# Chapter 1: Introduction to AI

## Introduction:

What is intelligence?

Intelligence is the computational part of the ability to achieve goals in the world.

Intelligence means:

"the capacity to learn and solve problems ", in particular,

- The ability to solve novel problems
- The ability to act rationally
- The ability to act like humans

## Definition of AI:

Artificial Intelligence is the science and engineering that is concerned with the theory and practice of developing systems that exhibit the characteristics we associate with intelligence in human behavior: perception, natural language processing, reasoning, planning and problem solving, learning and adaption etc.

The goal of work in artificial intelligence is to build machines that perform tasks normally requiring human intelligence. ______ Nilsson, Nills J; 1971

Research scientists in AI try to get machines to exhibit behavior that we call intelligent behavior when we observe it in human beings. __________ Slagle, James R; 1971

## Four views of AI:

Views of AI fall into four categories

1. Thinking Humanly
2. Thinking rationally
3. Acting Humanly
4. Acting Rationally

| | |
|---|---|
| Think like Humans | Think Rationally |
| Act like Humans | Act Rationally |

*Compiled by:*
*Ranjan Raj Aryal*
*Add. Associate Professor*
*Cosmos College Of Managemant & Technology*

## 1 . Act Humanly: Turing Test:

The Turing Test, proposed by Alan Turing (Turing, 1950), was designed to provide a satisfactory operational definition of intelligence. Turing defined intelligent behavior as the ability to achieve human-level performance in all cognitive tasks, sufficient to fool an interrogator.

Roughly speaking, the test he proposed is that the computer should be interrogated by human via a teletype and passed the test if the interrogator cannot tell if there is a computer or a human at the end.

## 2. Thinking Humanly:

Cognitive Science approach:

- Try to get "inside" our minds.

For example, conduct experiments with people to try to "reverse-engineer" how we reason, learning, remember, predict.

## 3. Thinking Rationally:

- Represent facts about the world via logic
- Use logical inference as a basis for reasoning about these facts.

## 4.Acting Rationally:

1. Decision Theory/Economics

- Set of future states of the world
- Set of possible actions an agent can take
- Utility = gain to an agent for each action/state pair
- An agent acts rationally if it selects the action that maximizes its "utility" or expected utility if there is uncertainty

2. Emphasis is on autonomous agent that behave rationally (make the best predictions, take the best actions)

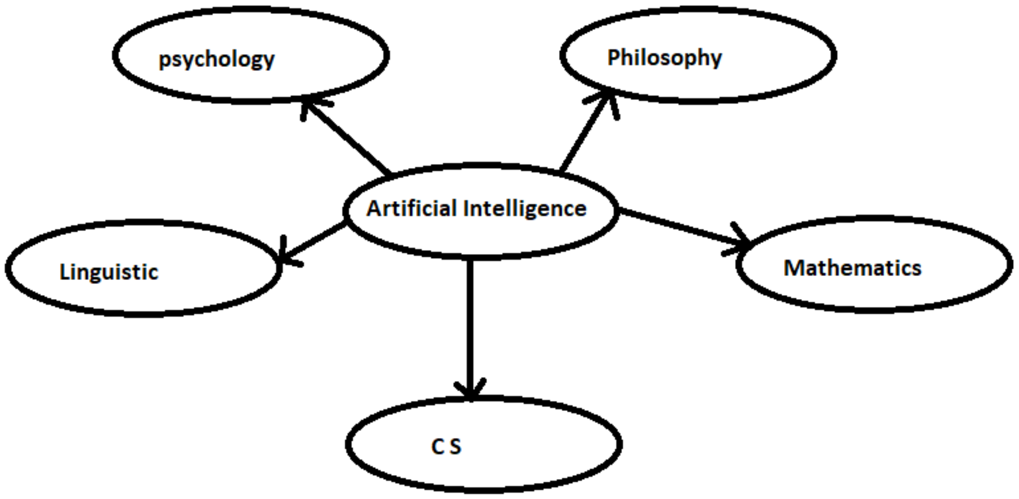
- On average over time
- Within computational limitations

## Applications of Artificial Intelligence:

1. Game playing: Machines that can play certain games e.g. Chess with great mastery mainly through brute (*characterized by an absence of reasoning or intelligence*) force computation which gives ability to look a hundred of thousands of positions per second
2. Speech and character recognition: The ability of computers to recognize voices and writing.

3. Natural Language Processing: Providing the computer with understanding of the domain of natural language.
4. Expert System: Systems that are able to make decisions and perform the work of professionals (human expert) e.g. diagnostic systems in hospitals
5. Robotics: Automation of tasks performed by a mechanical device through predefined programs.
6. Information predictors: For example, in banks, insurance companies, market surveys, whereas intelligence tools are used to detect trends and predict e. g customer behavior
7. Computer vision and pattern recognition: Computer processing of images from the real world and recognition of features present in images.

## AI and Related fields:



## Brief history of AI

1943 McCulloch & Walter Pills: Boolean circuit model of brain

1950 Turing's "Computing Machinery and Intelligence"

1950s Early AI programs, including Samuel's checkers program, Newell & Simon's logic Theorist, Gelernter's Geometry Engine.

1956 Dartmouth meeting: "Artificial Intelligence" adopted i.e. birth of AI

1956-58 LISP language developed

1965 Robinson's complete algorithm for logical reasoning

1966-74 AI discovers computational complexity Neural network research almost disappears

1969-79 Early development of knowledge-based system

| | |
|---|---|
| 1980-88 | Expert system industry booms |
| 1980 | AI becomes an industry |
| 1986 | Neural network return to popularity |
| 1987 | AI becomes a science |
| 1991 | AI system beats human chess master |
| 1995 | The emergence of intelligent agents |
| 2003 | Human-level AI back on the agenda |

## Definition and Importance of Knowledge:

Knowledge is an abstract term that attempts to capture an individual's understanding of a given subject. Knowledge is a subset of information. But it is a subset that has been extracted, filtered or formatted in a very special way. More specifically, the information we call knowledge is information that has been subjected to, and passed tests of validation. Common sense knowledge is information that has been validated by common sense experience.

Solving problems in a particular domain generally requires knowledge of the objects in the domain and knowledge of how to reason in that domain. The importance of domain knowledge in the area of understanding of domain knowledge in the area of understanding natural language is also apparent.

## Learning:

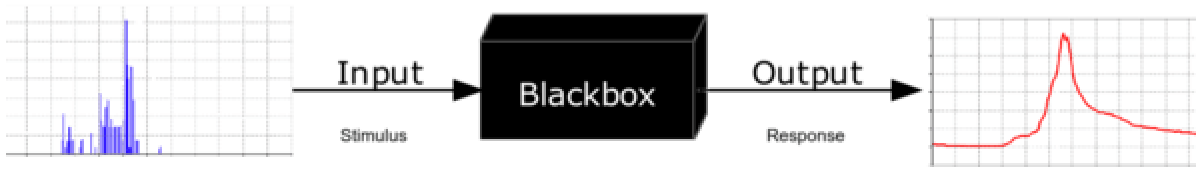
Learning is any process by which a system improves performance from experience. Learning is an area of AI that focuses on process of self-improvement. It implies that an organism or machine must be able to adopt to new situations. This requires knowledge acquisition, inference, updating/ refinement of knowledge base, acquisition of heuristics, applying faster searches etc.

# Chapter 2 Agent, Search and Game Playing

## Agent

An Agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators. This is analogous to a human agent has eye, ears and other organs for sensors and hands, legs, mouth, and other body parts for actuators. A robotic agent might have cameras and infrared range finders for sensors and various motors for actuators.

What is black box AI? Put simply it is the idea that we can understand what goes in and what comes out, but don't understand what goes on inside.



The job of AI is to design the agent program that implement the agent function mapping percepts to actions. We assume this program will run the architecture of computing device with physical sensors and actuators. This is called architecture.

agent=architecture + program

An example of a **taxi driver agent** is given below

| Agent Type | Percepts | Actions | Goals | Environment |
|---|---|---|---|---|
| Taxi Driver | Cameras, Speedometer, GPS, sonar, microphone | Steer, accelerate, brake, talk to passenger | Safe, fast, legal, comfortable trip, maximize profits | Roads, other traffic, pedestrians, customers |

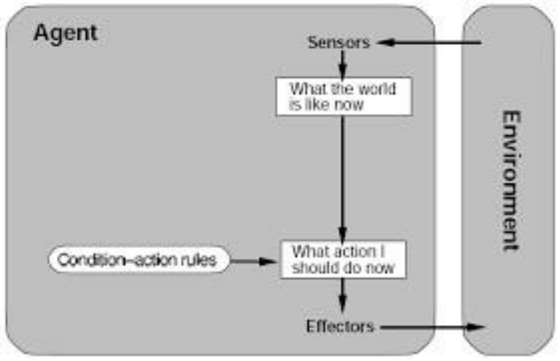
## Types of Agent Programs

- Simple reflex agents
- Goal-based Agents
- Model-based reflex agents
- Utility-based reflex agents

Compiled By:
Er.Ranjan Raj Aryal
Add. Associate Professor
Cosmos College of Management & Technology

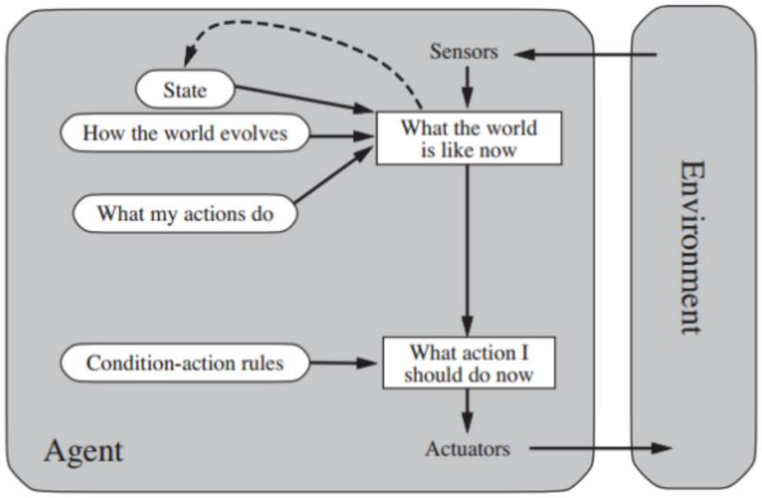
**a) Simple reflex agents**

- Simple agent which select actions on the basis of current percepts
- Useful for quick automated response
- E.g very similar nature of human to fire



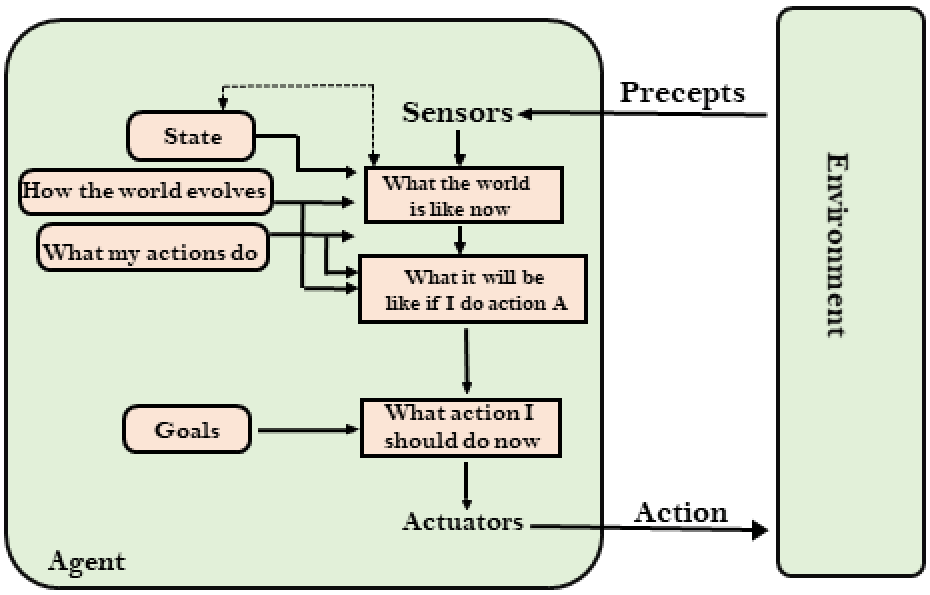
**b) Model-based reflex agents**

- made to deal with partial accessibility
- It keeps an internal state that depends on what it has seen before.
- To update we need two things:
  - Information on how the world evolves on its own
  - How the world is affected by the agents actions ?
  -



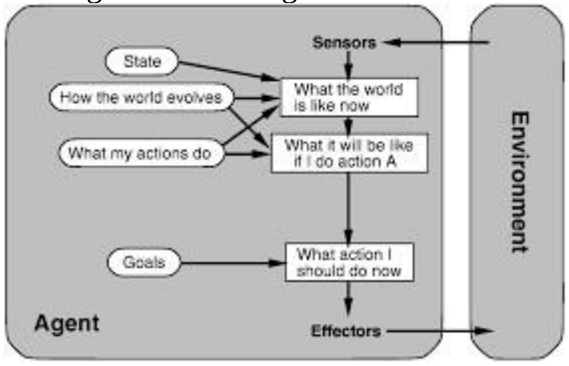
**c) Goal-based Agents**

- The agent can use goals with a set of actions and their predicted outcomes to see which actions achieve our goals
- Search and planning are two subfields to achieve an agent's goal.
- A simple example would be the shopping list; our goal is to pick up every thing on that list. This makes it easier to decide if you need to choose between milk and orange juice because you can only afford one.



**d) Utility-based reflex agents**

- A utility function maps each state after each action to real number representing how efficiently each action achieves the goal.
- They choose actions based on a preference (utility) for each state.
- Useful when many actions solving the same goal.
- Sometimes achieving the desired goal is not enough.
- We may look for a quicker, safer, cheaper trip to reach a destination.
- Because of the uncertainty in the world, a utility agent chooses the action that maximizes the expected utility.
- Example:
    - Quicker, safer, more reliable ways to get where going
    - Price comparison shopping
    - Bidding and evaluating bids on items in an auction


Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Goals
What action I should do now
Agent
Effectors
Environment

## Intentionality and Goals

Intentionality is the ability to think, feel and act in a deliberate way towards a purpose. It is a term of philosophy that comes up in relation to artificial intelligence. Where it is widely believed that intentionality is a common human trait and ability, it is less clear whether machines could ever possess such a dimension.

Most AI is focused on practical problems such as recognizing an image, driving a car or searching the internet. As such, goals are often more or less hardcoded into AI software. In other words, AI is currently mostly about learning how to answer a question, make a decision or predict something. It isn't about forming high level goals and acting with a purpose.

Thought experiments such as the Chinese Room suggest that AI is fundamentally different from consciousness because machines process symbols and don't "understand" them. If a machine were to develop consciousness, intentionality would be an early precursor. A super intelligent machine with intentionality could represent an existential as its goals might not align with the interests of humans.

| Overview: Intentionality | |
|---|---|
| Type | Thought Processes |
| Definition | The ability to think, feel and act in a deliberate way towards a purpose. |
| Related Concepts | Chinese Room<br>Artificial Intelligence<br>Thought Processes<br>Consciousness<br>Affective Computing<br>Superintelligence |

## Games

A game can be formally defined as a kind of search problem with the following components:

- **The initial state ,** which includes the board position and an indication of whose move it is.
- **A set of Operators,** which define the legal moves that a player can make.
- **A terminal Test,** which determines when the game is over. States where the game has ended are called terminal states.
- **Payoff function.** A utility function (also called a payoff function), which gives a numeric value for the outcome of a game. In chess, the outcome is a win, lose or draw, which we can represent by the values +1, -1 or 0. Some games have a wider variety of possible outcomes; for example, the payoffs in backgammon range from +192 to -192.

## Game Playing:

Game playing involves abstract and pure form of computation that seems to require intelligence. So, game playing has close relationship to intelligence and its well defined stated and rules. Game playing research has contributed ideas on how to make the best use of time to reach good decisions. Game playing is important in AI due to following reasons :

- The state of game is easy to represent.
- The rules of the game are limited and precise. Further the rules of the games are static, known to both players
- They provide a structured task. Therefore, success or failure can be easily measured.
- Games are examples from a class of problems concerned with reasoning about actions.
- Games simulate real life situation.

## Strategies Rules :

### Plies, Moves and Turns

A strategy is a list of the optimal choices for each player at each stage of a given game. It is common in game theory to refer to one player's turn as a "ply" of the game. One round of all the player's turns is called a "move". this originates in Chess, where one move consists of each player taking on turn. Because most turn-based AI is based on Chess- playing programs, the word "move" is often used in this context.

There are many more games, however, that treat each player's turn as a separate move and this is the terminology normally used in turn-based strategy games.

A game has at least two players. An instance of game begins with a player, choosing from a set of specified game rules. This choice is called a move. After first move, choosing from a set of specified game rules. This choice is called move. After first move , the new situation determines which player to make move and alternatives available to that player.

**Game Playing with Minimax Tic-Tac-Toe (TTT)**

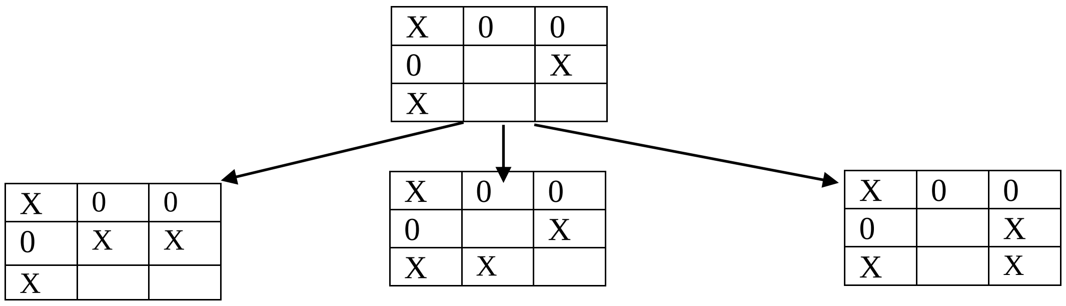
Assume that two players named X (MAX) and 0 (MIN) who are playing the game. MAX is playing first. Initially MAX has 9 possible moves. Play alternates between MAX and MIN until we reach leaf nodes corresponding to terminal states such that one player has 3 in a row or all squares are filled.

The number on each leaf node indicates the utility value of the terminal state from the point of view of MAX. High value is assumed to be good for MAX and bad for MIN

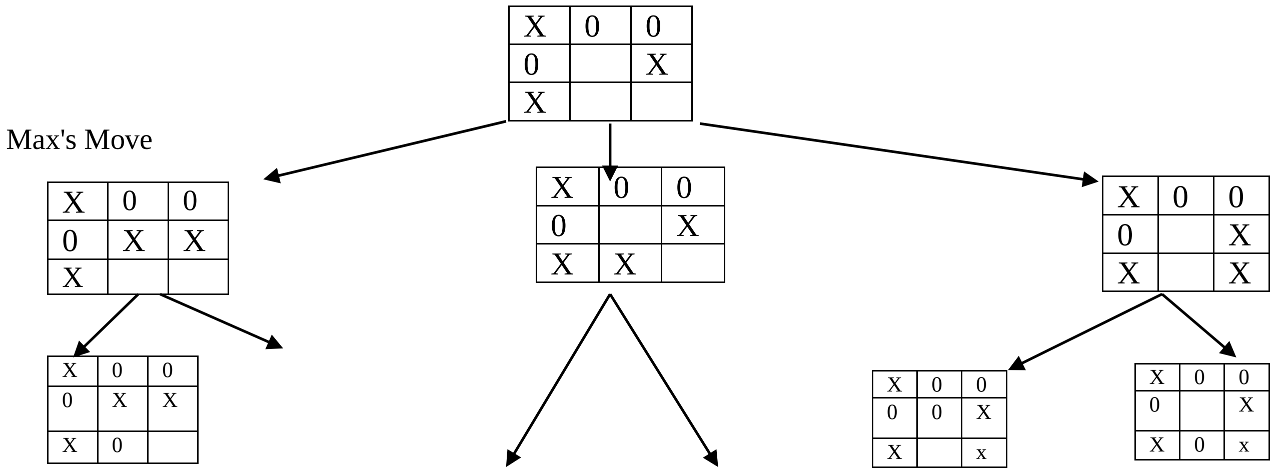
Consider the game with initial state as :

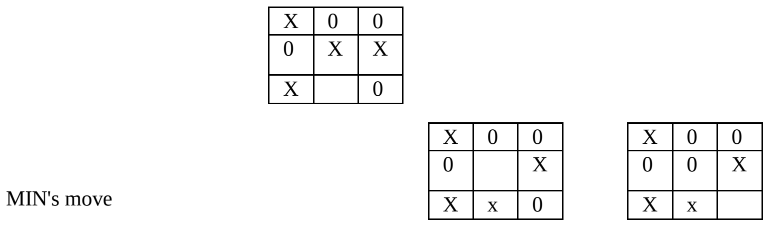
| | | |
|---|---|---|
| X | 0 | 0 |
| 0 | | X |
| X | | |

Step 1 MAX player moves by making X

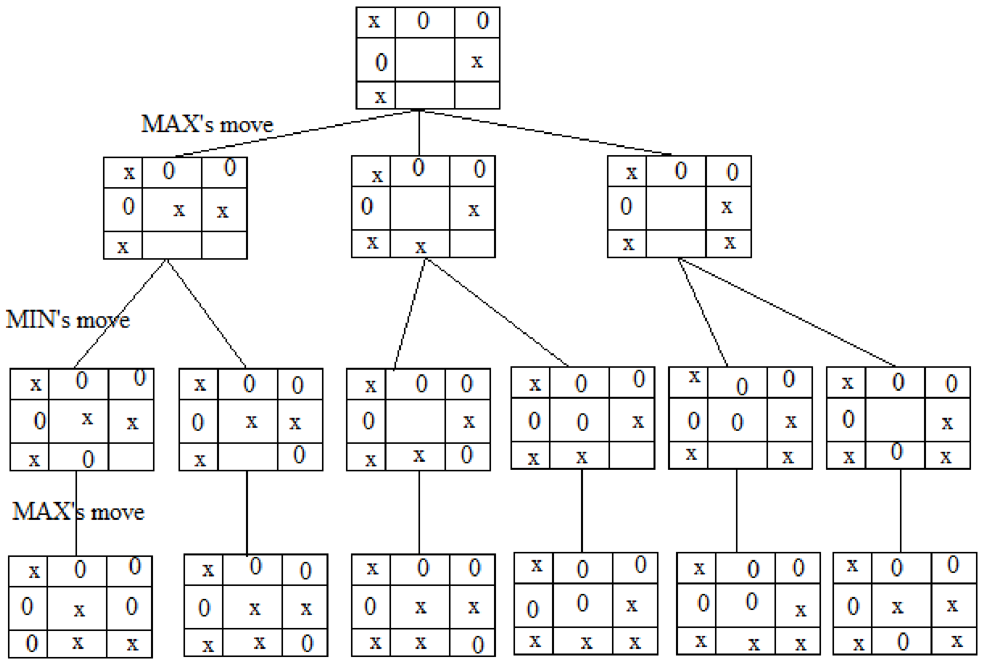


Step 2 MIN player moves by making 0





Step3 Again MAX's moves



## Evaluation Function

An evaluation function, also known as a heuristic evaluation function or static evaluation function, is a function used by game-playing programs to estimate the value or goodness of a position in the minimax and related algorithms.

## Utilitarian

It is one particular form of consequentialism, in which the "good consequence" is considered to be the one that maximizes happiness (or: welfare, benefit) for all people concerned. Consequentialism would look only at the consequences of an action to judge whether the action is right or wrong. A consequentialist would say that an action is morally right if its consequences lead to a situation that is clearly better than things were before the action.

