

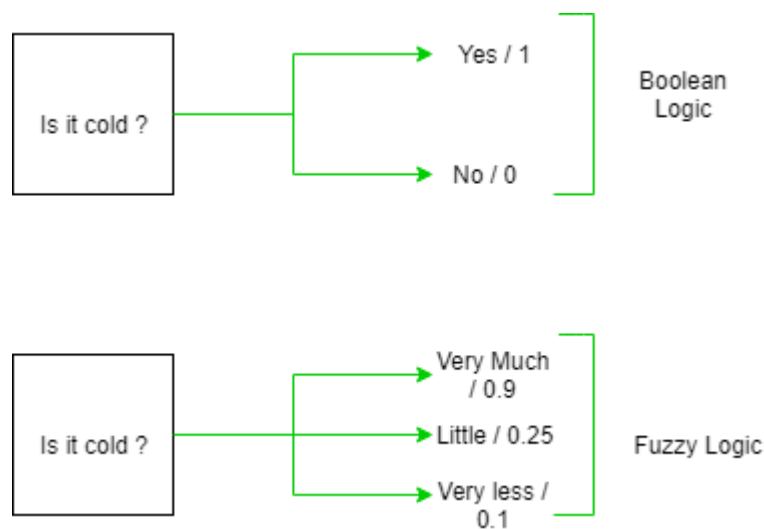
UNIT IV

Fuzzy Logic Systems (FLS) produce acceptable but definite output in response to incomplete, ambiguous, distorted, or inaccurate (fuzzy) input.

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.



Implementation

- It can be implemented in systems with various sizes and capabilities ranging from small micro-controllers to large, networked, workstation-based control systems.
- It can be implemented in hardware, software, or a combination of both.

Why Fuzzy Logic?

Fuzzy logic is useful for commercial and practical purposes.

- It can control machines and consumer products.
- It may not give accurate reasoning, but acceptable reasoning.
- Fuzzy logic helps to deal with the uncertainty in engineering.

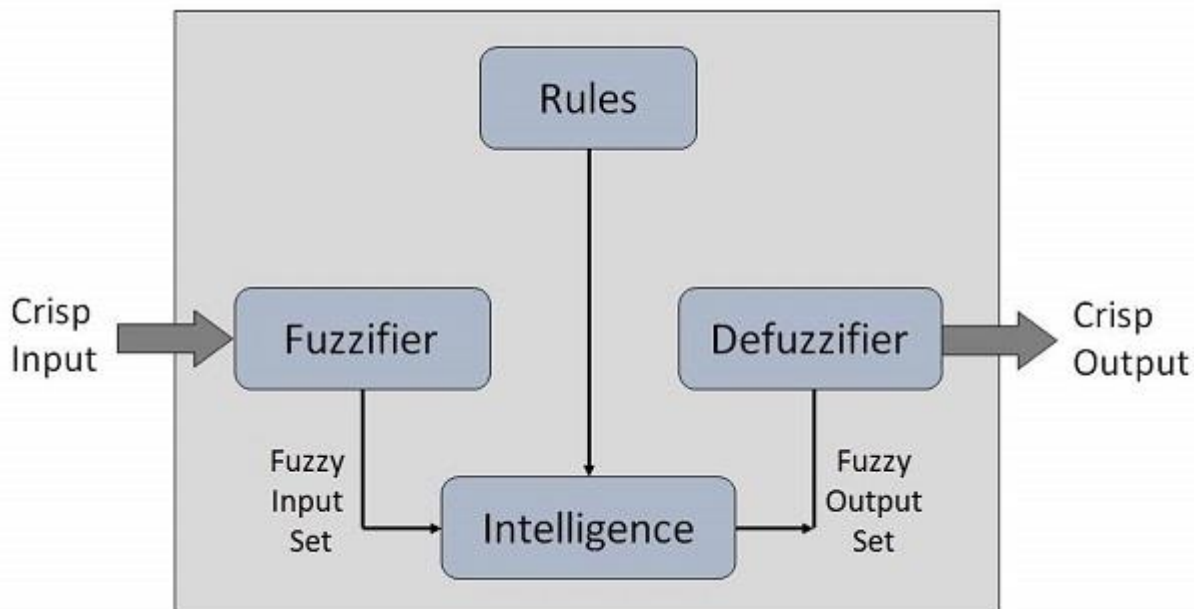
Fuzzy Logic Systems Architecture

It has four main parts as shown –

- **Fuzzification Module** – It transforms the system inputs, which are crisp numbers, into fuzzy sets. It splits the input signal into five steps such as –

LP x is Large Positive
MP x is Medium Positive
S x is Small
MN x is Medium Negative
LN x is Large Negative

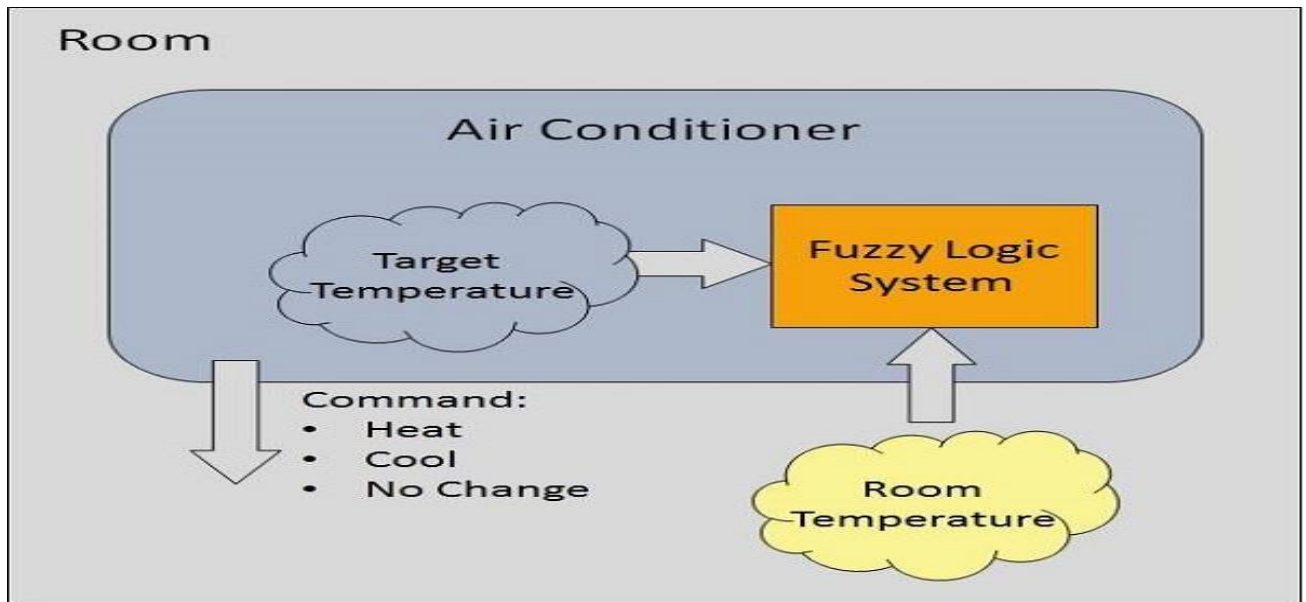
- **Knowledge Base** – It stores IF-THEN rules provided by experts.
- **Inference Engine** – It simulates the human reasoning process by making fuzzy inference on the inputs and IF-THEN rules.
- **Defuzzification Module** – It transforms the fuzzy set obtained by the inference engine into a crisp value.



The **membership functions work on** fuzzy sets of variables.

Example of a Fuzzy Logic System

Let us consider an air conditioning system with 5-level fuzzy logic system. This system adjusts the temperature of air conditioner by comparing the room temperature and the target temperature value.



