OCW
of
Soft
Computing
Paper Code: CS 702A

Paper Name: Soft Computing
Code: CS 702A
Contacts: 3L
Credits: 3
Allotted hours: 38L
Prerequisite:

1. A solid background in mathematical and programming Knowledge

Course Objective(s)

- To learn the basics of Soft Computing usage.
- To learn the basics of many optimization algorithm
- To learn to solve and optimize the real world problem using soft computing methodology.

Course Outcomes

- **CS702A.1** To acquire the knowledge of soft computing and hard computing
- **CS702A.2** To develop skill in soft computing methodology
- **CS702A.3** To understand the concept and techniques of designing and implementing of soft computing methods in real world problem
- **CS702A.4** To acquire the knowledge of the fuzzy Neural network and Genetic Language
- **CS702A.5** To analyze and optimized the problem of real-life applications

Soft Computing: Module-I [7L]

1. An Overview of Artificial Intelligence
2. Soft Computing: Introduction of soft computing, soft computing vs. hard computing, various types of soft computing techniques, applications of soft computing.
3. Artificial Intelligence : Introduction, Various types of production systems, characteristics of production systems, breadth first search, depth first search techniques, other Search Techniques like hill Climbing, Best first Search, A* algorithm, AO* Algorithms and various types of control

Soft Computing: Module-II [10L]
1. Introduction to derivative free optimization, GA; biological background, search space of genetic algorithm, genetic algorithm Vs. Traditional algorithm; Simple genetic algorithm, Genetic algorithm Operators: Encoding, selection criteria, Crossover, Mutation, advantages and disadvantages of genetic algorithm, Convergence of GA, Applications & advances in GA, Differences & similarities between GA & other traditional method.

Soft Computing: Module-III [12L]
1. Neural Network : Biological neuron, artificial neuron, definition of ANN, Taxonomy of neural net, Difference between ANN and human brain, Structure and Function of a single neuron, characteristics and applications of ANN, single layer network, Perceptron training algorithm, Linear separability, Widrow & Hebb;s learning rule/Delta rule, ADALINE, MADALINE, AI v/s ANN.
2. Introduction of MLP, different activation functions, Error back propagation algorithm, derivation of EBPA, momentum, heuristic, limitation, characteristics and application of EBPA.
3. Adaptive Resonance Theory: Architecture, classifications, Implementation and training, Associative Memory.

Soft Computing: Module-IV [11L]
1. Fuzzy Logic: Fuzzy set theory, Fuzzy set versus crisp set, Crisp relation & fuzzy relations, Fuzzy systems: crisp logic, fuzzy logic, introduction & features of membership functions,
2. Fuzzy rule base system: fuzzy propositions, formation, decomposition & aggregation of fuzzy rules, fuzzy reasoning, fuzzy inference systems, fuzzy decision making & Applications of fuzzy logic.

Text Books:
1. Fuzzy logic with engineering applications, Timothy J. Ross, John Wiley and Sons.
2. Neural Networks: A Comprehensive Foundation (2nd Edition), Simon Haykin, Prentice Hall.

Reference Books:
1. K.H.Lee. First Course on Fuzzy Theory and Applications, Springer-Verlag.
2. J. Yen and R. Langari.. Fuzzy Logic, Intelligence, Control and Information, Pearson Education.

CO-PO Mapping

CO	PO1	PO2	POP3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CS702A.1	2											
CS702A.2			2									
CS702A.3	1		3									
CS702A.4	2											
CS702A.5		1				3						

Soft Computing: Module-I

An Overview of Artificial Intelligence

Since the invention of computers or machines, their capability to perform various tasks went on growing exponentially. Humans have developed the power of computer systems in terms of their diverse working domains, their increasing speed, and reducing size with respect to time.

A branch of Computer Science named *Artificial Intelligence* pursues creating the computers or machines as intelligent as human beings.

What is Artificial Intelligence?

According to the father of Artificial Intelligence, John McCarthy, it is *"The science and engineering of making intelligent machines, especially intelligent computer programs"*.

Artificial Intelligence is a way of making a computer, a computer-controlled robot, or a software think intelligently, in the similar manner the intelligent humans think.

AI is accomplished by studying how human brain thinks, and how humans learn, decide, and work while trying to solve a problem, and then using the outcomes of this study as a basis of developing intelligent software and systems.

Philosophy of AI

While exploiting the power of the computer systems, the curiosity of human, lead him to wonder, *"Can a machine think and behave like humans do?"*

Thus, the development of AI started with the intention of creating similar intelligence in machines that we find and regard high in humans.

Goals of AI

- **To Create Expert Systems** − The systems which exhibit intelligent behavior, learn, demonstrate, explain, and advice its users.
- **To Implement Human Intelligence in Machines** − Creating systems that understand, think, learn, and behave like humans.

What Contributes to AI?

Artificial intelligence is a science and technology based on disciplines such as Computer Science, Biology, Psychology, Linguistics, Mathematics, and Engineering. A major thrust of AI is in the development of computer functions associated with human intelligence, such as reasoning, learning, and problem solving.

Out of the following areas, one or multiple areas can contribute to build an intelligent system.
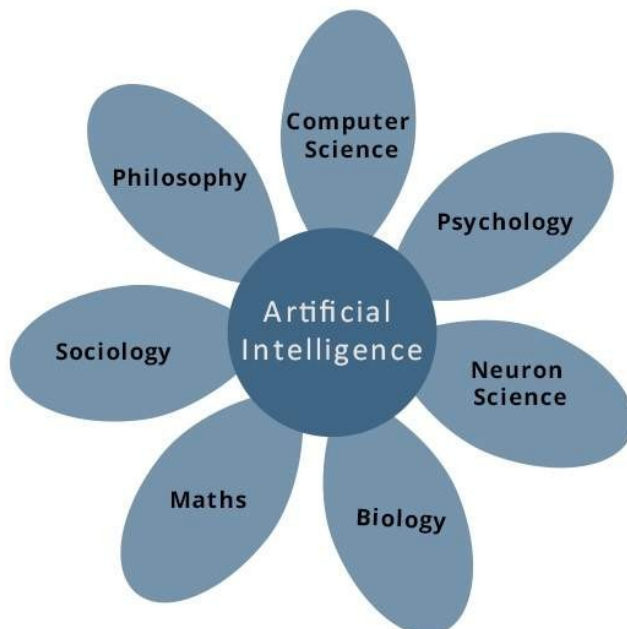
Programming Without and With AI

The programming without and with AI is different in following ways −

Programming Without AI	Programming With AI
A computer program without AI can answer the **specific** questions it is meant to solve.	A computer program with AI can answer the **generic** questions it is meant to solve.
Modification in the program leads to change in its structure.	AI programs can absorb new modifications by putting highly independent pieces of information together. Hence you can modify even a minute piece of information of program without affecting its structure.
Modification is not quick and easy. It may lead to affecting the program adversely.	Quick and Easy program modification.

What is AI Technique?

In the real world, the knowledge has some unwelcomed properties −
- Its volume is huge, next to unimaginable.
- It is not well-organized or well-formatted.
- It keeps changing constantly.

AI Technique is a manner to organize and use the knowledge efficiently in such a way that −
- It should be perceivable by the people who provide it.
- It should be easily modifiable to correct errors.
- It should be useful in many situations though it is incomplete or inaccurate.

AI techniques elevate the speed of execution of the complex program it is equipped with.

Applications of AI

AI has been dominant in various fields such as −
- **Gaming** − AI plays crucial role in strategic games such as chess, poker, tic-tac-toe, etc., where machine can think of large number of possible positions based on heuristic knowledge.
- **Natural Language Processing** − It is possible to interact with the computer that understands natural language spoken by humans.
- **Expert Systems** − There are some applications which integrate machine, software, and special information to impart reasoning and advising. They provide explanation and advice to the users.
- **Vision Systems** − These systems understand, interpret, and comprehend visual input on the computer. For example,
  - A spying aeroplane takes photographs, which are used to figure out spatial information or map of the areas.
  - Doctors use clinical expert system to diagnose the patient.
  - Police use computer software that can recognize the face of criminal with the stored portrait made by forensic artist.
- **Speech Recognition** − Some intelligent systems are capable of hearing and comprehending the language in terms of sentences and their meanings while a human talks to it. It can handle different accents, slang words, noise in the background, change in human's noise due to cold, etc.
- **Handwriting Recognition** − The handwriting recognition software reads the text written on paper by a pen or on screen by a stylus. It can recognize the shapes of the letters and convert it into editable text.
- **Intelligent Robots** − Robots are able to perform the tasks given by a human. They have sensors to detect physical data from the real world such as light, heat, temperature, movement, sound, bump, and

pressure. They have efficient processors, multiple sensors and huge memory, to exhibit intelligence. In addition, they are capable of learning from their mistakes and they can adapt to the new environment.

History of AI

Here is the history of AI during 20th century −

Year	Milestone / Innovation
1923	Karel Čapek play named "Rossum's Universal Robots" (RUR) opens in London, first use of the word "robot" in English.
1943	Foundations for neural networks laid.
1945	Isaac Asimov, a Columbia University alumni, coined the term *Robotics*.
1950	Alan Turing introduced Turing Test for evaluation of intelligence and published *Computing Machinery and Intelligence.* Claude Shannon published *Detailed Analysis of Chess Playing* as a search.
1956	John McCarthy coined the term *Artificial Intelligence*. Demonstration of the first running AI program at Carnegie Mellon University.
1958	John McCarthy invents LISP programming language for AI.
1964	Danny Bobrow's dissertation at MIT showed that computers can understand natural language well enough to solve algebra word problems correctly.
1965	Joseph Weizenbaum at MIT built *ELIZA,* an interactive problem that carries on a dialogue in English.
1969	Scientists at Stanford Research Institute Developed *Shakey*, a robot, equipped with locomotion, perception, and problem solving.
1973	The Assembly Robotics group at Edinburgh University built *Freddy*, the Famous Scottish Robot, capable of using vision to locate and assemble models.
1979	The first computer-controlled autonomous vehicle, Stanford Cart, was built.
1985	Harold Cohen created and demonstrated the drawing program, *Aaron*.
1990	Major advances in all areas of AI − • Significant demonstrations in machine learning • Case-based reasoning • Multi-agent planning • Scheduling • Data mining, Web Crawler • natural language understanding and translation • Vision, Virtual Reality • Games
1997	The Deep Blue Chess Program beats the then world chess champion, Garry Kasparov.
2000	Interactive robot pets become commercially available. MIT displays *Kismet*, a robot with a face that expresses emotions. The robot *Nomad* explores remote regions of Antarctica and locates meteorite

.Soft Computing: Introduction of soft computing,

In computer science, soft computing (sometimes referred to as computational intelligence, though CI does not have an agreed definition) is the use of inexact solutions to computationally hard tasks such as the solution of NP-complete problems, for which there is no known algorithm that can compute an exact solution in polynomial time. Soft computing differs from conventional (hard) computing in that, unlike hard computing, it is tolerant of imprecision, uncertainty, partial truth, and approximation. In effect the role model for soft computing is the human mind.

The principal constituents of Soft Computing (SC) are Fuzzy Logic (FL), Evolutionary Computation (EC), Machine Learning (ML) and Probabilistic Reasoning (PR), with the latter subsuming belief networks and parts of learning theory. Soft Computing became a formal area of study in Computer Science in the early 1990s.Earlier computational approaches could model and precisely analyze only relatively simple systems. More complex systems arising in biology, medicine, the humanities, management sciences, and similar fields often remained intractable to conventional mathematical and analytical methods. However, it should be pointed out that complexity of systems is relative and that many conventional mathematical models have been very productive in spite of their complexity.

Soft computing deals with imprecision, uncertainty, partial truth, and approximation to achieve computability, robustness and low solution cost. As such it forms the basis of a considerable amount of machine learning techniques. Recent trends tend to involve evolutionary and swarm intelligence based algorithms and bio-inspired computation.

Unlike hard computing schemes, which strive for exactness and full truth, soft computing techniques exploit the given tolerance of imprecision, partial truth, and uncertainty for a particular problem. Another common contrast comes from the observation that inductive reasoning plays a larger role in soft computing than in hard computing.

Soft computing vs. hard computing,

Definition of Hard computing
Hard computing is the traditional approach used in computing which needs an accurately stated analytical model. It was also proposed by Dr Lotfi Zadeh before soft computing. Hard computing approach produces a guaranteed, deterministic, accurate result and defines definite control actions using a mathematical model or algorithm. It deals with binary and crisp logic which require the exact input data sequentially. However, hard computing is not capable of solving the real world problems whose behaviour is extremely imprecise and where the information changes consistently.

Let's take an example if we need to find whether it will rain today or not? The answer could be yes or no, which means in two possible deterministic way we can answer the question or in other words, the answer contains a crisp or binary solution.

Key Differences Between Soft computing and Hard computing
The soft computing model is imprecision tolerant, partial truth, approximation. On the other hand, hard computing does not work on the above-given principles; it is very accurate and certain.
Soft computing employs fuzzy logic and probabilistic reasoning while hard computing is based on binary or crisp systems.
Hard computing has features such as precision and categoricity. As against, approximation and dispositionality are the characteristics of soft computing.
Soft computing approach is probabilistic in nature whereas hard computing is deterministic.
Soft computing can be easily operated on the noisy and ambiguous data. In contrast, hard computing can work only on exact input data.
Parallel computations can be performed in soft computing. On the contrary, in hard computing sequential computation is performed on the data.
Soft computing can produce approximate results while hard computing generates precise results.

BASIS FOR COMPARISON	SOFT COMPUTING	HARD COMPUTING
Basic	Tolerant to imprecision, uncertainty, partial truth and approximation.	Uses precisely stated analytical model.
Based on	Fuzzy logic and probabilistic reasoning	Binary logic and crisp system
Features	Approximation and dispositionality	Precision and categoricity
Nature	Stochastic	Deterministic
Works on	Ambiguous and noisy data	Exact input data
Computation	Can perform parallel computations	Sequential
Result	Approximate	Produces precise outcome.

Various types of soft computing techniques

Components
Components of soft computing include:

Machine learning, including:
Neural networks (NN)
Perceptron
Support Vector Machines (SVM)
Fuzzy logic (FL)
Evolutionary computation (EC), including:
Evolutionary algorithms
Genetic algorithms
Differential evolution
Metaheuristic and Swarm Intelligence
Ant colony optimization
Particle swarm optimization
Ideas about probability including:
Bayesian network
Generally speaking, soft computing techniques resemble biological processes more closely than traditional techniques, which are largely based on formal logical systems, such as sentential logic and predicate logic, or rely heavily on computer-aided numerical analysis (as in finite element analysis). Soft computing techniques are intended to complement each other.

Applications of soft computing

.

Computational techniques in computer science and some engineering disciplines, which attempt to study, model, and analyze very complex phenomena: those for which more conventional methods have not yielded low cost, analytic, and complete solutions. Earlier computational approaches could model and precisely analyze only relatively simple systems. More complex systems arising in biology, medicine, the humanities, management sciences, artificial intelligence, machine learning, and similar fields often remained intractable to conventional mathematical and analytical methods. Soft computing techniques include: fuzzy systems (FS), neural networks (NN), evolutionary computation (EC), probabilistic reasoning (PR), and other ideas (chaos theory, etc.). Soft

computing techniques often complement each other. Learn more in: Introduction and Trends to Fuzzy Logic and Fuzzy Databases

2.

Soft Computing refers to a partnership of computational techniques in computer science, artificial intelligence, machine learning and some engineering disciplines, which attempt to study, model, and analyze complex phenomena. The principle partners at this juncture are fuzzy logic, neuron-computing, probabilistic reasoning, and genetic algorithms. Thus the principle of soft computing is to exploit the tolerance for imprecision, uncertainty, and partial truth to achieve tractability, robustness, low cost solution, and better rapport with reality. Learn more in: Adaptive Neuro-Fuzzy Systems

3.

This is a term applied for defining approaches when the problem is not clear and its solution is unpredictable. Learn more in: Watermarking Using Intelligent Methods: Survey

4.

Collection of computational techniques in computer science, especially in artificial intelligence, such as fuzzy logic, neural networks, chaos theory, and evolutionary algorithms Learn more in: Harmony Search for Multiple Dam Scheduling

5.

In contrast to hard computing, soft computing is a collection of methods (fuzzy sets, rough sets, neutral nets, etc.) for dealing with ambiguous situations like imprecision and uncertainty, for example, human expressions like "high profit at reasonable risks". The objective of applying soft computing is to obtain robust solutions at reasonable costs. Learn more in: Uncertainty and Vagueness Concepts in Decision Making

6.

In contrast to "hard computing" soft computing is collection of methods (fuzzy sets, rough sets neutral nets etc.) for dealing with ambiguous situations like imprecision, uncertainty, e.g. human expressions like "high profit at reasonable risks". The objective of applying soft computing is to obtain robust solutions at reasonable costs. Learn more in: Granular Computing

7.

An older term for Computational Intelligence (see above). Learn more in: Intelligent Information Systems

8.

Collection of new computational techniques in computer science, artificial intelligence, machine learning, and many applied and engineering areas whose role model is the human mind and his guiding principle is exploit the tolerance for imprecision, uncertainty, partial truth, and approximation to achieve tractability, robustness, and low solution cost (Li, 1998; Verdegay, 2005; Zadeh, 1994)Learn more in: A Fuzzy Multi-Agent System for Combinatorial Optimization

9.

Soft computing encompasses a set of computational techniques and algorithms that are used to deal with complex systems. Soft computing exploits the given tolerance for imprecision, partial truth, and uncertainty for a particular problem. Learn more in: Soft Methods for Automatic Drug Infusion in Medical Care Environment

10.

Soft computing encompasses a set of computational techniques and algorithms that are used to deal with complex systems. Soft computing exploits the given tolerance for imprecision, partial truth, and uncertainty for a particular problem. Learn more in: Soft Methods for Automatic Drug Infusion in Medical Care Environment

11.

Is a field of computer science which is characterized by the use of inexact solutions to computationally hard tasks such as the NP-complete problems. Learn more in: Distributed Learning Algorithm Applications to the Scheduling of Wireless Sensor Networks

12.

Problem solving strategies that tolerate imprecision/uncertainty/approximation in the data, and also can handle partial information/non-exact solutions for optimization problems. Learn more in: Using Metaheuristics as Soft Computing Techniques for Efficient Optimization

13.

A set of artificial intelligence techniques provides efficient and feasible solutions in comparison with conventional computing. These techniques are also known as computational intelligence. They are basically integrated techniques to find solutions for the problems which are highly complex, ill- defined and difficult to model. Real world problems deal with imprecision and uncertainty can be easily handled using such techniques. Soft computing provides set of techniques which are hybridized and finally useful for designing intelligent systems. Learn more in: An Intelligent Process Development Using Fusion of Genetic Algorithm with Fuzzy Logic

14.
It is a term applied to a field within computer science which is characterized by the use of inexact solutions to computationally hard tasks such as the solution of NP-complete problems, for which there is no known algorithm that can compute an exact solution in polynomial time. Learn more in: Application of Soft Computing Techniques for Renewable Energy Network Design and Optimization

15.
It is the hybrid combination of algorithms that were designed to model and enable solutions to real world problems without using complex mathematical solutions. Learn more in: Churn Management of E-Banking Customers by Fuzzy AHP

16.
A partnership of techniques which in combination are tolerant of imprecision, uncertainty, partial truth, and approximation, and whose role model is the human mind. Its principal constituents are Fuzzy Logic (FL), Neural Computing (NC), Evolutionary Computation (EC) Machine Learning (ML) and Probabilistic Reasoning (PR) Learn more in: A Hybrid System for Automatic Infant Cry Recognition II

17.
A term applied to a field within computer science which is characterized by the use of inexact solutions to computationally-hard tasks. Learn more in: The Use of Soft Computing in Management

3. Artificial Intelligence : Introduction

What is Artificial Intelligence?

According to the father of Artificial Intelligence, John McCarthy, it is *"The science and engineering of making intelligent machines, especially intelligent computer programs".*

Artificial Intelligence is a way of **making a computer, a computer-controlled robot, or a software think intelligently**, in the similar manner the intelligent humans think.

AI is accomplished by studying how human brain thinks, and how humans learn, decide, and work while trying to solve a problem, and then using the outcomes of this study as a basis of developing intelligent software and systems.

Various types of production systems

Production System. Types of Production Systems.

A Knowledge representation formalism consists of collections of condition-action rules(Production Rules or Operators), a database which is modified in accordance with the rules, and a Production System Interpreter which controls the operation of the rules i.e The 'control mechanism' of a Production System, determining the order in which Production Rules are fired.
A system that uses this form of knowledge representation is called a production system.
A production system consists of rules and factors. Knowledge is encoded in a declarative from which comprises of a set of rules of the formSituation ------------ ActionSITUATION that implies ACTION.

Example:-

IF the initial state is a goal state THEN quit.
The major components of an AI production system are

i. A global database

ii. A set of production rules and

iii. A control system

The goal database is the central data structure used by an AI production system. The production system. The production rules operate on the global database. Each rule has a precondition that is either satisfied or not by the database. If the precondition is satisfied, the rule can be applied. Application of the rule changes the database. The control system chooses which applicable rule should be applied and ceases computation when a termination

condition on the database is satisfied. If several rules are to fire at the same time, the control system resolves the conflicts.

Four classes of production systems:-

1. A monotonic production system

2. A non monotonic production system

3. A partially commutative production system

4. A commutative production system.

Advantages of production systems:-

1. Production systems provide an excellent tool for structuring AI programs.

2. Production Systems are highly modular because the individual rules can be added, removed or modified independently.

3. The production rules are expressed in a natural form, so the statements contained in the knowledge base should the a recording of an expert thinking out loud.

Disadvantages of Production Systems:-

One important disadvantage is the fact that it may be very difficult analyse the flow of control within a production system because the individual rules don't call each other.

Production systems describe the operations that can be performed in a search for a solution to the problem. They can be classified as follows.

Monotonic production system :- A system in which the application of a rule never prevents the later application of another rule, that could have also been applied at the time the first rule was selected.

Partially commutative production system:-

A production system in which the application of a particular sequence of rules transforms state X into state Y, then any permutation of those rules that is allowable also transforms state x into state Y.

Theorem proving falls under monotonic partially communicative system. Blocks world and 8 puzzle problems like chemical analysis and synthesis come under monotonic, not partially commutative systems. Playing the game of bridge comes under non monotonic , not partially commutative system.

For any problem, several production systems exist. Some will be efficient than others. Though it may seem that there is no relationship between kinds of problems and kinds of production systems, in practice there is a definite relationship.

Partially commutative , monotonic production systems are useful for solving ignorable problems. These systems are important for man implementation standpoint because they can be implemented without the ability to backtrack to previous states, when it is discovered that an incorrect path was followed. Such systems increase the efficiency since it is not necessary to keep track of the changes made in the search process.

Monotonic partially commutative systems are useful for problems in which changes occur but can be reversed and in which the order of operation is not critical (ex: 8 puzzle problem).

Production systems that are not partially commutative are useful for many problems in which irreversible changes occur, such as chemical analysis. When dealing with such systems, the order in which operations are performed is very important and hence correct decisions have to be made at the first time itself.

Characteristics of production systems,

PRODUCTION SYSTEM AND ITS CHARACTERISTICS

The production system is a model of computation that can be applied to implement search algorithms and model human problem solving. Such problem solving knowledge can be packed up in the form of little quanta called productions. A production is a rule consisting of a situation recognition part and an action part. A production is a situation-action pair in which the left side is a list of things to watch for and the right side is a list of things to do so. When productions are used in deductive systems, the situation that trigger productions are specified combination of facts. The actions are restricted to being assertion of new facts deduced directly from the triggering combination. Production systems may be called premise conclusion pairs rather than situation action pair.

A production system consists of following components.

(a) A set of production rules, which are of the form A⊓B. Each rule consists of left hand side constituent that represent the current problem state and a right hand side that represent an output state. A rule is applicable if its left hand side matches with the current problem state.

> (b) A database, which contains all the appropriate information for the particular task. Some part of the database may be permanent while some part of this may pertain only to the solution of the current problem.

> (c) A control strategy that specifies order in which the rules will be compared to the database of rules and a way of resolving the conflicts that arise when several rules match simultaneously.

> (d) A rule applier, which checks the capability of rule by matching the content state with the left hand
>
> side of the rule and finds the appropriate rule from database of rules.

The important roles played by production systems include a powerful knowledge representation scheme. A production system not only represents knowledge but also action. It acts as a bridge between AI and expert systems. Production system provides a language in which the representation of expert knowledge is very natural. We can represent knowledge in a production system as a set of rules of the form

$$If \ (condition) \ THEN \ (condition)$$

along with a control system and a database. The control system serves as a rule interpreter and sequencer. The database acts as a context buffer, which records the conditions evaluated by the rules and information on which the rules act. The production rules are also known as condition – action, antecedent – consequent, pattern – action, situation – response, feedback – result pairs.
For example,
If (you have an exam tomorrow)
THEN (study the whole night)

The production system can be classified as monotonic, non-monotonic, partially commutative and commutative.
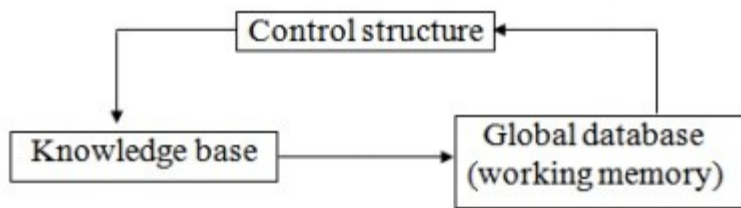
Figure Architecture of Production System

Features of Production System

Some of the main features of production system are:
Expressiveness and intuitiveness: In real world, many times situation comes like "i f this happen-you

will do that", "if this is so-then this should happ en" and many more. The production rules essentially tell

us what to do in a given situation.

1. **Simplicity:** The structure of each sentence in a production system is unique and uniform as they use "IF-THEN" structure. This structure provides simpli city in knowledge representation. This feature of production system improves the readability of production rules.

2. **Modularity:** This means production rule code the knowledge available in discrete pieces. Information can be treated as a collection of independent facts which may be added or deleted from the system with essentially no deletetious side effects.

3. **Modifiability:** This means the facility of modifying rules. It allows the development of production rules in a skeletal form first and then it is accurate to suit a specific application.
4. **Knowledge intensive:** The knowledge base of production system stores pure knowledge. This part does not contain any type of control or programming information. Each production rule is normally written as an English sentence; the problem of semantics is solved by the very structure of the representation.

Disadvantages of production system

1. **Opacity:** This problem is generated by the combination ofproduction rules. The opacity is generatedbecause of less prioritization of rules. More priority to a rule has the less opacity.
2. **Inefficiency:** During execution of a program several rules may active. A well devised control strategyreduces this problem. As the rules of the production system are large in number and they are hardly written in hierarchical manner, it requires some forms of complex search through all the production rules for each cycle of control program.

3. **Absence of learning:** Rule based production systems do not store the result of the problem for future use. Hence, it does not exhibit any type of learning capabilities. So for each time for a particular problem, some new solutions may come.

4. **Conflict resolution:** The rules in a production system should not have any type of conflict operations. When a new rule is added to a database, it should ensure that it does not have any conflicts with the existing rules.

Breadth first search and Depth First Search

Searching is the universal technique of problem solving in AI. There are some single-player games such as tile games, Sudoku, crossword, etc. The search algorithms help you to search for a particular position in such games.
Single Agent Pathfinding Problems

The games such as 3X3 eight-tile, 4X4 fifteen-tile, and 5X5 twenty four tile puzzles are single-agent-path-finding challenges. They consist of a matrix of tiles with a blank tile. The player is required to arrange the tiles by sliding a tile either vertically or horizontally into a blank space with the aim of accomplishing some objective.

The other examples of single agent pathfinding problems are Travelling Salesman Problem, Rubik's Cube, and Theorem Proving.

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.

Brute-Force Search Strategies

They are most simple, as they do not need any domain-specific knowledge. They work fine with small number of possible states.

Requirements −
- State description
- A set of valid operators
- Initial state
- Goal state description
-

Breadth-First Search

It starts from the root node, explores the neighboring nodes first and moves towards the next level neighbors. It generates one tree at a time until the solution is found. It can be implemented using FIFO queue data structure. This method provides shortest path to the solution.

If **branching factor** (average number of child nodes for a given node) = b and depth = d, then number of nodes at level $d = b^d$.

The total no of nodes created in worst case is $b + b^2 + b^3 + \ldots + b^d$.

Disadvantage − Since each level of nodes is saved for creating next one, it consumes a lot of memory space. Space requirement to store nodes is exponential.

