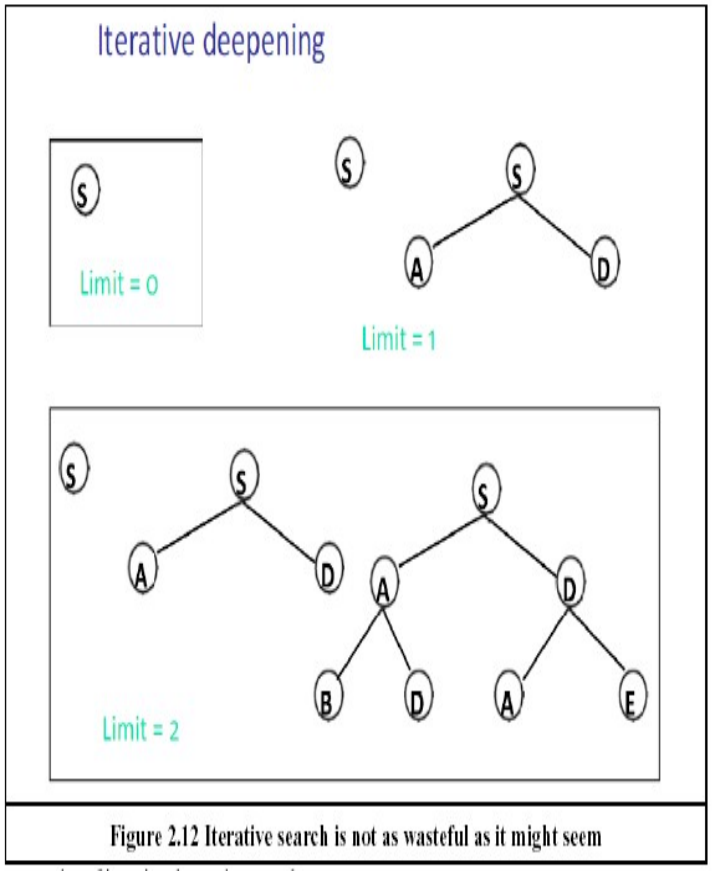


Figure 2.12 Iterative search is not as wasteful as it might seem

Complete?? Yes

Time?? $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

Space?? $O(bd)$

Optimal?? No, unless step costs are constant
Can be modified to explore uniform-cost tree

Numerical comparison for $b = 10$ and $d = 5$, solution at far right leaf:

$$N(\text{IDS}) = 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450$$
$$N(\text{BFS}) = 10 + 100 + 1,000 + 10,000 + 100,000 + 999,990 = 1,111,100$$

IDS does better because other nodes at depth d are not expanded

BFS can be modified to apply goal test when a node is generated

Figure 2.13 Properties of iterative deepening search

Bidirectional Search

- Bidirectional Search as the name suggests is a combination of forwarding and backward search.

    1. Forward Search: Looking in-front of the end from start.
    2. Backward Search: Looking from end to the start back-wards.

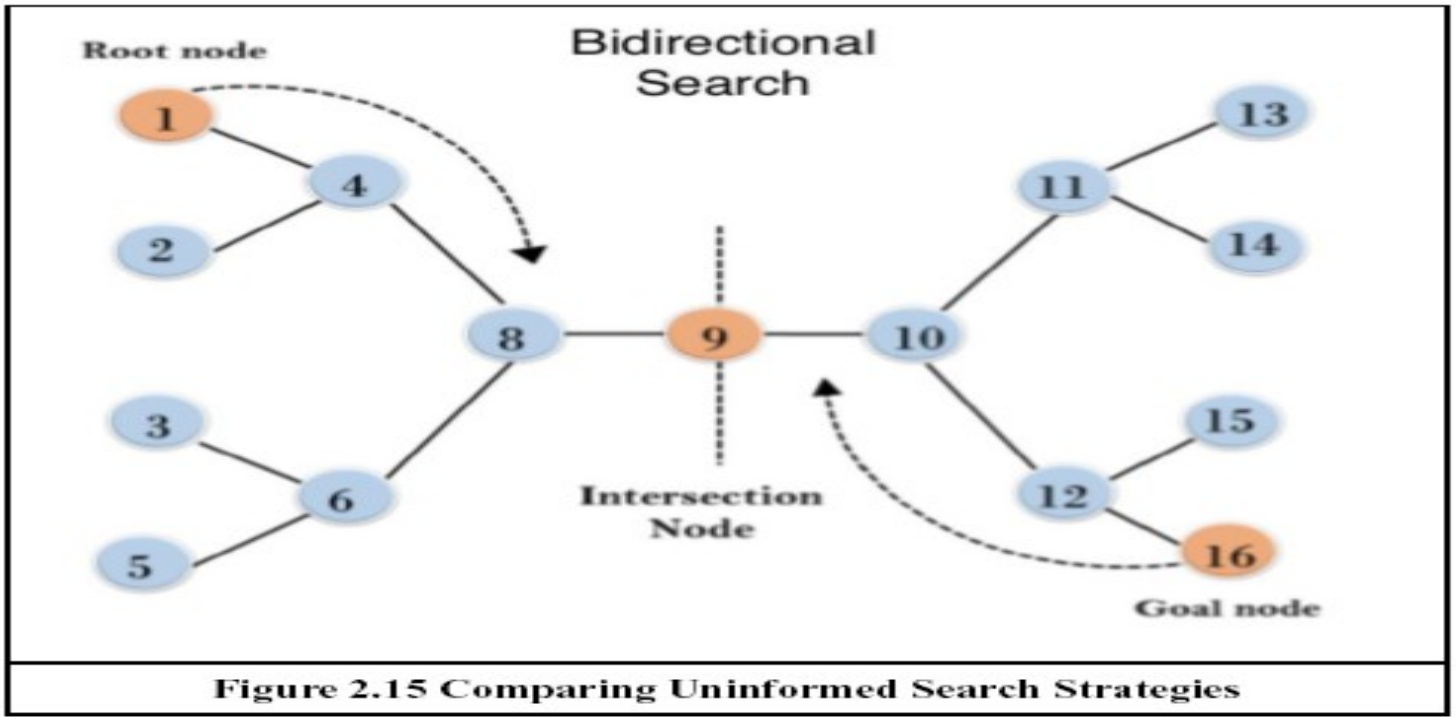


Figure 2.15 Comparing Uninformed Search Strategies

INFORMED SEARCH

- Informed search strategy is one that uses problem-specific knowledge beyond the definition of the problem itself.
- It can find solutions more efficiently than uninformed strategy.

Best-first search

- Best-first search is an instance of general TREE-SEARCH or GRAPH-SEARCH algorithm in which a node is selected for expansion based on an evaluation function f(n).
- The node with lowest evaluation is selected for expansion, because the evaluation measures the distance to the goal.
- This can be implemented using a priority-queue, a data structure that will maintain the fringe in ascending order of f-values.

HEURISTIC FUNCTIONS

- A **heuristic function is a function that ranks alternatives in** various search algorithms at each branching step basing on an available information in order to make a decision which branch is to be followed during a search.
- The key component of Best-first search algorithm is a **heuristic function, denoted by** h(n): h(n) = estimated cost of the **cheapest path from node n to a goal node.**

Greedy Best-first search

- **Greedy best-first search tries to expand the node that is closest to the goal, on the** grounds that this is likely to a solution quickly.
- It evaluates the nodes by using the heuristic function f(n) = h(n).

Example ,

- **Route-finding problems in Romania, the goal is to reach** Bucharest starting from the city Arad. We need to know the straight-line distances to Bucharest from various cities as shown in Figure. For example, the initial state is In(Arad),and the straight line distance heuristic hSLD (In(Arad)) is found to be 366.
- Using the **straight-line distance heuristic hSLD, the goal state can be reached faster.**

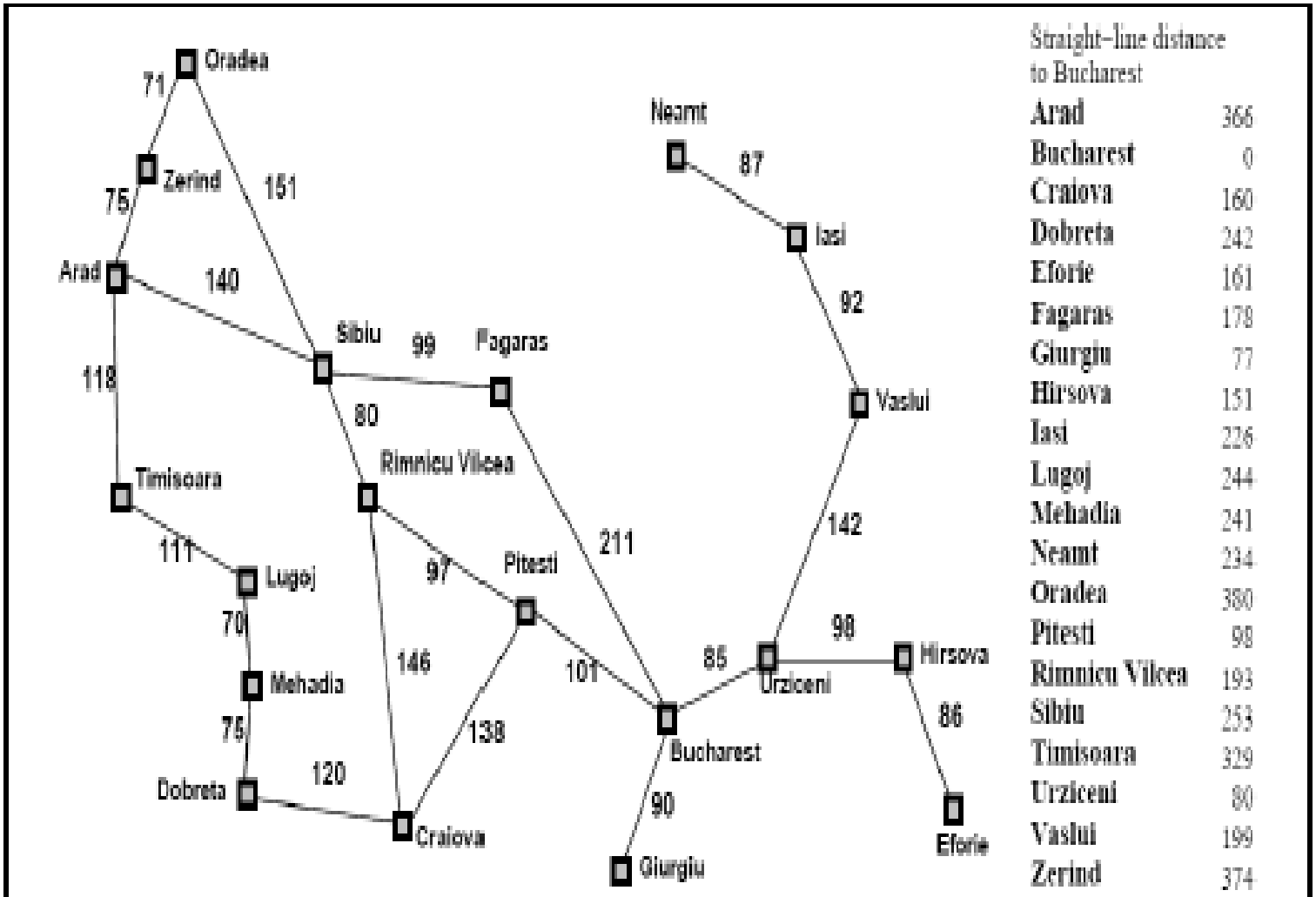


Figure 2.19 Values of h_{SLD} - straight line distances to Bucharest

LOCAL SEARCH ALGORITHMS AND OPTIMIZATION PROBLEMS

Local Search Algorithms:

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution. In such cases, we can **use local search algorithms. They operate using a single current state (rather than multiple paths) and generally move only to neighbors of that state.**
- The important applications of these class of problems are

  (a) Integrated-circuit design,

  (b) Factory-floor layout,

  (c) Job-shop scheduling,

  (d) Automatic programming,

  (e) Telecommunications network optimization,

  (f) Vehicle routing, and

  (g) Portfolio management.
- Key advantages of Local Search Algorithms

  (1) They use very little memory – usually a constant amount; and

  (2) they can often find reasonable solutions in large or infinite(continuous) state spaces for which systematic algorithms are unsuitable.

