ATRIFICIAL INTELLIGENCE

Chapter 1: Introduction to AI

Intelligence

Intelligence is:

– The ability to reason

– The ability to understand

– The ability to create

– The ability to Learn from experience

– The ability to plan and execute complex tasks The intelligent behavior may include
– Everyday tasks: recognize a friend, recognize who is calling, translate from one language to another, interpret a photograph, talk, and cook a dinner
– Formal tasks: prove a logic theorem, geometry, calculus, play chess, checkers, or Go

– Expert tasks: engineering design, medical designers, financial analysis

Artificial Intelligence

Artificial Intelligence (AI) is usually defined as the science of making computers do things that require intelligence when done by humans. It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is the branch of computer science concerned with making computers behave like humans.

AI is the branch of computer science concerned with making computers behave like humans. In other words, AI is the science and engineering of making intelligent machines, especially intelligent computer programs. The process may include

- Learning (Gaining of information and rules for using the information)
- Reasoning (Using the rules to reach approximate or definite conclusions) -Self-Correction

According to Barr and Feigenbaum:

"Artificial Intelligence is the part of computer science concerned with designing intelligence computer systems, that is, systems that exhibit the characteristics we associate with intelligence in human behavior."

According to Elaine Rich:

"AI is the study of how to make computers do things at which, at the moment, people are better"

An AI system should have

- Capability to provide reason about something
- Capability of natural language processing
- Capability of learning past experience
- Capability of self-correction

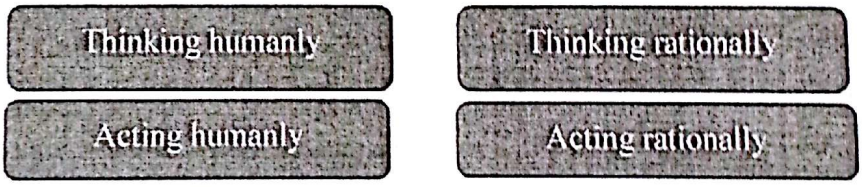
AI is thus the simulation of human intelligence processes by machines, especially computer systems. These processes include learning (the acquisition of information and rules for using the information), reasoning (using the rules to reach approximate or definite conclusions), and self-correction.
John McCarthy, an American computer scientist pioneer and inventor, was known as the father of Artificial Intelligence (AI) after playing a seminal role in defining the field devoted to the development of intelligent machines. The cognitive scientist coined the term in his 1955 proposal for the 1956 Dartmouth Conference, the first artificial intelligence conference.
Artificial intelligence includes

- Games playing: programming computers to play games such as chess and checkers.
- Expert systems: programming computers to make decisions in real-life situations (for example, some expert systems help doctors diagnose diseases based on symptoms).
- Natural language: programming computers to understand natural human languages.
- Neural networks: Systems that simulate intelligence by attempting to reproduce the types of physical connections that occur in animal brains.
- Robotics: programming computers to see and hear and react to other sensory stimuli.

The term intelligence covers many cognitive skills, including the ability to solve problems, learn, and understand language; AI addresses all of these. Intelligence therefore is the ability to comprehend; to understand and profit from experience. In other words, intelligence is the ability to adapt one's behavior to fit new circumstances. Research in AI has focused chiefly on the following components of intelligence: learning, reasoning, problem-solving, perception, and language-understanding.
In summary, AI is a branch of computer science that concerned with the study and creation of computer system that exhibit human like intelligence that

- Can learn new concept,
- Understand natural language and
- Can perform other types of tasks which require human type intelligence.

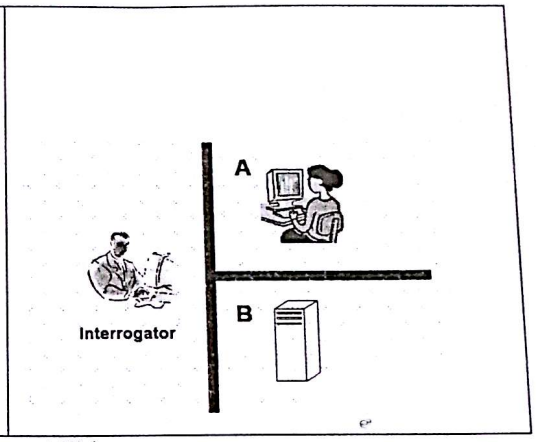
Views of AI fall into four categories



Acting humanly: *The Turing Test approach*

The Turing Test is a method for determining whether or not a computer is capable of thinking like a human. The test is named after Alan Turing, an English mathematician who pioneered artificial intelligence during the 1940s and 1950s, and who is credited with devising the original version of the test. According to this kind of test, a computer is deemed to have artificial intelligence if it can mimic human responses under specific conditions.

Consider the following setting. There are two rooms, A and B. One of the rooms contains a computer. The other contains a human. The interrogator is outside and does not know which one is a computer. He can ask questions through a teletype and receives answers from both A and B. The interrogator needs to identify whether A or B are humans. To pass the Turing test, the machine has to fool the interrogator into believing that it is human.



To pass a Turing test, a computer must have following capabilities:

- Natural Language Processing: Must be able to communicate in English successfully
- Knowledge representation: To store what it knows and hears.
- Automated reasoning: Answer the Questions based on the stored information.
- Machine learning: Must be able to adapt in new circumstances.

Thinking humanly: *The cognitive modeling approach*

- Make the machines with mind.
- **Cognition**: The action or process of acquiring knowledge and understanding through thought, experience and senses. Example: General Problem Solver(GPS)
- How do humans think?

Requires scientific theories of internal brain activities (cognitive model). Once we have precise theory of mind, it is possible to express the theory as a computer program.

- Two ways of doing this is:

Predicting and testing human behavior (cognitive science)

Identification from neurological data (Cognitive neuroscience)

Thinking rationally: *The "laws of thought approach"*

- Aristotle was one of the first who attempt to codify the right thinking that is irrefutable reasoning process. He gave Syllogisms that always yielded correct conclusion when correct premises are given.
- For example:

Ram is a man
Man is mortal
i.e. Ram is mortal

These laws of thought were supposed to govern the operation of the mind; their study initiated a field called **logic.** The logistic tradition in AI hopes to create intelligent systems using logic programming.

Acting rationally: *The rational agent approach*

- An agent is something that acts.
- Computer agent is expected to have following attributes:
    - Autonomous control
    - Perceiving their environment
    - Persisting over a prolonged period of time
    - Adapting to change
    - And capable of taking on another's goal
- Rational behavior: doing the right thing
- The right thing: that which is expected to maximize goal achievement, given the available information
- Rational Agent is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome.

In this approach the emphasis is given to correct inferences.

AI and related fields

Different fields have contributed to AI in the form of ideas, viewpoints and techniques.

Philosophy:

Logic, reasoning, mind as a physical system, foundations of learning, language and rationality.

Mathematics:

Formal representation and proof algorithms, computation, undesirability, intractability, probability.

Psychology:

Adaptation, phenomena of perception and motor control.

Economics:

Formal theory of rational decisions, game theory.

Linguistics:

Knowledge representation, grammar

Neuroscience:

Physical substrate for mental activities

Control theory:

Homeostatic systems, stability, optimal agent design

Brief History of AI

The term "Artificial Intelligence" was used for the first time in 1956 by an American scientist John McCarthy who is referred to as the Father of AI. McCarthy also come up with a programming language called LISP (i.e. List-Processing), which is still used to program computer in AI that allow the computer to learn.

Further, the major achievements can be listed as below:

1943	First electronic computer "Colossus" was developed.
1949	First commercial stored program computer was developed.
1950	- Alan Turing proposes the Turing test as a measure of machine intelligence. - Claude Shannon published a detail analysis of chess playing as search. - Isaac Asimov published his three laws of Robotics.
1951	The first working AI programs were written to run on the Ferranti Mark machine of the University of Manchester; a checkers-playing program written by Christopher Stavechey and a chess-playing program is written by Dietrich Prinz
1955	The first Dartmouth college summer AI conference is organized by John McCarthy, Marvin Minsky, Nathan Rochester of IBM and Claude Shannon.
1956	- The name artificial intelligence is used for the 1st time as the topic of the second Dartmouth Conference, organized by John McCarthy. - The first demonstration of the Logic Theorist (LT) written by Allen Newell, J.C. Shaw and Merbart Simon pus is called the first AI program
1957	The general problem Solver (GPS) demonstrated by Newell, Shaw and Simon
1958	John McCarthy at MIT invented the Lisp Programming Language.
1959	- John McCarthy and Marvin Minsky founded the MIT AI Lab. - First industrial robot company, animation was established.
1972	Prolog programming language was developed by Alain Colmerauer
1980	First National Conference of the American Association for Artificial Intelligence (AAAI) was held at Stratford.
Mid 1980's	Neural networks become widely used with the Back propagation algorithm.
1994	AI system exist in real environments with real sensory inputs (i.e. Intelligent Agents)
1997	First time AI system controlled a spacecraft named "Deep Space II"
2007	Checkers is solved by a team of researchers of the University of Alberta.
2011	AI received much public attention in February, 2011, with the Jeopardy exhibition match during which IBM's Watson soundly defeated the two greatest Jeopardy champions, Brad Rutter and Ken Jennings.
2017	Implementation of Deep learning as ML technology in Robot. The first robot citizen (citizen of Saudi Arabia) named Sophia with human like intelligence, reasoning capability.

2017 Onwards	Programmers are still trying to develop a computer which can successfully pass the "Turing Test" which have certain limitation

Importance and Application of AI

Artificial intelligence has been used in a wide range of fields including medical diagnosis, stock trading, robot control, law, remote sensing, scientific discovery and toys. Many thousands of AI applications are deeply embedded in the infrastructure of every industry. In the late 90s and early 21st century, AI technology became widely used as elements of larger systems, but the field is rarely credited for these successes.

Game Playing

Machines can play master level chess. There is some AI in them, but they well against people mainly through brute force copulation, looking at hundreds of thousands of positions.

Speech Recognition

It is possible to instruct some computers using speech. In 1990s, computer speech recognition reached a practical level for limited purposes.

Understanding Natural Language

To perform many natural language processing tasks such as machine translation, summarization, information extraction, word sense disambiguation need the AI in machine.

Computer Vision

Computer vision is concerned with the theory behind artificial system that extract information from images. The image data can take many forms such as videos sequences views from multiple cameras and data from a medical scanner. Application range from simple tasks such as industrial machine, vision system which count bottles speeding by on a production line to research into artificial intelligence and computers or robots that can comprehended the world around them.

Expert System

Expert system need the AI to perform its task. One of the first expert system was MYCIN in 1974 which diagnosis bacterial infections of the blood and suggests treatments. It did better that makes medical students practicing doctors provided to limitations were observed.

Finance

Financial institutions have long used artificial neural network systems to detect charges or claims outside of the norm, flagging these for human investigation. Use of AI in banking can be tracked back to 1987 when Security Pacific National Bank in USA set-up a Fraud Prevention Task force to counter the unauthorized use of debit cards.

Hospitals and medicine

Artificial neural networks are used as clinical decision support systems for medical diagnosis, such as in Concept Processing technology in EMR software.

Other tasks in medicine that can potentially be performed by artificial intelligence include:

- Computer-aided interpretation of medical images. Such systems help scan digital images, *e.g.* from computed tomography, for typical appearances and to highlight conspicuous sections, such as possible diseases. A typical application is the detection of a tumor.

- Heart soundanalysis
- Companion robots for the care of the elderly

Heavy industry

Robots have become common in many industries. They are often given jobs that are considered dangerous to humans. Robots have proven effective in jobs that are very repetitive which may lead to mistakes or accidents due to a lapse in concentration and other jobs which humans may find degrading. Japan is the leader in using and producing robots in the world. In 1999, 1,700,000 robots were in use worldwide.

Online and telephone customer service

Artificial intelligence is implemented in automated online assistants that can be seen as avatars on web pages. It can avail for enterprises to reduce their operation and training cost. A major underlying technology to such systems is natural language processing.

Toys and games

The 1990s saw some of the first attempts to mass-produce domestically aimed types of basic Artificial Intelligence for education, or leisure. This prospered greatly with the Digital Revolution, and helped introduce people, especially children, to a life of dealing with various types of Artificial Intelligence. AI has also been applied to video games, for example video game bots, which are designed to stand in as opponents where humans aren't available or desired **Music**

The evolution of music has always been affected by technology. With AI, scientists are trying to make the computer emulate the activities of the skillful musician. Composition, performance, music theory, sound processing are some of the major areas on which research in Music and Artificial Intelligence are focusing.

