Introduction

Basics of Soft Computing

Soft Computing

Introduction to Soft Computing, topics : Definitions, goals, and importance. Fuzzy computing : classical set theory, crisp & non-crisp set, capturing uncertainty, definition of fuzzy set, graphic interpretations of fuzzy set - small, prime numbers, universal space, empty. Fuzzy operations : inclusion, equality, comparability, complement, union, intersection. Neural computing : biological model, information flow in neural cell. Artificial neuron - functions, equation, elements, single and multi layer perceptrons. Genetic Algorithms : mechanics of biological evolution, taxonomy of artificial evolution & search optimization - enumerative, calculus-based and guided random search techniques, evolutionary algorithms (EAs). Associative memory : description of AM, examples of auto and hetero AM. Adaptive Resonance Theory : definitions of ART and other types of learning, ART description, model functions, training, and systems.

Introduction

Basics of Soft Computing

Soft Computing

Topics

(Lectures 01, 02, 03, 04, 05, 06 6 hours)

Introduction

Basics of Soft Computing

What is Soft Computing ?

- The idea of soft computing was initiated in 1981 when Lotfi A. *Zadeh* published his first paper on soft data analysis "What is Soft Computing", Soft Computing. Springer-Verlag Germany/USA 1997.].
- Zadeh, defined Soft Computing into one multidisciplinary system as the fusion of the fields of Fuzzy Logic, Neuro-Computing, Evolutionary and Genetic Computing, and Probabilistic Computing.
- Soft Computing is the fusion of methodologies designed to model and enable solutions to real world problems, which are not modeled or too difficult to model mathematically.
- The aim of Soft Computing is to exploit the tolerance for imprecision, uncertainty, approximate reasoning, and partial truth in order to achieve close resemblance with human like decision making.
- The Soft Computing – development history

SC	=	***EC***	+	***NN***	+	***FL***
Soft Computing		**Evolutionary Computing**		**Neural Network**		**Fuzzy Logic**
Zadeh 1981		**Rechenberg 1960**		**McCulloch 1943**		**Zadeh 1965**

EC	=	***GP***	+	***ES***	+	***EP***	+	***GA***
Evolutionary Computing		***Genetic Programming***		***Evolution Strategies***		***Evolutionary Programming***		***Genetic Algorithms***
Rechenberg 1960		**Koza 1992**		**Rechenberg 1965**		**Fogel 1962**		**Holland 1970**

1. Introduction

To begin, first explained, the definitions, the goals, and the importance of the soft computing. Later, presented its different fields, that is, Fuzzy Computing, Neural Computing, Genetic Algorithms, and more.

Definitions of Soft Computing (SC)

Lotfi A. Zadeh, 1992 : "Soft Computing is an emerging approach to computing which parallel the remarkable ability of the human mind to reason and learn in a environment of uncertainty and imprecision".

The Soft Computing consists of several computing paradigms mainly : **Fuzzy Systems, Neural Networks, and Genetic Algorithms.**

- Fuzzy set : for knowledge representation via fuzzy If – Then rules.
- Neural Networks : for learning and adaptation
- Genetic Algorithms : for evolutionary computation

These methodologies form the core of SC.
Hybridization of these three creates a successful synergic effect;
that is, hybridization creates a situation where different entities cooperate advantageously for a final outcome.

Soft Computing is still growing and developing.
Hence, a clear definite agreement on what comprises Soft Computing has not yet been reached. More new sciences are still merging into Soft Computing.

Goals of Soft Computing

Soft Computing is a new multidisciplinary field, to construct new generation of Artificial Intelligence, known as **Computational Intelligence**.

- The main goal of Soft Computing is to develop intelligent machines to provide solutions to real world problems, which are not modeled, or too difficult to model mathematically.

- Its aim is to exploit the tolerance for **Approximation, Uncertainty**, **Imprecision**, and **Partial Truth** in order to achieve close resemblance with human like decision making.

  Approximation : here the model features are similar to the real ones, but not the same.

  Uncertainty : here we are not sure that the features of the model are the same as that of the entity (belief).

  Imprecision : here the model features (quantities) are not the same as that of the real ones, but close to them.

Importance of Soft Computing

Soft computing differs from hard (conventional) computing. Unlike hard computing, the soft computing is **tolerant of imprecision, uncertainty, partial truth, and approximation**. The guiding principle of soft computing is to exploit these tolerance to achieve tractability, robustness and low solution cost. In effect, the role model for soft computing is the human mind.

The four fields that constitute Soft Computing (SC) are : **Fuzzy Computing** (FC), **Evolutionary Computing** (EC), **Neural computing** (NC), and **Probabilistic Computing** (PC), with the latter subsuming belief networks, chaos theory and parts of learning theory.

Soft computing is not a concoction, mixture, or combination, rather, **Soft computing is a partnership** in which each of the partners contributes a distinct methodology for addressing problems in its domain. In principal the constituent methodologies in Soft computing are complementary rather than competitive.

Soft computing may be viewed as a foundation component for the emerging field of Conceptual Intelligence.

2. Fuzzy Computing

In the real world there exists much fuzzy knowledge, that is, knowledge which is vague, imprecise, uncertain, ambiguous, inexact, or probabilistic in nature.

Human can use such information because the human thinking and reasoning frequently involve fuzzy information, possibly originating from inherently inexact human concepts and matching of similar rather then identical experiences.

The computing systems, based upon classical set theory and two-valued logic, can not answer to some questions, as human does, because they do not have completely true answers.

We want, the computing systems should not only give human like answers but also describe their reality levels. These levels need to be calculated using imprecision and the uncertainty of facts and rules that were applied.

2.1 Fuzzy Sets

Introduced by Lotfi Zadeh in 1965, the fuzzy set theory is an extension of classical set theory where elements have degrees of membership.

- **Classical Set Theory**

  – **Sets** are defined by a simple statement describing whether an element having a certain property belongs to a particular set.

  – When set **A** is contained in an universal space **X**, then we can state explicitly whether each element **x** of space **X** "is or is not" an element of **A**.

  – Set **A** is well described by a function called **characteristic function A**. This function, defined on the universal space **X**, assumes :
  value **1** for those elements **x** that belong to set **A**, and
  value **0** for those elements **x** that do not belong to set **A**.
  The notations used to express these mathematically are

$$\left.\begin{array}{l} A : X \rightarrow [0, 1] \\ A(x) = 1 ,\ x \text{ is a member of } A \\ A(x) = 0 ,\ x \text{ is not a member of } A \end{array}\right\} \quad \text{Eq.(1)}$$

  Alternatively, the set **A** can be represented for all elements $x \in X$ by its **characteristic function** $\mu_A(x)$ defined as

$$\mu_A(x) = \begin{cases} 1 & \text{if } x \in X \\ 0 & \text{otherwise} \end{cases} \quad \text{Eq.(2)}$$

  – Thus, in classical set theory $\mu_A(x)$ has only the values **0** ('false') and **1** ('true''). Such sets are called **crisp sets.**

- **Crisp and Non-crisp Set**

  – As said before, in classical set theory, the **characteristic function** $\mu_A(x)$ of Eq.(2) has only values **0** ('false') and **1** ('true'').
  Such sets are **crisp sets.**

  – For Non-crisp sets the characteristic function $\mu_A(x)$ can be defined.
  - The characteristic function $\mu_A(x)$ of Eq. (2) for the crisp set is generalized for the Non-crisp sets.
  - This generalized characteristic function $\mu_A(x)$ of Eq.(2) is called **membership function.**

  Such Non-crisp sets are called **Fuzzy Sets.**

  – Crisp set theory is not capable of representing descriptions and classifications in many cases; In fact, Crisp set does not provide adequate representation for most cases.

  – The proposition of Fuzzy Sets are motivated by the need to capture and represent real world data with uncertainty due to imprecise measurement.

  – The uncertainties are also caused by vagueness in the language.

- **Example 1 : Heap Paradox**

  This example represents a situation where vagueness and uncertainty are inevitable.

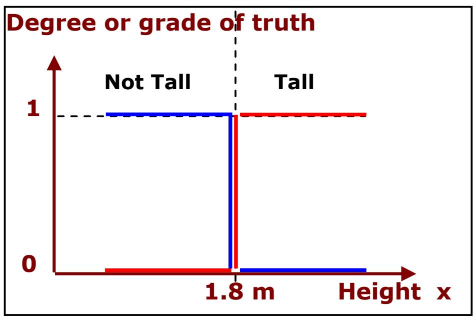
  - If we remove one grain from a heap of grains, we will still have a heap.
  - However, if we keep removing one-by-one grain from a heap of grains, there will be a time when we do not have a heap anymore.
  - The question is, at what time does the heap turn into a countable collection of grains that do not form a heap? There is no one correct answer to this question.

- **Example 2 : Classify Students for a basketball team**

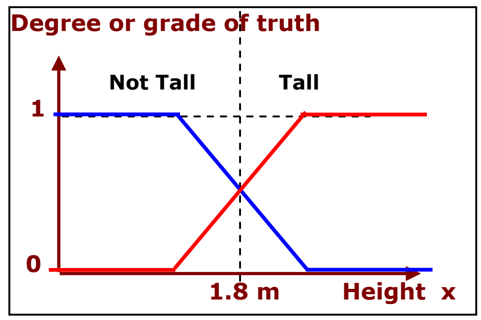
This example explains the grade of truth value.

- **tall students** qualify and **not tall students** do not qualify
- if students 1.8 m tall are to be qualified, then should we exclude a student who is $^1/_{10}$" less? or should we exclude a student who is 1" shorter?

- Non-Crisp Representation to represent the notion of a tall person.



Crisp logic



Non-crisp logic

Fig. 1 Set Representation – Degree or grade of truth

A student of height 1.79m would belong to both tall and not tall sets with a particular degree of membership.

As the height increases the membership grade within the tall set would increase whilst the membership grade within the not-tall set would decrease.

- **Capturing Uncertainty**

Instead of avoiding or ignoring uncertainty, Lotfi Zadeh introduced Fuzzy Set theory that captures uncertainty.

- A fuzzy set is described by a **membership function** $\mu_A(x)$ of **A**. This membership function associates to each element $x_\sigma \in X$ a number as $\mu_A(x_\sigma)$ in the closed unit interval **[0, 1]**.

  The number $\mu_A(x_\sigma)$ represents the **degree of membership** of x_σ in **A**.

- The notation used for membership function $\mu_A(x)$ of a fuzzy set **A** is

  $A : X \to [0, 1]$

- Each membership function maps elements of a given universal base set **X** , which is itself a crisp set, into real numbers in **[0, 1]** .

- Example

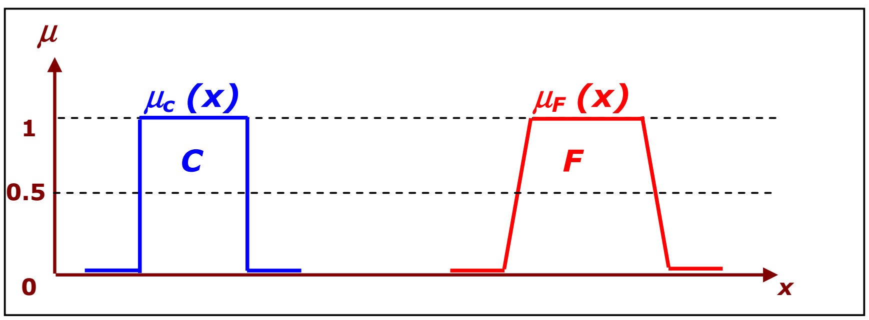


Fig. 2 Membership function of a Crisp set C and Fuzzy set F

- In the case of Crisp Sets the members of a set are :

  either out of the set, with membership of degree " **0** ",

  or in the set, with membership of degree " **1** ",

  Therefore, **Crisp Sets $\subseteq$ Fuzzy Sets**

  In other words, Crisp Sets are Special cases of Fuzzy Sets.

