LECTURE NOTES
ON
PRINCIPLES OF SOFT COMPUTING

PREPARED BY

DR. PRASHANTA KUMAR PATRA

COLLEGE OF ENGINEERING AND TECHNOLOGY, BHUBANESWAR

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		**Evolutionary**		**Neural**		**Fuzzy**
Computing		**Computing**		**Network**		**Logic**
Zadeh		**Rechenberg**		**McCulloch**		**Zadeh**
1981		**1960**		**1943**		**1965**

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

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.

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.

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

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

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

$$\propto_{\mathbf{A}} \mathbf{(x)} = \begin{cases} \mathbf{1} & \textbf{if } \mathbf{x} \in \mathbf{X} \\ \mathbf{0} & \textbf{otherwise} \end{cases} \qquad \text{Eq.(2)}$$

– Thus, in classical set theory $\propto_{\mathbf{A}}$ **(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 $\propto_A(x)$** of Eq.(2) has only values **0** ('false') and **1** ('true'').

    Such sets are **crisp sets.**

  – For Non-crisp sets the characteristic function $\propto_A(x)$ can be defined.

    - The characteristic function $\propto_A(x)$ of Eq. (2) for the crisp set is generalized for the Non-crisp sets.
    - This generalized characteristic function $\propto_A(x)$ 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.

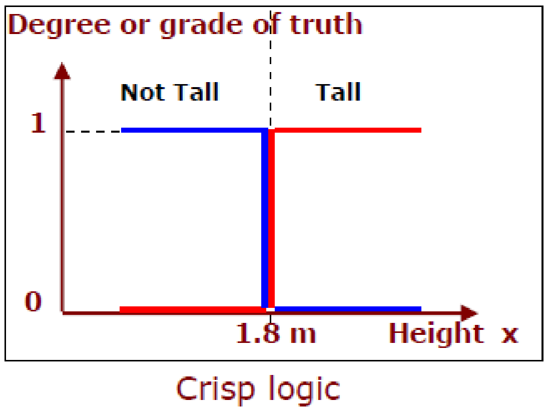
  - If we remove one grain from a heap of grains, we will still have a heap.
  - However, if we keep removing one-by-one grain from a heap of grains, there will be a time when we do not have a heap anymore.
  - The question is, at what time does the heap turn into a countable collection of grains that 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.



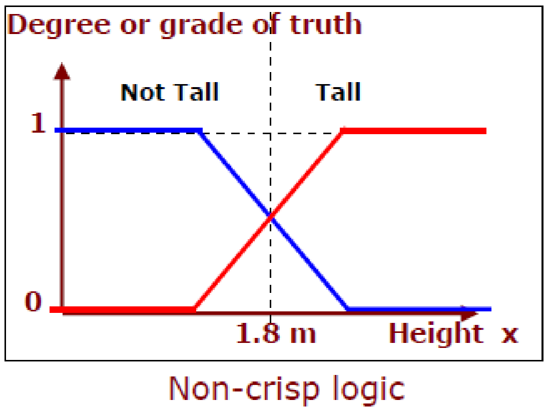


Fig. 1 Set Representation – Degree or grade of truth

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** μ_A **(x)** of **A**. This membership function associates to each element **x_σ** $\in$ **X** a number as μ_A **(x_σ)** in the closed unit interval **[0, 1]**.

  The number μ_A **(x_σ)** represents the **degree of membership** of **x_σ** in **A**.
- The notation used for membership function μ_A **(x)** of a fuzzy set **A** is

  A : X → [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

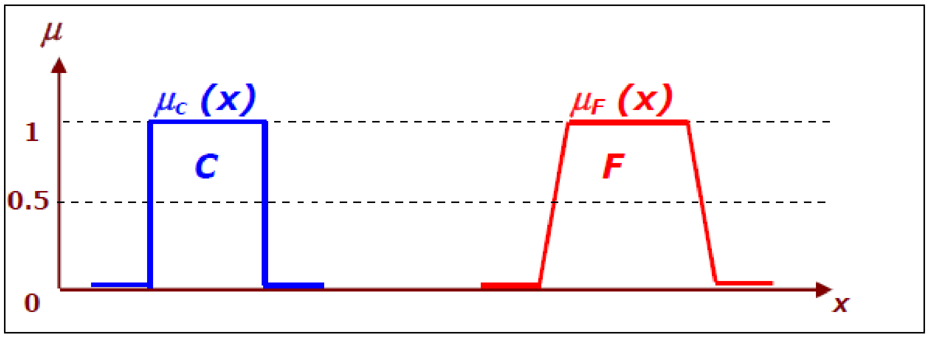


Fig. 2 Membership function of a Crisp set C and Fuzzy set F

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

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

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 $\leq$ **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 = { xi } = {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"}]

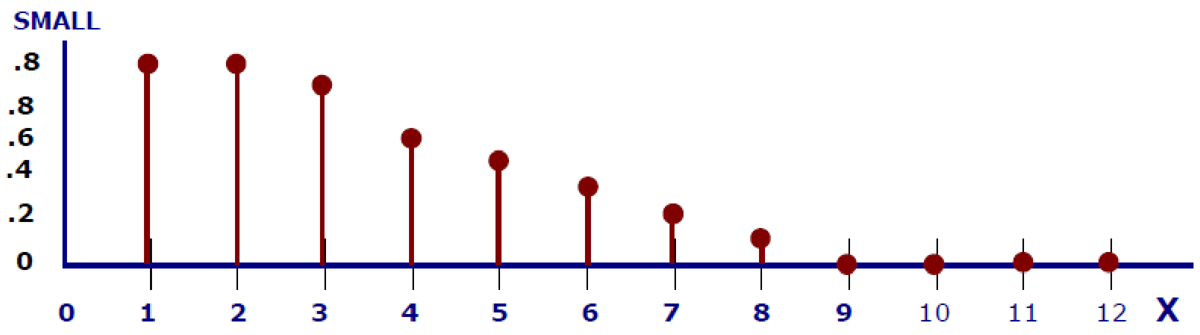


Fig Graphic Interpretation of Fuzzy Sets SMALL

- **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"}]

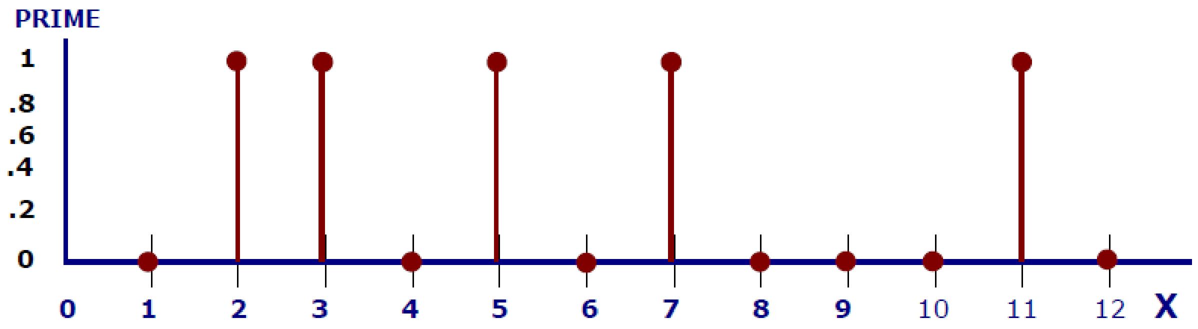


Fig Graphic Interpretation of Fuzzy Sets PRIME

- **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 = { xi } = {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 "}]

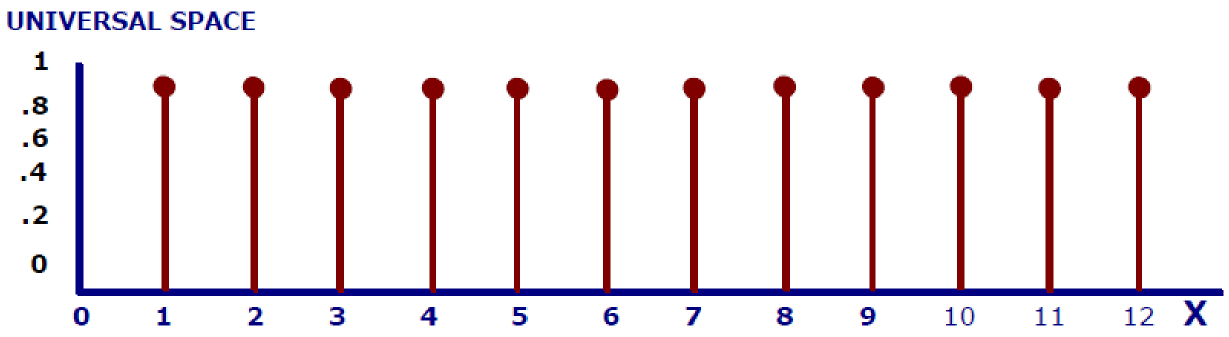


Fig Graphic Interpretation of Fuzzy Set UNIVERSALSPACE

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 "}]

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 $\cup$ MEDIUM]**
Intersection :	**FUZZYINTERSECTON = [SMALL $\cap$ 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 = { xi } = {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 :

nclusion

Include

**[VERYSMALL,
SMALL]**

- **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** if and only if are equal, which is denoted **X = Y**

  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]

  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]

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]

- **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}]

  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:**

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

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}]

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.

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.

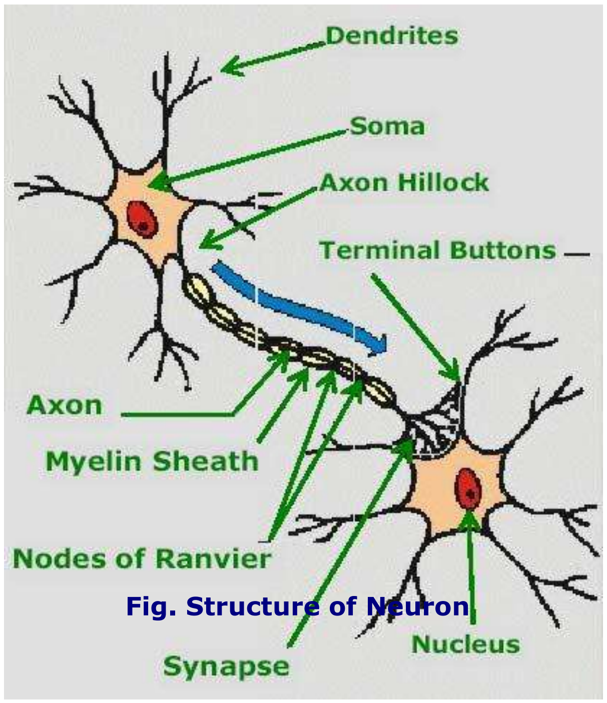


Fig. Structure of Neuron

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 $\propto$m) 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.

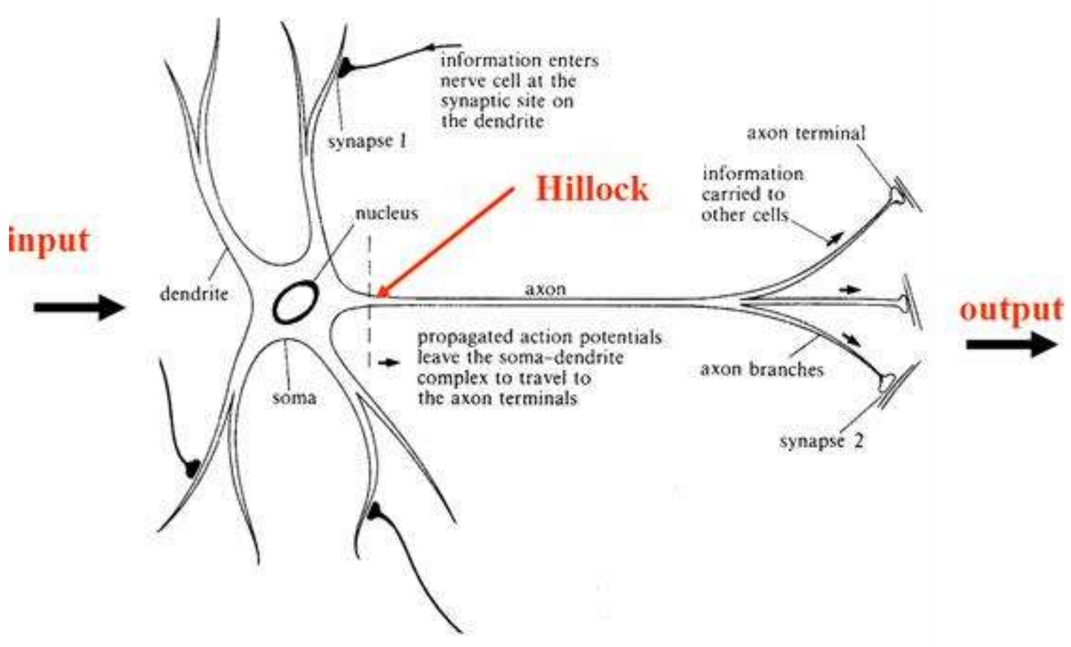

information enters nerve cell at the synaptic site on the dendrite
synapse 1
axon terminal
Hillock
information carried to other cells
input
nucleus
dendrite
axon
output
propagated action potentials leave the soma-dendrite complex to travel to the axon terminals
axon branches
soma
synapse 2