## Decision Making

Decision is obviously related to reasoning. One of the possible definitions of artificial intelligence (AI) refers to cognitive processes and especially to reasoning. Before making any decision, people also reason, Artificial Intelligence is being used in decision support for tasks such as aiding the decision maker to select actions in real-time and stressful decision problems;

reducing information overload, enabling up-to-date information, and providing a dynamic response with intelligent agents; enabling communication required for collaborative decisions; and dealing with uncertainty in decision problems.

## Planning

Planning is a key ability for intelligent systems, increasing their autonomy and flexibility through the construction of sequences of actions to achieve their goals. Planning techniques have been applied in a variety of tasks including robotics, process planning, web-based information gathering, autonomous agents and spacecraft mission control. Planning involves the representation of actions and world models, reasoning about the effects of actions, and techniques for efficiently searching the space of possible plans.

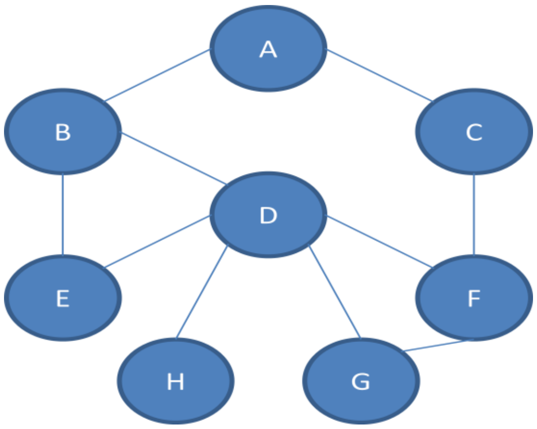
## Internal Representation:

Knowledge is the information about a domain that can be used to solve problems in that domain. To solve many problems requires much knowledge, and this knowledge must be represented in the computer. As part of designing a program to solve problems, we must define how the knowledge will be represented. A representation scheme is the form of the knowledge that is used in an agent. A representation of some piece of knowledge is the internal representation of the knowledge.

## Unit 2: Searching

## Searching

- Step in Problem Solving
- Searching is Performed through the State Space
- Searching accomplished by constructing a search tree



**Steps of Searching**

- Check whether the current state is the goal state or not
- Expand the current state to generate the new sets of states
- Choose one of the new states generated for search which entire depend on the selected search strategy
- Repeat the above steps until the goal state is reached or there are no more states to be expanded

**Searching: Types**

1) Blind Search or Uninformed Search

2) Informed Search or Heuristic Search

**1 ) Uninformed Search Techniques**

- Breadth First Search
- Uniform Cost Search

- Depth First Search
- Backtracking Search
- Depth Limited Search
- Iterative Deepening Depth First Search
- Bidirectional Search
- Search Strategy Comparison

**2) Informed Search or Heuristic Search**

- Best First Searching
    - Greedy Search
    - A* Search
- Local Search Algorithm & Optimization
- Iterative Improvement Algorithm
    - Hill Climbing Search
    - Simulated Annealing Search
    - Local Beam Search
    - Genetic Algorithm
- Adversarial Search Techniques
    - Mini-max Procedure
    - Alpha Beta Procedure

## Searching: Criteria to Measure

- **Completeness**: Ability to find the solution if the solution exists
- **Optimality**: Ability to find out the highest quality solution among the several solutions
    - Should maintain the information about the number of steps or the path cost from the current state to the goal state
- **Time Complexity**: Time taken to find out the solution

- **Space Complexity**: Amount of Memory required to perform the searching

**Searching: Evolution Function**

- A number to indicate how far we are from the goal
- Every move should reduce this number or if not never increase
- When this number becomes zero, the problem is solved (there may be some exceptions)

Eg:-

8 Puzzle Games

| 1 | 2 | 3 |
|---|---|---|
| 8 |  | 4 |
| 7 | 6 | 5 |

It's Goal State
Evolution Function = 0

| 2 | 8 | 3 |
|---|---|---|
| 1 | 4 |  |
| 7 | 6 | 5 |

It's Initial State
Evolution Function = -4

**Searching: Problem Classification**

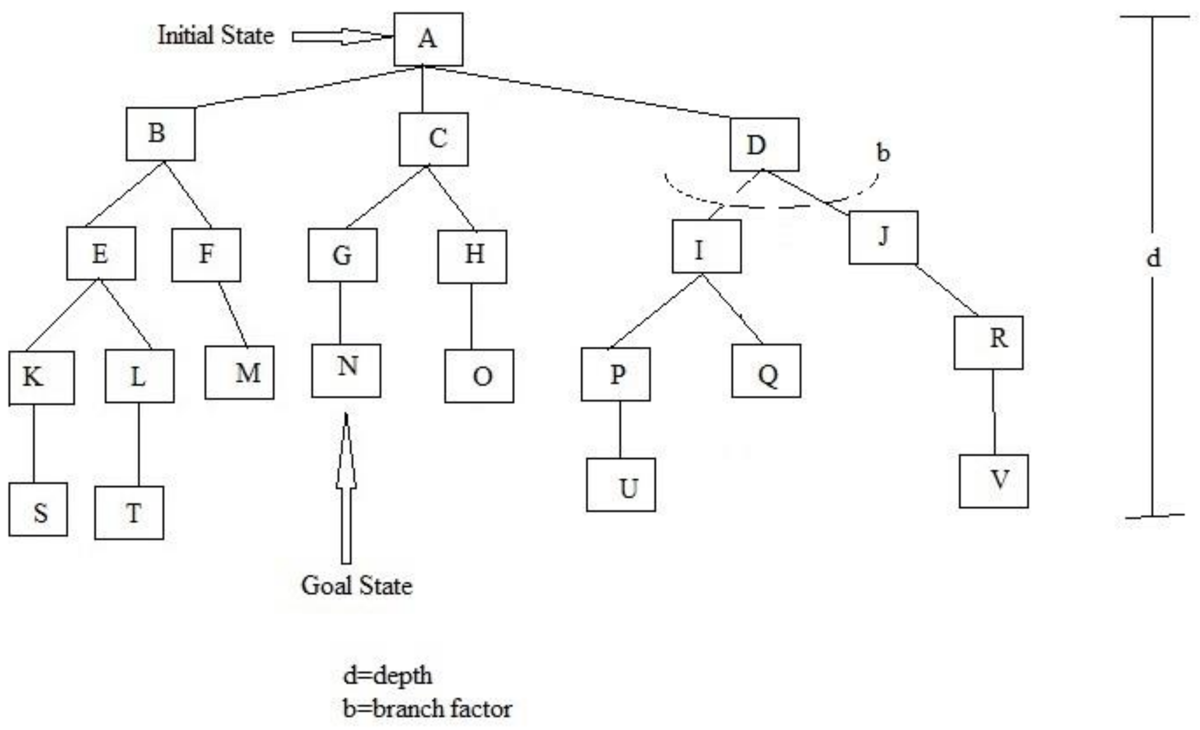
- **Ignorable**: Intermediate actions can be ignored. Example: Water Jug Problem
- **Recoverable**: The actions can be implemented to go the initial states. Example: 8 Puzzle Games
- **Irrecoverable**: The actions can't be implemented to reach the previous state. Example: Tic-Tac-Toe
- **Decomposable**: The problem can be broken into similar ones. Example: Bike Racing

## 1) Uninformed Search

- Search provided with problem definition only and no additional information about the state space
- Expansion of current state to new set of states is possible
- It can only distinguish between goal state and non-goal state
- Less effective compared to Informed search

## a ) Breadth First Search

- Root node is expanded first
- Then all the successors of the root node are expanded
- Then their successors are expanded and so on.
- Nodes, which are visited first will be expanded first (FIFO)
- All the nodes of depth 'd' are expanded before expanding any node of depth 'd+1'



| S.N | Next | Successors | Fringe(FIFO) | Visited |
|---|---|---|---|---|
| 1 | A (initial) | B,C,D | A | - |
| 2 | B | E,F | B,C,D | A |
| 3 | C | G,H | C,D,E,F | A,B |
| 4 | D | I,J | D,E,F,G,H | A,B,C |
| 5 | E | K,L | E,F,G,H,I,J | A,B,C,D |
| 6 | F | M | F,G,H,I,J,K,L | A,B,C,D,E |
| 7 | G | N | G,H,I,J,K,L,M | A,B,C,D,E,F |
| 8 | H | O | H,I,J,K,L,M,N | A,B,C,D,E,F,G |
| 9 | I | P,Q | I,J,K,L,M,N,O | A,B,C,D,E,F,G,H |
| 10 | J | R | J,K,L,M,N,O,P,Q | A,B,C,D,E,F,G,H,I |
| 11 | K | S | K,L,M,N,O,P,Q,R | A,B,C,D,E,F,G,H,I,J |

| | | | | |
|---|---|---|---|---|
| 12 | L | T | L,M,N,O,P,Q,R,S | A,B,C,D,E,F,G,H,I,J,K |
| 13 | M | - | M,N,O,P,Q,R,S,T | A,B,C,D,E,F,G,H,I,J,K,L |
| 14 | N (Goal) | - | N,O,P,Q,R,S,T | A,B,C,D,E,F,G,H,I,J,K,L,M |

## Breadth First Search: Four Criteria

### a ) Completeness

- d: depth of the shallowest goal
- b: branch factor
- This search strategy finds the shallowest goal first
- Complete, if the shallowest goal is at some finite depth

### b ) Optimality

- If the shallowest goal nodes were available, it would already have been reached
- Optimal, if the path cost is a non-decreasing function of the path of the node

### c) Time Complexity

- For a search tree a branching factor 'b' expanding the root yields 'b' nodes at the first level.
- Expanding 'b' nodes at first level yields $b^2$ nodes at the second level.
- Similarly, expanding the nodes at $(d+1)^{th}$ level yields $b^d$ node at $d^{th}$ level
- If the goal is in $d^{th}$ level, in the worst case, the goal node would be the last node in the $d^{th}$ level.
- Hence, We should expand $(b^d-1)$ nodes in the $d^{th}$ level (Except the goal node itself which doesn't need to be expanded)
- So, Total number of nodes generated at $d^{th}$ level = $b(b^d-1)$
  $=b^{d+1}-b$
- Again, Total number of nodes generated = $1+b+b^2+\ldots+b^{d+1}-b$
  $=O(b^{d+1})=O(b^d)$
- Hence, time complexity is $O(b^{d+1})$
  where, b= branching factor and
  d= level of goal node in the search table

### d) Space Complexity

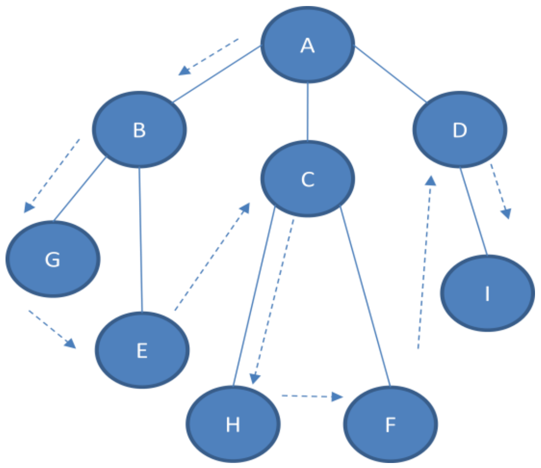
- Same as time complexity
- i.e. O($b^{d+1}$)
- Since each node has to be kept in the memory

**Disadvantages**

- Memory Wastage
- Irrelevant Operations
- Time Intensive
- It doesn't assure the optimal cost solution

## b) Depth First Search

- Expands the nodes at the deepest level of the search tree (LIFO)
- When a dead end is reached, the search backup to the next node that still has unexplored successors.



Consider the example figure of BFS

| S.N | Next | Successors | Fringe(LIFO) | Visited |
|---|---|---|---|---|
| 1 | A (initial) | B,C,D | A | - |
| 2 | B | EF | B,C,D | A |
| 3 | E | KL | E,F,C,D | A,B |
| 4 | K | S | K,L,F,C,D | A,B,E |
| 5 | S | - | S,L,F,C,D | A,B,E,K |
| 6 | L | T | L,F,C,D | A,B,E,K,S |
| 7 | T | - | T,F,C,D | A,B,E,K,S,L |
| 8 | F | M | F,C,D | A,B,E,K,S,L,T |

|  |  |  |  |  |
|---|---|---|---|---|
| 9 | M | - | M,C,D | A,B,E,K,S,L,T,F |
| 10 | C | GH | C,D | A,B,E,K,S,L,T,F,M |
| 11 | G | N | G,H,D | A,B,E,K,S,L,T,F,M,C |
| 12 | N(goal) | - | N,H,D | A,B,E,K,S,L,T,F,M,C,G |

**Depth First Search: Four Criteria**

**a ) Completeness**

- Can get stuck going down the wrong path
- It will always continue downwards without backing up
- If the path chose get infinitely down, even when shallow solution exists
- Not complete

**b) Optimality**

- The strategy might return a solution path that is longer than the optimal solution, if it starts with an unlucky path
- Not optimal

**c) Space Complexity**

- How many nodes can be in the queue (worst case) ?
- At depth l < d (max) we have b-1 nodes. So we have (d-1) * (b-1) nodes
- At max. depth d we have b nodes.
- It needs to store a single path from root to a leaf node and the remaining unexpanded sibling nodes for each node in the path
- For a search tree of branching factor ‘b’ and maximum tree depth ‘m’, only the storage of $b_{m+1}$ node is required
- So, total = (d-1) * (b-1) + b = O(bd) = O(bm)

Hence,
Space Complexity= O(b.m+1) = O(bm)

**d) Time Complexity**

- $O(b^m)$, in the worst case, since in the worst case all the $b^m$ nodes of the search tree would be generated. Hence, Time Complexity= $O(b^m)$
- Assume (worst case) that there is 1 goal leaf at the RHS
- So DFS will expand all nodes:

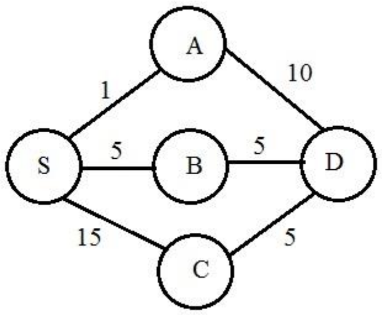
$=1 + b + b^2 + \ldots\ldots\ldots\ldots + b^d$

$=\mathbf{O}(b^d)$

$=\mathbf{O}(b^m)$

## c) Uniform-cost search (UCS)

- Uniform cost search is a search algorithm used to traverse, and find the shortest path in weighted trees and graphs.
- Uniform Cost Search or UCS begins at a root node and will continually expand nodes, taking the node with the smallest total cost from the root until it reaches the goal state.
- Uniform cost search doesn't care about how many steps a path has, only the total cost of the path.
- **UCS with all path costs equal to one is identical to breadth first search**.
- Similar to BFS except that it sorts (ascending order) the nodes in the fringe according to the cost of the node. where cost is the path cost.



| Fringe | Updated Fringe(Queue) | Next Node (Head of Fringe) | Successor (In sorted Ascending order) |
|---|---|---|---|
| $S_0$ | $A_1,B_5,C_{15}$ | S(Start) | C,B,A |
| $A_1$, $B_5$ , $C_{15}$ | $B_5,D_{11},C_{15}$ | A | D |
| $B_5$, $D_{11}$ ,$C_{15}$ | $D_{10},D_{11},C_{15}$ | B | D |
| $D_{10}$ ,$D_{11}$ ,$C_{15}$ | - | D(Goal) | - |

cost = 5+5=10. $S \rightarrow B \rightarrow D$

**Disadvantages**

- Doesn't care about the number of steps a path has but only about their cost
- It might get stuck in an infinite loop if it expands a node that has a zero cost action leading back to same state

**Uniform Cost Search: Four Criteria**

**a) Completeness**

- Complete, if the cost of every step is greater than or equal to some small positive constant E

**b) Optimality**

- The same ensures optimality

**c) Time Complexity**

- O(b $^{C^*/E}$)
- Where C*→ cost of optimal path
  and E→ small positive constant
- This complexity is much greater than that of Breadth first search

**d) Space Complexity**

- O(b $^{C^*/E}$)

## d) Backtracking Search

- It uses still less memory
- Only one successor is generated at a time rather than all
- Each partially expanded nodes remember which node to expand next

**Four Criteria**

- Completeness: Not Complete
- Optimality: Not Optimal
- Time Complexity= O($b^m$)
- Space Complexity= O(m)

## e) Depth Limited Search

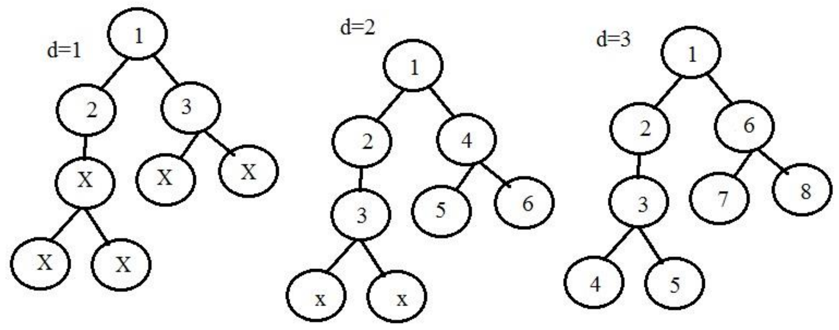
- Modification of depth first search
- Depth first search with predetermined limit ‘l’
- After the nodes at the level ‘l’ are explored, the search backtracks without going further deep i.e. the algorithm will not move down below.
- Hence, it solves the infinite path problem of the depth first search strategy.

**Four Criteria**

- Completeness: Complete except at additional source of incompleteness if l>d
- Optimality: Optimal except at l>d . i.e. the first path may be found to the goal instead of the shortest path.
- Time Complexity=O($b^l$)
- Space Complexity=O($b^l$)

## f) Iterative Deepening Depth First Search

- Finds the best limit by gradually increasing depth limit l first to 0, then to 1, 2 and so on.
- It is the derivative of Depth Limited Search.
- Combines the benefits of the depth first and breadth first search
- The complex part is to choose good depth limit
- This strategy addresses the issue of good depth limit by trying all possible depth limits
- The process is repeated until goal is found at depth limit 'd' which is the depth of shallowest goal.



Iterating increased depth searches with IDS

**Four Criteria**

- Completeness: as of Breadth First Search i.e. Complete if branching factor is finite
- Optimality: as of Breadth First Search i.e. optimal if the path cost is non decreasing function of depth
- Time Complexity= O($b^d$)
- Space Complexity= O($b^d$)

## g) Bidirectional Search

- Performs two simultaneous searches, one forward from initial state and the other backward from the last state
- Search stops when the two traversals meet in the middle

**Four Criteria**

- Completeness: Complete if both searches are B.F.S. and b is finite
- Optimality: Optimal if both searches are B.F.S.
- Time Complexity
    - For B.F.S. is $O(b^{d+1})$
    - If B.F. Bidirectional Search is used then the complexity = $O(b^{d/2})$
    - Since the forward and backward searches have to go halfway only
- Space Complexity= $O(b^{d/2})$

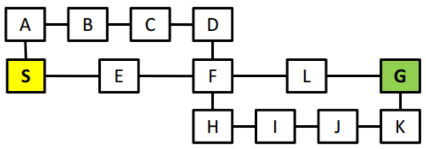
**Bidirectional Search (BFS)**

Forward:

– (<S>)→(<SA>,<SE>)→(<SE>,<SAB>)→**(<SAB>,<SEF>)**

Backward:

– (<G>)→(<GK>,<GL>)→(<GL>,<GKJ>)→**(<GKJ>,<GLF>)**



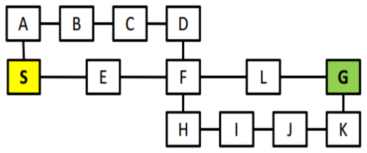
**Bidirectional Search (DFS)**

Forward:

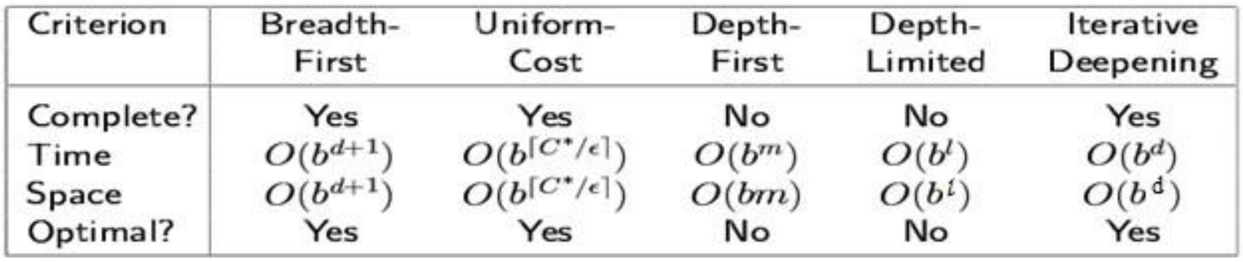
– (<S>) →(<SA>,<SE>) →(<SAB>,<SE>) →(<SABC>,<SE>) →(<SABCD>,<SE>) →**(<SABCDF>,<SE>)**

Backward:

– (<G>) →(<GK>,<GL>) →(<GKJ>,<GL>) →(<GKJI>,<GL>) →(<GKJIH>,<GL>) →**(<GKJIHF>,<GL>)**



## Summary of algorithms



| Criterion | Breadth-First | Uniform-Cost | Depth-First | Depth-Limited | Iterative Deepening |
|---|---|---|---|---|---|
| Complete? | Yes | Yes | No | No | Yes |
| Time | $O(b^{d+1})$ | $O(b^{\lceil C^*/\epsilon \rceil})$ | $O(b^m)$ | $O(b^l)$ | $O(b^d)$ |
| Space | $O(b^{d+1})$ | $O(b^{\lceil C^*/\epsilon \rceil})$ | $O(bm)$ | $O(b^l)$ | $O(b^d)$ |
| Optimal? | Yes | Yes | No | No | Yes |

## 2. Informed Search

- Strategy of problem solving where problem specific knowledge is known along with problem definition
- These search find solutions more efficiently by the use of heuristics
- Heuristic is a search technique that improves the efficiency of the search process
- By eliminating the unpromising states and their descendants from consideration, heuristic algorithms can find acceptable solutions.
- Heuristics are fallible i.e. they are likely to make mistakes as well
- It is the approach following an informed guess of next step to be taken
- It is often based on experience or intuition
- Heuristic have limited information and hence can lead to suboptimal solution or even fail to find any solution at all

### a) Best First Search

- A node is selected for expansion based on evaluation function f(n)

- A node with lowest evaluation function is expanded first
- The measure i.e. evaluation function must incorporate some estimate of the cost of the path from a state to the closest goal state
- The algorithm may have different evaluation function, one of such important function is the heuristic function h(n)
  where, h(n) is the estimated cost of the cheapest path from node n to the goal

**Best First Search: Types**

- Greedy Best First Search
- A* Search

## a.1 ) Greedy Best First Search

- The node whose state is judged to be the closest to the goal state is expanded first
- At each step it tries to be as close to the goal as it can
- It evaluates the nodes by using heuristic function
  hence, f(n)=h(n)
  where, h(n)=0, for the goal node

This search resembles depth first search in the way that it prefers to follow a single path all the way to the goal or if not found till the dead end and returns back up.