Optimization Problems

An local search algorithms are useful for solving pure **optimization problems, in which the aim is to find the best state according to an objective function.**

1. State Space Landscape:

- A landscape has both "**location" (defined by the state) and "elevation" (defined by the** value of the heuristic cost function or objective function).
- If elevation corresponds to **cost, then the aim is to find the lowest valley – a global minimum; if elevation corresponds to an objective function, then the aim is to find the highest peak – a global maximum.**

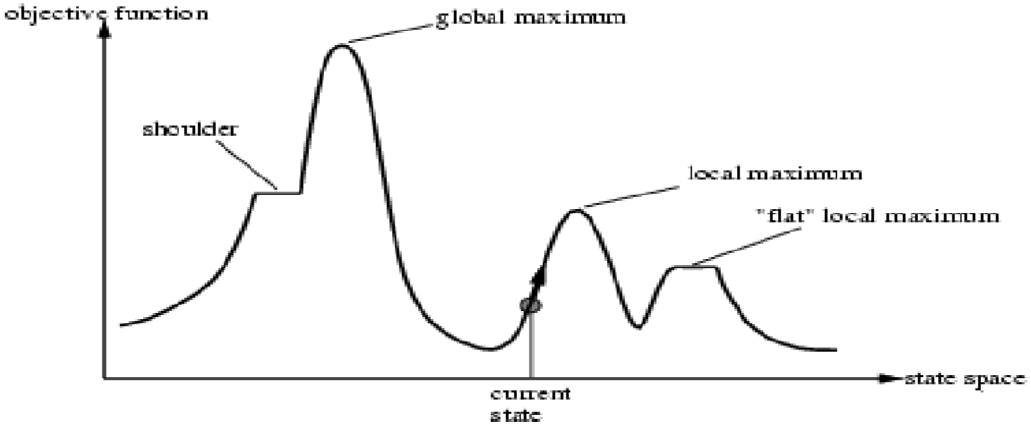


Figure 2.23 A one dimensional state space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum. Hill climbing search modifies the current state to try to improve it, as shown by the arrow. The various topographic features are defined in the text

2. Hill-climbing search

The **hill-climbing search algorithm is simply a loop that continually** moves in the direction of increasing value – that is, **uphill. It terminates when it reaches a "peak" where no neighbor has a higher value.**

```
function HILL-CLIMBING( problem) return a state that is a local maximum
        input: problem, a problem
 local variables: current, a
                          node.
             neighbor, a node.


        current ←MAKE-NODE(INITIAL-STATE[problem])
        loop do
                neighbor ← a highest valued successor of current
                if VALUE [neighbor] ≤ VALUE[current] then return STATE[current]
                current ←neighbor
```

Figure 2.24 The hill-climbing search algorithm (steepest ascent version), which is the most basic local search technique. At each step the current node is replaced by the best neighbor; the neighbor with the highest VALUE. If the heuristic cost estimate h is used, we could find the neighbor with the lowest h.

3. Simulated annealing search:

Simulated annealing is an algorithm that combines hill-climbing with a random walk in someway that yields both efficiency and completeness.

```
function SIMULATED-ANNEALING( problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"
   local variables: current, a node
                    next, a node
                    T, a "temperature" controlling prob. of downward steps

   current ← MAKE-NODE(INITIAL-STATE[problem])
   for t ← 1 to ∞ do
       T ← schedule[t]
       if T = 0 then return current
       next ← a randomly selected successor of current
       ΔE ← VALUE[next] – VALUE[current]
       if ΔE > 0 then current ← next
       else current ← next only with probability e^(Δ E/T)
```

Figure 2.26 The simulated annealing search algorithm, a version of stochastic hill climbing where some downhill moves are allowed.

4. Genetic algorithms

A Genetic algorithm (or GA) is a variant of stochastic beam search in which successor states are generated by combining two parent states, rather than by modifying a single state.

```
function GENETIC_ALGORITHM( population, FITNESS-FN) return an individual
    input: population, a set of individuals
        FITNESS-FN, a function which determines the quality of the individual
    repeat
        new_population←empty set
        loop for ifrom 1 to SIZE(population) do
         x ←RANDOM_SELECTION(population, FITNESS_FN)
       y ←RANDOM_SELECTION(population,
                FITNESS_FN)
         child ←REPRODUCE(x,y)
              if (small random probability) then child 
              MUTATE(child ) add child to new_population
        population ←new_population
    until some individual is fit enough or enough time has elapsed
    return the best individual
```

Figure 2.28 A genetic algorithm.

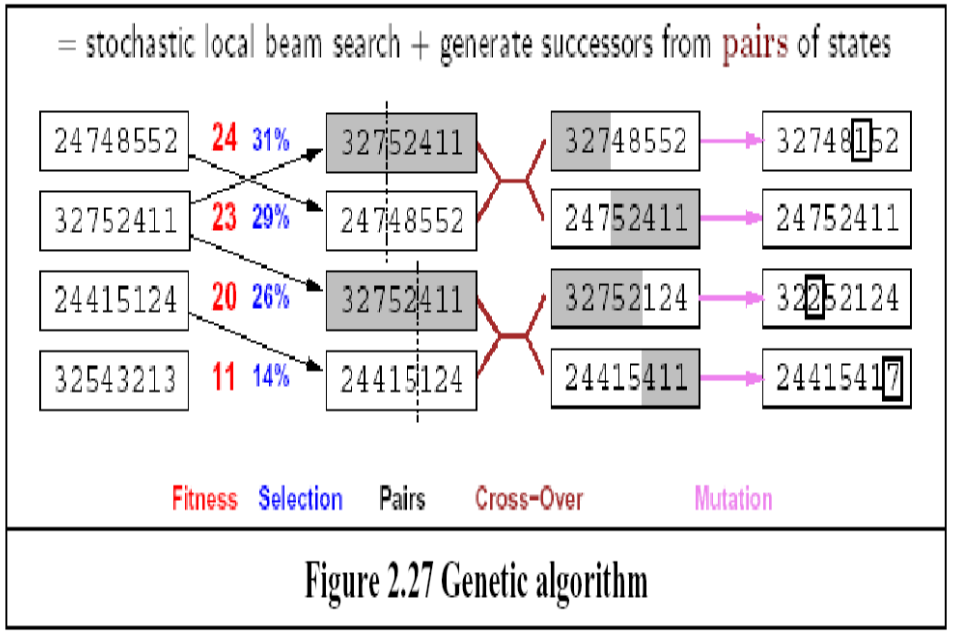


Figure 2.27 Genetic algorithm

SEARCHING WITH PARTIAL INFORMATION

Different types of incompleteness lead to three distinct problem types:

- **Sensorless problems (conformant):** If the agent has no sensors at all.
- **Contingency problem:** if the environment if partially observable or if action are uncertain (adversarial).
- **Exploration problems:** When the states and actions of the environment are unknown.

- No sensor
- Initial State(1,2,3,4,5,6,7,8)
- After action [Right] the state (2,4,6,8)
- After action [Suck] the state (4, 8)
- After action [Left] the state (3,7)
- After action [Suck] the state (8)
- Answer : [Right, Suck, Left, Suck] coerce the world into state 7 without any sensor
- Belief State: Such state that agent belief to be there

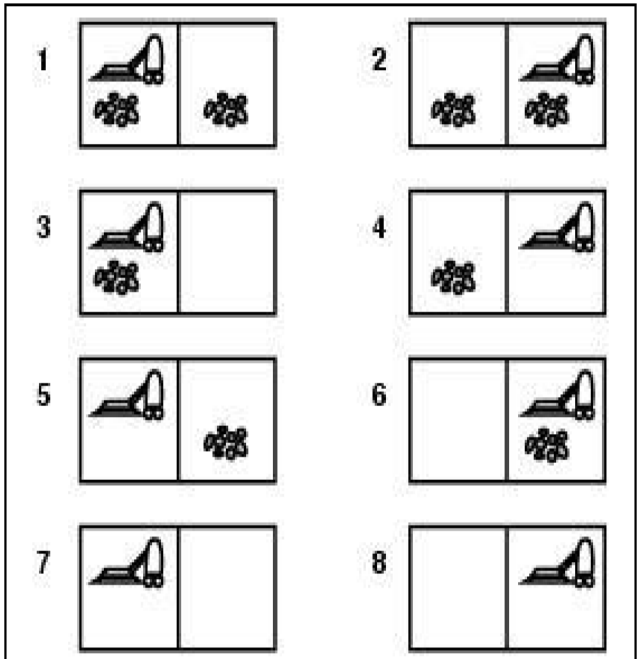


Figure 2.17 states and actions of the environment are unknown

ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

Unit -3

KNOWLEDGE REPRESENTATION

- First Order Predicate Logic
- Prolog Programming
- Unification
- Forward Chaining-
- Backward Chaining
- Resolution
- Knowledge Representation
- Ontological Engineering
- Categories and Objects
- Events
- Mental Events and Mental Objects
- Reasoning Systems for Categories
- Reasoning with Default Information.

FIRST ORDER PREDICATE LOGIC

- 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 a more easy 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, colors, 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 color, comes between
  - **Function:** Father of, best friend, third inning of, end of, ...

- As a natural language, first-order logic also has two main parts:
  - **Syntax**
  - **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.

Basic Elements of First-order logic:

Following are the basic elements of FOL syntax:

Constant	1, 2, A, John, Mumbai, cat,....
Variables	x, y, z, a, b,....
Predicates	Brother, Father, >,....
Function	sqrt, LeftLegOf, ...
Connectives	∧, ∨, ¬, ⇒, ⇔
Equality	==
Quantifier	∀, ∃

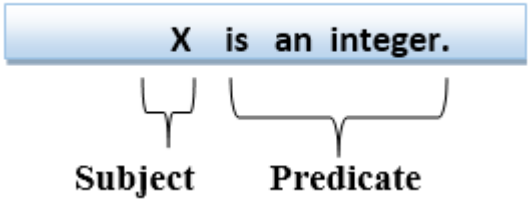
Atomic sentences:

- Atomic sentences are the most basic sentences of first-order logic. These sentences are formed from a predicate symbol followed by a parenthesis with a sequence of terms.
- We can represent atomic sentences as **Predicate (term1, term2, ..., term n)**.
- **Example: Ravi and Ajay are brothers: => Brothers(Ravi, Ajay). Chinky is a cat: => cat (Chinky)**.

Complex Sentences:

- Complex sentences are made by combining atomic sentences using connectives.
- **First-order logic statements can be divided into two parts:**
  - ✓ **Subject:** Subject is the main part of the statement.
  - ✓ **Predicate:** A predicate can be defined as a relation, which binds two atoms together in a statement.

✓ **Consider the statement:**



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 quantifier:
    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:

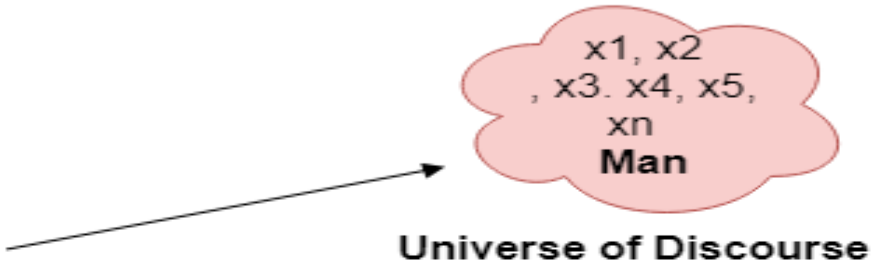
- **x1 drinks coffee**

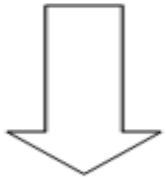
  ∧
- **x2 drinks**

  ∧
- **x3 drinks milk**

  ∧
- .
- .