Its complexity depends on the number of nodes. It can check duplicate nodes.
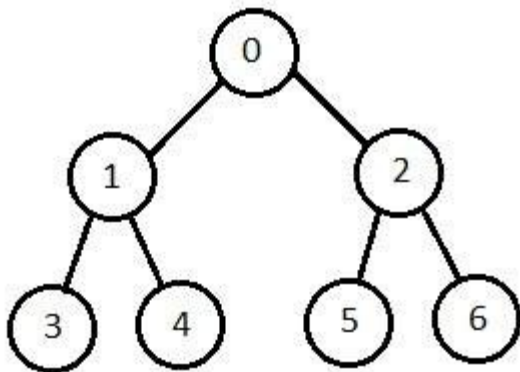
Depth-First Search

It is implemented in recursion with LIFO stack data structure. It creates the same set of nodes as Breadth-First method, only in the different order.

As the nodes on the single path are stored in each iteration from root to leaf node, the space requirement to store nodes is linear. With branching factor b and depth as m, the storage space is bm.

Disadvantage − This algorithm may not terminate and go on infinitely on one path. The solution to this issue is to choose a cut-off depth. If the ideal cut-off is d, and if chosen cut-off is lesser than d, then this algorithm may fail. If chosen cut-off is more than d, then execution time increases.

Its complexity depends on the number of paths. It cannot check duplicate nodes.
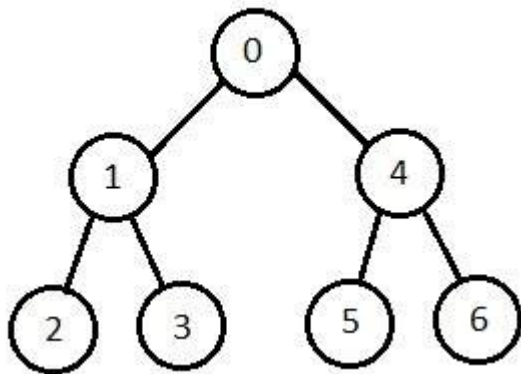
Bidirectional Search

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.

Uniform Cost Search

Sorting is done in increasing cost of the path to a node. It always expands the least cost node. It is identical to Breadth First search if each transition has the same cost.

It explores paths in the increasing order of cost.

Disadvantage − There can be multiple long paths with the cost ≤ C*. Uniform Cost search must explore them all.

Iterative Deepening Depth-First Search

It performs depth-first search to level 1, starts over, executes a complete depth-first search to level 2, and continues in such way till the solution is found.

It never creates a node until all lower nodes are generated. It only saves a stack of nodes. The algorithm ends when it finds a solution at depth d. The number of nodes created at depth d is b^d and at depth $d-1$ is b^{d-1}.
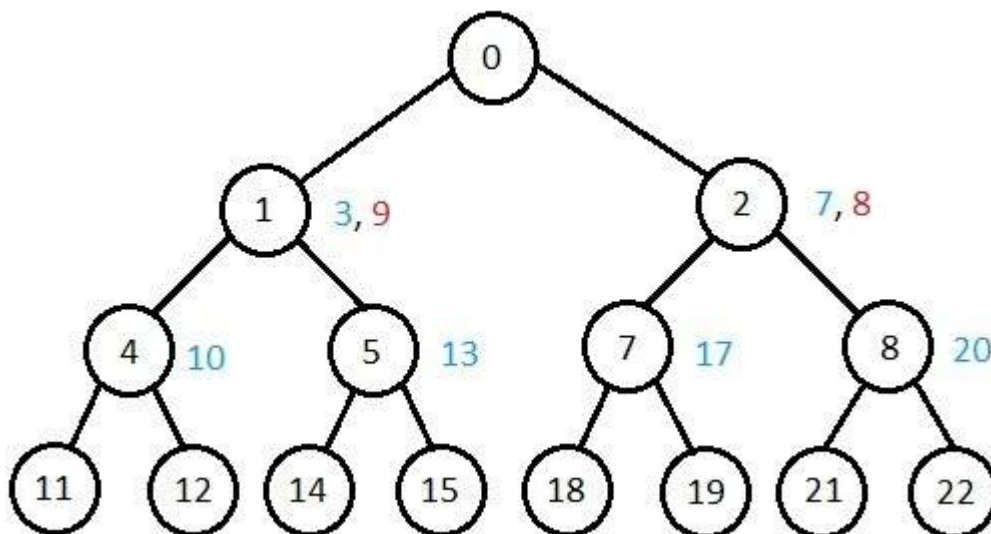
Comparison of Various Algorithms Complexities
Let us see the performance of algorithms based on various criteria −

Criterion	Breadth First	Depth First	Bidirectional	Uniform Cost	Interactive Deepening
Time	b^d	b^m	$b^{d/2}$	b^d	b^d
Space	b^d	b^m	$b^{d/2}$	b^d	b^d
Optimality	Yes	No	Yes	Yes	Yes
Completeness	Yes	No	Yes	Yes	Yes

, Best first Search, A* algorithm, AO* Algorithms and various types of control

Hill-Climbing Search

It is an iterative algorithm that starts with an arbitrary solution to a problem and attempts to find a better solution by changing a single element of the solution incrementally. If the change produces a better solution, an incremental change is taken as a new solution. This process is repeated until there are no further improvements. function Hill-Climbing (problem), returns a state that is a local maximum.

```
inputs: problem, a problem
local variables: current, a node
          neighbor, a node
current <-Make_Node(Initial-State[problem])
loop
  do neighbor <- a highest_valued successor of current
    if Value[neighbor] ≤ Value[current] then
    return State[current]
    current <- neighbor

end
```

Disadvantage − **This algorithm is neither complete, nor optimal**

Informed (Heuristic) Search Strategies
To solve large problems with large number of possible states, problem-specific knowledge needs to be added to increase the efficiency of search algorithms.
Heuristic Evaluation Functions
They calculate the cost of optimal path between two states. A heuristic function for sliding-tiles games is computed by counting number of moves that each tile makes from its goal state and adding these number of moves for all tiles.
Pure Heuristic Search
It expands nodes in the order of their heuristic values. It creates two lists, a closed list for the already expanded nodes and an open list for the created but unexpanded nodes.
In each iteration, a node with a minimum heuristic value is expanded, all its child nodes are created and placed in the closed list. Then, the heuristic function is applied to the child nodes and they are placed in the open list according to their heuristic value. The shorter paths are saved and the longer ones are disposed.

A * Search

It is best-known form of Best First search. It avoids expanding paths that are already expensive, but expands most promising paths first.

f(n) = g(n) + h(n), where
- g(n) the cost (so far) to reach the node
- h(n) estimated cost to get from the node to the goal
- f(n) estimated total cost of path through n to goal. It is implemented using priority queue by increasing f(n).

Greedy Best First Search

It expands the node that is estimated to be closest to goal. It expands nodes based on f(n) = h(n). It is implemented using priority queue.

Disadvantage − It can get stuck in loops. It is not optimal.

Local Search Algorithms

They start from a prospective solution and then move to a neighboring solution. They can return a valid solution even if it is interrupted at any time before they end.

.

Local Beam Search

In this algorithm, it holds k number of states at any given time. At the start, these states are generated randomly. The successors of these k states are computed with the help of objective function. If any of these successors is the maximum value of the objective function, then the algorithm stops.

Otherwise the (initial k states and k number of successors of the states = 2k) states are placed in a pool. The pool is then sorted numerically. The highest k states are selected as new initial states. This process continues until a maximum value is reached.

function BeamSearch(*problem*, *k*), returns a solution state.

```
start with k randomly generated states
loop
   generate all successors of all k states
   if any of the states = solution, then return the state
   else select the k best successors
end
```

Simulated Annealing

Annealing is the process of heating and cooling a metal to change its internal structure for modifying its physical properties. When the metal cools, its new structure is seized, and the metal retains its newly obtained properties. In simulated annealing process, the temperature is kept variable.

We initially set the temperature high and then allow it to 'cool' slowly as the algorithm proceeds. When the temperature is high, the algorithm is allowed to accept worse solutions with high frequency.

Start
- Initialize k = 0; L = integer number of variables;
- From i → j, search the performance difference Δ.
- If Δ <= 0 then accept else if exp(-Δ/T(k)) > random(0,1) then accept;
- Repeat steps 1 and 2 for L(k) steps.
- k = k + 1;

Repeat steps 1 through 4 till the criteria is met.

End

Travelling Salesman Problem

In this algorithm, the objective is to find a low-cost tour that starts from a city, visits all cities en-route exactly once and ends at the same starting city.

```
Start
   Find out all (n -1)! Possible solutions, where n is the total number of cities.
   Determine the minimum cost by finding out the cost of each of these (n -1)! solutions.
   Finally, keep the one with the minimum cost.
end
```
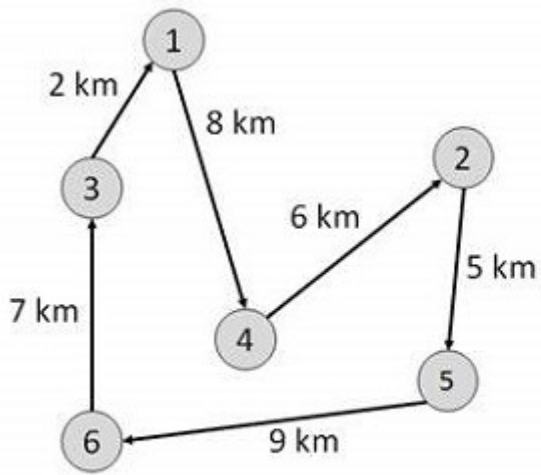
Total Distance = 37km
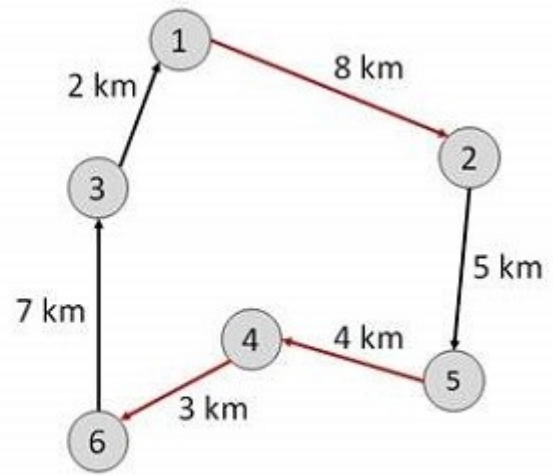
Total Distance = 31km

Soft Computing: Module-II

Introduction to derivative free optimization,

Derivative-free optimization is a discipline in mathematical optimization that does not use derivative information in the classical sense to find optimal solutions: Sometimes information about the derivative of the objective function f is unavailable, unreliable or impractical to obtain. For example, f might be non-smooth, or

time-consuming to evaluate, or in some way noisy, so that methods that rely on derivatives or approximate them via finite differences are of little use. The problem to find optimal points in such situations is referred to as derivative-free optimization, algorithms that do not use derivatives or finite differences are called derivative-free algorithms (note that this classification is not precise).[1] Derivative-free optimization is closely related to black-box optimization.[2]

Contents

Introduction

The problem to be solved is to numerically optimize an objective function {\displaystyle f\colon A\to \mathbb {R} } {\displaystyle f\colon A\to \mathbb {R} } for some set {\displaystyle A} A (usually {\displaystyle A\subset \mathbb {R} ^{n}} {\displaystyle A\subset \mathbb {R} ^{n}}), i.e. find {\displaystyle x_{0}\in A} {\displaystyle x_{0}\in A} such that without loss of generality {\displaystyle f(x_{0})\leq f(x)} {\displaystyle f(x_{0})\leq f(x)} for all {\displaystyle x\in A} x\in A.

When applicable, a common approach is to iteratively improve a parameter guess by local hill-climbing in the objective function landscape. Derivative-based algorithms use derivative information of {\displaystyle f} f to find a good search direction, since for example the gradient gives the direction of steepest ascent. Derivative-based optimization is efficient at finding local optima for continuous-domain smooth single-modal problems. However, they can have problems when e.g. {\displaystyle A} A is disconnected, or (mixed-)integer, or when {\displaystyle f} f is expensive to evaluate, or is non-smooth, or noisy, so that (numeric approximations of) derivatives do not provide useful information. A slightly different problem is when {\displaystyle f} f is multi-modal, in which case local derivative-based methods only give local optima, but might miss the global one.

In derivative-free optimization, various methods are employed to address these challenges using only function values of {\displaystyle f} f, but no derivatives. Some of these methods can be proved to discover optima, but some are rather metaheuristic since the problems are in general more difficult to solve compared to convex optimization. For these, the ambition is rather to efficiently find "good" parameter values which can be near-optimal given enough resources, but optimality guarantees can typically not be given. One should keep in mind that the challenges are diverse, so that one can usually not use one algorithm for all kinds of problems.

Algorithms

A non-exhaustive collection of derivative-free optimization algorithms follows:

Bayesian optimization
Coordinate descent and adaptive coordinate descent
Cuckoo search
Evolution strategies, Natural evolution strategies (CMA-ES, xNES, SNES)
Cross-entropy methods (CEM)
Genetic algorithms
LIPO algorithm
MCS algorithm
Nelder-Mead method
Particle swarm optimization
Pattern search
Powell's COBYLA, UOBYQA, NEWUOA, BOBYQA and LINCOA algorithms
Random search (including Luus-Jaakola)
Shuffled complex evolution algorithm
Simulated annealing
Subgradient method

GA; biological background, search space of genetic algorithm,

Genetic Algorithm (GA) is a search-based optimization technique based on the principles of **Genetics and Natural Selection**. It is frequently used to find optimal or near-optimal solutions to difficult problems which otherwise would take a lifetime to solve. It is frequently used to solve optimization problems, in research, and in machine learning.

Introduction to Optimization

Optimization is the process of **making something better**. In any process, we have a set of inputs and a set of outputs as shown in the following figure.

Optimization refers to finding the values of inputs in such a way that we get the "best" output values. The definition of "best" varies from problem to problem, but in mathematical terms, it refers to maximizing or minimizing one or more objective functions, by varying the input parameters.

The set of all possible solutions or values which the inputs can take make up the search space. In this search space, lies a point or a set of points which gives the optimal solution. The aim of optimization is to find that point or set of points in the search space.

What are Genetic Algorithms?

Nature has always been a great source of inspiration to all mankind. Genetic Algorithms (GAs) are search based algorithms based on the concepts of natural selection and genetics. GAs are a subset of a much larger branch of computation known as **Evolutionary Computation**.

GAs were developed by John Holland and his students and colleagues at the University of Michigan, most notably David E. Goldberg and has since been tried on various optimization problems with a high degree of success.

In GAs, we have a **pool or a population of possible solutions** to the given problem. These solutions then undergo recombination and mutation (like in natural genetics), producing new children, and the process is repeated over various generations. Each individual (or candidate solution) is assigned a fitness value (based on its objective function value) and the fitter individuals are given a higher chance to mate and yield more "fitter" individuals. This is in line with the Darwinian Theory of "Survival of the Fittest".

In this way we keep "evolving" better individuals or solutions over generations, till we reach a stopping criterion.

Genetic Algorithms are sufficiently randomized in nature, but they perform much better than random local search (in which we just try various random solutions, keeping track of the best so far), as they exploit historical information as well.

Advantages of GAs

GAs have various advantages which have made them immensely popular. These include −
- Does not require any derivative information (which may not be available for many real-world problems).
- Is faster and more efficient as compared to the traditional methods.
- Has very good parallel capabilities.
- Optimizes both continuous and discrete functions and also multi-objective problems.
- Provides a list of "good" solutions and not just a single solution.
- Always gets an answer to the problem, which gets better over the time.
- Useful when the search space is very large and there are a large number of parameters involved.

Limitations of GAs

Like any technique, GAs also suffer from a few limitations. These include −
- GAs are not suited for all problems, especially problems which are simple and for which derivative information is available.
- Fitness value is calculated repeatedly which might be computationally expensive for some problems.
- Being stochastic, there are no guarantees on the optimality or the quality of the solution.
- If not implemented properly, the GA may not converge to the optimal solution.

Genetic Algorithms are primarily used in optimization problems of various kinds, but they are frequently used in other application areas as well.

In this section, we list some of the areas in which Genetic Algorithms are frequently used. These are −

- **Optimization** − Genetic Algorithms are most commonly used in optimization problems wherein we have to maximize or minimize a given objective function value under a given set of constraints. The approach to solve Optimization problems has been highlighted throughout the tutorial.
- **Economics** − GAs are also used to characterize various economic models like the cobweb model, game theory equilibrium resolution, asset pricing, etc.
- **Neural Networks** − GAs are also used to train neural networks, particularly recurrent neural networks.
- **Parallelization** − GAs also have very good parallel capabilities, and prove to be very effective means in solving certain problems, and also provide a good area for research.
- **Image Processing** − GAs are used for various digital image processing (DIP) tasks as well like dense pixel matching.
- **Vehicle routing problems** − With multiple soft time windows, multiple depots and a heterogeneous fleet.
- **Scheduling applications** − GAs are used to solve various scheduling problems as well, particularly the time tabling problem.
- **Machine Learning** − as already discussed, genetics based machine learning (GBML) is a niche area in machine learning.
- **Robot Trajectory Generation** − GAs have been used to plan the path which a robot arm takes by moving from one point to another.
- **Parametric Design of Aircraft** − GAs have been used to design aircrafts by varying the parameters and evolving better solutions.
- **DNA Analysis** − GAs have been used to determine the structure of DNA using spectrometric data about the sample.
- **Multimodal Optimization** − GAs are obviously very good approaches for multimodal optimization in which we have to find multiple optimum solutions.
- **Traveling salesman problem and its applications** − GAs have been used to solve the TSP, which is a well-known combinatorial problem using novel crossover and packing strategies.

Genetic algorithm Vs. Traditional algorithm;

Difference Between Genetic Algorithm and Traditional Algorithm

Definition

Genetic algorithm is an algorithm for solving both constrained and unconstrained optimization problems that are based on Genetics and Natural Selection while traditional algorithm is an unambiguous specification that defines how to solve a problem. Thus, this is the main difference between genetic algorithm and traditional algorithm.

Usage

The specific use of each algorithm is an important difference between genetic algorithm and traditional algorithm. That is; the genetic algorithm helps to find the optimal solutions for difficult problems while traditional algorithm provides a step by step methodical procedure to solve a problem.

Complexity

Another difference between genetic algorithm and traditional algorithm is that a genetic algorithm is more advanced than a traditional algorithm.

Applications

Genetic Algorithm is used in fields such as research, Machine Learning and, Artificial Intelligence. Traditional algorithm is used in fields such as Programming, Mathematics, etc. Hence, this is also an important difference between genetic algorithm and traditional algorithm.

Simple genetic algorithm and Genetic algorithm Operators:

Basic Structure

The basic structure of a GA is as follows −

We start with an initial population (which may be generated at random or seeded by other heuristics), select parents from this population for mating. Apply crossover and mutation operators on the parents to generate new off-springs. And finally these off-springs replace the existing individuals in the population and the process repeats. In this way genetic algorithms actually try to mimic the human evolution to some extent.

Each of the following steps are covered as a separate chapter later in this tutorial.
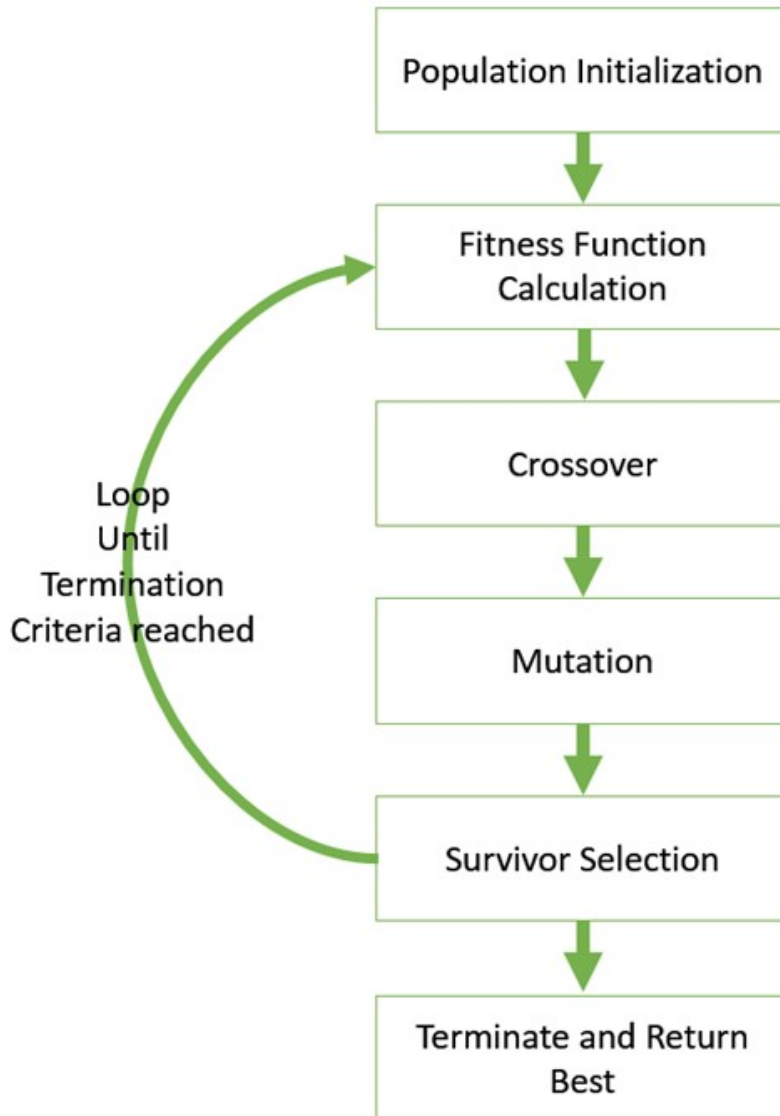
A generalized pseudo-code for a GA is explained in the following program −

```
GA()
  initialize population
  find fitness of population

  while (termination criteria is reached) do
    parent selection
    crossover with probability pc
    mutation with probability pm
    decode and fitness calculation
    survivor selection
    find best
  return best
```

Encoding, selection criteria, Crossover, Mutation

One of the most important decisions to make while implementing a genetic algorithm is deciding the representation that we will use to represent our solutions. It has been observed that improper representation can lead to poor performance of the GA.

Therefore, choosing a proper representation, having a proper definition of the mappings between the phenotype and genotype spaces is essential for the success of a GA.

In this section, we present some of the most commonly used representations for genetic algorithms. However, representation is highly problem specific and the reader might find that another representation or a mix of the representations mentioned here might suit his/her problem better.

Binary Representation

This is one of the simplest and most widely used representation in GAs. In this type of representation the genotype consists of bit strings.

For some problems when the solution space consists of Boolean decision variables – yes or no, the binary representation is natural. Take for example the 0/1 Knapsack Problem. If there are n items, we can represent a solution by a binary string of n elements, where the xth element tells whether the item x is picked (1) or not (0).

0	0	1	0	1	1	1	0	0	1

For other problems, specifically those dealing with numbers, we can represent the numbers with their binary representation. The problem with this kind of encoding is that different bits have different significance and therefore mutation and crossover operators can have undesired consequences. This can be resolved to some extent by using **Gray Coding,** as a change in one bit does not have a massive effect on the solution.

Real Valued Representation

For problems where we want to define the genes using continuous rather than discrete variables, the real valued representation is the most natural. The precision of these real valued or floating point numbers is however limited to the computer.

0.5	0.2	0.6	0.8	0.7	0.4	0.3	0.2	0.1	0.9

Integer Representation

For discrete valued genes, we cannot always limit the solution space to binary 'yes' or 'no'. For example, if we want to encode the four distances – North, South, East and West, we can encode them as **{0,1,2,3}**. In such cases, integer representation is desirable.

1	2	3	4	3	2	4	1	2	1

Permutation Representation

In many problems, the solution is represented by an order of elements. In such cases permutation representation is the most suited.

A classic example of this representation is the travelling salesman problem (TSP). In this the salesman has to take a tour of all the cities, visiting each city exactly once and come back to the starting city. The total distance of the tour has to be minimized. The solution to this TSP is naturally an ordering or permutation of all the cities and therefore using a permutation representation makes sense for this problem.

1	5	9	8	7	4	2	3	6	0

Population is a subset of solutions in the current generation. It can also be defined as a set of chromosomes. There are several things to be kept in mind when dealing with GA population −

- The diversity of the population should be maintained otherwise it might lead to premature convergence.
- The population size should not be kept very large as it can cause a GA to slow down, while a smaller population might not be enough for a good mating pool. Therefore, an optimal population size needs to be decided by trial and error.

The population is usually defined as a two dimensional array of – **size population, size x, chromosome size**.

Population Initialization

There are two primary methods to initialize a population in a GA. They are −

- **Random Initialization** − Populate the initial population with completely random solutions.
- **Heuristic initialization** − Populate the initial population using a known heuristic for the problem.

It has been observed that the entire population should not be initialized using a heuristic, as it can result in the population having similar solutions and very little diversity. It has been experimentally observed that the random solutions are the ones to drive the population to optimality. Therefore, with heuristic initialization, we just seed the population with a couple of good solutions, filling up the rest with random solutions rather than filling the entire population with heuristic based solutions.

It has also been observed that heuristic initialization in some cases, only effects the initial fitness of the population, but in the end, it is the diversity of the solutions which lead to optimality.

Population Models

There are two population models widely in use −

Steady State

In steady state GA, we generate one or two off-springs in each iteration and they replace one or two individuals from the population. A steady state GA is also known as **Incremental GA**.

Generational

In a generational model, we generate 'n' off-springs, where n is the population size, and the entire population is replaced by the new one at the end of the iteration.

Genetic Algorithms - Fitness Function

The fitness function simply defined is a function which takes a **candidate solution to the problem as input and produces as output** how "fit" our how "good" the solution is with respect to the problem in consideration. Calculation of fitness value is done repeatedly in a GA and therefore it should be sufficiently fast. A slow computation of the fitness value can adversely affect a GA and make it exceptionally slow.

In most cases the fitness function and the objective function are the same as the objective is to either maximize or minimize the given objective function. However, for more complex problems with multiple objectives and constraints, an **Algorithm Designer** might choose to have a different fitness function.

A fitness function should possess the following characteristics −

- The fitness function should be sufficiently fast to compute.
- It must quantitatively measure how fit a given solution is or how fit individuals can be produced from the given solution.

In some cases, calculating the fitness function directly might not be possible due to the inherent complexities of the problem at hand. In such cases, we do fitness approximation to suit our needs.

The following image shows the fitness calculation for a solution of the 0/1 Knapsack. It is a simple fitness function which just sums the profit values of the items being picked (which have a 1), scanning the elements from left to right till the knapsack is full.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | Item Number |

| 0 | 1 | 0 | 1 | 1 | 0 | 1 | Chromosome |

| 2 | 9 | 8 | 5 | 4 | 0 | 2 | Profit Values |

| 7 | 5 | 3 | 1 | 5 | 9 | 8 | Weight Values |

Knapsack capacity = 15
Total associated profit = 18
Last item not picked as it exceeds knapsack capacity

Genetic Algorithms - Parent Selection

Parent Selection is the process of selecting parents which mate and recombine to create off-springs for the next generation. Parent selection is very crucial to the convergence rate of the GA as good parents drive individuals to a better and fitter solutions.

However, care should be taken to prevent one extremely fit solution from taking over the entire population in a few generations, as this leads to the solutions being close to one another in the solution space thereby leading to a loss of diversity. **Maintaining good diversity** in the population is extremely crucial for the success of a GA. This taking up of the entire population by one extremely fit solution is known as **premature convergence** and is an undesirable condition in a GA.

Fitness Proportionate Selection

Fitness Proportionate Selection is one of the most popular ways of parent selection. In this every individual can become a parent with a probability which is proportional to its fitness. Therefore, fitter individuals have a higher chance of mating and propagating their features to the next generation. Therefore, such a selection strategy applies a selection pressure to the more fit individuals in the population, evolving better individuals over time.

Consider a circular wheel. The wheel is divided into **n pies**, where n is the number of individuals in the population. Each individual gets a portion of the circle which is proportional to its fitness value.

Two implementations of fitness proportionate selection are possible −

Roulette Wheel Selection

In a roulette wheel selection, the circular wheel is divided as described before. A fixed point is chosen on the wheel circumference as shown and the wheel is rotated. The region of the wheel which comes in front of the fixed point is chosen as the parent. For the second parent, the same process is repeated.

Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3
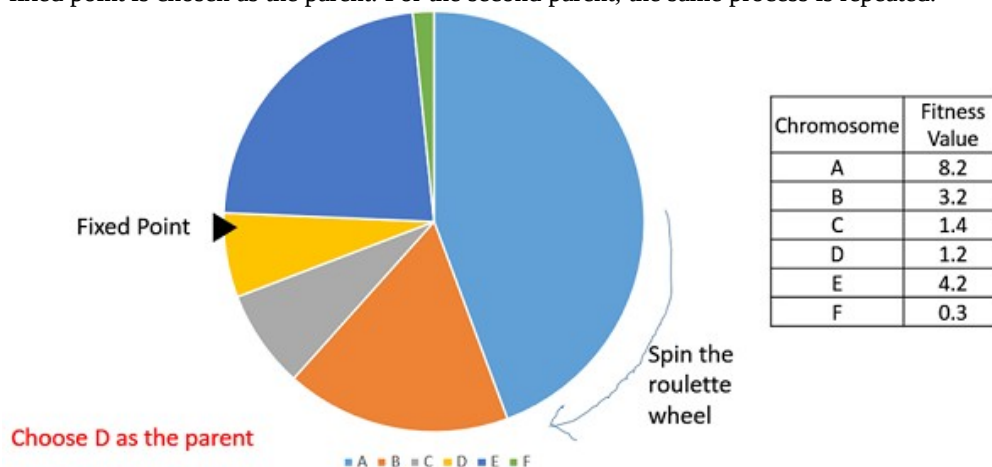
It is clear that a fitter individual has a greater pie on the wheel and therefore a greater chance of landing in front of the fixed point when the wheel is rotated. Therefore, the probability of choosing an individual depends directly on its fitness.

Implementation wise, we use the following steps −

- Calculate S = the sum of a finesses.
- Generate a random number between 0 and S.
- Starting from the top of the population, keep adding the finesses to the partial sum P, till P<S.
- The individual for which P exceeds S is the chosen individual.

Stochastic Universal Sampling (SUS)

Stochastic Universal Sampling is quite similar to Roulette wheel selection, however instead of having just one fixed point, we have multiple fixed points as shown in the following image. Therefore, all the parents are chosen in just one spin of the wheel. Also, such a setup encourages the highly fit individuals to be chosen at least once.
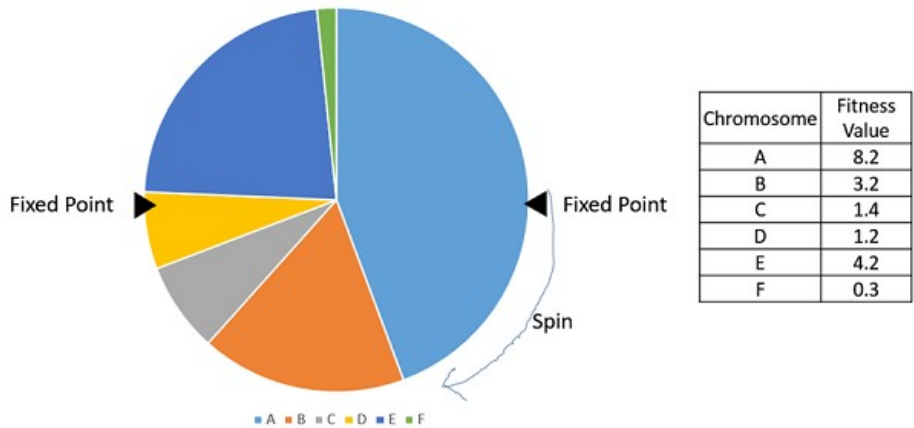
It is to be noted that fitness proportionate selection methods don't work for cases where the fitness can take a negative value.

Tournament Selection

In K-Way tournament selection, we select K individuals from the population at random and select the best out of these to become a parent. The same process is repeated for selecting the next parent. Tournament Selection is also extremely popular in literature as it can even work with negative fitness values.
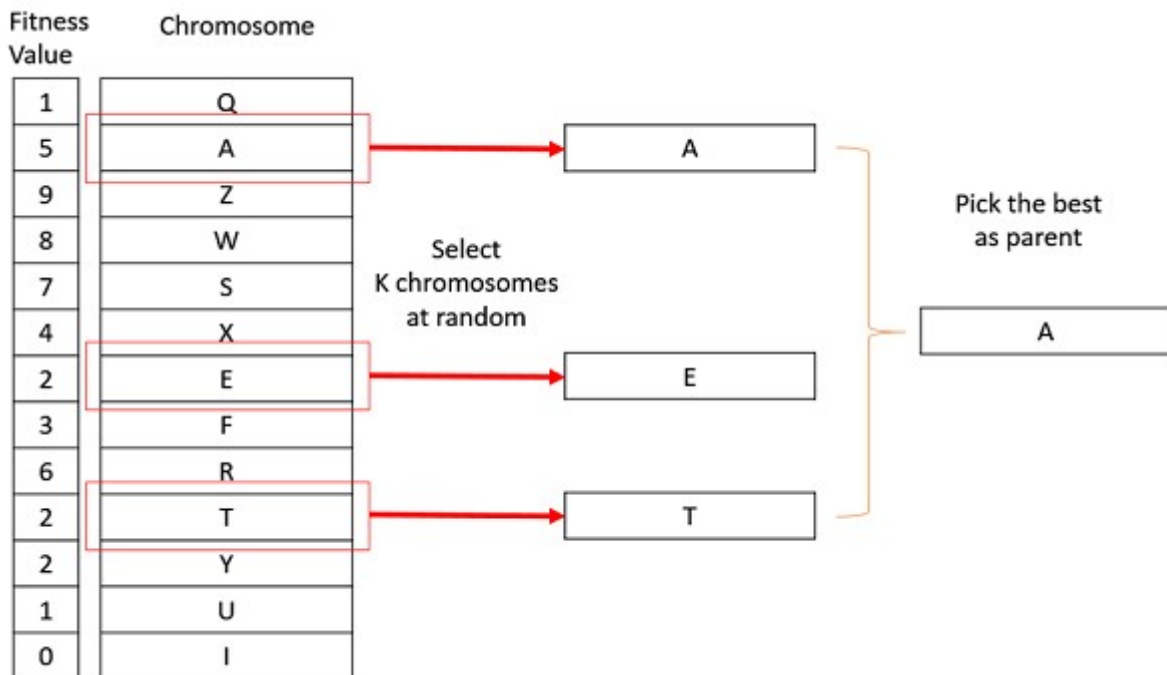
Rank Selection

Rank Selection also works with negative fitness values and is mostly used when the individuals in the population have very close fitness values (this happens usually at the end of the run). This leads to each individual having an almost equal share of the pie (like in case of fitness proportionate selection) as shown in the following image and hence each individual no matter how fit relative to each other has an approximately same probability of getting selected as a parent. This in turn leads to a loss in the selection pressure towards fitter individuals, making the GA to make poor parent selections in such situations.

Chromosome	Fitness Value
A	8.1
B	8.0
C	8.05
D	7.95
E	8.02
F	7.99

Fixed Point

Spin the roulette wheel

NO SELECTION PRESSURE

■ A ■ B ■ C ■ D ■ E ■ F
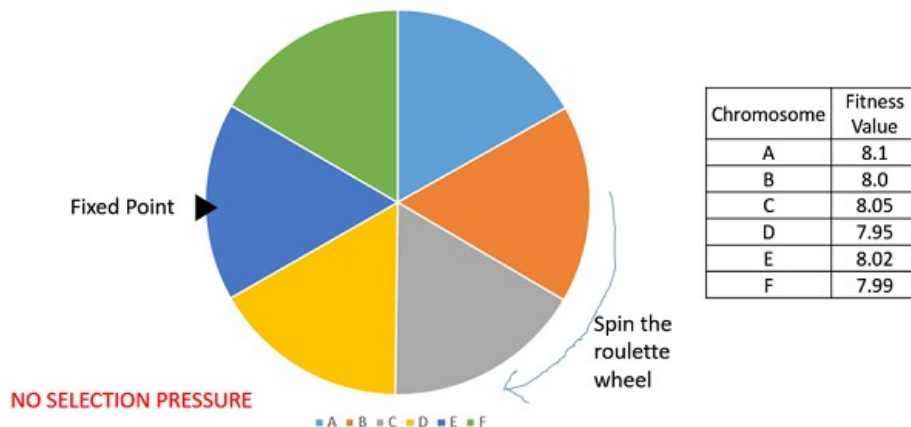
In this, we remove the concept of a fitness value while selecting a parent. However, every individual in the population is ranked according to their fitness. The selection of the parents depends on the rank of each individual and not the fitness. The higher ranked individuals are preferred more than the lower ranked ones.

Chromosome	Fitness Value	Rank
A	8.1	1
B	8.0	4
C	8.05	2
D	7.95	6
E	8.02	3
F	7.99	5

Random Selection

In this strategy we randomly select parents from the existing population. There is no selection pressure towards fitter individuals and therefore this strategy is usually avoided.

Genetic Algorithms - Crossover

In this chapter, we will discuss about what a Crossover Operator is along with its other modules, their uses and benefits.

Introduction to Crossover

The crossover operator is analogous to reproduction and biological crossover. In this more than one parent is selected and one or more off-springs are produced using the genetic material of the parents. Crossover is usually applied in a GA with a high probability – p_c .

Crossover Operators

In this section we will discuss some of the most popularly used crossover operators. It is to be noted that these crossover operators are very generic and the GA Designer might choose to implement a problem-specific crossover operator as well.

One Point Crossover

In this one-point crossover, a random crossover point is selected and the tails of its two parents are swapped to get new off-springs.
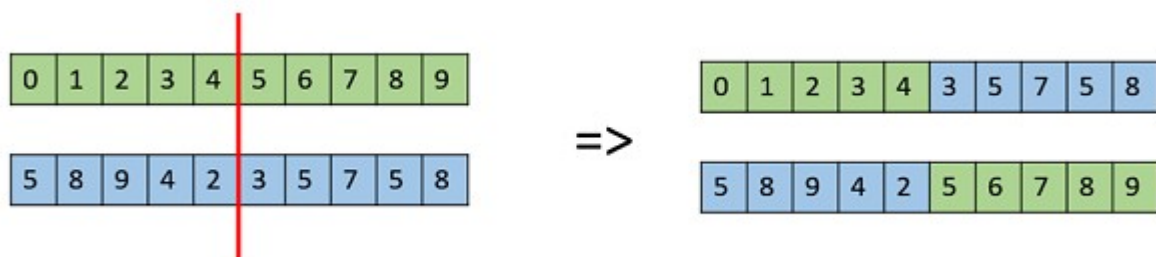
Multi Point Crossover

Multi point crossover is a generalization of the one-point crossover wherein alternating segments are swapped to get new off-springs.
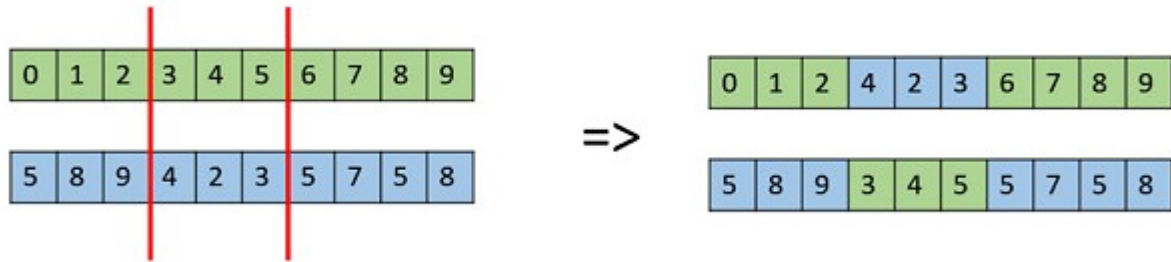
Uniform Crossover

In a uniform crossover, we don't divide the chromosome into segments, rather we treat each gene separately. In this, we essentially flip a coin for each chromosome to decide whether or not it'll be included in the off-spring. We can also bias the coin to one parent, to have more genetic material in the child from that parent.
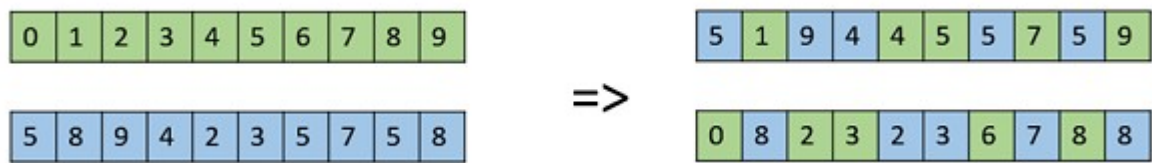
Whole Arithmetic Recombination

This is commonly used for integer representations and works by taking the weighted average of the two parents by using the following formulae −

- Child1 = $\alpha.x + (1-\alpha).y$
- Child2 = $\alpha.x + (1-\alpha).y$

Obviously, if $\alpha = 0.5$, then both the children will be identical as shown in the following image.

Davis' Order Crossover (OX1)

OX1 is used for permutation based crossovers with the intention of transmitting information about relative ordering to the off-springs. It works as follows −

- Create two random crossover points in the parent and copy the segment between them from the first parent to the first offspring.
- Now, starting from the second crossover point in the second parent, copy the remaining unused numbers from the second parent to the first child, wrapping around the list.
- Repeat for the second child with the parent's role reversed.
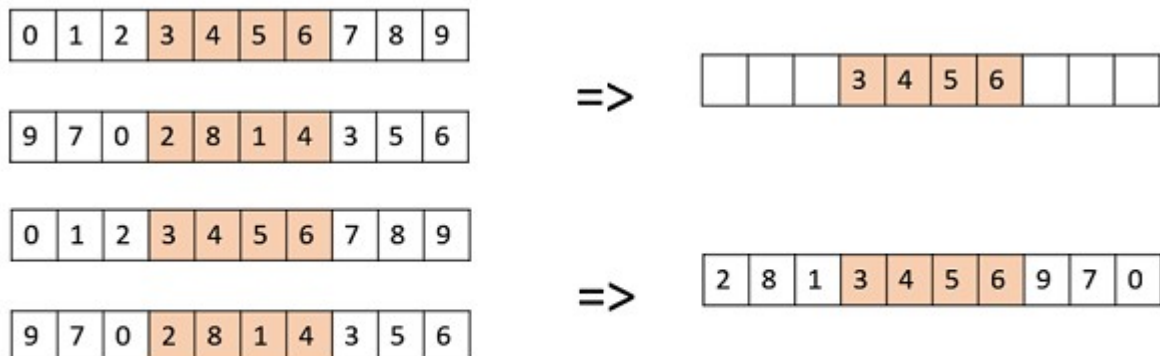
Repeat the same procedure to get the second child

There exist a lot of other crossovers like Partially Mapped Crossover (PMX), Order based crossover (OX2), Shuffle Crossover, Ring Crossover, etc.

Genetic Algorithms - Mutation

Introduction to Mutation

In simple terms, mutation may be defined as a small random tweak in the chromosome, to get a new solution. It is used to maintain and introduce diversity in the genetic population and is usually applied with a low probability – p_m. If the probability is very high, the GA gets reduced to a random search.

Mutation is the part of the GA which is related to the "exploration" of the search space. It has been observed that mutation is essential to the convergence of the GA while crossover is not.

Mutation Operators

In this section, we describe some of the most commonly used mutation operators. Like the crossover operators, this is not an exhaustive list and the GA designer might find a combination of these approaches or a problem-specific mutation operator more useful.

Bit Flip Mutation

In this bit flip mutation, we select one or more random bits and flip them. This is used for binary encoded GAs.

| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | => | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
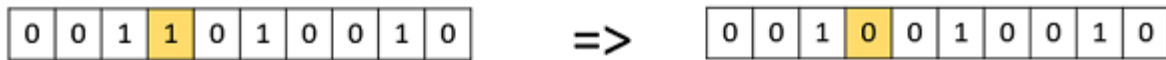
Random Resetting

Random Resetting is an extension of the bit flip for the integer representation. In this, a random value from the set of permissible values is assigned to a randomly chosen gene.

Swap Mutation

In swap mutation, we select two positions on the chromosome at random, and interchange the values. This is common in permutation based encodings.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | => | 1 | 6 | 3 | 4 | 5 | 2 | 7 | 8 | 9 | 0 |
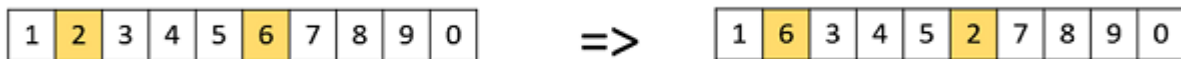
Scramble Mutation

Scramble mutation is also popular with permutation representations. In this, from the entire chromosome, a subset of genes is chosen and their values are scrambled or shuffled randomly.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | => | 0 | 1 | 3 | 6 | 4 | 2 | 5 | 7 | 8 | 9 |
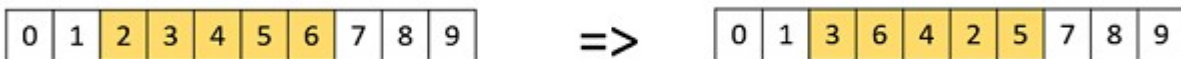
Inversion Mutation

In inversion mutation, we select a subset of genes like in scramble mutation, but instead of shuffling the subset, we merely invert the entire string in the subset.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | => | 0 | 1 | 6 | 5 | 4 | 3 | 2 | 7 | 8 | 9 |
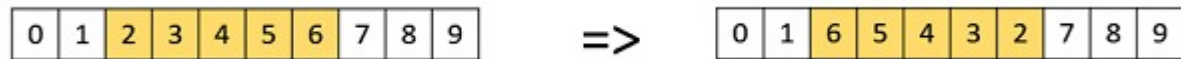
Genetic Algorithms - Survivor Selection

The Survivor Selection Policy determines which individuals are to be kicked out and which are to be kept in the next generation. It is crucial as it should ensure that the fitter individuals are not kicked out of the population, while at the same time diversity should be maintained in the population.

Some GAs employ **Elitism**. In simple terms, it means the current fittest member of the population is always propagated to the next generation. Therefore, under no circumstance can the fittest member of the current population be replaced.

The easiest policy is to kick random members out of the population, but such an approach frequently has convergence issues, therefore the following strategies are widely used.

Age Based Selection

In Age-Based Selection, we don't have a notion of a fitness. It is based on the premise that each individual is allowed in the population for a finite generation where it is allowed to reproduce, after that, it is kicked out of the population no matter how good its fitness is.

For instance, in the following example, the age is the number of generations for which the individual has been in the population. The oldest members of the population i.e. P4 and P7 are kicked out of the population and the ages of the rest of the members are incremented by one.
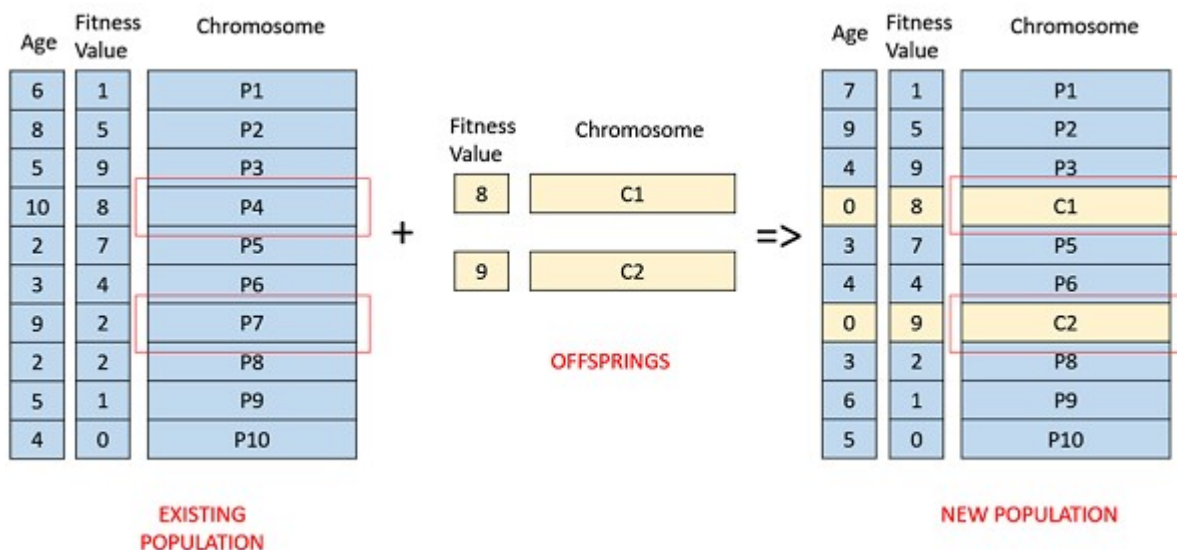
Fitness Based Selection

In this fitness based selection, the children tend to replace the least fit individuals in the population. The selection of the least fit individuals may be done using a variation of any of the selection policies described before – tournament selection, fitness proportionate selection, etc.

For example, in the following image, the children replace the least fit individuals P1 and P10 of the population. It is to be noted that since P1 and P9 have the same fitness value, the decision to remove which individual from the population is arbitrary.
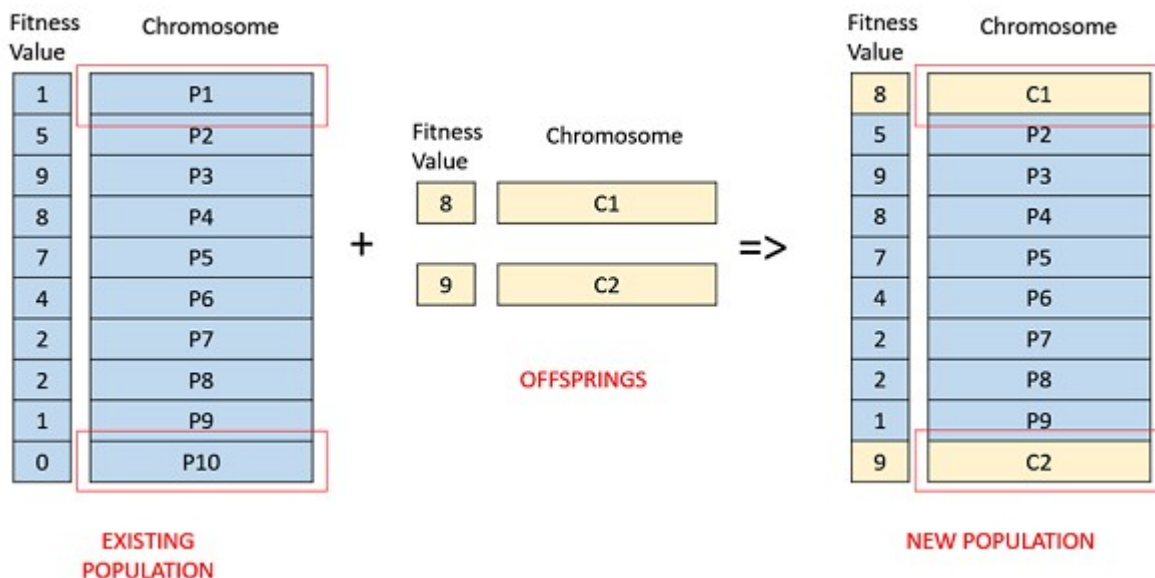
Genetic Algorithms - Termination Condition

The termination condition of a Genetic Algorithm is important in determining when a GA run will end. It has been observed that initially, the GA progresses very fast with better solutions coming in every few iterations, but this tends to saturate in the later stages where the improvements are very small. We usually want a termination condition such that our solution is close to the optimal, at the end of the run.

Usually, we keep one of the following termination conditions −

- When there has been no improvement in the population for X iterations.
- When we reach an absolute number of generations.
- When the objective function value has reached a certain pre-defined value.

For example, in a genetic algorithm we keep a counter which keeps track of the generations for which there has been no improvement in the population. Initially, we set this counter to zero. Each time we don't generate off-springs which are better than the individuals in the population, we increment the counter.

However, if the fitness any of the off-springs is better, then we reset the counter to zero. The algorithm terminates when the counter reaches a predetermined value.

Like other parameters of a GA, the termination condition is also highly problem specific and the GA designer should try out various options to see what suits his particular problem the best.

Genetic Algorithms - Application Areas

Genetic Algorithms are primarily used in optimization problems of various kinds, but they are frequently used in other application areas as well.

In this section, we list some of the areas in which Genetic Algorithms are frequently used. These are −

- **Optimization** − Genetic Algorithms are most commonly used in optimization problems wherein we have to maximize or minimize a given objective function value under a given set of constraints. The approach to solve Optimization problems has been highlighted throughout the tutorial.
- **Economics** − GAs are also used to characterize various economic models like the cobweb model, game theory equilibrium resolution, asset pricing, etc.
- **Neural Networks** − GAs are also used to train neural networks, particularly recurrent neural networks.
- **Parallelization** − GAs also have very good parallel capabilities, and prove to be very effective means in solving certain problems, and also provide a good area for research.
- **Image Processing** − GAs are used for various digital image processing (DIP) tasks as well like dense pixel matching.
- **Vehicle routing problems** − With multiple soft time windows, multiple depots and a heterogeneous fleet.
- **Scheduling applications** − GAs are used to solve various scheduling problems as well, particularly the time tabling problem.
- **Machine Learning** − as already discussed, genetics based machine learning (GBML) is a niche area in machine learning.
- **Robot Trajectory Generation** − GAs have been used to plan the path which a robot arm takes by moving from one point to another.
- **Parametric Design of Aircraft** − GAs have been used to design aircrafts by varying the parameters and evolving better solutions.
- **DNA Analysis** − GAs have been used to determine the structure of DNA using spectrometric data about the sample.
- **Multimodal Optimization** − GAs are obviously very good approaches for multimodal optimization in which we have to find multiple optimum solutions.
- **Traveling salesman problem and its applications** − GAs have been used to solve the TSP, which is a well-known combinatorial problem using novel crossover and packing strategies.

Genetic algorithms are substantially different to the more traditional search

Genetic algorithms are substantially different to the more traditional search andoptimization techniques. The five main differences are:

1. Genetic algorithms search a population of points in parallel, not from a single point.

2. Genetic algorithms do not require derivative information or other auxiliaryknowledge; only the objective function and corresponding fitness levels influencethe direction of the search

.3. Genetic algorithms use probabilistic transition rules, not deterministic rules.

4. Genetic algorithms work on an encoding of a parameter set not the parameter setitself (except where real-valued individuals are used).

5. Genetic algorithms may provide a number of potential solutions to a given problem and the choice of the final is left up to the use

Soft Computing: Module-III

Neural Network : Biological neuron, artificial neuron, definition of ANN, Taxonomy of neural net,

Yet another research area in AI, neural networks, is inspired from the natural neural network of human nervous system.

What are Artificial Neural Networks (ANNs)?

The inventor of the first neurocomputer, Dr. Robert Hecht-Nielsen, defines a neural network as −

"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."

Basic Structure of ANNs

The idea of ANNs is based on the belief that working of human brain by making the right connections, can be imitated using silicon and wires as living **neurons** and **dendrites**.