**Four Criteria**

**a ) Completeness**

- Can start down an infinite path and never return to any possibilities
- Not complete

**b) Optimality**

- Looks for immediate best choice and doesn't make careful analysis of long term options
- May give longer solution even if shorter solution exists
- Not optimal

**c) Space Complexity**

- O($b^m$) where, m is the maximum depth of search space, since all nodes have to be kept in memory

**d) Time Complexity**

$O(b^m)$

## a.2) A* Search

- Evaluates node by combining g(n), the cost to reach the node and h(n) the cost to get from node to goal
  f(n)=g(n)+h(n)
  where f(n) is the estimated cost of the cheapest solution through node n
- To find the cheapest solution, the first try node is the mode with lowest g(n)+h(n)
- Admissible Heuristic: h(n) is admissible if it never overestimates cost to reach the solution
  example: $h_{SLD}$ (straight line distance) as g(n) is exact, so f(n) is never overestimated

➢ When h(n) = actual cost to goal.

- Only nodes in the correct path are expanded.

-Optimal solution is found.

➢ When h(n) < actual cost to goal

-Additional nodes are expanded

-Optimal solution is found

➢ When h(n) > actual cost to goal
-Optimal solution can be overlooked.

**a) Optimality**

- Optimal if h(n) is admissible

**b) Completeness**

- Complete if h(n) is admissible

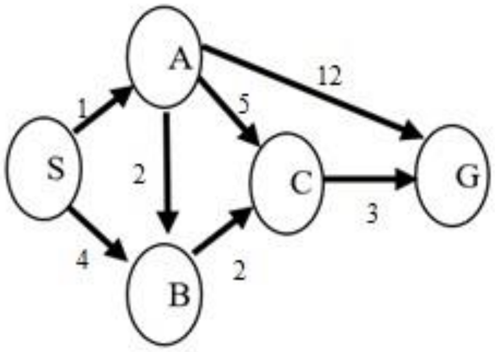
**c) Space Complexity**

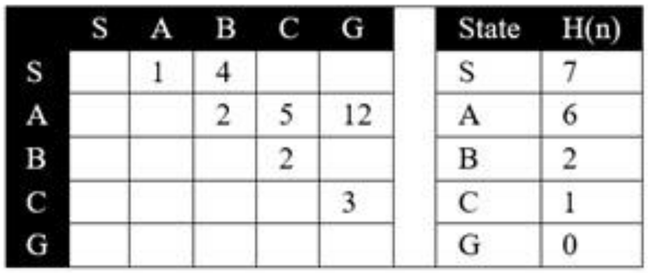
- $O(b^d)$ if h(n) is admissible

**d) Time Complexity**

- $O(b^d)$ if h(n) is admissible

**Eg**





| | S | A | B | C | G |
|---|---|---|---|---|---|
| S | | 1 | 4 | | |
| A | | | 2 | 5 | 12 |
| B | | | | 2 | |
| C | | | | | 3 |
| G | | | | | |

| State | H(n) |
|---|---|
| S | 7 |
| A | 6 |
| B | 2 |
| C | 1 |
| G | 0 |

H(n) is the number paths to reach goal from that state

**b) Local Search Algorithm and Optimization**

- Optimization
    - Aim to find the best state according to an objective function
- Local Search Algorithm
    - It operates using a single current state rather than multiple path and generally move only to neighbour of that state

**c) Iterative Improvement Algorithm**

- Consider that all the states are laid down on the surface of a landscape
- The height of a point on a landscape corresponds to process to move around the landscape trying to find the highest peaks, which are the optimal solutions
- Algorithm is suitable for problems where the path of the goal is irrelevant and only final configuration matters

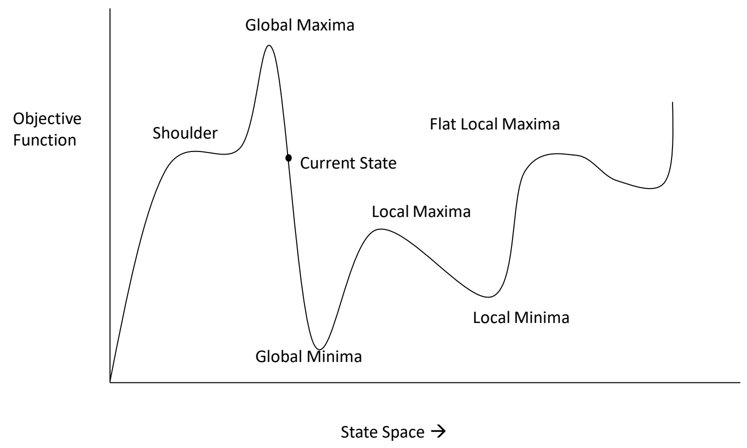


Fig.: A One Dimensional State Space Landscape [2]

**<u>Iterative Improvement Algorithm: Types</u>**

- Hill Climbing Search
- Simulated Annealing Search
- Local Beam Search
- Genetic Algorithm

## c. 1) Hill Climbing Search

- Hill climbing is an optimization technique for solving computationally hard problems.
- Used in problems with "the property that the state description itself contains all the information"
- The algorithm is memory efficient since it does not maintain a search tree
- Hill climbing attempts to iteratively improve the current state by means of an evaluation function
- Searching for a goal state = Climbing to the top of a hill
- Moves continuously in the direction of increasing value (uphill)
- Doesn't maintain a search tree so the current node data structure needs only record the state and its objective function value

- Hill climbing doesn’t look ahead beyond the immediate neighbours of the current state
- Also called greedy local search sometimes because it grabs a good neighbour state without thinking about where to go next
- it often makes very rapid progress towards the solution because it is usually quite easy to improve a bad state
- One move is selected and all other nodes are rejected and are never considered
- Halts if there is no successor

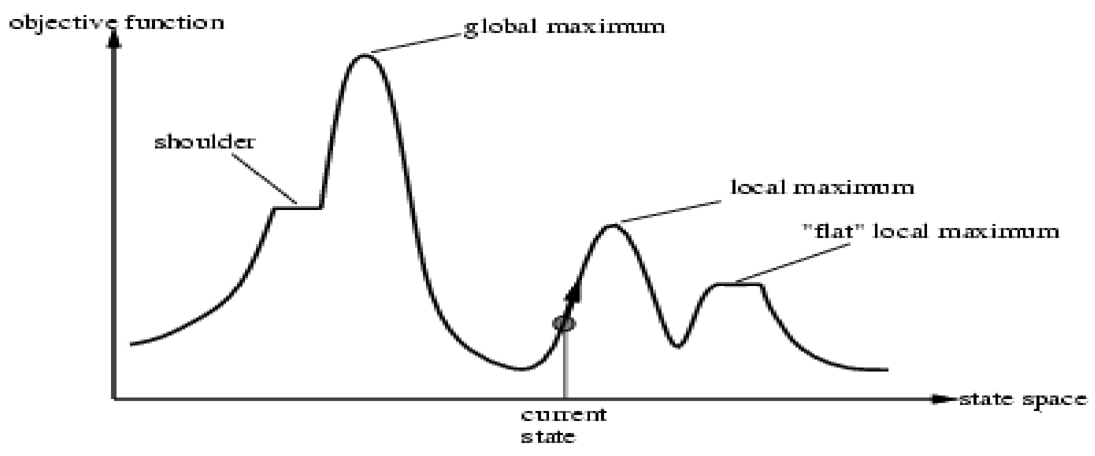
**Hill Climbing Search: Problems**

- Local Maxima
    - Peak that is higher than each of its neighbouring states but lower than the global maxima
    - Hill climbing halts whenever a local maxima is reached
- Plateau
    - An area of the state space landscape where the evaluation function is flat
    - Can be flat local maxima where no uphill exists or shoulder from which it is possible to progress
    - A hill climbing search might be unable to find its way off the plateau
- Ridges
    - A special kind of local maxima which is the result of a sequence of local maxima that is very difficult for greedy algorithms to navigate
    - It is an area of search space that is higher than the surrounding areas and that itself is at a slope

**Algorithm**

1. Determine successors of current state
2. Choose successor of maximum goodness
3. If goodness of best successor is less than current state's goodness, stop
4. Otherwise make best successor the current state and go to step 1

**Problem**: depending on initial state, can get stuck in local maxima



### Hill Climbing Search: Solution to the Problems

- Local Maxima
    - Backtrack to some earlier node and try going to different direction
- Plateau
    - Make a big jump in some direction to try to get a new section of the search space
    - If rule apply single small steps, apply them several times in the same direction
- Ridges
    - Apply two or more rules such as bidirectional search before doing the test
    - Moving in several directions at once

## c.2) Simulated Annealing Search

- Rather than starting from a different initial state all over again, when a current state shows no progress in the technique
- This search takes some downhill steps so that it can escape that particular local maxima and continue with other peaks in the state space
- A random pick is made for the move
    - If it improves the situation, it is accepted straight away
    - If it worsen the situation, it is accepted with some probability less than 1 which decreases exponentially with the badness of the move i.e. for bad moves the probability is low and for comparatively less bad one, it's higher
- The degree of badness of the move is determined by the amount $\Delta E$, by which the evaluation is worsened
- The probability also depends on the value of a objective function parameter 'T'

- For low value of T, probability is high and vice versa
- Hence, bad moves are more likely to be allowed at the beginning only
- This method is more common in VLSI layout problem solving, factory scheduling and travelling salesman problems

## c.3) Local Beam Search

- A path-based algorithm
- Keeps track of k-states rather than just one
- Begins with k randomly generated states, at each step, all the successors of all k states are generated
- If anyone is the goal, the algorithm halts
- Can quickly become concentrated in a small region of the state space

## c.4) Genetic algorithms

- A successor state is generated by combining two parent states
- Start with *k* randomly generated states (population)
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s)
- Evaluation function (fitness function). Higher values for better states.
- Produce the next generation of states by selection, crossover, and mutation

## d) Adversarial Search Techniques

- Often known as Games or Game Playing
- Used in competitive multi-agent environments
- Based on game theory
- Deterministic and fully observable environments in which there are two agents whose actions must alternate and in which the utility values at the end of the game are always equal and opposite
- This creates adversarial situation

### Optimal Decision in Adversarial Search

- A game can be defined as a kind of search problem with the following components:

- Initial State identifying the initial position in the game and identification of the first player
- Successor Function returning a list of (move, state) pairs
- Terminal Test which determine that the game is over
- Utility function which gives a numeric value for the terminal states.

## d.1) Minimax Algorithm

- Max is considered as the first player in the game and Min as the second player
- This algorithm computes the minimax decision from the current state
- It uses a recursive computation of minimax values of each successor state directly implementing some defined function
- The recursion proceeds from the initial node to all the leaf nodes
- Then the minimax values are backed up through the tree as the recursion unwinds
- It performs the depth first exploration of a game tree in a complete way

## d.2 ) Alpha Beta Pruning

- Minimax algorithm has to examine exponentially increasing number of moves
- As the exponential rise can't be avoided Pruning cut it into halves
- By not considering a large part of the tree number of states to be calculated is cut down
- When applied to a standard minimax tree, alpha beta pruning returns the same move as minimax would, but prunes away the branches which couldn't possibly influence the final decision
- Alpha beta pruning could be applied to the trees of any depth

# Unit 3: Pattern Recognition

**Introduction**

Pattern recognition refers to the task of placing some objects to correct class based on the measurements about the object. Pattern recognition is the study of how machines can

- observe the environment,
- learn to distinguish patterns of interest,
- make sound and reasonable about the categories of the patterns.

## Classification Problems:

Classification is a task that occurs very frequently in everyday life. Essentially it involves dividing up objects so that each is assigned to one of number of mutually exhaustive and exclusive categories known as classes. The term mutually exhaustive and exclusive simply means that each object must assigned to precisely one class, i.e. never to more than one and never to any class at all. So, organization of data in a given class is called classification.

Many practical decision-making tasks can be formulated as classification problems, i.e. assigning people or objects to one of a number of categories.

For example:

- customers who likely to buy or not buy a particular product in a supermarket.,
- people who are at high , medium or low risk of acquiring a certain illness,
- objects on a radar display which correspond to vehicles, people, buildings or trees,
- the likelihood of rain the next day for a weather forecast (very likely, likely, unlikely, very unlikely).

Classification is learning a function that maps a data item into one of several predefined classes. The target function is also known informally as a classification model.

Data Classification is done in two steps:

- **First Step**
  - model of predefined set of data classes or concept is made
  - model is constructed by analyzing database tuples (i.e samples, examples or objects). described by attributes.
  - each tuple is assumed to belong to predefined class .
  - individual tuples making up the training set are referred to as training samples.
  - Since the class label of each training sample is provided , this step is also called supervised learning. (i.e. told to which class each training sample belongs).
- **Second Step**

- model is used for classification.
- First, predictive accuracy of the model (or classifier) is estimated.
- There are several methods for estimating classifier accuracy.
- Samples are randomly selected and are independent of the training samples.
- Accuracy of model is given by test set.
- the known class label is compared with the learned model's class prediction.
- if the accuracy is acceptable, the model can be used to classify future data tuples of objects for which the class label is known.( Such data are called "unknown" or "previously unseen" data).

## **Evaluating Classifier:**

An algorithm implements classification , especially in a concrete implementation, is known as a classifier. The term "classifier" sometimes also refers to mathematical function, implemented by a classification algorithm that maps input data to a category.

### a) Bayesian Classifiers:

- They are statistical classifiers.
- Can predict class membership probabilities for e.g. probability that a given tuple belongs to a particular class.
- Based on Baye's Theorem.

### b) Naive Bayesian Classifier

- Studies comparing classification algorithms found a simple Bayesian classifier known as Naive Bayesian Classifier.
- based on conditional probability.
- comparable in performance with decision tree and selected neural network classifiers.
- Naive Bayesian Classifiers assume that the effect of an attribute value on a given class is independent of the values of the other attributes. This assumption is called "class conditional independence".
- It is made to simplify the computations involved , so in this sense considered "naive".
- makes use of all attributes contained in data and analyze them individually as though they are equally important and independent to each other.

## **Bayes Theorem:**

X: data sample whose class label is unknown

H: some Hypothesis, such as that the data sample X belongs to a specified class C.

P(H | X): the probability that the hypothesis H holds given the observed data sample X. It is also called posterior probability or posteriori probability.

e.g. suppose the world of data samples consists of fruits, described by their color and shape. Suppose that X is red and round, and that H is the hypothesis that X is an apple. Then P(H | X) reflects our confidence that X is an apple given that we have seen that X is red and round. In contrast, P(H) is the prior probability, or a priori probability of H. Similarly, P(X) is the prior probability of X.

P(X |H): posterior probability of X conditioned on H. That is, it is the probability that X is red and round given that we know that it is true that X is an apple .

Bayes theorem is:

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)}$$

**c) k-Nearest Neighbor classifiers**

- Nearest neighbor classifiers are based on learning by analogy.
- The training samples are described by n-dimensional numeric attributes.
- Each sample represents a point in an n-dimensional space, i.e. all training samples are stored in a n-dimensional pattern space.
- When given an unknown sample, a k-nearest neighbor classifier searches the pattern space for the k training samples that are closest to the unknown sample.
- These k training samples are the k "nearest neighbors" of the unknown sample.
- "closeness" is defined in terms of Euclidean distance, where the Euclidean distance between two points, X=($x_1$, $x_2$, ......$x_n$) and Y=($y_1$,$y_2$, .....,$y_n$) is :

$$d(X, Y) = \sqrt{\sum_{i=1}^{k} (x_i - y_i)^2}$$

- Nearest neighbor classifier are instance-based or lazy learners in that they store all of the training samples and do not build a classifier until a new (unlabeled) sample needs to be classified. This contrasts with eager learning methods such as decision tree, induction and back propagation, which construct a generalization model before receiving new samples to classify
- Lazy learners can incur expensive computational costs when the number of potential neighbor (i.e., stored training samples) with which to compare a given unlabeled sample is great. Therefore, they require efficient indexing techniques.
- Nearest neighbor classifiers can also be used for prediction i.e. to return a real-valued prediction for a given unknown sample.

**Training, Testing, Validation**

- **Training Dataset**:
    - ➢ a set of examples used for learning
    - ➢ The sample of data used to fit the model.
- **Validation Dataset**:
    - ➢ The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyper parameters.
    - ➢ The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration.
- **Test Dataset:**
    - ➢ The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.
    - ➢ a set of examples used only to assess the performance of a fully-trained classifier

## <u>**Over fitting and Complexity**</u>

Overfitting refers to a model that models the training data too well.

Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the models ability to generalize.

Overfitting is more likely with nonparametric and nonlinear models that have more flexibility when learning a target function. As such, many nonparametric machine learning algorithms also include parameters or techniques to limit and constrain how much detail the model learns.

For example, decision trees are a nonparametric machine learning algorithm that is very flexible and is subject to overfitting training data. This problem can be addressed by pruning a tree after it has learned in order to remove some of the detail it has picked up.

- Complexity (running time) increases with dimension d
- A lot of methods have at least O(nd$^2$) complexity, where n is the number of samples
- For example if we need to estimate covariance matrix .So as d becomes large, O(nd$^2$) complexity may be too costly.

- If d is large, n, the number of samples, may be too small for accurate parameter estimation. For example, covariance matrix has d$^2$ parameters:
- For accurate estimation, n should be much bigger than d$^2$

- Otherwise model is too complicated for the data,

# Intelligent System

## Unit 4: Neural Network

Er. Ranjan Raj Aryal

aryalranjan@gmail.com

## Introduction

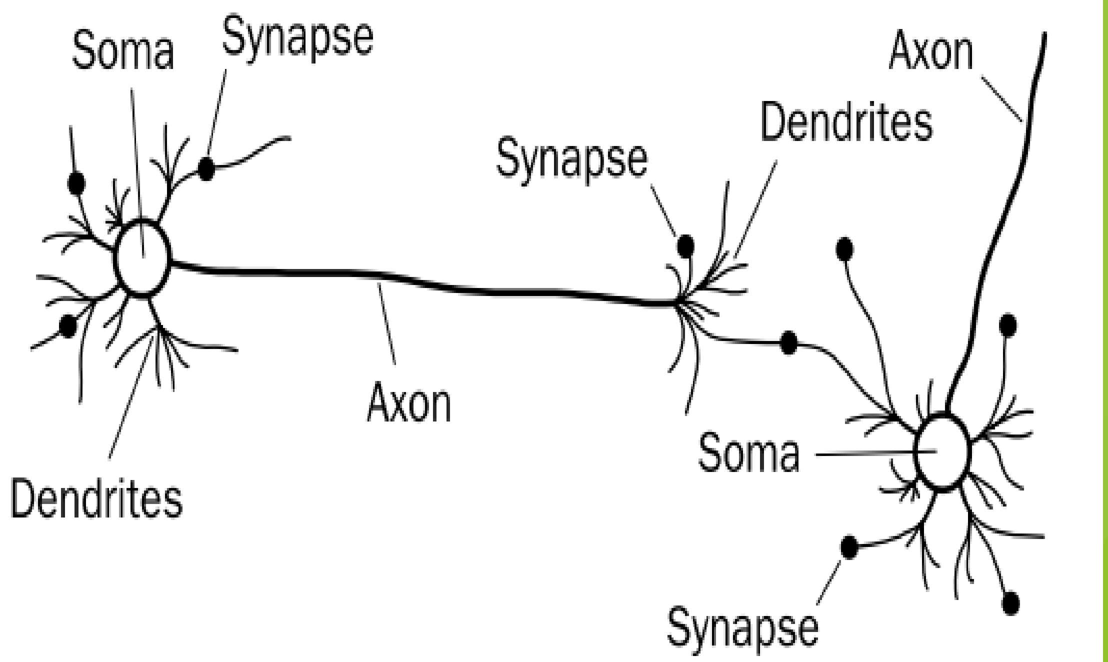
- Machine Learning involves adaptive mechanism that enable computer to learn from experience, example and analogy
- Learning capability improves performance of an Artificial System over time
- Machine learning mechanisms form the basis for adaptive systems
- The most popular approaches to machine learning  are Artificial Neural Network (ANN) and Genetic Algorithms(GA)
- Two types of Machine Learning :
    - Supervised Learning and
    - Unsupervised learning

## Biological Neural Networks

- A neural network can be defined as a model of reasoning based on the human
  brain. The brain consists of a densely interconnected set of nerve cells, or basic
  information-processing units, called **neurons**
- The human brain incorporates nearly 10 billion neurons and 60 trillion connections, **synapses**, between them. By using multiple neurons simultaneously, the brain can perform its functions much faster than the fastest computers in existence today.

## Biological Neural Networks

- ❑ Each neuron has a very simple structure, but an army of such elements constitutes a tremendous processing power
- ❑ **Neuron**: fundamental functional unit of all nervous system tissue
- ▶ **Soma**: cell body, contain nucleus
- ▶ **Dendrites**: a number of fibres, input
- ▶ **Axon**: single long fibre with many branches, output
- ▶ **Synapse**: junction of dendrites and axon, each neuron form synapse with 10 to 100000 other neurons



# Biological Neural Network

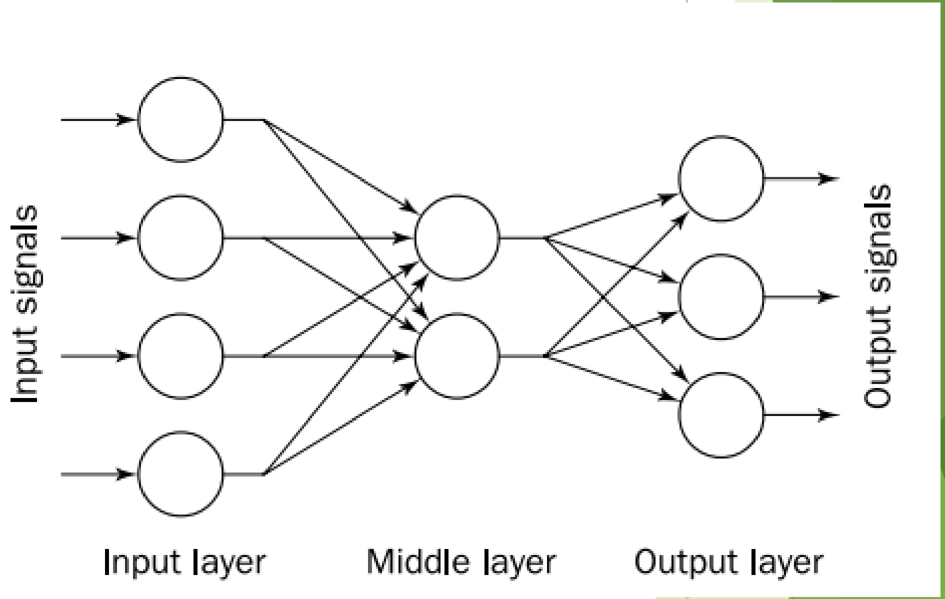
**HOW DOES BRAIN WORKS**

Signals are propagated from neuron to neuron by electrochemical reaction:

1. Chemical substances are released from the synapses and enter the dendrite, raising or lowering the electrical potential of the cell body
2. When a potential reaches a threshold, an electrical pulse or **action potential** is sent down the axon
3. The pulse spreads out along the branches of the axon, eventually reaching synapses and releasing transmitters into the bodies of other cells
    1. **Excitory** Synapses: Increase potential
    2. **Inhibitory** Synapses : Decreases potential

## Artificial Neural Network (ANN)

- Consists of a number of very simple and highly interconnected processors called neurons
- The neurons are connected by weighted links passing signals from one neuron to another.
- The output signal is transmitted through the neuron’s outgoing connection. The outgoing connection splits into a number of branches that transmit the same signal. The outgoing  ranches terminate at the incoming connections of other neurons in the network



## Analogy Between Biological and ANN

| Biological Neural Network | Artificial Neural Network |
|---|---|
| Soma | Neuron |
| Dendrite | Input |
| Axon | Output |
| Synapse | Weight |

## Learning in ANN

- Supervised learning
  - Uses a set of inputs for which the desired outputs are known
  - Example: Back propagation algorithm
- Unsupervised learning
  - Uses a set of inputs for which no desired output are known.
  - The system is self-organizing; that is, it organizes itself internally. A human must examine the final categories to assign meaning and determine the usefulness of the results.
  - Example: Self-organizing map