Fig. Structure of a neural cell in the human brain

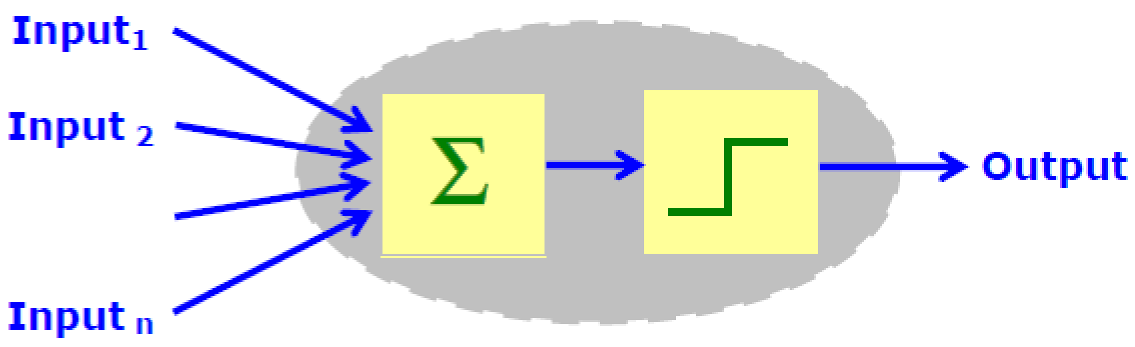
- Dendrites receive activation from other neurons.
- Soma processes the incoming activations and converts them into output activations.
- Axons act as transmission lines to send activation to other neurons.
- Synapses the junctions allow signal transmission between the axons and dendrites.
- The process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

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

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.

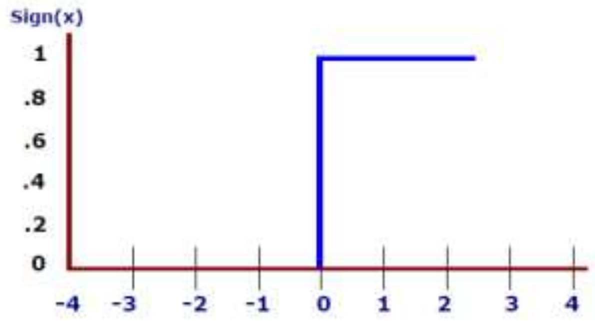
  In other words, the input to a neuron arrives in the form of signals. The signals build up in the cell. Finally the cell fires (discharges) through the output. The cell can start building up signals again.

- **Functions :**

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

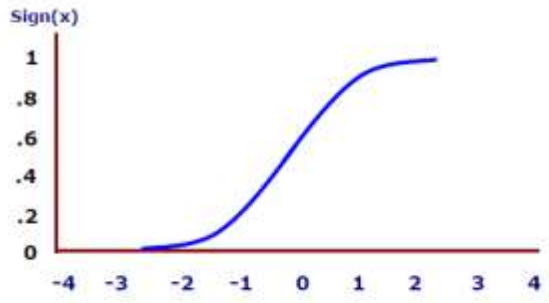
  - **Threshold or Sign function sgn(x)** : defined as

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



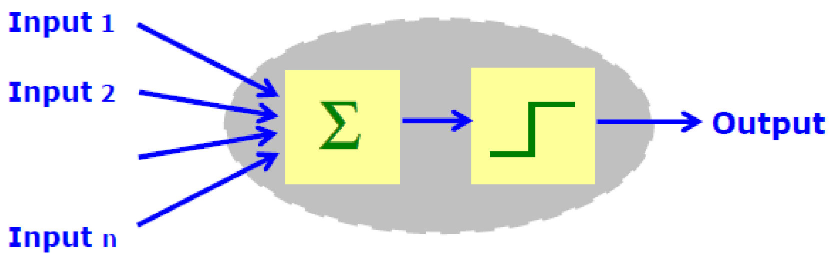
  - **Threshold or Sign function sigmoid (x)** : defined as a smoothed (differentiable) form of the threshold function

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



- **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 :

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

where Φ is the neuron's activation threshold.

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

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

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.

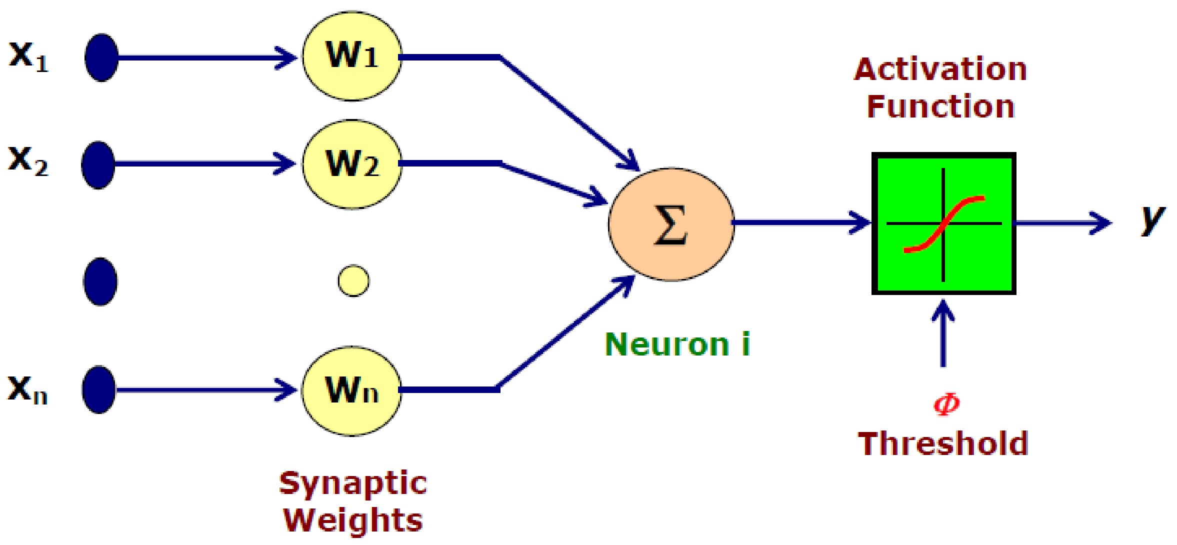


Fig Basic Elements of an Artificial Neuron

Weighting Factors

The values $\mathbf{W_1, W_2, \ldots W_n}$ are weights to determine the strength of input row vector $\mathbf{X = [x_1, x_2, \ldots, x_n]^T}$. Each input is multiplied by the associated weight of the neuron connection $\mathbf{X^T W}$. 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.

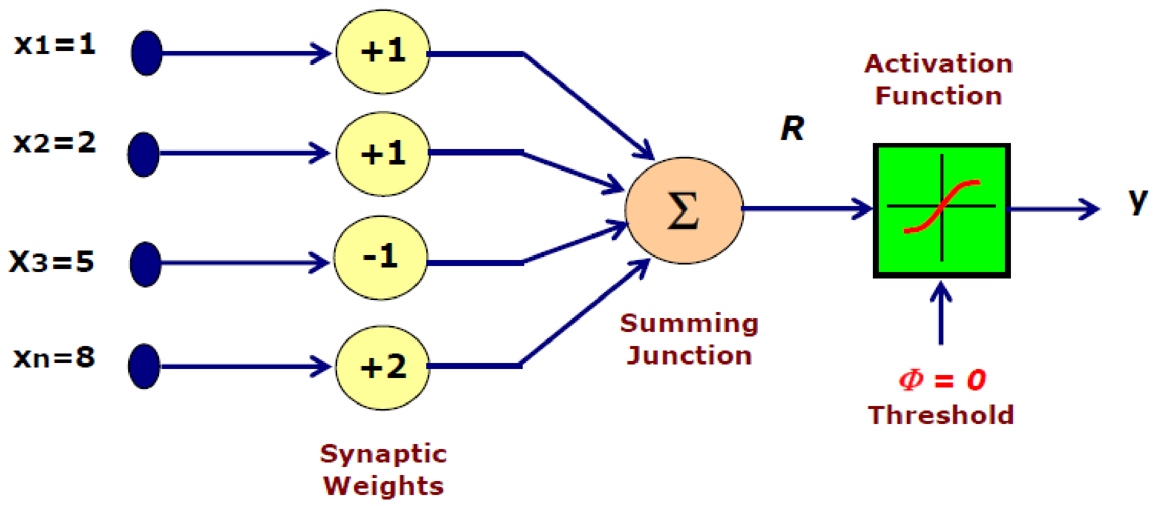


Fig Neuron Structure of Example

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:

$$y \text{ (threshold)} = 1$$

- **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**

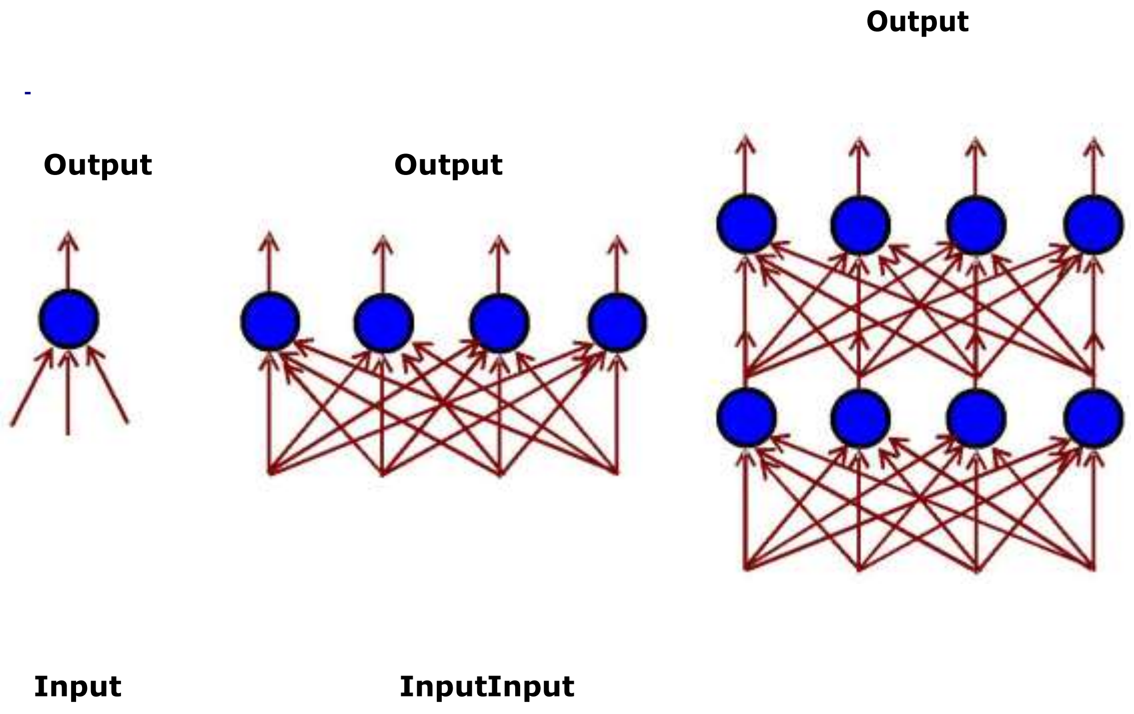


Fig Single and Multi - Layer Perceptrons

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.

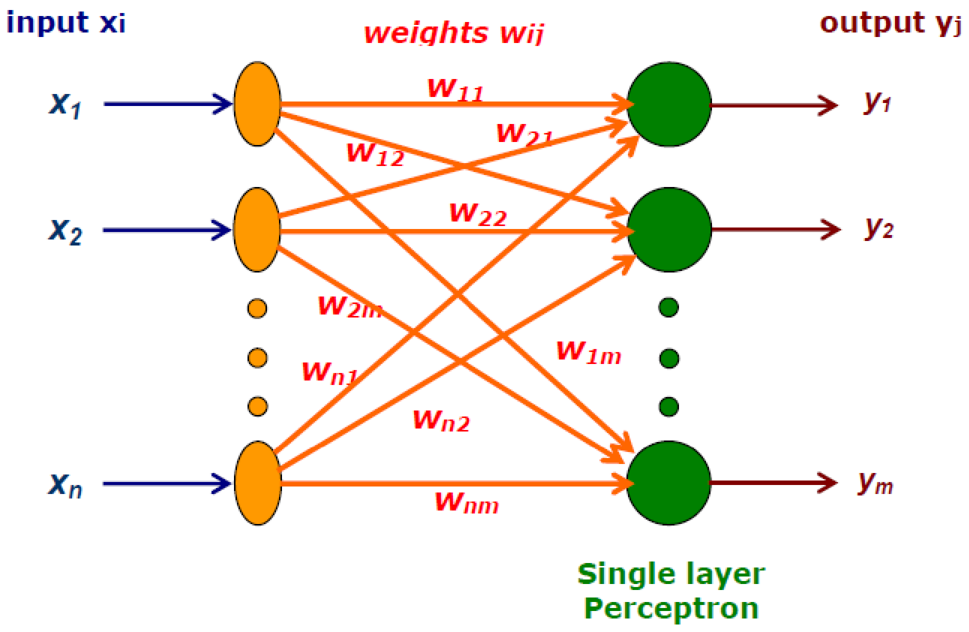


Fig. Simple Perceptron Model

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

A Perceptron is a powerful computational device.