The human brain is composed of 86 billion nerve cells called **neurons.** They are connected to other thousand cells by **Axons.** Stimuli from external environment or inputs from sensory organs are accepted by dendrites. These inputs create electric impulses, which quickly travel through the neural network. A neuron can then send the message to other neuron to handle the issue or does not send it forward.
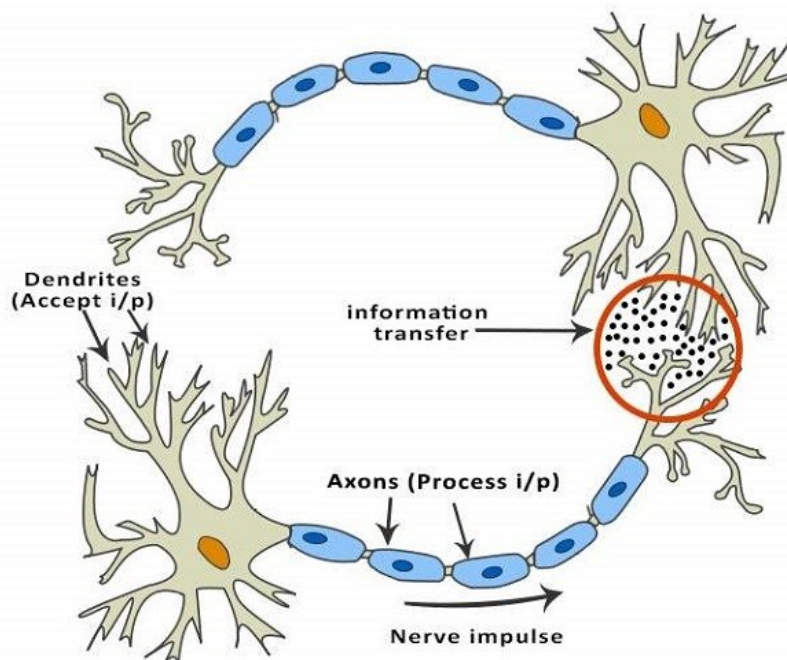
ANNs are composed of multiple **nodes**, which imitate biological **neurons** of human brain. The neurons are connected by links and they interact with each other. The nodes can take input data and perform simple operations on the data. The result of these operations is passed to other neurons. The output at each node is called its **activation** or **node value.**

Each link is associated with **weight.** ANNs are capable of learning, which takes place by altering weight values. The following illustration shows a simple ANN −
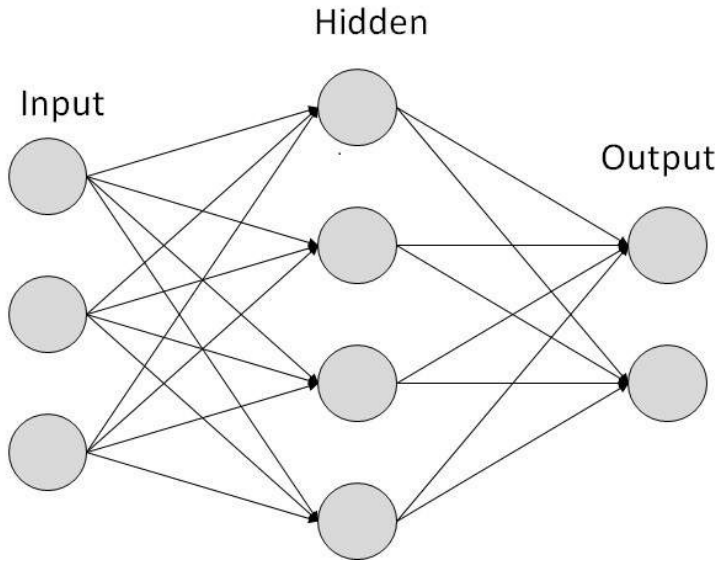
Types of Artificial Neural Networks

There are two Artificial Neural Network topologies − **FeedForward** and **Feedback.**

FeedForward ANN

In this ANN, the information flow is unidirectional. A unit sends information to other unit from which it does not receive any information. There are no feedback loops. They are used in pattern generation/recognition/classification. They have fixed inputs and outputs.
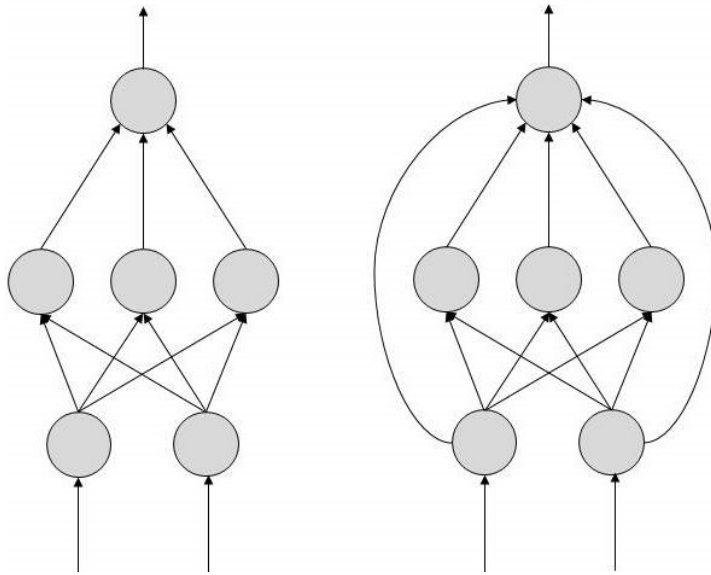
FeedBack ANN

Here, feedback loops are allowed. They are used in content addressable memories.
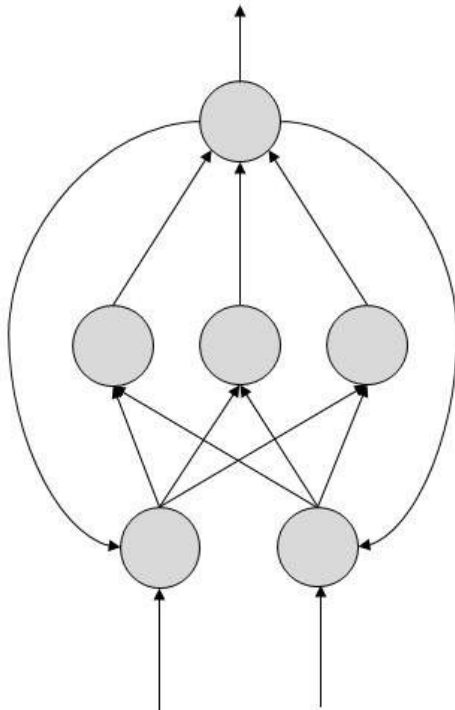
Working of ANNs

In the topology diagrams shown, each arrow represents a connection between two neurons and indicates the pathway for the flow of information. Each connection has a weight, an integer number that controls the signal between the two neurons.

If the network generates a "good or desired" output, there is no need to adjust the weights. However, if the network generates a "poor or undesired" output or an error, then the system alters the weights in order to improve subsequent results.

Machine Learning in ANNs

ANNs are capable of learning and they need to be trained. There are several learning strategies −

- **Supervised Learning** − It involves a teacher that is scholar than the ANN itself. For example, the teacher feeds some example data about which the teacher already knows the answers.
  For example, pattern recognizing. The ANN comes up with guesses while recognizing. Then the teacher provides the ANN with the answers. The network then compares it guesses with the teacher's "correct" answers and makes adjustments according to errors.
- **Unsupervised Learning** − It is required when there is no example data set with known answers. For example, searching for a hidden pattern. In this case, clustering i.e. dividing a set of elements into groups according to some unknown pattern is carried out based on the existing data sets present.
- **Reinforcement Learning** − This strategy built on observation. The ANN makes a decision by observing its environment. If the observation is negative, the network adjusts its weights to be able to make a different required decision the next time.

Back Propagation Algorithm

It is the training or learning algorithm. It learns by example. If you submit to the algorithm the example of what you want the network to do, it changes the network's weights so that it can produce desired output for a particular input on finishing the training.

Back Propagation networks are ideal for simple Pattern Recognition and Mapping Tasks.

Bayesian Networks (BN)

These are the graphical structures used to represent the probabilistic relationship among a set of random variables. Bayesian networks are also called **Belief Networks** or **Bayes Nets.** BNs reason about uncertain domain.

In these networks, each node represents a random variable with specific propositions. For example, in a medical diagnosis domain, the node Cancer represents the proposition that a patient has cancer.

The edges connecting the nodes represent probabilistic dependencies among those random variables. If out of two nodes, one is affecting the other then they must be directly connected in the directions of the effect. The strength of the relationship between variables is quantified by the probability associated with each node.

There is an only constraint on the arcs in a BN that you cannot return to a node simply by following directed arcs. Hence the BNs are called Directed Acyclic Graphs (DAGs).

BNs are capable of handling multivalued variables simultaneously. The BN variables are composed of two dimensions −

- Range of prepositions
- Probability assigned to each of the prepositions.

Consider a finite set X = {X_1, X_2, …,X_n} of discrete random variables, where each variable X_i may take values from a finite set, denoted by *Val(X_i)*. If there is a directed link from variable X_i to variable, X_j, then variable X_i will be a parent of variable X_j showing direct dependencies between the variables.

The structure of BN is ideal for combining prior knowledge and observed data. BN can be used to learn the causal relationships and understand various problem domains and to predict future events, even in case of missing data.

Building a Bayesian Network

A knowledge engineer can build a Bayesian network. There are a number of steps the knowledge engineer needs to take while building it.

Example problem − *Lung cancer.* A patient has been suffering from breathlessness. He visits the doctor, suspecting he has lung cancer. The doctor knows that barring lung cancer, there are various other possible diseases the patient might have such as tuberculosis and bronchitis.

Gather Relevant Information of Problem

- Is the patient a smoker? If yes, then high chances of cancer and bronchitis.
- Is the patient exposed to air pollution? If yes, what sort of air pollution?
- Take an X-Ray positive X-ray would indicate either TB or lung cancer.

Identify Interesting Variables

The knowledge engineer tries to answer the questions −

- Which nodes to represent?
- What values can they take? In which state can they be?

For now let us consider nodes, with only discrete values. The variable must take on exactly one of these values at a time.

Common types of discrete nodes are −

- **Boolean nodes** − They represent propositions, taking binary values TRUE (T) and FALSE (F).
- **Ordered values** − A node *Pollution* might represent and take values from {low, medium, high} describing degree of a patient's exposure to pollution.
- **Integral values** − A node called *Age* might represent patient's age with possible values from 1 to 120. Even at this early stage, modeling choices are being made.

Possible nodes and values for the lung cancer example −

Node Name	Type	Value	Nodes Creation
Polution	Binary	{LOW, HIGH, MEDIUM}	
Smoker	Boolean	{TRUE, FASLE}	
Lung-Cancer	Boolean	{TRUE, FASLE}	
X-Ray	Binary	{Positive, Negative}	

Create Arcs between Nodes

Topology of the network should capture qualitative relationships between variables.

For example, what causes a patient to have lung cancer? - Pollution and smoking. Then add arcs from node *Pollution* and node *Smoker* to node *Lung-Cancer*.

Similarly if patient has lung cancer, then X-ray result will be positive. Then add arcs from node *Lung-Cancer* to node *X-Ray*.
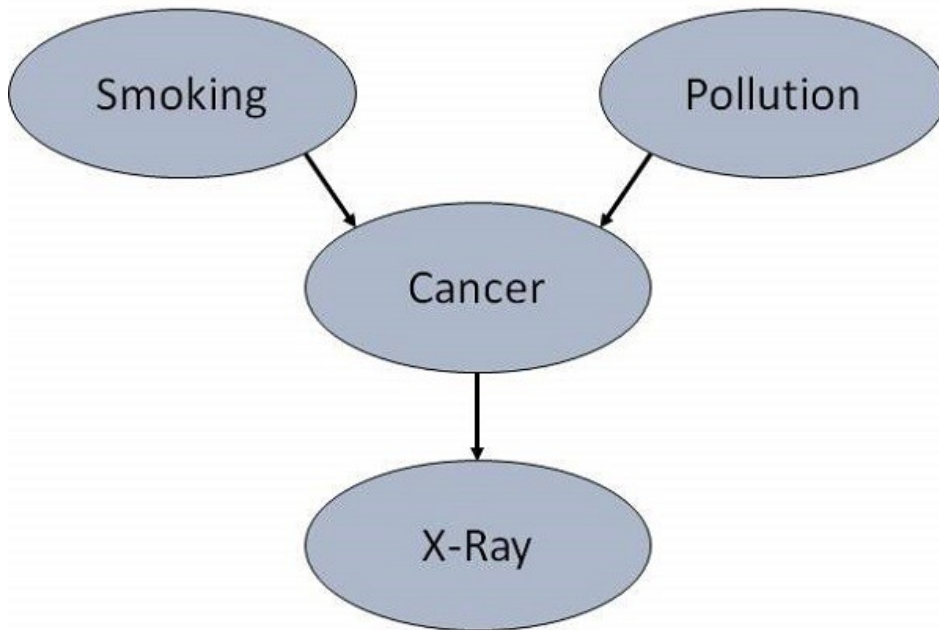
Specify Topology

Conventionally, BNs are laid out so that the arcs point from top to bottom. The set of parent nodes of a node X is given by Parents(X).

The *Lung-Cancer* node has two parents (reasons or causes): *Pollution* and *Smoker*, while node *Smoker* is an **ancestor** of node *X-Ray*. Similarly, *X-Ray* is a child (consequence or effects) of node *Lung-Cancer* and **successor** of nodes *Smoker* and *Pollution*.

Conditional Probabilities

Now quantify the relationships between connected nodes: this is done by specifying a conditional probability distribution for each node. As only discrete variables are considered here, this takes the form of a **Conditional Probability Table (CPT).**

First, for each node we need to look at all the possible combinations of values of those parent nodes. Each such combination is called an **instantiation** of the parent set. For each distinct instantiation of parent node values, we need to specify the probability that the child will take.

For example, the *Lung-Cancer* node's parents are *Pollution* and *Smoking.* They take the possible values = { (H,T), (H,F), (L,T), (L,F)}. The CPT specifies the probability of cancer for each of these cases as <0.05, 0.02, 0.03, 0.001> respectively.

Each node will have conditional probability associated as follows −

Smoking
P(S = T)
0.30

Pollution
P(P = L)
0.90

Lung-Cancer		
P	S	P (C = T|P, S)
H	T	0.05
H	F	0.02
L	T	0.03
L	F	0.001

X-Ray	
C	X = (Pos | C)
T	0.90
F	0.20

Applications of Neural Networks

They can perform tasks that are easy for a human but difficult for a machine −

- **Aerospace** − Autopilot aircrafts, aircraft fault detection.
- **Automotive** − Automobile guidance systems.
- **Military** − Weapon orientation and steering, target tracking, object discrimination, facial recognition, signal/image identification.
- **Electronics** − Code sequence prediction, IC chip layout, chip failure analysis, machine vision, voice synthesis.
- **Financial** − Real estate appraisal, loan advisor, mortgage screening, corporate bond rating, portfolio trading program, corporate financial analysis, currency value prediction, document readers, credit application evaluators.
- **Industrial** − Manufacturing process control, product design and analysis, quality inspection systems, welding quality analysis, paper quality prediction, chemical product design analysis, dynamic modeling of chemical process systems, machine maintenance analysis, project bidding, planning, and management.
- **Medical** − Cancer cell analysis, EEG and ECG analysis, prosthetic design, transplant time optimizer.
- **Speech** − Speech recognition, speech classification, text to speech conversion.
- **Telecommunications** − Image and data compression, automated information services, real-time spoken language translation.
- **Transportation** − Truck Brake system diagnosis, vehicle scheduling, routing systems.
- **Software** − Pattern Recognition in facial recognition, optical character recognition, etc.
- **Time Series Prediction** − ANNs are used to make predictions on stocks and natural calamities.
- **Signal Processing** − Neural networks can be trained to process an audio signal and filter it appropriately in the hearing aids.
- **Control** − ANNs are often used to make steering decisions of physical vehicles.
- **Anomaly Detection** − As ANNs are expert at recognizing patterns, they can also be trained to generate an output when something unusual occurs that misfits the pattern.
-

Difference between ANN and human brain,

The main differences

1. Size: our brain contains about 86 billion neurons and more than a 100 trillion (or according to some estimates 1000 trillion) synapses (connections). The number of "neurons" in artificial networks is much less than that (usually in the ballpark of 10–1000) but comparing their numbers this way is misleading. Perceptrons just take inputs on their "dendrites" and generate output on their "axon branches". A single layer perceptron network consists of several perceptrons that are not interconnected: they all just perform this very same task at once. Deep Neural Networks usually consist of input neurons (as many as the number of features in the data), output neurons (as many as the number of classes if they are built to solve a classification problem) and neurons in the hidden layers, in-between. All the layers are usually (but not necessarily) fully connected to the next layer, meaning that artificial neurons usually have as many connections as there are artificial neurons in the preceding and following layers combined. Convolutional Neural Networks also use different techniques to extract features from the data that are more sophisticated than what a few interconnected neurons can do alone. Manual feature extraction (altering data in a way that it can be fed to machine learning algorithms) requires human brain power which is also not taken into account when summing up the number of "neurons" required for Deep Learning tasks. The limitation in size isn't just computational: simply increasing the number of layers and artificial neurons does not always yield better results in machine learning tasks.

2. Topology: all artificial layers compute one by one, instead of being part of a network that has nodes computing asynchronously. Feedforward networks compute the state of one layer of artificial neurons and their weights, then use the results to compute the following layer the same way. During backpropagation, the algorithm computes some change in the weights the opposing way, to reduce the difference of the feedforward computational results in the output layer from the expected values of the output layer. Layers aren't connected to non-neighboring layers, but it's possible to somewhat mimic loops with recurrent and LSTM networks. In biological networks, neurons can fire asynchronously in parallel, have small-world nature with a small portion of highly connected neurons (hubs) and a large amount of lesser connected ones (the degree distribution at least partly follows the power-law). Since artificial neuron layers are usually fully connected, this small-world nature of biological neurons can only be simulated by introducing weights that are 0 to mimic the lack of connections between two neurons.

3. Speed: certain biological neurons can fire around 200 times a second on average. Signals travel at different speeds depending on the type of the nerve impulse, ranging from 0.61 m/s up to 119 m/s. Signal

travel speeds also vary from person to person depending on their sex, age, height, temperature, medical condition, lack of sleep etc. Action potential frequency carries <u>information for biological neuron networks</u>: information is carried by the firing frequency or the firing mode (tonic or burst-firing) of the output neuron and by the amplitude of the incoming signal in the input neuron in biological systems. Information in artificial neurons is instead carried over by the continuous, floating point number values of synaptic weights. How quickly feedforward or backpropagation algorithms are calculated carries no information, other than making the execution and training of the model faster. There are no refractory periods for artificial neural networks (periods while it is impossible to send another action potential, due to the sodium channels being lock shut) and artificial neurons do not experience "fatigue": they are functions that can be calculated as many times and as fast as the computer architecture would allow. Since artificial neural network models can be understood as just a bunch of matrix operations and finding derivatives, running such calculations can be highly optimized for vector processors (doing the very same calculations on large amounts of data points over and over again) and sped up by magnitudes using GPUs or dedicated hardware (like on <u>AI chips in recent SmartPhones</u>).

4.   Fault-tolerance: biological neuron networks due to their topology are also fault-tolerant. Information is stored redundantly so minor failures will not result in memory loss. They don't have one "central" part. The brain can also recover and heal to an extent. Artificial neural networks are not modeled for fault tolerance or self regeneration (similarly to fatigue, these ideas are not applicable to matrix operations), though recovery is possible by saving the current state (weight values) of the model and continuing the training from that save state. Dropouts can turn on and off random neurons in a layer during training, mimicking unavailable paths for signals and forcing some redundancy (dropouts are actually used to reduce the chance of overfitting). Trained models can be exported and used on different devices that support the framework, meaning that the same artificial neural network model will yield the same outputs for the same input data on every device it runs on. Training artificial neural networks for longer periods of time will not affect the efficiency of the artificial neurons. However, the hardware used for training can wear out really fast if used regularly, and will need to be replaced. Another difference is, that all processes (states and values) can be closely monitored inside an artificial neural network.

5.   Power consumption: the brain consumes about 20% of all the human body's energy—despite it's large cut, an adult brain operates on about 20 watts (barely enough to dimly light a bulb) being extremely efficient. Taking into account how humans can still operate for a while, when <u>only given some c-vitamin rich lemon juice and beef tallow</u>, this is quite remarkable. For benchmark: a single Nvidia GeForce Titan X GPU runs on <u>250 watts</u> alone, and requires a power supply instead of beef tallow. Our machines are way less efficient than biological systems. Computers also generate a lot of heat when used, with consumer GPUs operating safely between 50–80 degrees Celsius instead of 36.5–37.5 °C.

6.   Signals: an action potential is either triggered or not—biological synapses either carry a signal or they don't. Perceptrons work somewhat similarly, by accepting binary inputs, applying weights to them and generating binary outputs depending on whether the sum of these weighted inputs have reached a certain threshold (also called a step function). Artificial neurons accept continuous values as inputs and apply a simple non-linear, easily differentiable function (an activation function) on the sum of its weighted inputs to restrict the outputs' range of values. The activation functions are nonlinear so multiple layers in theory could <u>approximate any function</u>. Formerly sigmoid and hyperbolic tangent functions were used as activation functions, but these networks suffered from the vanishing gradient problem, meaning that the more the layers in a network, the less the changes in the first layers will affect the output, due to these functions <u>squashing their inputs into a very small output range</u>. These problems were overcome by the introduction of different activation functions such as <u>ReLU</u>. The final outputs of these networks are usually also squashed between 0—1 (representing probabilities for classification tasks) instead of outputting binary signals. As mentioned earlier, neither the frequency/speed of the signals nor the firing rates carry any information for artificial neural networks (this information is carried over by the input weights instead). The timing of the signals is synchronous, where artificial neurons in the same layer receive their input signals and then send their output signals all at once. Loops and time deltas can only be partly simulated with Recurrent (RNN) layers (that suffer greatly from the aforementioned vanishing gradient problem) or with Long short-term memory (LSTM) layers that act more like state machines or latch circuits than neurons. These are all considerable differences between biological and artificial neurons.

7.   Learning: we still do not understand how brains learn, or how redundant connections store and recall information. Brain fibers grow and reach out to connect to other neurons, neuroplasticity allows new connections to be created or areas to move and change function, and synapses may strengthen or weaken based on their importance. <u>Neurons that fire together, wire together</u> (although this is a very simplified theory and should not taken too literally). By learning, we are building on information that is already stored in the brain. Our knowledge deepens by repetition and during sleep, and tasks that once required a focus can be executed automatically once mastered. Artificial neural networks in the other hand, have a predefined

model, where no further neurons or connections can be added or removed. Only the weights of the connections (and biases representing thresholds) can change during training. The networks start with random weight values and will slowly try to reach a point where further changes in the weights would no longer improve performance. Just like there are many solutions for the same problems in real life, there is no guarantee that the weights of the network will be the best possible arrangement of weights to a problem —they will only represent one of the infinite approximations to infinite solutions. Learning can be understood as the process of finding optimal weights to minimize the differences between the network's expected and generated output: changing weights one way would increase this error, changing them the other way would decrees it. Imagine a foggy mountain top, where all we could tell is that <u>stepping towards a certain direction would take us downhill</u>. By repeating this process, we would eventually reach a valley where taking any step further would only take us higher. Once this valley is found we can say that we have reached a local minima. Note that it's possible that there are other, better valleys that are even lower from the mountain top (global minima) that we have missed, since we could not see them. Doing this in usually more than 3 dimensions is called gradient descent. To speed up this "learning process", instead of going through each and every example every time, random samples (batches) are taken from the data set and used for training iterations. This will only give an approximation of how to adjust the weights to reach a local minima (finding which direction to take downhill without carefully looking at all directions all the time), but it's still a pretty good approximation. We can also take larger steps when ascending the top and take smaller ones as we are reaching a valley where even small nudges could take us the wrong way. Walking like this downhill, going faster than carefully planning each and every step is called stochastic gradient descent. So the rate of how artificial neural networks learn can change over time (it decreases to ensure better performance), but there aren't any periods similar to human sleep phases when the networks would learn better. There is no neural fatigue either, although GPUs overheating during training can reduce performance. Once trained, an artificial neural network's weights can be exported and used to solve problem similar to the ones found in the training set. Training (backpropagation using an <u>optimization method</u> like stochastic gradient descent, over many layers and examples) is extremely expensive, but using a trained network (simply doing feedforward calculation) is ridiculously cheap. Unlike the brain, artificial neural networks don't learn by recalling information—they only learn during training, but will always "recall" the same, learned answers afterwards, without making a mistake. The great thing about this is that "recalling" can be done on much weaker hardware as many times as we want to. It is also possible to use previously pretrained models (to save time and resources by not having to start from a totally random set of weights) and improve them by training with additional examples that have the same input features. This is somewhat similar to how it's easier for the brain to learn certain things (like faces), by having dedicated areas for processing certain kinds of information.

So artificial and biological neurons do differ in more ways than the materials of their environment— biological neurons have only provided an inspiration to their artificial counterparts, but they are in no way direct copies with similar potential. If someone calls another human being smart or intelligent, we automatically assume that they are also capable of handling a large variety of problems, and are probably polite, kind and diligent as well. Calling a software intelligent only means that it is able to find an optimal solution to a set of problems.

Structure and Function of a single neuron, , single layer network,

Single-Layer Network
By connecting multiple neurons, the true computing power of the neural networks comes, though even a single neuron can perform substantial level of computation [Ler91]. The most common structure of connecting neurons into a network is by layers. The simplest form of layered network is shown in figure 2.7. The shaded nodes on the left are in the so-called *input layer*. The input layer neurons are to only pass and distribute the inputs and perform no computation. Thus, the only true layer of neurons is the one on the right. Each of the inputs $x_1, x_2, x_3, ..., x_N$ is connected to every artificial neuron in the output layer through the connection weight. Since every value of outputs $y_1, y_2, y_3, ..., y_N$ is calculated from the same set of input values, each output is varied based on the connection weights. Although the presented network is *fully connected*, the true biological neural network may not have all possible connections - the weight value of zero can be represented as ``no connection".
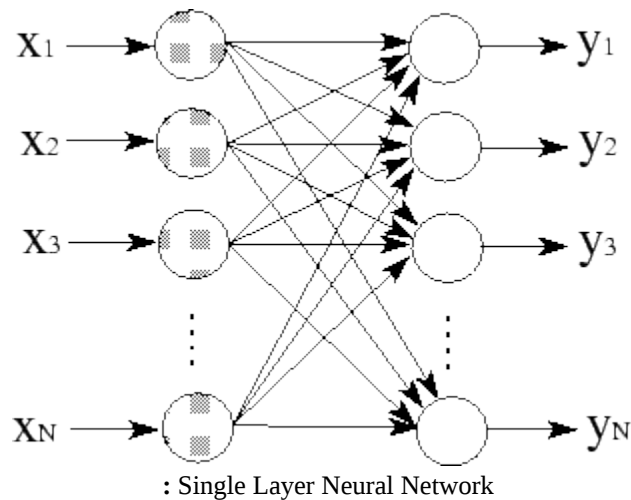
: Single Layer Neural Network

Perceptron training algorithm, Linear separability, Widrow & Hebb;s learning rule/Delta rule, ADALINE, MADALINE,

As the name suggests, **supervised learning** takes place under the supervision of a teacher. This learning process is dependent. During the training of ANN under supervised learning, the input vector is presented to the network, which will produce an output vector. This output vector is compared with the desired/target output vector. An error signal is generated if there is a difference between the actual output and the desired/target output vector. On the basis of this error signal, the weights would be adjusted until the actual output is matched with the desired output.

Perceptron

Developed by Frank Rosenblatt by using McCulloch and Pitts model, perceptron is the basic operational unit of artificial neural networks. It employs supervised learning rule and is able to classify the data into two classes.

Operational characteristics of the perceptron: It consists of a single neuron with an arbitrary number of inputs along with adjustable weights, but the output of the neuron is 1 or 0 depending upon the threshold. It also consists of a bias whose weight is always 1. Following figure gives a schematic representation of the perceptron.
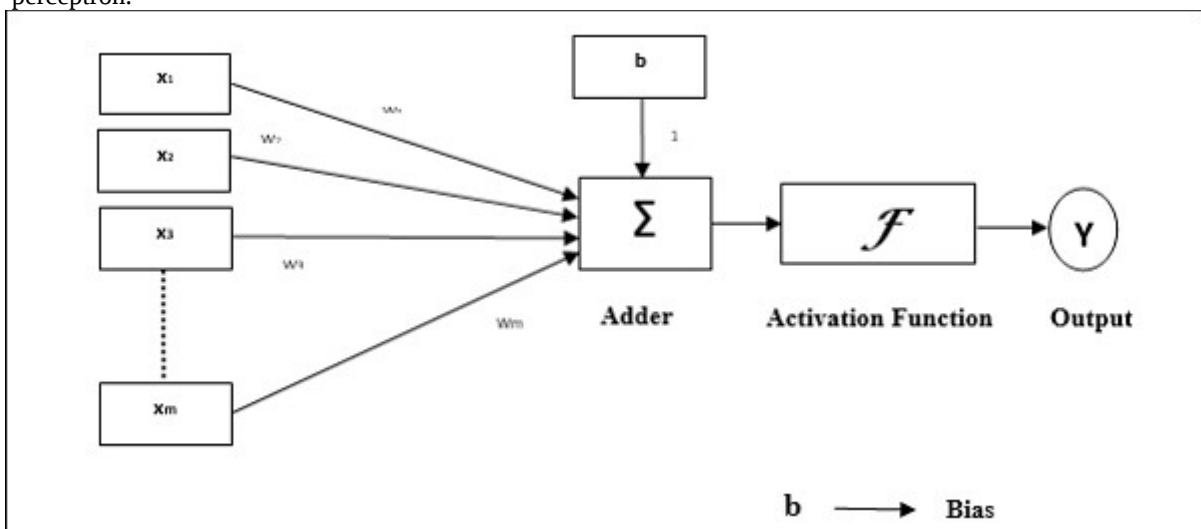
Perceptron thus has the following three basic elements −
- **Links** − It would have a set of connection links, which carries a weight including a bias always having weight 1.
- **Adder** − It adds the input after they are multiplied with their respective weights.

- **Activation function** − It limits the output of neuron. The most basic activation function is a Heaviside step function that has two possible outputs. This function returns 1, if the input is positive, and 0 for any negative input.

Training Algorithm

Perceptron network can be trained for single output unit as well as multiple output units.

Training Algorithm for Single Output Unit

Step 1 − Initialize the following to start the training −
- Weights
- Bias
- Learning rate $\alpha\alpha$

For easy calculation and simplicity, weights and bias must be set equal to 0 and the learning rate must be set equal to 1.

Step 2 − Continue step 3-8 when the stopping condition is not true.

Step 3 − Continue step 4-6 for every training vector **x**.