# The Neuron as a simple computing element: Diagram of a neuron

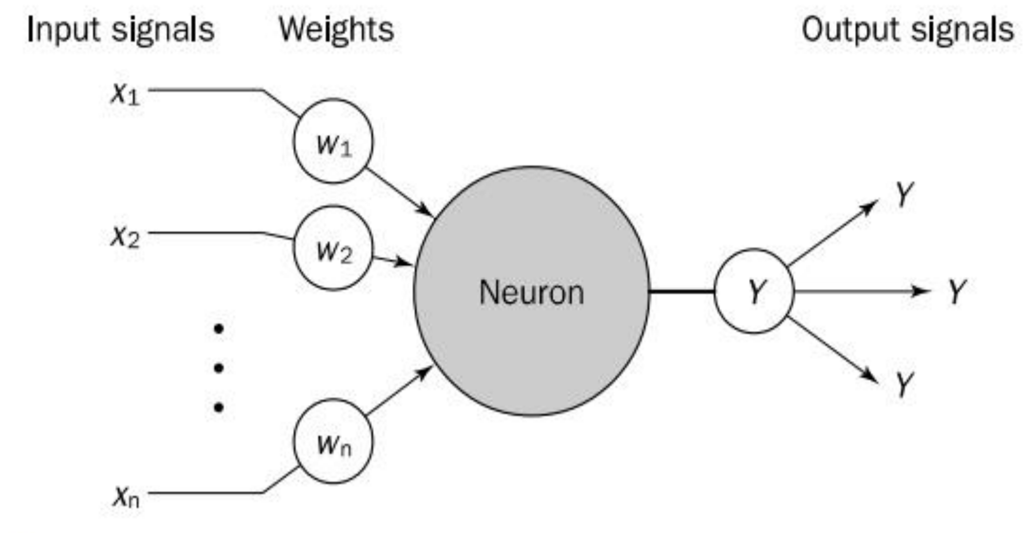
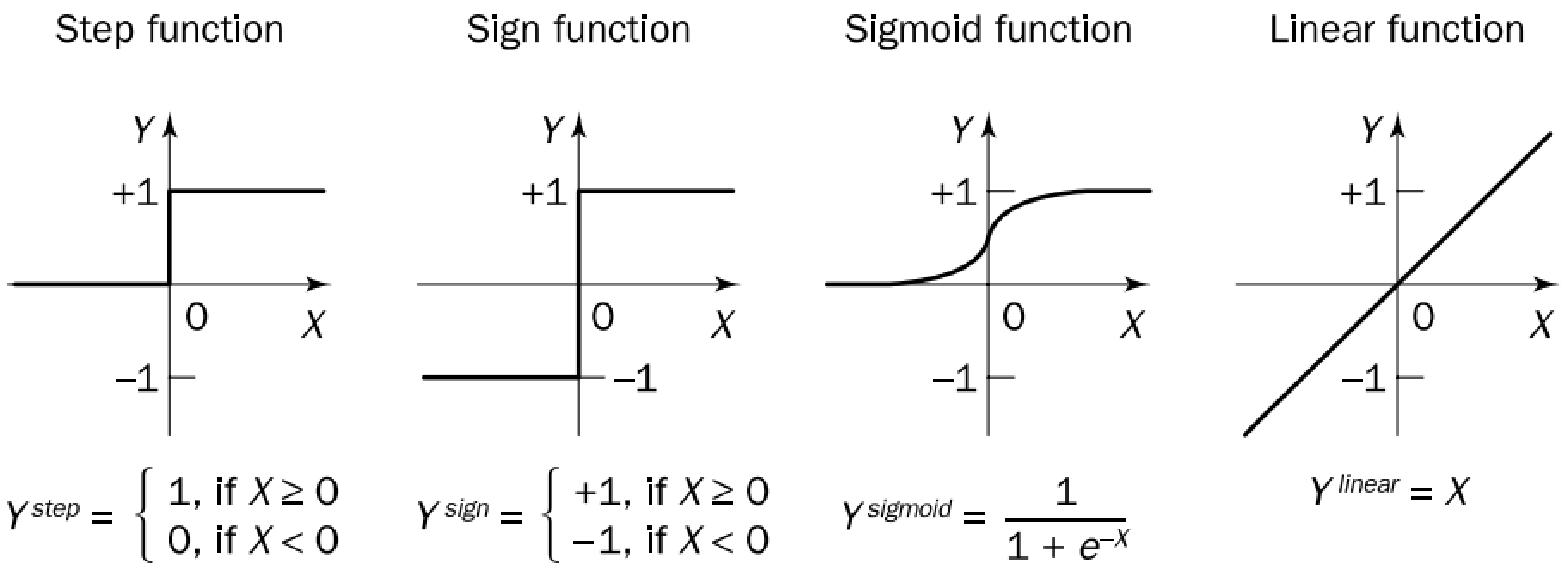


Diagram of a neuron

- The neuron computes the weighted sum of the input signals and compares the result with a **threshold value**, θ. If the net input is less than the threshold, the neuron output is –1. But if the net input is greater than or equal to the threshold, the neuron becomes activated and its output attains a value +1.
- The neuron uses the following transfer or **activation function**
- $X = \sum_{i=1}^{n} x_i \, w_i \qquad Y = \begin{Bmatrix} +1 \; if \; X \geq \theta \\ -1 \; if \; X \; \leq \theta \end{Bmatrix}$
- This type of activation function is called a **sign function**

## Activation Functions for Neuron



$$Y^{step} = \begin{cases} 1, \text{ if } X \geq 0 \\ 0, \text{ if } X < 0 \end{cases}$$

$$Y^{sign} = \begin{cases} +1, \text{ if } X \geq 0 \\ -1, \text{ if } X < 0 \end{cases}$$

$$Y^{sigmoid} = \frac{1}{1 + e^{-X}}$$

$$Y^{linear} = X$$

## Perceptron

- In 1958, Frank Rosenblatt introduced a training algorithm that provided the first procedure for training a simple ANN : a perceptron
- The perceptron is the simplest form of a neural network. It consists of a single neuron with *adjustable* synaptic weights and a **hard limiter**
- The operation of Rosenblatt's perceptron is based on the McCulloch and Pitts neuron model. The model consists of a **linear combiner** followed by a hard limiter
- The weighted sum of the inputs is applied to the hard limiter, which produces an output equal to +1 if its input is positive and -1 if its input is negative

## Perceptron (cont....)

- Perceptron is one of the earliest learning systems.
- It can be regarded as the triangle classifier.
- Mark I perceptron was capable of making binary decisions.
- The idea was to start from a binary image (in the form of Pixels; zeros or ones)
- A number of associator units receive inputs from different sets of image pixels and produce binary outputs in the following levels.
- The perceptron uses an error-correction learning algorithm in which the weights are modified after erroneous decisions as follows:
  - a) If the decision is correct, the weights are not adjusted
  - b) If the erroneous decision is the r*th* one in the list of possible decisions and the correct decisions is the s*th one*, some values aj are deducted from the weights wrj and added to the weight wsj (j=1,2,…,n).

## Perceptron

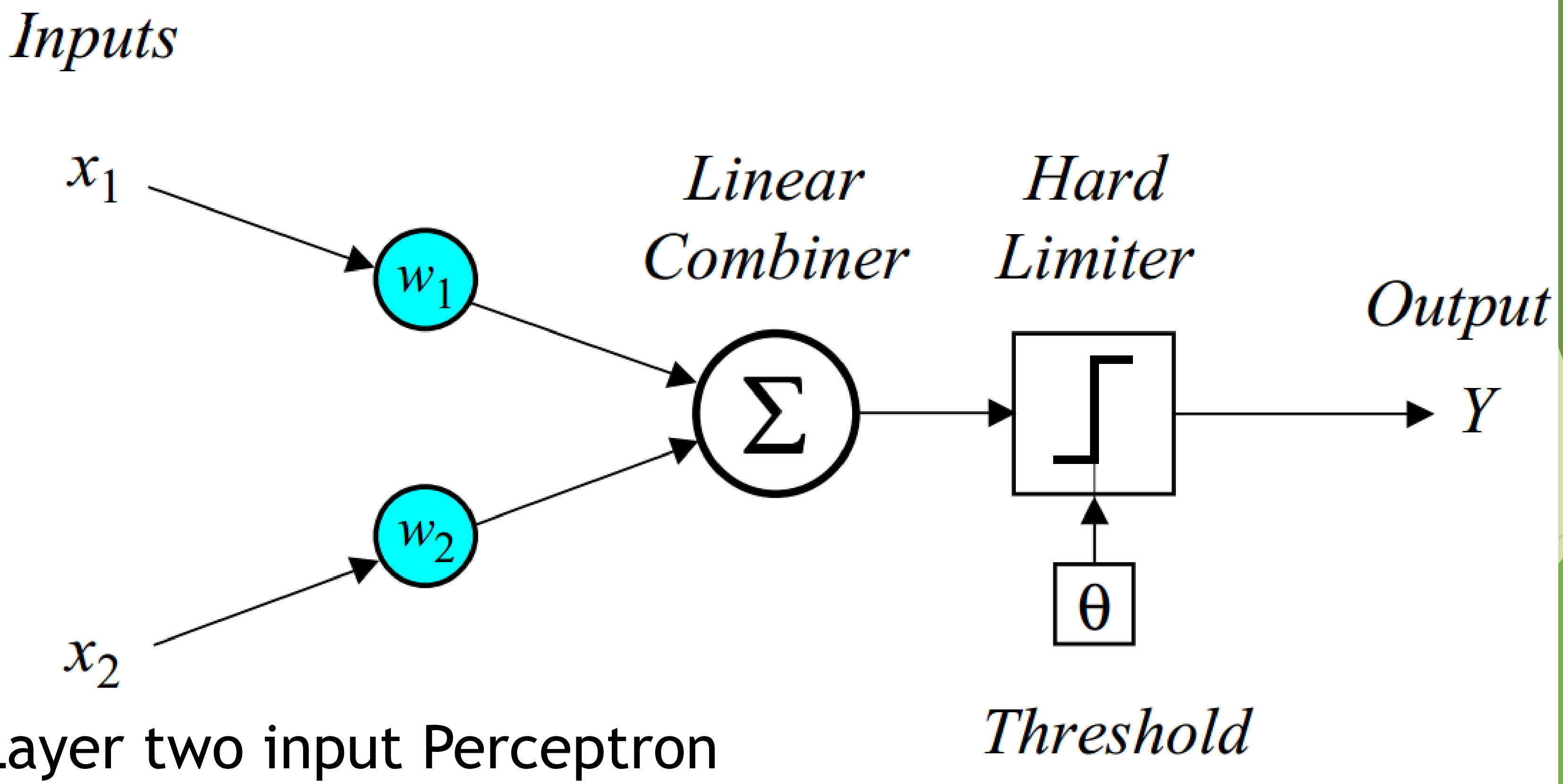


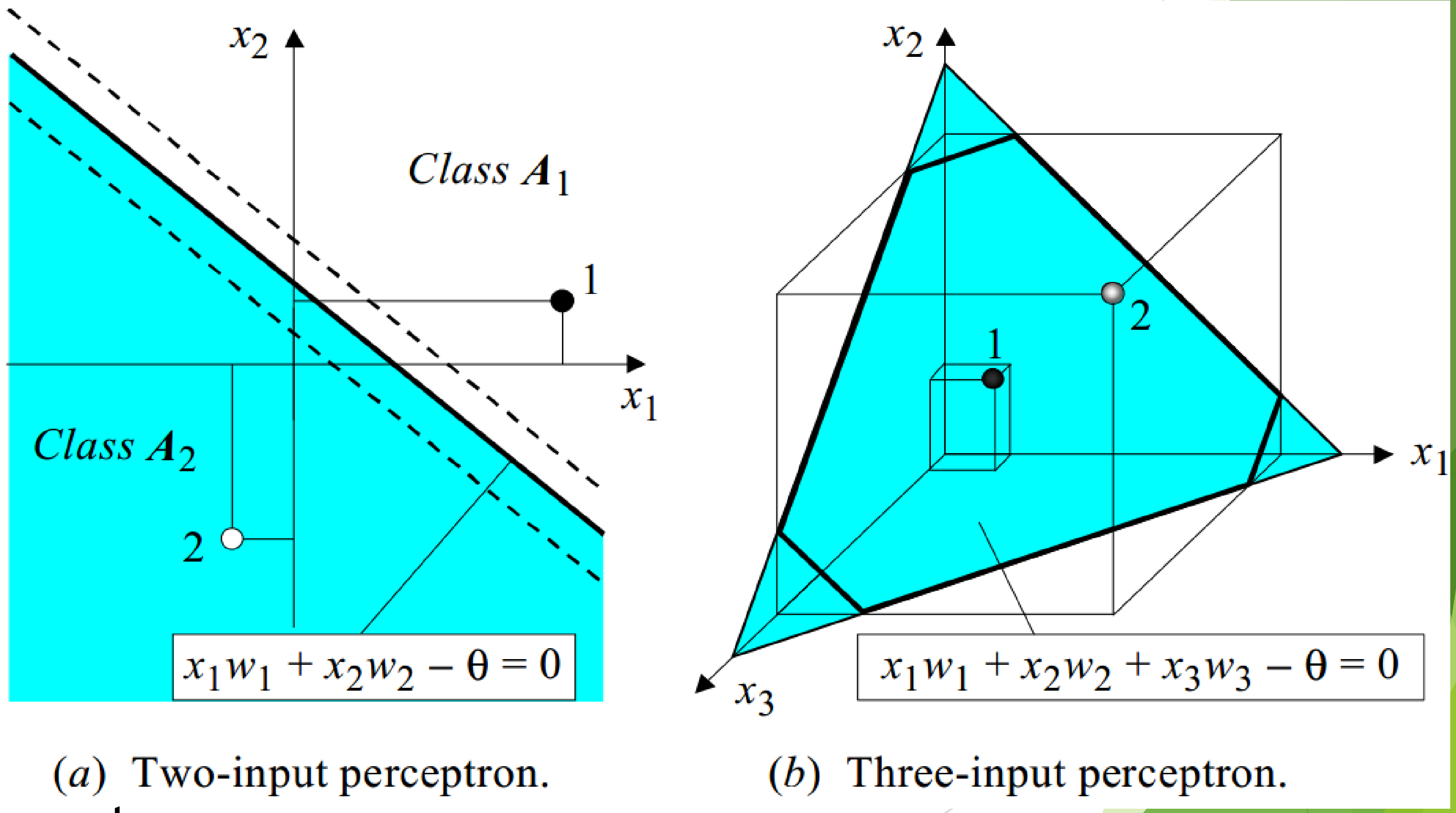
Fig: Single Layer two input Perceptron

## Perceptron

- The aim of the perceptron is to classify inputs, $x_1, x_2, x_3, x_4, \ldots, x_n$. Into one of two classes, say $A_1$, $A_2$.
- In the case of an elementary perceptron, the n-dimensional space is divided into a *hyperplane* into two decision regions. The hyperplane is defined by ***linearly separable function***

$$\sum_{i=1}^{n}(x_i w_i - \theta) = 0$$

## Perceptron



$x_2$
$x_1$
Class $A_1$
Class $A_2$
1
2
$x_1 w_1 + x_2 w_2 - \theta = 0$
(a) Two-input perceptron.
$x_2$
$x_1$
$x_3$
1
2
$x_1 w_1 + x_2 w_2 + x_3 w_3 - \theta = 0$
(b) Three-input perceptron.

Fig: Linear Separability in the perceptron

## Perceptron

- **How does the perceptron learn its classification tasks?**
- This is done by making small adjustment in the weights to reduce the difference between the actual and desired outputs of the perceptron. The initial weights are randomly assigned, usually in the range [-0.5, +0.5], and then updated to obtain the output consistent with the training examples.
- If at iteration p, the actual output is , $Y_{(p)}$ *and* the desired output is , $Y_{d(p)}$, then the error is given by :

$$e_p = Y_{d(p)} - Y_{(p)} \text{ , where } p = 1, 2, 3 , \ldots$$

  - Iteration *p* here refers to the *pth* training example presented to the perceptron
- If the error, $e_p$ is positive, we need to increase perceptron output $Y_{(p)}$, but if it is negative, we need to decrease $Y_{(p)}$

## Perceptron

- **Perceptron Learning Rule**

$$w_i(p+1) = w_i(p) + \alpha \times x_i(p) \times e(p)$$

where $p = 1, 2, 3, \ldots$

- α is the **learning rate** , a positive constant less than unity
- The perceptron learning rule was first proposed by Rosenblatt in 1960. Using this rule we can derive perceptron training algorithm for classification task

## Perceptron training algorithm

- **Step 1 : Initialization**
  - Set initial weights $w_1$, $w_2$, …, $w_n$ and threshold θ to random numbers in the range [-0.5, +0.5].

  - If error $e_p$ is positive, we need to increase perceptron output $Y_p$ , but if it is negative, we need to decrease $Y_p$

## Perceptron training algorithm (cont.…)

- **Step 2 : Activation**

Activate the perceptron by applying inputs $x_1(p), x_2(p), \ldots, x_n(p)$ and desired output $Y_d(p)$

Calculate t

$$Y(p) = step\left[\sum_{i=1}^{n} x_i(p) w_i(p) - \theta\right]$$

Where *n* is the number of the perceptron inputs, and *step* is activation function

## Perceptron training algorithm (cont.…)

- **Step 3: Weight Training**
- Update the weights of the perceptron

$$w_i(p+1) = w_i(p) + \Delta w_i(p)$$

- Where $\Delta w_i(p)$ is the weight correction at iteration *p* . The weight correction is computed by the delta rule:

$$\Delta w_i(p) = \alpha \times x_i(p) \times e(p)$$

- **Step 4: Iteration**
- Increase iteration *p* by 1, go back to *Step 2* and repeat the process until convergence

## Perceptron training algorithm (pseudocode)

```
inputs: examples, a set of examples, each with input  x = $x_1$, ………., $x_n$ and output y   network,
        a perceptron with weights $W_j$, j = 0…….n , and activation function g
repeat
    for each e in examples do
        $in \leftarrow \sum_{j=0}^{n} W_j x_j[e]$
        $Err \leftarrow y[e] - g(in)$
        $W_j \leftarrow W_j + \alpha * Err * g`(in) * x_j[e]$
until some stopping criterion is satisfied
```

## Multilayer Neural Network

- A multi layer perceptron is a feed forward neural network with one or more hidden layers
- The network consists of :
  - Input Layer
  - Hidden Layer
  - Output Layer
- The input signal is propagated in a forward direction in a layer-by-layer basis

# Multilayer Neural Network

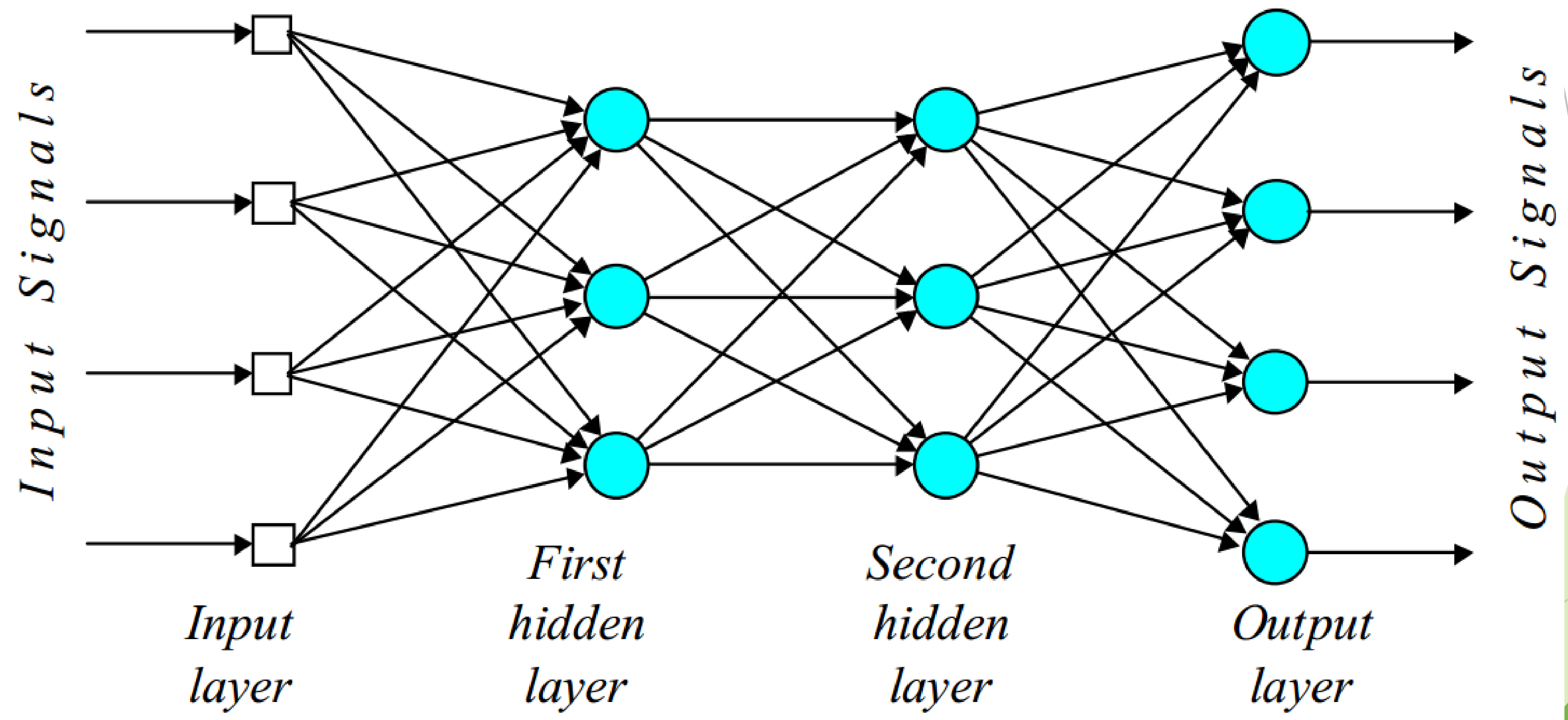


Fig: A multilayer Perceptron with two hidden layers

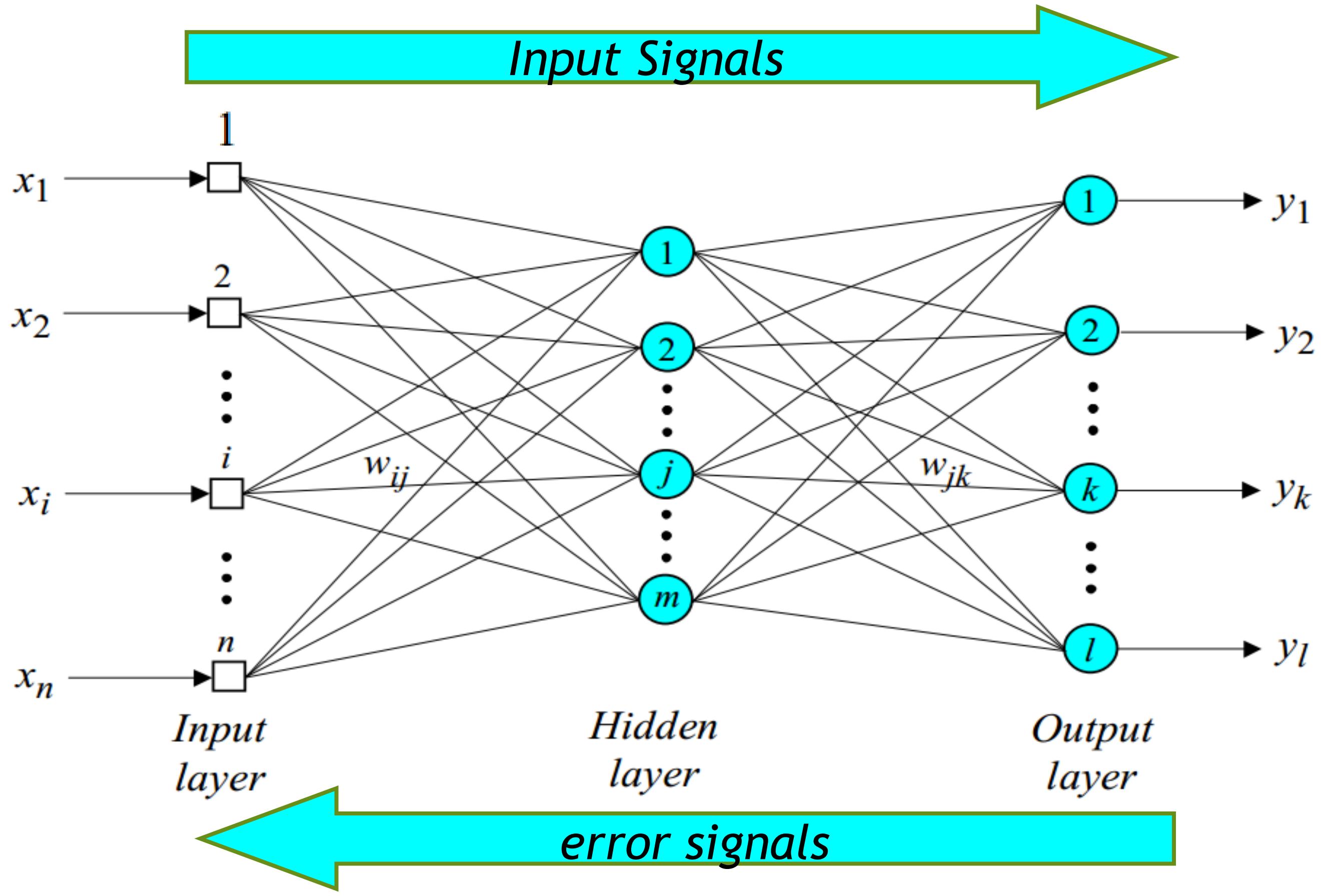
## Multilayer Neural Network

- The hidden layer “hides” its desired output. Neurons in the hidden layer can not be observed through the input/output behavior of the network. There is no obvious way to know what the desired output of the hidden layer should be.
- Commercial ANNs incorporate three and sometimes four layers, including one or two hidden layers. Each layer can contain from 10 to 1000 neurons. Experimental neural networks may have five or six layers, including three or four hidden layers, and utilize millions of neurons.