Genetic Algorithms

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.

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**

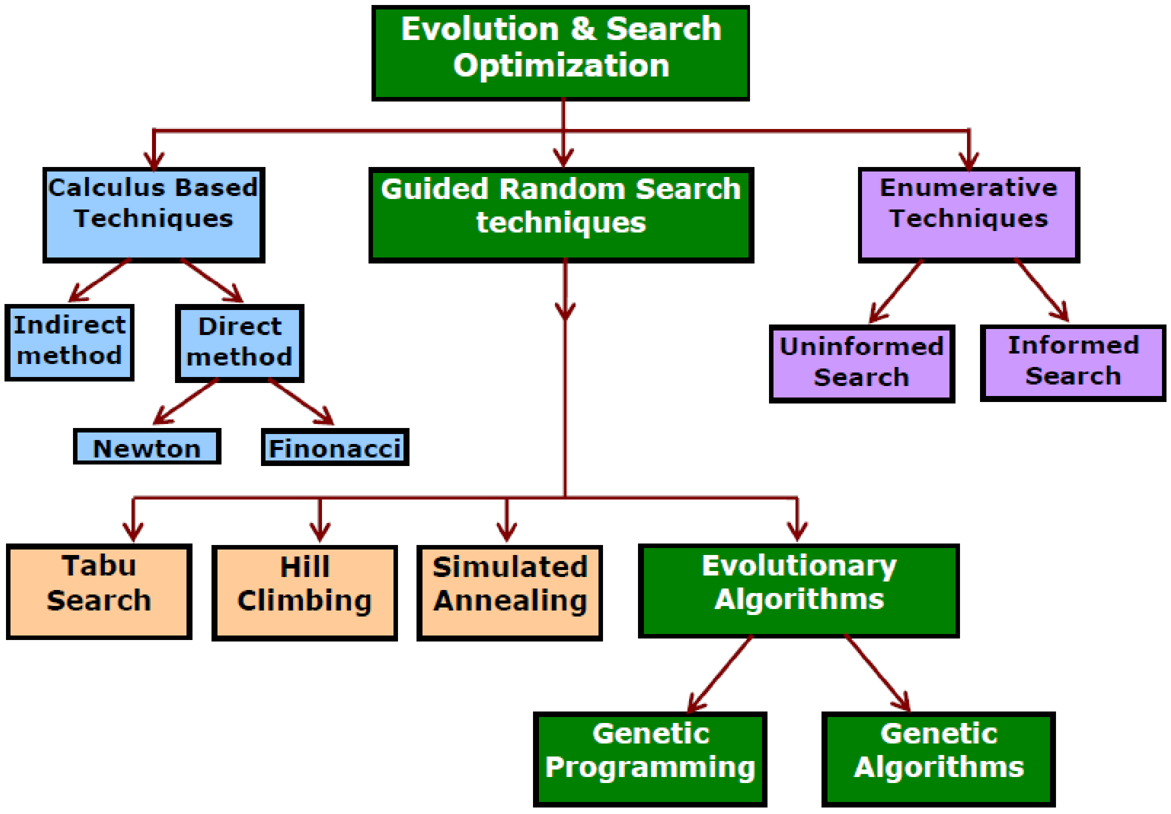


Fig Evolution & Search Optimization Techniques

- **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.

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.

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.

ƒ An **auto-associative memory** retrieves a previously stored pattern that most closely resembles the current pattern.

ƒ 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.

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.

- 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.

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.

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.

Fundamentals of Neural Networks

What is Neural Net ?

- A neural net is an artificial representation of the human brain that tries to simulate its learning process. An artificial neural network (ANN) is often called a "Neural Network" or simply Neural Net (NN).

- Traditionally, the word neural network is referred to a network of biological neurons in the nervous system that process and transmit information.

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

- The artificial neural networks are made of interconnecting artificial neurons which may share some properties of biological neural networks.

- Artificial Neural network is a network of simple processing elements (neurons) which can exhibit complex global behavior, determined by the connections between the processing elements and element parameters.

Introduction

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.

- 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 of ANNs as well.

Why Neural Network

Neural Networks follow a different paradigm for computing.

- The conventional computers are good for **-** fast arithmetic and does what programmer programs, ask them to do.
- The conventional computers are not so good for - interacting with noisy data or data from the environment, massive parallelism, fault tolerance, and adapting to circumstances.
- The neural network systems help where we can not formulate an algorithmic solution or where we can get lots of examples of the behavior we require.
- Neural Networks follow different paradigm for computing.

  The von Neumann machines are based on the processing/memory abstraction of human information processing.

  The neural networks are based on the parallel architecture of biological brains.

- Neural networks are a form of multiprocessor computer system, with
  - simple processing elements ,
  - a high degree of interconnection,
  - simple scalar messages, and
  - adaptive interaction between elements.

Research History

The history is relevant because for nearly two decades the future of Neural network remained uncertain.

McCulloch and Pitts (1943) are generally recognized as the designers of the first neural network. They combined many simple processing units together that could lead to an overall increase in computational power. They suggested many ideas like : a neuron has a threshold level and once that level is reached the neuron fires. It is still the fundamental way in which ANNs operate. The McCulloch and Pitts's network had a fixed set of weights.

Hebb (1949) developed the first learning rule, that is if two neurons are active at the same time then the strength between them should be increased.

In the 1950 and 60's, many researchers (Block, Minsky, Papert, and Rosenblatt worked on perceptron. The neural network model could be proved to converge to the correct weights, that will solve the problem. The weight adjustment (learning algorithm) used in the perceptron was found more powerful than the learning rules used by Hebb. The perceptron caused great excitement. It was thought to produce programs that could think.

Minsky & Papert (1969) showed that perceptron could not learn those functions which are not linearly separable.

The neural networks research declined throughout the 1970 and until mid 80's because the perceptron could not learn certain important functions.

Neural network regained importance in 1985-86. The researchers, Parker and LeCun discovered a learning algorithm for multi-layer networks called back propagation that could solve problems that were not linearly separable.

Biological Neuron 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 only makes the brain's abilities possible.

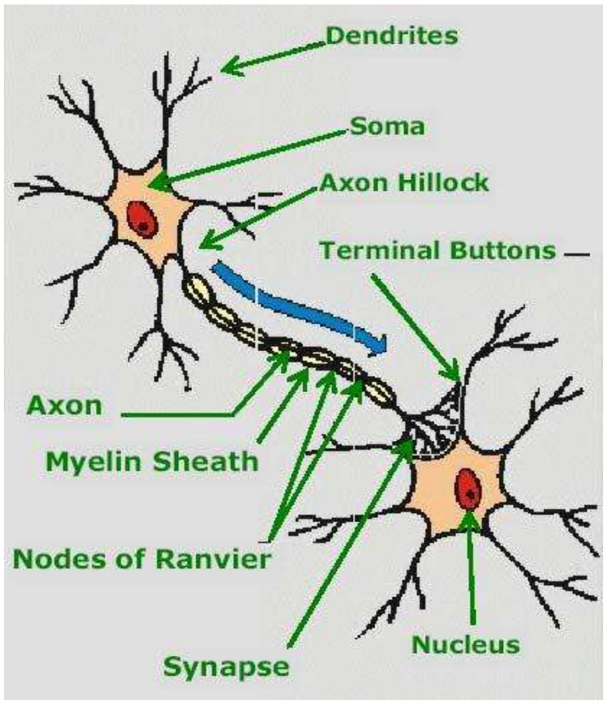


Fig. Structure of Neuron

Dendrites are branching fibers that extend 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 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.

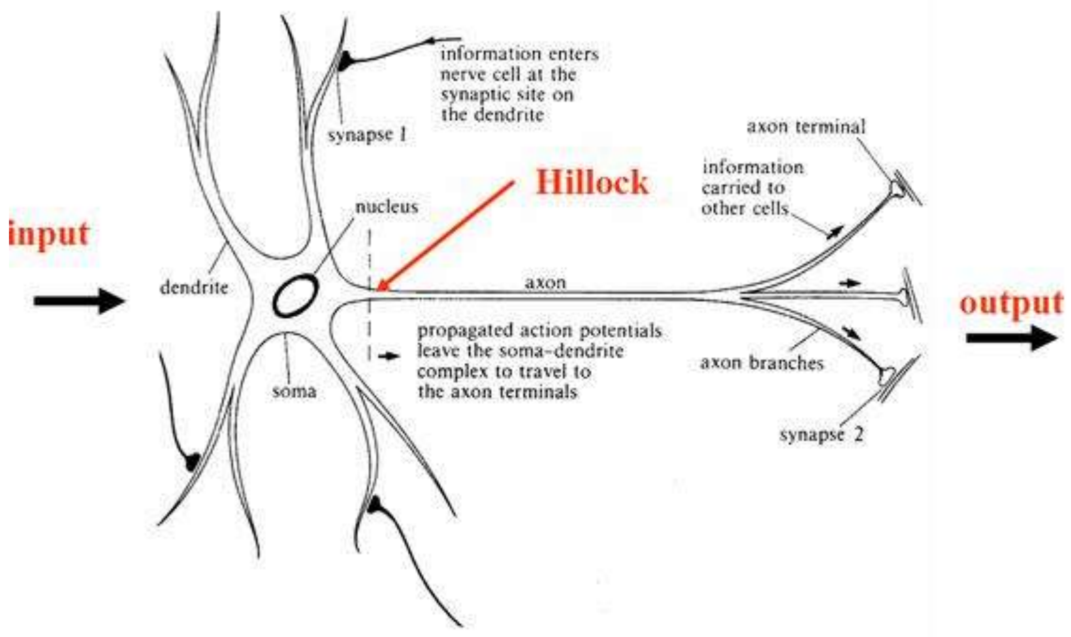
Nodes of Ranvier are the gaps (about 1 ∝m) between myelin sheath cells long axons are Since fat serves as a good insulator, the myelin sheaths speed the rate of transmission of an electrical impulse along the axon.

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.



information enters nerve cell at the synaptic site on the dendrite
synapse 1
axon terminal
Hillock
information carried to other cells
input
nucleus
dendrite
axon
output
propagated action potentials leave the soma-dendrite complex to travel to the axon terminals
axon branches
soma
synapse 2

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.

1.4 Artificial Neuron Model

An artificial neuron is a mathematical function conceived as a simple model of a real (biological) neuron.

- **The McCulloch-Pitts Neuron**

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

  ▪ A set of input connections brings in activations from other neurons.

  ▪ A processing unit sums the inputs, and then applies a non-linear activation function (i.e. squashing / 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 discharges (cell fires) through the output .
  - The cell can start building up signals again.

Single Layer Feed-forward Network

The Single Layer Feed-forward Network consists of a single layer of weights , where the inputs are directly connected to the outputs, via a series of weights. The synaptic links carrying weights connect every input to every output , but not other way. This way it is considered a network of **feed-forward** type. The sum of the products of the weights and the inputs is calculated in each neuron node, and if the value is above some threshold (typically **0**) the neuron fires and takes the activated value (typically **1**); otherwise it takes the deactivated value (typically **-1**).