Aviation

The Air Operations Division (AOD) uses AI for the rule based expert systems. The AOD has use for artificial intelligence for replacement operators for fighting and training simulators, mission management aids, support systems for tactical decision making, and post processing of the simulator data into symbolic summaries.

KNOWLEDGE AND LEARNING

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. A representation scheme specifies the form of the knowledge. A knowledge base is the representation of all of the knowledge that is stored by an agent.

A good representation scheme is a compromise among many competing objectives. A representation should be

- Rich enough to express the knowledge needed to solve the problem.

- Willing for efficient computation
- Able to be acquired from people, data and past experiences.

Knowledge is the body of facts and principles. Knowledge can be language, concepts, procedures, rules, ideas, abstractions, places, customs, and so on. (Study of knowledge is called Epistemology)

Types of knowledge

The types of knowledge include procedural knowledge, declarative knowledge and heuristic knowledge.

- **Meta Knowledge**

It is a knowledge about a knowledge and how to gain them.

- **Procedural knowledge**

Procedural knowledge is compiled or processed form of information. Procedural knowledge is related to the performance of some task. For example, sequence of steps to solve a problem is procedural knowledge.

- **Declarative knowledge**

Declarative knowledge is passive knowledge in the form of statements of facts about the world.

For example, mark statement of a student is declarative knowledge.

- **Heuristic knowledge**

Heuristics knowledge are rules of thumb or tricks. Heuristic knowledge is used to make judgments and also to simplify solution of problems. It is acquired through experience. An expert uses his knowledge that he has gathered due to his experience and learning.

- **Structural Knowledge**

Describes what relationship exists between concepts/ objects.

Learning:

Learning is acquiring new or modifying existing knowledge, behaviors, skills, values and may involve synthesizing different types of information.

Machine learning, a branch of AI, is a scientific discipline concerned with the design and development of algorithms that allow computers to evolve behaviors based on empirical data such as from sensor data or database.

Q. Machines can be made intelligent artificially but ultimately persons make the machines. So who is more intelligent - the artificial machine or the person? Discuss

Ans: Human have done considerable work in designing a machine but the machine may not need to do very much to operate well. An example is, thermostat. It is difficult to design a thermo stat so that it turns on and off at exactly the right temperature but the thermostat itself does not have to do more computations.

All the logic behind making the machine specifies what needs to be mechanized and how to be machinated but not in vice versa. The AI reasoning in human involves all the possibilities to determine how to make a complete machine. The natural intelligence of human made the AI which may not cope on real time. Hence, the above point are supportive on the favor of the humans.

3. Search Techniques

Searching

→ Searching is the process of finding the required states or nodes.

→ Searching is to be performed through the state space.

→ Search process is carried out by constructing a search tree.

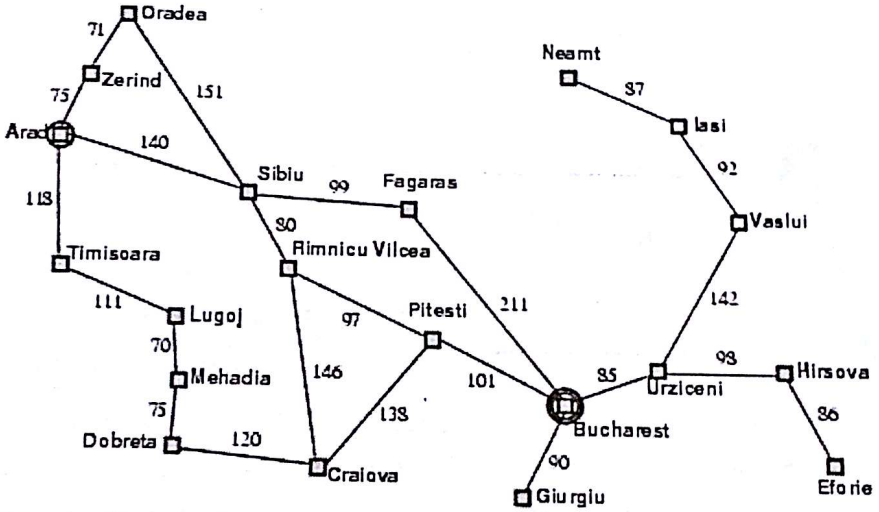
→ Search is a universal problem-solving technique.

→ Search involves systematic trial and error exploration of alternative solutions.

→ Many problems don't have a simple algorithmic solution. Casting these problems as search problems is often the easiest way of solving them.

Example: Route Planning

- State: Being in any one city
  - **Initial State:** Being in **Arad**
  - **Goal State:** Be in **Bucharest**
  - **Operators:** actions of driving from city X to city Y along the connecting roads, e.g., driving from Arad to Sibiu



State Vs. Nodes

- A state is a (representation of) a physical configuration
- Nodes in the search tree are data structures maintained by a search procedure representing *paths to a particular state*
- The same state can appear in several nodes if there is more than one path to that state
- The path can be reconstructed by following edges back to the root of the tree

Search Terminology

→ **Problem Space:** It is the environment in which the search takes place. (A set of states and set of operators to change those states)

→ **Problem Instance:** It is Initial state + Goal state

→ **Problem Space Graph:** It represents problem state. States are shown by nodes and operators are shown by edges.

→ **Depth of a problem:** Length of a shortest path or shortest sequence of operators from Initial State to goal state.

→ **Space Complexity:** The maximum number of nodes that are stored in memory.

→ **Time Complexity:** The maximum number of nodes that are created.

→ **Admissibility:** A property of an algorithm to always find an optimal solution.

→ **Branching Factor:** The average number of child nodes in the problem space graph.

→ **Depth:** Length of the shortest path from initial state to goal state.

Search strategies

→ A search strategy is defined by picking the order of node expansion

→ Strategies are evaluated along the following dimensions:

- ***Completeness***: does it generate to find a solution if there is any?
- ***Optimality***: does it always find the highest quality (least-cost) solution?
- ***Time complexity***: How long does it take to find a solution?
- ***Space complexity***: How much memory does it need to perform the search?

→ Time and space complexity are measured in terms of

- *b:* maximum branching factor of the search tree
- *d:* depth of the least-cost solution
- *m*: maximum depth of the state space (may be ∞)

The searching process in AI is broadly classified into two categories:

- **Uninformed Search or Blind Search/Brute force search**
  - Breath First Search
  - Depth First Search
  - Depth Limit Search
  - Bidirectional Search
- **Informed search or Heuristic search**
  - Hill climbing Search
  - Best first Search
  - Greedy Search
  - A* Search
  - Simulated Annealing

Uninformed search strategies

The search algorithms that do not use any extra information regarding the problem are called the blind searching or Brute force searching or uninformed searching. These can only expand current state to get a new set of states and distinguish a goal state from non-goal state. These searches typically visit all of the nodes of a tree in a certain order looking for a pre-specified goal. These type of searching are less effective than informed search.

The different types of uninformed search strategies are:

A. Breadth-First Search (BFS)

→ Proceeds level by level down the search tree

→ Starting from the root node (initial state) explores all children of the root node, left to right

→ If no solution is found, expands the first (leftmost) child of the root node, then expands the second node at depth 1 and so on …

→ Algorithm

- Place the start node in the queue
- Examine the node at the front of the queue
  - **a.** If the queue is empty, stop
  - **b.** If the node is the goal, stop
  - **c.** Otherwise, add the children of the node to the end of the queue

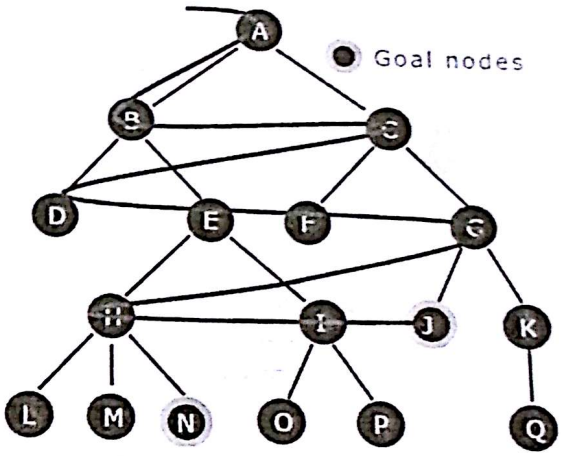


Fig. Breadth-first search (BFS)

→ **Properties**

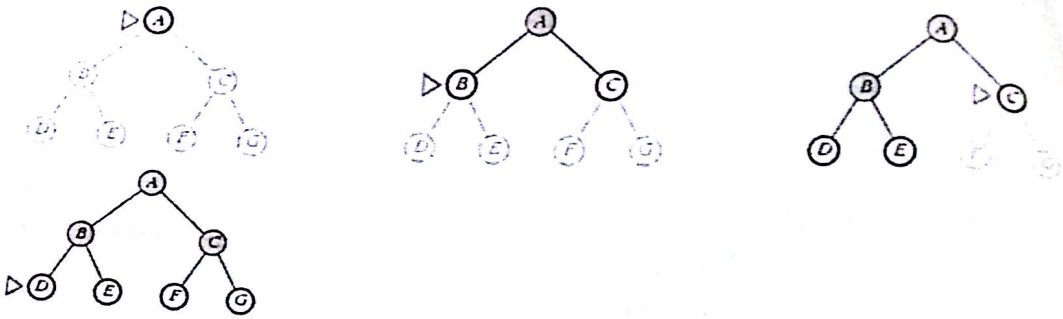
- Completeness: Complete if the goal node is at finite depth
- Optimality: It is guaranteed to find the shortest path
- b as branching factor and d as tree depth level

Time Complexity: $O(b^{d+1})$

Space Complexity: $O(b^{d+1})$

→ **BFS example (Find path from A to D)**

Put the start node in the queue, Examine the first element of the queue. If it is the goal, stop, otherwise put its children in the queue.



→ **Advantages**

- Guaranteed to find a solution (if one exists);
- Depending on the problem, can be guaranteed to find an *optimal* solution.

→ **Disadvantages**

- More complex to implement;
- Needs a lot of memory for storing the state space if the search space has a high branching factor.

B. Depth First Search (DFS)

→ DFS proceeds down a single branch of the tree at a time.

→ It expands the root node, then the leftmost child of the root node.

→ Always expands a node at the deepest level of the tree.

→ Only when the search hits a dead end (a partial solution which can't be extended), the search backtrack (retracing the steps) and expand nodes at higher levels.

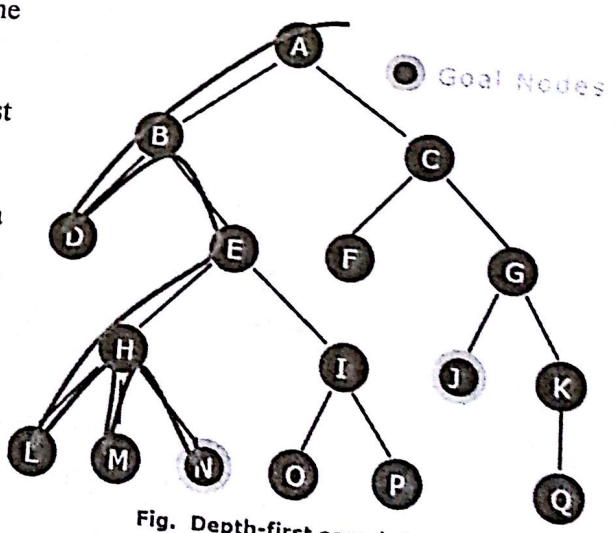


Fig. Depth-first search (DFS)

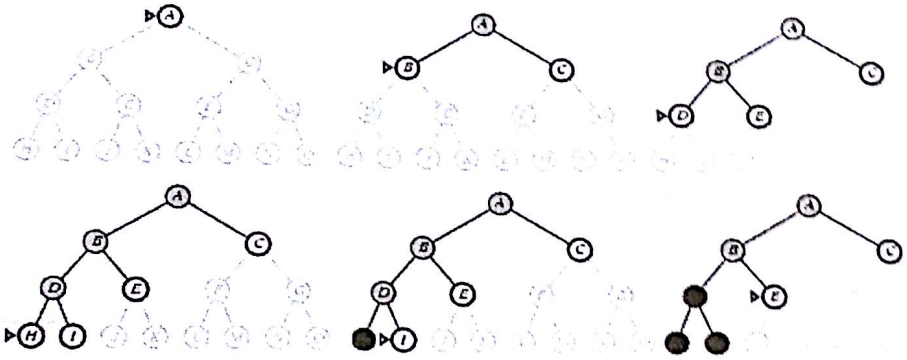
→ ***Algorithm***

- Put the start node on the stack
- While stack is not empty
  - **a.** Pop the stack
  - **b.** If the top of stack is the goal, stop
  - **c.** Other wiser push the nodes connected to the top of the stack on the stack (provided they are not already on the stack)

→ **Properties**

- ***Completeness***: Incomplete as it may get stuck down going down an infinite branch that doesn't leads to solution.
- ***Optimality***: The first solution found by the DFS may not be shortest.
- ***Space complexity***: b as branching factor and d as tree depth level, Space complexity = **O(b.d)**
- ***Time Complexity***: **$O(b^d)$**

→ **DFS example (find path from A to E)**



→ **Advantages**

- Simple to implement;
- Needs relatively small memory for storing the state-space.