  [Continued in next slide]

[Continued from previous slide :]

Example 1: Set of prime numbers (a crisp set)

If we consider space **X** consisting of natural numbers ≤ **12**

ie **X = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}**

Then, the set of prime numbers could be described as follows.

PRIME = {x contained in X | x is a prime number} = {2, 3, 5, 6, 7, 11}

Example 2: Set of SMALL (as non-crisp set)

A Set **X** that consists of SMALL cannot be described;

for example **1** is a member of SMALL and **12** is not a member of SMALL.

Set **A**, as SMALL, has un-sharp boundaries, can be characterized by a function that assigns a real number from the closed interval from **0** to **1** to each element **x** in the set **X**.

- **Definition of Fuzzy Set**

A **fuzzy set A** defined in the universal space **X** is a function defined in **X** which assumes values in the range **[0, 1]**.

A fuzzy set **A** is written as a set of pairs **{x, A(x)}** as

A = {{x , A(x)}} , x in the set X

where **x** is an element of the universal space **X**, and

A(x) is the value of the function **A** for this element.

The value **A(x)** is the **membership grade** of the element **x** in a fuzzy set **A**.

Example : Set **SMALL** in set **X** consisting of natural numbers ≤ **to 12**.

Assume: **SMALL(1) = 1, SMALL(2) = 1, SMALL(3) = 0.9, SMALL(4) = 0.6, SMALL(5) = 0.4, SMALL(6) = 0.3, SMALL(7) = 0.2, SMALL(8) = 0.1, SMALL(u) = 0 for u >= 9.**

Then, following the notations described in the definition above :

Set SMALL = {{1, 1 }, {2, 1 }, {3, 0.9}, {4, 0.6}, {5, 0.4}, {6, 0.3}, {7, 0.2}, {8, 0.1}, {9, 0 }, {10, 0 }, {11, 0}, {12, 0}}

Note that a fuzzy set can be defined precisely by associating with each **x** , its grade of membership in **SMALL**.

- **Definition of Universal Space**

Originally the universal space for fuzzy sets in fuzzy logic was defined only on the integers. Now, the universal space for fuzzy sets and fuzzy relations is defined with three numbers. The first two numbers specify the start and end of the universal space, and the third argument specifies the increment between elements. This gives the user more flexibility in choosing the universal space.

Example : The fuzzy set of numbers, defined in the universal space

X = { x_i } = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12} is presented as

SetOption [FuzzySet, UniversalSpace → {1, 12, 1}]

- **Graphic Interpretation of Fuzzy Sets SMALL**

The fuzzy set SMALL of small numbers, defined in the universal space

X = { x_i } = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12} is presented as

SetOption [FuzzySet, UniversalSpace → {1, 12, 1}]

The Set **SMALL** in set **X** is :

SMALL = FuzzySet {{1, 1 }, {2, 1 }, {3, 0.9}, {4, 0.6}, {5, 0.4}, {6, 0.3}, {7, 0.2}, {8, 0.1}, {9, 0 }, {10, 0 }, {11, 0}, {12, 0}}

Therefore **SetSmall** is represented as

SetSmall = **FuzzySet [{{1,1},{2,1}, {3,0.9}, {4,0.6}, {5,0.4},{6,0.3}, {7,0.2}, {8, 0.1}, {9, 0}, {10, 0}, {11, 0}, {12, 0}}** , **UniversalSpace → {1, 12, 1}]**

FuzzyPlot [SMALL, AxesLable → {"X", "SMALL"}]

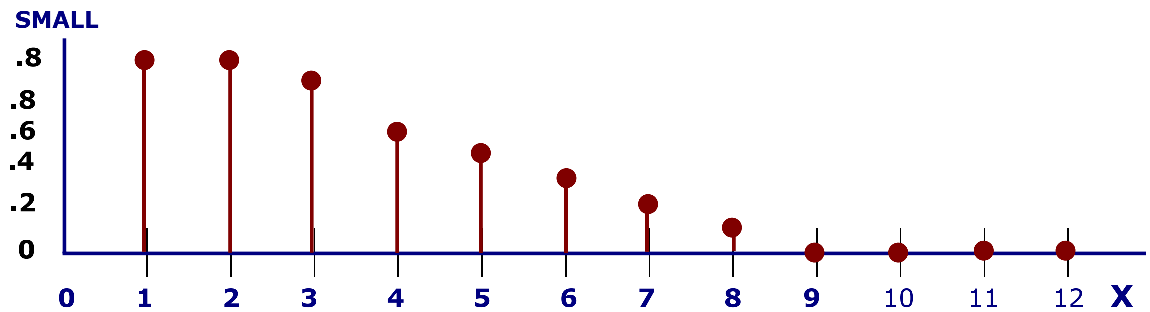


Fig Graphic Interpretation of Fuzzy Sets SMALL

- **Graphic Interpretation of Fuzzy Sets PRIME Numbers**

The fuzzy set PRIME numbers, defined in the universal space

X = { x_i } = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12} is presented as

SetOption [FuzzySet, UniversalSpace → {1, 12, 1}]

The Set **PRIME** in set **X** is :

PRIME = FuzzySet {{1, 0}, {2, 1}, {3, 1}, {4, 0}, {5, 1}, {6, 0}, {7, 1}, {8, 0}, {9, 0}, {10, 0}, {11, 1}, {12, 0}}

Therefore **SetPrime** is represented as

SetPrime = FuzzySet [{{1,0},{2,1}, {3,1}, {4,0}, {5,1},{6,0}, {7,1}, {8, 0}, {9, 0}, {10, 0}, {11, 1}, {12, 0}} , UniversalSpace → {1, 12, 1}]

FuzzyPlot [PRIME, AxesLable → {"X", "PRIME"}]

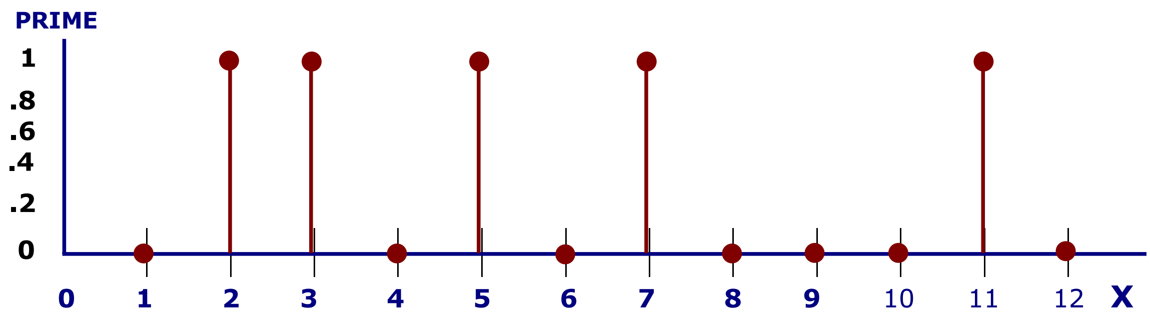


Fig Graphic Interpretation of Fuzzy Sets PRIME

- **Graphic Interpretation of Fuzzy Sets UNIVERSALSPACE**

In any application of sets or fuzzy sets theory, all sets are subsets of a fixed set called universal space or universe of discourse denoted by **X**. Universal space **X** as a fuzzy set is a function equal to **1** for all elements.

The fuzzy set **UNIVERSALSPACE** numbers, defined in the universal space **X = { x_i } = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}** is presented as

SetOption [FuzzySet, UniversalSpace → {1, 12, 1}]

The Set **UNIVERSALSPACE** in set **X** is :

UNIVERSALSPACE = FuzzySet {{1, 1}, {2, 1}, {3, 1}, {4, 1}, {5, 1}, {6, 1}, {7, 1}, {8, 1}, {9, 1}, {10, 1}, {11, 1}, {12, 1}}

Therefore **SetUniversal** is represented as

SetUniversal = FuzzySet [{{1,1},{2,1}, {3,1}, {4,1}, {5,1},{6,1}, {7,1}, {8, 1}, {9, 1}, {10, 1}, {11, 1}, {12, 1}} , UniversalSpace → {1, 12, 1}]

FuzzyPlot [UNIVERSALSPACE, AxesLable → {"X", " UNIVERSAL SPACE "}]

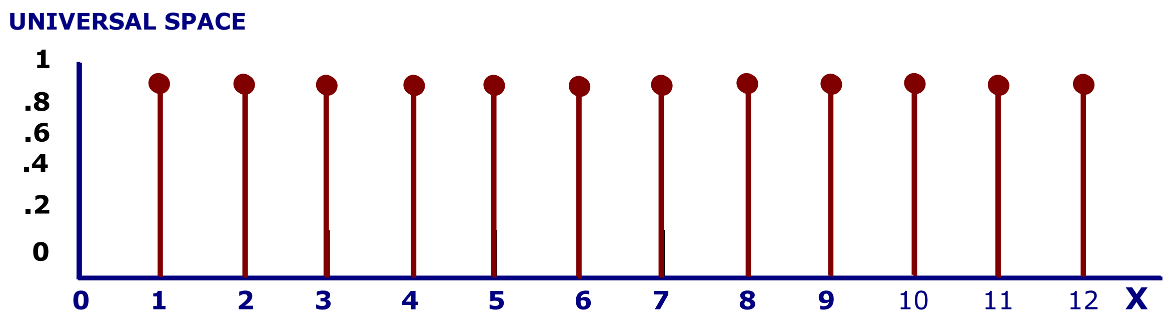


Fig Graphic Interpretation of Fuzzy Set UNIVERSALSPACE

- **Finite and Infinite Universal Space**

Universal sets can be finite or infinite.

Any universal set is finite if it consists of a specific number of different elements, that is, if in counting the different elements of the set, the counting can come to an end, else the set is infinite.

Examples:

1. Let **N** be the universal space of the days of the week.

   N = {Mo, Tu, We, Th, Fr, Sa, Su}. **N** is finite.

2. Let **M = {1, 3, 5, 7, 9, ...}.** **M** is infinite.

3. Let **L = {u | u is a lake in a city }.** **L** is finite.

   (Although it may be difficult to count the number of lakes in a city, but **L** is still a finite universal set.)

- **Graphic Interpretation of Fuzzy Sets EMPTY**

An empty set is a set that contains only elements with a grade of membership equal to **0**.

Example: Let EMPTY be a set of people, in Minnesota, older than 120.

The Empty set is also called the Null set.