  ∧
- **xn drinks milk**





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:

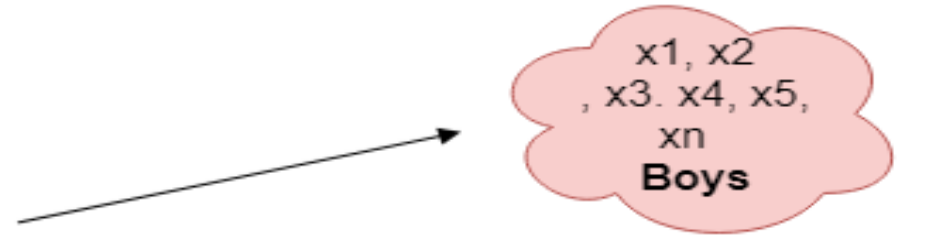
Some boys are intelligent.

- **x1 is intelligent**

  V
- **x2 is intelligentV**
- **x3 is intelligent**

  V
- .
- .

  V
- **xn is intelligent**



Universe of Discourse

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

∃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)].

Free and Bound Variables:

- The quantifiers interact with variables which appear in a suitable way. There are two types of variables in First-order logic which are given below:
- **Free Variable:** A variable is said to be a free variable in a formula if it occurs outside the scope of the quantifier.

  Example: ∀x ∃(y)[P (x, y, z)], where z is a free variable.

- **Bound Variable:** A variable is said to be a bound variable in a formula if it occurs within the scope of the quantifier.

  Example: ∀x [A (x) B(y)], here x and y are the bound variables.

Prolog Programming

Prolog is a logic programming language. It has important role in artificial intelligence. Unlike many other programming languages, Prolog is intended primarily as a declarative programming language

Key Features :

1. Unification : The basic idea is, can the given terms be made to represent the same structure.

2. Backtracking : When a task fails, prolog traces backwards and tries to satisfy previous task.

3. Recursion : Recursion is the basis for any search in program.

Advantages :

1. Easy to build database. Doesn't need a lot of programming effort.

2. Pattern matching is easy. Search is recursion based.

3. It has built in list handling. Makes it easier to play with any algorithm involving lists.

Disadvantages :

1. LISP (another logic programming language) dominates over prolog with respect to I/O features.

2. Sometimes input and output is not easy.

Applications :

- Prolog is highly used in artificial intelligence(AI). Prolog is also used for pattern matching over natural language parse trees.

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, $\boldsymbol{\Psi_1\sigma = \Psi_2}$□, 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:

Following are some basic 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.

Unification Algorithm:

Algorithm: Unify(Ψ_1, Ψ_2)

Step. 1: If Ψ_1 or Ψ_2 is a variable or constant, then:

a) If Ψ_1 or Ψ_2 are identical, then return NIL.

b) Else if $\Psi_1$is a variable,

a. then if Ψ_1 occurs in Ψ_2, then return FAILURE

b. Else return { (Ψ_2/ Ψ_1)}.

c) Else if Ψ_2 is a variable,

a. If Ψ_2 occurs in Ψ_1 then return FAILURE,

b. Else return {(Ψ_1/ Ψ_2)}.

d) Else return FAILURE.

Step.2: If the initial Predicate symbol in Ψ_1 and Ψ_2 are not same, then return FAILURE.

Step. 3: IF Ψ_1 and Ψ_2 have a different number of arguments, then return FAILURE.

Step. 4: Set Substitution set(SUBST) to NIL.

Step. 5: For i=1 to the number of elements in Ψ_1.

a) Call Unify function with the ith element of Ψ_1 and ith element of Ψ_2, and put the result into S.

b) If S = failure then returns Failure

c) If S ≠ NIL then do,

a. Apply S to the remainder of both L1 and L2.

b. SUBST= APPEND(S, SUBST).

Step.6: Return SUBST.

Implementation of the Algorithm

Step.1: Initialize the substitution set to be empty.

Step.2: Recursively unify atomic sentences:

- Check for Identical expression match.
- If one expression is a variable v_i, and the other is a term t_i which does not contain variable v_i, then:
  - Substitute t_i / v_i in the existing substitutions
  - Add t_i /v_i to the substitution setlist.
  - If both the expressions are functions, then function name must be similar, and the number of arguments must be the same in both the expression.

Inference engine

- The inference engine is the component of the intelligent system in artificial intelligence, which applies logical rules to the knowledge base to infer new information from known facts.
- The first inference engine was part of the expert system. Inference engine commonly proceeds in two modes, which are:

  a. Forward chaining

  b. Backward chaining

Forward-Chaining

- It is a down-up approach, as it moves from bottom to top.
- It is a process of making a conclusion based on known facts or data, by starting from the initial state and reaches the goal state.
- Forward-chaining approach is also called as data-driven as we reach to the goal using available data.
- Forward -chaining approach is commonly used in the expert system, such as CLIPS, business, and production rule systems.

Facts Conversion into FOL

- It is a crime for an American to sell weapons to hostile nations. (Let's say p, q, and r are variables)

  American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) …(1)

- Country A has some missiles. ∃p Owns(A, p) ∧ Missile(p). It can be written in twodefinite clauses by using Existential Instantiation, introducing new Constant T1.

  Owns(A, T1) …(2)

  Missile(T1) …(3)

- All of the missiles were sold to country A by Robert.

  ∀p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) …(4)

- Missiles are weapons.

  Missile(p) → Weapons (p) …(5)

- Enemy of America is known as hostile.

  Enemy(p, America) →Hostile(p) …(6)

- Country A is an enemy of America.

  Enemy (A, America) …(7)

- Robert is American

  American(Robert). …(8)

Forward chaining proof

Step-1

In the first step we will start with the known facts and will choose the sentences which do not have implications, such as: American (Robert), Enemy(A, America), Owns(A, T1), and Missile(T1). All these facts will be represented as below.

Figure 3.5

Step-2

At the second step, we will see those facts which infer from available facts and with satisfied premises.

Rule-(1) does not satisfy premises, so it will not be added in the first iteration.

Rule-(2) and (3) are already added.

Rule-(4) satisfy with the substitution {p/T1}, so Sells (Robert, T1, A) is added, which infers

from the conjunction of Rule (2) and (3).

Rule-(6) is satisfied with the substitution(p/A), so Hostile(A) is added and which infers from

Rule-(7).

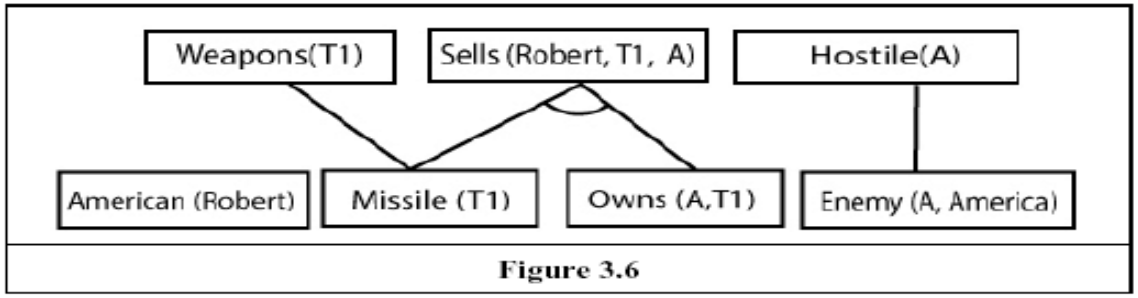


Figure 3.6

Step-3

- At step-3, as we can check Rule-(1) is satisfied with the substitution {p/Robert, q/T1,r/A}, so we can add Criminal (Robert) which infers all the available facts. And hence we reached our goal statement.

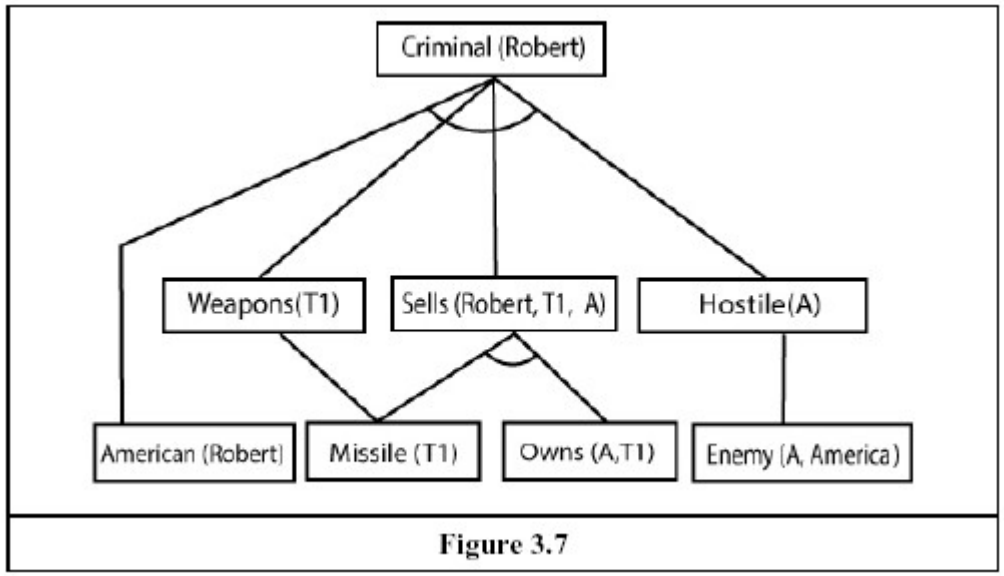


Figure 3.7

Hence it is proved that Robert is Criminal using forward chaining approach.

Backward chaining

- Backward-chaining is also known as a backward deduction or backward reasoning method when using an inference engine.
- It is known as a top-down approach.
- Backward-chaining is based on modus ponens inference rule.
- In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.
- It is called a goal-driven approach, as a list of goals decides which rules are selected and used.
- Backward -chaining algorithm is used in game theory, automated theorem proving tools, inference engines, proof assistants, and various AI applications.
- The backward-chaining method mostly used a depth-first search strategy for proof.

Example

- In backward-chaining, we will use the same above example, and will rewrite all the rules.

  American (p) $\wedge$ weapon(q) $\wedge$ sells (p, q, r) $\wedge$ hostile(r) $\rightarrow$ Criminal(p)…(1)

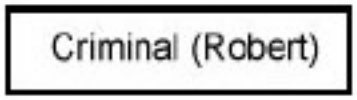
  Owns(A, T1) …(2)

- Missile(T1) …(3)
- ?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) …(4)
- Missile(p) → Weapons (p) …(5)
- Enemy(p, America) →Hostile(p) …(6)
- Enemy (A, America) …(7)
- American (Robert).

Backward-Chaining proof

Step-1

- At the first step, we will take the goal fact. And from the goal fact, we will infer other facts, and at last, we will prove those facts true. So our goal fact is "Robert is Criminal," so following is the predicate of it.



Step-2

- At the second step, we will infer other facts form goal fact which satisfies the rules. So as we can see in Rule-1, the goal predicate Criminal (Robert) is present with substitution {Robert/P}. So we will add all the conjunctive facts below the first level and will replace p with Robert.
- Here we can see American (Robert) is a fact, so it is proved here.

Step-3

- t At step-3, we will extract further fact Missile(q) which infer from Weapon(q), as it satisfies Rule-(5). Weapon (q) is also true with the substitution of a constant T1 at q.