Algorithm

- Define linguistic Variables and terms (start)
- Construct membership functions for them. (start)
- Construct knowledge base of rules (start)
- Convert crisp data into fuzzy data sets using membership functions. (fuzzification)
- Evaluate rules in the rule base. (Inference Engine)
- Combine results from each rule. (Inference Engine)
- Convert output data into non-fuzzy values. (defuzzification)

Development

Step 1 – Define linguistic variables and terms

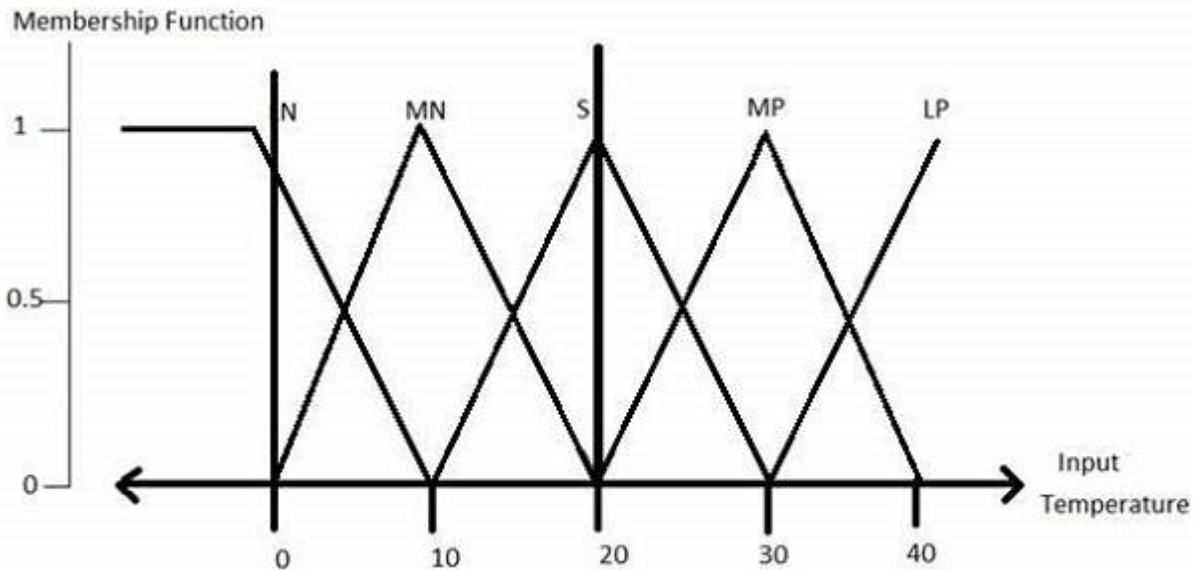
Linguistic variables are input and output variables in the form of simple words or sentences. For room temperature, cold, warm, hot, etc., are linguistic terms.

Temperature (t) = {very-cold, cold, warm, very-warm, hot}

Every member of this set is a linguistic term and it can cover some portion of overall temperature values.

Step 2 – Construct membership functions for them

The membership functions of temperature variable are as shown –



Step3 – Construct knowledge base rules

Create a matrix of room temperature values versus target temperature values that an air conditioning system is expected to provide.

RoomTemp./Target	Very_Cold	Cold	Warm	Hot	Very_Hot
Very_Cold	No_Change	Heat	Heat	Heat	Heat
Cold	Cool	No_Change	Heat	Heat	Heat
Warm	Cool	Cool	No_Change	Heat	Heat
Hot	Cool	Cool	Cool	No_Change	Heat
Very_Hot	Cool	Cool	Cool	Cool	No_Change

Build a set of rules into the knowledge base in the form of IF-THEN-ELSE structures.

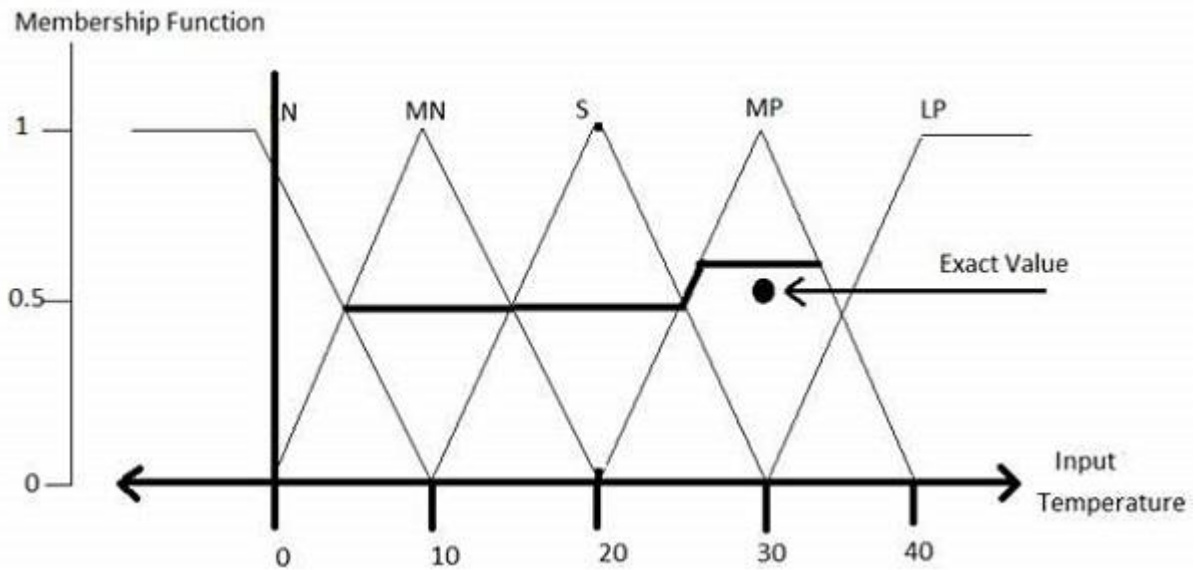
Sr. No.	Condition	Action
1	IF temperature=(Cold OR Very_Cold) AND target=Warm THEN	Heat
2	IF temperature=(Hot OR Very_Hot) AND target=Warm THEN	Cool
3	IF (temperature=Warm) AND (target=Warm) THEN	No_Change

Step 4 – Obtain fuzzy value

Fuzzy set operations perform evaluation of rules. The operations used for OR and AND are Max and Min respectively. Combine all results of evaluation to form a final result. This result is a fuzzy value.

Step 5 – Perform defuzzification

Defuzzification is then performed according to membership function for output variable.



Application Areas of Fuzzy Logic

The key application areas of fuzzy logic are as given –

Automotive Systems

- Automatic Gearboxes
- Four-Wheel Steering
- Vehicle environment control

Consumer Electronic Goods

- Hi-Fi Systems
- Photocopiers
- Still and Video Cameras
- Television

Domestic Goods

- Microwave Ovens
- Refrigerators
- Toasters
- Vacuum Cleaners
- Washing Machines

Environment Control

- Air Conditioners/Dryers/Heaters
- Humidifiers

Advantages of FLSs

- Mathematical concepts within fuzzy reasoning are very simple.
- You can modify a FLS by just adding or deleting rules due to flexibility of fuzzy logic.
- Fuzzy logic Systems can take imprecise, distorted, noisy input information.
- FLSs are easy to construct and understand.
- Fuzzy logic is a solution to complex problems in all fields of life, including medicine, as it resembles human reasoning and decision making.

Disadvantages of FLSs

- There is no systematic approach to fuzzy system designing.
- They are understandable only when simple.
- They are suitable for the problems which do not need high accuracy.