→ **Disadvantages**

- Can sometimes fail to find a solution;
- Can take a lot longer to find a solution.

C. Depth Limit Search

→ Perform depth first search but only to a pre-specified depth limit L.

→ No node on a path that is more than L steps from the initial state.

→ **Advantage:** Cut off level is introduced in DFS Technique.

→ **Disadvantage:** No guarantee to find the optimal solution.

→ **Properties of depth limit search**

- ***Completeness***: Incomplete as solution may be beyond specified depth level.
- ***Optimality***: not optimal
- ***Space complexity***: b as branching factor and l as tree depth level,

  Space complexity = **O (b.L)**
- ***Time Complexity***: **$O(b^L)$**

D. *Iterative deepening depth-first search*

→ Take the idea of depth limited search one step further.

→ Starting at depth limit L = 0, we iteratively increase the depth limit, performing a depth limited search for each depth limit.

→ Stop if no solution is found, or if the depth limited search failed without cutting off any nodes because of the depth limit.

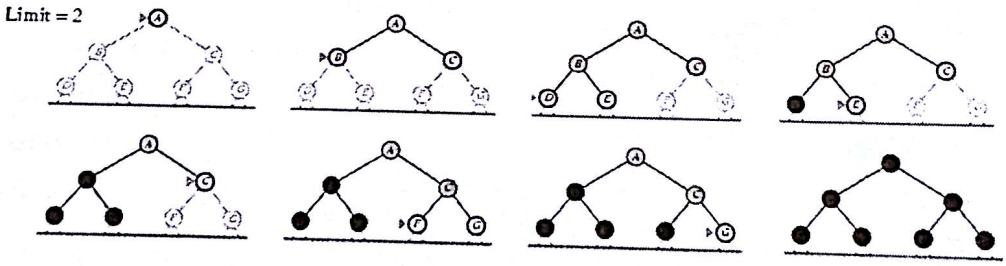
→ Iterative deepening search uses only linear space and not much more time than other uninformed algorithms

→ Search is helpful only if the solution is at given depth level.

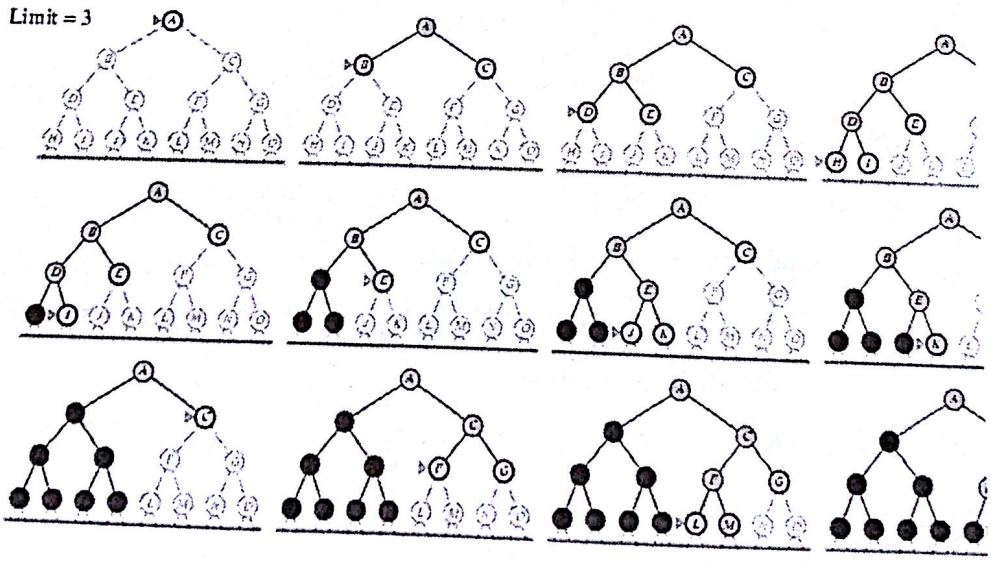
→ **An example of Iterative deepening DFS (depth level 3)**

First Iteration Search at level L=0

Second Iteration Search at level L=1



Third Iteration Search at level L=2



E. Bidirectional Search

It searches forward from initial state and backward from goal state till both meet to identify a common state.

The path from initial state is concatenated with the inverse path from the goal state. Each search is done only up to half of the total path.

Informed Search (Heuristic Search)

Informed search have problem specific knowledge apart from problem definition. They use experimental algorithm which improves efficiency of search process and the idea is to develop a domain specific heuristic function h(n) where h(n) guesses the cost of getting to the goal from node n.

→ **Heuristic Technique:** A heuristic technique, simply called heuristic, is any approach to problem solving, learning or discovery that employs a practical method not guaranteed to be optimal or perfect, but sufficient for immediate goals.

→ **Genetic Algorithm:** In the field of AI, a genetic algorithm is a search heuristic that mimics the process of natural selection. The heuristic (also called meta heuristic) is routinely used to generate useful solutions to optimize and search problems. The genetic algorithm belong to the larger class of evolutionary algorithms which generate solutions to optimization problems using techniques inspired by natural evolution such as inheritance, mutation, selection and cross over.

The different types of uninformed search strategies are:

A. Best-First Search (BFS)

Best-First search is a graph-based heuristic search algorithm. The graph is the search space that can be represented as a series of nodes connected by paths. The name "best-first" refers to the method of exploring the node with the best "score" first. An evaluation function is used to assign a score to each candidate node. The evaluation function must represent some estimate of the cost of the path from state to the closest goal state.

Judea Pearl described best-first search as estimating the promise of node n by a "heuristic evaluation function f(n) which, in general, may depend on the description of n, the description of the goal, the information gathered by the search up to that point, and most important, on any extra knowledge about the problem domain."

Algorithm

1. Put the initial node on a list START
2. If START = GOAL or START = EMPTY, then terminate search

3. Remove the first node from START and call this node-A
4. If A = GOAL, terminate the search with success
5. Else-if, node has successor and generate all of them. Find out how far they are from the GOAL node.
6. Sort all the children generated so far by remaining distance from the goal. Name the list as START-1. Replace START = START-1
7. Go to step-2

Applications

Best-first search and its more advanced variants have been used in such applications as games and web crawlers.

→ In a web crawler, each web page is treated as a node, and all the hyperlinks on the page are treated as unvisited successor nodes. A crawler that uses best-first search generally uses an evaluation function that assigns priority to links based on how closely the contents of their parent page resemble the search query.

→ In games, best-first search may be used as a path-finding algorithm for game characters. For example, it could be used by an enemy agent to find the location of the player in the game world.

→ Types

- Greedy Best First Search
- A* search

B. Greedy Best First Search

→ It tries to get as close as it can to the goal.

→ It expands the node that appears to be closest to the goal

→ It evaluates the node by using heuristic function only.

→ Evaluation function $f(n) = h(n)$

- heuristic= (estimate of cost from n to goal)
- $h(n) = 0$ for goal state

→ **Properties**

- ***Completeness:*** No – can get stuck in loops
- ***Time complexity***: $O(b^m)$, but a good heuristic can give dramatic improvement
- ***Space complexity***: $O(b^m)$, keeps all nodes in memory
- ***Optimality:*** No

→ **Advantages:**

The advantages of using a greedy algorithm is that solutions to smaller instances of the problem can be straight forward and easy to understand.

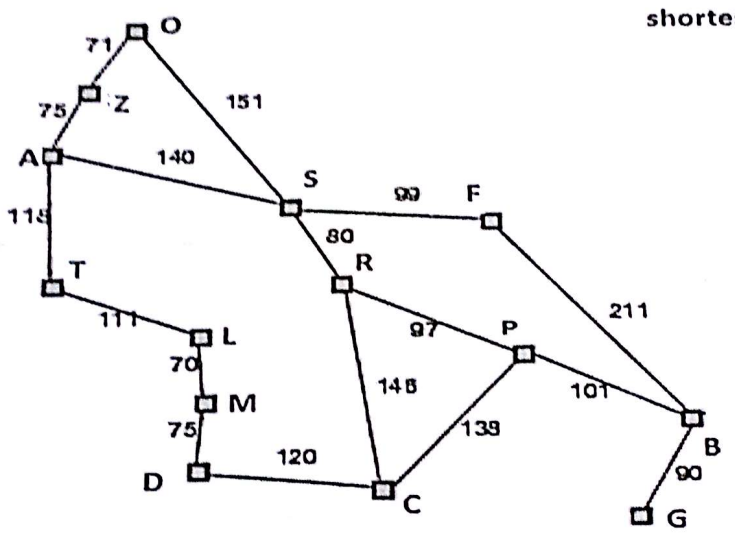
→ **Disadvantages:**

- The disadvantage is that it is entirely possible that the most optimal short-term solutions may lead to the worst long-term outcome.

→ **Applications**

This algorithm is used in Huffman encoding, minimum spanning tree, Dijkstra's algorithm etc.

→ **Given following graph of cities, starting at Arad city, problem is to reach to the Bucharest**

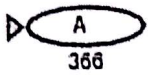


shortest line distance to B

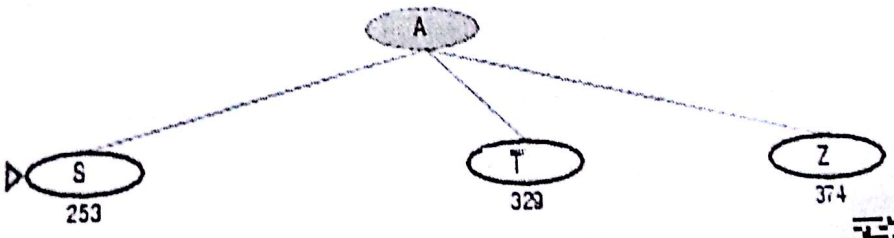
A	366
B	0
C	160
D	242
F	176
G	77
L	244
M	241
O	380
P	10
R	193
S	253
T	329
Z	374

Solution using greedy best first can be as below:

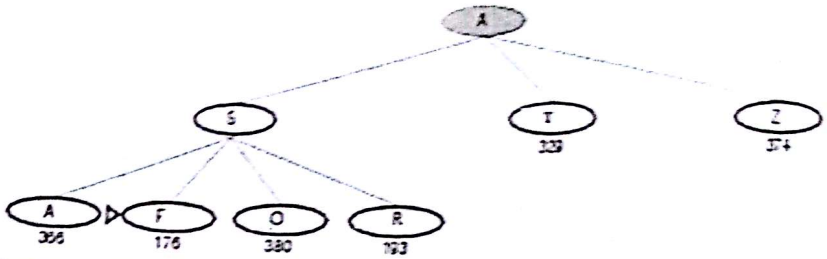
Step 1: Initial State



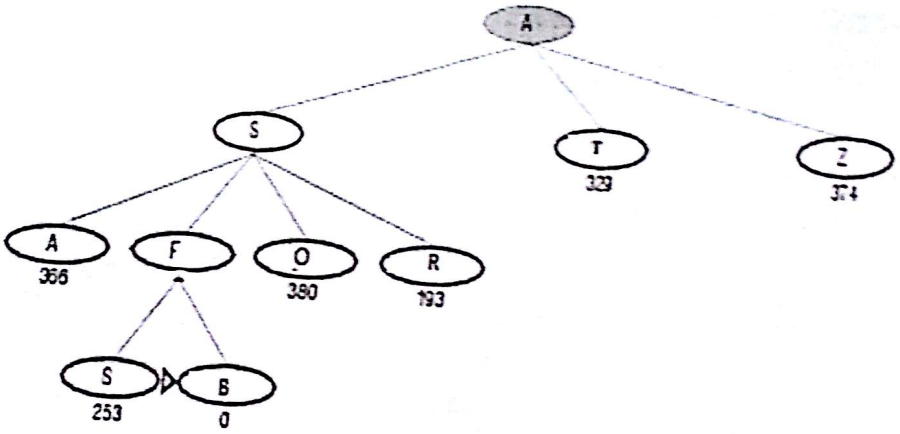
Step 2: After expanding A



Step 3: After expanding S



Step 4: After expanding F



C. *A* Search*

→ *It finds a minimal cost-path joining the start node and a goal node for node n.*

→ *Evaluation function:* $f(n) = g(n) + h(n)$

Where,

$g(n)$ = cost so far to reach n from root

$h(n)$ = estimated cost to goal from n

Thus, f(n) estimates always the lowest total cost of any solution path going through node n.