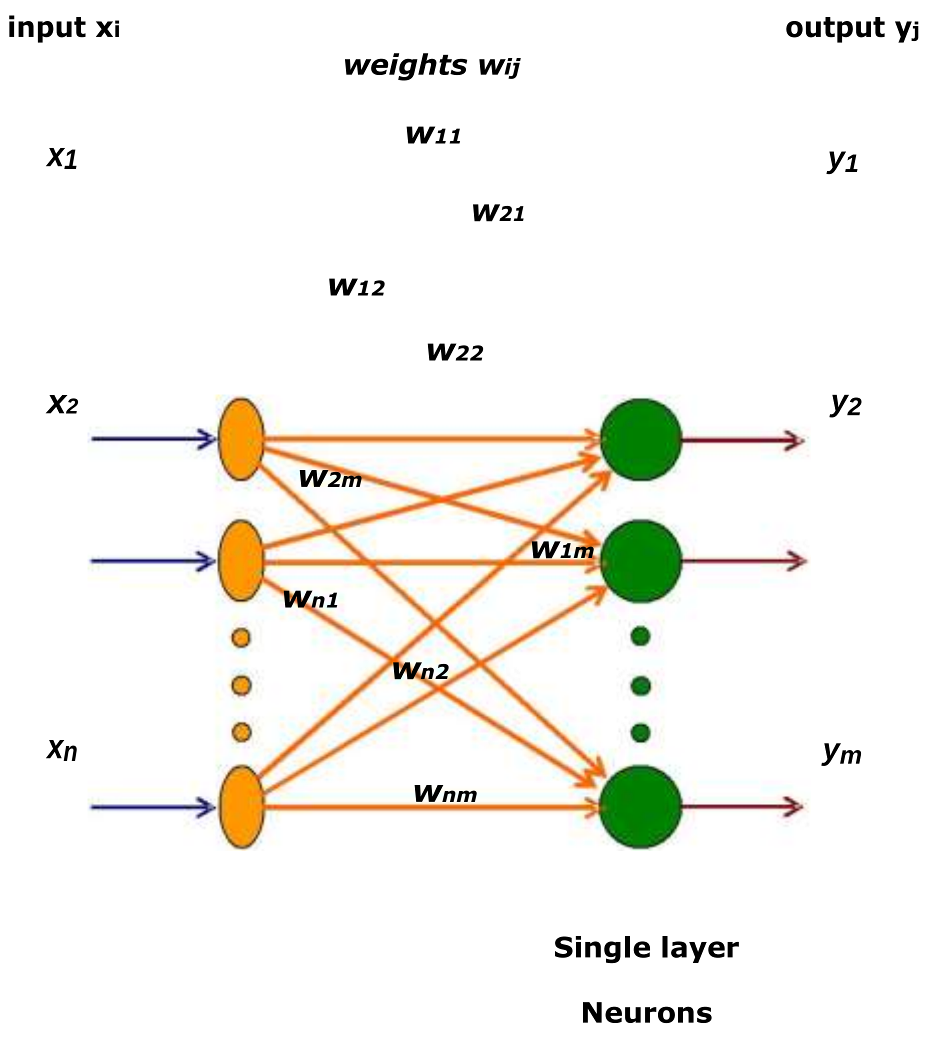


Fig. Single Layer Feed-forward Network

Multi Layer Feed-forward Network

The name suggests, it consists of multiple layers. The architecture of this class of network, besides having the input and the output layers, also have one or more intermediary layers called hidden layers. The computational units of the hidden layer are known as hidden neurons.

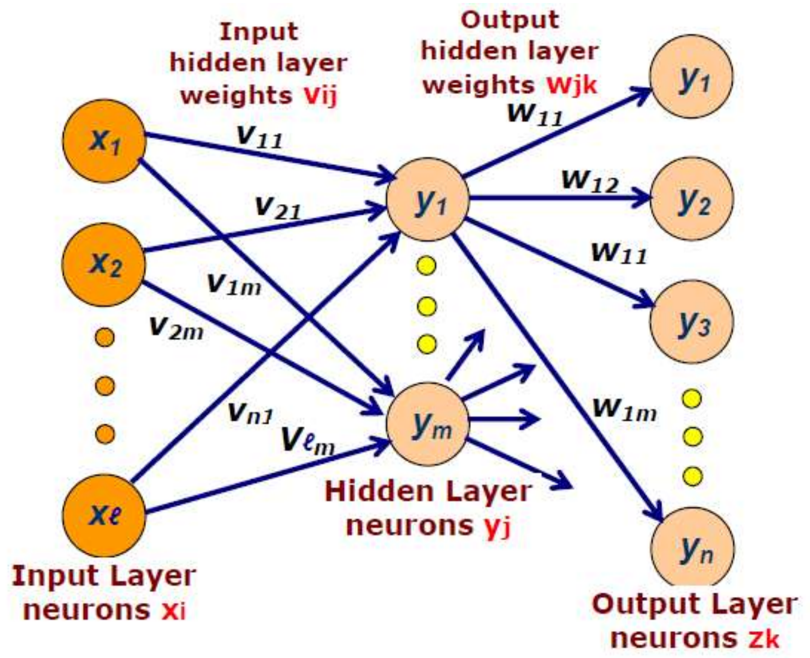


Fig. Multilayer feed-forward network in *(ℓ – m – n)* configuration.

- The hidden layer does intermediate computation before directing the input to output layer.
- The input layer neurons are linked to the hidden layer neurons; the weights on these links are referred to as **input-hidden layer weights**.
- The hidden layer neurons and the corresponding weights are referred to as **output-hidden layer weights**.

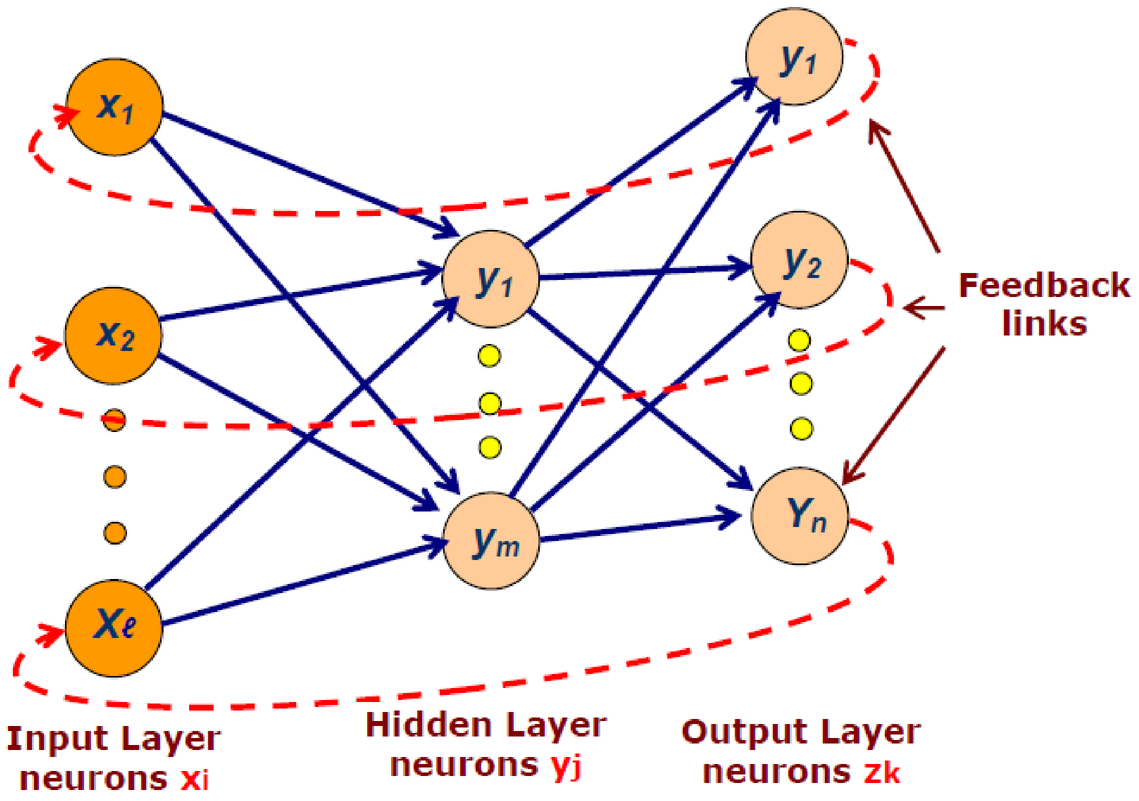
- A multi-layer feed-forward network with **ℓ** input neurons, **m_1** neurons in the first hidden layers, **m_2** neurons in the second hidden layers, and n output neurons in the output layers is written as **(ℓ - m_1 - m_2 – n)**.

The Fig. above illustrates a multilayer feed-forward network with a configuration ***(ℓ - m – n).***

Recurrent Networks

The Recurrent Networks differ from feed-forward architecture. A Recurrent network has at least one feed back loop.

Example :



There could be neurons with self-feedback links; that is the output of a neuron is fed back into it self as input.

Learning Methods in Neural Networks

The learning methods in neural networks are classified into three basic types :

- Supervised Learning,
- Reinforced Learning

These three types are classified based on :

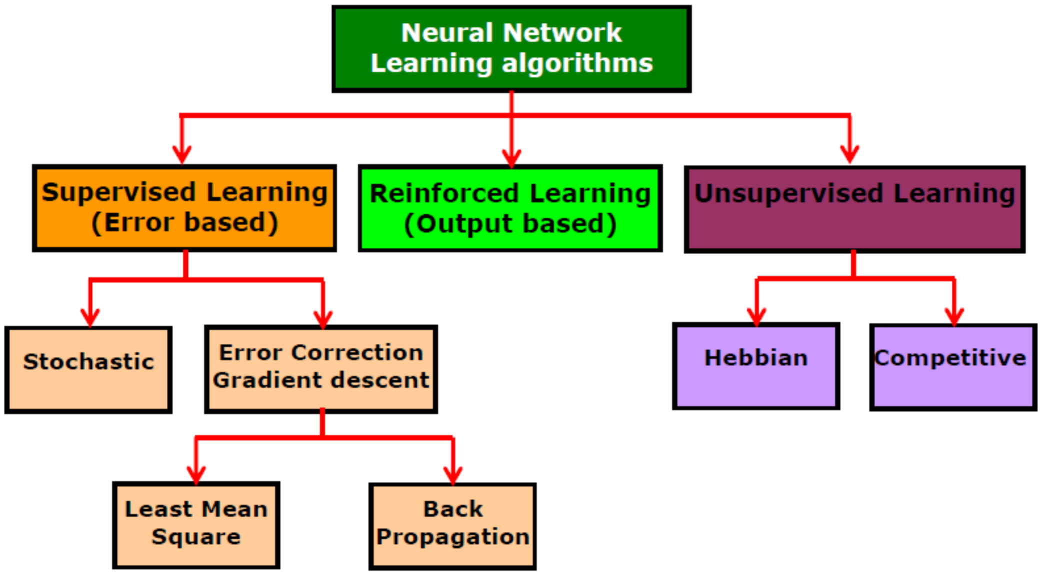
- presence or absence of **teacher** and
- the information provided for the system to learn.

These are further categorized, based on the **rules** used, as

- Hebbian,
- Gradient descent,
- Competitive and
- Stochastic learning.

- **Classification of Learning Algorithms**

  Fig. below indicate the hierarchical representation of the algorithms mentioned in the previous slide. These algorithms are explained in subsequent slides.



Neural Network Learning algorithms
Supervised Learning (Error based)
Reinforced Learning (Output based)
Unsupervised Learning
Stochastic
Error Correction Gradient descent
Hebbian
Competitive
Least Mean Square
Back Propagation

Fig. Classification of learning algorithms

b Supervised Learning

A teacher is present during learning process and presents expected output.

Every input pattern is used to train the network.

Learning process is based on comparison, between network's computed output and the correct expected output, generating "error".

The "error" generated is used to change network parameters that result improved performance.

c Unsupervised Learning

No teacher is present.

The expected or desired output is not presented to the network.

The system learns of it own by discovering and adapting to the structural features in the input patterns.

d Reinforced learning

A teacher is present but does not present the expected or desired output but only indicated if the computed output is correct or incorrect.

The information provided helps the network in its learning process.

A reward is given for correct answer computed and a penalty for a wrong answer.

Note : The Supervised and Unsupervised learning methods are most popular forms of learning compared to Reinforced learning.

- **Hebbian Learning**

  Hebb proposed a rule based on correlative weight adjustment.

  In this rule, the input-output pattern pairs ***(Xi , Yi)*** are associated by the weight matrix ***W***, known as correlation matrix computed as

$$W = \sum_{i=1}^{n} X_i \, Y_i^{T}$$

  where Y_i^{T} is the transpose of the associated output vector Y_i

  There are many variations of this rule proposed by the other researchers (Kosko, Anderson, Lippman) .

- **Gradient descent Learning**

This is based on the minimization of errors ***E*** defined in terms of weights and the activation function of the network.

- Here, the activation function of the network is required to be differentiable, because the updates of weight is dependent on the gradient of the error ***E***.

- If ΔW_{ij} is the weight update of the link connecting the ***i th*** and the ***j th*** neuron of the two neighboring layers, then ΔW_{ij} is defined as

$$\Delta W_{ij} = \eta \, (\partial E / \partial W_{ij})$$

where η is the learning rate parameters and $(\partial E / \partial W_{ij})$ is error gradient with reference to the weight W_{ij} .

Note : The Hoffs Delta rule and Back-propagation learning rule are the examples of Gradient descent learning.

- **Competitive Learning**

  In this method, those neurons which respond strongly to the input stimuli have their weights updated.

  When an input pattern is presented, all neurons in the layer compete, and the winning neuron undergoes weight adjustment .

  This strategy is called "winner-takes-all".

- **Stochastic Learning**

  In this method the weights are adjusted in a probabilistic fashion.

  - Example : Simulated annealing which is a learning mechanism employed by Boltzmann and Cauchy machines.