Fuzzy Logic - Membership Function

Membership Function

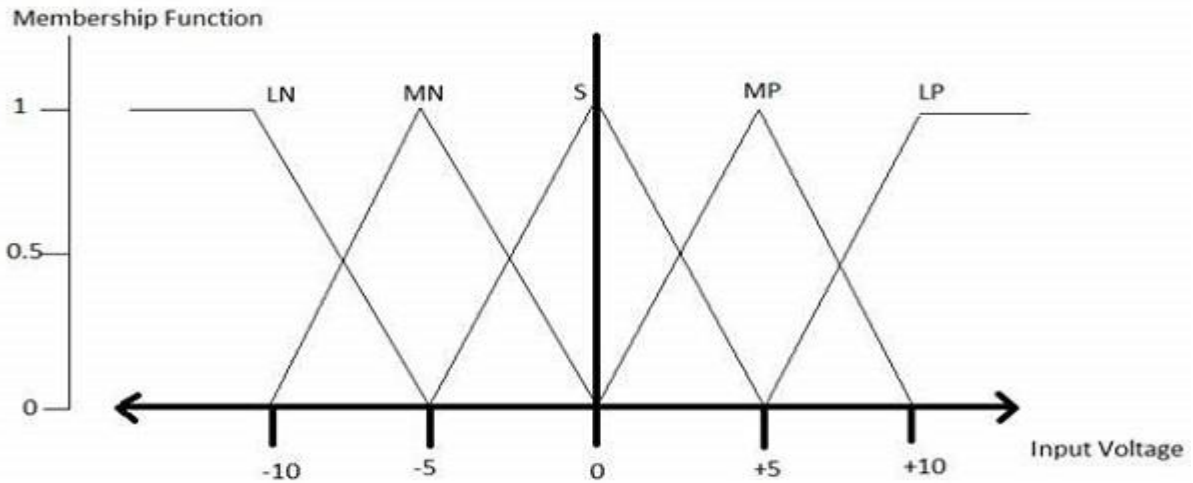
Membership functions allow you to quantify linguistic term and represent a fuzzy set graphically. A **membership function** for a fuzzy set A on the universe of discourse X is defined as $\mu_A: X \rightarrow [0,1]$.

Here, each element of X is mapped to a value between 0 and 1. It is called **membership value** or **degree of membership**. It quantifies the degree of membership of the element in X to the fuzzy set A .

- x axis represents the universe of discourse.
- y axis represents the degrees of membership in the [0, 1] interval.

There can be multiple membership functions applicable to fuzzify a numerical value. Simple membership functions are used as use of complex functions does not add more precision in the output.

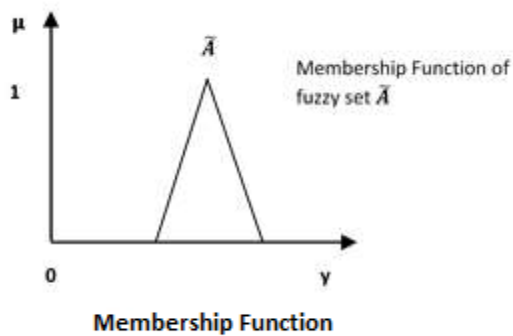
All membership functions for **LP**, **MP**, **S**, **MN**, and **LN** are shown as below –



The triangular membership function shapes are most common among various other membership function shapes such as trapezoidal, singleton, and Gaussian.

Here, the input to 5-level fuzzifier varies from -10 volts to +10 volts. Hence the corresponding output also changes.

Membership function defines the fuzziness in a fuzzy set irrespective of the elements in the set, which are discrete or continuous. The membership functions are generally represented in graphical form. There exist certain limitations for the shapes used to represent graphical form of membership function. The rules that describe fuzziness graphically are also fuzzy.



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 A in the universe of information U can be defined as a set of ordered pairs and it can be represented mathematically as –

$$A = \{(x, \mu_A(x)) | x \in X\}$$

Here $\mu_A(\cdot)$ = membership function of A ; this assumes values in the range from 0 to 1, i.e., $\mu_A(\cdot) \in [0, 1]$. The membership function $\mu_A(\cdot)$ maps X to the membership space M .

The dot (\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 , the core of a membership function is that region of universe that is characterized by full membership in the set. Hence, core consists of all those elements x of the universe of information such that,

$$\mu_A(x) = 1$$

Support

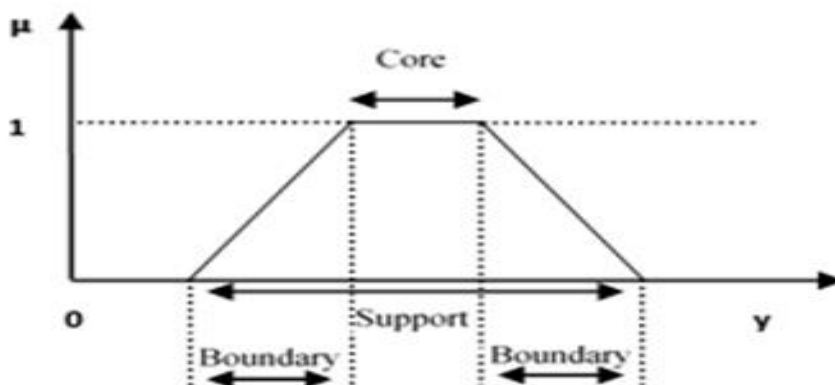
For any fuzzy set A , the support of a membership function is the region of universe that is characterized by a nonzero membership in the set. Hence core consists of all those elements x of the universe of information such that,

$$\mu_A(x) > 0$$

Boundary

For any fuzzy set A , the boundary of a membership function is the region of universe that is characterized by a nonzero but incomplete membership in the set. Hence, boundary consists of all those elements x of the universe of information such that,

$$1 > \mu_A(x) > 0$$



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 a fuzzy 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 –

$$A = \mu_1 Q(x_1) + \mu_2 Q(x_2) + \dots + \mu_n Q(x_n)$$

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

Grade Fuzzification (g-fuzzification) Method

It is quite similar to the above method but the main difference is that it kept x_i constant and μ_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 –

$$\mu_C(x^*) \geq \mu_C(x) \text{ for all } x \in X$$

Here, 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^* will be represented as –

$$x^* = \frac{\int \mu_C(x) \cdot x \, dx}{\int \mu_C(x) \, dx}$$

Weighted Average Method

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

$$x^* = \frac{\sum \mu_C(x_i^-) \cdot x_i^-}{\sum \mu_C(x_i^-)}$$

Mean-Max Membership

This method is also known as the middle of the maxima. Mathematically, the defuzzified output x^* will be represented as – $x^* = \sum_{i=1}^n x_i^- / n$

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.

Classical set is a collection of distinct objects. For example, a set of students passing grades.

Each individual entity in a set is called a member or an element of the set.

The classical set is defined in such a way that the universe of discourse is spitted into two groups members and non-members. Hence, In case classical sets, no partial membership exists.

Let A is a given set. The membership function can be use to define a set A is given by:

$$\mu_A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases}$$

Operations on classical sets:

For two sets A and B and Universe X:

Union:

$$A \cup B = \{x | x \in A \text{ or } x \in B\}$$

This operation is also called logical OR.

Intersection:

$$A \cap B = \{x | x \in A \text{ and } x \in B\}$$

This operation is also called logical AND.

Complement:

$$A' = \{x | x \notin A, x \in X\}$$

Difference:

$$A \setminus B = \{x | x \in A \text{ and } x \notin B\}$$

Properties of classical sets: For two sets A and B and Universe X:

Commutativity:

$$A \cup B = B \cup A$$

$$A \cap B = B \cap A$$

Associativity:

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

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

Distributivity:

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

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

Idempotency:

$$A \cup A = A$$

$$A \cap A = A$$

Identity:

$$A \cup \emptyset = A$$

$$A \cap X = A$$

$$A \cap \emptyset = \emptyset$$

$$A \cup X = X$$

Transitivity:

If $A \subseteq B$ and $B \subseteq C$, then $A \subseteq C$

Fuzzy set

Fuzzy set is a set having degrees of membership between 1 and 0. Fuzzy sets are represented with tilde character (~). For example, Number of cars following traffic signals at a particular time out of all cars present will have membership value between [0,1].

Partial membership exists when member of one fuzzy set can also be a part of other fuzzy sets in the same universe.

The degree of membership or truth is not same as probability, fuzzy truth represents membership in vaguely defined sets.