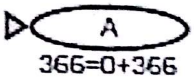
→ *Avoid expanding paths that are already expensive*

→ The main drawback of A* algorithm and indeed of any best-first search is its memory requirement.

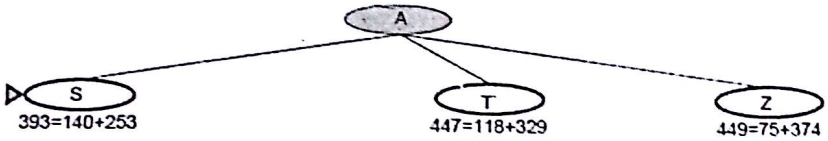
→ **A* search example (Find path from A to B)**

Here, evaluate nodes connected to source. Evaluate f(n)=g(n)+h(n) for each node. Select node with lowest f(n) value.

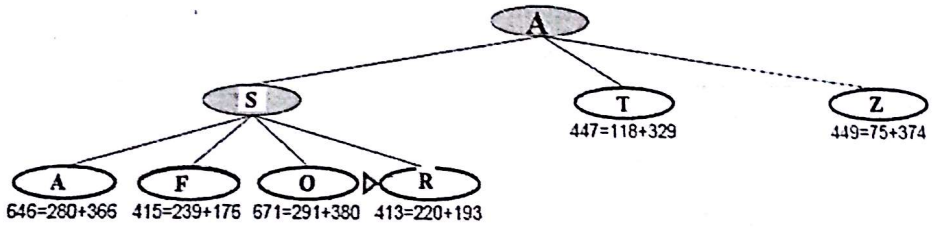
Step 1:



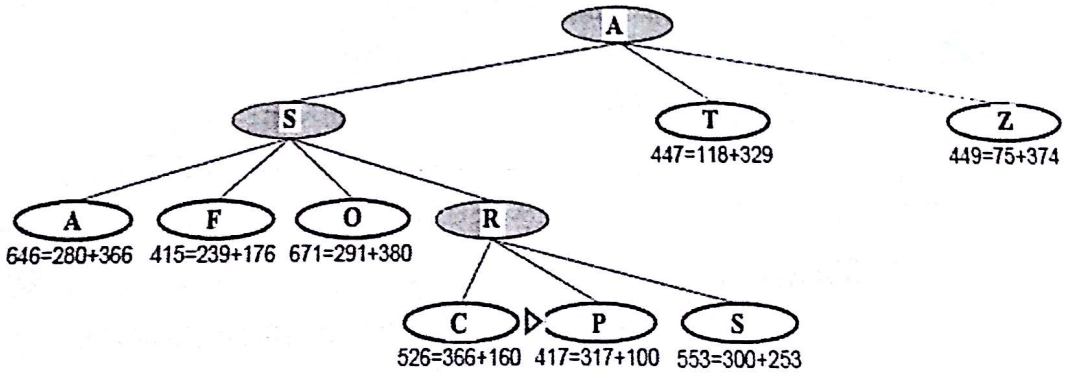
Step 2:



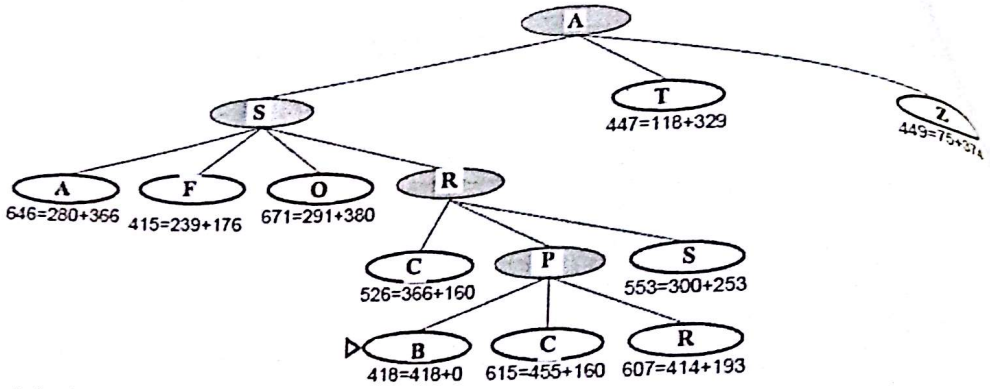
Step 3:



Step 4:



Step 5:



→ **Admissible heuristics**

- A heuristic h(n) is admissible if for every node n,

$$h(n) \leq h^*(n),$$

where $h^*(n)$ is the true cost to reach the goal state from n.

- Example: $h_{SLD}(n)$ (never overestimates the actual road distance)
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
- Theorem: If h(n) is admissible, A* using GRAPH-SEARCH is complete
- Space Complexity = $O(b^d)$
- Time Complexity = $O(b^d)$

→ **Consistent heuristics**

- A heuristic is consistent if for every node n, every successor n' of n generated by any action a,

$$h(n) \leq c(n, a, n') + h(n')$$

- If h is consistent, we have

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$

i.e., f(n) is non-decreasing along any path.

- Theorem: If h(n) is consistent, A* using GRAPH-SEARCH is optimal

Iterative Improvement Algorithms

A. Hill Climbing Search

→ *Hill climbing is an extension of depth-first search which uses some knowledge such as estimates of the distance of each node from the goal to improve the search.*

→ It is simply a loop that continually moves in the direction of increasing value—that is, uphill. It terminates when it reaches a "peak" where no neighbor has a higher value.

→ The algorithm does not maintain a search tree, so the data structure for the current node need only record the state and the value of the objective function.

→ Hill climbing does not look ahead beyond the immediate neighbors of the current state.

→ Hill climbing is sometimes called greedy local search because it grabs a good neighbor state without thinking ahead about where to go next.

→ **Algorithm**

- Push the root node on the stack
- Pop the stack & examine it
  - if the top of the stack is the goal, stop
  - if the stack is empty, stop, there is no path
  - Otherwise, sort the children of the current node and then push them on the stack

→ **Drawbacks**

- **Local maxima:** A local maximum is a peak that is higher than each of its neighboring states but lower than the global maximum. Hill-climbing algorithms that reach the vicinity of a local maximum will be drawn upward toward the peak but will then be stuck with nowhere else to go.

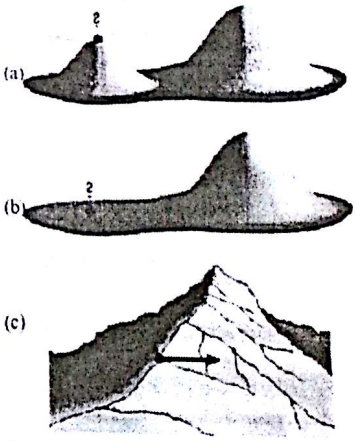


Figure 5.9 Local maxima, Plateaus and ridge situation for Hill Climbing

- **Plateaus:** A plateau is an area of the search space where evaluation function is flat, thus requiring random walk.
- **Ridges:** Where there are steep slopes and the search direction is not towards the top but towards the side.

→ **Remedies**

- **Back tracking for local maximum:** The back tracking help in undoing what is been done so far and permit to try totally different part to attain the global peak.
- **Big Jump:** A big jump is the solution to escape from plateaus because all neighbors' points have same value using the greedy approach.

- **Random restart:** Keep restarting the search from random locations until a goal is found. This will be the solution for Ridge.

B. Simulated Annealing

→ Simulated Annealing escapes local maxima by allowing some "bad" moves but gradually decrease their frequency.

→ Instead of restarting from a random point, we allow the search to take some downhill steps to try to escape local maxima.

→ Algorithm:

- A random pick is made for the move
- If it improves the situation, it is accepted straight away
- If it worsens the situation, it is accepted with some probability less than 1. i.e. for bad moves the probability is low and for comparatively less bad moves, it is higher.

→ Probability of downward steps is controlled by temperature parameter.

→ Applications of Simulated Annealing Search

- VLSI layout problem
- Factory scheduling
- Travelling salesman problem

Adversarial Search

→ Competitive environments in which the agents goals are in conflict, give rise to adversarial (oppositional) search, often known as games.

→ In AI, games means deterministic, fully observable environments in which there are two agents whose actions must alternate and in which utility values at the end of the game are always equal and opposite.

E.g. If first player wins, the other player necessarily loses.

A. Minimax Algorithm

→ It is a recursive algorithm for choosing the next move in a n-player game, usually a two player game

→ A value is associated with each position or state of the game

→ The value is computed by means of a position evaluation function and it indicates how good it would be for a player to reach the position.

→ The player then makes the move that maximizes the minimum value of the position from

the opponents possible moves called maximizing player and other player minimize the maximum value of the position called minimizing player.

→ **Properties**

- *Complete?* Yes (if tree is finite)
- *Optimal?* Yes (against an optimal opponent)
- *Time complexity?* $O(b^m)$
- *Space complexity?* $O(b^m)$ (depth-first exploration)

→ MiniMax Game Search

a. It is a Depth-first search with limited depth.

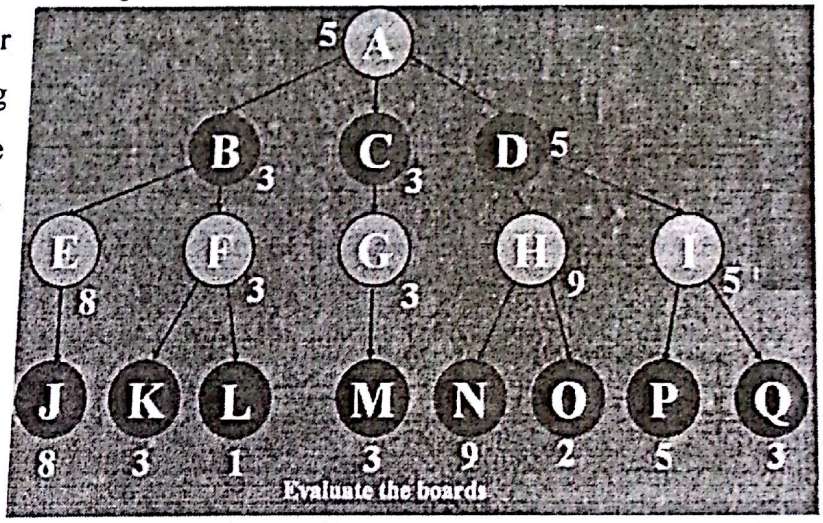
b. Assume the opponent will make the best move possible.

c. Algorithm

minimax (player,board)
if(game over in current board position)
return winner
if(max's turn)
return maximal score of calling minimax
else (min's turn)
return minimal score of calling minimax

→ **Example**

We first consider games with two players; MAX and MIN. MAX moves first, and then they take turns moving until the game is over. Each level of the tree alternates, MAX is trying to maximize her score, and MIN is trying to minimize MAX score in order to undermine her success. At the end of the game, points are awarded to the winning player and penalties are given to the loser.



B. Alpha Beta Pruning(Clipping)

→ Minimax Algorithm explores some parts of tree it doesn't have to..

→ The exact implementation of alpha-beta keeps track of the best move for each side as it moves throughout the tree.

→ Alpha–beta pruning gets its name from the following two parameters that describe bounds on the backed up values that appear anywhere along the path:

- α = the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX.
- β = the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN.