Step 4 − Activate each input unit as follows −

$$xi=si(i=1ton)xi=si(i=1ton)$$

Step 5 − Now obtain the net input with the following relation −

$$yin=b+\sum inxi.wiyin=b+\sum inxi.wi$$

Here **'b'** is bias and **'n'** is the total number of input neurons.

Step 6 − Apply the following activation function to obtain the final output.

$$f(yin)=\lceil\lfloor\{ 10-1ifyin>\theta if-\theta\leqslant yin\leqslant\theta ifyin<-\theta f(yin)=\{1ifyin>\theta 0if-\theta\leqslant yin\leqslant\theta-1ifyin<-\theta$$

Step 7 − Adjust the weight and bias as follows −

Case 1 − if **y ≠ t** then,

$$wi(new)=wi(old)+\alpha txiwi(new)=wi(old)+\alpha txi$$
$$b(new)=b(old)+\alpha tb(new)=b(old)+\alpha t$$

Case 2 − if **y = t** then,

$$wi(new)=wi(old)wi(new)=wi(old)$$
$$b(new)=b(old)b(new)=b(old)$$

Here **'y'** is the actual output and **'t'** is the desired/target output.

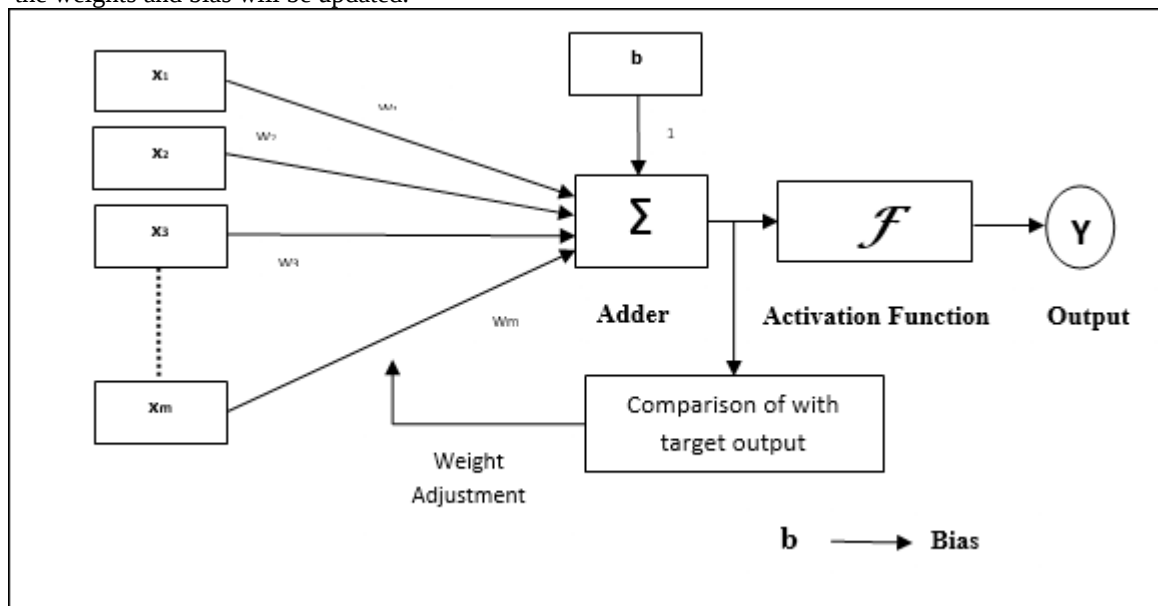**Step 8** − Test for the stopping condition, which would happen when there is no change in weight.

Training Algorithm for Multiple Output Units

The following diagram is the architecture of perceptron for multiple output classes.

Step 1 − Initialize the following to start the training −
- Weights
- Bias
- Learning rate $\alpha\alpha$

For easy calculation and simplicity, weights and bias must be set equal to 0 and the learning rate must be set equal to 1.

Step 2 − Continue step 3-8 when the stopping condition is not true.

Step 3 − Continue step 4-6 for every training vector **x**.

Step 4 − Activate each input unit as follows −

$$x_i = s_i \quad (i = 1 \text{ to } n)$$

Step 5 − Obtain the net input with the following relation −

$$y_{in} = b + \sum_i x_i w_{ij}$$

Here **'b'** is bias and **'n'** is the total number of input neurons.

Step 6 − Apply the following activation function to obtain the final output for each output unit **j = 1 to m** −

$$f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > \theta \\ 0 & \text{if } -\theta \leqslant y_{in} \leqslant \theta \\ -1 & \text{if } y_{in} < -\theta \end{cases}$$

Step 7 − Adjust the weight and bias for **x = 1 to n** and **j = 1 to m** as follows −

Case 1 − if $y_j \neq t_j$ then,

$$w_{ij}(new) = w_{ij}(old) + \alpha t_j x_i$$
$$b_j(new) = b_j(old) + \alpha t_j$$

Case 2 − if $y_j = t_j$ then,

$$w_{ij}(new) = w_{ij}(old)$$
$$b_j(new) = b_j(old)$$

Here **'y'** is the actual output and **'t'** is the desired/target output.

Step 8 − Test for the stopping condition, which will happen when there is no change in weight.

Adaptive Linear Neuron (Adaline)

Adaline which stands for Adaptive Linear Neuron, is a network having a single linear unit. It was developed by Widrow and Hoff in 1960. Some important points about Adaline are as follows −

- It uses bipolar activation function.
- It uses delta rule for training to minimize the Mean-Squared Error (MSE) between the actual output and the desired/target output.
- The weights and the bias are adjustable.

Architecture

The basic structure of Adaline is similar to perceptron having an extra feedback loop with the help of which the actual output is compared with the desired/target output. After comparison on the basis of training algorithm, the weights and bias will be updated.

Training Algorithm

Step 1 − Initialize the following to start the training −

- Weights
- Bias
- Learning rate α

For easy calculation and simplicity, weights and bias must be set equal to 0 and the learning rate must be set equal to 1.

Step 2 − Continue step 3-8 when the stopping condition is not true.

Step 3 − Continue step 4-6 for every bipolar training pair **s:t**.

Step 4 − Activate each input unit as follows −

$$x_i = s_i \quad (i = 1 \text{ to } n)$$

Step 5 − Obtain the net input with the following relation −

$$y_{in} = b + \sum_i x_i w_i$$

Here **'b'** is bias and **'n'** is the total number of input neurons.

Step 6 − Apply the following activation function to obtain the final output −

$$f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} \geqslant 0 \\ -1 & \text{if } y_{in} < 0 \end{cases}$$

Step 7 − Adjust the weight and bias as follows −

Case 1 − if **y ≠ t** then,

$$w_i(new) = w_i(old) + \alpha(t - y_{in})x_i$$
$$b(new) = b(old) + \alpha(t - y_{in})$$

Case 2 − if **y = t** then,

$$w_i(new) = w_i(old)$$
$$b(new) = b(old)$$

Here **'y'** is the actual output and **'t'** is the desired/target output.

$(t - y_{in})$ is the computed error.

Step 8 − Test for the stopping condition, which will happen when there is no change in weight or the highest weight change occurred during training is smaller than the specified tolerance.

Multiple Adaptive Linear Neuron (Madaline)

Madaline which stands for Multiple Adaptive Linear Neuron, is a network which consists of many Adalines in parallel. It will have a single output unit. Some important points about Madaline are as follows −

- It is just like a multilayer perceptron, where Adaline will act as a hidden unit between the input and the Madaline layer.
- The weights and the bias between the input and Adaline layers, as in we see in the Adaline architecture, are adjustable.
- The Adaline and Madaline layers have fixed weights and bias of 1.
- Training can be done with the help of Delta rule.

Architecture

The architecture of Madaline consists of **"n"** neurons of the input layer, **"m"** neurons of the Adaline layer, and 1 neuron of the Madaline layer. The Adaline layer can be considered as the hidden layer as it is between the input layer and the output layer, i.e. the Madaline layer.
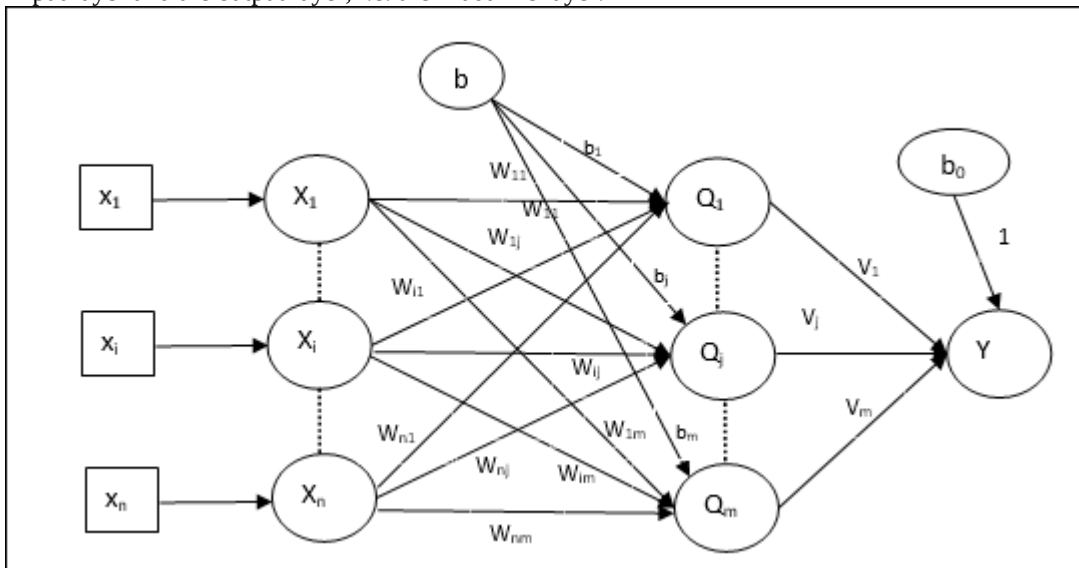
Training Algorithm

By now we know that only the weights and bias between the input and the Adaline layer are to be adjusted, and the weights and bias between the Adaline and the Madaline layer are fixed.

Step 1 − Initialize the following to start the training −

- Weights
- Bias
- Learning rate α

For easy calculation and simplicity, weights and bias must be set equal to 0 and the learning rate must be set equal to 1.

Step 2 − Continue step 3-8 when the stopping condition is not true.

Step 3 − Continue step 4-6 for every bipolar training pair **s:t**.

Step 4 − Activate each input unit as follows −

$$x_i = s_i (i = 1 \text{ to } n)$$

Step 5 − Obtain the net input at each hidden layer, i.e. the Adaline layer with the following relation −

$$Q_{inj} = b_j + \sum_i x_i w_{ij} \quad j = 1 \text{ to } m$$

Here **'b'** is bias and **'n'** is the total number of input neurons.

Step 6 − Apply the following activation function to obtain the final output at the Adaline and the Madaline layer −

$$f(x)=\{1 -1 if x \geqslant 0 if x<0 \quad f(x)=\{1 if x \geqslant 0 -1 if x<0$$

Output at the hidden (Adaline) unit

$$Q_j=f(Q_{inj}) \quad Q_j=f(Q_{inj})$$

Final output of the network

$$y=f(y_{in}) \quad y=f(y_{in})$$

i.e. $y_{inj}=b_0+\sum_{mj=1} Q_j v_j \quad y_{inj}=b_0+\sum_{j=1}^m Q_j v_j$

Step 7 − Calculate the error and adjust the weights as follows −

Case 1 − if **y ≠ t** and **t = 1** then,

$$w_{ij}(new)=w_{ij}(old)+\alpha(1-Q_{inj})x_i \quad w_{ij}(new)=w_{ij}(old)+\alpha(1-Q_{inj})x_i$$
$$b_j(new)=b_j(old)+\alpha(1-Q_{inj}) \quad b_j(new)=b_j(old)+\alpha(1-Q_{inj})$$

In this case, the weights would be updated on **Q_j** where the net input is close to 0 because **t = 1**.

Case 2 − if **y ≠ t** and **t = -1** then,

$$w_{ik}(new)=w_{ik}(old)+\alpha(-1-Q_{ink})x_i \quad w_{ik}(new)=w_{ik}(old)+\alpha(-1-Q_{ink})x_i$$
$$b_k(new)=b_k(old)+\alpha(-1-Q_{ink}) \quad b_k(new)=b_k(old)+\alpha(-1-Q_{ink})$$

In this case, the weights would be updated on **Q_k** where the net input is positive because **t = -1**.

Here **'y'** is the actual output and **'t'** is the desired/target output.

**Case 3** − if **y = t** then

There would be no change in weights.

Step 8 − Test for the stopping condition, which will happen when there is no change in weight or the highest weight change occurred during training is smaller than the specified tolerance.

Back Propagation Neural Networks

Back Propagation Neural (BPN) is a multilayer neural network consisting of the input layer, at least one hidden layer and output layer. As its name suggests, back propagating will take place in this network. The error which is calculated at the output layer, by comparing the target output and the actual output, will be propagated back towards the input layer.

Architecture

As shown in the diagram, the architecture of BPN has three interconnected layers having weights on them. The hidden layer as well as the output layer also has bias, whose weight is always 1, on them. As is clear from the diagram, the working of BPN is in two phases. One phase sends the signal from the input layer to the output layer, and the other phase back propagates the error from the output layer to the input layer.
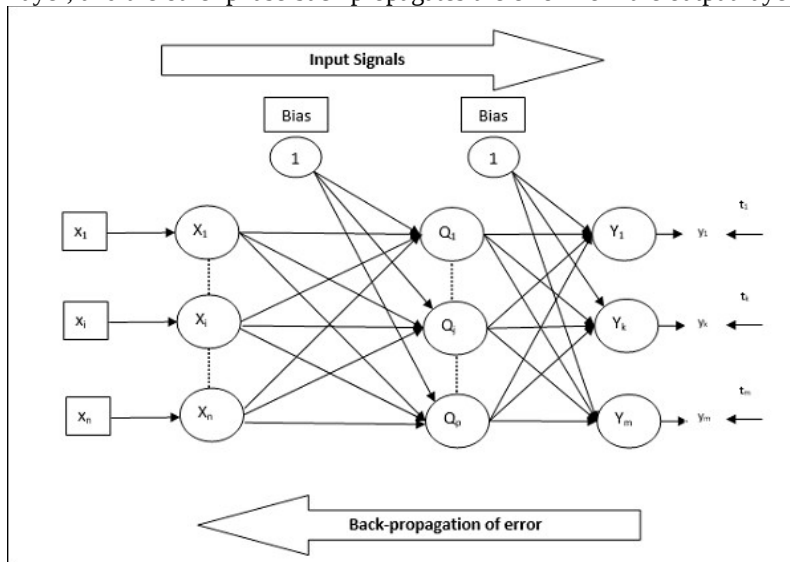
Training Algorithm

For training, BPN will use binary sigmoid activation function. The training of BPN will have the following three phases.

- **Phase 1** − Feed Forward Phase
- **Phase 2** − Back Propagation of error
- **Phase 3** − Updating of weights

All these steps will be concluded in the algorithm as follows

Step 1 − Initialize the following to start the training −

- Weights

- Learning rate α

For easy calculation and simplicity, take some small random values.

Step 2 − Continue step 3-11 when the stopping condition is not true.

Step 3 − Continue step 4-10 for every training pair.

Phase 1

Step 4 − Each input unit receives input signal x_i and sends it to the hidden unit for all **i = 1 to n**

Step 5 − Calculate the net input at the hidden unit using the following relation −

$$Q_{inj} = b_{0j} + \sum_{i=1}^{n} x_i v_{ij} \quad j = 1 \text{ to } p$$

Here b_{0j} is the bias on hidden unit, v_{ij} is the weight on **j** unit of the hidden layer coming from **i** unit of the input layer.

Now calculate the net output by applying the following activation function

$$Q_j = f(Q_{inj})$$

Send these output signals of the hidden layer units to the output layer units.

Step 6 − Calculate the net input at the output layer unit using the following relation −

$$y_{ink} = b_{0k} + \sum_{j=1}^{p} Q_j w_{jk} \quad k = 1 \text{ to } m$$

Here b_{0k} is the bias on output unit, w_{jk} is the weight on **k** unit of the output layer coming from **j** unit of the hidden layer.

Calculate the net output by applying the following activation function

$$y_k = f(y_{ink})$$

Phase 2

Step 7 − Compute the error correcting term, in correspondence with the target pattern received at each output unit, as follows −

$$\delta k = (t_k - y_k) f'(y_{ink})$$

On this basis, update the weight and bias as follows −

$$\Delta v_{jk} = \alpha \delta k Q_{ij}$$
$$\Delta b_{0k} = \alpha \delta k$$

Then, send δk back to the hidden layer.

Step 8 − Now each hidden unit will be the sum of its delta inputs from the output units.

$$\delta_{inj} = \sum_{k=1}^{m} \delta k w_{jk}$$

Error term can be calculated as follows −

$$\delta j = \delta_{inj} f'(Q_{inj})$$

On this basis, update the weight and bias as follows −

$$\Delta w_{ij} = \alpha \delta j x_i$$
$$\Delta b_{0j} = \alpha \delta j$$

Phase 3

Step 9 − Each output unit *(y_k k = 1 to m)* updates the weight and bias as follows −

$$v_{jk}(new) = v_{jk}(old) + \Delta v_{jk}$$
$$b_{0k}(new) = b_{0k}(old) + \Delta b_{0k}$$

Step 10 − Each output unit *(z_i j = 1 to p)* updates the weight and bias as follows −

$$w_{ij}(new) = w_{ij}(old) + \Delta w_{ij}$$
$$b_{0j}(new) = b_{0j}(old) + \Delta b_{0j}$$

Step 11 − Check for the stopping condition, which may be either the number of epochs reached or the target output matches the actual output.

Generalized Delta Learning Rule

Delta rule works only for the output layer. On the other hand, generalized delta rule, also called as **back-propagation** rule, is a way of creating the desired values of the hidden layer.

Mathematical Formulation

For the activation function $y_k = f(y_{ink})$ the derivation of net input on Hidden layer as well as on output layer can be given by

$$y_{ink} = \sum_i z_i w_{jk}$$

And $y_{inj} = \sum_i x_i v_{ij}$

Now the error which has to be minimized is

$$E = \frac{1}{2} \sum_k [t_k - y_k]^2$$

By using the chain rule, we have

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} \left(\frac{1}{2} \sum_k [t_k - y_k]^2 \right)$$
$$= \frac{\partial}{\partial w_{jk}} \left(\frac{1}{2} [t_k - t(y_{ink})]^2 \right)$$
$$= -[t_k - y_k] \frac{\partial}{\partial w_{jk}} f(y_{ink})$$
$$= -[t_k - y_k] f(y_{ink}) \frac{\partial}{\partial w_{jk}} (y_{ink})$$
$$= -[t_k - y_k] f'(y_{ink}) z_j$$

Now let us say $\delta k = -[t_k - y_k] f'(y_{ink})$

The weights on connections to the hidden unit z_j can be given by −

$$\partial E \partial vij = -\sum k \delta k \partial \partial vij(yink) \partial E \partial vij = -\sum k \delta k \partial \partial vij(yink)$$

Putting the value of yinkyink we will get the following

$$\delta j = -\sum k \delta kwjkf'(zinj)\delta j = -\sum k \delta kwjkf'(zinj)$$

Weight updating can be done as follows −

For the output unit −

$$\Delta wjk = -\alpha \partial E \partial wjk \Delta wjk = -\alpha \partial E \partial wjk$$
$$= \alpha \delta kzj = \alpha \delta kzj$$

For the hidden unit −

$$\Delta vij = -\alpha \partial E \partial vij \Delta vij = -\alpha \partial E \partial vij$$
$$= \alpha \delta jxi$$

AI v/s ANN.

Artificial intelligence (AI) and artificial neural networks (ANN) are two exciting and intertwined fields in computer science. There are, however, several differences between the two that are worth knowing about. The key difference is that neural networks are a stepping stone in the search for artificial intelligence.

Artificial intelligence is a vast field that has the goal of creating intelligent machines, something that has been achieved many times depending on how you define intelligence. Despite the fact that we have computers that can win at "Jeopardy" and beat chess champions, the goal of AI is generally seen as a quest for general intelligence, or intelligence that can be applied to diverse and unrelated situational problems. A grossly simplified example is the pain from getting burned. When this happens for the first time, a connection is made in your brain that identifies the sensory information known as fire (flames, smell of smoke, heat) and relates it with pain. This is how you learn, at a very young age, how to avoid getting burned. Through this same neural network, we can do a lot of general learning like "ice cream tastes good" and even make deductive leaps like "there are always clouds before rain" or "stocks always rally in December." These leaps are not always correct (there is bad ice cream and there are stocks that drop in December), but they can be corrected through experience, thus allowing adaptive learning.

Artificial neural networks try to recreate this learning system on computers by constructing a simple framework program to respond to a problem and receive feedback on how it does. A computer can optimize its response by doing the same problem thousands of times and adjusting its response according to the feedback it receives. The computer can then be given a different problem, which it can approach in the same way as it learned from the previous one. By varying the problems and the number of approaches to solving them that the computer has learned, computer scientists can teach a computer to be a generalist.

Although this conjures up images of computers taking over the world and harvesting humans as seen in Hollywood movies like "The Martrix," we are still a long way from neural networking our way to artificial intelligence. The problems being tested on neural networks are all expressed mathematically. You can't hold a flower up to a computer and tell it to guess the color by the smell, because the smell would have to be expressed in numbers and then the computer would have to catalog those numbers in memory, along with images of flowers emitting that smell.

That said, artificial neural networks that can be given more inputs of things like smell – and the capacity to learn from all those inputs – may be on track to produce the first artificial intelligence that meets the standards of even the most hardcore AI enthusiast.

In essence, artificial neural networks are models of human neural networks that are designed to help computers learn. Artificial intelligence is the Holy Grail some computer scientists are trying to achieve using techniques like mimicking neural networks.

Example: Here's the basic structure:
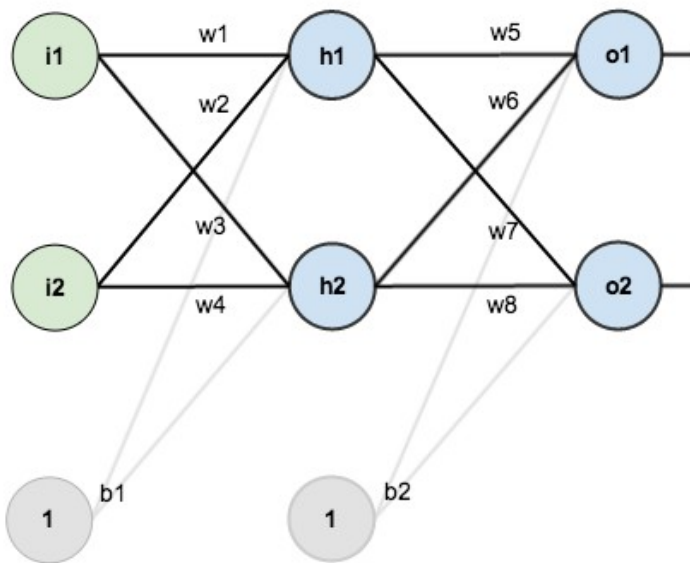
In order to have some numbers to work with, here are the initial weights, the biases, and training inputs/outputs:
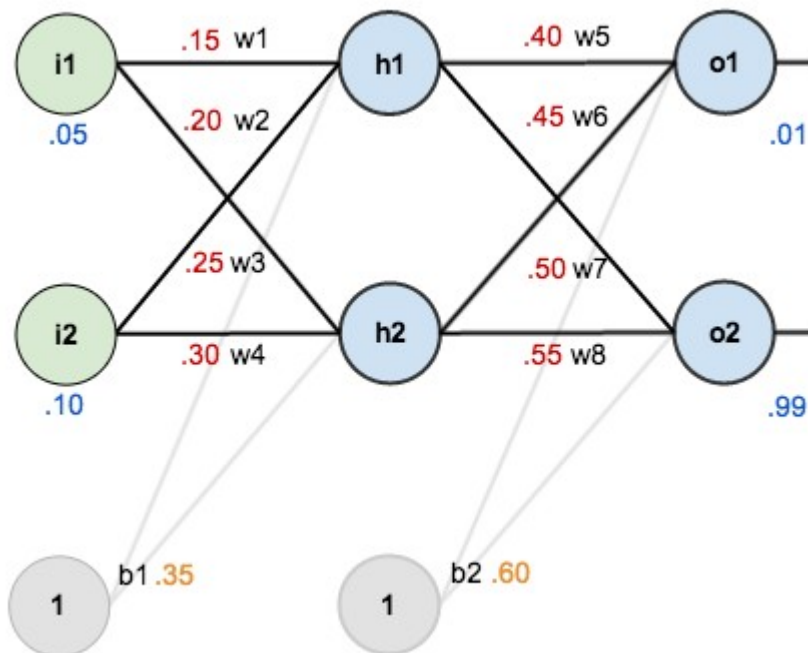
The goal of backpropagation is to optimize the weights so that the neural network can learn how to correctly map arbitrary inputs to outputs.

For the rest of this tutorial we're going to work with a single training set: given inputs 0.05 and 0.10, we want the neural network to output 0.01 and 0.99.

The Forward Pass

To begin, lets see what the neural network currently predicts given the weights and biases above and inputs of 0.05 and 0.10. To do this we'll feed those inputs forward though the network.

We figure out the *total net input* to each hidden layer neuron, *squash* the total net input using an *activation function* (here we use the *logistic function*), then repeat the process with the output layer neurons.

Total net input is also referred to as just *net input* by some sources.

Here's how we calculate the total net input for h_1:

$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$

$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$

We then squash it using the logistic function to get the output of h_1:

$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$

Carrying out the same process for h_2 we get:

$out_{h2} = 0.596884378$

We repeat this process for the output layer neurons, using the output from the hidden layer neurons as inputs.
Here's the output for o_1:

$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$

$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$

$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$

And carrying out the same process for o_2 we get:

$out_{o2} = 0.772928465$

Calculating the Total Error

We can now calculate the error for each output neuron using the [squared error function](#) and sum them to get the total error:

$E_{total} = \sum \frac{1}{2}(target - output)^2$

Some sources refer to the target as the *ideal* and the output as the *actual*.

The $\frac{1}{2}$ is included so that exponent is cancelled when we differentiate later on. The result is eventually multiplied by a learning rate anyway so it doesn't matter that we introduce a constant here [1].

For example, the target output for o_1 is 0.01 but the neural network output 0.75136507, therefore its error is:

$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$

Repeating this process for o_2 (remembering that the target is 0.99) we get:

$E_{o2} = 0.023560026$

The total error for the neural network is the sum of these errors:

$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$

The Backwards Pass

Our goal with backpropagation is to update each of the weights in the network so that they cause the actual output to be closer the target output, thereby minimizing the error for each output neuron and the network as a whole.
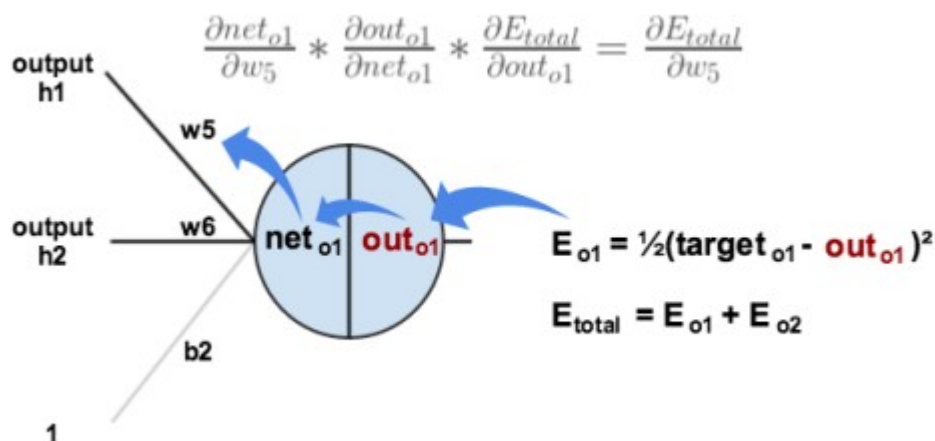
Output Layer

Consider w_5. We want to know how much a change in w_5 affects the total error, aka $\frac{\partial E_{total}}{\partial w_5}$.

$\frac{\partial E_{total}}{\partial w_5}$ is read as "the partial derivative of E_{total} with respect to w_5". You can also say "the gradient with respect to w_5".

By applying the [chain rule](#) we know that:

Visually, here's what we're doing:

$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$

$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2$

$E_{total} = E_{o1} + E_{o2}$

We need to figure out each piece in this equation.
First, how much does the total error change with respect to the output?

$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$

$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

Next, how much does the output of $o1$ change with respect to its total net input?