## Back Propagation

- Learning in a multilayer network proceeds the same way as for a perceptron
- A training set of input patterns is presented to the network
- The network computes its output pattern, and if there is an error –or other word difference between actual and desired output pattern – the weight are adjusted to reduce the error
- In a back-propagation neural network, the learning algorithm has two phases
- First, a training input pattern is presented to the network input layer. The network propagates the input pattern from layer to layer until the output pattern is generated by the out layer
- If this pattern is different from the desired output, an error is calculated and then propagated backwards through the network from the output layer to the input layer. The weights are modified as the error is propagated

## Back Propagation



## Backpropagation algorithm

- 1. Build a network with the chosen number of inputs, hidden and output units.
- 2.Initialize all the weights to low random values
- 3. Choose a single training pair at random.
- 4. Copy the input pattern to the input layer.
- 5. Cycle the network so that the activations from the inputs generate the activations in the hidden and output layers.
- 6. Calculate the error derivative between the output activation and the target output.
- 7. Backpropagate the summed products of the weights and errors in the output layer in order to calculate the error in the hidden units.
- 8. Update the weights attached to each unit according to errors in that unit, the output from the unit below it and the learning parameters until the error is sufficiently low or the network setteles.

## Back Propagation Training Algorithm

- **Step 1: Initialization**
- Set all the weights and threshold levels of the network to random numbers uniformly distributed inside a small range:

$$\left(-\frac{2.4}{F_i}, +\frac{2.4}{F_i}\right)$$

- where $F_i$ is the total number of inputs of neuron $i$ in the network. The weight initialization is done on a neuron-by-neuron basis.

# Back Propagation Training Algorithm

- **Step 2: Activation**
- Activate the back-propagation neural network by applying inputs $x_1(p)$, $x_2(p)$, …, $x_n(p)$ and desired outputs $y_{d,1}(p)$, $y_{d,2}(p)$,…, $y_{d,n}(p)$
- A) Calculate the actual output of the neurons in the hidden layers:

$$\boxed{y_j(p) = sigmoid\left[\sum_{i=1}^{n} x_i(p) \cdot w_{ij}(p) - \theta_j\right]}$$

- Where *n* is the number of inputs of neuron *j* in the hidden layer, and *sigmoid* is the *sigmoid* activation function

## Back Propagation Training Algorithm

- **Step 2: Activation(contd…)**
- b) Calculate the actual outputs of the neurons in the output layer:

$$\boxed{y_k(p) = sigmoid\left[\sum_{j=1}^{m} x_{jk}(p) \cdot w_{jk}(p) - \theta_k\right]}$$

- Where *m* is the number of inputs of neuron *k* in the output layer.

## Back Propagation Training Algorithm

- **Step 3: weight Training**
- Update the weights in the back-propagation network propagating backward the errors associated with output neurons.
- (a) Calculate the error gradient for the neurons in the output layer:

$$\boxed{\delta_k(p) = y_k(p) \cdot [1 - y_k(p)] \cdot e_k(p)}$$

where $\boxed{e_k(p) = y_{d,k}(p) - y_k(p)}$

Calculate the weight corrections:

$$\boxed{\Delta w_{jk}(p) = \alpha \cdot y_j(p) \cdot \delta_k(p)}$$

Update the weights at the output neurons:

$$\boxed{w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p)}$$

## Back Propagation Training Algorithm

- **Step 3: weight Training(cont….)**
- (b) Calculate the error gradient for the neurons in the hidden layer:

$$\boxed{\delta_j(p) = y_j(p) \cdot [1 - y_j(p)] \cdot \sum_{k=1}^{l} \delta_k(p)\, w_{jk}(p)}$$

Calculate the weight corrections:

$$\boxed{\Delta w_{ij}(p) = \alpha \cdot x_i(p) \cdot \delta_j(p)}$$

Update the weights at the hidden neurons:

$$\boxed{w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)}$$

## Back Propagation Training Algorithm

- **Step 3: Iteration**
- Increase iteration *p* by one, go back to *Step* 2 and repeat the process until the selected error criterion is satisfied.
- As an example, we may consider the three layer back-propagation network. Suppose that the network is required to perform logical operation Exclusive-OR. Recall that a single-layer perceptron could not do this operation. Now we will apply the three layer net.

## Back Propagation Training Algorithm

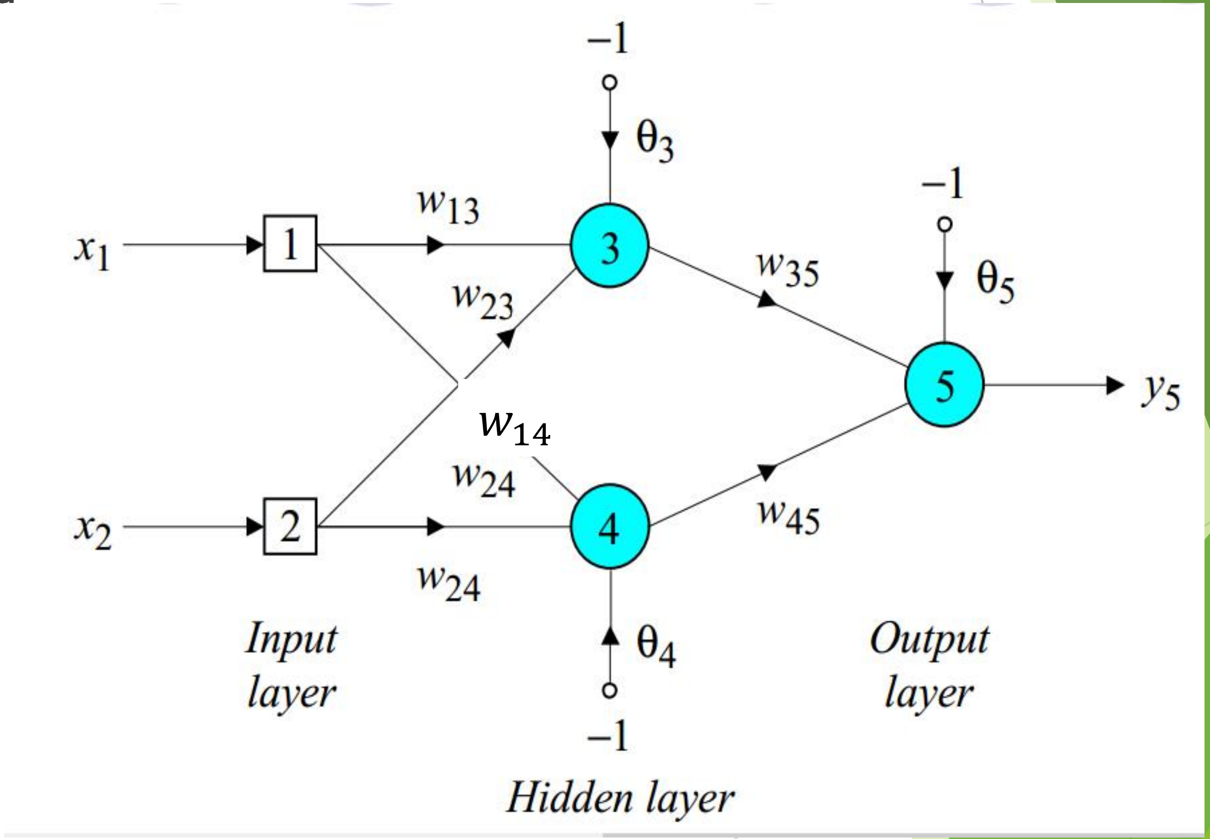
**inputs:** *examples*, a set of examples, each with input vector **x** and output vector **y** *network*, a multilayer network with L layers, weights $W_{j,i}$, activation function g

```
repeat
    for each e in examples do
        for each node j in the input layer do $a_j \leftarrow x_j[e]$
        for $l$ = 2 to M do
            $in_i \leftarrow \sum_j W_{j,i} a_j$
            $a_i \leftarrow g(in_i)$
        for each node $i$ in the output layer do
            $\Delta_i \leftarrow g`(in_i) * (y_i[e] - a_i)$
        for $l$ = M -1 to 1 do
            for each node $j$ in layer $l$ do
                $\Delta_j \leftarrow g`(in_j) \sum_i W_{j,i} \Delta_i$
                for each node $i$ in layer $l$+1 do
                    $W_{j,i} \leftarrow W_{j,i} + \alpha * a_j * \Delta_i$
until some stopping criterion is satisfied
```

# Example: Three-layer network for solving the Exclusive-OR operation

- The effect of the threshold applied to a neuron in the hidden layer is represented by its weight, $\theta$, connected to a fixed input equal to -1
- The initial weights and threshold levels are set randomly as follows:

$w_{13} = 0.5, w_{14} = 0.9, w_{23} = 0.4, w_{24} = 1.0, w_{35} = -1.2,$
$w_{45} = 1.1, \theta_3 = 0.8, \theta_4 = -0.1$ and $\theta_5 = 0.3.$



## Example: Three-layer network for solving the Exclusive-OR operation

- We consider a training set where inputs $x_1$ ,and $x_2$ are equal to 1 and desired output $y_{d,5}$ is 0. The actual output of neurons 3 and 4 in the hidden layers are calculated as:

$$\boxed{\begin{aligned} y_3 &= sigmoid\,(x_1 w_{13} + x_2 w_{23} - \theta_3) = 1/\left[1 + e^{-(1\cdot 0.5 + 1\cdot 0.4 - 1\cdot 0.8)}\right] = 0.5250 \\ y_4 &= sigmoid\,(x_1 w_{14} + x_2 w_{24} - \theta_4) = 1/\left[1 + e^{-(1\cdot 0.9 + 1\cdot 1.0 + 1\cdot 0.1)}\right] = 0.8808 \end{aligned}}$$

- Now the actual output of neuron 5 in the output layer is determined as:

$$\boxed{y_5 = sigmoid(y_3 w_{35} + y_4 w_{45} - \theta_5) = 1/\left[1 + e^{-(-0.52501.2 + 0.88081.1 - 1\cdot 0.3)}\right] = 0.5097}$$

Thus, the following error is obtained:

$$\boxed{e = y_{d,5} - y_5 = 0 - 0.5097 = -0.5097}$$

## Example: Three-layer network for solving the Exclusive-OR operation

- The next step is weight training. To update the weights and threshold levels in our network, we propagate the error, $e$ , from the output layer backward to the input layer.
- First, we calculate the error gradient for neuron 5 in the output layer

$$\delta_5 = y_5 (1 - y_5) e = 0.5097 \cdot (1 - 0.5097) \cdot (-0.5097) = -0.1274$$

- Then we determine the weight corrections assuming that the learning rate parameter, $\alpha$, is equal to 0.1

$$\Delta w_{35} = \alpha \cdot y_3 \cdot \delta_5 = 0.1 \cdot 0.5250 \cdot (-0.1274) = -0.0067$$
$$\Delta w_{45} = \alpha \cdot y_4 \cdot \delta_5 = 0.1 \cdot 0.8808 \cdot (-0.1274) = -0.0112$$
$$\Delta \theta_5 = \alpha \cdot (-1) \cdot \delta_5 = 0.1 \cdot (-1) \cdot (-0.1274) = -0.0127$$

# Example: Three-layer network for solving the Exclusive-OR operation

- Now we calculate the error gradients for neuron 3 and 4 in the hidden layer:

$$\delta_3 = y_3(1-y_3)\cdot\delta_5\cdot w_{35} = 0.5250\cdot(1-0.5250)\cdot(-0.1274)\cdot(-1.2) = 0.0381$$

$$\delta_4 = y_4(1-y_4)\cdot\delta_5\cdot w_{45} = 0.8808\cdot(1-0.8808)\cdot(-0.1274)\cdot 1.1 = -0.0147$$

- We, then, determine the weight corrections:

$$\Delta w_{13} = \alpha\cdot x_1\cdot\delta_3 = 0.1\cdot 1\cdot 0.0381 = 0.0038$$
$$\Delta w_{23} = \alpha\cdot x_2\cdot\delta_3 = 0.1\cdot 1\cdot 0.0381 = 0.0038$$
$$\Delta\theta_3 = \alpha\cdot(-1)\cdot\delta_3 = 0.1\cdot(-1)\cdot 0.0381 = -0.0038$$
$$\Delta w_{14} = \alpha\cdot x_1\cdot\delta_4 = 0.1\cdot 1\cdot(-0.0147) = -0.0015$$
$$\Delta w_{24} = \alpha\cdot x_2\cdot\delta_4 = 0.1\cdot 1\cdot(-0.0147) = -0.0015$$
$$\Delta\theta_4 = \alpha\cdot(-1)\cdot\delta_4 = 0.1\cdot(-1)\cdot(-0.0147) = 0.0015$$

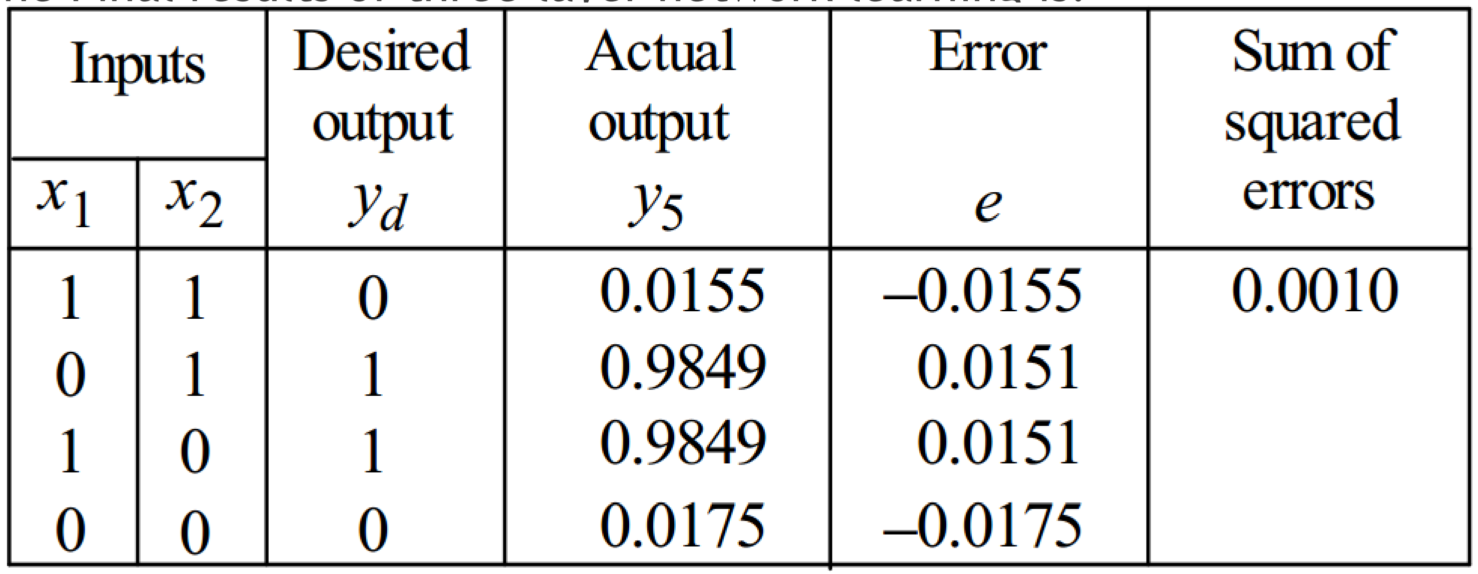
## Example: Three-layer network for solving the Exclusive-OR operation

- At last, we update all weights and thresholds
- The training process is updated till the sum of squared error is less than 0.001

$$
\begin{aligned}
w_{13} &= w_{13} + \Delta w_{13} = 0.5 + 0.0038 = 0.5038 \\
w_{14} &= w_{14} + \Delta w_{14} = 0.9 - 0.0015 = 0.8985 \\
w_{23} &= w_{23} + \Delta w_{23} = 0.4 + 0.0038 = 0.4038 \\
w_{24} &= w_{24} + \Delta w_{24} = 1.0 - 0.0015 = 0.9985 \\
w_{35} &= w_{35} + \Delta w_{35} = -1.2 - 0.0067 = -1.2067 \\
w_{45} &= w_{45} + \Delta w_{45} = 1.1 - 0.0112 = 1.0888 \\
\theta_3 &= \theta_3 + \Delta\theta_3 = 0.8 - 0.0038 = 0.7962 \\
\theta_4 &= \theta_4 + \Delta\theta_4 = -0.1 + 0.0015 = -0.0985 \\
\theta_5 &= \theta_5 + \Delta\theta_5 = 0.3 + 0.0127 = 0.3127
\end{aligned}
$$

## Example: Three-layer network for solving the Exclusive-OR operation

- The Final results of three layer network learning is:



| Inputs | | Desired output | Actual output | Error | Sum of squared errors |
|---|---|---|---|---|---|
| $x_1$ | $x_2$ | $y_d$ | $y_5$ | $e$ | |
| 1 | 1 | 0 | 0.0155 | –0.0155 | 0.0010 |
| 0 | 1 | 1 | 0.9849 | 0.0151 | |
| 1 | 0 | 1 | 0.9849 | 0.0151 | |
| 0 | 0 | 0 | 0.0175 | –0.0175 | |

# Gradient Descent

- Gradient descent is an iterative minimisation method. The gradient of the error function always
  shows in the direction of the steepest ascent of the error function.
- It is determined as the derivative of the activation function multiplied by the error at the neuron output

For Neuron *k* in the output layer

$$\delta_k(p) = \frac{\partial y_k(p)}{\partial X_k(p)} \times e_k(p)$$

Where, yk(p) is the output of neuron k at iteration p, and Xk(p) is the net weighted input of neuron k at same iteration.

## Characteristics of ANN

- Adaptive learning
- Self-organization
- Error tolerance
- Real-time operation
- Parallel information processing

## Benefits and Limitations of ANN

| Benefits | Limitations |
|---|---|
| Ability to tackle new kind of problems | Performs less well at tasks humans tend to find difficult |
| Robustness | Lack of explanation facilities |
| | Require large amount of test data |

## Thank You!!!

# Unit 5: Probabilistic Methods

## Introduction to Probabilistic Reasoning

- One of the most common characteristics of the human information available is its imperfection due to partial observability, non-deterministic or combination of both
- An agent may not know what state it is in or will be after certain sequence of actions
- Agent can cope with these defects and make rational judgments and rational decisions to handle such uncertainty and draw valid conclusions

**What is uncertainty?**

- The lack of the exact knowledge that would enable us to reach a perfectly reliable conclusion
- Classical Logic permits only exact reasoning i.e., perfect knowledge always exists
- In Real world such clear-cut knowledge could not be provided to systems

| | | |
|---|---|---|
| **IF A is true** | **and** | **IF B is true** |
| **THEN A is not false** | | **THEN B is not false** |

## Sources of Uncertain Knowledge

- **Weak Implication**: Domain experts and knowledge engineer have rather painful or hopeless task of establishing concrete correlation between IF(Condition) and THEN(action) part of rules. Vague Data.
- **Imprecise Language :** NLP is ambiguous and imprecise. We define facts in terms of **often, sometimes, frequently, hardly ever.** Such can affect IF-THEN implication
- **Unknown Data:** incomplete and missing data should be processes to an approx. reasoning with this values
- **Combining the views of different experts:** Large system uses data from many experts
- The basic Concept of probability plays significant role in our life like we try to determine the probability of rain, prospect of promotion, likely hood of winning in Black Jack
- The probability of an event is the proportion of cases in which the event occurs (Good, 1959)

- Probability, mathematically, is indexed between 0 and 1
- Most events have probability index strictly between 0 and 1, which means that each event has at lease two possible outcomes: favorable outcome or success and unfavorable outcomes or failure

$$\mathrm{P(success)} = \frac{The\ number\ of\ successes}{The\ number\ of\ possible\ outcomes}$$

$$\mathrm{P}(failure) = \frac{The\ number\ of\ failure}{The\ number\ of\ possible\ outcomes}$$

- If *s* is the number of success and *f* is the number of failures then:

$$\mathrm{P}(success) = \frac{s}{s+f}$$

$$\mathrm{P}(failure) = \frac{f}{s+f}$$

and

$$p + q = 1$$

- Let us consider classical examples with a coin and a dice. If we throw a coin, the probability of getting a head will be equal to the probability of getting a tail. In a single throw, s = f = 1, and therefore the probability of getting a head (or a tail) is 0.5.
- Consider now a dice and determine the probability of getting a 6 from a single throw. If we assume a 6 as the only success, then s = 1 and f = 5, since there is just one way of getting a 6, and there are five ways of not getting a 6 in a single throw. Therefore, the probability of getting a 6 is

$$\mathrm{P} = \frac{1}{1+5} = 0.1666$$

Likewise, the probability of not getting 6 is

$$\mathrm{q} = \frac{5}{1+5} = 0.8333$$

Above instances are for independent events i.e. mutually exclusive events which can not happen simultaneously

In the dice experiment, the two events of obtaining a 6 and, for example, a 1 are mutually exclusive because we cannot obtain a 6 and a 1 simultaneously in a single throw. However, events that are not independent may affect the likelihood of one or the other occurring. Consider, for instance, the probability of getting a 6 in a single throw, knowing this time that a 1 has not come up. There are still five ways of not getting a 6, but one of them can be eliminated as we know that a 1 has not been obtained. Thus,

$$\mathrm{p} = \frac{1}{1 + (5 - 1)}$$

- Let A and B be two not mutually exclusive events, but occur conditionally on the occurrence of other.
- The probability of event A will occur if event B occurs is called conditional Probability

$$\mathrm{p(A|B)} = \frac{\textit{the number of times A and B can occur}}{\textit{the number of times B can occur}}$$

The probability of both A and B will occur is called joint probability $(\boldsymbol{A} \cap \boldsymbol{B})$

$\mathrm{p(A|B)} = \frac{p(A \cap B)}{p(B)}$, the probability of A occurring given B has occurred

$\mathrm{p(B|A)} = \frac{p(B \cap A)}{p(A)}$, the probability of B occurring given A has occurred

- For *n* number of mutually exclusive event B we have

$$\begin{gathered} p(A \cap B_1) = p(A|B_1) \times p(B_1) \\ p(A \cap B_2) = p(A|B_2) \times p(B_2) \\ \vdots \\ p(A \cap B_n) = p(A|B_n) \times p(B_n) \end{gathered}$$

or when combined:

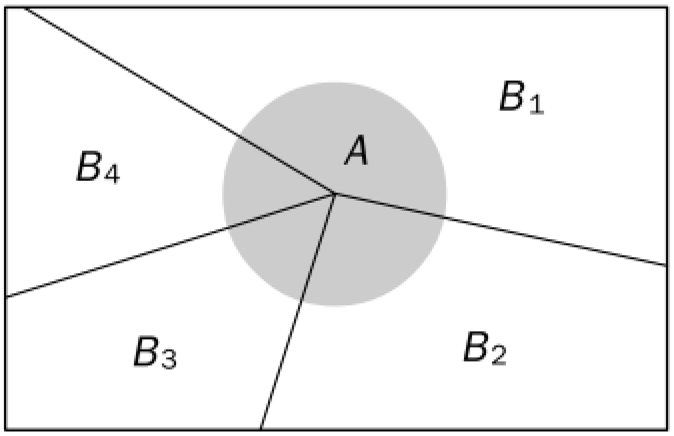
$$\sum_{i=1}^{n} p(A \cap B_i) = \sum_{i=1}^{n} p(A|B_i) \times p(B_i)$$

- Summed over an exhaustive list of events for Bi, we get :

$$\sum_{i=1}^{n} p(A \cap B_i) = p(A)$$

- Which reduces to:

$$p(A) = \sum_{i=1}^{n} p(A|B_i) \times p(B_i)$$



- If the occurrence of A depends on only two mutually exclusive events, i.e. B and NOT B, then above equation becomes

$$p(A) = p(A|B) \times p(B) + p(A|\neg B) \times p(\neg B)$$

- Similarly,

$$p(B) = p(B|A) \times p(A) + p(B|\neg A) \times p(\neg A)$$

- Substituting above equations in Bayesian Equation, we get:

$$p(A|B) = \frac{p(B|A) \times p(A)}{p(B|A) \times p(A) + p(B|\neg A) \times p(\neg A)}$$

**Conditional Probability**

Simple probability deals with independent events. If we know the probability of A, p(A) p(B). If two events are interdependent and the outcome of one affects the outcome of other, then we need to consider conditional probability. The conditional probability p(B|A), indicates the probability of event B given that we know event has occurred.