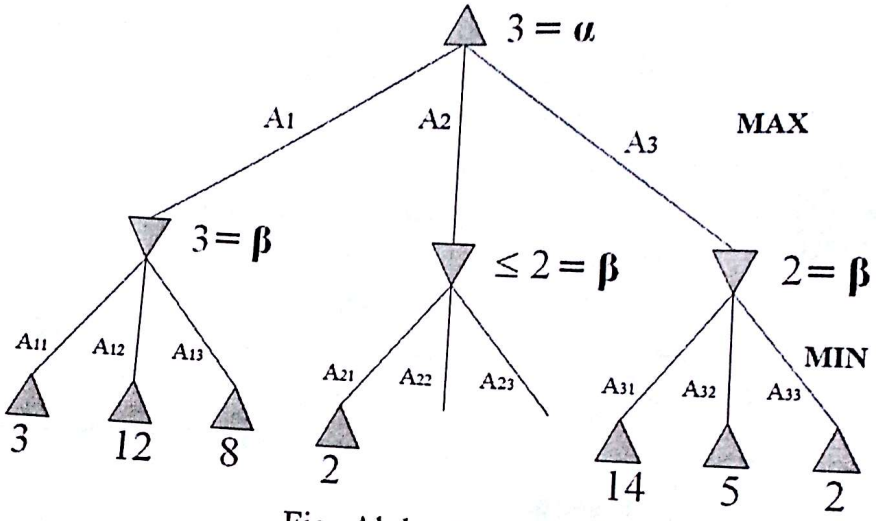
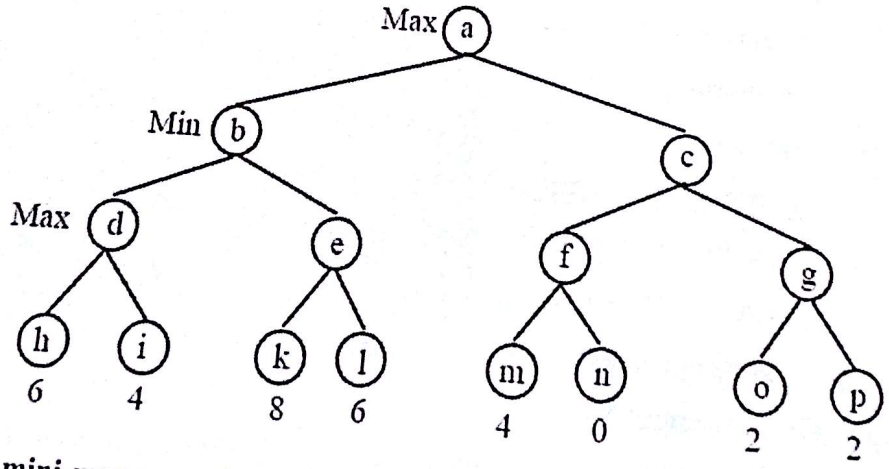


Fig. Alpha-Beta Pruning

Q. Consider the following game tree (drawn from the point of view of the Maximizing player)



a) Use the mini-max procedure and show what moves should be chosen by the two players.

Neural Networks

Humans and other animals process information with *neural networks*. These are formed from *trillions* of **neurons** (nerve cells) exchanging brief electrical pulses called **action potentials**. ***Computer algorithms that mimic these biological structures are formally called artificial neural networks.***

Artificial neural network is an interconnected group of artificial neurons that uses a mathematical model for information processing based on a connectionist approach to computation.

Artificial neural network is an adaptive system that changes its structure based on external or internal information that flows through the network. Neural network mimic certain processing of the human brain as:

- Neural computing is an information processing paradigm, inspired by biological system, composed of a large number of highly interconnected processing elements(neurons) working in unison to solve specific problems.
- ANNs, like people, learn by example.
- An ANN is configured for a specific application such as pattern recognition or data classification through a learning process.
- Learning in biological system involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well.

1. Network Structure

Many different neural network structures have been tried, some based on imitating what a biologist sees under the microscope, some based on a more mathematical analysis of the problem. The most commonly used structure is shown in Fig. 26-5. This neural network is formed in three layers,

1. **The input layer,**
2. **The hidden layer, and**
3. **Output layer**

Each layer consists of one or more **nodes**, represented in this diagram by the small circles. The lines between the nodes indicate the flow of information from one node to the next. In this particular type of neural network, the information flows only from the input to the output (that is, from left-to-right). Other types of neural networks have more complex connections, such as feedback paths.

The nodes of the input layer are **passive**, meaning they do not modify the data. They receive a single value on their input, and duplicate the value to their multiple outputs. In comparison, the nodes of the hidden and output layer are **active**. This means they modify the data as shown in Fig. 26-6.

The variables: XI_1, XI_2 ... XI_{15} hold the data to be evaluated (see Fig. 26-5). For example, they may be pixel values from an image, samples from an audio signal, stock market prices on successive days, etc. They may also be the output of some other algorithm, such as the classifiers in our cancer detection example: diameter, brightness, edge sharpness, etc.

Each value from the **input layer** is duplicated and sent to *all* of the **hidden nodes**. This is called a **fully interconnected** structure. As shown in Fig. 26-6, the values entering **a hidden node** are multiplied by **weights**, a set of predetermined numbers stored in the program. The weighted inputs are then added to produce a single number.

This is shown in the diagram by the symbol, $\sum$. Before leaving the node, this number is passed through a nonlinear mathematical function called a *sigmoid*. This is an "s" shaped curve that limits the node's output. That is, the input to the sigmoid is a value between $-\infty$ and $+\infty$, while its output can only be between 0 and 1.

The outputs from the hidden layer are represented in the flow diagram (Fig 26-5) by the variables: $X2_1$, $X2_2$, $X2_3$ and $X2_4$. Just as before, **each of these values is duplicated and applied to the next layer**. The **active nodes of the output layer combine and modify the data to produce the two output values of this network, $X3_1$ and $X3_2$.**

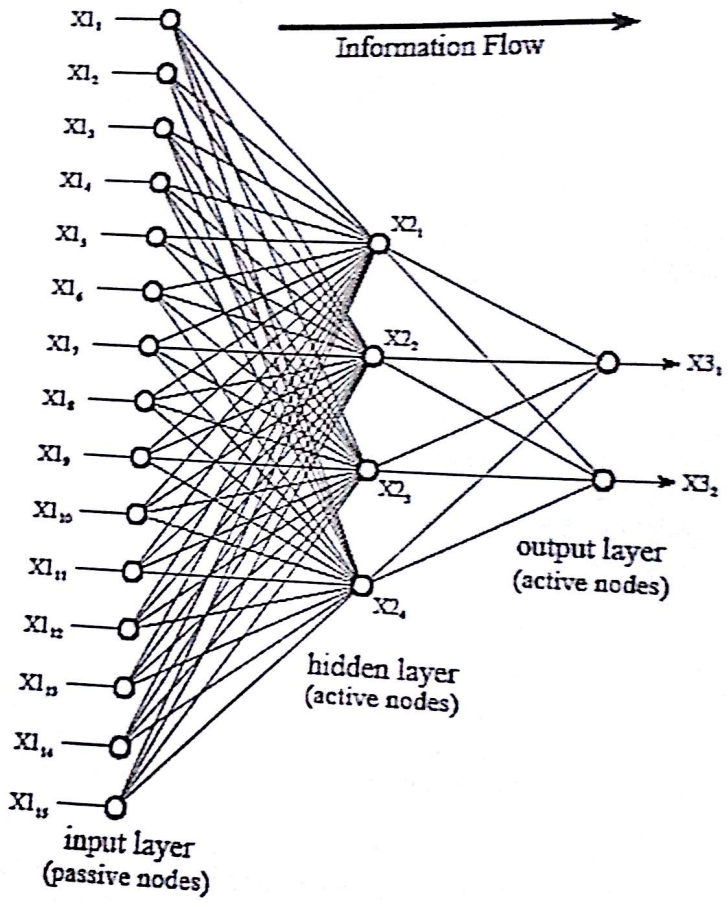


FIGURE 26-5
Neural network architecture. This is the most common structure for neural networks: three layers with full interconnection. The input layer nodes are passive, doing nothing but relaying the values from their single input to their multiple outputs. In comparison, the nodes of the hidden and output layers are active, modifying the signals in accordance with Fig. 26-6. The action of this neural network is determined by the weights applied in the hidden and output nodes.

Neural networks can have any number of layers, and any number of nodes per layer. **Most applications use the three layer structure with a maximum of a few hundred input nodes. The hidden layer is usually about 10% the size of the input layer.** In the case of target detection, the output layer only needs a single node. The output of this node is thresholded to provide a positive or negative indication of the target's presence or absence in the input data.

Example

As an example, imagine a neural network for recognizing objects in a sonar signal. Suppose that 1000 samples from the signal are stored in a computer. How does the computer determine if these data represent a submarine, whale, undersea mountain, or nothing at all? Conventional DSP would approach this problem with mathematics and algorithms, such as correlation and frequency spectrum analysis. With a neural network, the 1000 samples are simply fed into the input layer, resulting in values popping from the output layer. By selecting the proper weights, the output can be configured to report a wide range of information. For instance, there might be outputs for: submarine (yes/no), whale (yes/no), undersea mountain (yes/no), etc.

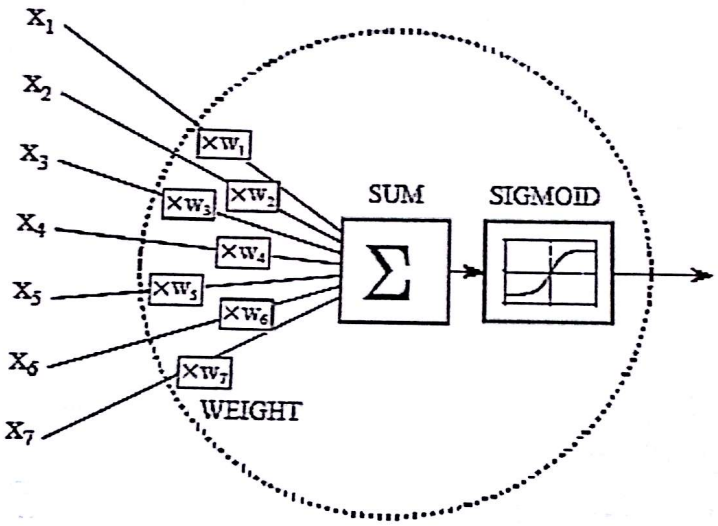


FIGURE 26-6: Neural network active node. This is a flow diagram of the active nodes used in the hidden and output layers of the neural network. Each input is multiplied by a weight (the Wn values) and then summed. This produces a single value that is passed through an "s" shaped non-linear function called a ***sigmoid***.

With other weights, the outputs might classify the objects as: metal or non-metal, biological or non-biological, enemy or ally, etc. No algorithms, no rules, no procedures; only a relationship between the input and output dictated by the values of the weights selected.

The exact shape of the sigmoid is not important, only that it is a **smooth threshold**. For comparison, a **simple threshold** produces a value of *one* when $x > 0$, and a value of *zero* when $x < 0$. The sigmoid performs this same basic threshold function, but is also *differentiable*.

Example:

The neuron shown consists of four inputs with the weights.