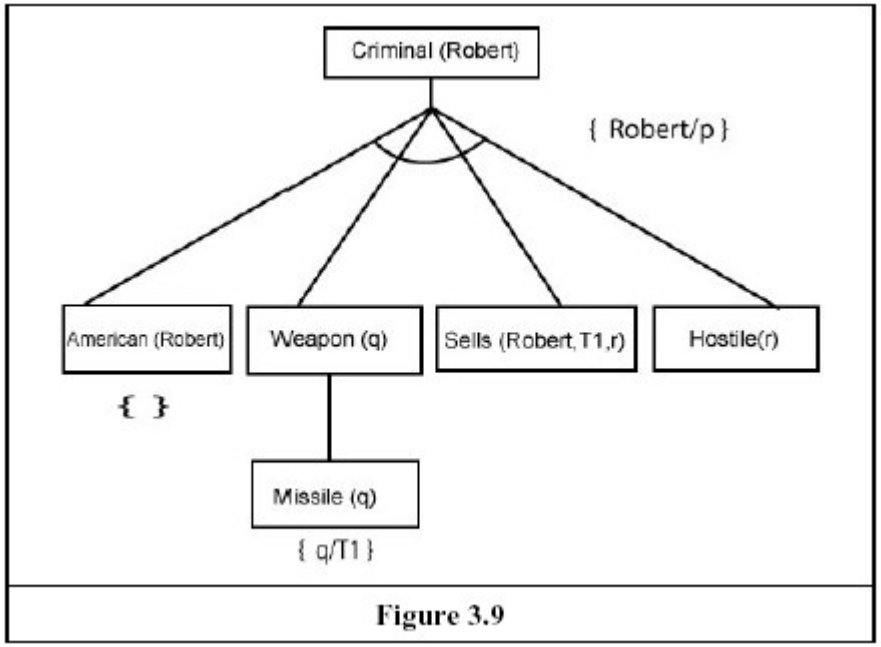


Figure 3.9

Step-4

- At step-4, we can infer facts Missile(T1) and Owns(A, T1) form Sells(Robert, T1, r) which satisfies the Rule- 4, with the substitution of A in place of r. So these two statements are proved here.

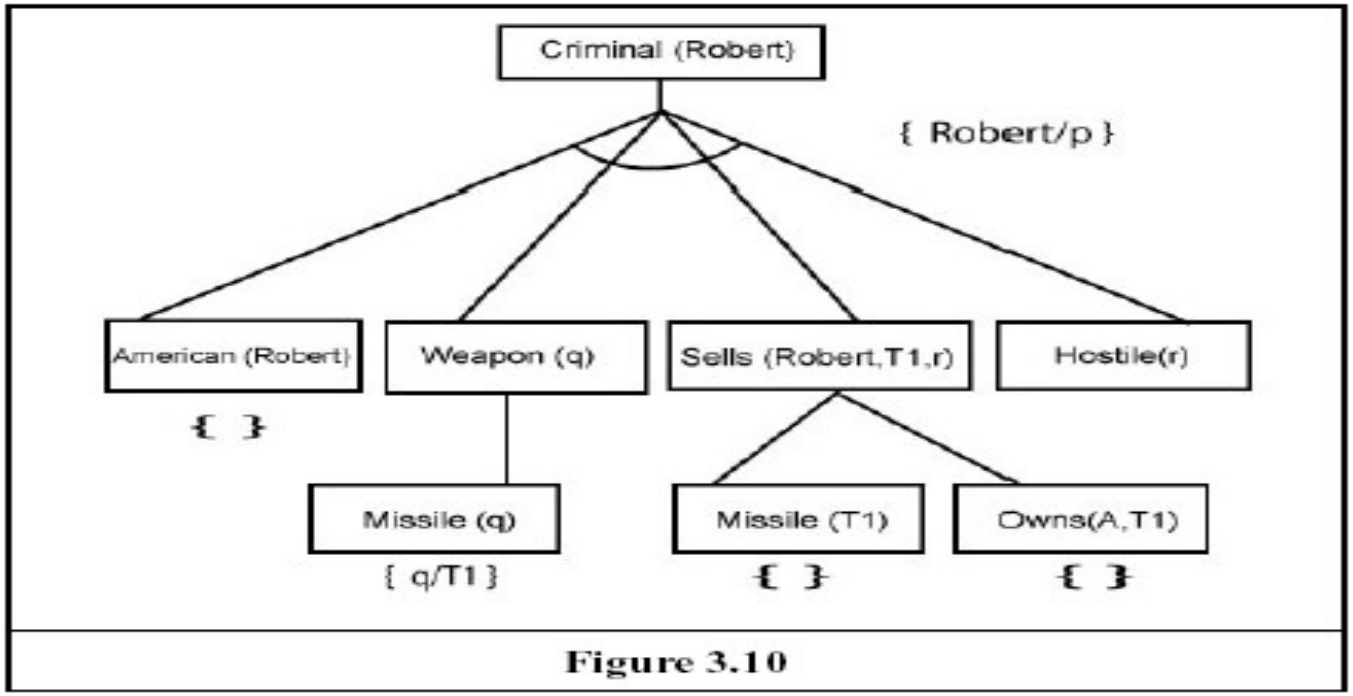


Figure 3.10

Step-5

- At step-5, we can infer the fact Enemy(A, America) from Hostile(A) which satisfies Rule- 6. And hence all the statements are proved true using backward chaining.

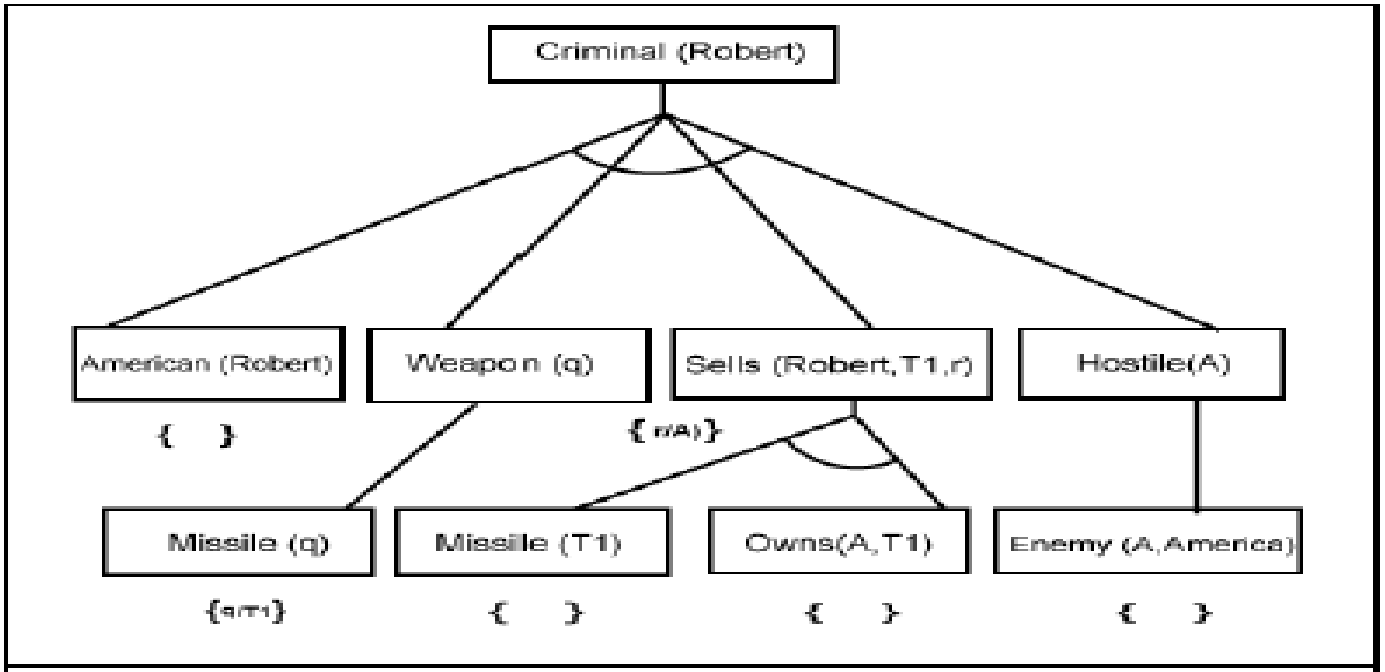


Figure 3.11

- Suppose you have a production system with the FOUR rules: R1: IF A AND C then F R2: IF A AND E, THEN G R3: IF B, THEN E R4: R3: IF G, THEN D and you have four initial facts: A, B, C, D. PROVE A&B TRUE THEN D IS TRUE. Explain what is meant by “forward chaining”, and show explicitly how it can be used in this case to determine new facts.

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**.

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. $\forall x$: food(x) $\rightarrow$ likes(John, x)

b. food(Apple) $\wedge$ food(vegetables)

c. $\forall x \forall y$: eats(x, y) $\wedge \neg$ killed(x) $\rightarrow$ food(y)

d. eats (Anil, Peanuts) $\wedge$ alive(Anil).

e. $\forall x$: eats(Anil, x) $\rightarrow$ eats(Harry, x)

f. $\forall x$: $\neg$ killed(x) $\rightarrow$ alive(x) } **added predicates.**

g. $\forall x$: alive(x) $\rightarrow \neg$ 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 ∧ over disjunction ¬.

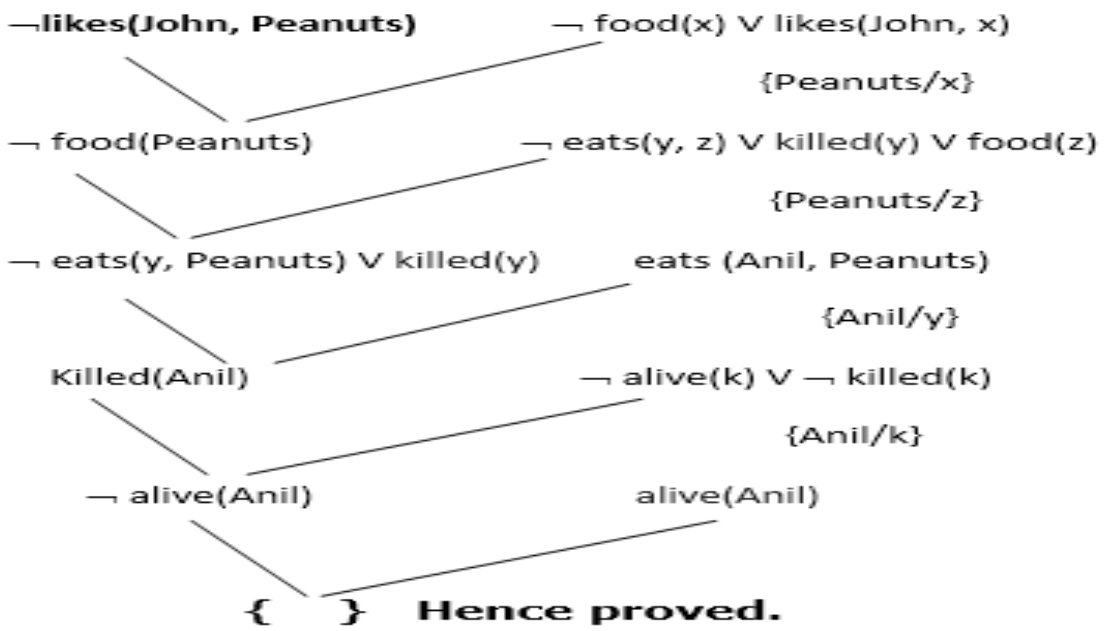
This step will not make any change in this problem.