The partial [derivative of the logistic function](#) is the output multiplied by 1 minus the output:

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

Finally, how much does the total net input of $o1$ change with respect to w_5?

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$

Putting it all together:

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

To decrease the error, we then subtract this value from the current weight (optionally multiplied by some learning rate, eta, which we'll set to 0.5):

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

We can repeat this process to get the new weights w_6, w_7, and w_8:

$$w_6^+ = 0.408666186$$
$$w_7^+ = 0.511301270$$
$$w_8^+ = 0.561370121$$

We perform the actual updates in the neural network *after* we have the new weights leading into the hidden layer neurons (ie, we use the original weights, not the updated weights, when we continue the backpropagation algorithm below).

Hidden Layer

Next, we'll continue the backwards pass by calculating new values for w_1, w_2, w_3, and w_4.
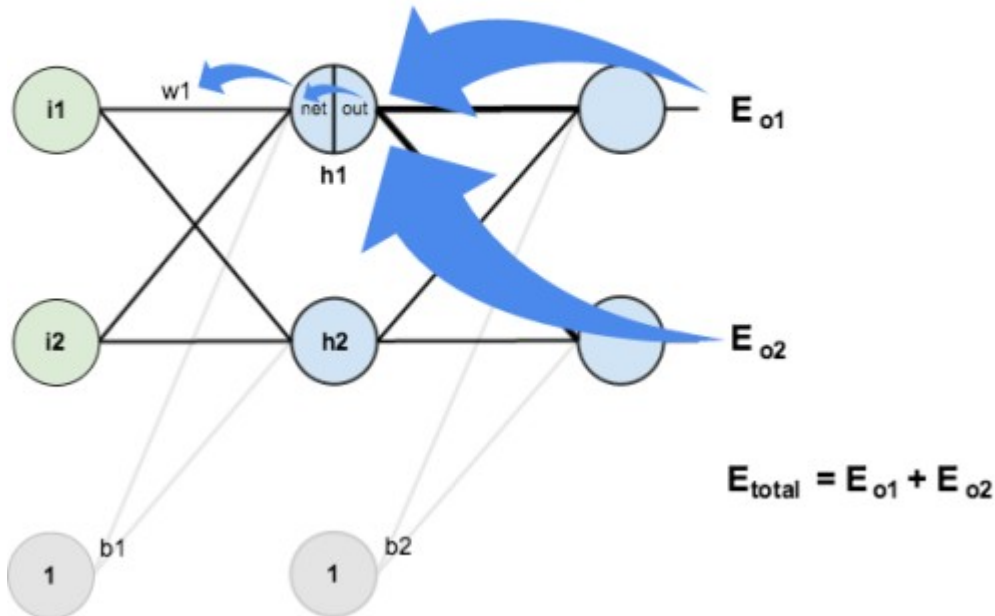
Big picture, here's what we need to figure out:

Visually:

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\downarrow$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

$$E_{total} = E_{o1} + E_{o2}$$

We're going to use a similar process as we did for the output layer, but slightly different to account for the fact that the output of each hidden layer neuron contributes to the output (and therefore error) of multiple output neurons. We know that out_{h1} affects both out_{o1} and out_{o2} therefore the $\frac{\partial E_{total}}{\partial out_{h1}}$ needs to take into consideration its effect on the both output neurons:

Starting with $\frac{\partial E_{o1}}{\partial out_{h1}}$:

We can calculate $\frac{\partial E_{o1}}{\partial net_{o1}}$ using values we calculated earlier:

And $\frac{\partial net_{o1}}{\partial out_{h1}}$ is equal to w_5:
$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$
$$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.40$$
Plugging them in:

Following the same process for $\frac{\partial E_{o2}}{\partial out_{h1}}$, we get:
$$\frac{\partial E_{o2}}{\partial out_{h1}} = -0.019049119$$
Therefore:

Now that we have $\frac{\partial E_{total}}{\partial out_{h1}}$, we need to figure out $\frac{\partial out_{h1}}{\partial net_{h1}}$ and then $\frac{\partial net_{h1}}{\partial w}$ for each weight:
$$out_{h1} = \frac{1}{1+e^{-net_{h1}}}$$
$$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1 - out_{h1}) = 0.59326999(1 - 0.59326999) = 0.241300709$$
We calculate the partial derivative of the total net input to h_1 with respect to w_1 the same as we did for the output neuron:
$$net_{h1} = w_1 * i_1 + w_3 * i_2 + b_1 * 1$$
$$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$$

Putting it all together:

$$\frac{\partial E_{total}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$$

You might also see this written as:

$$\frac{\partial E_{total}}{\partial w_1} = \left(\sum_o \delta_o * w_{ho} \right) * out_{h1}(1 - out_{h1}) * i_1$$

$$\frac{\partial E_{total}}{\partial w_1} = \delta_{h1} i_1$$

We can now update w_1:

$$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$$

Repeating this for w_2, w_3, and w_4

$$w_2^+ = 0.19956143$$
$$w_3^+ = 0.24975114$$
$$w_4^+ = 0.29950229$$

Finally, we've updated all of our weights! When we fed forward the 0.05 and 0.1 inputs originally, the error on the network was 0.298371109. After this first round of backpropagation, the total error is now down to 0.291027924. It might not seem like much, but after repeating this process 10,000 times, for example, the error plummets to 0.0000351085. At this point, when we feed forward 0.05 and 0.1, the two outputs neurons generate 0.015912196 (vs 0.01 target) and 0.984065734 (vs 0.99 target).

limitation, characteristics back propagation

Gradient descent with backpropagation is not guaranteed to find the global minimum of the error function, but only a local minimum; also, it has trouble crossing plateaus in the error function landscape. This issue, caused by the non-convexity of error functions in neural networks, was long thought to be a major drawback, but Yann LeCun et al. argue that in many practical problems, it is not.[7]

Backpropagation learning does not require normalization of input vectors; however, normalization could improve performance

Application of EBPA.

As an elective for my Bachelor's degree, I took a graduate-level class in Neural Networks and found it to be extremely exciting. In one of the final assignments, we were individually asked to apply and evaluate back-propagation in solving several types of problems that include classification, function estimation, and time-series prediction.

MATLAB's ability to efficiently calculate vectors made it the tool of choice in building the "Back Prop" framework. I then built a series of tests to evaluate the effectiveness of "Back Prop" configurations for each specific problem. Although the goal was to ultimately solve each problem, setting-up and refactoring the problem to work under the constraints of neural networks was also a rewarding task in itself.

Application #1: Classification

In this classification problem, the goal is to identify whether a certain "data point" belongs to Class 1, 2, or 3 (see above). Random points are assigned to a certain class, and the neural network is trained to find the pattern. When training is complete, it will use what it has learned to accurately classify new points.
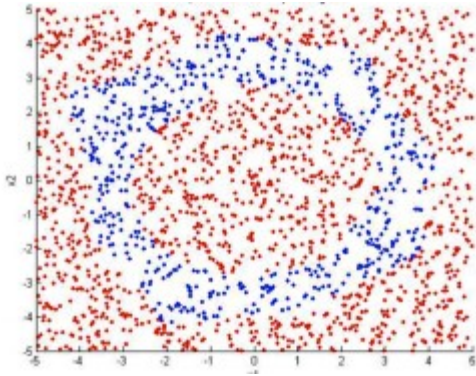
Here, the network was able to distinguish group 1 (red) from group 2 (blue).
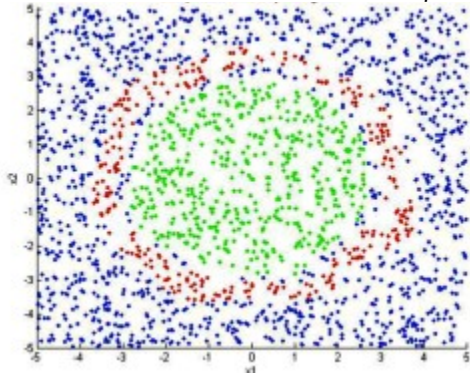
Here, the system was trained to identify three groups.

Application #2: Function Approximation

In this problem, the network tries to approximate the value of a certain function. It is fed with noisy data, and the goal is to find the true pattern. After training, the network successfully estimates the value of the gaussian function (below).
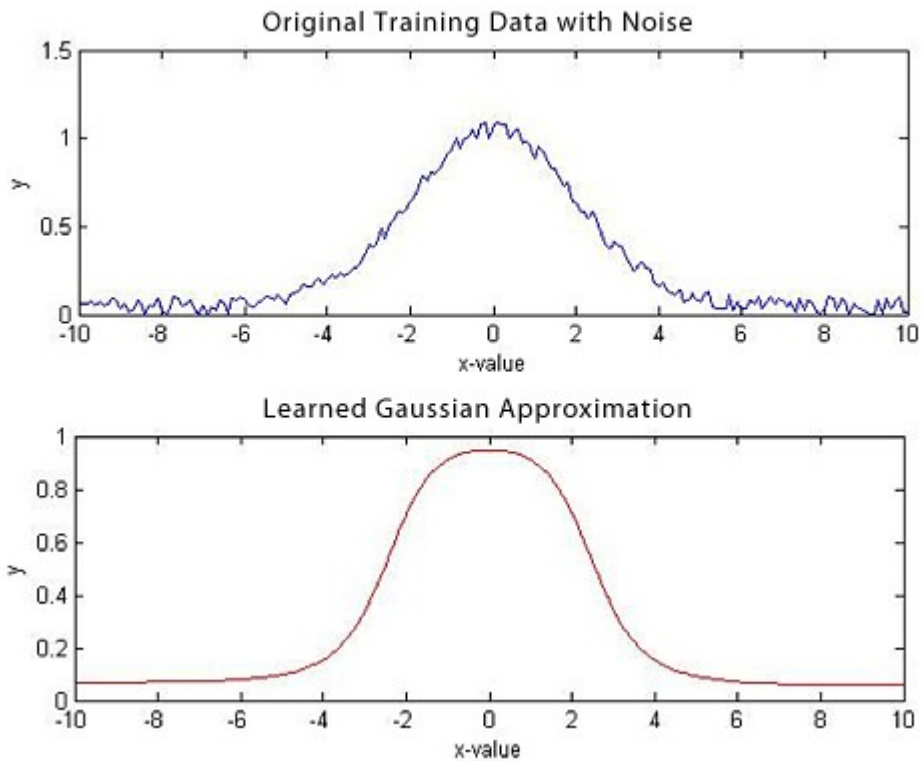
Estimating the Gaussian function

Application #3: Time-series Prediction

In this problem, the goal is to design a neural network to predict a value based on a given time-series data (i.e. stock market prediction based on given trends). To approach this problem, the inputs to the neural network have

to be refactored in chunks, and the resulting output will be the next data item directly following that chunk (see below)
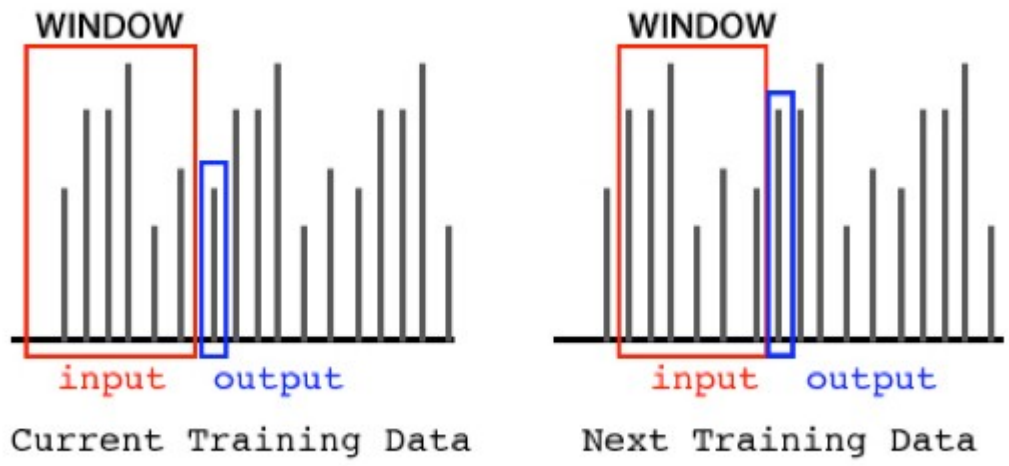
The input space and output space for the time-series prediction problem.

In this specific problem, goal was to predict time-series data based on a sine wave. After training the network, lo' and behold system was able to accurately predict 1000 data points for the sine wave. The results are seen below.
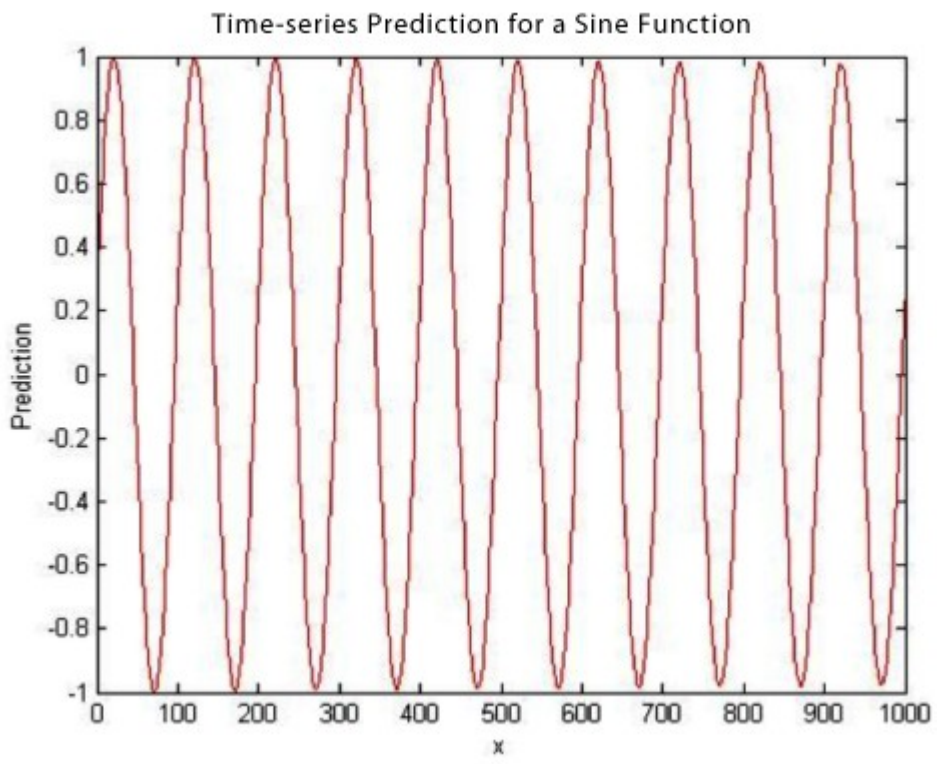
1000 sine wave predictions for the trained network.

Besides acquiring a fascination with Neural Networks, learning and interacting with the brilliant graduate students in the class also made it a worthwhile experience.

.

Adaptive Resonance Theory: Architecture, classifications, Implementation and training, Associative Memory.

his network was developed by Stephen Grossberg and Gail Carpenter in 1987. It is based on competition and uses unsupervised learning model. Adaptive Resonance Theory (ART) networks, as the name suggests, is always open to new learning (adaptive) without losing the old patterns (resonance). Basically, ART network is a vector classifier which accepts an input vector and classifies it into one of the categories depending upon which of the stored pattern it resembles the most.

Operating Principal

The main operation of ART classification can be divided into the following phases −

- **Recognition phase** − The input vector is compared with the classification presented at every node in the output layer. The output of the neuron becomes "1" if it best matches with the classification applied, otherwise it becomes "0".
- **Comparison phase** − In this phase, a comparison of the input vector to the comparison layer vector is done. The condition for reset is that the degree of similarity would be less than vigilance parameter.
- **Search phase** − In this phase, the network will search for reset as well as the match done in the above phases. Hence, if there would be no reset and the match is quite good, then the classification is over. Otherwise, the process would be repeated and the other stored pattern must be sent to find the correct match.

ART1

It is a type of ART, which is designed to cluster binary vectors. We can understand about this with the architecture of it.
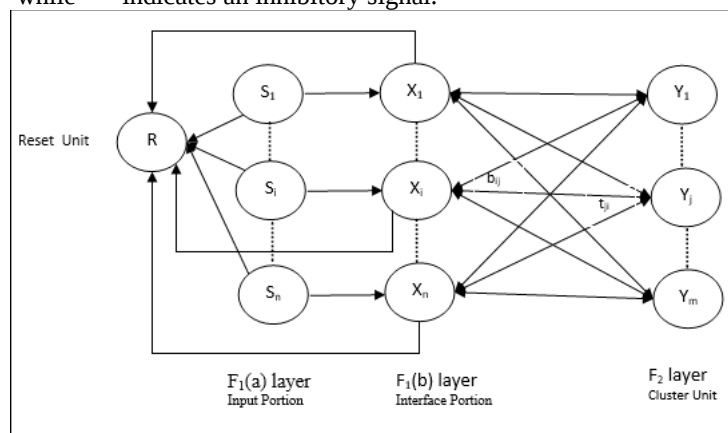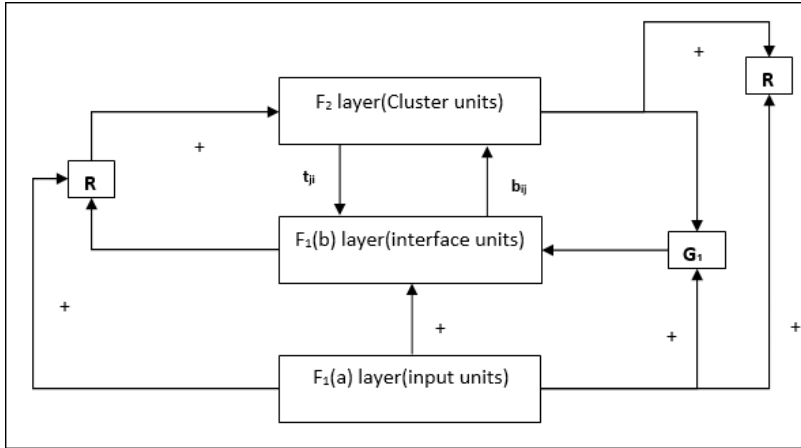
Architecture of ART1

It consists of the following two units −

Computational Unit − It is made up of the following −

- **Input unit (F_1 layer)** − It further has the following two portions −
  - o **F_1(a) layer (Input portion)** − In ART1, there would be no processing in this portion rather than having the input vectors only. It is connected to F_1(b) layer (interface portion).
  - o **F_1(b) layer (Interface portion)** − This portion combines the signal from the input portion with that of F_2 layer. F_1(b) layer is connected to F_2 layer through bottom up weights b_{ij} and F_2layer is connected to F_1(b) layer through top down weights t_{ji}.
- **Cluster Unit (F_2 layer)** − This is a competitive layer. The unit having the largest net input is selected to learn the input pattern. The activation of all other cluster unit are set to 0.
- **Reset Mechanism** − The work of this mechanism is based upon the similarity between the top-down weight and the input vector. Now, if the degree of this similarity is less than the vigilance parameter, then the cluster is not allowed to learn the pattern and a rest would happen.

Supplement Unit − Actually the issue with Reset mechanism is that the layer F_2 must have to be inhibited under certain conditions and must also be available when some learning happens. That is why two supplemental units namely, G_1 and G_2 is added along with reset unit, **R**. They are called **gain control units**. These units receive and send signals to the other units present in the network. **'+'** indicates an excitatory signal, while **'−'** indicates an inhibitory signal.

Reset Unit

F$_1$(a) layer — Input Portion   F$_1$(b) layer — Interface Portion   F$_2$ layer — Cluster Unit

Parameters Used

Following parameters are used −

- **n** − Number of components in the input vector
- **m** − Maximum number of clusters that can be formed
- **b_{ij}** − Weight from $F_1(b)$ to F_2 layer, i.e. bottom-up weights
- **t_{ji}** − Weight from F_2 to $F_1(b)$ layer, i.e. top-down weights
- **ρ** − Vigilance parameter
- **∥x∥** − Norm of vector x

Algorithm

Step 1 − Initialize the learning rate, the vigilance parameter, and the weights as follows −

$$\alpha > 1 \text{ and } 0 < \rho \leq 1 \alpha > 1 \text{ and } 0 < \rho \leq 1$$
$$0 < b_{ij}(0) < \frac{\alpha}{\alpha - 1 + n} \text{ and } t_{ij}(0) = 1 0 < b_{ij}(0) < \frac{\alpha}{\alpha - 1 + n} \text{ and } t_{ij}(0) = 1$$

Step 2 − Continue step 3-9, when the stopping condition is not true.

Step 3 − Continue step 4-6 for every training input.

Step 4 − Set activations of all $F_1(a)$ and F_1 units as follows

$F_2 = 0$ and $F_1(a)$ = input vectors

Step 5 − Input signal from $F_1(a)$ to $F_1(b)$ layer must be sent like

$$s_i = x_i s_i = x_i$$

Step 6 − For every inhibited F_2 node

$y_j = \sum_i b_{ij} x_i y_j = \sum_i b_{ij} x_i$ the condition is **$y_j \neq -1$**

Step 7 − Perform step 8-10, when the reset is true.

Step 8 − Find **J** for **$y_J \geq y_j$** for all nodes **j**

Step 9 − Again calculate the activation on $F_1(b)$ as follows

$$x_i = s_i t_{Ji} x_i = s_i t_{Ji}$$

Step 10 − Now, after calculating the norm of vector **x** and vector **s**, we need to check the reset condition as follows −

If ∥**x**∥/ ∥**s**∥ < vigilance parameter **ρ**, then inhibit node **J** and go to step 7

Else If ∥**x**∥/ ∥**s**∥ ≥ vigilance parameter **ρ**, then proceed further.

Step 11 − Weight updating for node **J** can be done as follows −

$$b_{ij}(new) = \frac{\alpha x_i}{\alpha - 1 + \|x\|} b_{ij}(new) = \frac{\alpha x_i}{\alpha - 1 + \|x\|}$$
$$t_{ij}(new) = x_i t_{ij}(new) = x_i$$

Step 12 − The stopping condition for algorithm must be checked and it may be as follows −

- Do not have any change in weight.
- Reset is not performed for units.
- Maximum number of epochs reached.

Soft Computing: Module-IV

Fuzzy Logic: Fuzzy set theory

Fuzzy Logic | Introduction

The term fuzzy refers to things which are not clear or are vague. In the real world many times we encounter a situation when we can't determine whether the state is true or false, their fuzzy logic provides a very valuable flexibility for reasoning. In this way, we can consider the inaccuracies and uncertainties of any situation.

In boolean system truth value, 1.0 represents absolute truth value and 0.0 represents absolute false value. But in the fuzzy system, there is no logic for absolute truth and absolute false value. But in fuzzy logic, there is intermediate value too present which is partially true and partially false.

Architecture

Its Architecture contains four parts :

RULE BASE: It contains the aet of rules and the IF-THEN conditions provided by the experts to govern the decision making system, on the basis of linguistic information. Recent developments in fuzzy theory offer several effective methods for the design and tuning of fuzzy controllers. Most of these developments reduce the number of fuzzy rules.

FUZZIFICATION: It is used to convert inputs i.e. crisp numbers into fuzzy sets. Crisp inputs are basically the exact inputs measured by sensors and passed into the control system for processing, such as temperature, pressure, rpm's, etc.

INFERENCE ENGINE: It determines the matching degree of the current fuzzy input with respect to each rule and decides which rules are to be fired according to the input field. Next, the fired rules are combined to form the control actions.

DEFUZZIFICATION: It is used to convert the fuzzy sets obtained by inference engine into a crisp value. There are several defuzzification methods available and the best suited one is used with a specific expert system to reduce the error.

Membership function

Definition: A graph that defines how each point in the input space is mapped to membership value between 0 and 1. Input space is often referred as the universe of discourse or universal set (u), which contain all the possible elements of concern in each particular application.

There are largely three types of fuzzifiers:
singleton fuzzifier,
Gaussian fuzzifier, and
trapezoidal or triangular fuzzifier
What is Fuzzy Control?

It is a technique to embody human-like thinkings into a control system.
It may not be designed to give accurate reasoning but it is designed to give acceptable reasoning.
It can emulate human deductive thinking, that is, the process people use to infer conclusions from what they know.
Any uncertainties can be easily dealt with the help of fuzzy logic.
Advantages of Fuzzy Logic System

This system can work with any type of inputs whether it is imprecise, distorted or noisy input information.
The construction of Fuzzy Logic Systems is easy and understandable.
Fuzzy logic comes with mathematical concepts of set theory and the reasoning of that is quite simple.
It provides a very efficient solution to complex problems in all fields of life as it resembles human reasoning and decision making.
The algorithms can be described with little data, so little memory is required.
Disadvantages of Fuzzy Logic Systems

Many researchers proposed different ways to solve a given problem through fuzzy logic which lead to ambiguity.There is no systematic approach to solve a given problem through fuzzy logic.

Proof of its characteristics is difficult or impossible in most cases because every time we do not get mathematical description of our approach.

As fuzzy logic works on precise as well as imprecise data so most of the time accuracy iscompromised.

Application of FL

It is used in the aerospace field for altitude control of spacecraft and satellite.

It has used in the automotive system for speed control, traffic control.

It is used for decision making support systems and personal evaluation in the large company business.

It has application in chemical industry for controlling the pH, drying, chemical distillation process.

Fuzzy logic are used in Natural language processing and various intensive applications in Artificial Intelligence.

Fuzzy logic are extensively used in modern control systems such as expert systems.

Fuzzy Logic is used with Neural Networks as it mimics how a person would make decisions, only much faster. It is done by Aggregation of data and changing into more meaningful data by forming partial truths as Fuzzy sets.

Fuzzy set versus crisp set

Fuzzy logic is a form of many-valued logic in which the truth values of variables may be any real number between 0 and 1 inclusive. It is employed to handle the concept of partial truth, where the truth value may range between completely true and completely false. By contrast, in Boolean logic, the truth values of variables may only be the integer values 0 or 1.

The term *fuzzy logic* was introduced with the 1965 proposal of fuzzy set theory by Lotfi Zadeh. Fuzzy logic had however been studied since the 1920s, as infinite-valued logic—notably by Łukasiewicz and Tarski.

It is based on the observation that people make decisions based on imprecise and non-numerical information, fuzzy models or sets are mathematical means of representing vagueness and imprecise information, hence the term fuzzy. These models have the capability of recognising, representing, manipulating, interpreting, and utilising data and information that are vague and lack certainty.

Fuzzy logic has been applied to many fields, from control theory to artificial intelligence.

A **set** is an unordered collection of different elements. It can be written explicitly by listing its elements using the set bracket. If the order of the elements is changed or any element of a set is repeated, it does not make any changes in the set.

Example

- A set of all positive integers.
- A set of all the planets in the solar system.
- A set of all the states in India.
- A set of all the lowercase letters of the alphabet.

Mathematical Representation of a Set

Sets can be represented in two ways −

Roster or Tabular Form

In this form, a set is represented by listing all the elements comprising it. The elements are enclosed within braces and separated by commas.

Following are the examples of set in Roster or Tabular Form −

- Set of vowels in English alphabet, A = {a,e,i,o,u}
- Set of odd numbers less than 10, B = {1,3,5,7,9}

Set Builder Notation

In this form, the set is defined by specifying a property that elements of the set have in common. The set is described as A = {x:p(x)}

Example 1 − The set {a,e,i,o,u} is written as

$$A = \{x:x \text{ is a vowel in English alphabet}\}$$

Example 2 − The set {1,3,5,7,9} is written as

B = {x:1 ≤ x < 10 and (x%2) ≠ 0}

If an element x is a member of any set S, it is denoted by x ∈ S and if an element y is not a member of set S, it is denoted by y ∉ S.

Example − If S = {1,1.2,1.7,2},1 ∈ S but 1.5 ∉ S

Cardinality of a Set

Cardinality of a set S, denoted by $|S|$, is the number of elements of the set. The number is also referred as the cardinal number. If a set has an infinite number of elements, its cardinality is ∞.

Example − $|\{1,4,3,5\}| = 4, |\{1,2,3,4,5,\ldots\}| = \infty$

If there are two sets X and Y, $|X| = |Y|$ denotes two sets X and Y having same cardinality. It occurs when the number of elements in X is exactly equal to the number of elements in Y. In this case, there exists a bijective function 'f' from X to Y.

$|X| \leq |Y|$ denotes that set X's cardinality is less than or equal to set Y's cardinality. It occurs when the number of elements in X is less than or equal to that of Y. Here, there exists an injective function 'f' from X to Y.

$|X| < |Y|$ denotes that set X's cardinality is less than set Y's cardinality. It occurs when the number of elements in X is less than that of Y. Here, the function 'f' from X to Y is injective function but not bijective.

If $|X| \leq |Y|$ and $|X| \leq |Y|$ then $|X| = |Y|$. The sets X and Y are commonly referred as **equivalent sets**.