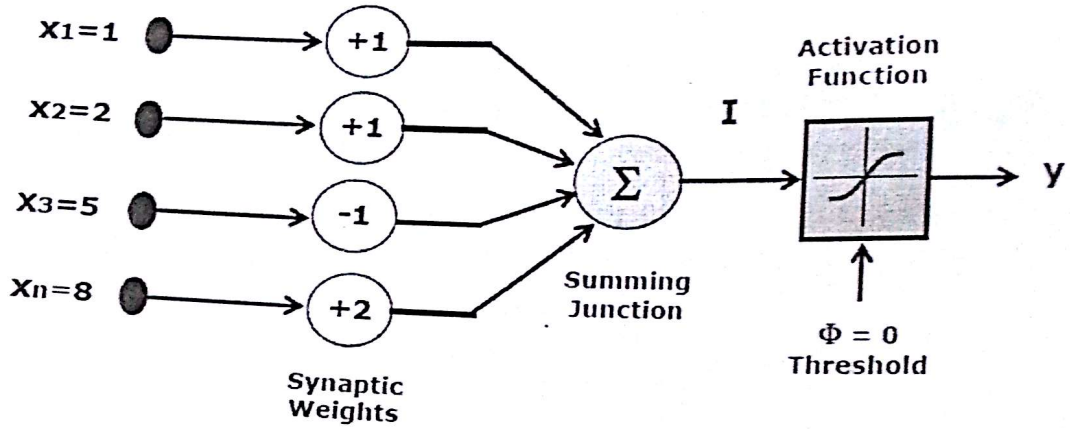


Fig Neuron Structure of Example

The output **I** of the network, prior to the activation function stage, is

$$I = X^T . W = \begin{bmatrix} 1 & 2 & 5 & 8 \end{bmatrix} \bullet \begin{pmatrix} +1 \\ +1 \\ -1 \\ +2 \end{pmatrix} = 14$$

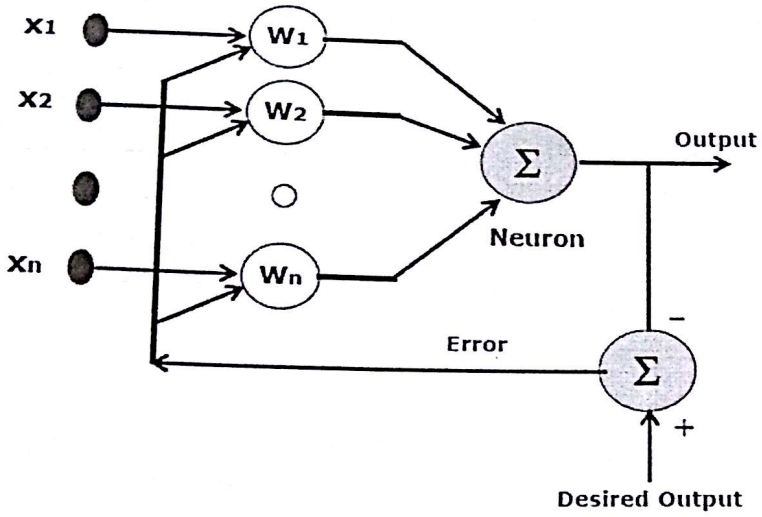
$$= (1 \times 1) + (2 \times 1) + (5 \times -1) + (8 \times 2) = 14$$

With a binary activation function the outputs of the neuron is:

y (threshold) = 1;

2. Adaline

ADALINE (Adaptive Linear Neuron or later **Adaptive Linear Element)** is a single layer neural network. It was developed by Professor Bernard Widrow and his graduate student Ted Hoff at Stanford University in 1960. It consists of a weight, a bias and a summation function.



Given the following variables:

- x is the input vector
- w is the weight vector
- n is the number of inputs
- θsome constant
- y is the output

Then we find that the output is $$y = \sum_{j=1}^{n} x_j w_j + \theta$$

The basic structure of an ADALINE is similar to a neuron with a linear activation and a feedback loop. During the training phase of ADALINE, the input vector as well as the desired output Teacher) are presented to the network.

Learning algorithm

Let us assume:

- $\eta$is the learning rate (some constant)
- d is the desired output
- o is the actual output

Then the weights are updated as follows

$$w \leftarrow w + \eta(d - o)x$$

The ADALINE converges to the least squares error which is $E = (d - o)^2$

Uses of ADALINE

In practice, an ADALINE is used to

- Make binary decisions; the output is sent through a binary threshold.
- Realizations of logic gates such as AND, NOT and OR .
- Realize only those logic functions that are linearly separable.

3. Medaline

Madaline (Multiple Adaline) is a **two layer neural network** with a set of **ADALINEs** in parallel as its input layer and a single PE (processing element) in its output layer.

For problems with multiple input variables and one output, each input is applied to one Adaline.

For similar problems with multiple outputs, madalines in parallel can be used.

The madaline network is useful for problems which involve prediction based on multiple inputs, such as **weather forecasting** (Input variables: barometric pressure, difference in pressure. Output variables: rain, cloudy, sunny).

A MADALINE (many Adaline) network is created by combining a number of Adalines. The network of ADALINES can span many layers.

4. Perceptron

The **perceptron** is a type of artificial neural network invented in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt.

It can be seen as the simplest kind of feed forward neural network: a linear classifier.

The perceptron is a binary classifier which maps its input x (a real-valued vector) to an output value $f(x)$ (a single binary value):

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

Where w is **a vector of real-valued weights**, $w \cdot x$ is the dot product (which here computes a weighted sum), and b is the 'bias', a constant term that does not depend on any input value.

The value of $f(x)$ (0 or 1) is used to classify x as either a positive or a negative instance, in the case of a binary classification problem. If b is negative, then the weighted combination of inputs must produce a positive value greater than |b|. Spatially, the bias alters the position (though not the orientation) of the decision boundary. The perceptron learning algorithm does not terminate if the learning set is not linearly separable.

Single Layer Perceptron Learning Algorithm

The training of Perceptron is a supervised learning algorithm where weights are adjusted to minimize error when ever the output does not match the desired output.

- If the output is correct then no adjustment of weights is done.

  i.e. $W_{ij}^{K+1} = W_{ij}^{K}$

- If the output is 1 but should have been 0 then the weights are decreased on the active input link

  i.e. $W_{ij}^{K+1} = W_{ij}^{K} - \alpha . x_i$

- If the output is 0 but should have been 1 then the weights are increased on the active input link

  i.e. $W_{ij}^{K+1} = W_{ij}^{K} + \alpha . x_i$

Where

W_{ij}^{K+1} is the new adjusted weight, W_{ij}^{K} is the old weight

x_i is the input and α is the learning rate parameter.

α small leads to slow and α large leads to fast learning.

Here, bias is not considered in the learning algorithm of the perceptron.

Algorithm:

i. Initialize weights and threshold.
ii. Present input and desired output
iii. Calculate the actual output
iv. Adapts weights

Steps iii and iv are repeated until the iteration error is less than a user-specified error threshold or a predetermined number of iterations have been completed.

Algorithm (in detail)

- **Step 1 :**

  Create a peceptron with $(n+1)$ input neurons $x_0, x_1, \ldots, x_n$, where $x_0 = 1$ is the bias input.

  Let O be the output neuron.

- **Step 2 :**

  Initialize weight $W = (w_0, w_1, \ldots, w_n)$ to random weights.

- **Step 3 :**

  Iterate through the input patterns X_j of the training set using the weight set; ie compute the weighted sum of inputs $\text{net } j = \sum_{i=1}^{n} x_i w_i$ for each input pattern j.

- **Step 4 :**

  Compute the output y_j using the step function

$$y_j = f(\text{net}_j) = \begin{cases} 1 & \text{if } \text{net}_j \geq 0 \\ 0 & \text{if } \text{net}_j < 0 \end{cases} \quad \text{where} \quad \text{net}_j = \sum_{i=1}^{n} x_i w_{ij}$$

- **Step 5 :**

  Compare the computed output y_j with the target output y_j for each input pattern j.

  If all the input patterns have been classified correctly, then output (read) the weights and exit.

- **Step 6 :**

  Otherwise, update the weights as given below :

  If the computed outputs y_j is 1 but should have been 0,

  Then $w_i = w_i - \alpha x_i$, $i = 0, 1, 2, \ldots, n$

  If the computed outputs y_j is 0 but should have been 1,

  Then $w_i = w_i + \alpha x_i$, $i = 0, 1, 2, \ldots, n$

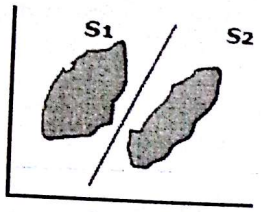
  where α is the learning parameter and is constant.

- **Step 7 :**

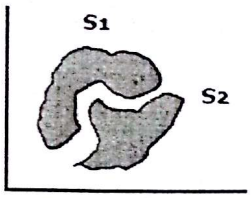
  goto step 3

- **END**

Note Perceptron cannot handle tasks that are not linearly separable.



(a) Linearly separable patterns



(b) Not Linearly separable patterns

Note : Perceptron cannot find weights for classification problems that are not linearly separable.

5. Multilayer Perceptron

Single Layer Perceptron

Definition : An arrangement of one input layer of neurons feed forward to one output layer of neurons is known as Single Layer Perceptron.

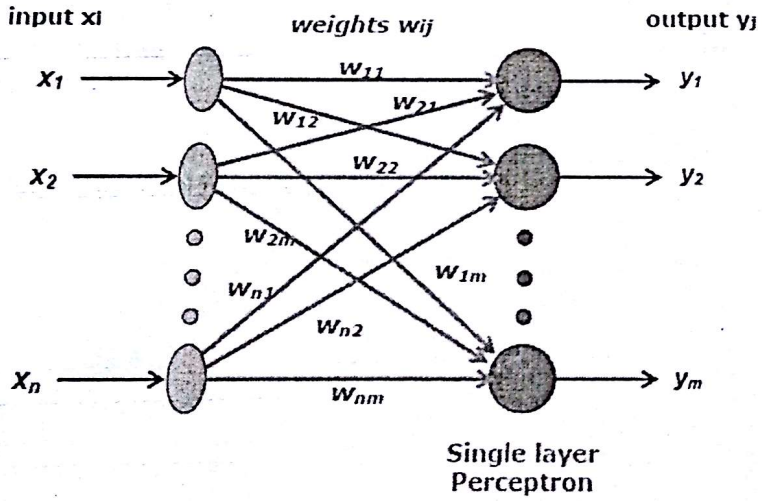


Fig. Simple Perceptron Model

$$y_j = f(net_j) = \begin{cases} 1 & \text{if } net_j \geq 0 \\ 0 & \text{if } net_j < 0 \end{cases} \qquad \text{where } net_j = \sum_{i=1}^{n} x_i \, w_{ij}$$

Multilayer Perceptron

The *multilayer perceptron* (MLP) is a hierarchical structure of several perceptrons, and overcomes the shortcomings of these single-layer networks.

The *multilayer perceptron* is an artificial neural network that learns nonlinear function mappings. The multilayer perceptron is capable of learning a rich variety of nonlineardecision surfaces.

Nonlinear functions can be represented by multilayer perceptrons with units that use nonlinear activation functions. Multiple layers of cascaded linear units still produce only linear mappings.

A neural network with one or more layers of nodes between the input and the output nodes is called *multilayer network.*

The multilayer *network structure*, or *architecture*, or *topology*, consists of an input layer, two or more hidden layers, and one output layer. The input nodes pass values to the first hidden layer, its nodes to the second and so on till producing outputs.

A network with a layer of input units, a layer of hidden units and a layer of output units is a *two-layer network.* A network with two layers of hidden units is a *three-layer network*, and so on. A justification for this is that the layer of input units is used only as an input channel and can therefore be discounted.

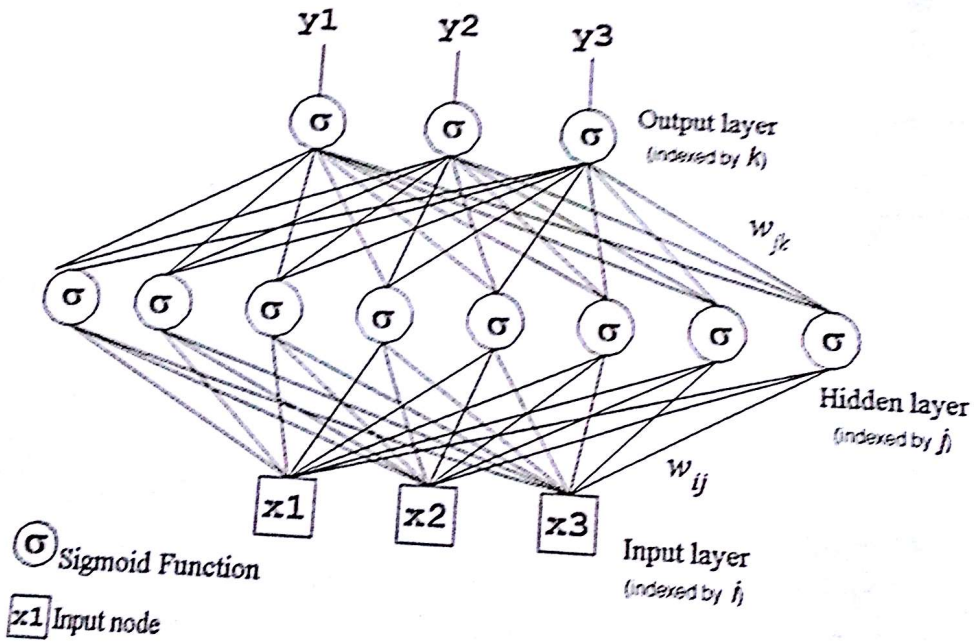


Figure 1. Multilayer Perceptron (MLP)

The algorithm for Perceptron Learning is based on the back-propagation rule discussed previously. The multilayer network MLP has a *highly connected topology* since every input is connected to all

nodes in the first hidden layer, every unit in the hidden layers is connected to all nodes in the next layer, and so on.