A fuzzy set A_{\sim} in the universe of discourse, U , can be defined as a set of ordered pairs and it is given by

$$\tilde{A} = \{(x, \mu_{\tilde{A}}(x)) | x \in X\}$$

When the universe of discourse, U, is discrete and finite, fuzzy set \tilde{A} is given by

$$\tilde{A} = \sum_{i=1}^n \frac{\mu_{\tilde{A}}(x_i)}{x_i} = \frac{\mu_{\tilde{A}}(x_1)}{x_1} + \frac{\mu_{\tilde{A}}(x_2)}{x_2} + \dots + \frac{\mu_{\tilde{A}}(x_n)}{x_n}$$

$$\tilde{A} = \int \frac{\mu_{\tilde{A}}(x)}{x}$$

where “n” is a finite value.

Fuzzy sets also satisfy every property of classical sets.


Operations on Fuzzy Sets

Given ‘X’ to be universe of discourse, A and B are two fuzzy sets with membership function $\mu_A(x)$ and $\mu_B(x)$ then,

Union

The union of two fuzzy sets A and B is a new fuzzy set $A \cup B$ also on ‘X’ with membership function defined as follow:


$$\mu_{(A \cup B)}(x) = \max(\mu_A(x), \mu_B(x)) = \mu_A(x) \vee \mu_B(x)$$


 max operator

Intersection

Intersection of fuzzy sets A & B, is a new fuzzy set $A \cap B$ also on ‘X’ whose membership function is defined by

$$\mu_{(A \cap B)} = \min(\mu_A(x), \mu_B(x)) = \mu_A(x) \wedge \mu_B(x)$$


 minimum operator

Compliment

Compliment of a fuzzy set A is \bar{A} with membership function

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x)$$

Product of Two Fuzzy Sets

The product of two fuzzy sets A & B is a new fuzzy set $A.B$ with membership function:

$$\mu_{A.B}(x) = \mu_A(x) \cdot \mu_B(x)$$

Equality

Two fuzzy sets A and B are said to be equal i.e, $A = B$ if and only if $\mu_A(x) = \mu_B(x)$ Which means their membership values must be equal.

Product of Fuzzy Sets with a Crisp Number

Multiplying a fuzzy set A by a crisp number 'n' results in a new fuzzy set n.A, whose membership function is

$$\mu_{n.A}(x) = n \cdot \mu_A(x)$$

Power of a Fuzzy Set

The alpha power of a fuzzy set A is a new fuzzy set A^α whose membership function is:

$$\mu_{A^\alpha}(x) = [\mu_A(x)]^\alpha$$

that is, individual memberships power of α

Difference of Fuzzy Sets

The differences of two fuzzy sets A and B is a new fuzzy set A-B which is defined as

$$A - B = (A \cap \overline{B})$$

Disjunctive Sum of A & B

It is the new fuzzy set defined as follow:

$$A \oplus B = (\overline{A} \cap B) \cup (A \cap \overline{B})$$

Aggregation operation

Aggregation operations on fuzzy sets are operations by which several fuzzy sets are combined in a desirable way to produce a single fuzzy set.

Aggregation operation on n fuzzy set ($2 \leq n$) is defined by a function

$$h: [0,1]^n \rightarrow [0,1]$$

Axioms for aggregation operations fuzzy sets[edit]

Axiom h1. Boundary condition

$$h(0, 0, \dots, 0) = 0 \text{ and } h(1, 1, \dots, 1) = \text{one}$$

Axiom h2. Monotonicity

For any pair $\langle a_1, a_2, \dots, a_n \rangle$ and $\langle b_1, b_2, \dots, b_n \rangle$ of n -tuples such that $a_i, b_i \in [0,1]$ for all $i \in N_n$, if $a_i \leq b_i$ for all $i \in N_n$, then $h(a_1, a_2, \dots, a_n) \leq h(b_1, b_2, \dots, b_n)$; that is, h is monotonic increasing in all its arguments.

Axiom h3. Continuity

h is a continuous function.

NEURO FUZZY SYSTEMS

Neuro-fuzzy hybridization results in a hybrid intelligent system that synergizes these two techniques by combining the human-like reasoning style of fuzzy systems with the learning and connectionist structure of neural networks. Neuro-fuzzy hybridization is widely termed as fuzzy neural network (FNN) or neuro-fuzzy system (NFS) in the literature. Neuro-fuzzy system (the more popular term is used henceforth) incorporates the human-like reasoning style of fuzzy

systems through the use of fuzzy sets and a linguistic model consisting of a set of IF-THEN fuzzy rules. The main strength of neuro-fuzzy systems is that they are universal approximators with the ability to solicit interpretable IF-THEN rules.

The strength of neuro-fuzzy systems involves two contradictory requirements in fuzzy modeling: interpretability versus accuracy. In practice, one of the two properties prevails. The neuro-fuzzy in fuzzy modeling research field is divided into two areas: linguistic fuzzy modeling that is focused on interpretability, mainly the Mamdani model; and precise fuzzy modeling that is focused on accuracy, mainly the Takagi-Sugeno-Kang (TSK) model.

Although generally assumed to be the realization of a fuzzy system through connectionist networks, this term is also used to describe some other configurations including:

- Deriving fuzzy rules from trained RBF networks.
- Fuzzy logic based tuning of neural network training parameters.
- Fuzzy logic criteria for increasing a network size.
- Realising fuzzy membership function through clustering algorithms in unsupervised learning in SOMs and neural networks.
- Representing fuzzification, fuzzy inference and defuzzification through multi-layers feed-forward connectionist networks.

It must be pointed out that interpretability of the Mamdani-type neuro-fuzzy systems can be lost. To improve the interpretability of neuro-fuzzy systems, certain measures must be taken, wherein important aspects of interpretability of neuro-fuzzy systems are also discussed.^[2]

A recent research line addresses the data stream mining case, where neuro-fuzzy systems are sequentially updated with new incoming samples on demand and on-the-fly. Thereby, system updates do not only include a recursive adaptation of model parameters, but also a dynamic evolution and pruning of model components (neurons, rules), in order to handle concept drift and dynamically changing system behavior adequately and to keep the systems/models "up-to-date" anytime. Comprehensive surveys of various evolving neuro-fuzzy systems approaches can be found in ^[3] and ^[4]

Pseudo outer-product based fuzzy neural networks

Pseudo outer product-based fuzzy neural networks (POPFNN) are a family of neuro-fuzzy systems that are based on the linguistic fuzzy model.^[5]

Three members of POPFNN exist in the literature:

- **POPFNN-AARS(S)**, which is based on the Approximate Analogical Reasoning Scheme^[6]
- **POPFNN-CRI(S)**, which is based on commonly accepted fuzzy Compositional Rule of Inference^[7]
- **POPFNN-TVR**, which is based on Truth Value Restriction

The "POPFNN" architecture is a five-layer neural network where the layers from 1 to 5 are called: input linguistic layer, condition layer, rule layer, consequent layer, output linguistic layer. The fuzzification of the inputs and the defuzzification of the outputs are respectively performed by the input linguistic and output linguistic layers while the fuzzy inference is collectively performed by the rule, condition and consequence layers.

The learning process of POPFNN consists of three phases:

1. Fuzzy membership generation

2. Fuzzy rule identification
3. Supervised fine-tuning

Various fuzzy membership generation algorithms can be used: Learning Vector Quantization (LVQ), Fuzzy Kohonen Partitioning (FKP) or Discrete Incremental Clustering (DIC). Generally, the POP algorithm and its variant LazyPOP are used to identify the fuzzy rules.

Introduction to 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 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.

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.