The fuzzy set **EMPTY**, defined in the universal space

X = { x_i } = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12} is presented as

SetOption [FuzzySet, UniversalSpace → {1, 12, 1}]

The Set **EMPTY** in set **X** is :

EMPTY = FuzzySet {{1, 0}, {2, 0}, {3, 0}, {4, 0}, {5, 0}, {6, 0}, {7, 0}, {8, 0}, {9, 0}, {10, 0}, {11, 0}, {12, 0}}

Therefore **SetEmpty** is represented as

SetEmpty = **FuzzySet [{{1,0},{2,0}, {3,0}, {4,0}, {5,0},{6,0}, {7,0}, {8, 0}, {9, 0}, {10, 0}, {11, 0}, {12, 0}} , UniversalSpace → {1, 12, 1}]**

FuzzyPlot [EMPTY, AxesLable → {"X", " UNIVERSAL SPACE "}]

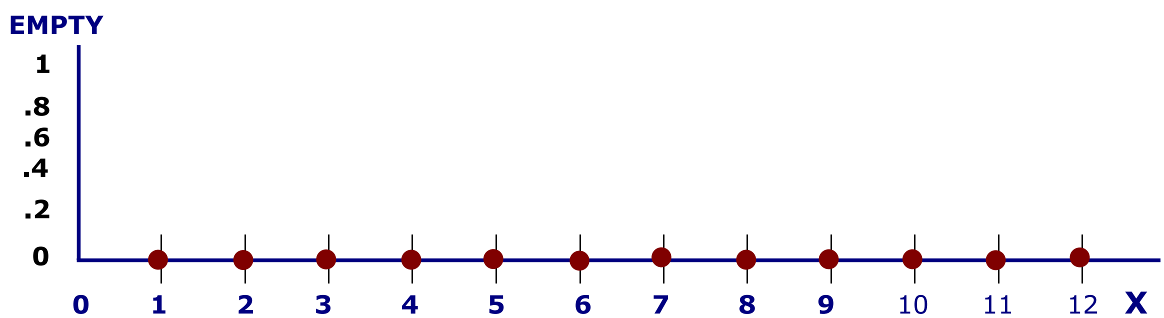


Fig Graphic Interpretation of Fuzzy Set EMPTY

2.2 Fuzzy Operations

A fuzzy set operations are the operations on fuzzy sets. The fuzzy set operations are generalization of crisp set operations. Zadeh [1965] formulated the fuzzy set theory in the terms of standard operations: Complement, Union, Intersection, and Difference.

In this section, the graphical interpretation of the following standard fuzzy set terms and the Fuzzy Logic operations are illustrated:

Inclusion :	**FuzzyInclude [VERYSMALL, SMALL]**
Equality :	**FuzzyEQUALITY [SMALL, STILLSMALL]**
Complement :	**FuzzyNOTSMALL = FuzzyCompliment [Small]**
Union :	**FuzzyUNION = [SMALL ∪ MEDIUM]**
Intersection :	**FUZZYINTERSECTON = [SMALL ∩ MEDIUM]**

- **Inclusion**

Let **A** and **B** be fuzzy sets defined in the same universal space **X**.

The fuzzy set **A** is included in the fuzzy set **B** if and only if for every **x** in the set **X** we have **A(x)** ≤ **B(x)**

Example :

The fuzzy set **UNIVERSALSPACE** numbers, defined in the universal space **X = { x$_i$ } = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}** is presented as

SetOption [FuzzySet, UniversalSpace → {1, 12, 1}]

The fuzzy set B SMALL

The Set **SMALL** in set **X** is :

SMALL = FuzzySet {{1, 1 }, {2, 1 }, {3, 0.9}, {4, 0.6}, {5, 0.4}, {6, 0.3}, {7, 0.2}, {8, 0.1}, {9, 0 }, {10, 0 }, {11, 0}, {12, 0}}

Therefore **SetSmall** is represented as

SetSmall = FuzzySet [{{1,1},{2,1}, {3,0.9}, {4,0.6}, {5,0.4},{6,0.3}, {7,0.2}, {8, 0.1}, {9, 0}, {10, 0}, {11, 0}, {12, 0}} , UniversalSpace → {1, 12, 1}]

The fuzzy set A VERYSMALL

The Set **VERYSMALL** in set **X** is :

VERYSMALL = FuzzySet {{1, 1 }, {2, 0.8 }, {3, 0.7}, {4, 0.4}, {5, 0.2}, {6, 0.1}, {7, 0 }, {8, 0 }, {9, 0 }, {10, 0 }, {11, 0}, {12, 0}}

Therefore **SetVerySmall** is represented as

SetVerySmall = FuzzySet [{{1,1},{2,0.8}, {3,0.7}, {4,0.4}, {5,0.2},{6,0.1}, {7,0}, {8, 0}, {9, 0}, {10, 0}, {11, 0}, {12, 0}} , UniversalSpace → {1, 12, 1}]

The Fuzzy Operation : Inclusion

Include [VERYSMALL, SMALL]

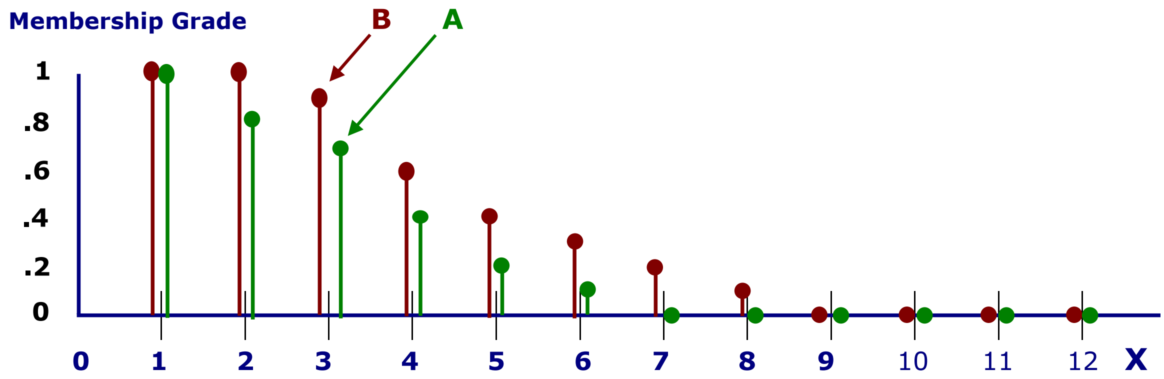


Fig Graphic Interpretation of Fuzzy Inclusion
FuzzyPlot [SMALL, VERYSMALL]

- **Comparability**

Two fuzzy sets **A** and **B** are comparable

if the condition **A ⊂ B or B ⊂ A** holds, ie,

if one of the fuzzy sets is a subset of the other set, they are comparable.

Two fuzzy sets **A** and **B** are incomparable

if the condition **A ⊄ B or B ⊄ A** holds.

Example 1:

Let **A = {{a, 1}, {b, 1}, {c, 0}}** and

B = {{a, 1}, {b, 1}, {c, 1}}.

Then **A** is comparable to **B**, since **A** is a subset of **B**.

Example 2 :

Let **C = {{a, 1}, {b, 1}, {c, 0.5}} and**

D = {{a, 1}, {b, 0.9}, {c, 0.6}}.

Then **C** and **D** are not comparable since

C is not a subset of **D** and

D is not a subset of **C**.

Property Related to Inclusion :

for all **x** in the set **X**, if **A(x) ⊂ B(x) ⊂ C(x), then accordingly A ⊂ C.**

- **Equality**

Let **A** and **B** be fuzzy sets defined in the same space **X**.

Then **A** and **B** are equal, which is denoted **X = Y**

if and only if for all **x** in the set **X**, **A(x) = B(x).**

Example.

The fuzzy set B SMALL

SMALL = FuzzySet {{1, 1 }, {2, 1 }, {3, 0.9}, {4, 0.6}, {5, 0.4}, {6, 0.3}, {7, 0.2}, {8, 0.1}, {9, 0 }, {10, 0 }, {11, 0}, {12, 0}}

The fuzzy set A STILLSMALL

STILLSMALL = FuzzySet {{1, 1 }, {2, 1 }, {3, 0.9}, {4, 0.6}, {5, 0.4}, {6, 0.3}, {7, 0.2}, {8, 0.1}, {9, 0 }, {10, 0 }, {11, 0}, {12, 0}}

The Fuzzy Operation : Equality

Equality [SMALL, STILLSMALL]

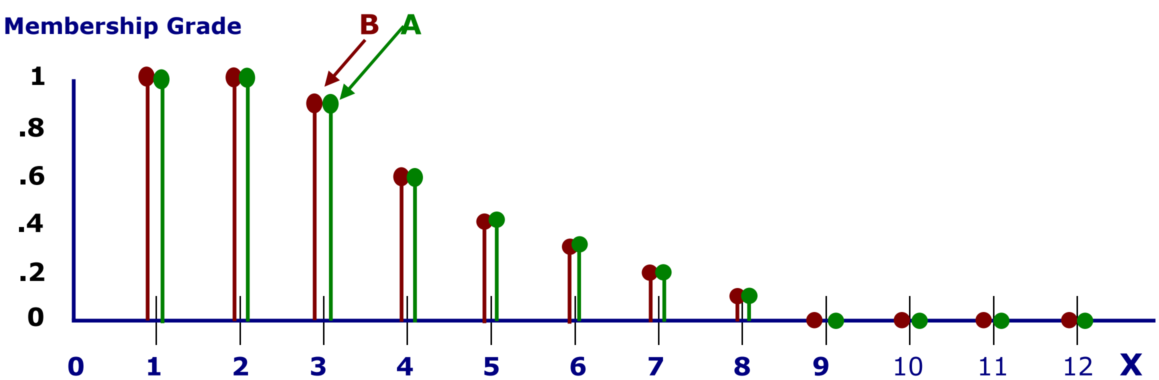


Fig Graphic Interpretation of Fuzzy Equality
FuzzyPlot [SMALL, STILLSMALL]

Note : If equality **A(x) = B(x)** is not satisfied even for one element **x** in the set **X**, then we say that **A** is not equal to **B**.

- **Complement**

Let **A** be a fuzzy set defined in the space **X**.

Then the fuzzy set **B** is a complement of the fuzzy set **A**, if and only if, for all **x** in the set **X**, **B(x) = 1 - A(x).**

The complement of the fuzzy set **A** is often denoted by **A'** or **Ac** or $\bar{A}$

Fuzzy Complement : **Ac(x) = 1 – A(x)**

Example 1.

The fuzzy set A SMALL

SMALL = FuzzySet {{1, 1 }, {2, 1 }, {3, 0.9}, {4, 0.6}, {5, 0.4}, {6, 0.3}, {7, 0.2}, {8, 0.1}, {9, 0 }, {10, 0 }, {11, 0}, {12, 0}}

The fuzzy set Ac NOTSMALL

NOTSMALL = FuzzySet {{1, 0 }, {2, 0 }, {3, 0.1}, {4, 0.4}, {5, 0.6}, {6, 0.7}, {7, 0.8}, {8, 0.9}, {9, 1 }, {10, 1 }, {11, 1}, {12, 1}}

The Fuzzy Operation : Compliment

NOTSMALL = Compliment [SMALL]

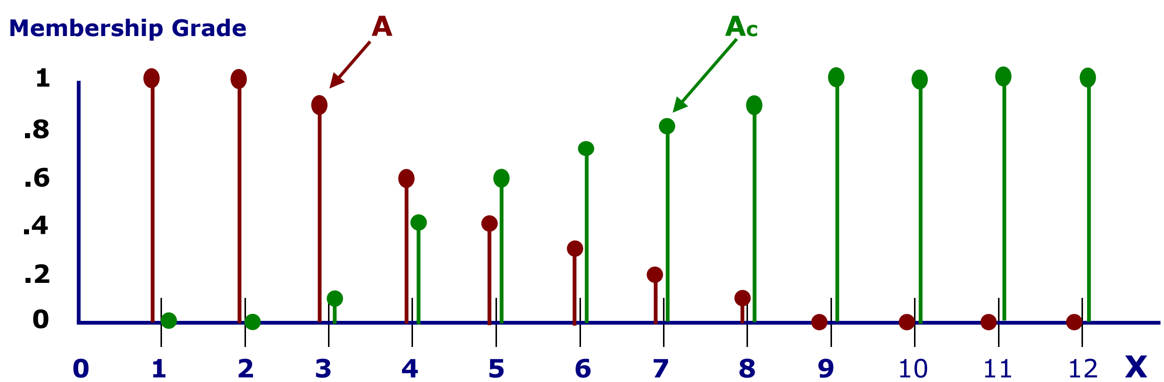


Fig Graphic Interpretation of Fuzzy Compliment
FuzzyPlot [SMALL, NOTSMALL]

Example 2.