- **Taxonomy Of Neural Network Systems**

  In the previous sections, the Neural Network Architectures and the Learning methods have been discussed. Here the popular neural network

  systems are listed. The grouping of these systems in terms of architectures and the learning methods are presented in the next slide.

  - **Neural Network Systems**
    - ADALINE (Adaptive Linear Neural Element)
    - ART (Adaptive Resonance Theory)
    - AM (Associative Memory)
    - BAM (Bidirectional Associative Memory)
    - Boltzmann machines
    - BSB (Brain-State-in-a-Box)
    - Cauchy machines
    - Hopfield Network
    - LVQ (Learning Vector Quantization)
    - Neoconition

- Perceptron
- RBF (Radial Basis Function)
- RNN (Recurrent Neural Network)
- SOFM (Self-organizing Feature Map)

30

- **Classification of Neural Network**

A taxonomy of neural network systems based on Architectural types and the Learning methods is illustrated below.

		Learning Methods			
		Gradient descent	**Hebbian**	**Competitive**	**Stochastic**
	Single-layer feed-forward	**ADALINE, Hopfield, Percepton,**	**AM, Hopfield,**	**LVQ, SOFM**	**-**
	Multi-layer feed- forward	**CCM, MLFF, RBF**	**Neocognition**		
	Recurrent Networks	**RNN**	**BAM, BSB, Hopfield,**	**ART**	**Boltzmann and Cauchy machines**

Table : Classification of Neural Network Systems with respect to learning methods and Architecture types

31

■ **Single-Layer NN Systems**

Here, a simple Perceptron Model and an ADALINE Network Model is presented.

6.1 Single layer Perceptron

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

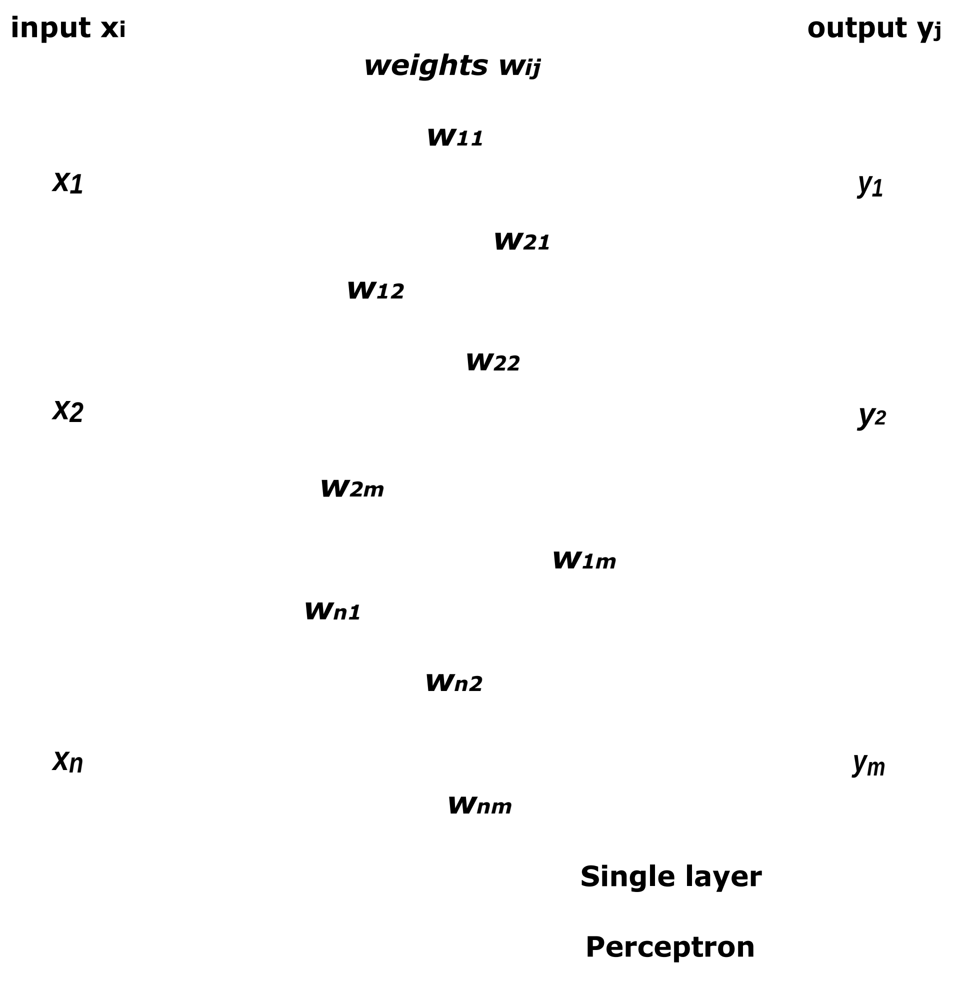


Fig. Simple Perceptron Model

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

■ **Learning Algorithm : Training Perceptron**

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

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

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

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

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

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

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

Where

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

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

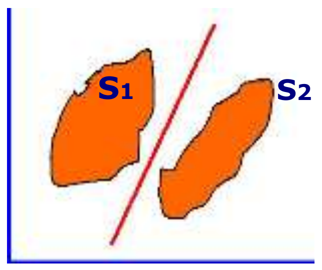
α small leads to slow and α large leads to fast learning.

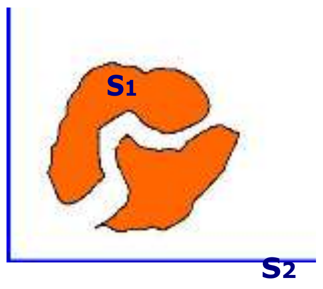
- **Perceptron and Linearly Separable Task**

Perceptron can not handle tasks which are not separable.

- Definition : Sets of points in 2-D space are linearly separable if the sets can be separated by a straight line.
- Generalizing, a set of points in n-dimensional space are linearly separable if there is a hyper plane of (n-1) dimensions separates the sets.

Example





(a) Linearly separable patterns **(b) Not Linearly separable patterns**

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

- **XOR Problem :**
  Exclusive OR operation

Input x1	Input x2	Output
0	0	0
1	1	0
0	1	1
1	0	1

XOR truth table

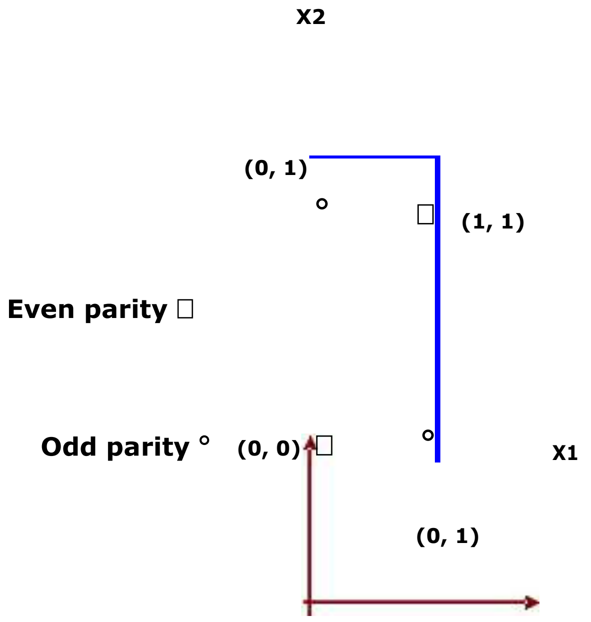


Fig. Output of XOR in

X1 , x2 plane

Even parity is, even number of 1 bits in the input

Odd parity is, odd number of 1 bits in the input

- There is no way to draw a single straight line so that the circles are on one side of the line and the dots on the other side.
- Perceptron is unable to find a line separating even parity input patterns from odd parity input patterns.

35

- **Perceptron Learning Algorithm**

The algorithm is illustrated step-by-step.

- **Step 1 :**

Create a peceptron with **(n+1)** input neurons $x_0, x_1, \ldots, x_n$,

where $x_0 = 1$ is the bias input.

$x_i \; w_i$

Let **O** be the output neuron.

- **Step 2 :**

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

- **Step 3 :**

for

Iterate through the input patterns X_j of the training set using the weight set; ie compute the weighted sum of inputs $net_j = \sum^{n}$

$i=1$

for each input pattern **j** .

- **Step 4 :**

Compute the output y_j using the step function

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

- **Step 5 :**

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

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

= **Step 6 :**

Otherwise, update the weights as given below :

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

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

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

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

where α is the learning parameter and is constant.

z **Step 7 :**

goto step 3

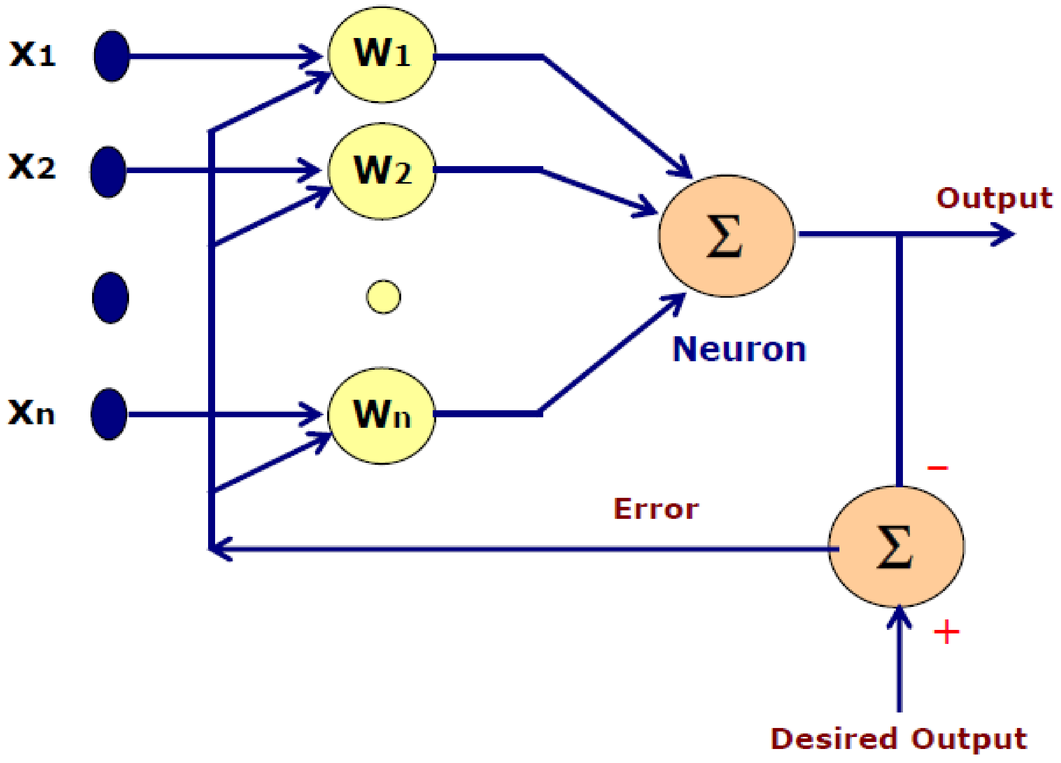
• **END**

36

6.2 ADAptive LINear Element (ADALINE)

An ADALINE consists of a single neuron of the McCulloch-Pitts type, where its weights are determined by the normalized least mean square (LMS) training law. The LMS learning rule is also referred to as delta rule. It is a well-established supervised training method that has been used over a wide range of diverse applications.

- **Architecture of a simple ADALINE**



- **ADALINE Training Mechanism**

  (Ref. Fig. in the previous slide - Architecture of a simple ADALINE)

  - The basic structure of an ADALINE is similar to a linear neuron with an extra feedback loop.

  - During the training phase of ADALINE, the input vector

    $X = [x_1 , x_2 , . . . , x_n]^T$ as well as desired output are presented

    to the network.

  - The weights are adaptively adjusted based on delta rule.

  - After the ADALINE is trained, an input vector presented to the

    network with fixed weights will result in a scalar output.

  - Thus, the network performs an **n** dimensional mapping to a scalar value.

  - The activation function is not used during the training phase. Once the weights are properly adjusted, the response of the trained unit can be tested by applying various inputs, which are

    not in the training set.If the network produces consistent

    responses to a high degree with the test inputs, it is said that the network could generalize. The process of training and generalization are two important attributes of this network.

Usage of ADLINE

In practice, an ADALINE is used to Make binary decisions; the output is sent through a binary threshold.

- Realizations of logic gates such as AND, NOT and OR .
- Realize only those logic functions that are linearly separable.

■ Applications of Neural Network

Neural Network Applications can be grouped in following categories:

Clustering:

A clustering algorithm explores the similarity between patterns and places similar patterns in a cluster. Best known applications include data compression and data mining.

Classification/Pattern recognition:

The task of pattern recognition is to assign an input pattern (like handwritten symbol) to one of many classes. This category includes algorithmic implementations such as associative memory.