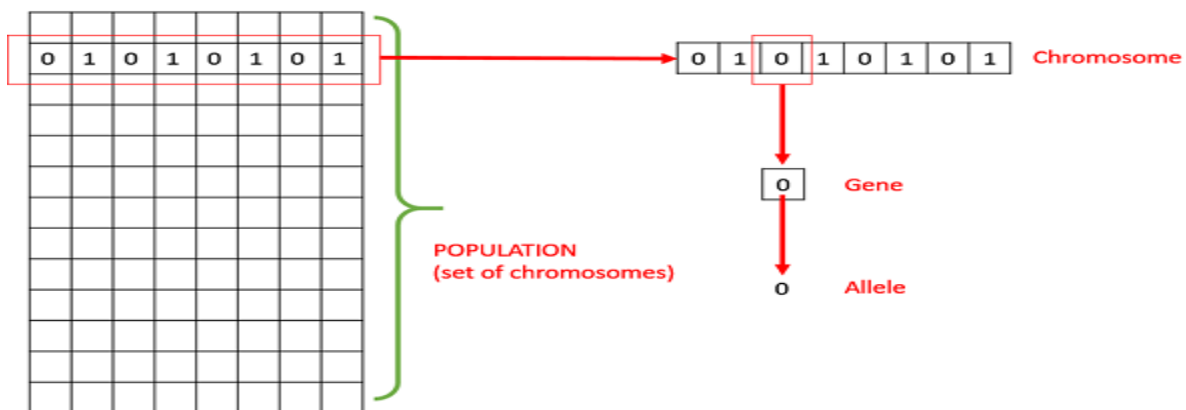
- Being stochastic, there are no guarantees on the optimality or the quality of the solution.
- If not implemented properly, the GA may not converge to the optimal solution.

Genetic Algorithms – Fundamentals

Basic Terminology

Before beginning a discussion on Genetic Algorithms, it is essential to be familiar with some basic terminology which will be used throughout this tutorial.

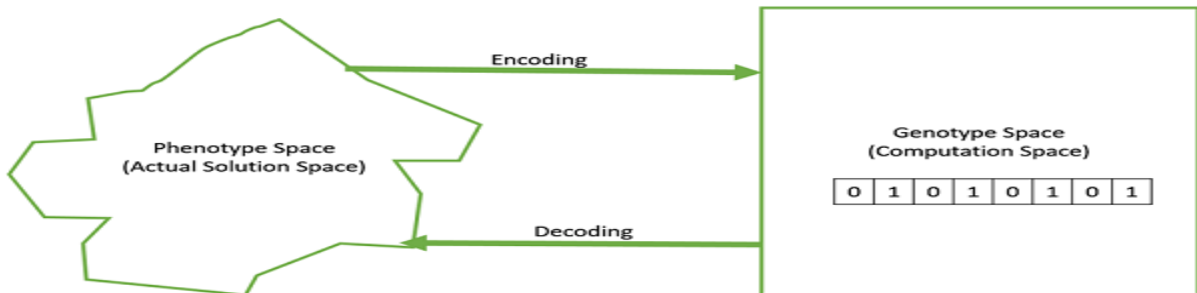
- **Population** – It is a subset of all the possible (encoded) solutions to the given problem. The population for a GA is analogous to the population for human beings except that instead of human beings, we have Candidate Solutions representing human beings.
- **Chromosomes** – A chromosome is one such solution to the given problem.
- **Gene** – A gene is one element position of a chromosome.
- **Allele** – It is the value a gene takes for a particular chromosome.



- **Genotype** – Genotype is the population in the computation space. In the computation space, the solutions are represented in a way which can be easily understood and manipulated using a computing system.
- **Phenotype** – Phenotype is the population in the actual real world solution space in which solutions are represented in a way they are represented in real world situations.
- **Decoding and Encoding** – For simple problems, the **phenotype and genotype** spaces are the same. However, in most of the cases, the phenotype and genotype spaces are different. **Decoding** is a process of transforming a solution from the genotype to the phenotype space, **while encoding** is a process of transforming from the phenotype to genotype space. Decoding should be fast as it is carried out repeatedly in a GA during the fitness value calculation.

For example, consider the 0/1 Knapsack Problem. The Phenotype space consists of solutions which just contain the item numbers of the items to be picked.

However, in the genotype space it can be represented as a binary string of length n (where n is the number of items). A **0 at position x** represents that x^{th} item is picked while a 1 represents the reverse. This is a case where genotype and phenotype spaces are different.

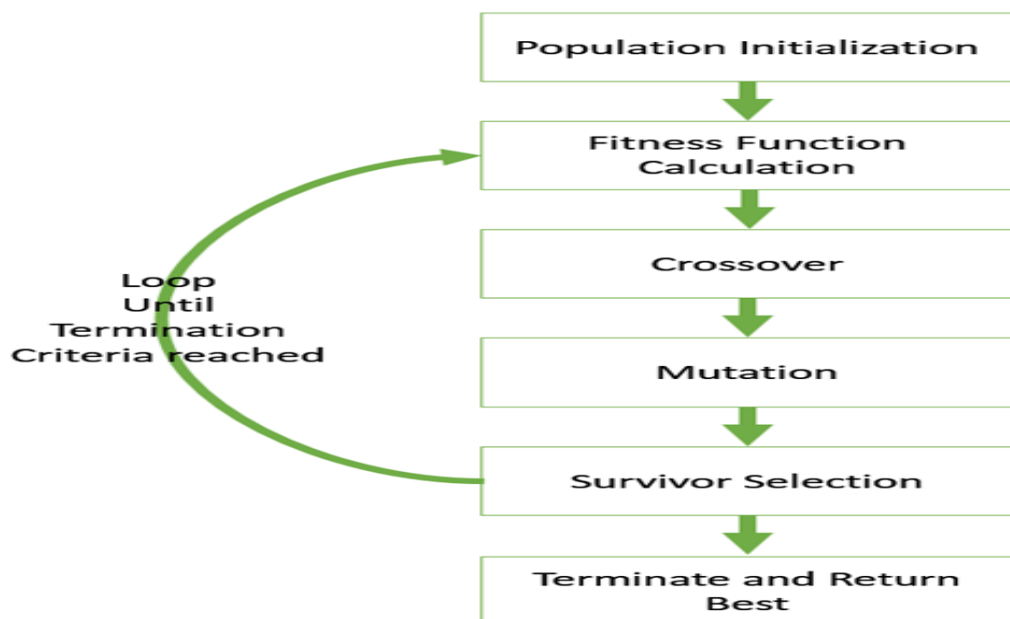


- **Fitness Function** – A fitness function simply defined is a function which takes the solution as input and produces the suitability of the solution as the output. In some cases, the fitness function and the objective function may be the same, while in others it might be different based on the problem.
- **Genetic Operators** – These alter the genetic composition of the offspring. These include crossover, mutation, selection, etc.

Basic Structure of GA

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.