The empty set Φ and the universal set **X**, as fuzzy sets, are complements of one another.

Φ' = X , X' = Φ

The fuzzy set B EMPTY

Empty = FuzzySet {{1, 0 }, {2, 0 }, {3, 0}, {4, 0}, {5, 0}, {6, 0}, {7, 0}, {8, 0}, {9, 0 }, {10, 0 }, {11, 0}, {12, 0}}

The fuzzy set A UNIVERSAL

Universal = FuzzySet {{1, 1 }, {2, 1 }, {3, 1}, {4, 1}, {5, 1}, {6, 1}, {7, 1}, {8, 1}, {9, 1 }, {10, 1 }, {11, 1}, {12, 1}}

The fuzzy operation : Compliment

EMPTY = Compliment [UNIVERSALSPACE]

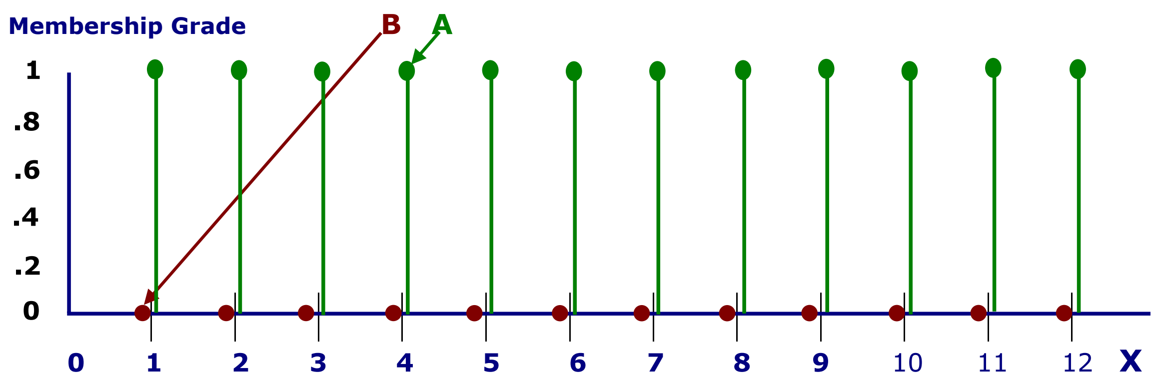


Fig Graphic Interpretation of Fuzzy Compliment
FuzzyPlot [EMPTY, UNIVERSALSPACE]

- **Union**

Let **A** and **B** be fuzzy sets defined in the space **X**.

The union is defined as the smallest fuzzy set that contains both **A** and **B**.

The union of **A** and **B** is denoted by **A ∪ B.**

The following relation must be satisfied for the union operation :

for all x in the set X, (A ∪ B)(x) = Max (A(x), B(x)).

Fuzzy Union : **(A ∪ B)(x) = max [A(x), B(x)] for all x ∈ X**

Example 1 : Union of Fuzzy **A** and **B**

A(x) = 0.6 and B(x) = 0.4 ∴ (A ∪ B)(x) = max [0.6, 0.4] = 0.6

Example 2 : Union of **SMALL** and **MEDIUM**

The fuzzy set A SMALL

SMALL = FuzzySet {{1, 1 }, {2, 1 }, {3, 0.9}, {4, 0.6}, {5, 0.4}, {6, 0.3}, {7, 0.2}, {8, 0.1}, {9, 0 }, {10, 0 }, {11, 0}, {12, 0}}

The fuzzy set B MEDIUM

MEDIUM = FuzzySet {{1, 0 }, {2, 0 }, {3, 0}, {4, 0.2}, {5, 0.5}, {6, 0.8}, {7, 1}, {8, 1}, {9, 0.7 }, {10, 0.4 }, {11, 0.1}, {12, 0}}

The fuzzy operation : Union

FUZZYUNION = [SMALL ∪ MEDIUM]

SetSmallUNIONMedium = **FuzzySet [{{1,1},{2,1}, {3,0.9}, {4,0.6}, {5,0.5}, {6,0.8}, {7,1}, {8, 1}, {9, 0.7}, {10, 0.4}, {11, 0.1}, {12, 0}} , UniversalSpace → {1, 12, 1}]**

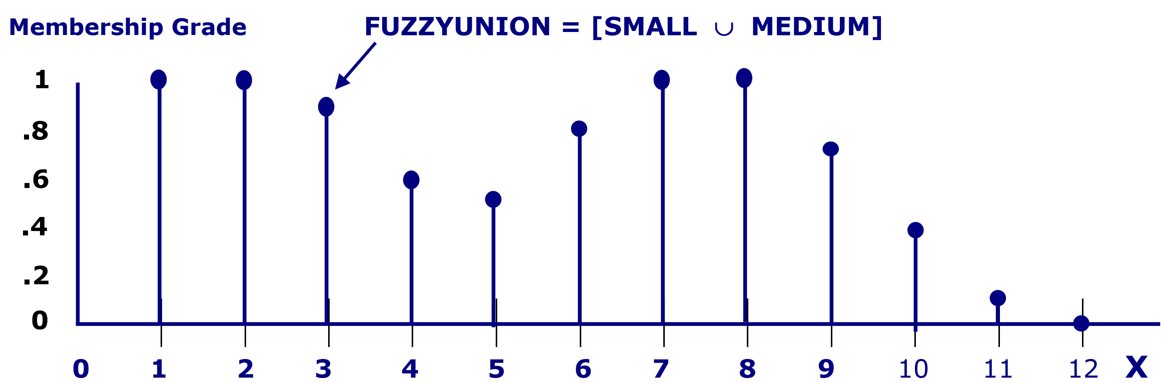


Fig Graphic Interpretation of Fuzzy Union
FuzzyPlot [UNION]

The notion of the union is closely related to that of the connective "or".

Let **A** is a class of "Young" men, **B** is a class of "Bald" men.

If "David is Young" or "David is Bald," then David is associated with the union of **A** and **B.** Implies David is a member of **A ∪ B**.

- **Properties Related to Union**

The properties related to union are :

Identity, Idempotence, Commutativity and Associativity.

- **Identity:**

  A ∪ Φ = A

  input = Equality [SMALL ∪ EMPTY , SMALL]

  output = True

  A ∪ X = X

  input = Equality [SMALL ∪ UnivrsalSpace , UnivrsalSpace]

  output = True

- **Idempotence :**

  A ∪ A = A

  input = Equality [SMALL ∪ SMALL , SMALL]

  output = True

- **Commutativity :**

  A ∪ B = B ∪ A

  input = Equality [SMALL ∪ MEDIUM, MEDIUM ∪ SMALL]

  output = True

- **Associativity:**

  A ∪ (B∪ C) = (A∪ B) ∪ C

  input = Equality [SMALL ∪ (MEDIUM ∪ BIG) , (SMALL ∪ MEDIUM**)** ∪ BIG]

  output = True

SMALL = FuzzySet {{1, 1 }, {2, 1 }, {3, 0.9}, {4, 0.6}, {5, 0.4}, {6, 0.3}, {7, 0.2}, {8, 0.1}, {9, 0.7 }, {10, 0.4 }, {11, 0}, {12, 0}}

MEDIUM = FuzzySet {{1, 0 }, {2, 0 }, {3, 0}, {4, 0.2}, {5, 0.5}, {6, 0.8}, {7, 1}, {8, 1}, {9, 0 }, {10, 0 }, {11, 0.1}, {12, 0}}

BIG = FuzzySet [{{1,0}, {2,0}, {3,0}, {4,0}, {5,0}, {6,0.1}, {7,0.2}, {8,0.4}, {9,0.6}, {10,0.8}, {11,1}, {12,1}}]

Medium ∪ BIG = FuzzySet [{1,0},{2,0}, {3,0}, {4,0.2}, {5,0.5}, {6,0.8}, {7,1}, {8, 1}, {9, 0.6}, {10, 0.8}, {11, 1}, {12, 1}]

Small ∪ Medium = FuzzySet [{1,1},{2,1}, {3,0.9}, {4,0.6}, {5,0.5}, {6,0.8}, {7,1}, {8, 1}, {9, 0.7}, {10, 0.4}, {11, 0.1}, {12, 0}]

- **Intersection**

Let **A** and **B** be fuzzy sets defined in the space **X.**

The intersection is defined as the greatest fuzzy set included both **A** and **B.**

The intersection of **A** and **B** is denoted by **A ∩ B.**

The following relation must be satisfied for the union operation :

for all x in the set X, (A ∩ B)(x) = Min (A(x), B(x)).

Fuzzy Intersection : (A ∩ B)(x) = min [A(x), B(x)] for all x ∈ X

Example 1 : Intersection of Fuzzy **A** and **B**

A(x) = 0.6 and B(x) = 0.4 ∴ (A ∩ B)(x) = min [0.6, 0.4] = 0.4

Example 2 : Union of **SMALL** and **MEDIUM**

The fuzzy set A SMALL

SMALL = FuzzySet {{1, 1 }, {2, 1 }, {3, 0.9}, {4, 0.6}, {5, 0.4}, {6, 0.3}, {7, 0.2}, {8, 0.1}, {9, 0 }, {10, 0 }, {11, 0}, {12, 0}}

The fuzzy set B MEDIUM

MEDIUM = FuzzySet {{1, 0 }, {2, 0 }, {3, 0}, {4, 0.2}, {5, 0.5}, {6, 0.8}, {7, 1}, {8, 1}, {9, 0.7 }, {10, 0.4 }, {11, 0.1}, {12, 0}}

The fuzzy operation : Intersection

FUZZYINTERSECTION = min [SMALL ∩ MEDIUM]

SetSmallINTERSECTIONMedium = FuzzySet [{{1,0},{2,0}, {3,0}, {4,0.2}, {5,0.4}, {6,0.3}, {7,0.2}, {8, 0.1}, {9, 0}, {10, 0}, {11, 0}, {12, 0}} , UniversalSpace → {1, 12, 1}]

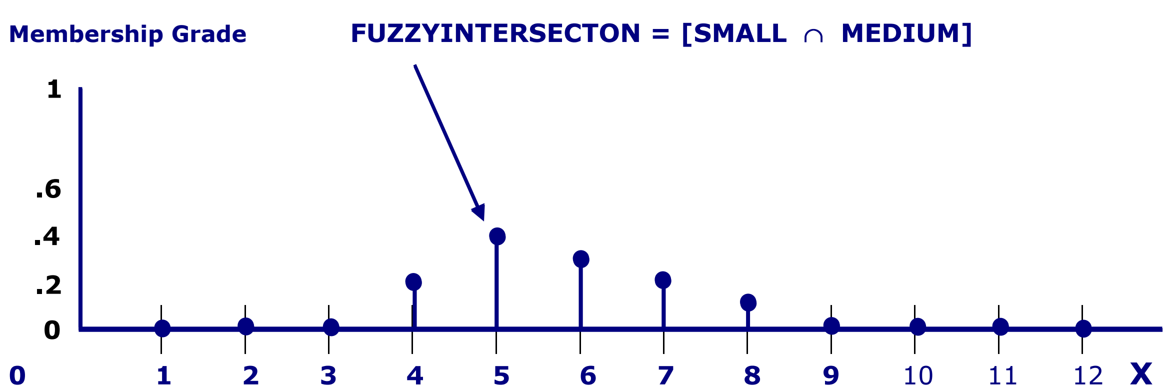


Fig Graphic Interpretation of Fuzzy Intersection
FuzzyPlot [INTERSECTION]

3. Neural Computing

Neural Computers mimic certain processing capabilities of the human brain.