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:

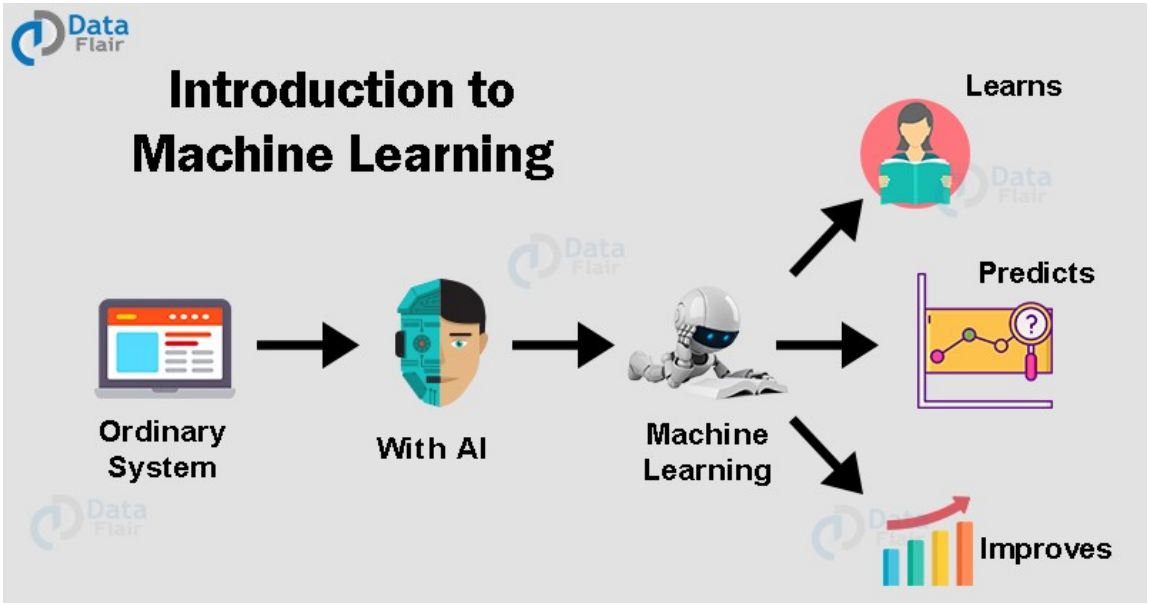


Hence the negation of the conclusion has been proved as a complete contradiction with the given set of statements.

UNIT 4

MACHINE LEARNING

- Machine learning (ML) is **a type of artificial intelligence (AI)** that allows software applications to become more accurate at predicting outcomes without being explicitly programmed to do so. Machine learning algorithms use historical data as input to predict new output values.



•Machine Learning is the most popular technique of **predicting** the **future** or **classifying information** to help people in making necessary decisions.

•Machine Learning algorithms are trained over instances or examples through which they learn from **past experiences** and also **analyze** the **historical data**.

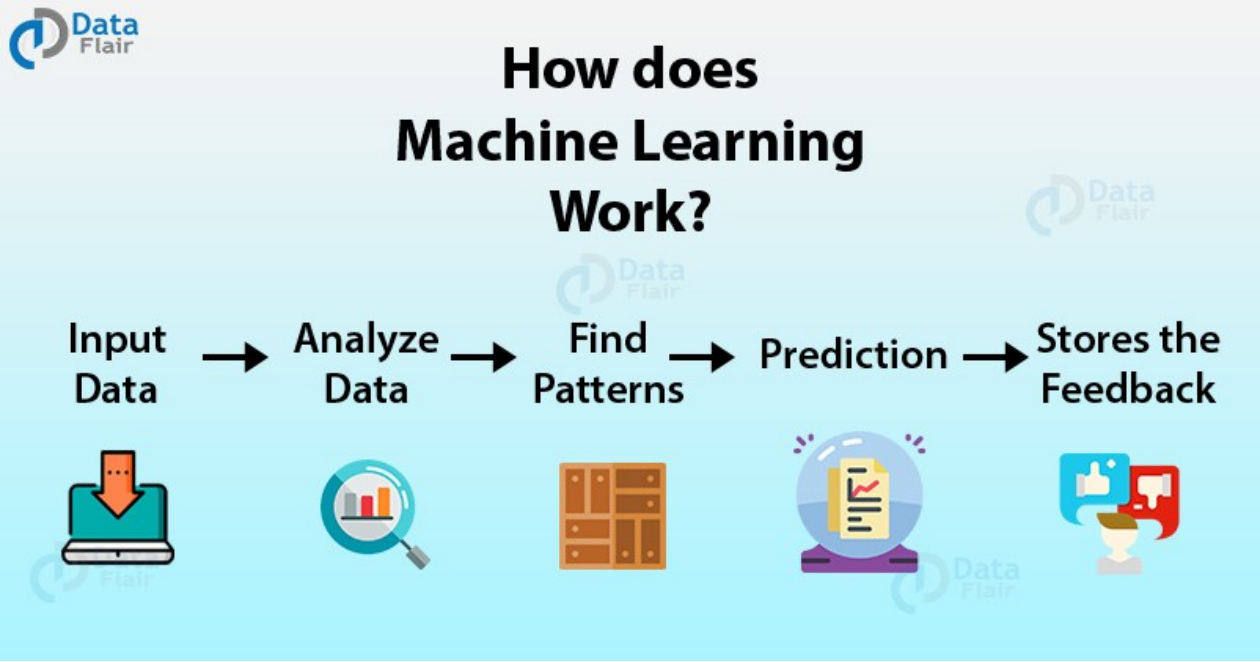
•Therefore, as it trains over the examples, again and again, it is able to identify patterns in order to make predictions about the future.

Why Machine Learning?

- Machine Learning has revolutionized industries like **medicine**, **healthcare**, **manufacturing**, **banking**, and several other industries. Therefore, Machine Learning has become an **essential part** of modern industry.
- Data is powerful and in order to harness the power of this data, added by the massive increase in computation power, Machine Learning has added another dimension to the way we perceive information. Example:
- The electronic devices we use is a **powerful machine learning algorithms**.
- Machine Learning example – **Google** is able to provide you with appropriate search results based on browsing habits.

How does machine learning works?

- With an exponential increase in data, there is a need for having a system that can handle this **massive load** of **data**.
- Machine Learning models like **Deep Learning** allow the vast majority of data to be handled with an **accurate generation** of **predictions**.
- Machine Learning has revolutionized the way we **perceive information** and the **various insights** we can gain out of it.



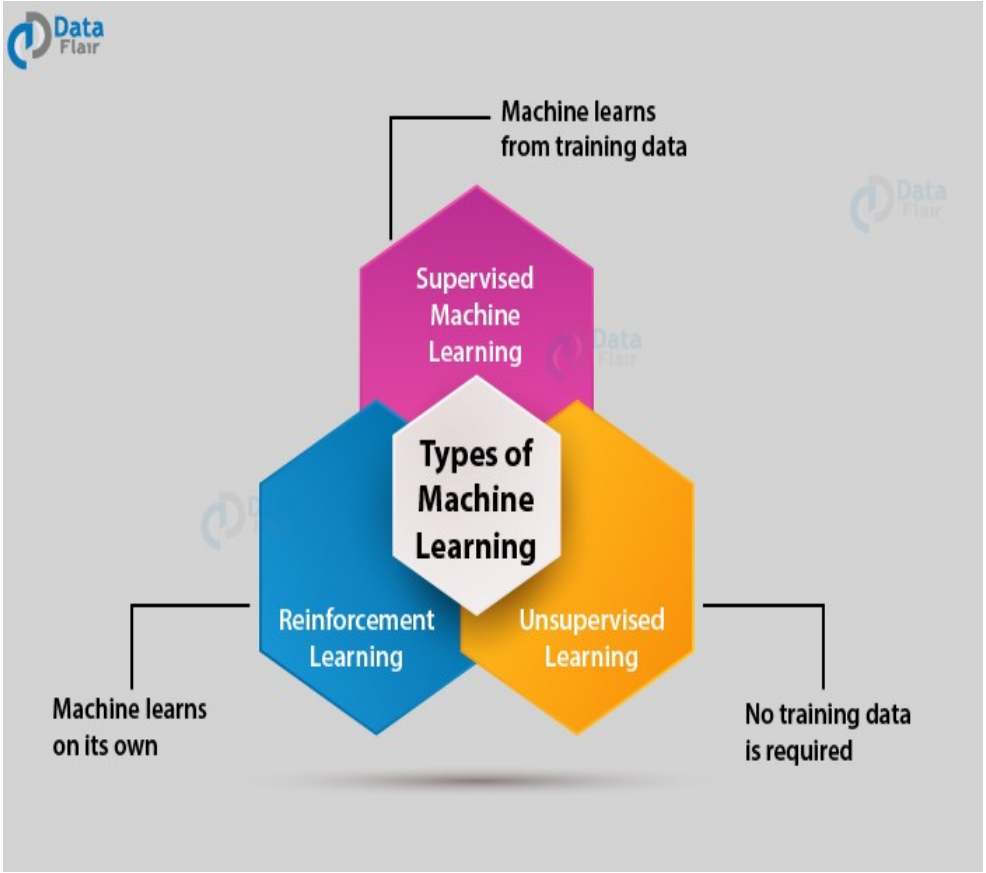
- These machine learning algorithms use the patterns contained in the training data to perform **classification** and **future predictions**.

- Whenever any new input is introduced to the **ML model**, it applies its learned patterns over the new data to **make future predictions**.

- Based on the final accuracy, one can **optimize** their models using various **standardized approaches**.

Types of ML

- **Supervised Learning**
- **Unsupervised Learning**
- **Reinforcement Learning**



Supervised Learning

- Supervised learning is that the machine learning task of learning a function that maps an **input** to an **output** supported example input-output pairs.
- In Supervised Learning, the dataset on which we train our model is **labeled**. There is a clear and **distinct mapping** of input and output. Based on the example inputs, the model is able to get **trained** in the **instances**.
- An example of supervised learning is **spam filtering**.