Types of Sets

Sets can be classified into many types; some of which are finite, infinite, subset, universal, proper, singleton set, etc.

Finite Set

A set which contains a definite number of elements is called a finite set.

Example − S = {x|x ∈ N and 70 > x > 50}

Infinite Set

A set which contains infinite number of elements is called an infinite set.

Example − S = {x|x ∈ N and x > 10}

Subset

A set X is a subset of set Y (Written as X ⊆ Y) if every element of X is an element of set Y.

Example 1 − Let, X = {1,2,3,4,5,6} and Y = {1,2}. Here set Y is a subset of set X as all the elements of set Y is in set X. Hence, we can write Y ⊆ X.

Example 2 − Let, X = {1,2,3} and Y = {1,2,3}. Here set Y is a subset (not a proper subset) of set X as all the elements of set Y is in set X. Hence, we can write Y ⊆ X.

Proper Subset

The term "proper subset" can be defined as "subset of but not equal to". A Set X is a proper subset of set Y (Written as X ⊂ Y) if every element of X is an element of set Y and $|X| < |Y|$.

Example − Let, X = {1,2,3,4,5,6} and Y = {1,2}. Here set Y ⊂ X, since all elements in Y are contained in X too and X has at least one element which is more than set Y.

Universal Set

It is a collection of all elements in a particular context or application. All the sets in that context or application are essentially subsets of this universal set. Universal sets are represented as U.

Example − We may define U as the set of all animals on earth. In this case, a set of all mammals is a subset of U, a set of all fishes is a subset of U, a set of all insects is a subset of U, and so on.

Empty Set or Null Set

An empty set contains no elements. It is denoted by Φ. As the number of elements in an empty set is finite, empty set is a finite set. The cardinality of empty set or null set is zero.

Example – S = {x|x ∈ N and 7 < x < 8} = Φ

Singleton Set or Unit Set

A Singleton set or Unit set contains only one element. A singleton set is denoted by {s}.

Example − S = {x|x ∈ N, 7 < x < 9} = {8}

Equal Set

If two sets contain the same elements, they are said to be equal.

Example − If A = {1,2,6} and B = {6,1,2}, they are equal as every element of set A is an element of set B and every element of set B is an element of set A.

Equivalent Set

If the cardinalities of two sets are same, they are called equivalent sets.

Example − If A = {1,2,6} and B = {16,17,22}, they are equivalent as cardinality of A is equal to the cardinality of B. i.e. $|A| = |B| = 3$

Overlapping Set

Two sets that have at least one common element are called overlapping sets. In case of overlapping sets −

$$n(A \cup B) = n(A) + n(B) - n(A \cap B)$$
$$n(A \cup B) = n(A-B) + n(B-A) + n(A \cap B)$$
$$n(A) = n(A-B) + n(A \cap B)$$
$$n(B) = n(B-A) + n(A \cap B)$$

Example − Let, A = {1,2,6} and B = {6,12,42}. There is a common element '6', hence these sets are overlapping sets.

Disjoint Set
Two sets A and B are called disjoint sets if they do not have even one element in common. Therefore, disjoint sets have the following properties −

$$n(A \cap B) = \phi \ n(A \cap B) = \phi$$
$$n(A \cup B) = n(A) + n(B) \ n(A \cup B) = n(A) + n(B)$$

Example − Let, A = {1,2,6} and B = {7,9,14}, there is not a single common element, hence these sets are overlapping sets.

Operations on Classical Sets

Set Operations include Set Union, Set Intersection, Set Difference, Complement of Set, and Cartesian Product.
Union
The union of sets A and B (denoted by A ∪ BA ∪ B) is the set of elements which are in A, in B, or in both A and B. Hence, A ∪ B = {x|x ∈ A OR x ∈ B}.
Example − If A = {10,11,12,13} and B = {13,14,15}, then A ∪ B = {10,11,12,13,14,15} − The common element occurs only once.

A ∪ B
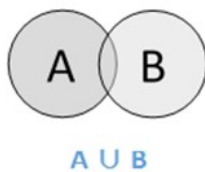
Intersection
The intersection of sets A and B (denoted by A ∩ B) is the set of elements which are in both A and B. Hence, A ∩ B = {x|x ∈ A AND x ∈ B}.

A ∩ B
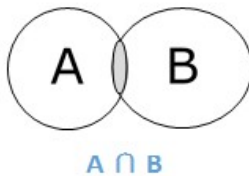
Difference/ Relative Complement
The set difference of sets A and B (denoted by A–B) is the set of elements which are only in A but not in B. Hence, A − B = {x|x ∈ A AND x ∉ B}.
Example − If A = {10,11,12,13} and B = {13,14,15}, then (A − B) = {10,11,12} and (B − A) = {14,15}. Here, we can see (A − B) ≠ (B − A)

A - B   B - A

Complement of a Set
The complement of a set A (denoted by A′) is the set of elements which are not in set A. Hence, A′ = {x|x ∉ A}.
More specifically, A′ = (U−A) where U is a universal set which contains all objects.
Example − If A = {x|x belongs to set of add integers} then A′ = {y|y does not belong to set of odd integers}

Complement of set A
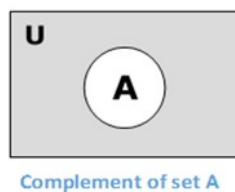
Cartesian Product / Cross Product
The Cartesian product of n number of sets A1,A2,…An denoted as A1 × A2...× An can be defined as all possible ordered pairs (x1,x2,…xn) where x1 ∈ A1,x2 ∈ A2,…xn ∈ An
Example − If we take two sets A = {a,b} and B = {1,2},

The Cartesian product of A and B is written as − A × B = {(a,1),(a,2),(b,1),(b,2)}
And, the Cartesian product of B and A is written as − B × A = {(1,a),(1,b),(2,a),(2,b)}

Properties of Classical Sets

Properties on sets play an important role for obtaining the solution. Following are the different properties of classical sets −

Commutative Property

Having two sets **A** and **B**, this property states −

$$A \cup B = B \cup A \quad A \cup B = B \cup A$$
$$A \cap B = B \cap A \quad A \cap B = B \cap A$$

Associative Property

Having three sets **A**, **B** and **C**, this property states −

$$A \cup (B \cup C) = (A \cup B) \cup C \quad A \cup (B \cup C) = (A \cup B) \cup C$$
$$A \cap (B \cap C) = (A \cap B) \cap C \quad A \cap (B \cap C) = (A \cap B) \cap C$$

Distributive Property

Having three sets **A**, **B** and **C**, this property states −

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C) \quad A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$
$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C) \quad A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

Idempotency Property

For any set **A**, this property states −

$$A \cup A = A \quad A \cup A = A$$
$$A \cap A = A \quad A \cap A = A$$

Identity Property

For set **A** and universal set **X**, this property states −

$$A \cup \varphi = A \quad A \cup \varphi = A$$
$$A \cap X = A \quad A \cap X = A$$
$$A \cap \varphi = \varphi \quad A \cap \varphi = \varphi$$
$$A \cup X = X \quad A \cup X = X$$

Transitive Property

Having three sets **A**, **B** and **C**, the property states −

$$\text{If } A \subseteq B \subseteq C \quad A \subseteq B \subseteq C, \text{ then } A \subseteq C \quad A \subseteq C$$

Involution Property

For any set **A**, this property states −

$$\overline{\overline{A}} = A \quad \overline{\overline{A}} = A$$

De Morgan's Law

It is a very important law and supports in proving tautologies and contradiction. This law states −

$$\overline{A \cap B} = \overline{A} \cup \overline{B} \quad \overline{A \cap B} = \overline{A} \cup \overline{B}$$
$$\overline{A \cup B} = \overline{A} \cap \overline{B}$$

Fuzzy sets can be considered as an extension and gross oversimplification of classical sets. It can be best understood in the context of set membership. Basically it allows partial membership which means that it contain elements that have varying degrees of membership in the set. From this, we can understand the difference between classical set and fuzzy set. Classical set contains elements that satisfy precise properties of membership while fuzzy set contains elements that satisfy imprecise properties of membership.
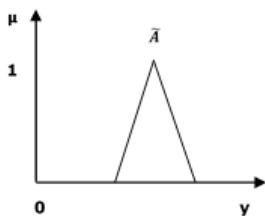
Membership Function of Fuzzy set $\tilde{A}$ Membership Function of classical set A
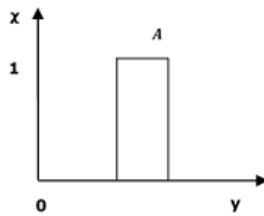
Mathematical Concept

A fuzzy set $\tilde{A}$ $A\sim$ in the universe of information U U can be defined as a set of ordered pairs and it can be represented mathematically as −

$$\tilde{A} = \{(y, \mu_{\tilde{A}}(y)) | y \in U\} \quad A\sim = \{(y, \mu_{A\sim}(y)) | y \in U\}$$

Here $\mu_{\tilde{A}}(y)$ $\mu_{A\sim}(y)$ = degree of membership of y y in \widetilde{A}, assumes values in the range from 0 to 1, i.e., $\mu_{\tilde{A}}(y) \in [0,1]$ $\mu_{A\sim}(y) \in [0,1]$.

Representation of fuzzy set

Let us now consider two cases of universe of information and understand how a fuzzy set can be represented.

Case 1

 When universe of information UU is discrete and finite −

$$\tilde{A}=\{\mu\tilde{A}(y1)y1+\mu\tilde{A}(y2)y2+\mu\tilde{A}(y3)y3+...\}A\sim=\{\mu A\sim(y1)y1+\mu A\sim(y2)y2+\mu A\sim(y3)y3+...\}$$
$$=\{\sum ni=1\mu\tilde{A}(yi)yi\}=\{\sum i=1n\mu A\sim(yi)yi\}$$

Case 2

 When universe of information UU is continuous and infinite −

$$\tilde{A}=\{\int\mu\tilde{A}(y)y\}A\sim=\{\int\mu A\sim(y)y\}$$

 In the above representation, the summation symbol represents the collection of each element.

Operations on Fuzzy Sets

 Having two fuzzy sets A˜A~ and B˜B~, the universe of information UU and an element y of the universe, the following relations express the union, intersection and complement operation on fuzzy sets.

Union/Fuzzy 'OR'

 Let us consider the following representation to understand how the **Union/Fuzzy 'OR'** relation works −

$$\mu\tilde{A}\cup\tilde{B}(y)=\mu\tilde{A}\vee\mu\tilde{B}\forall y\in U\mu A\sim\cup B\sim(y)=\mu A\sim\vee\mu B\sim\forall y\in U$$

 Here ∨ represents the 'max' operation.

Fuzzy set $\tilde{A}$ Fuzzy set $\tilde{B}$ Union of two Fuzzy sets
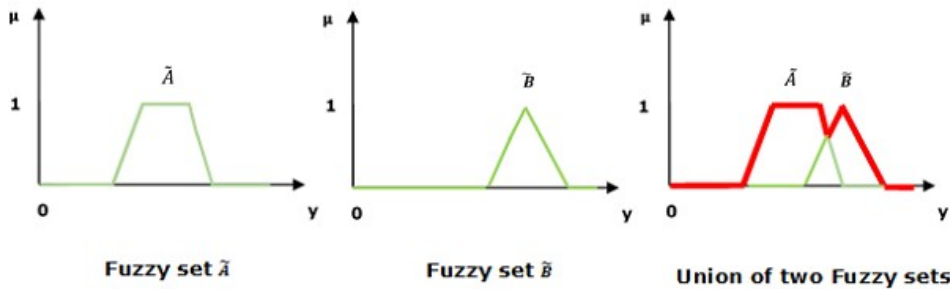
Intersection/Fuzzy 'AND'

 Let us consider the following representation to understand how the **Intersection/Fuzzy 'AND'** relation works −

$$\mu\tilde{A}\cap\tilde{B}(y)=\mu\tilde{A}\wedge\mu\tilde{B}\forall y\in U\mu A\sim\cap B\sim(y)=\mu A\sim\wedge\mu B\sim\forall y\in U$$

 Here ∧ represents the 'min' operation.

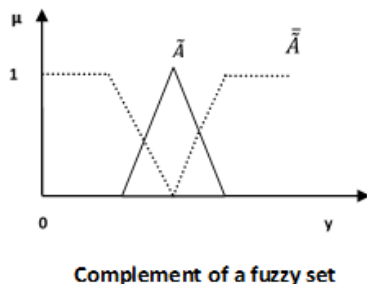Fuzzy set $\tilde{A}$ Fuzzy set $\tilde{B}$ Intersection of two Fuzzy sets
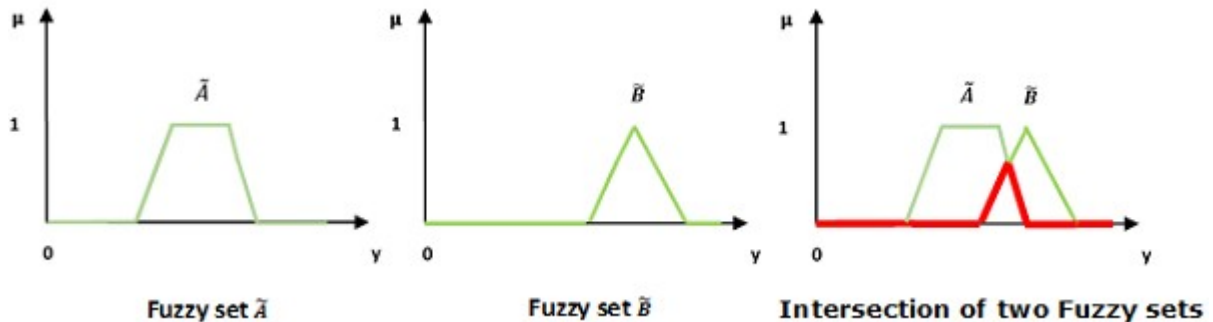
Complement/Fuzzy 'NOT'

 Let us consider the following representation to understand how the **Complement/Fuzzy 'NOT'** relation works −

$$\mu\tilde{A}=1-\mu\tilde{A}(y)y\in U\mu A\sim=1-\mu A\sim(y)y\in U$$

Complement of a fuzzy set

Properties of Fuzzy Sets

Let us discuss the different properties of fuzzy sets.

Commutative Property

Having two fuzzy sets A˜A~ and B˜B~, this property states −

$$A˜ \cup B˜=B˜ \cup A˜A~ \cup B~=B~ \cup A~$$
$$A˜\cap B˜=B˜\cap A˜A~\cap B~=B~\cap A~$$

Associative Property

Having three fuzzy sets A˜A~, B˜B~ and C˜C~, this property states −

$$A˜ \cup (B˜ \cup C˜)=(A˜ \cup B˜) \cup C˜A~ \cup (B~ \cup C~)=(A~ \cup B~) \cup C~$$
$$A˜\cap(B˜\cap C˜)=(A˜ \cup B˜) \cup C˜A~\cap(B~\cap C~)=(A~ \cup B~) \cup C~$$

Distributive Property

Having three fuzzy sets A˜A~, B˜B~ and C˜C~, this property states −

$$A˜ \cup (B˜\cap C˜)=(A˜ \cup B˜)\cap(A˜ \cup C˜)A~ \cup (B~\cap C~)=(A~ \cup B~)\cap(A~ \cup C~)$$
$$A˜\cap(B˜ \cup C˜)=(A˜\cap B˜) \cup (A˜\cap C˜)A~\cap(B~ \cup C~)=(A~\cap B~) \cup (A~\cap C~)$$

Idempotency Property

For any fuzzy set A˜A~, this property states −

$$A˜ \cup A˜=A˜A~ \cup A~=A~$$
$$A˜\cap A˜=A˜A~\cap A~=A~$$

Identity Property

For fuzzy set A˜A~ and universal set UU, this property states −

$$A˜ \cup \varphi=A˜A~ \cup \varphi=A~$$
$$A˜\cap U=A˜A~\cap U=A~$$
$$A˜\cap\varphi=\varphi A~\cap\varphi=\varphi$$
$$A˜ \cup U=UA~ \cup U=U$$

Transitive Property

Having three fuzzy sets A˜A~, B˜B~ and C˜C~, this property states −

$$IfA˜ \subseteq B˜ \subseteq C˜,thenA˜ \subseteq C˜IfA~ \subseteq B~ \subseteq C~,thenA~ \subseteq C~$$

Involution Property

For any fuzzy set A˜A~, this property states −

$$\overline{\overline{A˜}}=A˜\overline{\overline{A~}}=A~$$

De Morgan's Law

This law plays a crucial role in proving tautologies and contradiction. This law states −

$$\overline{A˜\cap B˜}=\overline{A˜} \cup \overline{B˜} \quad \overline{A~\cap B~}=\overline{A~} \cup \overline{B~}$$
$$\overline{A˜ \cup B˜}=\overline{A˜}\cap\overline{B˜}$$

We already know that fuzzy logic is not logic that is fuzzy but logic that is used to describe fuzziness. This fuzziness is best characterized by its membership function. In other words, we can say that membership function represents the degree of truth in fuzzy logic.

Membership Function
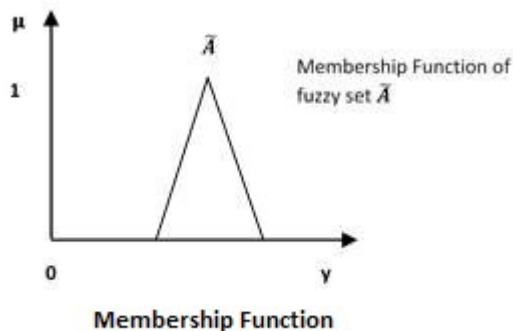
Following are a few important points relating to the membership function −

- Membership functions were first introduced in 1965 by Lofti A. Zadeh in his first research paper "fuzzy sets".
- Membership functions characterize fuzziness (i.e., all the information in fuzzy set), whether the elements in fuzzy sets are discrete or continuous.
- Membership functions can be defined as a technique to solve practical problems by experience rather than knowledge.
- Membership functions are represented by graphical forms.
- Rules for defining fuzziness are fuzzy too.

Mathematical Notation

We have already studied that a fuzzy set $\tilde{A}$ in the universe of information U can be defined as a set of ordered pairs and it can be represented mathematically as −

$$A\tilde{} = \{(y, \mu A\tilde{}(y)) | y \in U\} A\sim = \{(y, \mu A\sim(y)) | y \in U\}$$

Here $\mu A\tilde{}(\cdot) \mu A\sim(\cdot)$ = membership function of $A\tilde{}A\sim$; this assumes values in the range from 0 to 1, i.e., $\mu A\tilde{}(\cdot) \in [0,1] \mu A\sim(\cdot) \in [0,1]$. The membership function $\mu A\tilde{}(\cdot) \mu A\sim(\cdot)$ maps UU to the membership space MM.

The dot $(\cdot)(\cdot)$ in the membership function described above, represents the element in a fuzzy set; whether it is discrete or continuous.

Features of Membership Functions

We will now discuss the different features of Membership Functions.

Core

For any fuzzy set $A\tilde{}A\sim$, the core of a membership function is that region of universe that is characterize by full membership in the set. Hence, core consists of all those elements yy of the universe of information such that,

$$\mu A\tilde{}(y) = 1 \mu A\sim(y) = 1$$

Support

For any fuzzy set $A\tilde{}A\sim$, the support of a membership function is the region of universe that is characterize by a nonzero membership in the set. Hence core consists of all those elements yy of the universe of information such that,

$$\mu A\tilde{}(y) > 0 \mu A\sim(y) > 0$$

Boundary

For any fuzzy set $A\tilde{}A\sim$, the boundary of a membership function is the region of universe that is characterized by a nonzero but incomplete membership in the set. Hence, core consists of all those elements yy of the universe of information such that,

$$1 > \mu A\tilde{}(y) > 0 1 > \mu A\sim(y) > 0$$
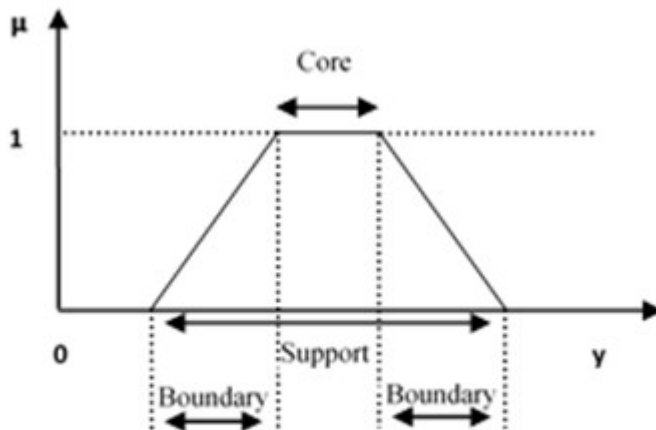
Features of Membership Function

Fuzzification

It may be defined as the process of transforming a crisp set to a fuzzy set or a fuzzy set to fuzzier set. Basically, this operation translates accurate crisp input values into linguistic variables.

Following are the two important methods of fuzzification −

Support Fuzzification(s-fuzzification) Method

In this method, the fuzzified set can be expressed with the help of the following relation −

$$A\tilde{} = \mu_1 Q(x_1) + \mu_2 Q(x_2) + ... + \mu_n Q(x_n) A\sim = \mu_1 Q(x_1) + \mu_2 Q(x_2) + ... + \mu_n Q(x_n)$$

Here the fuzzy set $Q(x_i)Q(x_i)$ is called as kernel of fuzzification. This method is implemented by keeping $\mu_i \mu_i$ constant and $x_i x_i$ being transformed to a fuzzy set $Q(x_i)Q(x_i)$.

Grade Fuzzification (g-fuzzification) Method

It is quite similar to the above method but the main difference is that it kept $x_i x_i$ constant and $\mu_i \mu_i$ is expressed as a fuzzy set.

Defuzzification

It may be defined as the process of reducing a fuzzy set into a crisp set or to convert a fuzzy member into a crisp member.

We have already studied that the fuzzification process involves conversion from crisp quantities to fuzzy quantities. In a number of engineering applications, it is necessary to defuzzify the result or rather "fuzzy result" so that it must be converted to crisp result. Mathematically, the process of Defuzzification is also called "rounding it off".

The different methods of Defuzzification are described below −

Max-Membership Method

This method is limited to peak output functions and also known as height method. Mathematically it can be represented as follows −

$$\mu\tilde{A}(x*) > \mu\tilde{A}(x) \text{ for all } x \in X \quad \mu A\sim(x*) > \mu A\sim(x) \text{ for all } x \in X$$

Here, $x* \, x*$ is the defuzzified output.

Centroid Method

This method is also known as the center of area or the center of gravity method. Mathematically, the defuzzified output $x* \, x*$ will be represented as −

$$x* = \frac{\int \mu\tilde{A}(x).x\,dx}{\int \mu\tilde{A}(x).dx} \quad x* = \frac{\int \mu A\sim(x).x\,dx}{\int \mu A\sim(x).dx}$$

Weighted Average Method

In this method, each membership function is weighted by its maximum membership value. Mathematically, the defuzzified output $x* \, x*$ will be represented as −

$$x* = \frac{\sum \mu\tilde{A}(\overline{x_i}).\overline{x_i}}{\sum \mu\tilde{A}(\overline{x_i})} \quad x* = \frac{\sum \mu A\sim(\overline{x_i}).\overline{x_i}}{\sum \mu A\sim(\overline{x_i})}$$

Mean-Max Membership

This method is also known as the middle of the maxima. Mathematically, the defuzzified output $x* \, x*$ will be represented as −

$$x* = \frac{\sum_{i=1}^{n} \overline{x_i}}{n}$$

Logic, which was originally just the study of what distinguishes sound argument from unsound argument, has now developed into a powerful and rigorous system whereby true statements can be discovered, given other statements that are already known to be true.

Predicate Logic

This logic deals with predicates, which are propositions containing variables.

A predicate is an expression of one or more variables defined on some specific domain. A predicate with variables can be made a proposition by either assigning a value to the variable or by quantifying the variable.

Following are a few examples of predicates −

- Let E(x, y) denote "x = y"
- Let X(a, b, c) denote "a + b + c = 0"
- Let M(x, y) denote "x is married to y"

Propositional Logic

A proposition is a collection of declarative statements that have either a truth value "true" or a truth value "false". A propositional consists of propositional variables and connectives. The propositional variables are dented by capital letters (A, B, etc). The connectives connect the propositional variables.

A few examples of Propositions are given below −

- "Man is Mortal", it returns truth value "TRUE"
- "12 + 9 = 3 − 2", it returns truth value "FALSE"

The following is not a Proposition −

- **"A is less than 2"** − It is because unless we give a specific value of A, we cannot say whether the statement is true or false.

Connectives

In propositional logic, we use the following five connectives −

- OR (∨ ∨)
- AND (∧ ∧)
- Negation/ NOT (¬¬)
- Implication / if-then (→ →)
- If and only if (⇔⇔)

OR (∨ ∨)

The OR operation of two propositions A and B (written as A ∨ BA ∨ B) is true if at least any of the propositional variable A or B is true.

The truth table is as follows −

A	B	A ∨ B
True	True	True

True	False	True
False	True	True
False	False	False

AND (∧ ∧)

The AND operation of two propositions A and B (written as A ∧ BA ∧ B) is true if both the propositional variable A and B is true.

The truth table is as follows −

A	B	A ∧ B
True	True	True
True	False	False
False	True	False
False	False	False

Negation (¬¬)

The negation of a proposition A (written as ¬A¬A) is false when A is true and is true when A is false.

The truth table is as follows −

A	¬A
True	False
False	True

Implication / if-then (→ →)

An implication A→BA→B is the proposition "if A, then B". It is false if A is true and B is false. The rest cases are true.

The truth table is as follows −

A	B	A→B
True	True	True
True	False	False
False	True	True
False	False	True

If and only if (⇔⇔)

A⇔BA⇔B is a bi-conditional logical connective which is true when p and q are same, i.e., both are false or both are true.

The truth table is as follows −

A	B	A⇔B
True	True	True
True	False	False
False	True	False

False	False	True

Well Formed Formula

Well Formed Formula (wff) is a predicate holding one of the following −

- All propositional constants and propositional variables are wffs.
- If x is a variable and Y is a wff, ∀xY and ∃xY are also wff.
- Truth value and false values are wffs.
- Each atomic formula is a wff.
- All connectives connecting wffs are wffs.

Quantifiers

The variable of predicates is quantified by quantifiers. There are two types of quantifier in predicate logic −

- Universal Quantifier
- Existential Quantifier

Universal Quantifier

Universal quantifier states that the statements within its scope are true for every value of the specific variable. It is denoted by the symbol ∀.

∀xP(x) is read as for every value of x, P(x) is true.

Example − "Man is mortal" can be transformed into the propositional form ∀xP(x). Here, P(x) is the predicate which denotes that x is mortal and the universe of discourse is all men.

Existential Quantifier

Existential quantifier states that the statements within its scope are true for some values of the specific variable. It is denoted by the symbol ∃.

∃xP(x) for some values of x is read as, P(x) is true.

Example − "Some people are dishonest" can be transformed into the propositional form ∃x P(x) where P(x) is the predicate which denotes x is dishonest and the universe of discourse is some people.

Nested Quantifiers

If we use a quantifier that appears within the scope of another quantifier, it is called a nested quantifier.

Example

- ∀ a∃bP(x,y) where P(a,b) denotes a+b = 0
- ∀ a∀b∀cP(a,b,c) where P(a,b) denotes a+(b+c) = (a+b)+c

Note − ∀a∃bP(x,y) ≠ ∃a∀bP(x,y)

Following are the different modes of approximate reasoning −

Categorical Reasoning

In this mode of approximate reasoning, the antecedents, containing no fuzzy quantifiers and fuzzy probabilities, are assumed to be in canonical form.

Qualitative Reasoning

In this mode of approximate reasoning, the antecedents and consequents have fuzzy linguistic variables; the input-output relationship of a system is expressed as a collection of fuzzy IF-THEN rules. This reasoning is mainly used in control system analysis.

Syllogistic Reasoning

In this mode of approximation reasoning, antecedents with fuzzy quantifiers are related to inference rules. This is expressed as −

$$x = S_1 A's \text{ are } B's$$
$$y = S_2 C's \text{ are } D's$$

$$z = S_3 E's \text{ are } F's$$

Here A,B,C,D,E,F are fuzzy predicates.

- S_1 and S_2 are given fuzzy quantifiers.
- S_3 is the fuzzy quantifier which has to be decided.

Dispositional Reasoning

In this mode of approximation reasoning, the antecedents are dispositions that may contain the fuzzy quantifier "usually". The quantifier **Usually** links together the dispositional and syllogistic reasoning; hence it pays an important role.