The input signals, initially these are the input examples; propagate through the neural network in a forward direction on a layer-by-layer basis that is why they are often called *feed forward multilayer networks.*

Two kinds of signals pass through these networks:

- *Function signals*: the input examples propagated through the hidden units and processed by their activation functions emerge as outputs.

- *Error signals*: the errors at the output nodes are propagated backward layer-by-layer through the network so that each node returns its error back to the nodes in the previous hidden layer.

6. Radial Bias Function

A **radial basis function (RBF)** is a real-valued function whose value depends only on the distance from the origin, so that $\phi(\mathbf{x}) = \phi(\|\mathbf{x}\|)$; or alternatively on the distance from some other point *c*, called a *center*, so that $\phi(\mathbf{x}, \mathbf{c}) = \phi(\|\mathbf{x} - \mathbf{c}\|)$.
Any function ϕ that satisfies the property $\phi(\mathbf{x}) = \phi(\|\mathbf{x}\|)$ is a radial function.

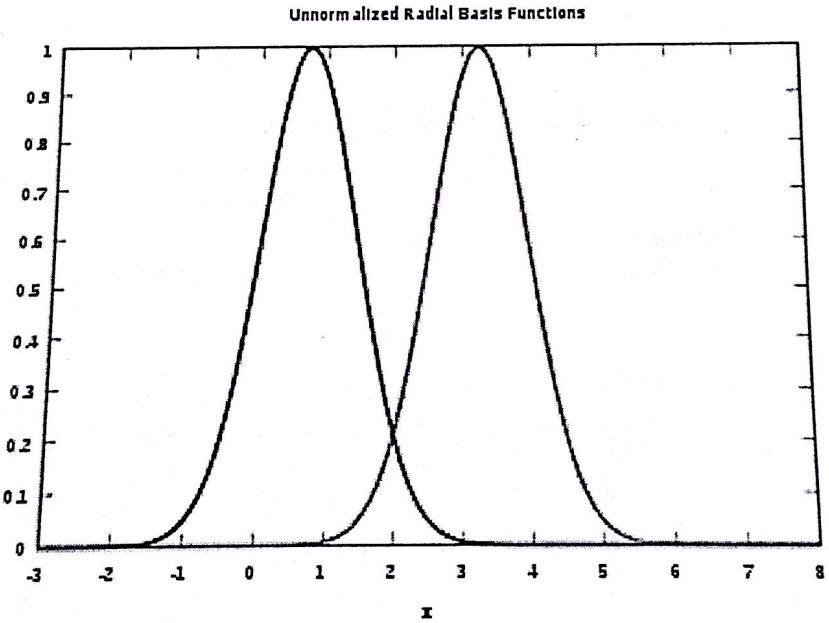


Figure: Two un-normalized Gaussian radial basis functions in one input dimension. The basis function centers are located at x_1=0.75 and x_2=3.25.

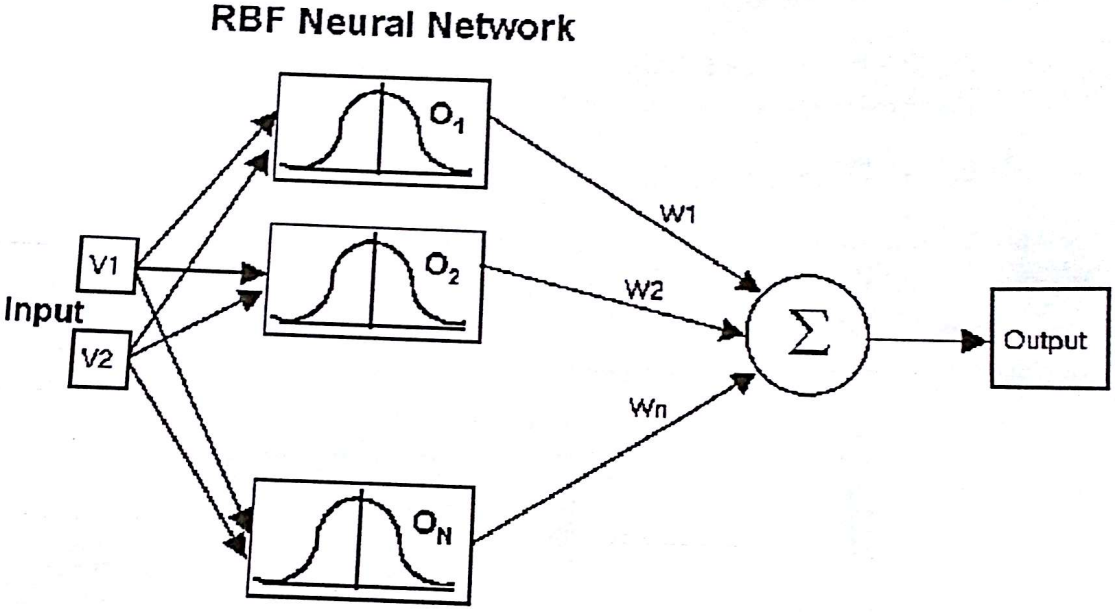
Radius Basis Function Network

The sum

$$y(\mathbf{x}) = \sum_{i=1}^{N} w_i \, \phi(\|\mathbf{x} - \mathbf{x}_i\|),$$

can also be interpreted as a rather simple single-layer type of **artificial neural network** called a **radial basis function network,** with the radial basis functions taking on the role of the activation functions of the network.

RBF Network Architecture



RBF networks have three layers:

1. **Input layer**

There is one neuron in the input layer for each predictor variable. In the case of categorical variables, N-1 neurons are used where N is the number of categories. The input neurons (or processing before the input layer) standardize the range of the values by subtracting the median and dividing by the inter-quartile range. The input neurons then feed the values to each of the neurons in the hidden layer.

2. **Hidden layer**

This layer has a variable number of neurons (the optimal number is determined by the training process). **Each neuron consists of a radial basis function centered on a point with as many dimensions as there are predictor variables.** The spread (radius) of the RBF function may be different for each dimension. The centers and spreads are determined by the

training process. When presented with the x vector of input values from the input layer, a hidden neuron computes the Euclidean distance of the test case from the neuron's center point and then applies the RBF kernel function to this distance using the spread values. The resulting value is passed to the summation layer.

3. **Summation layer**

The value coming out of a neuron in the hidden layer is multiplied by a weight associated with the neuron (W1, W2, ..., Wn in this figure) and passed to the summation which adds up the weighted values and presents this sum as the output of the network.

Not shown in this figure is a bias value of 1.0 that is multiplied by a weight W0 and fed into the summation layer. For classification problems, there is one output (and a separate set of weights and summation unit) for each target category. The value output for a category is the probability that the case being evaluated has that category.

7. Hopfield Network

A **Hopfield network** is a form of **recurrent artificial neural network** invented by **John Hopfield.**

Hopfield network can be seen as **content-addressable memory or associative** and can be used for different pattern recognition problems.

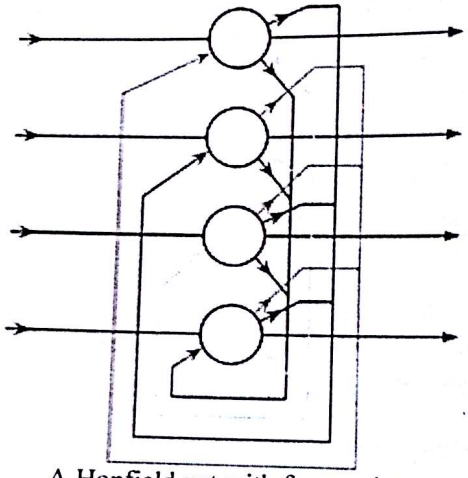
Hopfield networks provide a model for understanding human memory.

The Hopfield net is **a fully connected, symmetrically weighted network where each node functions both as input and output node.**
The idea is that, depending on the weights, some states are unstable and the net will iterate a number of times to settle in a stable state. The net is initialized to have a stable state with some known patterns. Then, the function of the network is to receive a noisy or unclassified pattern as input and produce the known, learnt pattern as output.

Properties of the Hopfield network

- A recurrent network with all nodes connected to all other nodes
- Nodes have binary outputs (either 0,1 or -1,1)
- Weights between the nodes are **symmetric** $W_{ij} = W_{ji}$
- No connection from a node to **itself** is allowed
- Nodes are updated **asynchronously** (i.e. nodes are selected at random)
- The network has no **"hidden" nodes or layer**

Structure



A Hopfield net with four nodes

The units in Hopfield nets are binary threshold units, i.e. the units only take on two different values for their states and the value is determined by whether or not the units' input exceeds their threshold.

Hopfield nets can either have units that take on values of 1 or -1, or units that take on values of 1 or 0. So, the two possible definitions for unit i's activation, a_i, are:

(1) $$a_i \leftarrow \begin{cases} 1 & \text{if } \sum_j w_{ij} s_j > \theta_i, \\ -1 & \text{otherwise.} \end{cases}$$

(2) $$a_i \leftarrow \begin{cases} 1 & \text{if } \sum_j w_{ij} s_j > \theta_i, \\ 0 & \text{otherwise.} \end{cases}$$

Where, w_{ij} is the strength of the connection weight from unit j to unit i (the weight of the connection), s_j is the state of unit j and θ_i is the threshold of unit i.

The connections in a Hopfield net typically have the following restrictions:

- $w_{ii} = 0, \forall i$ (no unit has a connection with itself)
- $w_{ij} = w_{ji}, \forall i, j$ (connections are symmetric)

Activation algorithm

A node i is chosen at random for updating. Every node has a fixed threshold U_i and the nodes output or current state s_i is set by first calculating the netinput to the node which is the sum of all the input connection multiplied with their weights:

$$\text{netinput}_i = \sum_{j \neq i} W_{ij} s_j$$

And then the state is set to 0 or 1 according to this simple rule:

$$s_i = \begin{cases} 0 & \text{netinput}_i < U_i \\ 1 & \text{netinput}_i > U_i \end{cases}$$

Learning algorithm

The learning algorithm of the Hopfield network is unsupervised, meaning that there is no "teacher" telling the network what is the correct output for a certain input.

The algorithm is based on the principle of Hebbian learning which states that neurons that are active at the same time increase the weight between them, often simplified as "cells that fire together, wire together".

Training a Hopfield net involves lowering the energy of states that the net should "remember". This allows the net to serve as a **content addressable memory system,** that is to say, the network will converge to a "remembered" state if it is given only part of the state.

The net can be used to recover from a distorted input the trained state that is most similar to that input. This is called **associative memory** because it recovers memories on the basis of similarity.

For example, if we train a Hopfield net with five units so that the state (1, 0, 1, 0, 1) is an energy minimum, and we give the network the state (1, 0, 0, 0, 1) it will converge to (1, 0, 1, 0, 1).

Thus, the network is properly trained when the energy of states which the network should remember are local minima.

Example: Pattern recognition

How can we use this network to recognize patterns, where the input patterns can be noisy compared to the actual stored patterns?

Let us say we want to recognize which symbol a MxN sized image resembles to a number of stored symbols/images. Also assume that the image is black and white only. One easy way to do

this is to have one node for each pixel in the image, so that we have MxN nodes in total. A node is on, i.e. its state is equal to 1, if its pixel is black, and is off, or 0, if its pixel is white.

By first training the network with the symbols/images we want it to learn. And then setting the network in a state given by a input pattern/image which is noisy. The network will after a lot of updates in the nodes state values, according to the activation algorithm, converge to the stored symbol which the input symbol resembles the most.

Hopfield Network as Model of Associative memory

Let us say you hear a melody of a song and suddenly remember when you where on a concert hearing your favorite band playing just that song. That is associative memory. You get an input, which is a fragment of a memory you have stored in your brain, and get an output of the entire memory you have stored in your brain.