$P(B|A) = \frac{P(A \cap B)}{P(A)}$ $\quad P(A) \neq 0$

Note: A∩B denotes the events that occurs if and only if both A and B simultaneously in the same performance of the experiment under consideration.

**Unconditional Probability**

Given random variable X, P(X=x) denotes the unconditional probability of prior probability that X has value x in the absence of any other information. So prior probability or unconditional probability is the degree of belief according to a proposition in the absence of any other information.

**Joint probability distribution**

The expression P(a) is said to be defining prior probability distribution for the random variable 'a'. To denote probabilities of all random variables combinations, the expression $P(a_1, a_2)$ can be used. This is called Joint probability distribution.

**Example:** the probability that Rohit has cavity=0.5 then prior probability is written as:

P(cavity)=0.5 or P(cavity=true)=0.5

and the P(weather)={0.6,0.3,0.8,0.2}

defines the probability distribution for the random variable weather {Sunny, Rainy, Cloudy, Snow}. The Joint probability distribution is P {Weather, Cavity} can be represented as 4 x 2 tables of probabilities.

Full Joint probability distribution consists of three random variables such as {Weather, Cavity, Toothache}. It will be represented as 4 x 2 x2 table of probabilities.

When agent obtains evidence concerning previously unknown random variables in the domain, then prior probability is not used. Based on new information conditional or posterior probability are calculated. The notation is P(A|B).

Example: P(Cavity|Toothache)=0.7

It means, if patient has toothache (and no other information is known) then the chances of having cavity are 0.7.

Conditional probability can be defined in terms of unconditional probabilities.

$P(a|b)=\frac{P(a \wedge b)}{P(b)}$ it holds whenever P(b)>0

or, $P(a \wedge b) = P(a|b) * P(b)$. This is called product rule.

**<u>Bayesian Networks</u>**

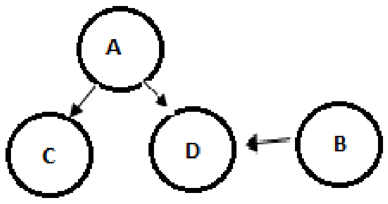
- A Bayesian Network is a data structure which represents the dependencies among variables and gives a concise specification of any full joint probability distribution.
- Also called Belief Networks or Probabilistic inference network
- It is a powerful knowledge representation and reasoning mechanism.
- It represents events and causal relationship between uncertainty as conditional probabilities involving random variables.
- Given the values of subset of these variables (evidence variables) Bayesian networks can compute the probabilities of another subset of variables (Query variables).
- A Bayesian Network is a directed acyclic graph (**DAG),** where there is a node for each random variable, and a directed arc from A to B whenever A is a direct causal influence on B.
- Thus **arcs** represent direct causal relationships and the **nodes** represent states of affairs.
- If there is are from A to B then, A is said to be parent of B.
- Each node $X_i$, has a conditional probability distribution $P(X_i \mid Parents(X_i))$ that quantifies the effect of the parents on the node. The set of nodes and links is called **topology of the network**.

In Bayesian Network, the joint probability distribution can be written as the product of the local distributions of each node and its parents such as:

$$P(X_1, X_{2,........,}X_n) = \prod_{i=1}^{n} P(Xi \mid \text{Parents } (Xi))$$

Where the set of parent nodes of a node $X_i$ is represented by parents ($X_i$).

**Example:** The graph shown in following figure represents a Bayesian belief network having four nodes A,B,C,D. {A,B} representing evidences and {C,D} representing hypothesis. Here A and B are unconditional nodes and C and D are conditional nodes. Arcs describes its dependency.



Suppose P(A)=0.3

P(B)=0.7

P(C|A)=0.4

P( C|~A)=0.3

P(D|A,B)=0.7

P(D|A,~B=0.3

P(D|~A,B)=0.2

P(D|~A,~B)=0.01

These probability values represent the complete Bayesian belief network. They can also be expressed in the form of conditional probability tables as:

| P(A) | P(B) |
|---|---|
| 0.3 | 0.7 |

| A | P(C) |
|---|---|
| T | 0.4 |

|  |  |
|---|---|
| F | 0.3 |

| A | B | P(D) |
|---|---|---|
| T | T | 0.7 |
| T | F | 0.3 |
| F | T | 0.2 |
| F | F | 0.01 |

We know that Joint Probability for four variables can be computed as:

P(A,B,C,D) = P(D|A,B,C) * P(C|A,B) * P(B|A) *P(A).

However, this can be simplified using Bayesian belief network as some of these terms will not be required. For example, C is not dependent on B; D is not dependent on C. Hence P(C|A,B) is reduced to P(C|A) and P(D|A,B,C) is reduced to P(D|A,B). Further, A and B are independent, therefore P(B|A) is reduced to P(B).

So, P(A,B,C,D) = P(D|A, B) * P(C|A) * P(B) * P(A)

=0.7 *0.4*0.7*0.3

=0.0588

**Constructing a Bayesian Network**

1. Initialize BN := $\{(X_1,X_2,......,X_n), E\}$ where $E = \phi$
2. Fix any order of variables $X_1,X_2,........,X_n$.
3. For i :=1,2,......,n
   - (i) Choose from $X_1,X_2,........,X_{i-1}$ ,a minimal set.
     Parents($X_i$) of parents so that $P(X_i |x_{i-1}.......X_1 ) = P(X_i |Parents(X_i))$
   - (ii) For each $X_j \in$ Parents ($X_i$), insert ($X_j$ , $X_i$) into E
   - (iii) Associate $X_i$ with CPT($X_i$) corresponding to $P(X_i |Parents(X_i))$.

**Advantages:**

- Handles situations where some data entries are missing.
- used to predict consequences of intervention
- allow to learn causal relationship.
- provides natural representation for conditional independence.

**Hidden Markov Models (HMM)**

The random variables have fixed values in the static words. However, there are many cases like sequence of spoken words, weather forecast etc., Where the environment changes in time. These are called dynamic problems. HMM is a widely used model for processing sequential data. It is used in speech recognition, NLP modeling, online handshaking recognition and for the analysis of biological sequences such as protein and DNA.

**<u>Markov Assumption</u>**

It states that the current state depends on only a finite history of previous state. The process which follows Markov assumption is called Markov process or Markov Chain.

**<u>Markov Model</u>**

Consider weather:

- ✓ We have three types of weather; Sunny, rainy, and foggy
- ✓ Assume weather, lasts all day i.e., does not change from rainy to sunny in the middle of the day
- ✓ Weather predictions is all about trying to guess what the weather will be like tomorrow based on history of observation of weather.
- ✓ Assuming we have collected data based on today, yesterday, the day before and so forth.

$$P\ (W_n \mid W_{n-1}, W_{n-2}, \ldots\ldots\ldots\ldots, W_1\ )$$

Using above equation, we can give probabilities of types of weather for tomorrow and the next day using n days of history. Suppose if we know that the weather for the past three days was { Sunny, Sunny, Foggy } in chronological order, the probability that tomorrow would be rainy in given by:

$$P\ (W_4 = \text{Rainy} \mid W_3 = \text{Foggy},\ W_2 = \text{Sunny},\ W_1 = \text{Sunny})$$

For it, the larger ‘n’ is; more statistics we must collected. Suppose n = 5, then we must collect statistics for $3^5 = 243$ past histories. Therefore we will make a simplifying assumption, called Markov Assumption.

In a sequence { $W_1, W_2, W_3, \ldots\ldots, W_n$ ):

$$P\ (W_n \mid W_{n-1}, W_{n-2}, \ldots\ldots\ldots\ldots, W_1\ ) \approx P\ (W_n \mid W_{n-1}\ )$$

This is called First order Markov Assumption, since we say that the probability of an observation at time n only depends on the observation at time n-1. A second-order Markov assumption would have the observation at time n depend on n-1 and n-2. In general, when people talk about Markov assumptions, they usually mean first-order Markov assumption.

We can express the joint probability using the Markov assumption.

$$P(W_1, \ldots\ldots\ldots\ldots, W_n) = \prod_{i=1}^{n} P(W_i \mid W_{i-1})$$

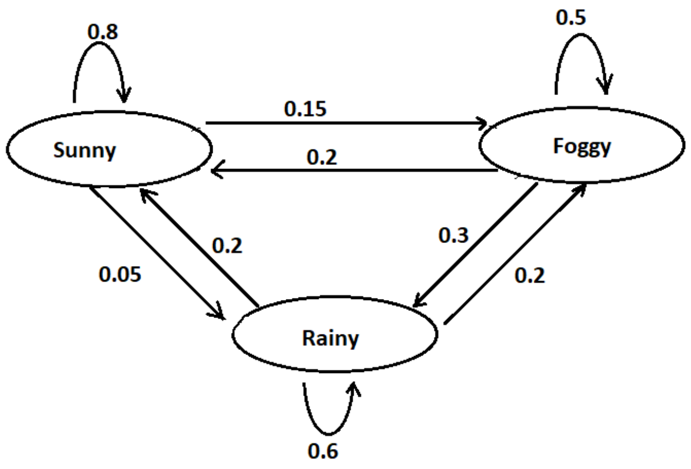
**Note:** This equation has a profound affect on the number of histories that we have to find statistics for we now only need $3^2 = 9$ numbers to characterize the probabilities of all of the sequences. This assumption may or may not be valid assumption depending on the situation (in case of weather, it's probably not valid) but we use these to simplify the situation.

**Example:**

Probabilities of Tomorrow's weather based on today's weather.

| | | Tomorrow's weather | | |
|---|---|---|---|---|
| | | Sunny | Rainy | Foggy |
| Today's Weather | Sunny | 0.8 | 0.05 | 0.15 |
| | Rainy | 0.2 | 0.6 | 0.2 |
| | Foggy | 0.2 | 0.3 | 0.5 |

Draw Finite Automation:



Questions:

1. Given that today is Sunny, what's the probability that tomorrow is sunny and the day after is rainy ?

**Solution:**

This translates into:

P(w2 = sunny, w3= Rainy | w1 = Sunny)

= P(w3= Rainy | w2 = sunny) * P(w2 = Sunny | w1 = Sunny )

= (0.05) * (0.8)

= 0.04

2) Given that today is foggy, what's the probability that it will be rainy two days from now ?

Solution:

There are three ways to get from foggy today to rainy two days from now: {foggy, foggy, rainy}, {foggy, rainy, rainy} and {foggy, sunny, rainy}. Therefore, we have to sum over these paths:

P(w3 = Rainy | W1= Foggy) = P(w2 = Foggy, w3 =Rainy | w1 = Foggy ) +

P(w2 = Rainy, w3 =Rainy |w1 = Foggy) +

P(w2 = Sunny, w3 =Rainy | w1 = Foggy)

= P(w3 = Rainy | w2 = Foggy ) * P(w2 = Foggy | w1= Foggy) +

P(w3 = Rainy| w2 = Rainy) * P(w2 = Rainy | w1 = Foggy) +

P(w2 = Rainy | w2 = Sunny) * P(w2 = Sunny | w1 = Foggy)

= (0.3) * (0.5) + (0.6) * (0.3) + (0.05) * (0.2)

= 0.34

**Hidden Markov Model:**

Suppose you are locked in a room for several days and you are asked for the weather outside. The only piece of evidence you have is whether the person who comes into the room carrying your daily meal is carrying an umbrella or not.

Let's suppose the following probabilities

| DAY | Probability of Umbrella |
|---|---|
| Sunny | 0.1 |
| Rainy | 0.8 |
| Foggy | 0.3 |

Remember equation:

$$P(W_1, \ldots\ldots\ldots, W_n) = \prod_{i=1}^{n} P(W_i \mid W_{i-1})$$

Now, we have to factor in the fact that the actual weather is hidden from you. We do that by using Baye's rule.

$$P(W_1, \ldots\ldots, W_n \mid u_1, \ldots\ldots, u_n) = \frac{P(u_1 \ldots\ldots u_n \mid W_1 \ldots\ldots W_n)\, P(W_1 \ldots\ldots W_n)}{P(u_1 \ldots\ldots u_n)}$$

Where,

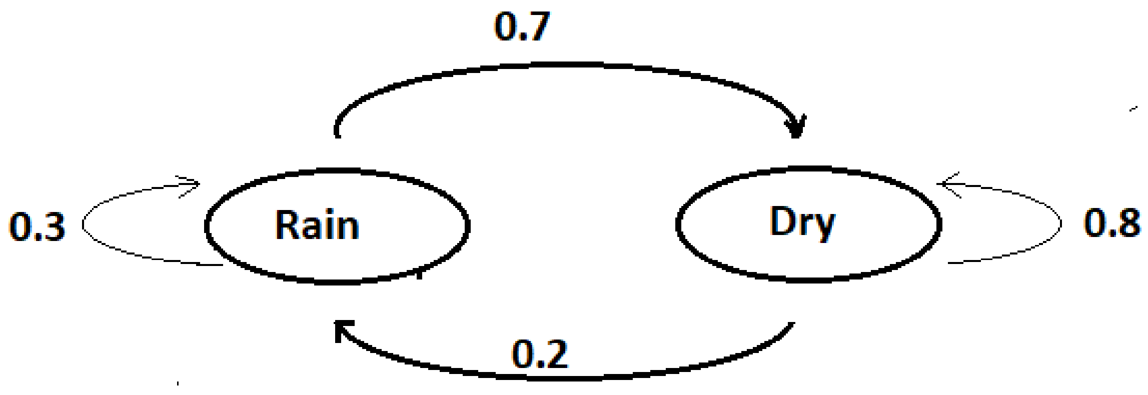
$u_i$ is true if caretaker brought an umbrella on day i, and false if the caretaker didn't.

The probability $P(u_1 \ldots\ldots u_n \mid W_1 \ldots\ldots W_n)$ can be entimated as :

$$\prod_{i=1}^{n} P(u_i \mid W_i)$$

If you assume, for all i , given $W_i$, $u_i$ is independent of all $u_j$ and $w_j$ for all $j \neq i$.

**Example:**



Two states: Rain, Dry

Transition probabilities:

P(Rain | Rain ) = 0.3 , P ( Dry | Rain ) = 0.7

P (Rain | Dry ) = 0.2, P (Dry | Dry ) = 0.8

Initial probabilities: say

P( Rain ) = 0.4 P ( Dry ) = 0.6

**Calculation of HMM**

By Markov chain property, probability of state sequence can be found by the formula :

$P = (S_{i1} , S_{i2} , \ldots\ldots , S_{ik} ) = P (S_{ik} | S_{i1}, S_{i2}, \ldots\ldots, S_{ik-1} )$

$P (S_{i1}, S_{i2}, \ldots\ldots, S_{ik-1} )$

$=P(S_{ik} | S_{ik-1} ) . P (S_{i1}, S_{i2}, \ldots\ldots, S_{ik-1} )$

$= P(S_{ik} | S_{ik-1} ). P(S_{ik-1} | S_{ik-2} ) \ldots\ldots P(S_{i2} | S_{i1} ). P(S_{i1} )$

Suppose we want to calculate a probability of a sequence of states in out examples

{ Dry, Dry,Rain, Rain}

P( { Dry, Dry,Rain, Rain} )

=P(Rain|Rain) P(Rain|Dry) P(Dry|Dry) P(Dry)

=0.3 * 0.2 * 0.8 * 0.6

**Application**

- Speech Recognition
- Text Processing
- Bio-informatics
- Financial

**Dynamic Bayes Network**

- Dynamic Bayes Network is a generalization of hidden Markov Model having Kalman Filters.
- If a Bayesian Network that relates Variables to each other over adjacent time steps.
- At any point of time ‘ t ’, the value of the variable can be calculated from the internal regression and the immmediate prior value (at time t-1 ).
- Used in robotics, data mining speech recognition, digital forensics, protein sequencing and bioinformatics.

# Unit 6 Genetic Algorithm

## Introduction

- A Genetic Algorithm (GA) is a randomized search and optimization techniques guided by the principle of natural genetic system. Recently, there has been a great deal of interest in GAs and their application to various engineering fields. The GA is also being applied to a wide range of optimization and learning problems in many domains.
- The main advantage of the GA formulation is that fairly accurate results may be obtained using a very simple algorithm. It is a method of finding a good answer to a problem, based on feedback received from its repeated attempts at a solution.
- GAs are optimization algorithms based on the principle of biological evolution. They use probabilistic rules to guide their search.
- By considering many points in the search space simultaneously, they reduce the chances of converging to local minima.
- **Algorithm**: An algorithm is a sequence of instructions to solve a problem. Most of the algorithms are static.
- **A Genetic Algorithm(GA)** is adaptive (dynamic) a model of machine learning algorithm that derives its behavior from a metaphor of some of the mechanisms of evolution in nature.

## Background

- On 1 July 1858, **Charles Darwin,** presented his **theory of evolution**. This day marks the beginning of a revolution in Biology.
- **Darwin**'s classical **theory of evolution** , together with Weismann's **theory of natural selection** and Mandel's concept of **genetics** , now represent the **Neo-Darwinism**
- **Neo-Darwinism** is based on process of reproduction, mutation, competition and selection.

- Evolution can be seen as a process leading to the maintenance of a population's ability to survive and reproduce in a specific environment. This ability is called **evolutionary fitness**.
- Evolutionary fitness can also be viewed as a measure of the organism's ability to anticipate changes in its environment.
- The fitness, or the quantitative measure of the ability to predict environmental changes and respond adequately, can be considered as the quality that is optimized in natural life

## Simulation of Natural Evolution

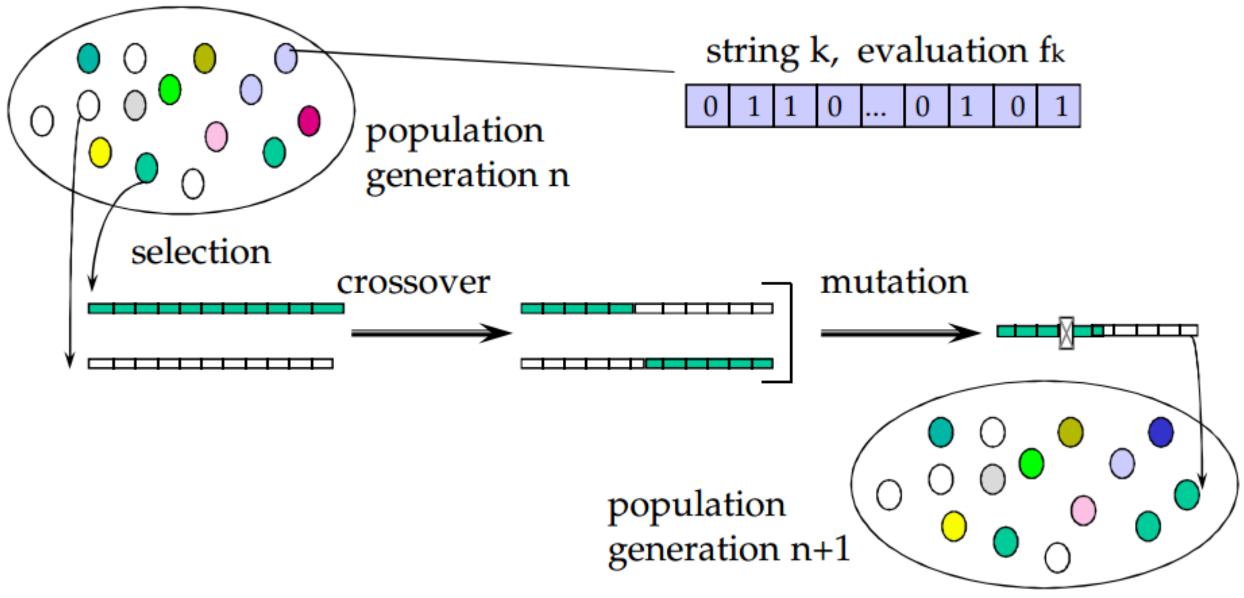
- All methods of evolutionary computation simulate natural evolution by creating a population of individuals, evaluating their fitness, generating a new population through genetic operations, and repeating this process a number of times.
- We focus on **Genetic Algorithm** as most of the other algorithms can be viewed as variations of genetic algorithms.