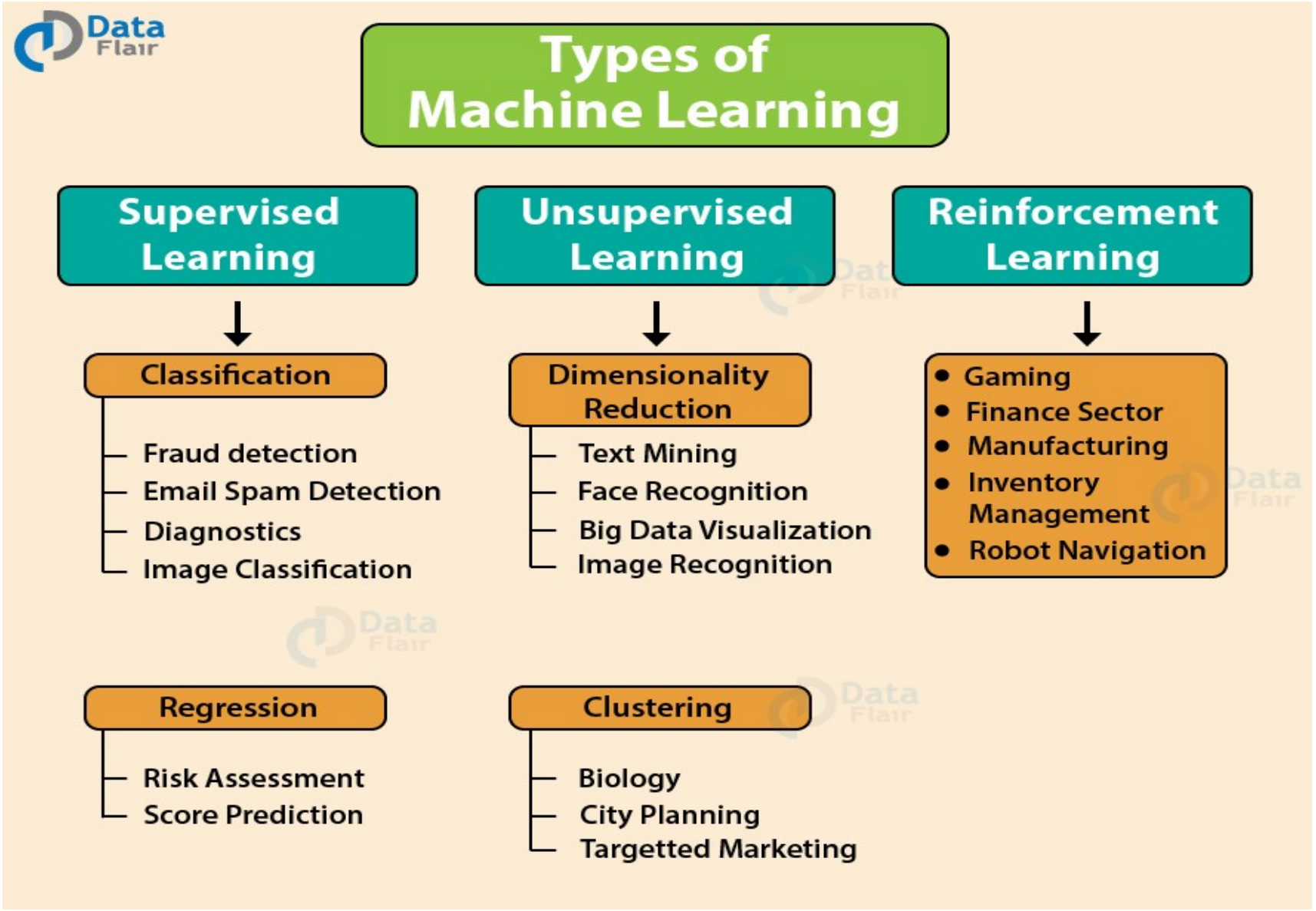
Unsupervised Learning

- It allows the model to figure on its own to get **patterns** and **knowledge** that was **previously undetected**. It mainly deals with the **unlabeled data**.
- In Unsupervised Learning, there is no labeled data. The algorithm identifies the **patterns** within the **dataset** and **learns** them. The algorithm groups the data into **various clusters** based on their **density.** Using it, one can perform **visualization** on **high dimensional data**.
- One example of this type of Machine learning algorithm is the **Principle Component Analysis**
- **K-Mean Cluster** is another type of Unsupervised Learning where the data is clustered in groups of a similar order. The learning process in Unsupervised Learning is solely on the basis of **finding patterns** in the **data**..

Reinforcement Learning

- Reinforcement learning is one among three basic machine learning paradigms, alongside supervised learning and unsupervised learning.
- Reinforcement Learning is an **emerging** and **most popular** type of Machine Learning Algorithm. It is used in various **autonomous systems** like **cars** and **industrial robotics**. The aim of this algorithm is to reach a goal in a **dynamic environment**. It can reach this **goal** based on several rewards that are provided to it by the system.

- It is most heavily used in **programming robots** to perform **autonomous actions**. It is also used in making **intelligent self-driving cars**.
- Let us consider the case of **robotic navigation**.
- Furthermore, the **efficiency** can be improved with further **experimentation** with the agent in its environment. This the main principle behind **reinforcement learning**.


Data Flair
Types of Machine Learning
Supervised Learning
Unsupervised Learning
Reinforcement Learning
Classification
Fraud detection
Email Spam Detection
Diagnostics
Image Classification
Dimensionality Reduction
Text Mining
Face Recognition
Big Data Visualization
Image Recognition
Gaming
Finance Sector
Manufacturing
Inventory Management
Robot Navigation
Regression
Risk Assessment
Score Prediction
Clustering
Biology
City Planning
Targetted Marketing

Machine Learning Algorithms

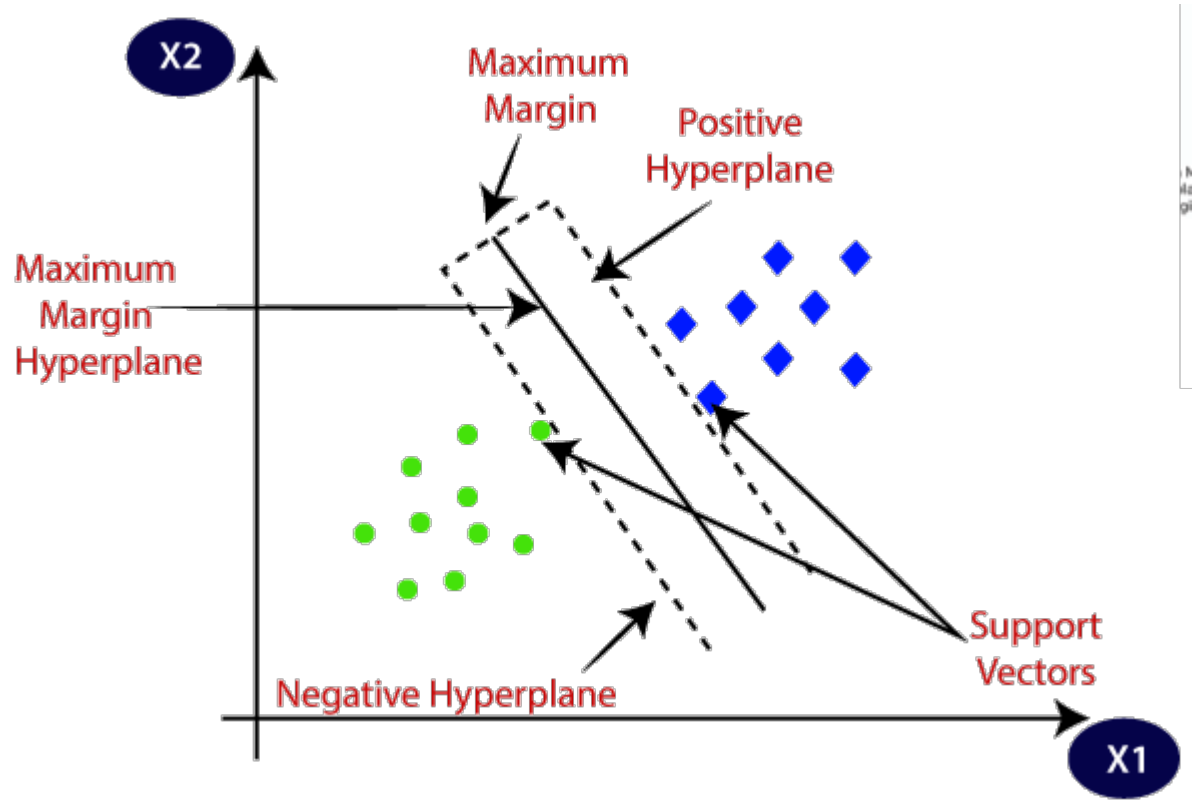
- Linear regression
- Logistic regression
- Decision tree
- SVM algorithm
- Naive Bayes algorithm
- KNN algorithm
- K-means
- Random forest algorithm
- Dimensionality reduction algorithms
- Gradient boosting algorithm and AdaBoosting algorithm

UNIT 5

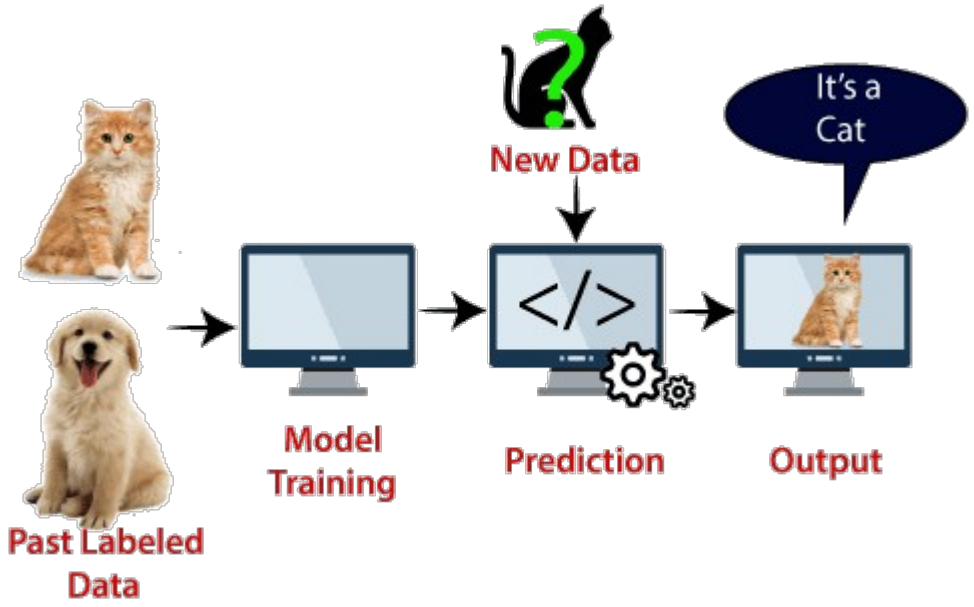
Support Vector Machine Algorithm

- Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems.
- However, primarily, it is used for Classification problems in Machine Learning.
- The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

- SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.



EXAMPLE



SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.

Types of SVM

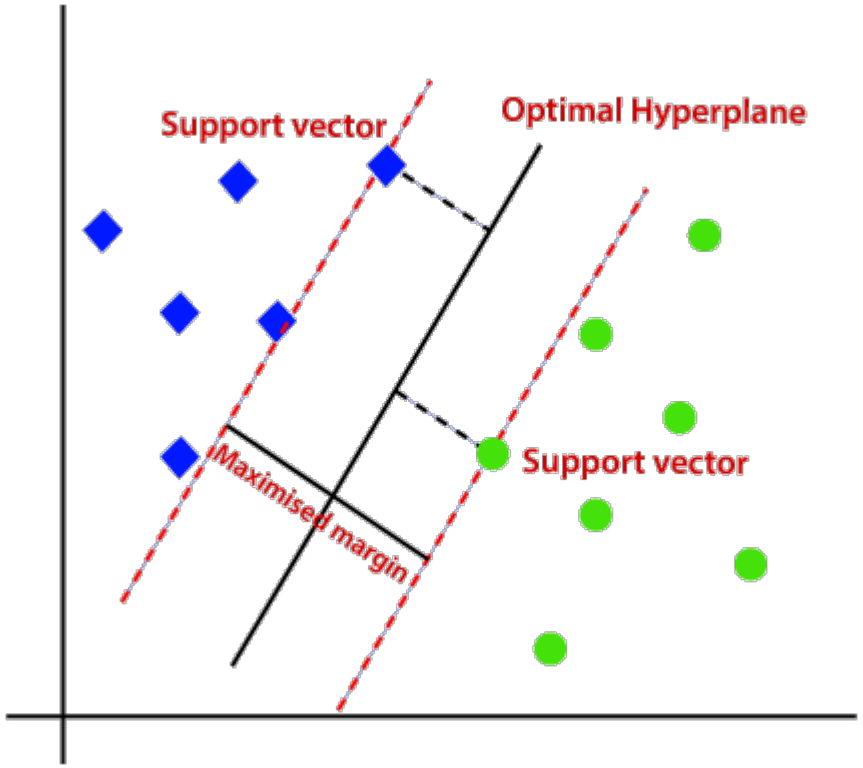
- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

Hyperplane and Support Vectors in the SVM algorithm:

- **Hyperplane:** There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.
- **Support Vectors:**
- The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

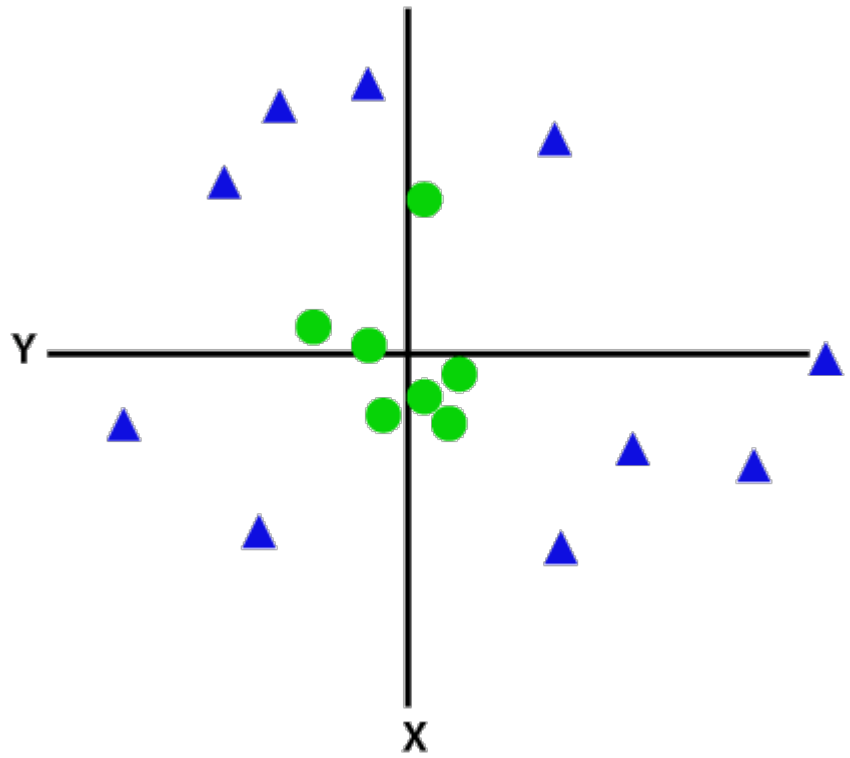
How does SVM works?