- **Function approximation :**

The tasks of function approximation is to find an estimate of the unknown function subject to noise. Various engineering and scientific disciplines require function approximation.

- **Prediction Systems:**

The task is to forecast some future values of a time-sequenced data. Prediction has a significant impact on decision support systems. Prediction differs from function approximation by considering time factor. System may be dynamic and may produce different results for the same input data based on system state (time).

Back-Propagation Network

What is BPN ?

- A single-layer neural network has many restrictions. This network can accomplish very limited classes of tasks.

  Minsky and Papert (1969) showed that a two layer feed-forward network can overcome many restrictions, but they did not present a solution to the problem as "how to adjust the weights from input to hidden layer" ?

- An answer to this question was presented by Rumelhart, Hinton and Williams in 1986. The central idea behind this solution is that the errors for the units of the hidden layer are determined by back-propagating the errors of the units of the output layer.

  This method is often called the Back-propagation learning rule.

  Back-propagation can also be considered as a generalization of the delta rule for non-linear activation functions and multi-layer networks.

- Back-propagation is a systematic method of training multi-layer artificial neural networks.

1. Back-Propagation Network – Background

Real world is faced with a situations where data is incomplete or noisy. To make reasonable predictions about what is missing from the information

available is a difficult task when there is no a good theory available that may to help reconstruct the missing data. It is in such situations the Back-propagation (Back-Prop) networks may provide some answers.

- A BackProp network consists of at least three layers of units :
  - an **input layer**,
  - at least one intermediate **hidden layer**, and
  - an **output layer**.

- Typically, units are connected in a feed-forward fashion with input units fully connected to units in the hidden layer and hidden units fully connected to units in the output layer.

- When a BackProp network is cycled, an input pattern is propagated forward to the output units through the intervening input-to-hidden and hidden-to-output weights.

- The output of a BackProp network is interpreted as a classification decision.

- With BackProp networks, learning occurs during a training phase.
  The steps followed during learning are :

  - each input pattern in a training set is applied to the input units and then propagated forward.

  - the pattern of activation arriving at the output layer is compared with the correct (associated) output pattern to calculate an error signal.

  - the error signal for each such target output pattern is then back-propagated from the outputs to the inputs in order to appropriately adjust the weights in each layer of the network.

  - after a BackProp network has learned the correct classification for a set of inputs, it can be tested on a second set of inputs to see how well it classifies untrained patterns.

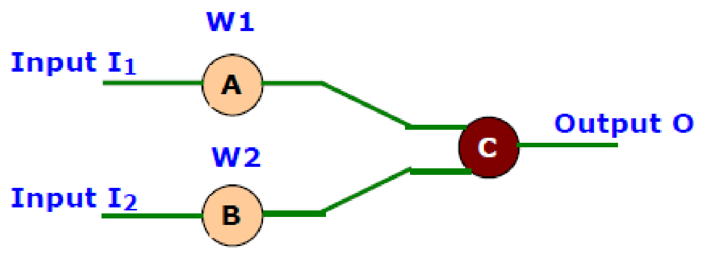
- An important consideration in applying BackProp learning is how well the network generalizes.

1.1 Learning :

AND function

Implementation of AND function in the neural network.

AND		
X1	X2	Y
0	0	0
0	1	0
1	0	0
1	1	1



AND function implementation

- there are 4 inequalities in the AND function and they must be satisfied.

$w_1 0 + w_2 0 < \theta$, $w_1 0 + w_2 1 < \theta$,

$w_1 1 + w_2 0 < \theta$, $w_1 1 + w_2 1 > \theta$

- one possible solution :

  if both weights are set to 1 and the threshold is set to 1.5, then

$(1)(0) + (1)(0) < 1.5$ **assign 0** , $(1)(0) + (1)(1) < 1.5$ **assign 0**

$(1)(1) + (1)(0) < 1.5$ **assign 0** , $(1)(1) + (1)(1) > 1.5$ **assign 1**

Although it is straightforward to explicitly calculate a solution to the AND function problem, but the question is "how the network can learn such a solution". That is, given random values for the weights can we define an incremental procedure which will cover a set of weights which implements AND function.

1.2 Simple Learning Machines

Rosenblatt (late 1950's) proposed learning networks called **Perceptron**. The task was to discover a set of connection weights which correctly classified a set of binary input vectors. The basic architecture of the perceptron is similar to the simple AND network in the previous example.

A perceptron consists of a set of input units and a single output unit.

in the AND network, the output of the perceptron is calculated ***n***

i=1

If the net input is greater than the threshold **θ** , then the output unit is

turned **on** , otherwise it is turned **off**.

To address the learning question, Rosenblatt solved two problems.

- first, defined a cost function which measured error.
- second, defined a procedure or a rule which reduced that error by

  appropriately adjusting each of the weights in the network.

However, the procedure (or **learning rule**) required to assesses the relative contribution of each weight to the total error.

The learning rule that Roseblatt developed, is based on determining the difference between the actual output of the network with the target output (**0** or **1**), called "**error measure**"

- **Error Measure** (learning rule)

  Mentioned in the previous slide, the error measure is the difference between actual output of the network with the target output (**0** or **1**).

  – If the input vector is correctly classified (i.e., zero error), then the weights are left unchanged, and

    the next input vector is presented.

  – If the input vector is incorrectly classified (i.e., not zero error), then there are two cases to consider :

    Case 1 : If the output unit is **1** but need to be **0** then

    ◇ the threshold is incremented by **1** (to make it less likely that the output unit would be turned on if the same input vector was presented again).

    ◇ If the input **Ii** is **0**, then the corresponding weight **Wi** is left unchanged.

    ◇ If the input **Ii** is **1**, then the corresponding weight **Wi** is

      decreased by **1**.

    Case 2 : If output unit is **0** but need to be **1** then the opposite changes are made.

09

- **Perceptron Learning Rule : Equations**

The perceptron learning rules are govern by two equations, –
one that defines the change in the threshold and

– the other that defines change in the weights,

The **change in the threshold** is given by

$$\Delta\ \theta = -(t_p - o_p) = -d_p$$

where **p** specifies the presented input pattern,

o_p actual output of the input pattern I_{pi}

t_p specifies the correct classification of the input pattern ie target,

d_p is the difference between the target and actual outputs.

The **change in the weights** are given by

$$\Delta\ w_i = (t_p - o_p)\ I_{pi} = -d_p\ I_{pi}$$

10

1.3 Hidden Layer

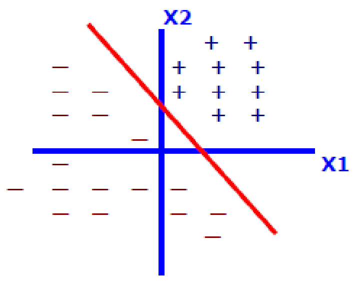
Back-propagation is simply a way to determine the error values in hidden layers. This needs be done in order to update the weights.

The best example to explain where back-propagation can be used is the XOR problem.

Consider a simple graph shown below.

- all points on the right side of the line are +ve, therefore the output of the neuron should be **+ve**.

- all points on the left side of the line are **–ve**, therefore the output of the neuron should be **–ve**.



With this graph, one can make a simple table of inputs and outputs as shown below.

AND

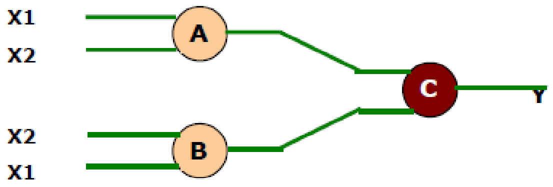
X1	***X2***	***Y***
1	**1**	**1**
1	**0**	**0**
0	**1**	**0**
0	**0**	**0**

Training a network to operate as an AND switch can be done easily through only one neuron *(see previous slides)*

But a XOR problem can't be solved using only one neuron.
If we want to train an XOR, we need 3 neurons, fully-connected in a feed-forward network as shown below.

XOR

X1	***X2***	***Y***
1	**1**	**0**
1	**0**	**1**
0	**1**	**1**
0	**0**	**0**



- **Back Propagation Network**

Learning By Example

Consider the Multi-layer feed-forward back-propagation network below.

The subscripts **I, H, O** denotes input, hidden and output neurons.

The weight of the arc between **i** **th** input neuron to **j** **th** hidden layer is **V_{ij}** .

The weight of the arc between **i th** hidden neuron to **j th** out layer is **W_{ij}**

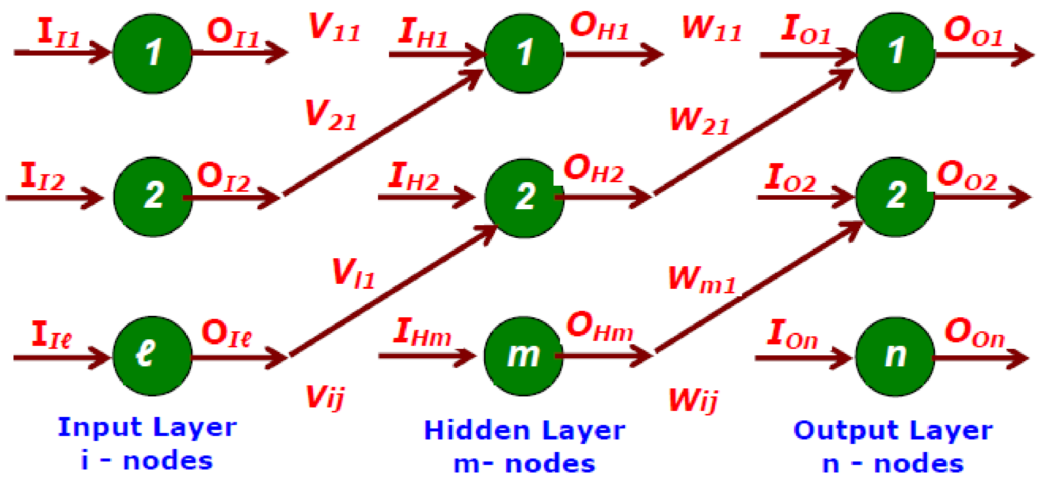


Fig Multi-layer feed-forward back-propagation network

2.1 Computation of Input, Hidden and Output Layers

- **Input Layer Computation**

Consider linear activation function.

If the output of the input layer is the input of the input layer and the transfer function is **1**, then

$$\underset{\ell \times 1}{\{ O \}_I} = \underset{\ell \times 1}{\{ I \}_I} \quad \text{(denotes matrix row, column size)}$$

The hidden neurons are connected by synapses to the input neurons.

- Let V_{ij} be the weight of the arc between i^{th} input neuron to j^{th} hidden layer.

- The input to the hidden neuron is the weighted sum of the outputs of the input neurons. Thus the equation

  $I_{Hp} = V_{1p} O_{I1} + V_{2p} O_{I2} + \ldots. + V_{1p} O_{I\ell}$ **where** $(p = 1, 2, 3 \ldots, m)$
  denotes weight matrix or connectivity matrix between input neurons

  and a hidden neurons as **[V]**.

we can get an input to the hidden neuron as $\ell \times m$

$$\underset{m \times 1}{\{ I \}_H} = \underset{m \times \ell}{[V]^T} \underset{\ell \times 1}{\{ O \}_I} \quad \text{(denotes matrix row, column size)}$$

- **Hidden Layer Computation**

Shown below the **p**th neuron of the hidden layer. It has input from the output of the input neurons layers. If we consider transfer function as sigmoidal function then the output of the **p**th hidden neuron is given by

$$O_{Hp} = \frac{1}{\left(1 + e^{-\lambda\,(I_{HP} - \theta_{HP})}\right)}$$

where	O_{Hp}	is the output of the **p**th hidden neuron,
	I_{Hp}	is the input of the **p**th hidden neuron, and is
	θ_{HP}	the threshold of the **p**th neuron;

Note : a non zero threshold neuro
that is always held at **-1** and the
weight value as shown in Fig. below

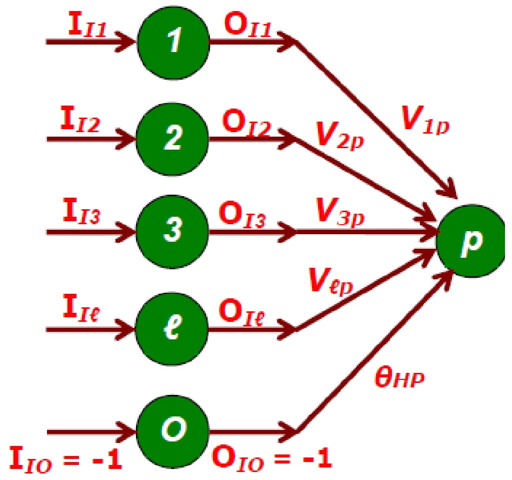


**Fig. Example of Treating thresh
in hidden layer**

Treating each component of the input of the hidden neuron separately, we get the outputs of the hidden neuron as given by above equation .