## Genetic Algorithm

- In early 1970s John Holland introduced the concept of genetic algorithm
- His aim was to make computers do what nature does. Holland was concerned with algorithms that manipulate strings of binary digits
- Each artificial "chromosomes" consists of a number of "genes", and each gene is represented by 0 or 1
- Two mechanisms link a GA to the problem it is solving : **Encoding** and **Evaluation**
- The GA uses a measure of fitness of individual chromosomes to carry out reproduction. As reproduction takes place, the crossover operator exchanges part of two single chromosomes and the mutation operator changes the gene value in some randomly chosen location of the chromosome

### Basic Genetic Algorithm

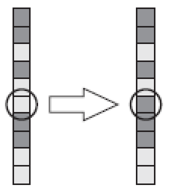
- The GA maintains a set of possible solutions (population) represented as a string of binary numbers (0/1).
- New strings are produced in each and every generation by the repletion of a two-step cycle. This involves first decoding each individual string and assessing its ability to solve the problem.
- Each string is assigned fitness values, depending on how well it has performed in an Environment.
- In the second stage, the fittest string is preferentially chosen for recombination, which involves the selection of two strings, and the switching of the segments to the right of the meeting point of the two strings. This is called ***crossover***
- Another genetic operator is called ***mutation.*** It is used to maintain genetic diversity within a small population of strings. There is a small probability that any bit in a string will be flipped from its present value to its opposite (e.g. 0 to 1).
- The GA goes through the following cycle: Evaluate, Select and Mate and Mutate until some kind of stopping criteria are reached.

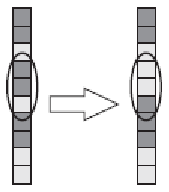


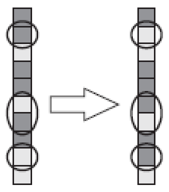
## GA Operators and Parameters

- Fitness function: The fitness function is defined over the genetic representation and measures the *quality* of the represented solution.
- Selection Operator: Selects parents for reproduction based on relative fitness of candidates in the population
    - Roulette Wheel Selection
    - Ranking Selection
- **Mutation Operator:**
    - Changes a randomly selected gene in the chromosome
    - mimics random changes in genetic code
    - Background operator to provide exploration in search to avoid being trapped on a local optimum
    - Mutation probability is quiet small in nature and is kept low for GAs, typically in the range between [ 0.001 – 0.01] or by formula :

P(m) = 1/no, of bits in Chromosomes


a: Single-gene mutation.


b: Multi-gene mutation

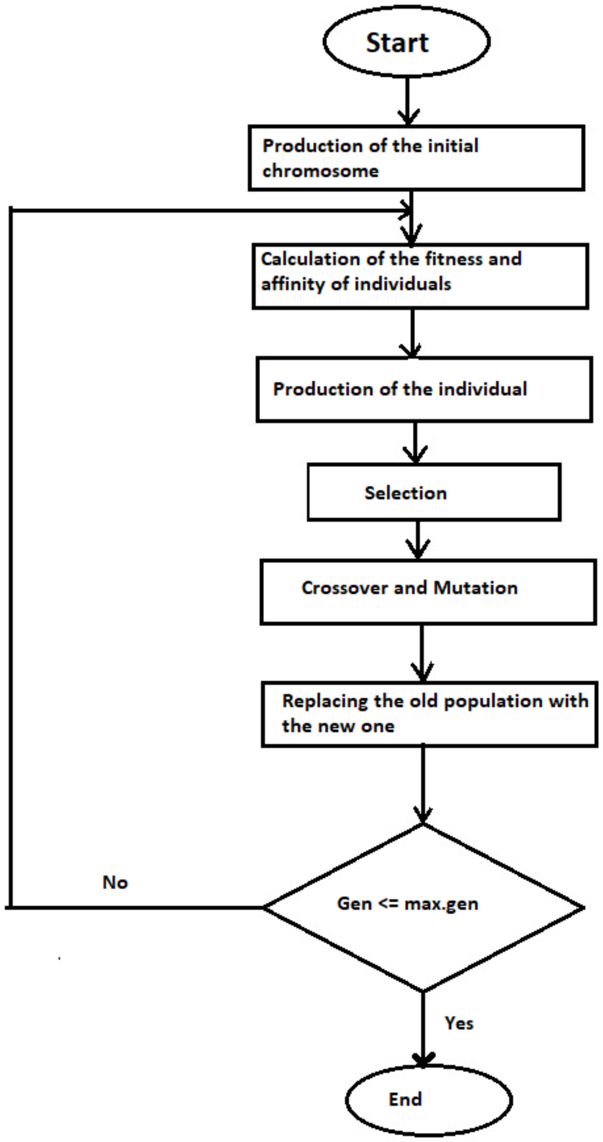

c: Multi-gene mutation

- **Elitism Approach** : Saves the best individual in next generation
- **Basic GA Parameters:**
    - Population size
    - Crossover rate (Probability)
    - Mutation Rate (Probability)

- Number of Generation ( a Stopping Criterion)

**Steps in Genetic Algorithm**

1. Represent the problem variable as a chromosome of a fixed length, choose the size of a chromosome population *N*, the crossover probability *pc* and the mutation probability *pm*.
2. Define a fitness function to measure the fitness of an individual chromosome in the problem domain.
3. Randomly generate an initial population of chromosomes of size *N*: *x*1, *x*2,.., *xN*
4. Calculate the fitness of each individual chromosome: *f*(*x*1), *f*(*x*2), . . . , *f*(*xN*)
5. Select a pair of chromosomes for mating from the current population based on their fitness.
6. Create a pair of offspring chromosomes by applying the genetic operators − **crossover** and **mutation**.
7. Place the created offspring chromosomes in the new population.
8. Repeat *Step 5* until the size of the new chromosome population becomes equal to the size of the initial population, *N*.
9. Replace the initial (parent) chromosome population with the new (offspring) population.
10. Go to *Step 4*, and repeat the process until the termination criterion is satisfied


Start
Production of the initial chromosome
Calculation of the fitness and affinity of individuals
Production of the individual
Selection
Crossover and Mutation
Replacing the old population with the new one
Gen <= max.gen
No
Yes
End

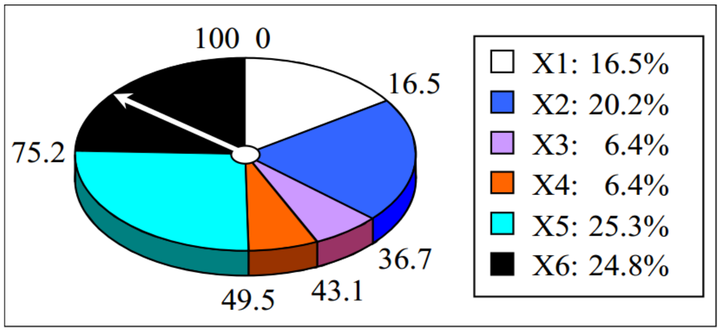
**GA: Case Study**

- In natural selection, only the fittest species can survive, breed, and thereby pass their genes on to the next generation. GAs use a similar approach, but unlike nature, the size of the chromosome population remains unchanged from one generation to the next.
- The last column in Table shows the ratio of the individual chromosome's fitness to the population's total fitness. This ratio determines the chromosome's chance of being

selected for mating. The chromosome's average fitness improves from one generation to the next

**Roulette-Wheel Selection**

- The most commonly used chromosome selection technique is the roulette wheel selection.



- In our example, we have an initial population of 6 chromosomes. Thus, to establish the same population in the next generation, the roulette wheel would be spun six times.

- Once a pair of parent chromosomes is selected, the **crossover** operator is applied

If needed **Mutation** is also carried out to avoid being trapped in local minimum

**<u>Evolutionary Programming</u>**

- Evolutionary programming is a technique used to search for the optimal solution to a problem by evolving a population of candidate solution over a number of generations or iterations.
- The solution is evolved through mutation and competitive selection
- The structure of an evolutionary programming (EP) algorithm is shown in figure below
- In this approach, the real-valued decision variables to be determined are represented by a trial n-dimensional vector.
- Each vector is an individual of the population to be evolved.
- Genetic Algorithm (GA) methods are used for encoding, decoding and fitness function formulation, Evolutionary Programming (EP), on the other hand, is better for obtaining the global optimum, which relies on mutation rather than crossover.
- Due to inherent flexibilities in the fitness function and the ease of coding, the EP method produces the best solution with fewer generations.

- EP is a probabilistic search technique which generates the initial parent vectors distributed uniformly in intervals within the limits and obtains the global optimum solution over a number of iterations.
- The main stages of this technique are initialization, creation of offspring vectors by mutation and competition and selection of the best vectors in order to evaluate the best fitness solution

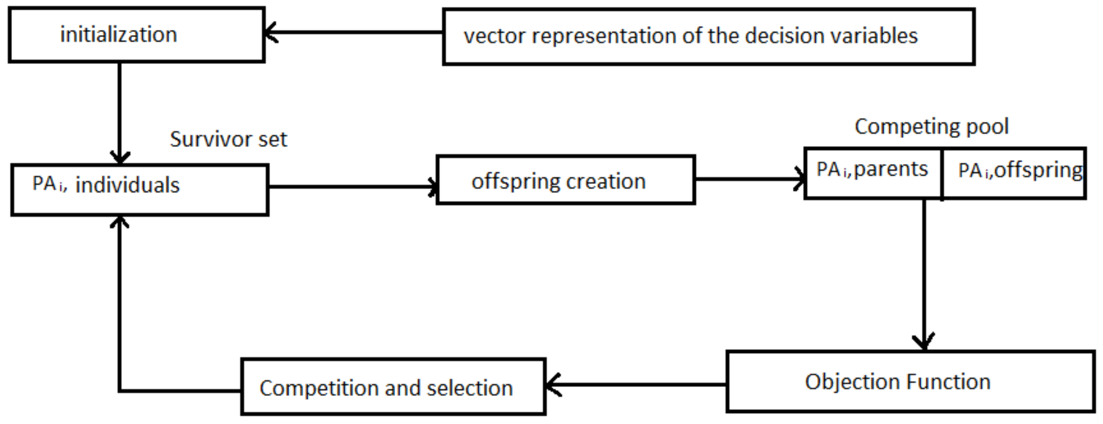


Fig: Structure of an evolutionary programming algorithm

## Initialization

- An initial population of parent individuals (PA $_i$, i = 1,2,3,…, K) is generated randomly within a feasible range in each dimension.

## Mutation (Creation of offspring)

- Each Parent vector PA $_i$ generates an offspring vector by adding a Gaussian random variable with a zero mean and pre-selected standard deviation to each individual of PA $_i$ .
- The K parents create K offspring, thus resulting in 2K individuals in the competing pool.

## Competition and selection

- Each individual in the competing pool is evaluated for its fitness.
- All individuals compete with each other for selection.

- The best K individuals with the maximum fitness values are retained to be parents in the next generation.
- The process of creating offspring and selecting those with maximum fitness is repeated until there is no appreciable improvement in the maximum fitness value or it has been repeated a prescribed number of times.

**Applications**

Genetic algorithms are used to solve many large problems including:

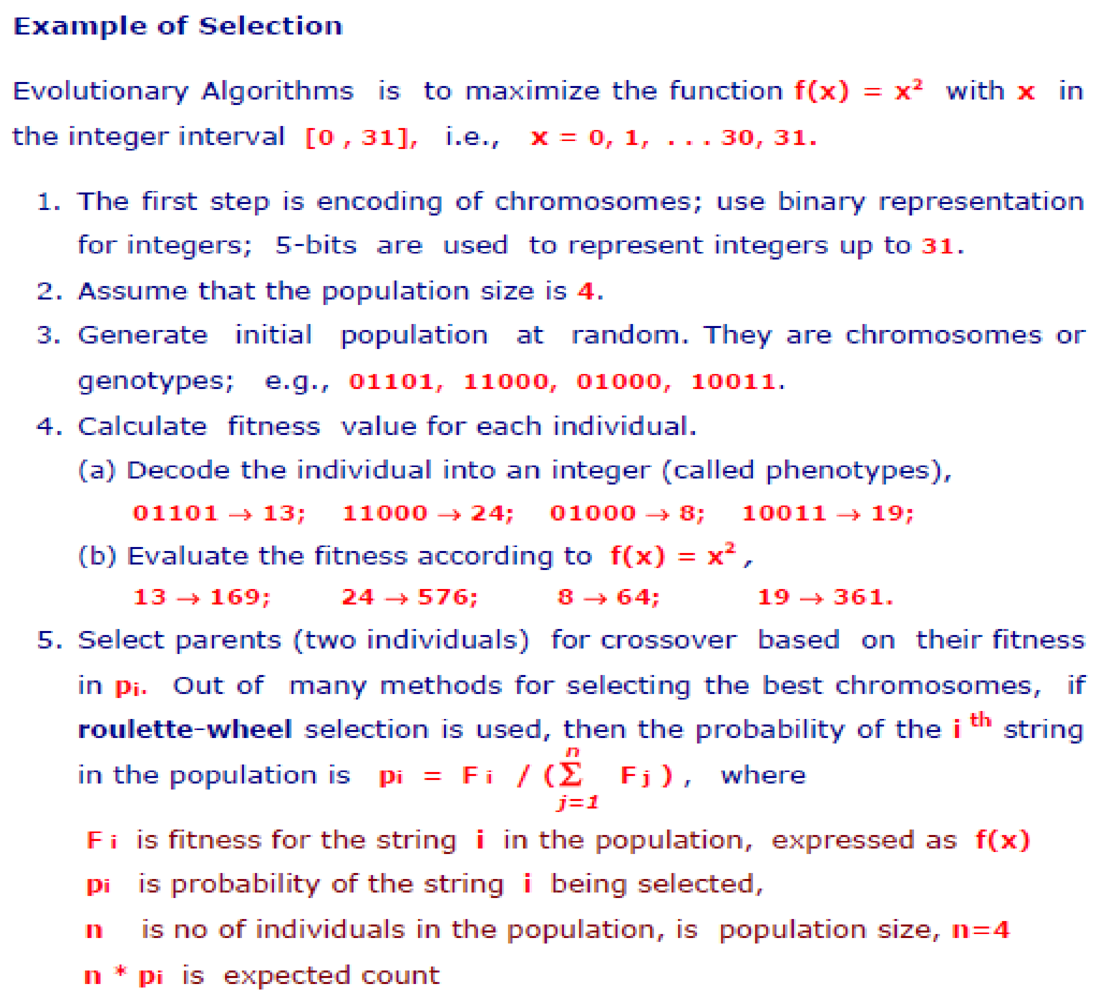
- Scheduling
- Transportation
- Chemistry, Chemical Engineering
- Layout and circuit design
- Medicine
- Data Mining and Data Analysis
- Economics and Finance
- Networking and Communication
- Game etc.

**Advantages of Genetic Algorithm:**

- It can solve every optimization problem which can be described with the chromosome encoding.
- It solves problems with multiple solutions.
- Since the genetic algorithm execution technique is not dependent on the error surface, we can solve multi-dimensional, non-differential, non-continuous, and even non-parametrical problems.
- Structural genetic algorithm gives us the possibility to solve the solution structure and solution parameter problems at the same time by means of genetic algorithm.
- Genetic algorithm is a method which is very easy to understand and it practically does not demand the knowledge of mathematics.
- Genetic algorithms are easily transferred to existing simulations and model

**Disadvantages**

- May be Slow
- May be drop of the quality because of crossover.



**Example of Selection**

Evolutionary Algorithms is to maximize the function $f(x) = x^2$ with $x$ in the integer interval $[0 , 31]$, i.e., $x = 0, 1, \ldots 30, 31$.

1. The first step is encoding of chromosomes; use binary representation for integers; 5-bits are used to represent integers up to 31.
2. Assume that the population size is 4.
3. Generate initial population at random. They are chromosomes or genotypes; e.g., 01101, 11000, 01000, 10011.
4. Calculate fitness value for each individual.
   (a) Decode the individual into an integer (called phenotypes),
   $01101 \rightarrow 13$; $11000 \rightarrow 24$; $01000 \rightarrow 8$; $10011 \rightarrow 19$;
   (b) Evaluate the fitness according to $f(x) = x^2$ ,
   $13 \rightarrow 169$; $24 \rightarrow 576$; $8 \rightarrow 64$; $19 \rightarrow 361$.
5. Select parents (two individuals) for crossover based on their fitness in $p_i$. Out of many methods for selecting the best chromosomes, if **roulette-wheel** selection is used, then the probability of the $i$ th string in the population is $p_i = F_i \; / \; ( \sum_{j=1}^{n} F_j )$ , where

   $F_i$ is fitness for the string $i$ in the population, expressed as $f(x)$
   $p_i$ is probability of the string $i$ being selected,
   $n$ is no of individuals in the population, is population size, $n=4$
   $n * p_i$ is expected count

| String No | Initial Population | X value | Fitness Fi $f(x) = x^2$ | p i | Expected count N * Prob *i* |
|---|---|---|---|---|---|
| 1 | 0 1 1 0 1 | 13 | 169 | 0.14 | 0.58 |
| 2 | 1 1 0 0 0 | 24 | 576 | 0.49 | 1.97 |
| 3 | 0 1 0 0 0 | 8 | 64 | 0.06 | 0.22 |
| 4 | 1 0 0 1 1 | 19 | 361 | 0.31 | 1.23 |
| Sum | | | 1170 | 1.00 | 4.00 |
| Average | | | 293 | 0.25 | 1.00 |
| Max | | | 576 | 0.49 | 1.97 |

The string no 2 has maximum chance of selection.

6. Produce a new generation of solutions by picking from the existing pool of solutions with a preference for solutions which are better suited than others:

We divide the range into four bins, sized according to the relative fitness of the solutions which they represent.

| *Strings* | *Prob i* | *Associated Bin* |
|---|---|---|
| 0 1 1 0 1 | 0.14 | 0.0 . . . 0.14 |
| 1 1 0 0 0 | 0.49 | 0.14 . . . 0.63 |
| 0 1 0 0 0 | 0.06 | 0.63 . . . 0.69 |
| 1 0 0 1 1 | 0.31 | 0.69 . . . 1.00 |

By generating **4** uniform **(0, 1)** random values and seeing which bin they fall into we pick the four strings that will form the basis for the next generation.

| *Random No* | *Falls into bin* | *Chosen string* |
|---|---|---|
| 0.08 | 0.0 . . . 0.14 | 0 1 1 0 1 |
| 0.24 | 0.14 . . . 0.63 | 1 1 0 0 0 |
| 0.52 | 0.14 . . . 0.63 | 1 1 0 0 0 |
| 0.87 | 0.69 . . . 1.00 | 1 0 0 1 1 |

7. Randomly pair the members of the new generation

Random number generator decides for us to mate the first two strings together and the second two strings together.

8. Within each pair swap parts of the members solutions to create offspring which are a mixture of the parents :

For the first pair of strings: **0 1 1 0 1 , 1 1 0 0 0**

– We randomly select the crossover point to be after the fourth digit.

Crossing these two strings at that point yields:

**0 1 1 0 1 ⇒ 0 1 1 0 |1 ⇒ 0 1 1 0 0**

**1 1 0 0 0 ⇒ 1 1 0 0 |0 ⇒ 1 1 0 0 1**

For the second pair of strings: **1 1 0 0 0 , 1 0 0 1 1**

– We randomly select the crossover point to be after the second digit.

Crossing these two strings at that point yields:

**1 1 0 0 0 ⇒ 1 1 |0 0 0 ⇒ 1 1 0 1 1**

**1 0 0 1 1 ⇒ 1 0 |0 1 1 ⇒ 1 0 0 0 0**

9. Randomly mutate a very small fraction of genes in the population :

With a typical mutation probability of per bit it happens that none of the bits in our population are mutated.

10. Go back and re-evaluate fitness of the population (new generation) :

This would be the first step in generating a new generation of solutions. However it is also useful in showing the way that a single iteration of the genetic algorithm has improved this sample.

| *String No* | *Initial Population (chromosome)* | *X value (Pheno types)* | *Fitness $f(x) = x^2$* | *Prob i (fraction of total)* | *Expected count* |
|---|---|---|---|---|---|
| 1 | 0 1 1 0 0 | 12 | 144 | 0.082 | 0.328 |
| 2 | 1 1 0 0 1 | 25 | 625 | 0.356 | 1.424 |
| 3 | 1 1 0 1 1 | 27 | 729 | 0.415 | 1.660 |
| 4 | 1 0 0 0 0 | 16 | 256 | 0.145 | 0.580 |
| Total (sum) | | | 1754 | 1.000 | 4.000 |
| Average | | | 439 | 0.250 | 1.000 |
| Max | | | 729 | 0.415 | 1.660 |

Observe that :

1. Initial populations : At start step 5 were

   **0 1 1 0 1 , 1 1 0 0 0 , 0 1 0 0 0 , 1 0 0 1 1**

   After one cycle, new populations, at step 10 to act as initial population

   **0 1 1 0 0 , 1 1 0 0 1 , 1 1 0 11 , 1 0 0 0 0**

2. The total fitness has gone from **1170** to **1754** in a single generation.
3. The algorithm has already come up with the string 11011 (i.e **$x = 27$**) as a possible solution.

# Unit 7 Expert System

**What is Expert System ?**

Expert system is an artificial intelligence program that has expert-level knowledge about a particular domain and knows how to use its knowledge to respond properly. Domain refers to area within which the task is being performed. Ideally, the expert system should substitute a human expert.

Edward Feigenbaum of Stanford University has defined expert system as "an intelligent computer program that uses knowledge and inference procedures to solve programs that are difficult enough to require significant human expertise for their solutions."

Expert systems are the computer application developed to solve complex problems in a particular domain, at the level of extra-ordinary human intelligence and expertise.

**Why Expert system ?**

Expert System is built because of two factors : either to replace or to help an expert.

- To enable the use of expertise after working hours or at different locations.
- To automate a routine task that requires human expertise all the time unattended, thus reducing operational costs.
- To replace a retiring or leaving employee who is an expert.
- To hire an expert is costly.
- To help Expert in their routine to improve productivity
- Effective Management of problems.
- Easy access of the information.

**Comparisons between an Expert System and Human Expert**

| Factor | Human Expert | Expert System |
|---|---|---|
| Time (can be obtained) | Working days Only | Anytime |
| Geography | Local | Anywhere |
| Safety | Cannot be replaced | Can be replaced |
| Damages | Yes | No |
| Speed and Efficiency | Changes | Consistent |
| Cost | High | Intermediate |

## Characteristics of the Expert System

i ) **High Performance.** They should perform at the level of a human expert.

ii) **Adequate response time.** They should have the ability to respond in a reasonable amount of time. Time is crucial for real time system

iii) **Reliability.** They must be reliable and should not crash.

iv) **Understandable:** They should not be a black box instead it should be able to explain the steps of reasoning process. It should justify its conclusions in the same way a human expert explains why he/she arrived at a particular conclusion.

v) Should be able to display the intelligent behavior.

vi) Should be able to explain the reasoning.

vii) Should be able to draw conclusion.

viii)Should be able deal with the certainty

ix) Does not possess the ability to deal with the mixed knowledge.

x) Limited to narrow problems.

xi) Are very much difficult to maintain.

## Components of Expert Systems

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

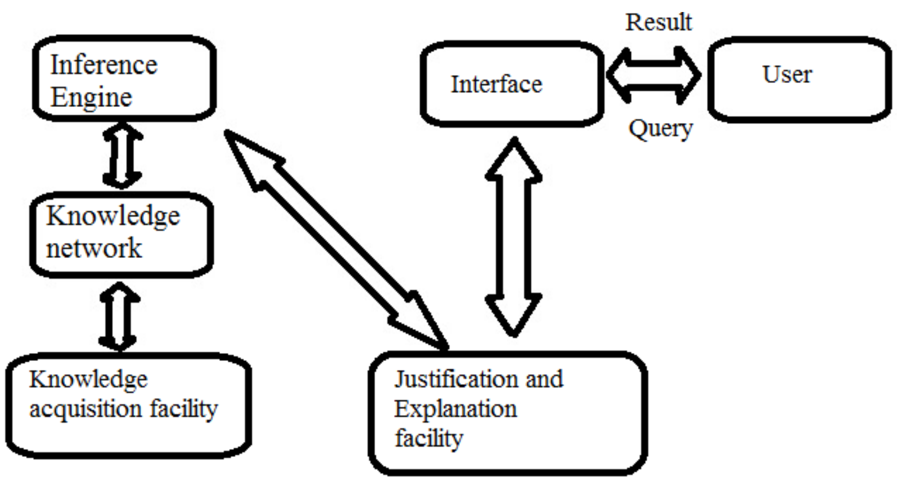

Inference Engine
Knowledge network
Knowledge acquisition facility
Justification and Explanation facility
Interface
Result
Query
User

**Fig: Basic Components of an Expert System**

**Knowledge Base**

- It contains domain specific and high quality knowledge, where knowledge is defined as the collection of data, information (data and facts) and past experience.
- Knowledge is required to exhibit intelligence.
- Success of Expert System depends on collection of highly accurate and precise knowledge.