- Neural Computing is an information processing paradigm, inspired by biological system, composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems.

- A neural net is an artificial representation of the human brain that tries to simulate its learning process. The term "artificial" means that neural nets are implemented in computer programs that are able to handle the large number of necessary calculations during the learning process.

- Artificial Neural Networks (ANNs), like people, learn by example.

- An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process.

- Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true for ANNs as well.

3.1 Biological Model:

The human brain consists of a large number (more than a billion) of neural cells that process information. Each cell works like a simple processor. The massive interaction between all cells and their parallel processing, makes the brain's abilities possible. The structure of neuron is shown below.

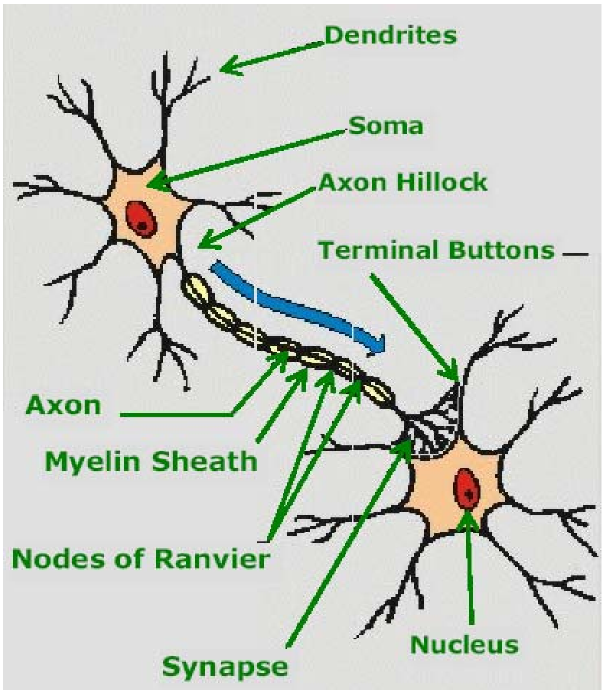


Fig. Structure of Neuron

Dendrites are the branching fibers extending from the cell body or soma.

Soma or cell body of a neuron contains the nucleus and other structures, support chemical processing and production of neurotransmitters.

Axon is a singular fiber carries information away from the soma to the synaptic sites of other neurons (dendrites and somas), muscles, or glands.

Axon hillock is the site of summation for incoming information. At any moment, the collective influence of all neurons, that conduct as impulses to a given neuron, will determine whether or not an action potential will be initiated at the axon hillock and propagated along the axon.

Myelin Sheath consists of fat-containing cells that insulate the axon from electrical activity. This insulation acts to increase the rate of transmission of signals. A gap exists between each myelin sheath cell along the axon. Since fat inhibits the propagation of electricity, the signals jump from one gap to the next.

Nodes of Ranvier are the gaps (about 1 μm) between myelin sheath cells long axons. Since fat serves as a good insulator, the myelin sheaths speed the rate of transmission of an electrical impulse along the axon.

Synapse is the point of connection between two neurons or a neuron and a muscle or a gland. Electrochemical communication between neurons takes place at these junctions.

Terminal Buttons of a neuron are the small knobs at the end of an axon that release chemicals called neurotransmitters.

- **Information flow in a Neural Cell**

The input /output and the propagation of information are shown below.

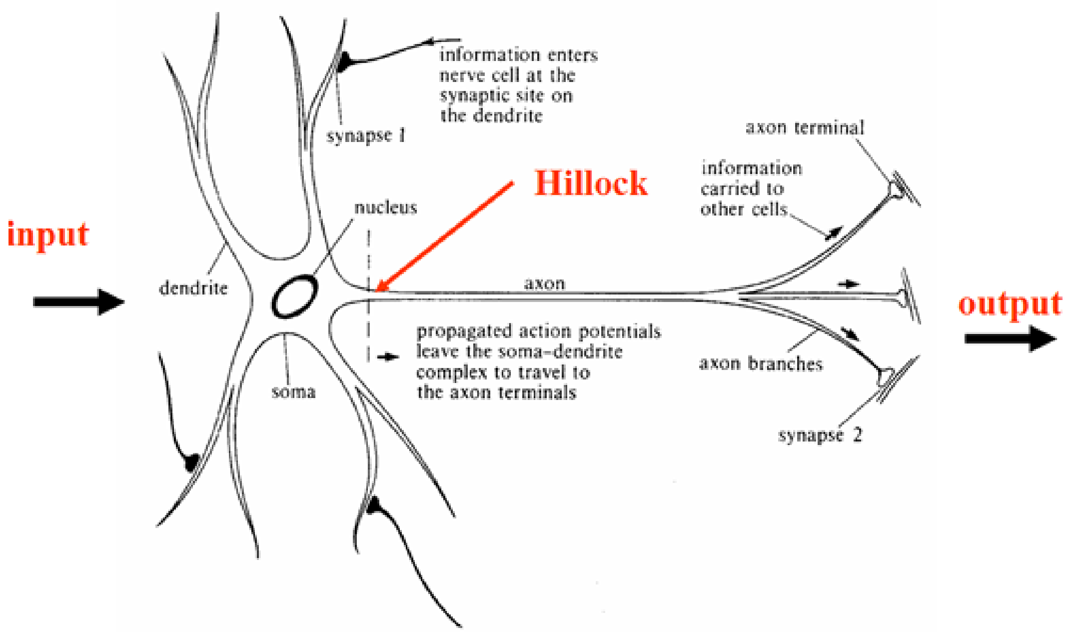


Fig. Structure of a neural cell in the human brain

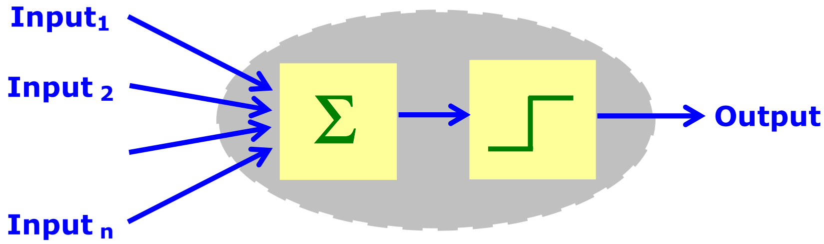
- Dendrites receive activation from other neurons.
- Soma processes the incoming activations and converts them into output activations.
- Axons act as transmission lines to send activation to other neurons.
- Synapses the junctions allow signal transmission between the axons and dendrites.
- The process of transmission is by diffusion of chemicals called neuro-transmitters.

McCulloch-Pitts introduced a simplified model of this real neurons.

3.2 Artificial Neuron

- **The McCulloch-Pitts Neuron**

This is a simplified model of real neurons, known as a Threshold Logic Unit.



- A set of synapses (ie connections) brings in activations from other neurons.
- A processing unit sums the inputs, and then applies a non-linear activation function (i.e. transfer / threshold function).
- An output line transmits the result to other neurons.

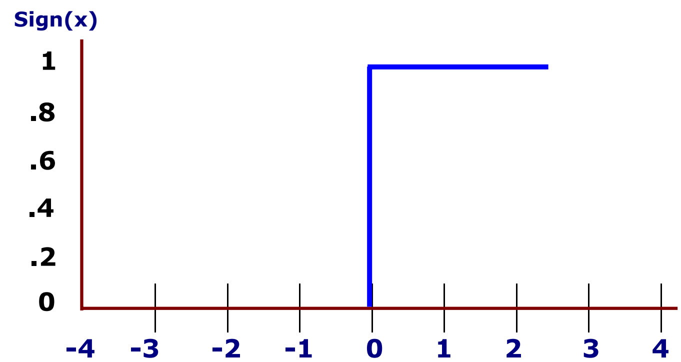
In other words, the input to a neuron arrives in the form of signals. The signals build up in the cell. Finally the cell fires (discharges) through the output. The cell can start building up signals again.

- **Functions :**

  The function **y = f(x)** describes a relationship, an input-output mapping, from **x** to **y**.

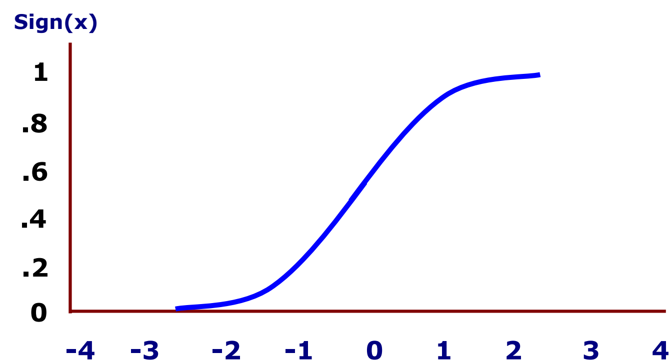
  - **Threshold or Sign function sgn(x)** : defined as

$$\text{sgn}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$



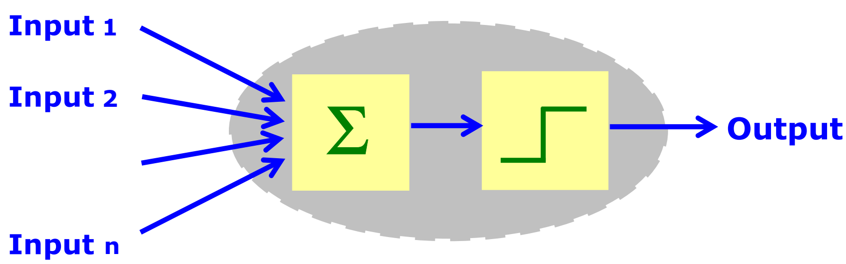
  - **Threshold or Sign function sigmoid (x)** : defined as a smoothed (differentiable) form of the threshold function

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$



- **McCulloch-Pitts (M-P) Neuron Equation**

Fig below is the same previously shown simplified model of a real neuron, as a threshold Logic Unit.