Our memories function in what is called an **associative** or **content-addressable** fashion. That is, a memory does not exist in some isolated fashion, located in a particular set of neurons. All memories are in some sense strings of memories - you remember someone in a variety of ways - by the color of their hair or eyes, the shape of their nose, their height, the sound of their voice, or perhaps by the smell of a favorite perfume. Thus memories are stored in *association* with one another. These different sensory units lie in completely separate parts of the brain, so it is clear that the memory of the person must be distributed throughout the brain in some fashion.

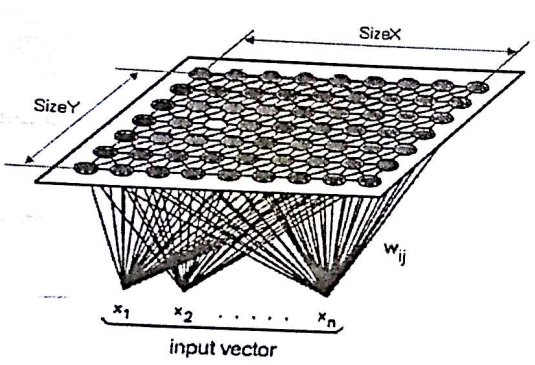
The **Hopfield neural network** is a simple artificial network which is able to store certain memories or patterns in a manner rather similar to the brain. That is; the Hopfield network explained here works in the same way. When the network is presented with an input, i.e. put in a state, the networks nodes will start to update and converge to a state which is a previously stored pattern. The learning algorithm "stores" a given pattern in the network by adjusting the weights. There is off course a limit to how many "memories" you can store correctly in the network, and empirical results show that for every pattern to be stored correctly, the number of patterns has to between 10-20% compared to the number of nodes.

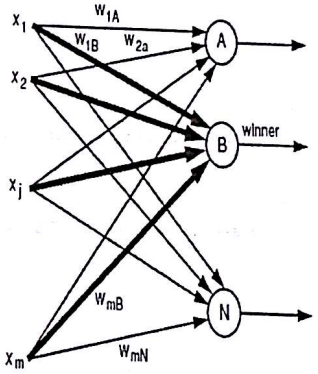
Q. The main idea of Hopfield Network is content addressable memory. Explain why it is not good for constraint satisfaction problem. Then what could be the alternative method in such case? *[2010]*

Q Explain the Hopfield network as a model of Content-Addressable Memory. [2007]

8. Kohonen Network

- A Kohonen map, or **self-organizing feature map** or **self-organizing map**, is a form of neural network invented by Kohonen in the 1980s
- The Kohonen map uses the winner-take-all-algorithm (a form of competitive learning)
  - Only one neuron provides the output of the network in response to a given input (the neuron that has the highest activation level)
  - Kohonen map is particularly useful for clustering data where the clusters are not known in advance
- Unsupervised Networks: do not require target outputs for each input vector in the training data
- Closely related to clustering
- Inputs are connected to a two-dimensional grid of neurons





- Multi-dimensional data can be mapped onto a two-dimensional surface
- A Kohonen map has two layers:
  - Input layer
  - **Cluster layer** (Output layer)
- Each input node is connected to every node in the cluster layer

Learning in Kohonen Network

- All weights are set to small random values
- Learning rate is a small positive value
- Input vector is presented to the input layer of the map.
- Input layer feeds the input data to the cluster layer.
- The neuron in the cluster layer that most closely matches the input data is declared the winner.

  - This neuron provides the output classification of the map and also has its weights updated.
- For the winner node, the weight vector is rewarded
- To determine which neuron wins,
  - Weights of output node is treated as a vector
  - This weight vector is compared with the input vector
  - The neuron whose weight vector is closest to the input vector is the winner

Strength of Kohonen Network

- Unsupervised architecture
- Requires no target output vectors
- Simply organises itself into the best representation for the data used in training

Kohonen Neural Network (In Detail)

The Kohonen neural network works differently than the feed forward neural network. The Kohonen neural network contains only an input and output layer of neurons. There is no hidden layer in a Kohonen neural network. It is **unsupervised** learning network.

The input to a Kohonen neural network is given to the neural network using the input neurons. These input neurons are each given the floating point numbers that make up the input pattern to the network. A Kohonen neural network requires that these inputs be normalized to the range between -1 and 1. Presenting an input pattern to the network will cause a reaction from the output neurons.

The output of a Kohonen neural network is very different from the output of a feed forward neural network. If we had a neural network with five output neurons we would be given an output that consisted of five values. This is not the case with the Kohonen neural network. In a Kohonen neural network only one of the output neurons actually produces a value. Additionally, this single value is either true or false.

When the pattern is presented to the Kohonen neural network, one single output neuron is chosen as the output neuron. Therefore, the output from the Kohonen neural network is usually the index of the neuron (i.e. Neuron #5) that fired.

The structure of a typical Kohonen neural network is shown in below Figure.

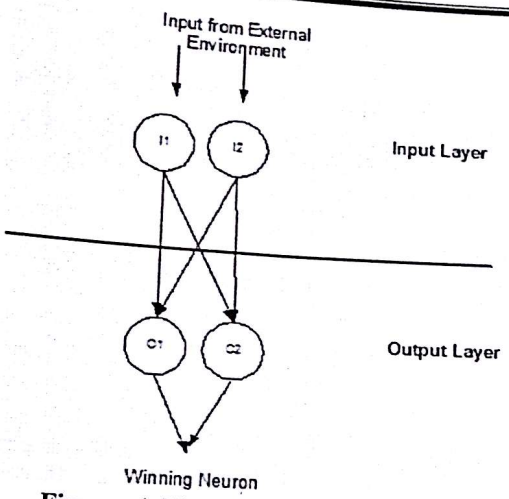


Figure: A Kohonen Neural Network

Now that you understand the structure of the Kohonen neural network we will examine how the network processes information. To examine this process we will step through the calculation process. For this example we will consider a very simple Kohonen neural network. This network will have only two input and two output neurons. The input given to the two input neurons is shown in Table 6.1.

Table 1: Sample Inputs to a Kohonen Neural Network

Input Neuron 1 (I1)	0.5
Input Neuron 2 (I2)	0.75

We must also know the connection weights between the neurons. These connection weights are given in Table 6.2.

Table 2: Connection weights in the sample Kohonen neural network

I1->O1	0.1
I2->O1	0.2
I1->O2	0.3
I2->O2	0.4

Using these values we will now examine which neuron would win and produce output. We will begin by normalizing the input.

Normalizing the Input

The Kohonen neural network requires that its input be normalized. Because of this some texts refer to the normalization as a third layer. For the purposes of this book the Kohonen neural network is considered a two layer network because there are only two actual neuron layers at work in the Kohonen neural network.

The requirements that the Kohonen neural network places on its input data are one of the most severe limitations of the Kohonen neural network. Input to the Kohonen neural network should be between the values -1 and 1. In addition, each of the inputs should fully use the range. If one, or more, of the input neurons were to use only the numbers between 0 and 1, the performance of the neural network would suffer.

To normalize the input we must first calculate the "vector length" of the input data, or vector. This is done by summing the squares of the input vector. In this case it would be.

$$(0.5 * 0.5) + (0.75 * 0.75) = 0.8125$$

This would result in a "vector length" of 0.8125. If the length becomes too small, say less than the length is set to that same arbitrarily small value. In this case the "vector length" is a sufficiently large number. Using this length we can now determine the normalization factor. The normalization factor is the reciprocal of the square root of the length. For our value the normalization factor is calculated as follows.

$$\frac{1}{\sqrt{0.8125}}$$

This results in a normalization factor of 1.1094. This normalization process will be used in the next step where the output layer is calculated.

Calculating Each Neuron's Output

To calculate the output the input vector and neuron connection weights must both be considered. First the "dot product" of the input neurons and their connection weights must be calculated. The result of this is as follows.

$$|0.5 \quad 0.75| \bullet |0.1 \quad 0.2| = (0.5*0.75)+(0.1*0.2) = 0.395$$

This calculation will be performed for the first output neuron. This calculation will have to be done for each of the output neurons. Through this example we will only examine the calculations for the first output neuron. The calculations necessary for the second output neuron are calculated in the same way.

This output must now be normalized by multiplying it by the normalization factor that was determined in the previous step. You must now multiply the dot product of 0.395 by the normalization factor of 1.1094. This results in an output of 0.438213. Now that the output has been calculated and normalized it must be mapped to a bipolar number.

Mapping to Bipolar

In the bipolar system the binary zero maps to -1 and the binary remains a 1. Because the input to the neural network normalized to this range we must perform a similar normalization to the output of the neurons. To make this mapping we add one and divide the result in half. For the output of 0.438213 this would result in a final output of 0.7191065.

The value 0.7191065 is the output of the first neuron. This value will be compared with the outputs of the other neuron. By comparing these values we can determine a "winning" neuron.

Choosing the Winner

We have seen how to calculate the value for the first output neuron. If we are to determine a winning output neuron we must also calculate the value for the second output neuron. We will now quickly review the process to calculate the second neuron. For a more detailed description you should refer to the previous section.

The second output neuron will use exactly the same normalization factor as was used to calculate the first output neuron. As you recall from the previous section the normalization factor is 1.1094. If we apply the dot product for the weights of the second output neuron and the input vector we get a value of 0.45. This value is multiplied by the normalization factor of 1.1094 to give the value of 0.0465948. We can now calculate the final output for neuron 2 by converting the output of 0.0465948 to bipolar yields 0.49923.

As you can see we now have an output value for each of the neurons. **The first neuron has an output value of 0.7191065 and the second neuron has an output value of 0.49923. To choose the winning neuron we choose the output that has the largest output value.** In this case the winning neuron is the first output neuron with an output of 0.7191065, which beats neuron two's output of 0.49923.

9. Elastic net model

Elastic Net Algorithm

The elastic net algorithm is an iterative procedure where M points, with M larger than the number of cities N, are lying on a circular ring or "rubber band" originally located at the center of the cities. The rubber band is gradually elongated until it passes sufficiently near each city to

define a tour. During that process two forces apply: one for minimizing the length of the ring, and the other one for minimizing the distance between the cities and the points on the ring. These forces are gradually adjusted as the procedure evolves.

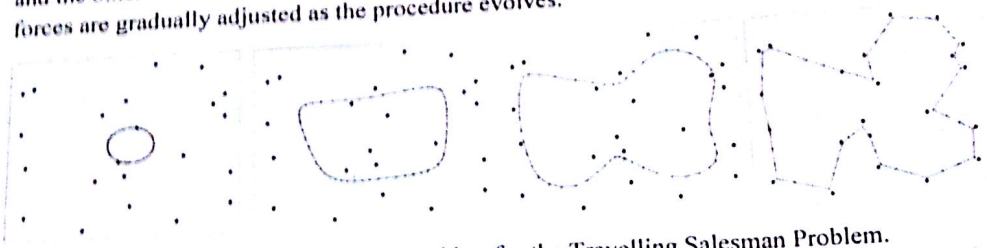


Figure: An Elastic Net Algorithm for the Travelling Salesman Problem.

10. Back –propagation

Back-propagation is a common method of training artificial neural networks so as to minimize the objective function. Arthur E. Bryson and Yu-Chi Ho described it as a multi-stage dynamic system optimization method in 1969.

It is a supervised learning method, and is a generalization of the delta rule. It requires a dataset of the desired output for many inputs, making up the training set.

It is most useful for feed-forward networks (networks that have no feedback, or simply, that have no connections that loop).

Back-propagation requires that the activation function used by the artificial neurons (or "nodes") be differentiable.

Understanding Back Propagation

When a learning pattern is clamped (fixed), the activation values are propagated to the output units, and the actual network output is compared with the desired output values, we usually end up with an error in each of the output units. Let's call this error eo for a particular output unit o. We have to bring eo to zero. The simplest method to do this is the greedy method: we strive to change the connections in the neural network in such a way that, next time around, the error eo will be zero for this particular pattern. We know from the delta rule that, in order to reduce an error, we have to adapt its incoming weights according to.

$$\Delta w_{ho} = (d_o - y_o) y_h.$$

That's step one. But it alone is not enough: when we only apply this rule, the weights from input to hidden units are never changed, and we do not have the full representational power of the feed-forward network as promised by the universal approximation theorem. In order to adapt the weights from input to hidden units, we again want to apply the delta rule. In this case, however,

Refer to class lectures for detailed example and explanation (notes)