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
```

Genotype Representation

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.

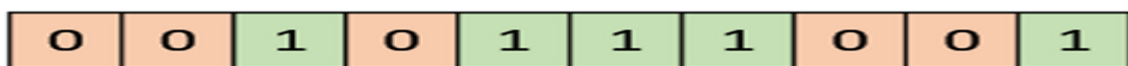
Therefore, choosing a proper representation, having a proper definition of the mappings between the phenotype and genotype spaces is essential for the success of a GA.

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.

1.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 x^{th} element tells whether the item x is picked (1) or not (0).



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.

2.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
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

3.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
---	---	---	---	---	---	---	---	---	---

4.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

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.**

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” or 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.

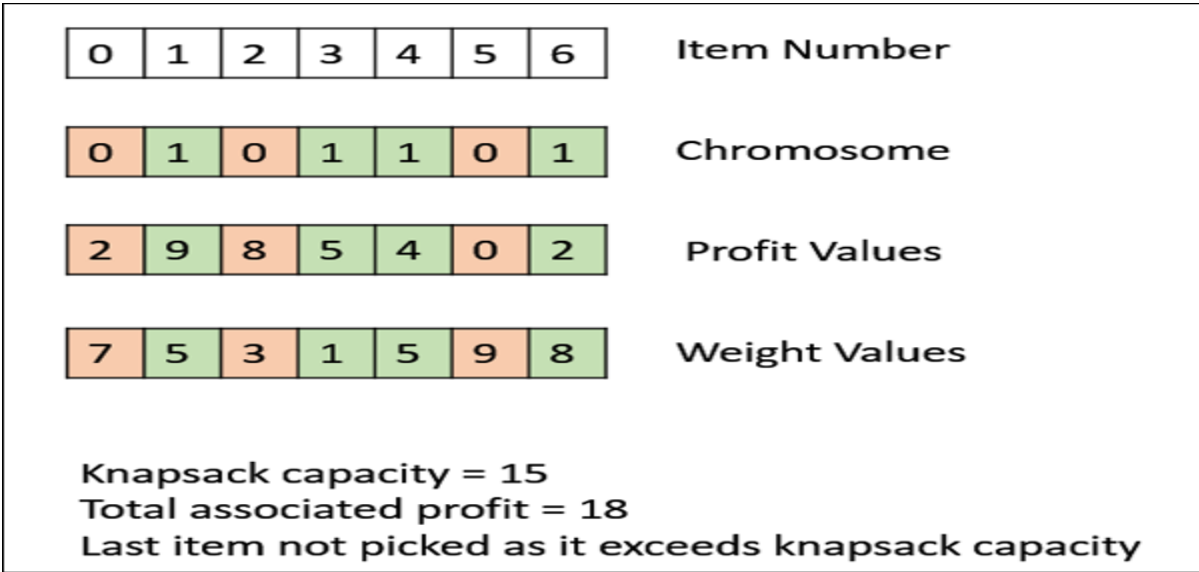
In most cases the fitness function and the objective function are the same as the objective is to either maximize or minimize the given objective function. However, for more complex problems with multiple objectives and constraints, an **Algorithm Designer** might choose to have a different fitness function.

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.



Genetic Operators-

The basic operators that are to be discussed in this section include: encoding, selection, recombination and mutation operators.

1. Encoding: Encoding is a process of representing individual genes. The process can be performed using bits, numbers, trees, arrays, lists or any other objects. The encoding depends mainly on solving the problem. For example, one can encode directly real or integer numbers.

1. 1 Binary Encoding

The most common way of encoding is a binary string, which would be represented as in Each chromosome encodes a binary (bit) string. Each bit in the string can represent some characteristics of the solution. Every bit string therefore is a solution but not necessarily the best solution. Another possibility is that the whole string can represent a number. The way bit strings can code differs from problem to problem.

Binary encoding gives many possible chromosomes with a smaller number of alleles. On the other hand, this encoding is not natural for many problems and sometimes corrections must be made after genetic operation is completed. Binary coded strings with 1s and 0s are mostly used.

Chromosome1	110100110010
Chromosome2	100100100100

The length of the string depends on the accuracy. In such coding

1. Integers are represented exactly.
2. Finite number of real numbers can be represented.
3. Number of real numbers represented increases with string length.

1.2 Octal Encoding

This encoding uses string made up of octal numbers (0-7)

Chromosome1	0123421065
Chromosome2	7654343210

1.3 Hexadecimal Encoding

This encoding uses string made up of hexadecimal numbers (0-9, A-F)

Chromosome1	05AB
Chromosome2	3DBA

1.4 Permutation Encoding (Real Number Coding)

Every chromosome is a string of numbers, represented in a sequence. Sometimes corrections have to be done after genetic operation is complete. In permutation encoding, every chromosome is a string of integer/real values, which represents number in a sequence. Permutation encoding is only useful for ordering problems. Even for this problem, some types of crossover and mutation corrections must be made to leave the chromosome consistent (i.e., have real sequence in it).

Chromosome1	145678239
Chromosome2	567812349

1.5 Value Encoding

Every chromosome is a string of values and the values can be anything connected to the problem. This encoding produces best results for some special problems. On the other hand, it is often necessary to develop new genetic operator's specific to the problem. Direct value encoding can be used in problems, where some complicated values, such as real numbers, are

used. Use of binary encoding for this type of problems would be very difficult.