The equation for the output of a McCulloch-Pitts neuron as a function of **1** to **n** inputs is written as :

$$\textbf{Output} = \text{sgn}\left(\sum_{i=1}^{n} \text{Input}_i - \Phi\right)$$

where Φ is the neuron's activation threshold.

$$\textbf{If} \sum_{i=1}^{n} \text{Input}_i \geq \Phi \quad \textbf{then Output} = 1$$

$$\textbf{If} \sum_{i=1}^{n} \text{Input}_i < \Phi \quad \textbf{then Output} = 0$$

Note : The McCulloch-Pitts neuron is an extremely simplified model of real biological neurons. Some of its missing features include: non-binary input and output, non-linear summation, smooth thresholding, stochastic (non-deterministic), and temporal information processing.

- **Basic Elements of an Artificial Neuron**

It consists of three basic components - weights, thresholds, and a single activation function.

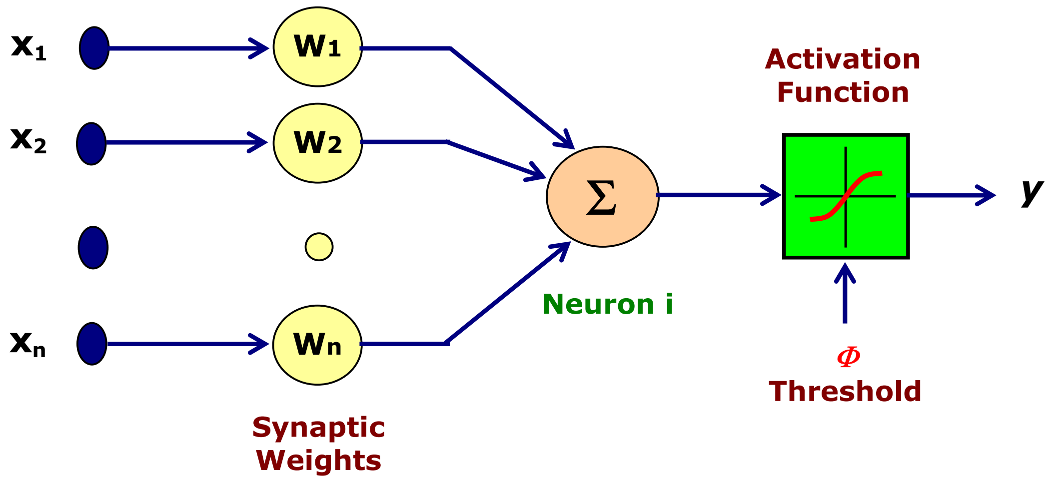


Fig Basic Elements of an Artificial Neuron

Weighting Factors

The values W_1 , W_2 , . . . W_n are weights to determine the strength of input row vector $X = [x_1 , x_2 , . . . , x_n]^T$. Each input is multiplied by the associated weight of the neuron connection $X^T W$. The **+ve** weight excites and the **-ve** weight inhibits the node output.

Threshold

The node's internal threshold Φ is the magnitude offset. It affects the activation of the node output **y** as:

$$y = \sum_{i=1}^{n} X_i W_i - \Phi_k$$

Activation Function

An activation function performs a mathematical operation on the signal output. The most common activation functions are, Linear Function, Threshold Function, Piecewise Linear Function, Sigmoidal (S shaped) function, Tangent hyperbolic function and are chose depending upon the type of problem to be solved by the network.

- **Example :**

A neural network consists four inputs with the weights as shown.

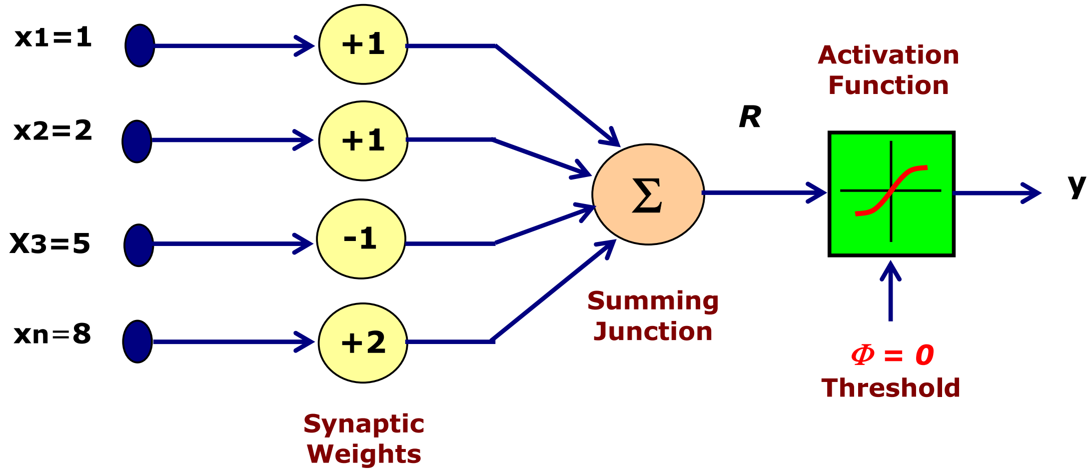


Fig Neuron Structure of Example

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

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

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

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

y (threshold) = 1

- **Networks of McCulloch-Pitts Neurons**

One neuron can not do much on its own. Usually we will have many neurons labeled by indices **k, i, j** and activation flows between them via synapses with strengths **w_{ki}, w_{ij}** :

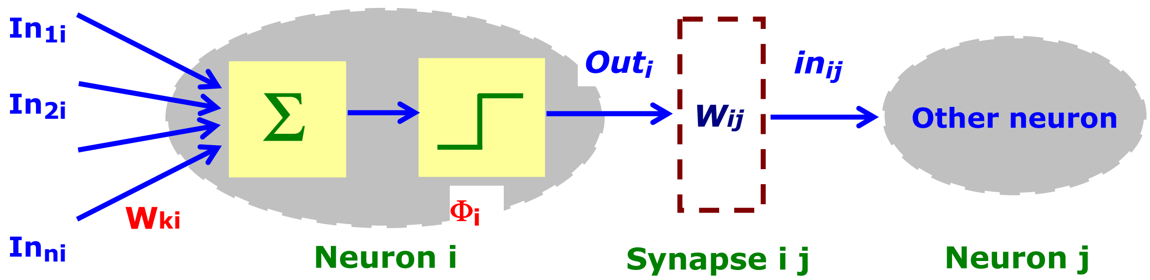


Fig McCulloch-Pitts Neuron

$in_{ki}\ Wki = Out_k$, $\quad Out_i = \text{sgn}\left(\sum_{k=1}^{n} In_{ki} - \Phi_i \right)$, $\quad In_{ij} = Out_i\ Wij$

- **Single and Multi - Layer Perceptrons**

A perceptron is a name for simulated neuron in the computer program. The usually way to represent a neuron model is described below. The neurons are shown as circles in the diagram. It has several inputs and a single output. The neurons have gone under various names.

- Each individual cell is called either a **node** or a **perceptron**.
- A neural network consisting of a layer of nodes or perceptrons between the input and the output is called a **single layer perceptron**.
- A network consisting of several layers of single layer perceptron stacked on top of other, between input and output , is called a **multi-layer perceptron**
-

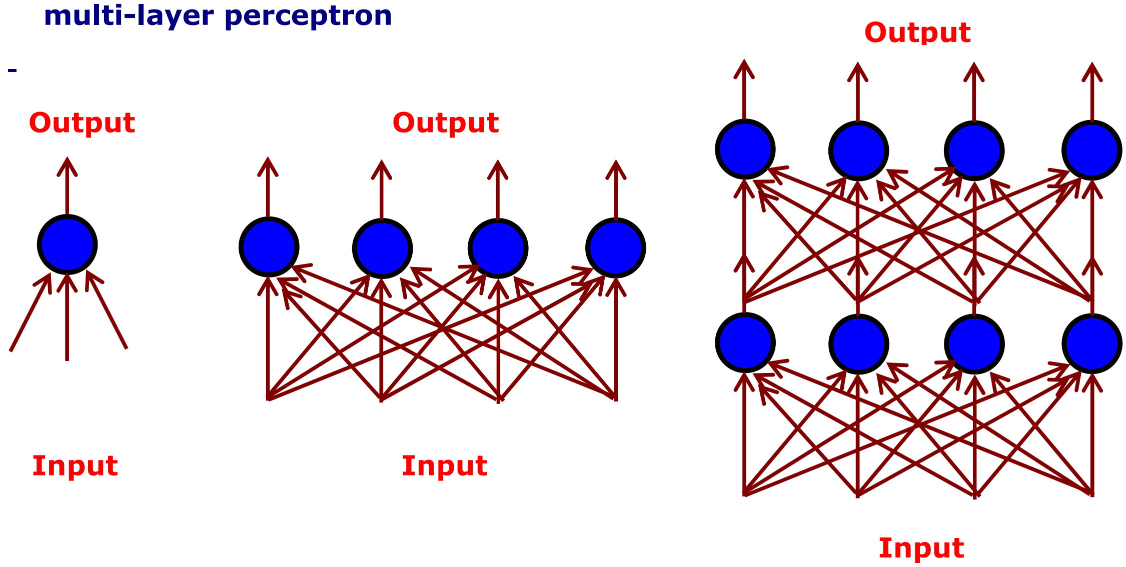


Fig Single and Multi - Layer Perceptrons

Multi-layer perceptrons are more powerful than single-layer perceptrons.

- **Perceptron**

Any number of McCulloch-Pitts neurons can be connected together in any way.

Definition : An arrangement of one input layer of McCulloch-Pitts neurons, that is feeding forward to one output layer of McCulloch-Pitts neurons is known as a Perceptron.

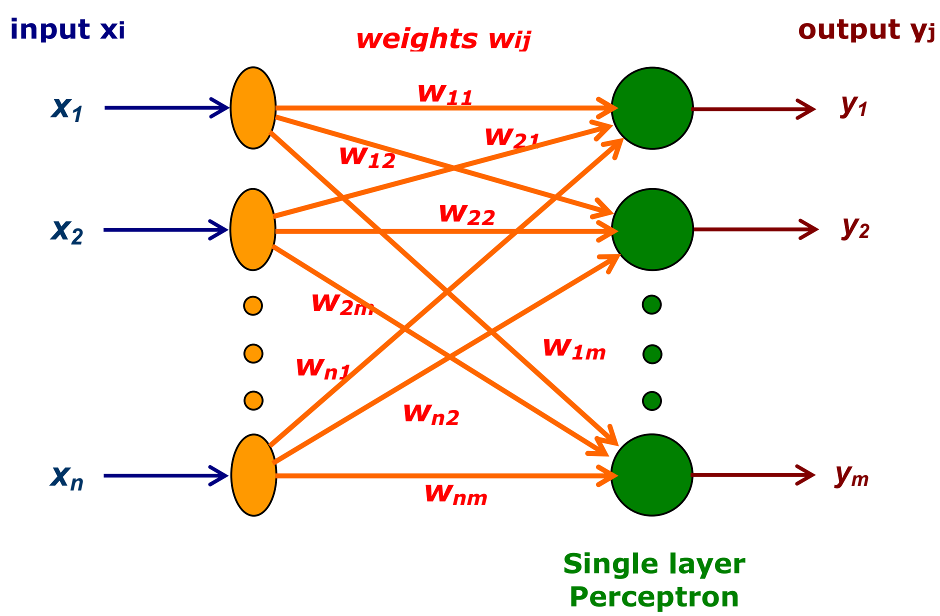


Fig. Simple Perceptron Model

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

A Perceptron is a powerful computational device.

4. Genetic Algorithms

Idea of evolutionary computing was introduced in the year 1960s by I. Rechenberg in his work "Evolution strategies". His idea was then developed by other researchers.