The input to the output neuron is the weighted sum of the outputs of the hidden neurons. Accordingly, I_{oq} the input to the **q**th output neuron is given by the equation

$$I_{oq} = W_{1q} O_{H1} + W_{2q} O_{H2} + \ldots + W_{mq} O_{Hm} , \quad \text{where } (q = 1, 2, 3 \ldots, n)$$

It denotes weight matrix or connectivity matrix between hidden neurons and output neurons as **[W]**, we can get input to output neuron as

$$\{ I \}_O = [W]^T \{ O \}_H$$

n x 1 n x m m x 1 (denotes matrix row, column size)

- **Output Layer Computation**

Shown below the **q**th neuron of the output layer. It has input from the output of the hidden neurons layers.

If we consider transfer function as sigmoidal function then the output of the **q**th output neuron is given by

Note : A non zero threshold neuron, is computationally equivalent to an input that is always held at **-1** and the non-zero threshold becomes the connecting weight value as shown in Fig. below.

Note : Here again the threshold may be tackled by considering extra **o**th neuron in the hidden layer with output of **-1** and the threshold value θ_{Oq} becomes the connecting weight value as shown in Fig. below.

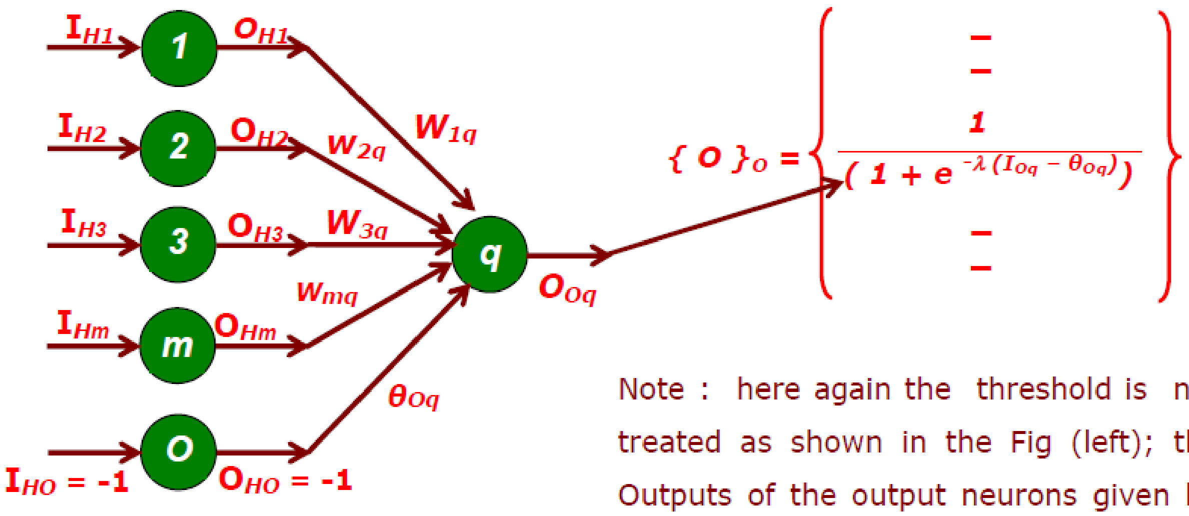


Fig. Example of Treating threshold in output layer

Note : here again the threshold is not treated as shown in the Fig (left); the Outputs of the output neurons given by the above equation.

$$O_{Oq} = \frac{1}{(1 + e^{-\lambda (I_{Oq} - \theta_{Oq})})}$$

where **O_{Oq}** is the output of the **q**th output neuron,

I_{Oq} is the input to the **q**th output neuron, and

θ_{Oq} is the threshold of the **q**th neuron;

2.2 Calculation of Error

(refer the earlier slides - Fig. "Multi-layer feed-forward back-propagation network" and a table indicating an 'nset' of input and out put data for the purpose of training*)*

Consider any **r** th output neuron. For the target out value **T**, mentioned in the table- 'nset' of input and output data" for the purpose of training, calculate output **O** .

The error norm in output for the **r** th output neuron is

$$E^1_r = (1/2)\, e^2_r = (1/2)\, (T - O)^2$$

where E^1_r is **1/2** of the second norm of the error e_r in the **r** th neuron for the given training pattern.

e^2_r is the square of the error, considered to make it independent of sign **+ve** or **–ve** , ie consider only the absolute value.

The Euclidean norm of error E^1 for the first training pattern is given by

$$E^1 = (1/2) \sum_{r=1}^{n} (T_{or} - O_{or})^2$$

This error function is for one training pattern. If we use the same technique for all the training pattern, we get

$$E\,(V, W) = \sum_{r=1}^{nset} E_j\,(V, W, I)$$

where **E** is error function depends on **m (1 + n)** weights of **[W]** and **[V]**.

All that is stated is an **optimization** problem solving, where the objective or cost function is usually defined to be maximized or minimized with respect to a set of parameters. In this case, the network parameters that optimize the error function **E** over the '**nset**' of pattern sets **[I** $_{\text{nset}}$ **, t** $_{\text{nset}}$ **]** are synaptic weight values **[V]** and **[W]** whose sizes are

$$\underset{\ell \times m}{[\,\mathbf{V}\,]} \quad \text{and} \quad \underset{m \times n}{[\,\mathbf{W}\,]}$$

16

- **Back-Propagation Algorithm**

The benefits of hidden layer neurons have been explained. The hidden layer allows ANN to develop its own internal representation of input-output mapping. The complex internal representation capability allows the hierarchical network to learn any mapping and not just the linearly separable ones.

The step-by-step algorithm for the training of Back-propagation network is presented in next few slides. The network is the same , illustrated before,

has a three layer. The input layer is with **ℓ** nodes, the hidden layer with **m** nodes and the output layer with **n** nodes. An example for training a BPN with five training set have been shown for better understanding.

17

3.1 Algorithm for Training Network

The basic algorithm loop structure, and the step by step procedure of Back- propagation algorithm are illustrated in next few slides.

- **Basic algorithm loop structure**

```
Initialize the weights

Repeat

   For each training pattern

     "Train on that pattern"

  End

Until the error is acceptably low.
```

18

- **Back-Propagation Algorithm -** Step-by-step procedure

Step 1 :

Normalize the I/P and O/P with respect to their maximum values. For each training pair, assume that in normalized form there are

ℓ inputs given by $\{ I \}_I$ and

•x 1

outputs given by $\{ O \}_O$

n x 1

- **Step 2 :**

Assume that the number of neurons in the hidden layers lie

between **1 < m < 21**

19

2. **Step 3 :**

Let **[V]** represents the weights of synapses connecting input neuron and hidden neuron

Let **[W]** represents the weights of synapses connecting hidden neuron and output neuron

Initialize the weights to small random values usually from **-1 to +1;**

$$[V]^0 = [\text{random weights}]$$

$$[W]^0 = [\text{random weights}]$$

$$[\Delta V]^0 = [\Delta W]^0 = [0]$$

For general problems λ can be assumed as **1** and threshold value as **0**.

- **Step 4 :**

  For training data, we need to present one set of inputs and outputs. Present the pattern as inputs to the input layer $\{ I \}_I$.

  then by using linear activation function, the output of the input layer may be evaluated as

$$\underset{\ell \times 1}{\{ O \}_I} = \underset{\ell \times 1}{\{ I \}_I}$$

2. **Step 5 :**

   Compute the inputs to the hidden layers by multiplying corresponding weights of synapses as

$$\underset{m \times 1}{\{ I \}_H} = \underset{m \times \ell}{[V]^T} \underset{\ell \times 1}{\{ O \}_I}$$

- **Step 6 :**

  Let the hidden layer units, evaluate the output using the sigmoidal function as

$$\{ O \}_H = \frac{\left\{ \begin{matrix} - \\ - \\ 1 \end{matrix} \right\}}{(1 + e^{-(I_{Hi})})}$$

–

–

m x 1

21

- **Step 7 :**

  Compute the inputs to the output layers by multiplying corresponding weights of synapses as

$$\{ I \}_O = [W]^T \{ O \}_H$$

$$n \times 1 \qquad n \times m \qquad m \times 1$$

- **Step 8 :**

  Let the output layer units, evaluate the output using sigmoidal function as

$$\{ O \}_O = \left\{ \begin{matrix} - \\ - \\ \dfrac{1}{(1 + e^{-(I_{Oj})})} \\ - \\ - \end{matrix} \right\}$$

Note : This output is the network output

■ **Step 9 :**

Calculate the error using the difference between the network output and the desired output as for the **j** $^{\text{th}}$ training set as

$$E_P = \sqrt{\frac{\sum (T_j - O_{oj})^2}{n}}$$

A **Step 10 :**

Find a term **{ d }** as

$$\{ d \} = \left\{ \begin{array}{c} - \\ - \\ (T_k - O_{ok})\, O_{ok}\, (1 - O_{ok}) \\ - \\ - \end{array} \right\}_{n \times 1}$$

- **Step 11 :**

Find **[Y]** matrix as

$$[\,Y\,] = \{\,O\,\}_H \;\square\, d \,\square$$

m x n m x 1 1 x n

- **Step 12 :**

Find

$$[\,\Delta\; W\,]^{t+1} = \alpha \quad [\,\Delta\; W\,]^{t} + \eta\;[\,Y\,]$$

m x n m x n m x n

Step 13 :

Find

$$\{\,e\,\} = [\,W\,]\,\{\,d\,\}$$

m x 1 m x n n x 1

$$\{\,d^*\,\} = \quad e_i \left\{ \begin{matrix} - \\ - \\ (O_{Hi})\,(1 - O_{Hi}) \\ - \\ - \end{matrix} \right\}$$

m x 1 m x 1

Find **[X]** matrix as

= **{ I }$_I$** □ **d***

[X] = **{ O }$_I$** □ **d*** □ □

1 x m ℓ x 11 x m ℓ x 11 x m

24

- **Step 14 :**

Find
$$\underset{1 \times m}{[\Delta V]^{t+1}} = \alpha \underset{1 \times m}{[\Delta V]^{t}} + \eta \underset{1 \times m}{[X]}$$

1 **Step 15 :**

Find
$$[V]^{t+1} = [V]^{t} + [\Delta V]^{t+1}$$

$$[W]^{t+1} = [W]^{t} + [\Delta W]^{t+1}$$

f **Step 16 :**

Find error rate as

$$\textbf{error rate} = \frac{\Sigma E_p}{\textbf{nset}}$$

- **Step 17 :**

Repeat steps 4 to 16 until the convergence in the error rate is less than the tolerance value

- **End of Algorithm**

Note : The implementation of this algorithm, step-by-step 1 to 17, assuming one example for training BackProp Network is illustrated in the next section.

25

3.2 Example : Training Back-Prop Network

- **Problem :**

Consider a typical problem where there are 5 training sets.

Table : Training sets

S. No.	Input		Output
	I_1	I_2	O
1	0.4	-0.7	0.1
2	0.3	-0.5	0.05
3	0.6	0.1	0.3
4	0.2	0.4	0.25
5	0.1	-0.2	0.12

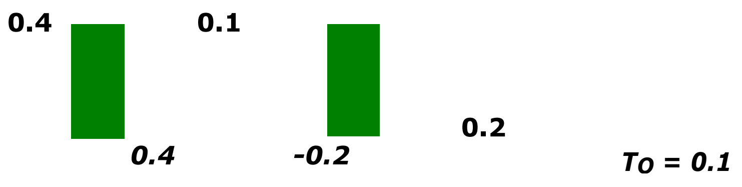
In this problem,

- there are two inputs and one output.

▪ the values lie between **-1** and **+1** i.e., no need to normalize the values.

- assume two neurons in the hidden layers.

• the NN architecture is shown in the Fig. below.



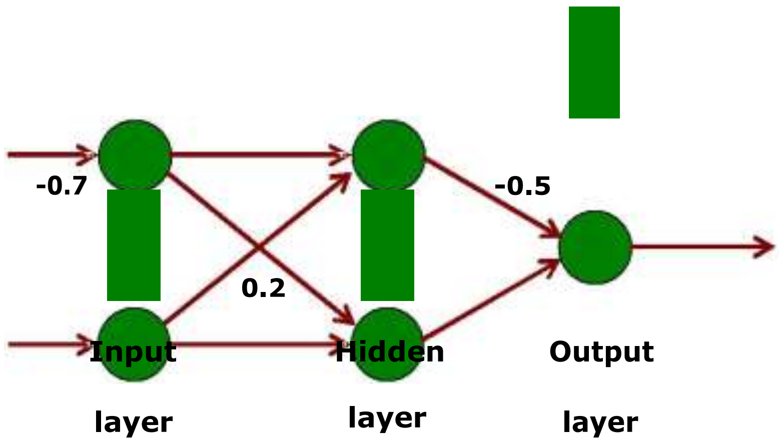


Fig. Multi layer feed forward neural network (MFNN) architecture

with data of the first training set

The solution to problem are stated step-by-step in the subsequent slides.