Chromosome A	1.235 5.323 0.454 2.321 2.454
Chromosome B	(left), (back), (left), (right), (forward)

Genetic operators: - 2. Parent Selection

Parent Selection is the process of selecting parents which mate and recombine to create offsprings 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.

1. Fitness Proportionate Selection

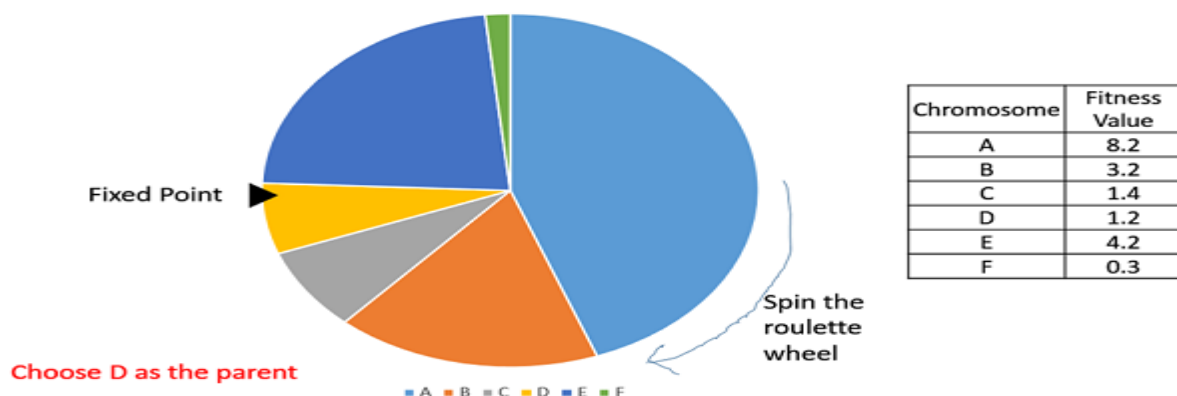
Fitness Proportionate Selection is one of the most popular ways of parent selection. In this every individual can become a parent with a probability which is proportional to its fitness. Therefore, fitter individuals have a higher chance of mating and propagating their features to the next generation.

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 –

1.a Roulette Wheel Selection

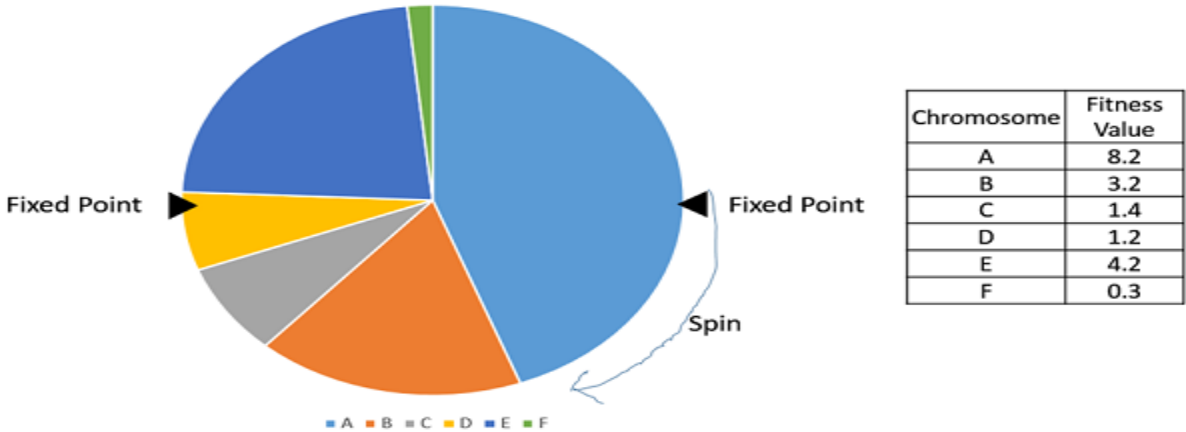
In a roulette wheel selection, the circular wheel is divided as described before. A fixed point is chosen on the wheel circumference as shown and the wheel is rotated. The region of the wheel which comes in front of the fixed point is chosen as the parent. For the second parent, the same process is repeated.



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.

1.b Stochastic Universal Sampling (SUS)

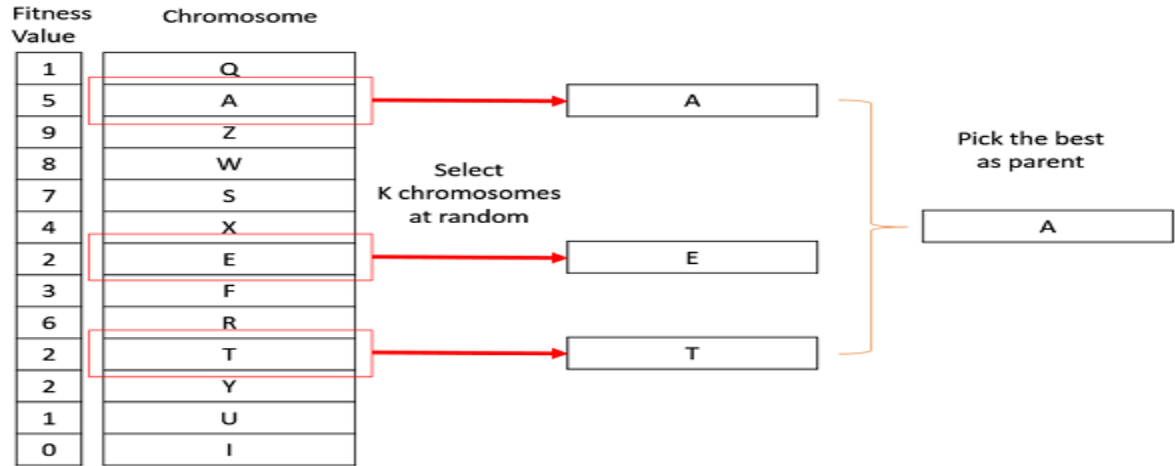
Stochastic Universal Sampling is quite similar to Roulette wheel selection, however instead of having just one fixed point, we have multiple fixed points as shown in the following image. Therefore, all the parents are chosen in just one spin of the wheel. Also, such a setup encourages the highly fit 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.

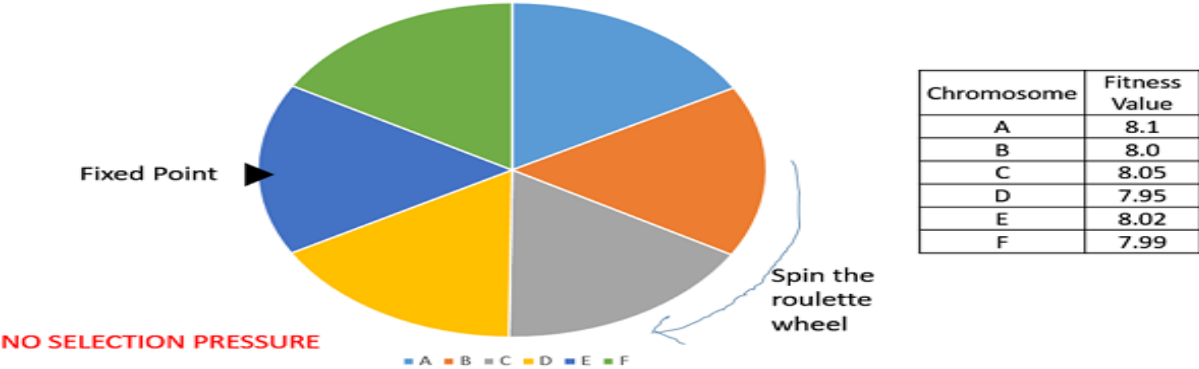
2. Tournament Selection

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.



3. 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.



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

4.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

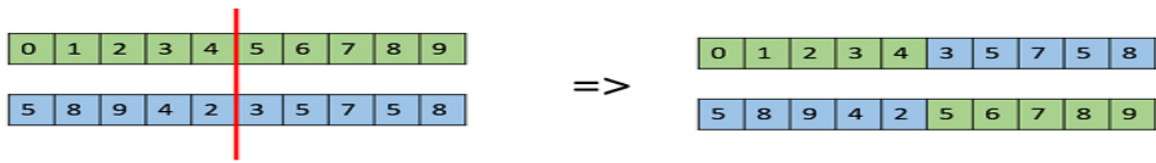
Genetic Operators 3. Crossover(Recombination)

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

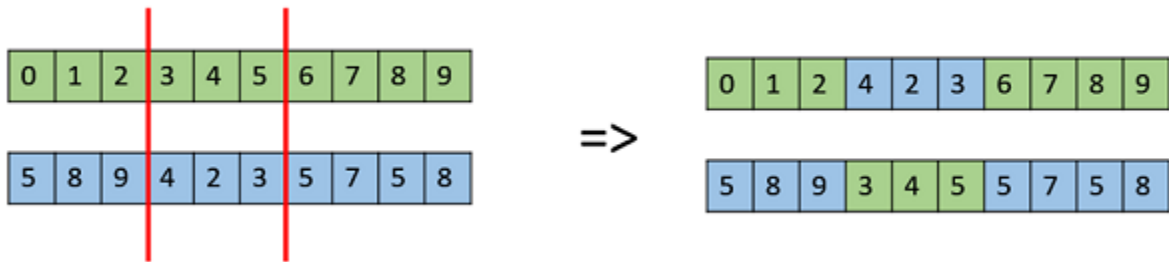
a. One Point Crossover

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.



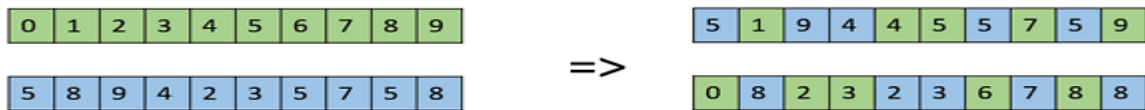
b. Multi Point Crossover

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



c. 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.



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

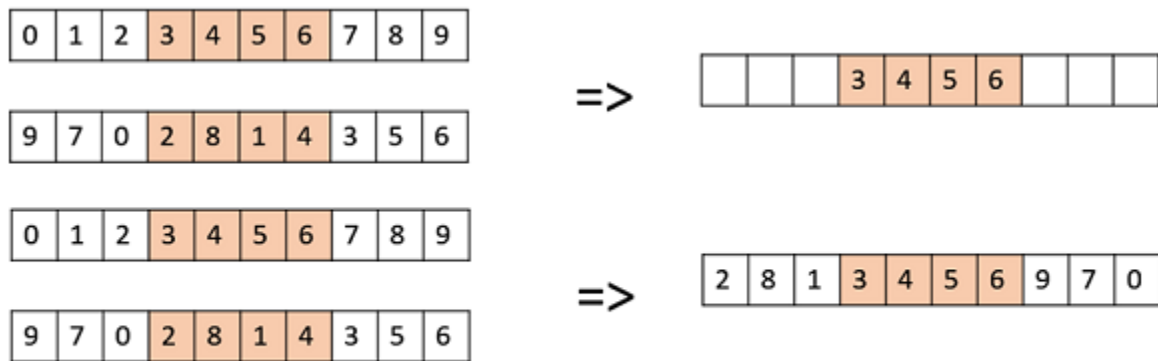
Obviously, if $\alpha = 0.5$, then both the children will be identical as shown in the following image.



e. 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 Operators– 4.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.

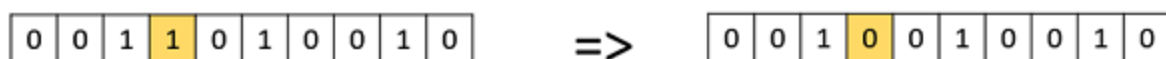
Mutation is the part of the GA which is related to the “exploration” of the search space. It has been observed that mutation is essential to the convergence of the GA while crossover is not.

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.

a.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.

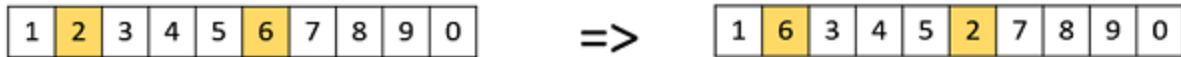


b. Random Resetting

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.

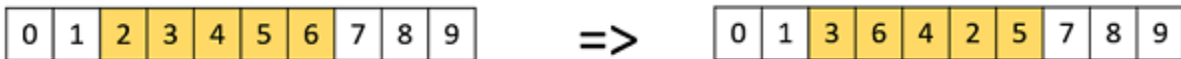
c. Swap Mutation

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



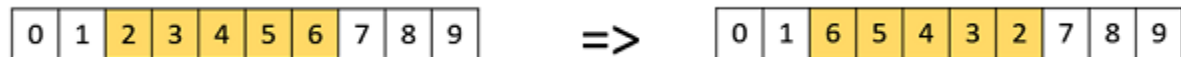
d. Scramble Mutation

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.



e. 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.



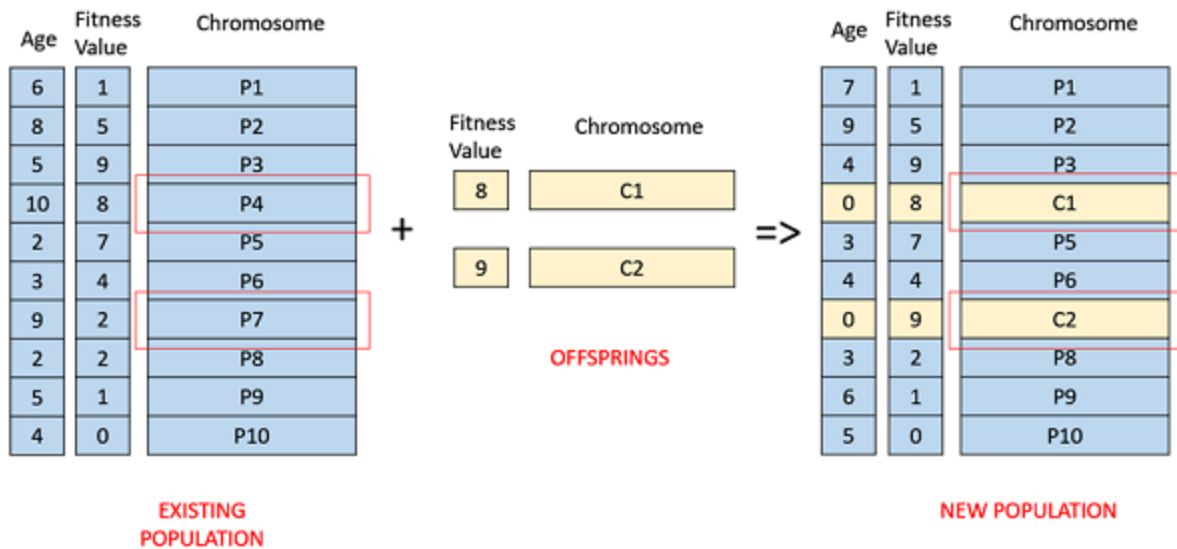
Genetic Algorithms - Survivor Selection

The Survivor Selection Policy determines which individuals are to be kicked out and which are to be kept in the next generation. 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.

1. 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.

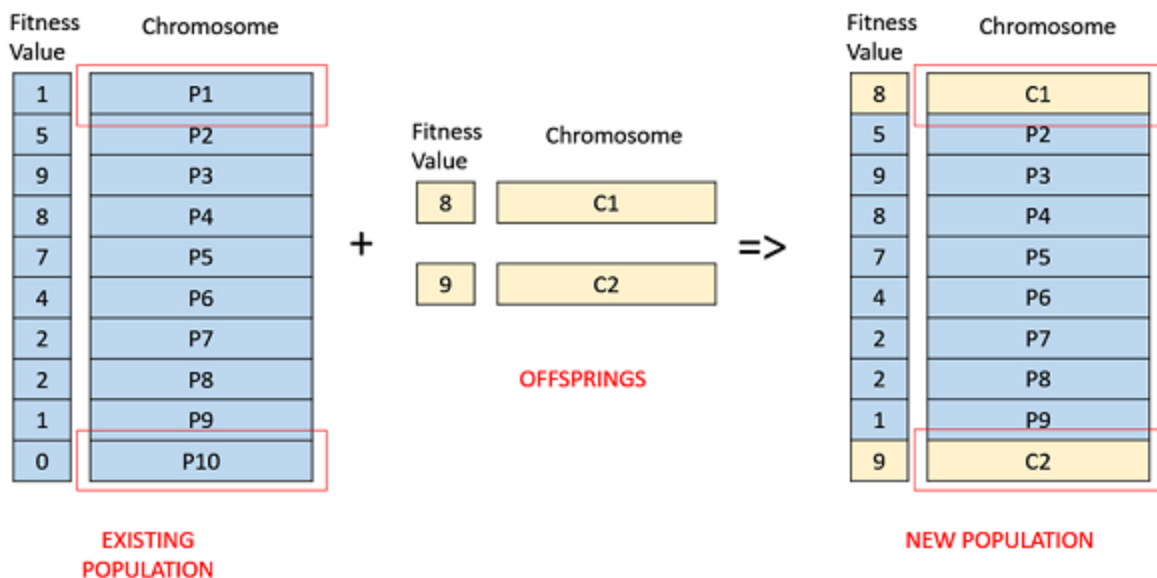
For instance, in the following example, the age is the number of generations for which the individual has been in the population. The oldest members of the population i.e. P4 and P7 are kicked out of the population and the ages of the rest of the members are incremented by one.