- **Linear SVM:**
- **2D space**
- **X1 and y1 is features (green, Blue)**
- **Classify these two feature by Using classifier alg.**

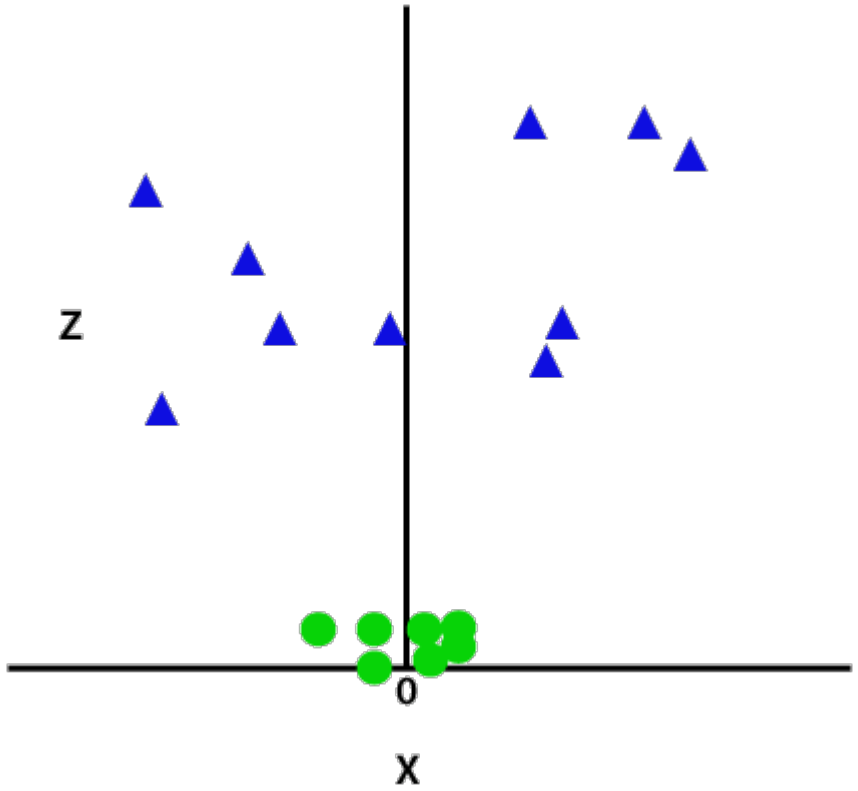


Non-Linear SVM:

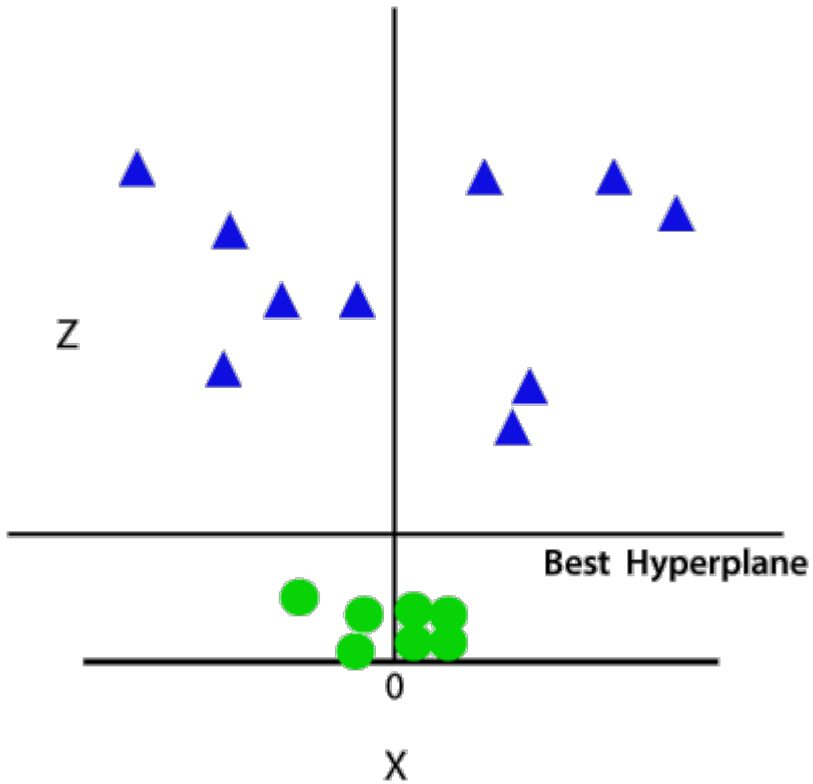
- Can’t draw single straight line



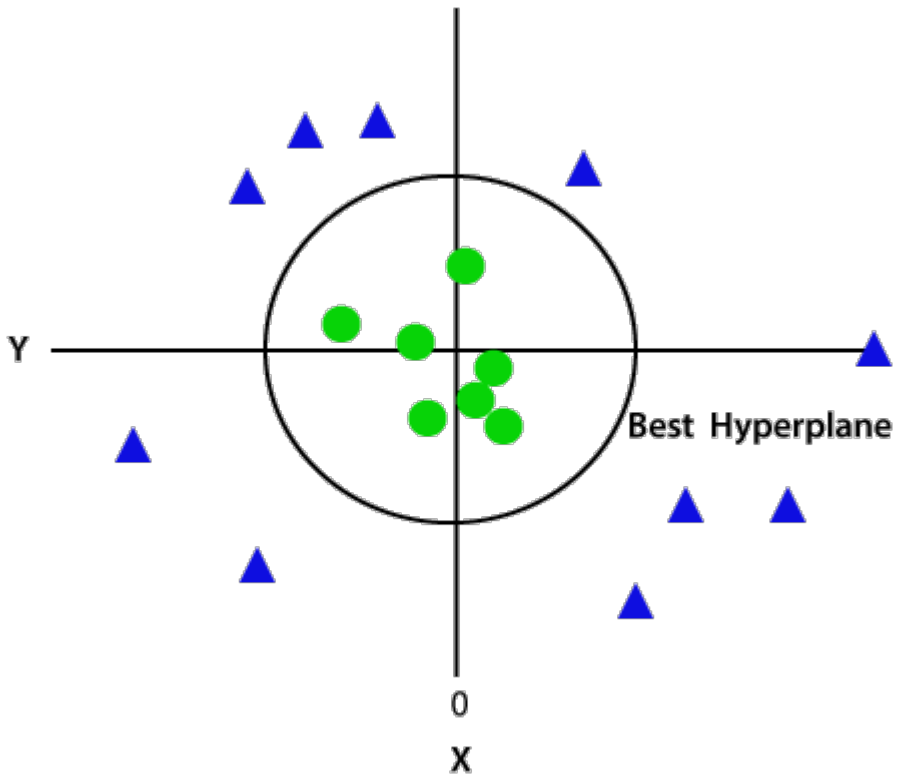
3D line



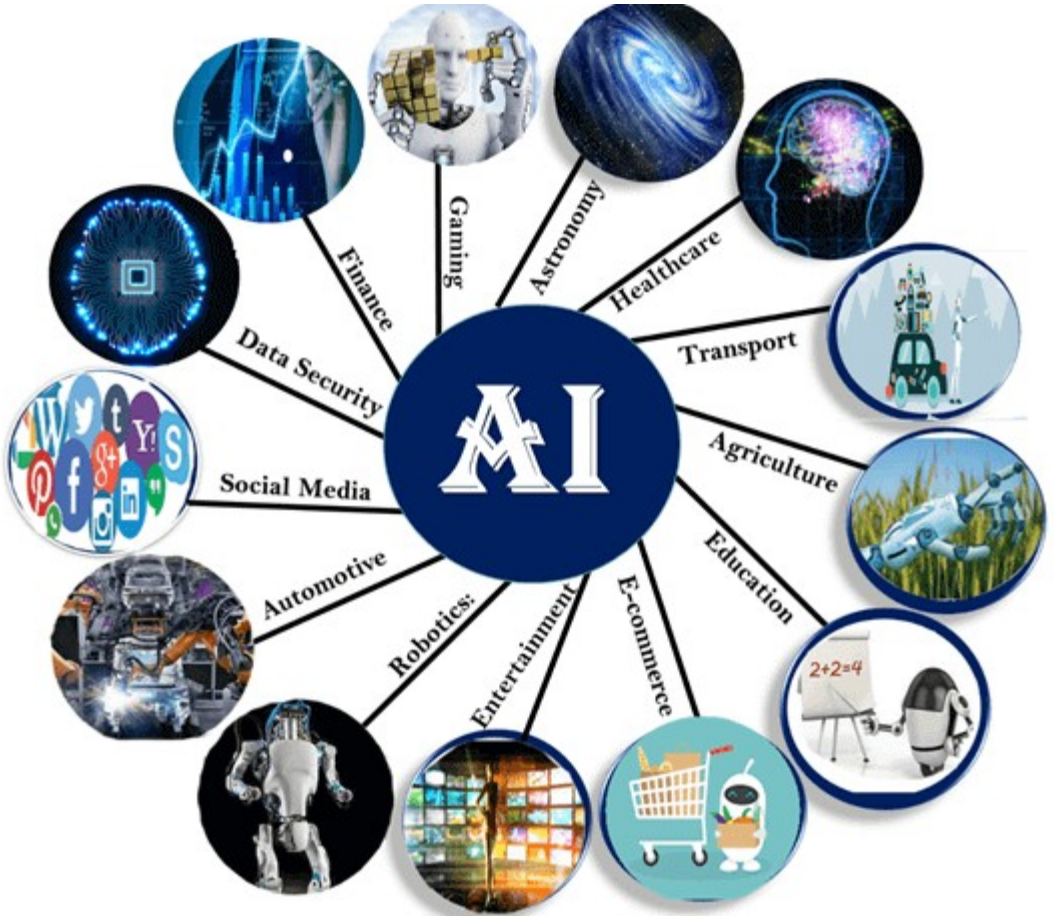
3D space



After converts 3D to 2D space

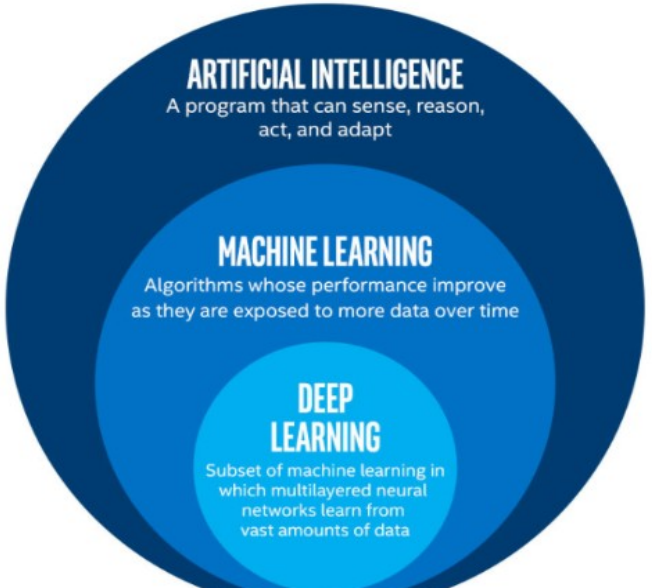


Application of AI



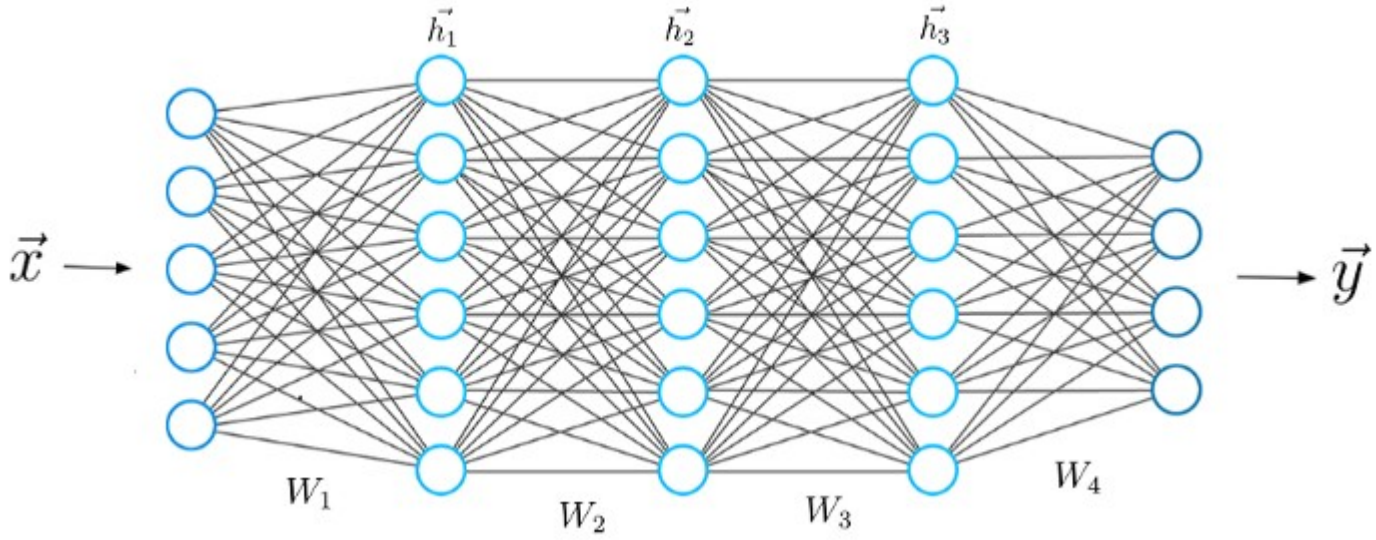
What exactly is Deep Learning?

- Deep Learning is a subset of Machine Learning, which on the other hand is a subset of Artificial Intelligence.
- Artificial Intelligence is a general term that refers to techniques that enable computers to mimic human behavior.
- Machine Learning represents a set of algorithms trained on data that make all of this possible.



- Deep Learning, on the other hand, is just a type of Machine Learning, inspired by the structure of a human brain.
- Deep learning algorithms attempt to draw similar conclusions as humans would by continually analyzing data with a given logical structure.
- To achieve this, deep learning uses a multi-layered structure of algorithms called neural networks

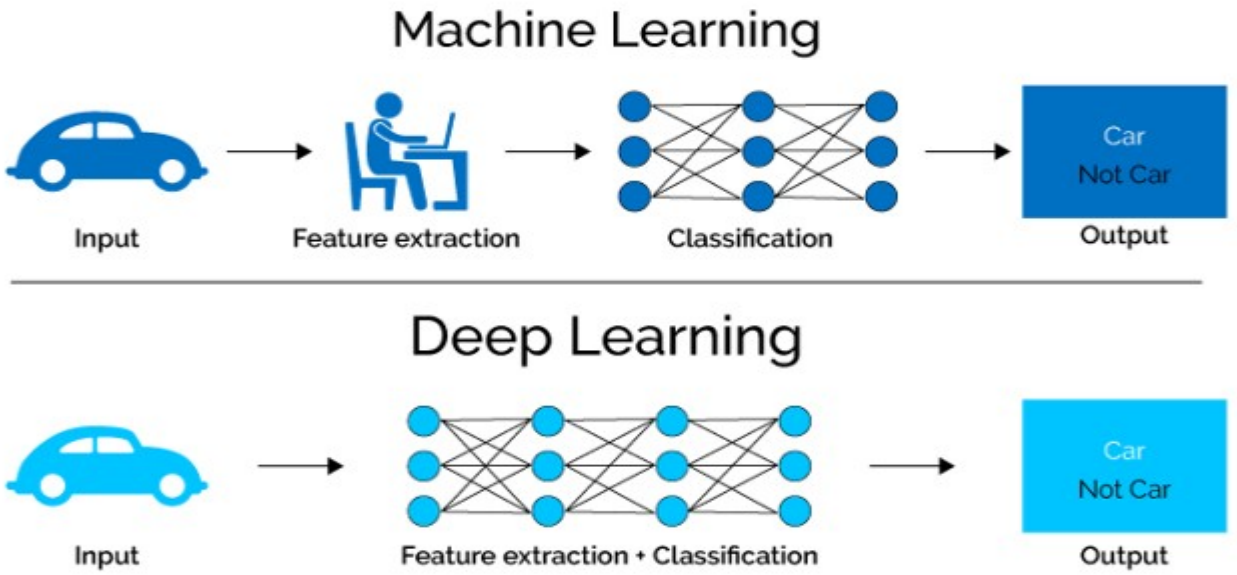
A neural network generally consists of a collection of connected units or nodes. We call these nodes neurons. These artificial neurons loosely model the biological neurons of our brain.



- **Neural networks enable us to perform many tasks, such as clustering, classification or regression.**
- With neural networks, we can group or sort unlabeled data according to similarities among the samples in this data. Or in the case of classification, we can train the network on a labeled dataset in order to classify the samples in this dataset into different categories.
- *In general, neural networks can perform the same tasks as classical algorithms of machine learning.*

Advantages

- *The first advantage of deep learning over machine learning is the needlessness of the so-called feature extraction.*
- Feature Extraction is usually quite complex and requires detailed knowledge of the problem domain. This preprocessing layer must be adapted, tested and refined over several iterations for optimal results.



- *Second thing is, the huge amounts of data we can feed to these algorithms*
- Deep Learning models tend to increase their accuracy with the increasing amount of training data, where's traditional machine learning models such as SVM and Naive Bayes classifier stop improving after a saturation point.

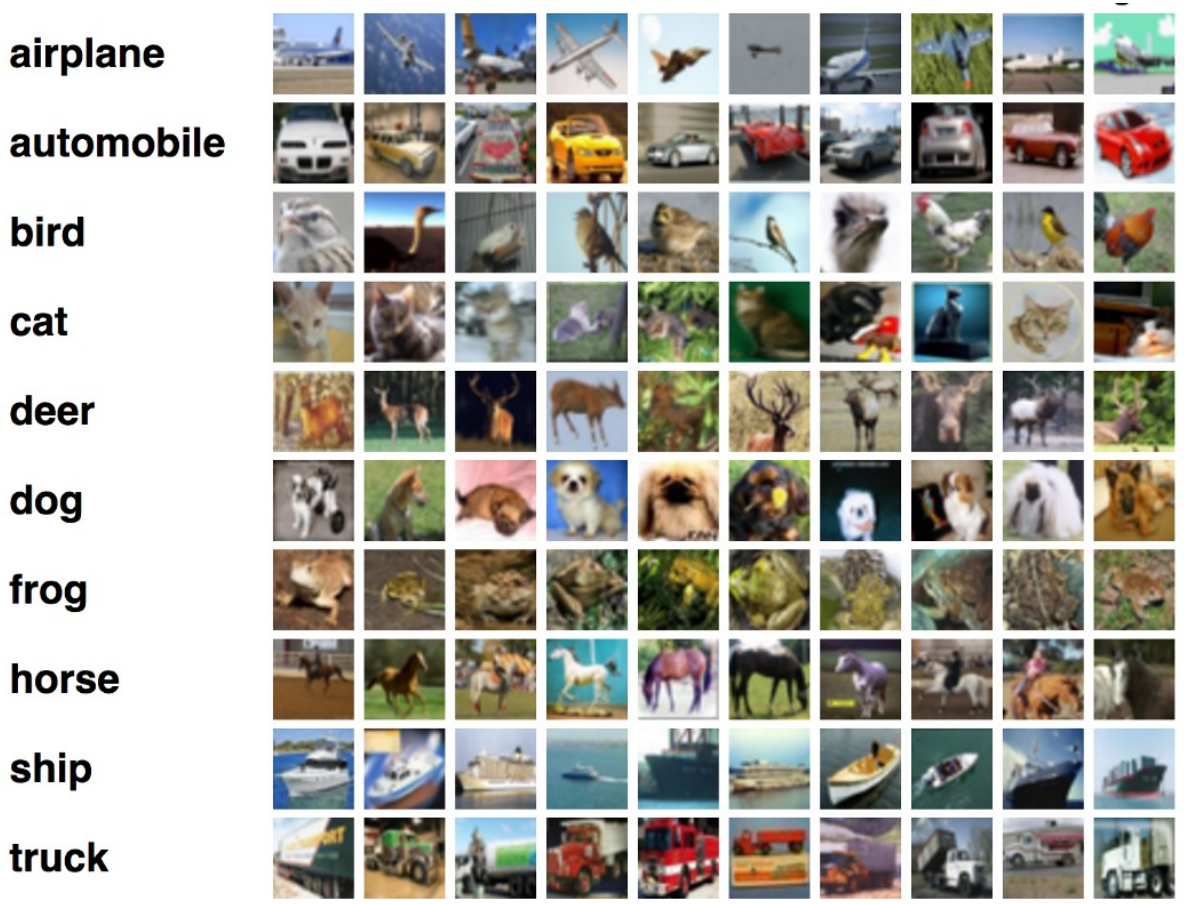
What Is Computer Vision?

- Computer vision is an area of machine learning dedicated to interpreting and understanding images and video. It is used to help teach computers to “see” and to use visual information to perform visual tasks that humans can.
- Computer vision models are designed to translate visual data based on features and contextual information identified during training. This enables models to interpret images and video and apply those interpretations to predictive or decision making tasks.

Image Classification

- Image classification is where a computer can analyse an image and identify the ‘class’ the image falls under. (Or a probability of the image being part of a ‘class’.) A class is essentially a label, for instance, ‘car’, ‘animal’, ‘building’ and so on.
- For example, you input an image of a sheep. Image classification is the process of the computer analysing the image and telling you it’s a sheep. (Or the probability that it’s a sheep.)

Teaching computer to recognize images and classify them



Robotics in Deep Learning

- Robotic platforms now deliver vast amounts of sensor data from large unstructured environments.
- Robots can learn, navigate, and make decisions all by themselves
- Important Robot Components
- The element that enables a robot to become a physical part of its surroundings are the components that are located on-board; Specifically, the **Sensors**.
- Sensors are crucial in a robotic system as they reduce the need for interaction, hence increasing the autonomous levels in a system.

A list of sensors that are available on the market are as follows

- Light sensors.
- – Temperature sensors.
- – Pressure sensors.
- – Position sensors.
- – Hall sensors.
- – Flex sensors.
- – Sound sensors.
- – Ultrasonic sensors.
- – Touch sensors.
- – PIR sensors.
- – Tilt sensors.
- – Gas sensors.