For example, the projection rule of inference in dispositional reasoning can be given as follows −

usually((L,M) is R) ⇒ usually (L is [R ↓ L])

Here **[R ↓ L]** is the projection of fuzzy relation **R** on **L**

Fuzzy Logic Rule Base

It is a known fact that a human being is always comfortable making conversations in natural language. The representation of human knowledge can be done with the help of following natural language expression −

IF antecedent **THEN** consequent

The expression as stated above is referred to as the Fuzzy IF-THEN rule base.

Canonical Form

Following is the canonical form of Fuzzy Logic Rule Base −

Rule 1 − If condition C1, then restriction R1

Rule 2 − If condition C1, then restriction R2

.

..

Rule n − If condition C1, then restriction Rn

Interpretations of Fuzzy IF-THEN Rules

Fuzzy IF-THEN Rules can be interpreted in the following four forms −

Assignment Statements

These kinds of statements use "=" (equal to sign) for the purpose of assignment. They are of the following form −

a = hello

climate = summer

Conditional Statements

These kinds of statements use the "IF-THEN" rule base form for the purpose of condition. They are of the following form −

IF temperature is high THEN Climate is hot

IF food is fresh THEN eat.

Unconditional Statements

They are of the following form −

GOTO 10

turn the Fan off

Linguistic Variable

We have studied that fuzzy logic uses linguistic variables which are the words or sentences in a natural language. For example, if we say temperature, it is a linguistic variable; the values of which are very hot or cold, slightly hot or cold, very warm, slightly warm, etc. The words very, slightly are the linguistic hedges.

Characterization of Linguistic Variable

Following four terms characterize the linguistic variable −

- Name of the variable, generally represented by x.
- Term set of the variable, generally represented by t(x).
- Syntactic rules for generating the values of the variable x.
- Semantic rules for linking every value of x and its significance.

Propositions in Fuzzy Logic

As we know that propositions are sentences expressed in any language which are generally expressed in the following canonical form −

s as P

Here, *s* is the Subject and *P* is Predicate.

For example, "*Delhi is the capital of India*", this is a proposition where "*Delhi*" is the subject and "*is the capital of India*" is the predicate which shows the property of subject.

We know that logic is the basis of reasoning and fuzzy logic extends the capability of reasoning by using fuzzy predicates, fuzzy-predicate modifiers, fuzzy quantifiers and fuzzy qualifiers in fuzzy propositions which creates the difference from classical logic.

Propositions in fuzzy logic include the following −

Fuzzy Predicate

Almost every predicate in natural language is fuzzy in nature hence, fuzzy logic has the predicates like tall, short, warm, hot, fast, etc.

Fuzzy-predicate Modifiers

We discussed linguistic hedges above; we also have many fuzzy-predicate modifiers which act as hedges. They are very essential for producing the values of a linguistic variable. For example, the words very, slightly are modifiers and the propositions can be like "*water is slightly hot*."

Fuzzy Quantifiers

It can be defined as a fuzzy number which gives a vague classification of the cardinality of one or more fuzzy or non-fuzzy sets. It can be used to influence probability within fuzzy logic. For example, the words many, most, frequently are used as fuzzy quantifiers and the propositions can be like "*most people are allergic to it*."

Fuzzy Qualifiers

Let us now understand Fuzzy Qualifiers. A Fuzzy Qualifier is also a proposition of Fuzzy Logic. Fuzzy qualification has the following forms −

Fuzzy Qualification Based on Truth
It claims the degree of truth of a fuzzy proposition.
Expression − It is expressed as *x is t*. Here, *t* is a fuzzy truth value.
Example − (Car is black) is NOT VERY True.
Fuzzy Qualification Based on Probability
It claims the probability, either numerical or an interval, of fuzzy proposition.
Expression − It is expressed as *x is λ*. Here, *λ* is a fuzzy probability.
Example − (Car is black) is Likely.
Fuzzy Qualification Based on Possibility
It claims the possibility of fuzzy proposition.
Expression − It is expressed as *x is π*. Here, *π* is a fuzzy possibility.
Example − (Car is black) is Almost Impossible.

Fuzzy rule base system: fuzzy propositions, formation, decomposition & aggregation of fuzzy rules, fuzzy reasoning, fuzzy inference systems,

Fuzzy Inference System is the key unit of a fuzzy logic system having decision making as its primary work. It uses the "IF…THEN" rules along with connectors "OR" or "AND" for drawing essential decision rules.

Characteristics of Fuzzy Inference System
Following are some characteristics of FIS −
- The output from FIS is always a fuzzy set irrespective of its input which can be fuzzy or crisp.
- It is necessary to have fuzzy output when it is used as a controller.
- A defuzzification unit would be there with FIS to convert fuzzy variables into crisp variables.

Functional Blocks of FIS
The following five functional blocks will help you understand the construction of FIS −
- **Rule Base** − It contains fuzzy IF-THEN rules.
- **Database** − It defines the membership functions of fuzzy sets used in fuzzy rules.
- **Decision-making Unit** − It performs operation on rules.
- **Fuzzification Interface Unit** − It converts the crisp quantities into fuzzy quantities.
- **Defuzzification Interface Unit** − It converts the fuzzy quantities into crisp quantities. Following is a block diagram of fuzzy interference system.
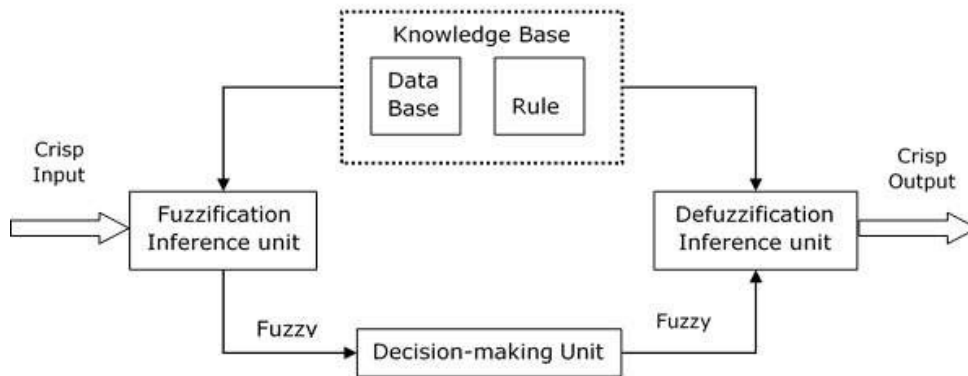
Working of FIS

The working of the FIS consists of the following steps −
- A fuzzification unit supports the application of numerous fuzzification methods, and converts the crisp input into fuzzy input.
- A knowledge base - collection of rule base and database is formed upon the conversion of crisp input into fuzzy input.
- The defuzzification unit fuzzy input is finally converted into crisp output.

Methods of FIS

Let us now discuss the different methods of FIS. Following are the two important methods of FIS, having different consequent of fuzzy rules −

- Mamdani Fuzzy Inference System
- Takagi-Sugeno Fuzzy Model (TS Method)

Mamdani Fuzzy Inference System

This system was proposed in 1975 by Ebhasim Mamdani. Basically, it was anticipated to control a steam engine and boiler combination by synthesizing a set of fuzzy rules obtained from people working on the system.

Steps for Computing the Output

Following steps need to be followed to compute the output from this FIS −

- **Step 1** − Set of fuzzy rules need to be determined in this step.
- **Step 2** − In this step, by using input membership function, the input would be made fuzzy.
- **Step 3** − Now establish the rule strength by combining the fuzzified inputs according to fuzzy rules.
- **Step 4** − In this step, determine the consequent of rule by combining the rule strength and the output membership function.
- **Step 5** − For getting output distribution combine all the consequents.
- **Step 6** − Finally, a defuzzified output distribution is obtained.

Following is a block diagram of Mamdani Fuzzy Interface System.
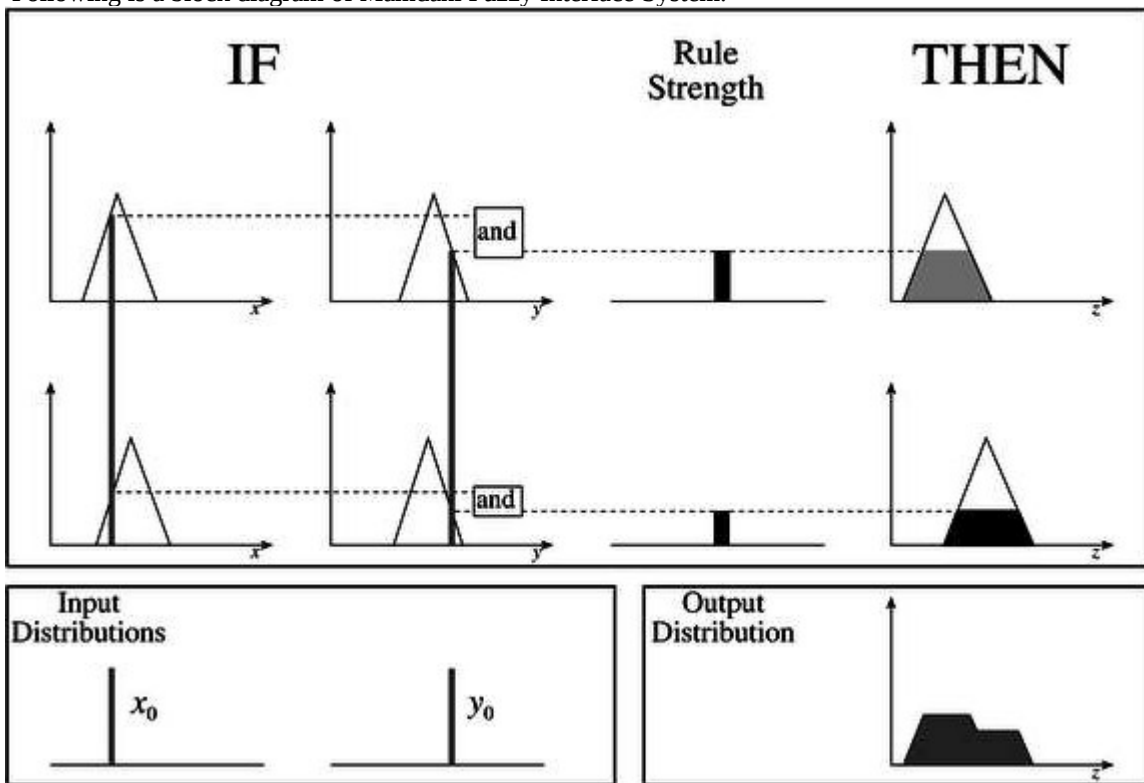
Takagi-Sugeno Fuzzy Model (TS Method)

This model was proposed by Takagi, Sugeno and Kang in 1985. Format of this rule is given as −

$$IF\ x\ is\ A\ and\ y\ is\ B\ THEN\ Z = f(x,y)$$

Here, *AB* are fuzzy sets in antecedents and $z = f(x,y)$ is a crisp function in the consequent.

Fuzzy Inference Process

The fuzzy inference process under Takagi-Sugeno Fuzzy Model (TS Method) works in the following way −

- **Step 1: Fuzzifying the inputs** − Here, the inputs of the system are made fuzzy.
- **Step 2: Applying the fuzzy operator** − In this step, the fuzzy operators must be applied to get the output.

Rule Format of the Sugeno Form

The rule format of Sugeno form is given by −

$$if\ 7 = x\ and\ 9 = y\ then\ output\ is\ z = ax+by+c$$

Comparison between the two methods

Let us now understand the comparison between the Mamdani System and the Sugeno Model.

- **Output Membership Function** − The main difference between them is on the basis of output membership function. The Sugeno output membership functions are either linear or constant.

- **Aggregation and Defuzzification Procedure** − The difference between them also lies in the consequence of fuzzy rules and due to the same their aggregation and defuzzification procedure also differs.
- **Mathematical Rules** − More mathematical rules exist for the Sugeno rule than the Mamdani rule.
- **Adjustable Parameters** − The Sugeno controller has more adjustable parameters than the Mamdani controller.

e have studied in our previous chapters that Fuzzy Logic is an approach to computing based on "degrees of truth" rather than the usual "true or false" logic. It deals with reasoning that is approximate rather than precise to solve problems in a way that more resembles human logic, hence database querying process by the two valued realization of Boolean algebra is not adequate.

Fuzzy Scenario of Relations on Databases

The Fuzzy Scenario of Relations on Databases can be understood with the help of the following example −

Example

Suppose we have a database having the records of persons who visited India. In simple database, we will have the entries made in the following way −

Name	Age	Citizen	Visited Country	Days Spent	Year of Visit
John Smith	35	U.S.	India	41	1999
John Smith	35	U.S.	Italy	72	1999
John Smith	35	U.S.	Japan	31	1999

Now, if anyone queries about the person who visited India and Japan in the year 99 and is the citizen of US, then the output will show two entries having the name of John Smith. This is simple query generating simple output.

But what if we want to know whether the person in the above query is young or not. According to the above result, the age of the person is 35 years. But can we assume the person to be young or not? Similarly, same thing can be applied on the other fields like days spent, year of visit, etc.

The solution of the above issues can be found with the help of Fuzzy Value sets as follows −
- FV(Age){ very young, young, somewhat old, old }
- FV(Days Spent){ barely few days, few days, quite a few days, many days }
- FV(Year of Visit){distant past, recent past, recent }
- Now if any query will have the fuzzy value then the result will also be fuzzy in nature.

Fuzzy Query System

A fuzzy query system is an interface to users to get information from the database using (quasi) natural language sentences. Many fuzzy query implementations have been proposed, resulting in slightly different languages. Although there are some variations according to the particularities of different implementations, the answer to a fuzzy query sentence is generally a list of records, ranked by the degree of matching.

In modeling natural language statements, quantified statements play an important role. It means that NL heavily depends on quantifying construction which often includes fuzzy concepts like "almost all", "many", etc. Following are a few examples of quantifying propositions −
- Every student passed the exam.
- Every sport car is expensive.
- Many students passed the exam.
- Many sports cars are expensive.

In the above examples, the quantifiers "Every" and "Many" are applied to the crisp restrictions "students" as well as crisp scope "(person who)passed the exam" and "cars" as well as crisp scope "sports".

Fuzzy Events, Fuzzy Means and Fuzzy Variances

With the help of an example, we can understand the above concepts. Let us assume that we are a shareholder of a company named ABC. And at present the company is selling each of its share for ₹40. There are three different companies whose business is similar to ABC but these are offering their shares at different rates - ₹100 a share, ₹85 a share and ₹60 a share respectively.

Now the probability distribution of this price takeover is as follows −

Price	₹100	₹85	₹60
Probability	0.3	0.5	0.2

Now, from the standard probability theory, the above distribution gives a mean of expected price as below −

$$100 \times 0.3 + 85 \times 0.5 + 60 \times 0.2 = 84.5\ 100 \times 0.3 + 85 \times 0.5 + 60 \times 0.2 = 84.5$$

And, from the standard probability theory, the above distribution gives a variance of expected price as below −

$$(100 - 84.5)2 \times 0.3 + (85 - 84.5)2 \times 0.5 + (60 - 84.5)2 \times 0.2 = 124.825\ (100 - 84.5)2 \times 0.3 + (85 - 84.5)2 \times 0.5 + (60 - 84.5)2 \times 0.2 = 124.825$$

Suppose the degree of membership of 100 in this set is 0.7, that of 85 is 1, and the degree of membership is 0.5 for the value 60. These can be reflected in the following fuzzy set −

$$\{0.7 100, 1 85, 0.5 60,\}\ \{0.7 100, 1 85, 0.5 60,\}$$

The fuzzy set obtained in this manner is called a fuzzy event.

We want the probability of the fuzzy event for which our calculation gives −

$$0.7 \times 0.3 + 1 \times 0.5 + 0.5 \times 0.2 = 0.21 + 0.5 + 0.1 = 0.81\ 0.7 \times 0.3 + 1 \times 0.5 + 0.5 \times 0.2 = 0.21 + 0.5 + 0.1 = 0.81$$

Now, we need to calculate the fuzzy mean and the fuzzy variance, the calculation is as follows −

Fuzzy_mean $= (1 0.81) \times (100 \times 0.7 \times 0.3 + 85 \times 1 \times 0.5 + 60 \times 0.5 \times 0.2) = (1 0.81) \times (100 \times 0.7 \times 0.3 + 85 \times 1 \times 0.5 + 60 \times 0.5 \times 0.2) = 85.8 = 85.8$

Fuzzy_Variance $= 7496.91 - 7361.91 = 135.27$

t is an activity which includes the steps to be taken for choosing a suitable alternative from those that are needed for realizing a certain goal.

Steps for Decision Making

Let us now discuss the steps involved in the decision making process −

- **Determining the Set of Alternatives** − In this step, the alternatives from which the decision has to be taken must be determined.
- **Evaluating Alternative** − Here, the alternatives must be evaluated so that the decision can be taken about one of the alternatives.
- **Comparison between Alternatives** − In this step, a comparison between the evaluated alternatives is done.

Types of Decision

Making We will now understand the different types of decision making.

Individual Decision Making

In this type of decision making, only a single person is responsible for taking decisions. The decision making model in this kind can be characterized as −

- Set of possible actions
- Set of goals $G_i (i \in X_n); G_i (i \in X_n);$
- Set of Constraints $C_j (j \in X_m) C_j (j \in X_m)$

The goals and constraints stated above are expressed in terms of fuzzy sets.

Now consider a set A. Then, the goal and constraints for this set are given by −

$G_i(a) G_i(a) = $ composition$[G_i(a)][G_i(a)] = G_{1i}(G_i(a)) G_{1i}(G_i(a))$ with $G_{1i} G_{i1}$

$C_j(a) C_j(a) = $ composition$[C_j(a)][C_j(a)] = C_{1j}(C_j(a)) C_{j1}(C_j(a))$ with $C_{1j} C_{j1}$ for $a \in A\ a \in A$

The fuzzy decision in the above case is given by −

$$FD = \min[i \in X_n n f G_i(a), j \in X_m f C_j(a)]\ FD = \min[i \in X_n \inf G_i(a), j \in X_m \inf C_j(a)]$$

Multi-person Decision Making

Decision making in this case includes several persons so that the expert knowledge from various persons is utilized to make decisions.

Calculation for this can be given as follows −

Number of persons preferring $x_i x_i$ **to** $x_j x_j = N(x_i, x_j) N(x_i, x_j)$

Total number of decision makers $= n\ n$

Then, $SC(x_i, x_j) = N(x_i, x_j) n\ SC(x_i, x_j) = N(x_i, x_j) n$

Multi-objective Decision Making

Multi-objective decision making occurs when there are several objectives to be realized. There are following two issues in this type of decision making −

- To acquire proper information related to the satisfaction of the objectives by various alternatives.
- To weigh the relative importance of each objective.

Mathematically we can define a universe of n alternatives as −

$A = [a_1, a_2, ..., a_i, ..., a_n] A = [a_1, a_2, ..., a_i, ..., a_n]$

And the set of "m" objectives as $O = [o_1, o_2, ..., o_i, ..., o_n] O = [o_1, o_2, ..., o_i, ..., o_n]$

Multi-attribute Decision Making

Multi-attribute decision making takes place when the evaluation of alternatives can be carried out based on several attributes of the object. The attributes can be numerical data, linguistic data and qualitative data.

Mathematically, the multi-attribute evaluation is carried out on the basis of linear equation as follows −

$$Y = A_1 X_1 + A_2 X_2 + ... + A_i X_i + ... + A_r X_r$$

Fuzzy logic is applied with great success in various control application. Almost all the consumer products have fuzzy control. Some of the examples include controlling your room temperature with the help of air-

conditioner, anti-braking system used in vehicles, control on traffic lights, washing machines, large economic systems, etc.

Why Use Fuzzy Logic in Control Systems

A control system is an arrangement of physical components designed to alter another physical system so that this system exhibits certain desired characteristics. Following are some reasons of using Fuzzy Logic in Control Systems −

- While applying traditional control, one needs to know about the model and the objective function formulated in precise terms. This makes it very difficult to apply in many cases.
- By applying fuzzy logic for control we can utilize the human expertise and experience for designing a controller.
- The fuzzy control rules, basically the IF-THEN rules, can be best utilized in designing a controller.

Assumptions in Fuzzy Logic Control (FLC) Design

While designing fuzzy control system, the following six basic assumptions should be made −

- **The plant is observable and controllable** − It must be assumed that the input, output as well as state variables are available for observation and controlling purpose.
- **Existence of a knowledge body** − It must be assumed that there exist a knowledge body having linguistic rules and a set of input-output data set from which rules can be extracted.
- **Existence of solution** − It must be assumed that there exists a solution.
- **'Good enough' solution is enough** − The control engineering must look for 'good enough' solution rather than an optimum one.
- **Range of precision** − Fuzzy logic controller must be designed within an acceptable range of precision.
- **Issues regarding stability and optimality** − The issues of stability and optimality must be open in designing Fuzzy logic controller rather than addressed explicitly.

Architecture of Fuzzy Logic Control

The following diagram shows the architecture of Fuzzy Logic Control (FLC).
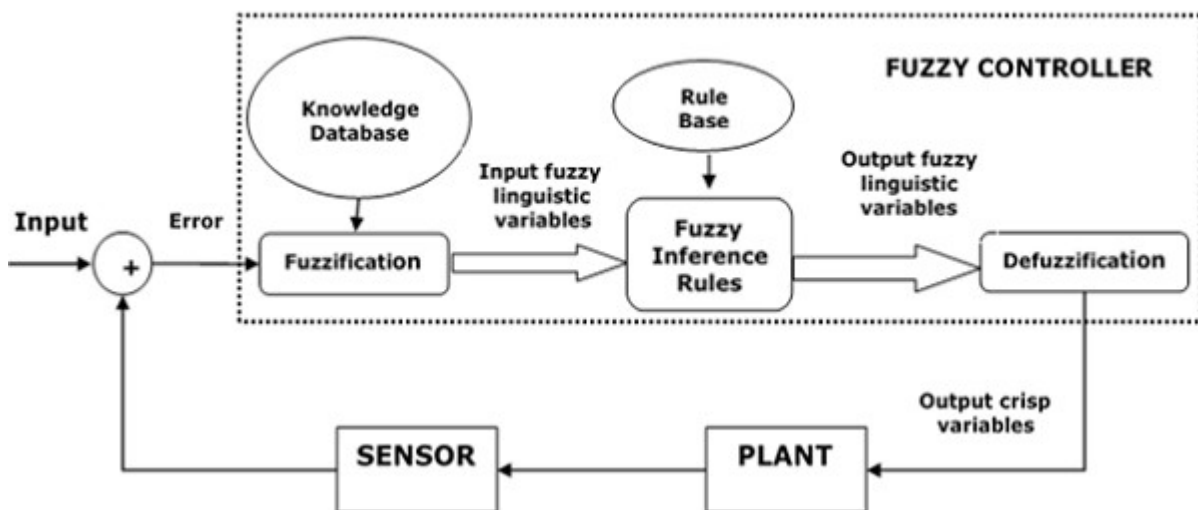
Major Components of FLC

Followings are the major components of the FLC as shown in the above figure −

- **Fuzzifier** − The role of fuzzifier is to convert the crisp input values into fuzzy values.
- **Fuzzy Knowledge Base** − It stores the knowledge about all the input-output fuzzy relationships. It also has the membership function which defines the input variables to the fuzzy rule base and the output variables to the plant under control.
- **Fuzzy Rule Base** − It stores the knowledge about the operation of the process of domain.
- **Inference Engine** − It acts as a kernel of any FLC. Basically it simulates human decisions by performing approximate reasoning.
- **Defuzzifier** − The role of defuzzifier is to convert the fuzzy values into crisp values getting from fuzzy inference engine.

Steps in Designing FLC

Following are the steps involved in designing FLC −

- **Identification of variables** − Here, the input, output and state variables must be identified of the plant which is under consideration.

- **Fuzzy subset configuration** − The universe of information is divided into number of fuzzy subsets and each subset is assigned a linguistic label. Always make sure that these fuzzy subsets include all the elements of universe.
- **Obtaining membership function** − Now obtain the membership function for each fuzzy subset that we get in the above step.
- **Fuzzy rule base configuration** − Now formulate the fuzzy rule base by assigning relationship between fuzzy input and output.
- **Fuzzification** − The fuzzification process is initiated in this step.
- **Combining fuzzy outputs** − By applying fuzzy approximate reasoning, locate the fuzzy output and merge them.
- **Defuzzification** − Finally, initiate defuzzification process to form a crisp output.

Advantages of Fuzzy Logic Control

Let us now discuss the advantages of Fuzzy Logic Control.

- **Cheaper** − Developing a FLC is comparatively cheaper than developing model based or other controller in terms of performance.
- **Robust** − FLCs are more robust than PID controllers because of their capability to cover a huge range of operating conditions.
- **Customizable** − FLCs are customizable.
- **Emulate human deductive thinking** − Basically FLC is designed to emulate human deductive thinking, the process people use to infer conclusion from what they know.
- **Reliability** − FLC is more reliable than conventional control system.
- **Efficiency** − Fuzzy logic provides more efficiency when applied in control system.

Disadvantages of Fuzzy Logic Control

We will now discuss what are the disadvantages of Fuzzy Logic Control.

- **Requires lots of data** − FLC needs lots of data to be applied.
- **Useful in case of moderate historical data** − FLC is not useful for programs much smaller or larger than historical data.
- **Needs high human expertise** − This is one drawback as the accuracy of the system depends on the knowledge and expertise of human beings.
- **Needs regular updating of rules** − The rules must be updated with time.
-

Applications of fuzzy logic

In this chapter, we will discuss the fields where the concepts of Fuzzy Logic are extensively applied.

Aerospace

In aerospace, fuzzy logic is used in the following areas −

- Altitude control of spacecraft
- Satellite altitude control
- Flow and mixture regulation in aircraft deicing vehicles

Automotive

In automotive, fuzzy logic is used in the following areas −

- Trainable fuzzy systems for idle speed control
- Shift scheduling method for automatic transmission
- Intelligent highway systems
- Traffic control
- Improving efficiency of automatic transmissions

Business

In business, fuzzy logic is used in the following areas −

- Decision-making support systems
- Personnel evaluation in a large company

Defense

In defense, fuzzy logic is used in the following areas −

- Underwater target recognition
- Automatic target recognition of thermal infrared images
- Naval decision support aids
- Control of a hypervelocity interceptor
- Fuzzy set modeling of NATO decision making

Electronics

In electronics, fuzzy logic is used in the following areas −
- Control of automatic exposure in video cameras
- Humidity in a clean room
- Air conditioning systems
- Washing machine timing
- Microwave ovens
- Vacuum cleaners

Finance

In the finance field, fuzzy logic is used in the following areas −
- Banknote transfer control
- Fund management
- Stock market predictions

Industrial Sector

In industrial, fuzzy logic is used in following areas −
- Cement kiln controls heat exchanger control
- Activated sludge wastewater treatment process control
- Water purification plant control
- Quantitative pattern analysis for industrial quality assurance
- Control of constraint satisfaction problems in structural design
- Control of water purification plants

Manufacturing

In the manufacturing industry, fuzzy logic is used in following areas −
- Optimization of cheese production
- Optimization of milk production

Marine

In the marine field, fuzzy logic is used in the following areas −
- Autopilot for ships
- Optimal route selection
- Control of autonomous underwater vehicles
- Ship steering

Medical

In the medical field, fuzzy logic is used in the following areas −
- Medical diagnostic support system
- Control of arterial pressure during anesthesia
- Multivariable control of anesthesia
- Modeling of neuropathological findings in Alzheimer's patients
- Radiology diagnoses
- Fuzzy inference diagnosis of diabetes and prostate cancer

Securities

In securities, fuzzy logic is used in following areas −
- Decision systems for securities trading
- Various security appliances

Transportation

In transportation, fuzzy logic is used in the following areas −
- Automatic underground train operation
- Train schedule control
- Railway acceleration
- Braking and stopping

Pattern Recognition and Classification

In Pattern Recognition and Classification, fuzzy logic is used in the following areas −
- Fuzzy logic based speech recognition
- Fuzzy logic based
- Handwriting recognition
- Fuzzy logic based facial characteristic analysis
- Command analysis
- Fuzzy image search

Psychology

In Psychology, fuzzy logic is used in following areas −
- Fuzzy logic based analysis of human behavior
- Criminal investigation and prevention based on fuzzy logic reasoning

Text Books:

1. Fuzzy logic with engineering applications, Timothy J. Ross, John Wiley and Sons.
2. Neural Networks: A Comprehensive Foundation (2nd Edition), Simon Haykin, Prentice Hall.
Reference Books:
1. K.H.Lee. First Course on Fuzzy Theory and Applications, Springer-Verlag.
2. J. Yen and R. Langari.. Fuzzy Logic, Intelligence, Control and Information, Pearson Education.

CO-PO Mapping

CO	PO1	PO2	POP3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CS702A.1	2											
CS702A.2			2									
CS702A.3	1		3									
CS702A.4	2											
CS702A.5		1				3						