Genetic Algorithms (GAs) were invented by John Holland in early 1970's to mimic some of the processes observed in natural evolution.

Later in 1992 John Koza used GAs to evolve programs to perform certain tasks. He called his method "Genetic Programming" (GP).

GAs simulate natural evolution, a combination of selection, recombination and mutation to evolve a solution to a problem.

GAs simulate the survival of the fittest, among individuals over consecutive generation for solving a problem. Each generation consists of a population of character strings that are analogous to the chromosome in our DNA (Deoxyribonucleic acid). DNA contains the genetic instructions used in the development and functioning of all known living organisms.

- **What are Genetic Algorithms**

  - Genetic Algorithms (GAs) are adaptive heuristic search algorithm based on the evolutionary ideas of natural selection and genetics.

  - Genetic algorithms (GAs) are a part of evolutionary computing, a rapidly growing area of artificial intelligence. GAs are inspired by Darwin's theory about evolution - "survival of the fittest".

  - GAs represent an intelligent exploitation of a random search used to solve optimization problems.

  - GAs, although randomized, exploit historical information to direct the search into the region of better performance within the search space.

  - In nature, competition among individuals for scanty resources results in the fittest individuals dominating over the weaker ones.

- **Why Genetic Algorithms**

  "Genetic Algorithms are good at taking large, potentially huge search spaces and navigating them, looking for optimal combinations of things, solutions you might not otherwise find in a lifetime.” - Salvatore Mangano Computer Design, May 1995.

  - GA is better than conventional AI, in that it is more robust.
  - Unlike older AI systems, GAs do not break easily even if the inputs changed slightly, or in the presence of reasonable noise.
  - In searching a large state-space, multi-modal state-space, or n-dimensional surface, a GA may offer significant benefits over more typical search of optimization techniques, like - linear programming, heuristic, depth-first, breath-first.

- **Mechanics of Biological Evolution**

Genetic Algorithms are a way of solving problems by mimicking processes the nature uses - Selection, Crosses over, Mutation and Accepting to evolve a solution to a problem.

- Every **organism** has a set of **rules**, describing how that organism is built, and encoded in the **genes** of an organism.
- The genes are connected together into long strings called **chromosomes**.
- Each gene represents a specific **trait** (feature) of the organism and has several different settings, e.g. setting for a hair color gene may be black or brown.
- The genes and their settings are referred as an organism's **genotype.**
- When two organisms mate they share their genes. The resultant offspring may end up having half the genes from one parent and half from the other parent. This process is called **crossover** (recombination).
- The newly created offspring can then be mutated. A gene may be **mutated** and expressed in the organism as a completely new trait. Mutation means, that the elements of DNA are a bit changed. This change is mainly caused by errors in copying genes from parents.
- The **fitness** of an organism is measured by success of the organism in its life.

4.1 Artificial Evolution and Search Optimization

The problem of finding solutions to problems is itself a problem with no general solution. Solving problems usually mean looking for solutions, which will be the best among others.

- In engineering and mathematics finding the solution to a problem is often thought as a process of optimization.
- Here the process is : first formulate the problems as mathematical models expressed in terms of functions; then to find a solution, discover the parameters that optimize the model or the function components that provide optimal system performance.

The well-established search / optimization techniques are usually classified in to three broad categories : Enumerative, Calculus-based, and Guided random search techniques. A taxonomy of Evolution & Search Optimization classes is illustrated in the next slide.

- **Taxonomy of Evolution & Search Optimization Classes**

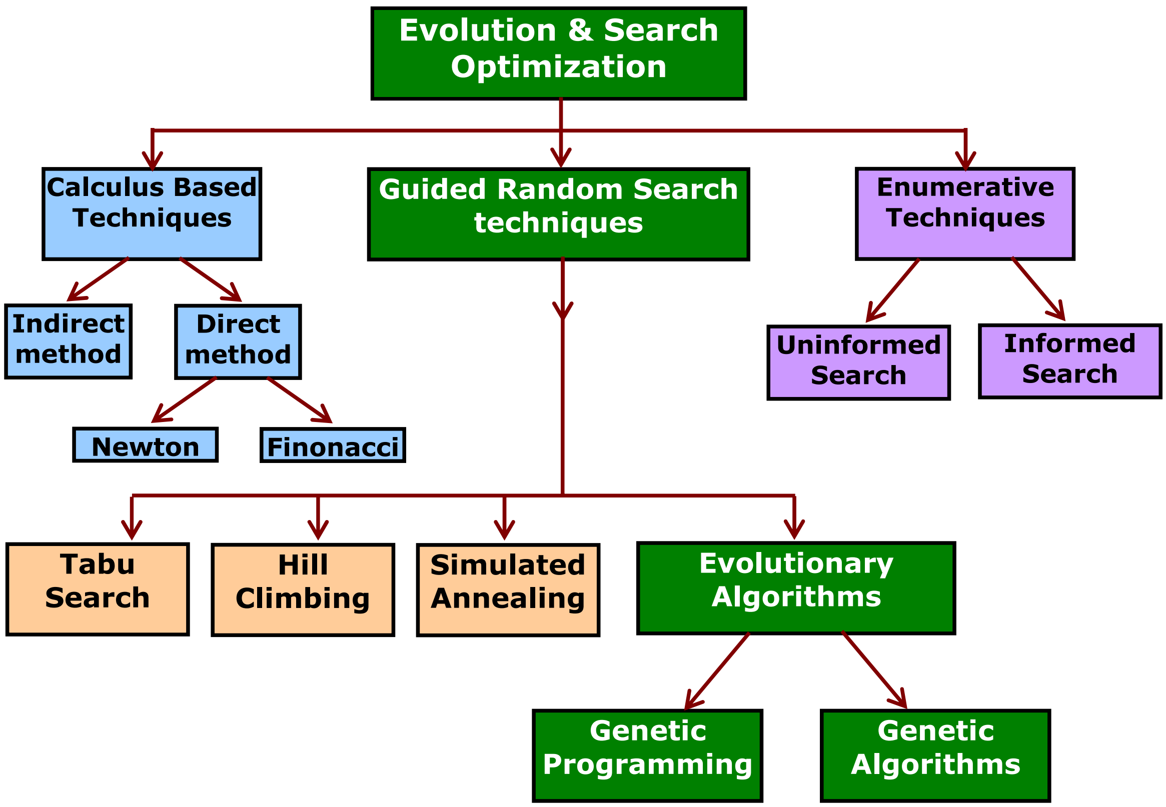


Fig Evolution & Search Optimization Techniques

Each of these techniques are briefly explained in the next three slides.

- **Enumerative Methods**

These are the traditional search and control strategies. They search for a solution in a problem space within the domain of artificial intelligence. There are many control structures for search. The depth-first search and breadth-first search are the two most common search strategies. Here the search goes through every point related to the function's domain space (finite or discretized), one point at a time. They are very simple to implement but usually require significant computation. These techniques are not suitable for applications with large domain spaces.

In the field of AI, enumerative methods are subdivide into two categories : uninformed and informed methods.

- **Uninformed or blind methods :** Such as mini-max algorithm searches all points in the space in a predefined order; this is used in game playing;

- **Informed methods :** Such as Alpha-Beta and A*, does more sophisticated search using domain specific knowledge in the form of a cost function or heuristic in order to reduce the cost of the search.

- **Calculus based techniques**

  Here a set of necessary and sufficient conditions to be satisfied by the solutions of an optimization problem. They subdivide into direct and indirect methods.

  - **Direct or Numerical methods**, such as Newton or Fibonacci, seek extremes by "hopping" around the search space and assessing the gradient of the new point, which guides the search. This is simply the notion of "hill climbing", which finds the best local point by climbing the steepest permissible gradient. These techniques can be used only on a restricted set of "well behaved" functions.

  - **Indirect methods** search for local extremes by solving the usually non-linear set of equations resulting from setting the gradient of the objective function to zero. The search for possible solutions (function peaks) starts by restricting itself to points with zero slope in all directions.

- **Guided Random Search techniques**

  These are based on enumerative techniques but they use additional information to guide the search. Two major subclasses are simulated annealing and evolutionary algorithms. Both are evolutionary processes.

  - **Simulated annealing** uses a thermodynamic evolution process to search minimum energy states.

  - **Evolutionary algorithms (EAs)** use natural selection principles. This form of search evolves throughout generations, improving the features of potential solutions by means of biological inspired operations. Genetic Algorithms (GAs) are a good example of this technique.

Our main concern is, how does an Evolutionary algorithm :

- implement and carry out search,
- describes the process of search,
- what are the elements required to carry out search, and
- what are the different search strategies.

4.2 Evolutionary Algorithms (EAs)

Evolutionary algorithms are search methods. They take inspirations from natural selection and survival of the fittest in the biological world, and therefore differ from traditional search optimization techniques. EAs involve search from a "population" of solutions, and not from a single point. Each iteration of an EA involves a competitive selection that weeds out poor solutions. The solutions with high "fitness" are "recombined" with other solutions by swapping parts of a solution with another. Solutions are also "mutated" by making a small change to a single element of the solution. Recombination and mutation are used to generate new solutions that are biased towards regions of the space for which good solutions have already been seen.

Evolutionary search algorithm (issues related to search) **:**

In the search space, each point represent one feasible solution. Each feasible solution is marked by its value or fitness for the problem. The issues related to search are :

- Search for a solution point, means finding which one point (or more) among many feasible solution points in the search space is the solution. This requires looking for some extremes, minimum or maximum.
- Search space can be whole known, but usually we know only a few points and we are generating other points as the process of finding solution continues.
- Search can be very complicated. One does not know where to look for the solution and where to start.
- What we find is some suitable solution, not necessarily the best solution. The solution found is often considered as a good solution, because it is not often possible to prove what is the real optimum solution.

5. Associative Memory

An associative memory is a content-addressable structure that maps a set of input patterns to a set of output patterns. The associative memory are of two types : auto-associative and hetero-associative.

- An **auto-associative memory** retrieves a previously stored pattern that most closely resembles the current pattern.
- In a **hetero-associative memory**, the retrieved pattern is, in general, different from the input pattern not only in content but possibly also in type and format.

- **Example : Associative Memory**

The figure below shows a memory containing names of several people.

If the given memory is content-addressable,

Then using the erroneous string "Crhistpher Columbos" as key is sufficient to retrieve the correct name "Christopher Colombus."

In this sense, this type of memory is robust and fault-tolerant, because this type of memory exhibits some form of error-correction capability.