**Components of Knowledge**

Knowledge base of Expert System stores factual and heuristic knowledge.

- **Factual Knowledge:** information widely accepted by the knowledge Engineers and scholars in the task domain.
- **Heuristic Knowledge:** It is about practice, accurate judgment, one's ability of evaluation and guessing.

**Knowledge Representation**

- Method used to organize and formalize the knowledge base.
- It is the form of IF-THEN-ELSE rules.

**Knowledge Acquisition**

- Expert System majority depends on the quality, completeness and accuracy of the information stored in the knowledge base.
- Knowledge base is formed by reading from various experts, scholars and the knowledge Engineers.
- Knowledge Engineer acquires information from subject expert by recording, interviewing and observing him at work. Etc.
- Knowledge Engineer then categorizes and organizes the information in a meaningful way, in the form of IF-THEN-ELSE rules, to be used by interference machine.
- Knowledge also monitors the development of ES.

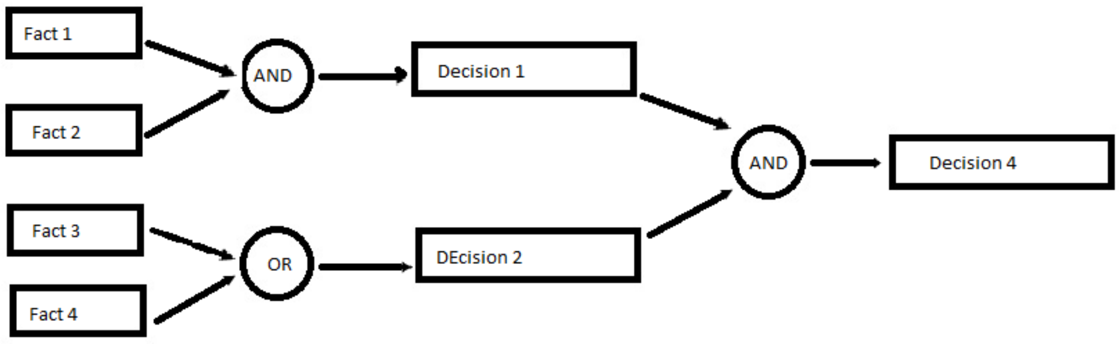
**Interface Engine**

- Interface engine is essential in deducting a correct, flawless solution.
- In case of Knowledge-based ES, Interface Engine acquires and manipulates the knowledge from the knowledge base to arrive at a particular solution.
- In case of rule based ES, it-
  - ➢ Applies rules repeatedly to the facts, which are obtained from earlier rule application
  - ➢ Adds new knowledge in the knowledge base if required.
  - ➢ Resolves rules conflict when multiple ruled are applicable to a particular case.

    To recommend the solution, the interface engine uses the following strategies:
  - ➢ Forward Chaining
  - ➢ Backward Chaining

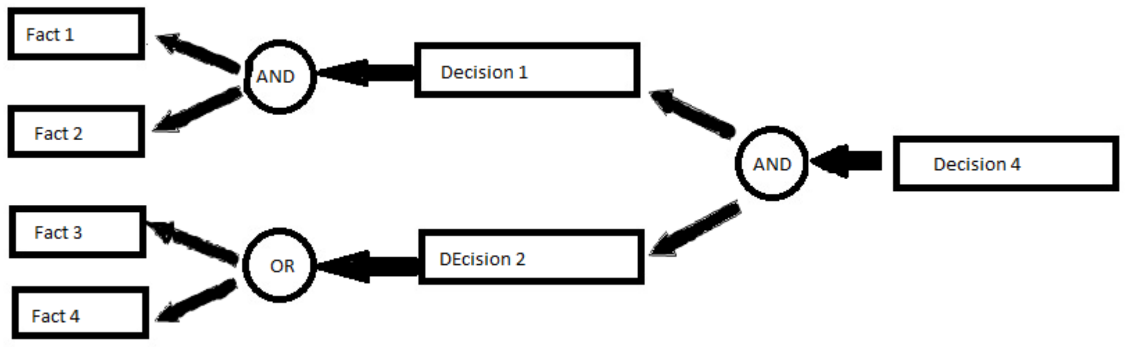
**Forward Chaining:**

- It is a strategy to explain the question, "What can happen next?"
- This strategy is followed for working on conclusion, result or effect.



**Backward Chaining**

- It can finds the answer “Why this happened ?”
- This strategy is used for finding out cause or reason.
- For example, diagnosis of blood cancer in humans.



**User Interface**

- It provides interaction between user of ES and ES itself.
- It uses Natural Language Processing
- User need not to be necessarily an expert system
- It explains how an ES has arrived at a particular recommendation.

**<u>Development of Expert System</u>**

**Step 1: Identify Problem Domain**

- Problem must be suitable for an expert system to solve it
- Find the expert in task domain for the ES project.
- Establish cost-effectiveness of the system

**Step 2 Design The system**

- Identify the ES Technology
- Know and establish the degree if integration with the other systems and databases
- Realize how the concepts can represent the domain knowledge best.

**Step 3 Develop the prototype**

From Knowledge Base, The Knowledge engineer works to:

- Acquire domain knowledge from the expert.
- Represent it in the form of IF-THEN-ELSE rules

**Step 4 Test and Refine the prototype**

- The knowledge engineer uses samples cases to test the prototype for any deficiencies in performance.
- End users test the prototypes of ES

**Step 5 Develop and Complete the ES**

- Test and ensure the interaction of the ES with all elements of its environment, including end users, database, and other information systems
- Document the ES project well
- Train the user to use ES

**Step 6 Maintain the ES**

- Keep the knowledge base up-to-date regular review and update
- Cater for new interfaces with other information systems, as those systems evolve.

**Expert System Technology**

- ES technologies include:

**i) Expert System Development Environment.**
  - Workstations, minicomputers, mainframes
  - Programming language like LISP and PROLOG
  - Large database

**ii) Tools**
  - Powerful editors and debugging tools with multi-windows.
  - They provide rapid prototyping
  - Have inbuilt definitions of model, knowledge representation and interference design

**iii) Shells**
  - It is an ES without knowledge base.
  - Provides the developers with knowledge acquisition, inference engine, user interface and explanation facility
  - Example: Java expert System Shell (JESS) that provides fully developed JAVA API for an expert system

## Capabilities of Expert Systems

1. Capturing of the expertise
2. Codifying the expertise
3. Duplicating the expertise
4. Transferring the expertise
5. Saving the human effort's time
6. Saving the maintenance
7. Update the knowledge base on the regular basis

## Limitations of Expert Systems:

1. Not adopted in highly sophisticated sensory inputs
2. Mainly function in the domain of the extracted, cognitive, logical thinking process.
3. Multi-dimensional problems from multiple users can not be faced by ES
4. Lack of common-sense knowledge and broad –ranging contextual information
5. Very narrow range of knowledge is incorporated in the Expert Systems
6. Could not respond to the outside range of the expertise
7. Human's self awareness is lacking in Expert System.

## Probability Based Expert system

- Must incorporate uncertainty
- Probability of single event A, P(A) can be calculated as:
- $P(A_1 \text{ AND } A_2 \text{ AND } A_3 .......A_n)=P(A_1A_2A_3...A_n)$ –"AND" combined probability
- $P(A_1 \text{ OR } A_2 \text{ OR } A_3 .......A_n)=P(A_1+A_2+A_3+...+A_n)$ –"OR" combined probability

Uncertainty in Expert System can exist in two possible ways

**Type 1 uncertainty** : Uncertainty about the initial inputs or at least one of them which propagates through the rule to final conclusion

**Type 2 uncertainty** : Uncertainty about the validity of the rule even if input is valid

## Monte Carlo Simulation:

- General procedure for solving any combined probability problems
  **Steps:**
  1.) Sequence of random input is executed whose occurring probability is denoted by $E_i$
  2.) If random number generated < $E_i$ THEN input = True for current trial which will propagate to the nodes

3.) For AND node, E will be true if ($E_1E_2...E_n$) = True
For OR node, E node be true if ($E_1+E_2+...+E_n$) = True
4.) The process ends when final conclusion being true.

# Intelligent System

## Unit 8:Swarm Intelligence

Er. Ranjan Raj Aryal

aryalranjan@gmail.com

## Introduction

- Nature has guided us to watch and learn the intelligent mechanism evolved by it.
- Marching of ant in an army,
- The waggle dance of honey bee,
- the nest building of social wasp, birds flocking in high skies,
- fish schools in deep waters,
- foraging activities of microorganisms,
- the construction of termite mound, etc.
- These activities have puzzled biologists over the years.
- The last decade has seen an explosion of research in this field variously referred to as collective intelligence, swarm intelligence and emergent behaviors.
- Swarm Intelligence is used for the collective behavior of a group of animals as a single living creature,.

## Characteristics of Swarm Intelligence

- ***Adaptation:*** Swarm Intelligence is a specialization in the field of self-organizing systems
- ***Robustness*:** When a route of ant is blocked, this can be observed that they find other new shortest new route to their destination.
- ***Reliable*:** Agents can be added or removed without compromising the total system due to its distributed nature. This adaption is called Reliable.
- ***Simplicity*:** Single part may break down without impairing the overall system such that complex system are convenient to work

## Ant Intelligent System (Background)

- In 1991 Marco Dorigo and his colleague proposed ant algorithm as multi agent approach to solve difficult combinatorial optimization problem
- There is currently a lot of ongoing activity in the scientific community to extend/apply ant-based algorithms to many different discrete optimization problems
- This experiment can provide a global optimal solution to a given complex problem structure such as local search, image mapping and compression, database search etc.
- Ant algorithms were first tested and validated on the Travelling Salesman Problem

## Ant Intelligent System (Importance)

- Ant Colony Intelligent System is being used as an intelligent tool to help researchers to solve many problems in different areas of science and technology
- Scientists, now a days, are using real ant colonies to solve many combinatorial optimization problems in different engineering applications
- Traditional, sequential, logic-based digital computing excels in many years
- Features like positive feedback, distributed computation, and constructive greedy heuristics approaches made ACIS successful with tremendous potential

## Ant Colony System

- An artificial ant colony system is a random stochastic population-based heuristic algorithm of agent.
- It simulates the natural behavior of ants, developing mechanisms of cooperation and learning,
- This enables the exploration of the positive feedback between agents as a search mechanism

## Biological Ant colony System

- Ants perform their task autonomously without central coordination
- When they act as a community, they can solve complex problems emerging in their daily lives through mutual cooperation known as swarm intelligence
- Swarm intelligence has four basic ingredients

    a) positive feedback

    b) negative feedback (saturation, exhaustion, competition)

    c) amplification of fluctuation (random walk)

    d) mutual interaction

## Biological Ant colony System(cont..)

- Ant can smell pheromones; while choosing their path, they tend to choose that paths marked by strong pheromone concentrations.
- This trail allows ants to find their way back to the food.

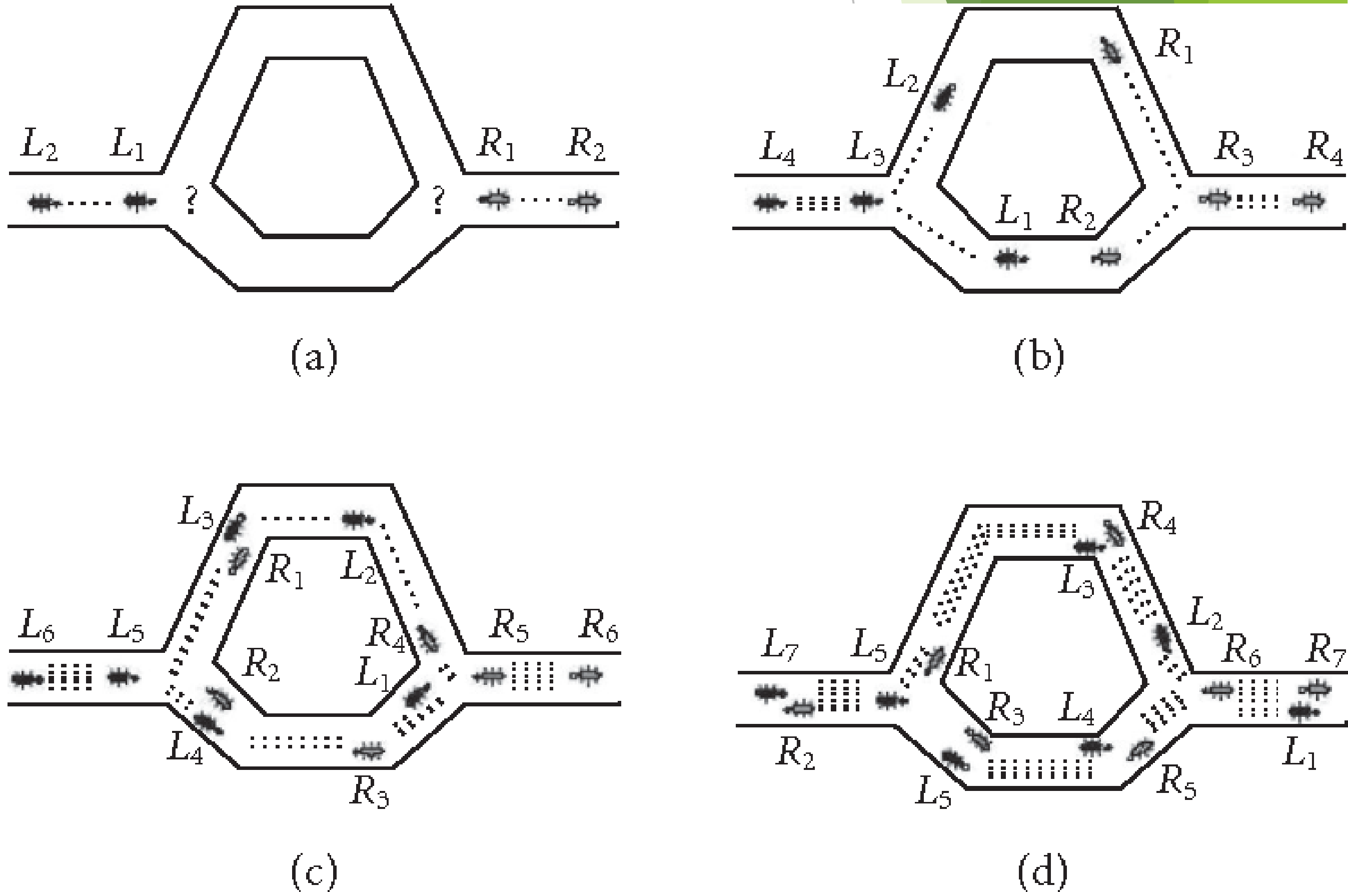

$L_2$ $L_1$ ? ? $R_1$ $R_2$
(a)
$L_2$ $R_1$ $L_4$ $L_3$ $R_3$ $R_4$ $L_1$ $R_2$
(b)
$L_3$ $R_1$ $L_2$ $L_6$ $L_5$ $R_4$ $R_5$ $R_6$ $R_2$ $L_1$ $L_4$ $R_3$
(c)
$R_4$ $L_3$ $L_2$ $L_7$ $L_5$ $R_1$ $R_6$ $R_7$ $R_3$ $L_4$ $R_2$ $L_1$ $L_5$ $R_5$
(d)

*Fig: Foraging behaviors of ants moving from their nest (origin) to the food source (destination), taking the shortest possible route through pheromone mediation [stage (a) to stage (d)]*

## Artificial Ant colony System

- A colony of cooperating individuals, an artificial pheromone trail, a sequence of local moves for finding the shortest path, a stochastic decision policy using local information
- Artificial ant lives in a discrete world and their moves consists of transitions from discrete state to discrete state
- Artificial ants have an internal state
- Artificial ants deposit a particular amount of pheromone, which is the function of the quality of the solution found

## Development of Ant Colony System

- Ant System was the first example of an Ant Colony Optimization which consists of three algorithms which was proposed by Dorigo in 1992
    - Ant Cycle
    - Ant Density
    - Ant quality
- In ant density and ant quality, ants can update pheromone trail directly after a move from one node to an adjacent node.
- In ant cycle update was carried out only after all the ants had constructed their tours and amount of pheromone deposited by each ant was set of function denoting the tour quality

## Application of Ant Colony Intelligence

- The application of Ant Colony Intelligence is distinguished in two areas:
- ***Static Combinatorial Optimization Problem:***
  - Characteristics of problem are defined and do not change till the problem is solved
  - Ex: TSP, Graph coloring, vehicle routing etc
- ***Dynamic Combinatorial Optimization Problem***
  - Some values are set by the dynamics of an underlying system
  - Problem changes on the fly so runtime and optimization algorithm must be capable of adapting on the fly to changing environment
  - Ex: Communication Networks

## Working of Ant Colony System

- Focuses on following two rules
  - Specifying how ants construct or modify a solution for the problem in hand
  - Updating pheromone trail
- Incorporates two basic activities:
  - ***Probabilistic transition rule:***

## Working of Ant Colony System(cont..)

- In a simple ACO algorithm, the main task of each artificial ant, is to find a shortest path between a pair of nodes
- Simple ant colony optimization algorithm can be used to find the solution to a shortest path problem defined on the graph G, where a solution is a path on the graph connection a source node to a destination node.
- Path length given by arc (i,j), variable t $_{ij.}$
- Amount of pheromone $t_0$ is assigned to all the arcs.
- The decision rules of an ant k located in node i use the pheromone trails t $_{ij}$ to compute the probability with which it should choose node j $\epsilon$ $N_i$ as the next node to move, where $N_i$ is the set of one-step neighbors of node i.

$$P^k_{ij} = \begin{cases} t_{ij} / \sum_{j \in N_i} t_{ij} & \text{if } j \in N_i \\ 0 & \text{if } j \notin N_i \end{cases}$$

## Working of Ant Colony System(cont..)

- While building a solution, ants deposits pheromone information on the arcs they use.
- They deposit a constant amount Δt of pheromone.
- Consider an ant that at time T moves from node i to j. It will change the pheromone value $t_{ij}$ as follows:

$$T_{ij}(T) \longleftarrow t_{ij}(T) + \Delta t$$

- Using this rule, which simulates real ants' pheromone deposits on arc (i,j), an ant using the arc connecting nose i to node j increases the probability that other ants will use the same arc in the future.

# ANT COLONY ALGORITHM

- The ant system (AS) is the earliest algorithm model proposed in the ACO algorithm.
- In the AS, the ant *k* on the node *i* calculates the probability of reaching each node, according to the probability selection formula, and based on its result $P_{ij}$ selects the next node *j*.

$$P_{ij} = \frac{\tau_{ij}^{\alpha} \eta_{ij}^{\beta}}{\sum_{h \in N_k} \tau_{ih}^{\alpha} \eta_{ih}^{\beta}}$$

*Where,*

- $N_k$ *is the* collection of all the nodes that ant *k* can visit currently,
- $\tau_{ij}$ is the amount of information along the route (*i*, *j*).
- $\eta_{ij}$ is the heuristic function, with its expression in formula, the reciprocal of the distance $d_{ij}$ between the two neighboring cities *i*, *j*.
- α is the information heuristic factor, and β is the expected heuristic factor.

*(Note: For more details please read from class note)*

## PARTICLE SWARM OPTIMIZATION

- Particle swarm optimization (PSO), originated from the analysis of behavior of birds catching food,
- It was put forward by American scholars in the early 1990s.
- American scholars Kennedy and Eberhart found, during their analysis, that the flying birds often scattered, concentrated, or changed directions in an instant, adjusting their flight – fact, which is usually unexpected.
- After summarizing the rules, they found that the flying pace of the whole flock of birds would generally keep consistent, and a proper distance was maintained between each individual bird.
- They concluded that, in the behavior rules of social animals, there has been an invisible information sharing platform for those seemingly unstructured and dispersed biological groups.
- Inspired by this, scholars simulated the behavior of birds constantly, and proposed the concept of particle swarm optimization

## PARTICLE SWARM OPTIMIZATION(cont..)

- Particle swarm optimization has become a better-developed optimization algorithm, in recent years.
- It searches the optimal solution through continuous iteration, and it finally employs the size of the value of objective function, or the function to be optimized (also known as the fitness function in the particle swarm), in order to evaluate the quality of the solution.
- The algorithm initializes the position of each particle into the solution of problems to be optimized.
- In the movement process of the particle swarm, information is conveyed between each individual influencing the others,
- Particle's moving state is influenced by the speed and direction of its colleagues, and of the whole particle swarm.
- Each particle adjusts its own speed and direction according to the historical optimal positions of itself and its colleagues, and keeps flying and searching for the optimal position – the optimal solution.
- This has proved that the particle has the memory function, and particles with good positions and directions have the tendency to approach the optimal solution.

## Procedure for Particle Swarm Optimization

**Step 1.** Initialize the parameters: initialize the position and speed of the particle to random numbers in the D-dimensional search space.

**Step 2.** Evaluate the particle's position: use a fitness function to evaluate each particle's position.

**Step 3.** Make a comparison between:

(1) compare the fitness value of step 2 with the particle's personal best value *pbest*, and make the best value become the newest *pbest*;

(2) compare the particle's fitness value with the global best value *gbest*, and the best one becomes *gbest*.

**Step 4.** Update the particle: according to the formula and, update the particle's speed and position

## Procedure for Particle Swarm Optimization

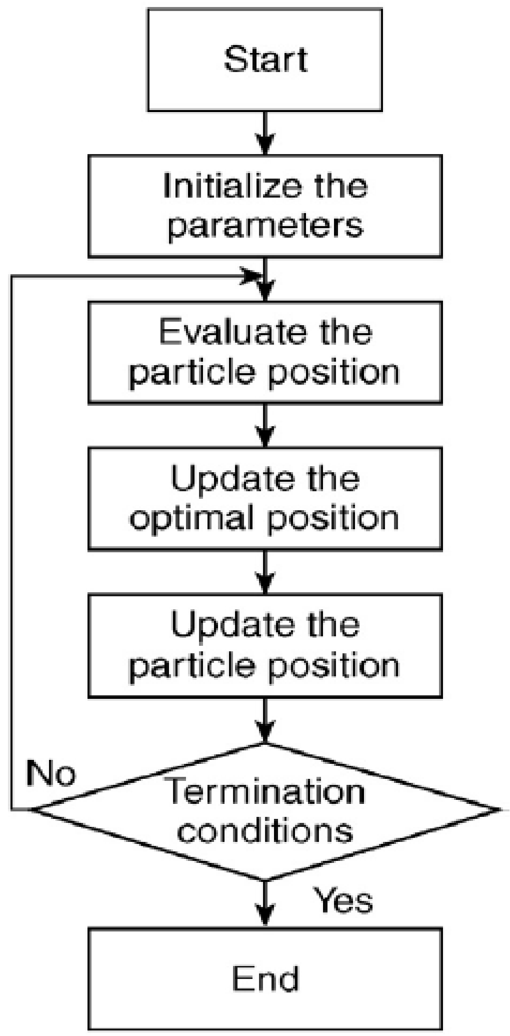

Start
Initialize the parameters
Evaluate the particle position
Update the optimal position
Update the particle position
Termination conditions
No
Yes
End

**Figure 2.5** The procedure of basic particle swarm optimization.

Thank You!!!