2. 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. 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 evolving a neural network

Here, we try to improve upon the brute force method by applying a genetic algorithm to evolve a network with the goal of achieving optimal hyper parameters in a fraction the time of a brute force search.

Let's say it takes five minutes to train and evaluate a network on your dataset. And let's say we have four parameters with five possible settings each. To try them all would take $(5^{*}4) * 5$ minutes, or 3,125 minutes, or about 52 hours.

Now let's say we use a genetic algorithm to evolve 10 generations with a population of 20 (more on what this means below), with a plan to keep the top 25% plus a few more, so ~8 per generation. This means that in our first generation we score 20 networks ($20 * 5 = 100$ minutes). Every generation after that only requires around 12 runs, since we don't have the score the ones we keep. That's $100 + (9 \text{ generations} * 5 \text{ minutes} * 12 \text{ networks}) = 640$ minutes, or 11 hours.

We've just reduced our parameter tuning time by almost 80%! That is, assuming it finds the best parameters...

Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection. — Wikipedia

How do genetic algorithms work?

At its core, a genetic algorithm...

1. Creates a population of (randomly generated) members
2. Scores each member of the population based on some goal. This score is called a fitness function.

3. Selects and *breeds* the best members of the population to produce more like them
4. Mutates some members randomly to attempt to find even better candidates
5. Kills off the rest (survival of the fittest and all), and
6. Repeats from step 2. Each iteration through these steps is called a generation.

Repeat this process enough times and you should be left with the very best *possible* members of a population. Sounds like a lot evolution, right? Same deal.

Applying genetic algorithms to Neural Networks

We'll attempt to evolve a fully connected network (MLP). Our goal is to find the best parameters for an image classification task.

We'll tune four parameters:

- Number of layers (or the network depth)
- Neurons per layer (or the network width)
- Dense layer activation function
- Network optimizer

The steps we'll take to evolve the network, similar to those described above, are:

1. Initialize N random networks to create our population.
2. Score each network. This takes some time: We have to train the weights of each network and then see how well it performs at classifying the test set. Since this will be an image classification task, we'll use classification accuracy as our fitness function.
3. Sort all the networks in our population by score (accuracy). We'll keep some percentage of the top networks to become part of the next generation and to breed children.
4. We'll also randomly keep a few of the non-top networks. This helps find potentially lucky combinations between worse-performers and top performers, and also helps keep us from getting stuck in a local maximum.
5. Now that we've decided which networks to keep, we randomly mutate some of the parameters on some of the networks.
6. Here comes the fun part: Let's say we started with a population of 20 networks, we kept the top 25% (5 nets), randomly kept 3 more loser networks, and mutated a few of them.

We let the other 12 networks die. In an effort to keep our population at 20 networks, we need to fill 12 open spots. It's time to breed!

Breeding

Breeding is where we take two members of a population and generate one or more child, where that child represents a combination of its parents.

In our neural network case, each child is a combination of a random assortment of parameters from its parents. For instance, one child might have the same number of layers as its mother and the rest of its parameters from its father. A second child of the same parents may have the opposite. You can see how this mirrors real-world biology and how it can lead to an optimized network quickly.