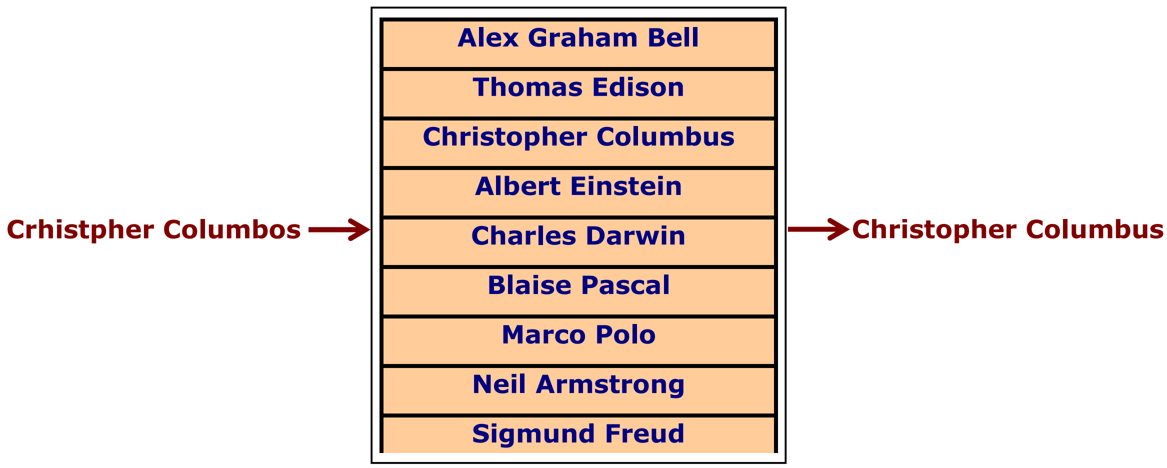


Fig. A content-addressable memory, Input and Output

- **Description of Associative Memory**

An associative memory is a content-addressable structure that maps specific input representations to specific output representations.

- A content-addressable memory is a type of memory that allows, the recall of data based on the degree of similarity between the input pattern and the patterns stored in memory.
- It refers to a memory organization in which the memory is accessed by its content and not or opposed to an explicit address in the traditional computer memory system.
- This type of memory allows the recall of information based on partial knowledge of its contents.

[Continued in next slide]

[Continued from previous slide]

- It is a system that "associates" two patterns **(X, Y)** such that when one is encountered, the other can be recalled.

  - Let **X** and **Y** be two vectors of length **m** and **n** respectively.
  - Typically, **XÎ** $\{-1, +1\}^m$**,** **Y Î** $\{-1, +1\}^n$
  - The components of the vectors can be thought of as pixels when the two patterns are considered as bitmap images.

- There are two classes of associative memory:
  - auto-associative and
  - hetero-associative.

  An **auto-associative** memory is used to retrieve a previously stored pattern that most closely resembles the current pattern.

  In a **hetero-associative** memory, the retrieved pattern is, in general, different from the input pattern not only in content but possibly also different in type and format.

- Artificial neural networks can be used as associative memories.
  The simplest artificial neural associative memory is the linear associater**.** The other popular ANN models used as associative memories are Hopfield model and Bidirectional Associative Memory (BAM) models.

6. Adaptive Resonance Theory (ART)

ART stands for "*Adaptive Resonance Theory*", invented by Stephen Grossberg in 1976. ART encompasses a wide variety of neural networks, based explicitly on neurophysiology. The word *"Resonance*" is a concept, just a matter of being within a certain threshold of a second similarity measure.

The basic ART system is an **unsupervised learning model***,* similar to many iterative clustering algorithm where each case is processed by finding the "nearest" cluster seed that resonate with the case and update the cluster seed to be "closer" to the case. If no seed resonate with the case then a new cluster is created.

Note : The terms *nearest* and *closer* are defined in many ways in clustering algorithm. In ART, these two terms are defined in slightly different way by introducing the concept of "resonance".

- **Definitions of ART and other types of Learning**

ART is a neural network topology whose dynamics are based on Adaptive Resonance Theory (ART). Grossberg developed ART as a theory of human cognitive information processing. The emphasis of ART neural networks lies at unsupervised learning and self-organization to mimic biological behavior. Self-organization means that the system must be able to build stable recognition categories in real-time.

The unsupervised learning means that the network learns the significant patterns on the basis of the inputs only. There is no feedback. There is no external teacher that instructs the network or tells to which category a certain input belongs. Learning in biological systems always starts as unsupervised learning; Example : For the newly born, hardly any pre-existing categories exist.

The other two types of learning are reinforcement learning and supervised learning. In reinforcement learning the net receives only limited feedback, like *"on this input you performed well"* or *"on this input you have made an error"*. In supervised mode of learning a net receives for each input the correct response.

Note: A system that can learn in unsupervised mode can always be adjusted to learn in the other modes, like reinforcement mode or supervised mode. But a system specifically designed to learn in supervised mode can never perform in unsupervised mode.

- **Description of Adaptive Resonance Theory**

The basic ART system is an **unsupervised learning model**.

The model typically consists of :

– a comparison field and a recognition field composed of neurons,

– a vigilance parameter, and

– a reset module.

The functions of each of these constituents are explained below.

- **Comparison field and Recognition field**
  - The Comparison field takes an input vector (a 1-D array of values) and transfers it to its best match in the Recognition field; the best match is, the single neuron whose set of weights (weight vector) matches most closely the input vector.
  - Each Recognition Field neuron outputs a negative signal (proportional to that neuron's quality of match to the input vector) to each of the other Recognition field neurons and inhibits their output accordingly.
  - Recognition field thus exhibits lateral inhibition, allowing each neuron in it to represent a category to which input vectors are classified.

- **Vigilance parameter**

  It has considerable influence on the system memories:
  - higher vigilance produces highly detailed memories,
  - lower vigilance results in more general memories

- **Reset module**

  After the input vector is classified, the Reset module compares the strength of the recognition match with the vigilance parameter.
  - If the vigilance threshold is met, Then training commences.
  - Else, the firing recognition neuron is inhibited until a new input vector is applied;

- **Training ART-based Neural Networks**

Training commences only upon completion of a search procedure. What happens in this search procedure :

- The Recognition neurons are disabled one by one by the reset function until the vigilance parameter is satisfied by a recognition match.
- If no committed recognition neuron's match meets the vigilance threshold, then an uncommitted neuron is committed and adjusted towards matching the input vector.

Methods of training ART-based Neural Networks:

There are two basic methods, the slow and fast learning.

- Slow learning method : here the degree of training of the recognition neuron's weights towards the input vector is calculated using differential equations and is thus dependent on the length of time the input vector is presented.
- Fast learning method : here the algebraic equations are used to calculate degree of weight adjustments to be made, and binary values are used.

Note : While fast learning is effective and efficient for a variety of tasks, the slow learning method is more biologically plausible and can be used with continuous-time networks (i.e. when the input vector can vary continuously).

- **Types of ART Systems :**

The ART Systems have many variations :
ART 1, ART 2, Fuzzy ART, ARTMAP

- **ART 1:** The simplest variety of ART networks, accept only binary inputs.
- **ART 2 :** It extends network capabilities to support continuous inputs.
- **Fuzzy ART :** It Implements fuzzy logic into ART's pattern recognition, thus enhances generalizing ability. One very useful feature of fuzzy ART is complement coding, a means of incorporating the absence of features into pattern classifications, which goes a long way towards preventing inefficient and unnecessary category proliferation.
- **ARTMAP :** Also known as Predictive ART, combines two slightly modified ARTs , may be two ART-1 or two ART-2 units into a supervised learning structure where the first unit takes the input data and the second unit takes the correct output data, then used to make the minimum possible adjustment of the vigilance parameter in the first unit in order to make the correct classification.

7. Applications of Soft Computing

The applications of Soft Computing have proved two main advantages.

- First, in solving nonlinear problems, where mathematical models are not available, or not possible.
- Second, introducing the human knowledge such as cognition, recognition, understanding, learning, and others into the fields of computing.

This resulted in the possibility of constructing intelligent systems such as autonomous self-tuning systems, and automated designed systems.

The relevance of soft computing for pattern recognition and image processing is already established during the last few years. The subject has recently gained importance because of its potential applications in problems like :

- Remotely Sensed Data Analysis,
- Data Mining, Web Mining,
- Global Positioning Systems,
- Medical Imaging,
- Forensic Applications,
- Optical Character Recognition,
- Signature Verification,
- Multimedia,
- Target Recognition,
- Face Recognition and
- Man Machine Communication.

8. References : Textbooks

1. *"Neural Network, Fuzzy Logic, and Genetic Algorithms - Synthesis and Applications", by S. Rajasekaran and G.A. Vijayalaksmi Pai, (2005), Prentice Hall, Chapter 1-15, page 1-435.*

2. *"Soft Computing and Intelligent Systems - Theory and Application", by Naresh K. Sinha and Madan M. Gupta (2000), Academic Press, Chapter 1-25, page 1-625.*

3. *"Soft Computing and Intelligent Systems Design - Theory, Tools and Applications", by Fakhreddine karray and Clarence de Silva (2004), Addison Wesley, chapter 1-10, page 1-533.*

4. *"Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence" by J. S. R. Jang, C. T. Sun, and E. Mizutani, (1996), Prentice Hall, Chapter 1-15, page 1-607.*

5. *"Soft Computing : Integrating Evolutionary, Neural, and Fuzzy Systems", by Tettamanzi, Andrea, Tomassini, and Marco. (2001), Springer, Chapter 1-9, page 1-323.*

6. *"Neural Networks: A Comprehensive Foundation", by Simon S. Haykin, (1999), Prentice Hall, Chapter 1-15, page 1-889.*

7. *"Elements of Artificial Neural Networks", by Kishan Mehrotra, Chilukuri K. Mohan and Sanjay Ranka, (1996), MIT Press, Chapter 1-7, page 1-339.*

8. *"Fundamentals of Neural Networks: Architecture, Algorithms and Applications", by Laurene V. Fausett, (1993), Prentice Hall, Chapter1-7, page 1-449.*

9. *"Neural Network Design", by Martin T. Hagan, Howard B. Demuth and Mark Hudson Beale, (1996) , PWS Publ. Company, Chapter 1-19, page 1-1 to 19-14.*

10. *"An Introduction to Neural Networks", by James A. Anderson, (1997), MIT Press, Chapter 1- 17, page 1-585.*

11. *"Fuzzy Sets and Fuzzy Logic: Theory and Applications", by George J. Klir and Bo Yuan, (1995), Prentice Hall, Chapter 1-17, page 1-574.*

12. *"Introduction To Fuzzy Sets And Fuzzy Logic", by M Ganesh, (2008), Prentice-hall, Chapter 1-10, page 1- 256.*

13. *"Fuzzy Logic: Intelligence, Control, and Information", by John Yen, Reza Langari, (1999), Prentice Hall, Chapter 1-17, page 1-543.*

14. *"Fuzzy Logic with Engineering Applications", by Timothy Ross, (2004), John Wiley & Sons Inc, Chapter 1-15 , page 1-623.*

15. *"Fuzzy Logic and Neuro Fuzzy Applications Explained", by Constantin Von Altrock, (1995), Prentice Hall, Chapter 1-8, page 1-368.*

16. *"Genetic Algorithms in Search, Optimization, and Machine Learning", by David E. Goldberg, (1989), Addison-Wesley, Chapter 1-8, page 1- 432.*

17. *"An Introduction to Genetic Algorithms", by Melanie Mitchell, (1998), MIT Press, Chapter 1- 6, page 1- 203,*

18. *"Genetic Algorithms: Concepts And Designs", by K. F. Man, K. S. and Tang, S. Kwong, (201), Springer, Chapter 1- 10, page 1- 348,*

19. *"Genetic algorithms and engineering design", by Mitsuo Gen, and Runwei Cheng, (1997), John Wiley & Sons Inc, chapter 1- 10, page 1-411.*

20. *"Practical genetic algorithms", by Randy L. Haupt, (2004), John Wiley & Sons Inc, Chapter 1- 7, page 1- 251.*

21. *Related documents from open source, mainly internet. An exhaustive list is being prepared for inclusion at a later date.*