26

- **Step 1 :** Input the first training set data *(ref eq. of step 1)*

$$\underset{\ell \times 1}{\{ O \}_I} = \underset{\ell \times 1}{\{ I \}_I} = \underset{2 \times 1}{\begin{Bmatrix} 0.4 \\ -0.7 \end{Bmatrix}}$$

from training set s.no 1

- **Step 2 :** Initialize the weights as *(ref eq. of step 3 & Fig)*

$$[V]^0 = \underset{2x2}{\begin{bmatrix} 0.1 & 0.4 \\ -0.2 & 0.2 \end{bmatrix}} \; ; \quad [W]^0 = \underset{2 \times 1}{\begin{bmatrix} 0.2 \\ -0.5 \end{bmatrix}}$$

from fig initialization *from fig initialization*

- **Step 3 :** Find $\{ I \}_H = [V]^T \{ O \}_I$ as *(ref eq. of step 5)*

$$\{ I \}_H = \begin{Bmatrix} 0.1 & -0.2 \\ -0.4 & 0.2 \end{Bmatrix} \begin{Bmatrix} 0.4 \\ -0.7 \end{Bmatrix} = \begin{Bmatrix} 0.18 \\ 0.02 \end{Bmatrix}$$

Values from step 1 & 2

- **Step 4 :** *(ref eq. of step 6)*

$$\{ O \}_H = \left\{ \begin{array}{c} \dfrac{1}{(1 + e^{-(0.18)})} \\ \\ \dfrac{1}{(1 + e^{-(0.02)})} \end{array} \right\} = \left\{ \begin{array}{c} 0.5448 \\ \\ 0.505 \end{array} \right\}$$

Values from step 3 values

28

- **Step 5 :** *(ref eq. of step 7)*

$$\{ I \}_O = [W]^T \{ O \}_H = \begin{pmatrix} 0.2 & -0.5 \end{pmatrix} \begin{pmatrix} 0.5448 \\ 0.505 \end{pmatrix} = -\,0.14354$$

Values from step 2 , from step 4

- **Step 6 :** *(ref eq. of step 8)*

$$\{ O \}_O = \frac{1}{\left(1 + e^{-(0.14354)} \right)} = 0.4642$$

Values from step 5

- **Step 7 :** *(ref eq. of step 9)*

$$\text{Error} = (T_O - O_{O1})^2 = (0.1 - 0.4642)^2 = 0.13264$$

table first training set o/p *from step 6*

29

- **Step 8 :** *(ref eq. of step 10)*

  - **= $(T_O - O_{O1})(O_{O1})(1 - O_{O1})$**

    = (0.1 – 0.4642) (0.4642) (0.5358) = – 0.09058

Training o/p *all from step 6*

(ref eq. of step 11)

$$[Y] = \{O\}_H (d) = \begin{Bmatrix} 0.5448 \\ 0.505 \end{Bmatrix} (-0.09058) = \begin{Bmatrix} -0.0493 \\ -0.0457 \end{Bmatrix}$$

from values at step 4 *from values at step 8 above*

- **Step 9 :** *(ref eq. of step 12)*

$[\Delta W]^1 = \alpha [\Delta W]^0 + \eta [Y]$ assume η **=0.6**

$$= \begin{Bmatrix} -0.02958 \\ -0.02742 \end{Bmatrix}$$

from values at step 2 &step 8 above

- **Step 10 :** *(ref eq. of step 13)*

$$\{e\} = [W]\{d\} = \begin{Bmatrix} 0.2 \\ -0.5 \end{Bmatrix} (-0.09058) = \begin{Bmatrix} -0.018116 \\ -0.04529 \end{Bmatrix}$$

from values at step 8 above

from values at step 2

30

- **Step 11 :** *(ref eq. of step 13)*

$$\{ d^* \} = \left\{ \begin{array}{c} (-0.018116)\ (0.5448)\ (1 - 0.5448) \\ (0.04529)\ (0.505)\ (1 - 0.505) \end{array} \right\} = \left\{ \begin{array}{c} -0.00449 \\ -0.01132 \end{array} \right\}$$

from values at step 10 at step 4 at step 8

- **Step 12 :** *(ref eq. of step 13)*

$$[X] = \{ O \}_I \ (d^*) = \left\{ \begin{array}{c} 0.4 \\ -0.7 \end{array} \right\} (-0.00449 \quad 0.01132)$$

from values at step 1 *from values at step 11 above*

$$= \left\{ \begin{array}{cc} -0.001796 & 0.004528 \\ 0.003143 & -0.007924 \end{array} \right\}$$

- **Step 13 :** *(ref eq. of step 14)*

$$[\Delta V]^1 = \alpha \ [\Delta V]^0 + \eta \ [X] = \left\{ \begin{array}{cc} -0.001077 & 0.002716 \\ 0.001885 & -0.004754 \end{array} \right\}$$

from values at step 2 & step 8 above

31

- **Step 14 :** *(ref eq. of step 15)*

$$[V]^1 = \underbrace{\begin{bmatrix} 0.1 & 0.4 \\ -0.2 & 0.2 \end{bmatrix}}_{\text{from values at step 2}} + \underbrace{\begin{bmatrix} -0.001077 & 0.002716 \\ 0.001885 & -0.004754 \end{bmatrix}}_{\text{from values at step 13}}$$

$$= \begin{bmatrix} -0.0989 & 0.04027 \\ 0.1981 & -0.19524 \end{bmatrix}$$

$$[W]^1 = \underbrace{\begin{bmatrix} 0.2 \\ -0.5 \end{bmatrix}}_{\text{from values at step 2,}} + \underbrace{\begin{bmatrix} -0.02958 \\ -0.02742 \end{bmatrix}}_{\text{from values at step 9}} = \begin{bmatrix} 0.17042 \\ -0.52742 \end{bmatrix}$$

- **Step 15 :**

  With the updated weights **[V]** and **[W]** , error is calculated again and next training set is taken and the error will then get adjusted.

- **Step 16 :**

  Iterations are carried out till we get the error less than the tolerance.

- **Step 17 :**

  Once the weights are adjusted the network is ready for inferencing new objects .

Associative Memory

What is Associative Memory ?

- An associative memory is a content-addressable structure that maps a set of input patterns to a set of output patterns.

— A content-addressable structure 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.

— There are two types of associative memory : 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.

- Neural networks are used to implement these associative memory models called NAM (Neural associative memory).

03

1. Associative Memory

An associative memory is a content-addressable structure that maps a set of input patterns to a set of output patterns. A content-addressable

structure refers to a memory organization where the memory is accessed by its content as opposed to an explicit address in the traditional computer

memory system. The associative memory are of two types : auto-associative and hetero-associative.

ƒ An **auto-associative memory** retrieves a previously stored pattern that most closely resembles the current pattern.

ƒ In **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.

1.1 Description of Associative Memory

An associative memory is a content-addressable structure that allows, the recall of data, based on the degree of similarity between the input pattern and the patterns stored in memory.

- **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.

- Associative memory is a system that associates two patterns **(X, Y)** such that when one is encountered, the other can be recalled. The associative memory are of two types : auto-associative memory and hetero-associative memory.

  Auto-associative memory

  Consider, **y[1], y[2], y[3], ... y[M],** be the number of stored pattern vectors and let **y(m)** be the components of these vectors, representing features extracted from the patterns. The auto-associative memory will output a pattern vector **y(m)** when inputting a noisy or incomplete version of **y(m)**.

Hetero-associative memory

Here the memory function is more general. Consider, we have a number of key-response pairs **{c(1), y(1)}, {c(2), y(2)}, ... , {c(M), y(M)}**. The hetero-associative memory will output a pattern vector **y(m)** if a noisy or incomplete verson of the **c(m)** is given.

- Neural networks are used to implement associative memory models.

  - **Linear associater** is the simplest artificial neural associative memory.

  - **Hopfield model** and **Bidirectional Associative Memory (BAM)** are the other popular ANN models used as associative memories.

These models follow different neural network architectures to memorize information.

1.2 Working of Associative Memory

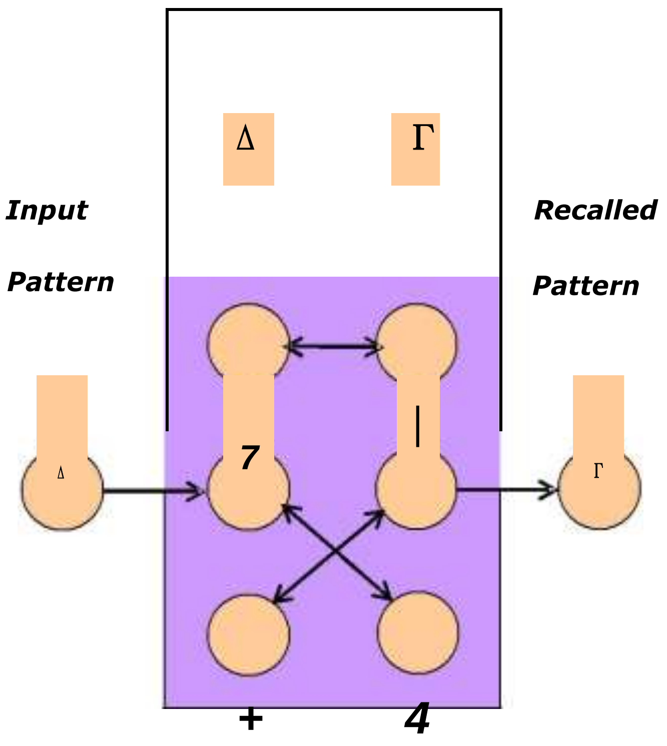
■ **Example**

An associative memory is a storehouse of associated patterns which are encoded in some form.

– When the storehouse is triggered or excited with a pattern, then the associated pattern pair is recalled or appears at the output.

– The input could be an exact or distorted or partial representation of a stored pattern.

Fig below illustrates the working of an associated memory.



The associated pattern pairs

(Δ , Γ), (| , +), (7 , 4).

The association is represented by the symbol

The associated pattern pairs are stored the memory.

.

Fig. Working of an associated memory

W e
h n

the memory is triggered with an input pattern say

Δ then the associated pattern Γ is retrieved automatically.

1.3 Associative Memory - Classes

As stated before, there are two classes of associative memory:

- auto-associative and
- hetero-associative memory.

An auto-associative memory, also known as auto-associative correlator, is used to retrieve a previously stored pattern that most closely resembles the current pattern;

A hetero-associative memory, also known as hetero-associative correlator, is used to retrieve pattern in general, different from the input pattern not only in content but possibly also different in type and format.

Examples

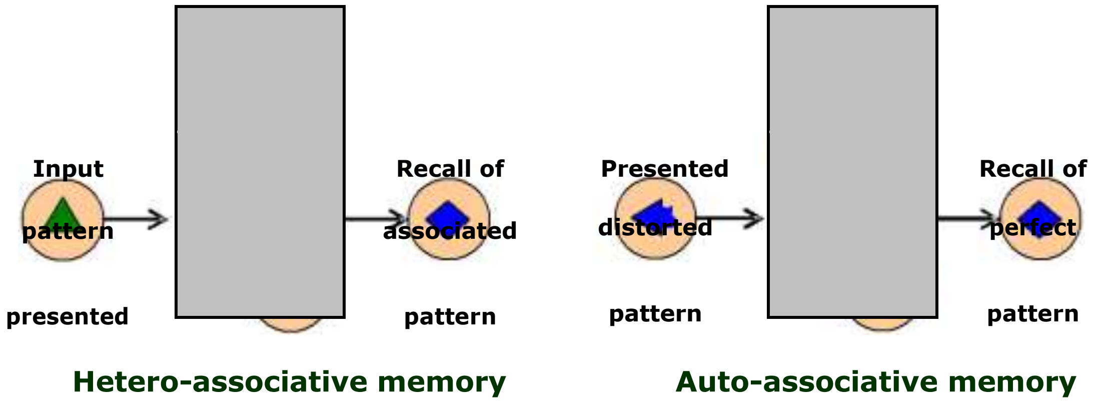


Fig. Hetero and Auto Associative memory Correlators

08

1.4 Related Terms

Here explained : Encoding or memorization, Retrieval or recollection, Errors and Noise, Memory capacity and Content-addressability.

- **Encoding or memorization**

Building an associative memory means, constructing a connection weight matrix **W** such that when an input pattern is presented, and the stored pattern associated with the input pattern is retrieved.

This process of constructing the connection weight matrix is called **encoding**. During encoding, for an associated pattern pair (X_k, Y_k) , the weight values of the correlation matrix W_k are computed as

$(w_{ij})_k = (x_i)_k \ (y_j)_k$, where

$(x_i)_k$ represents the i^{th} component of pattern X_k , and

$(y_j)_k$ represents the j^{th} component of pattern Y_k

for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

Constructing of the connection weight matrix **W** is accomplished by summing up the individual correlation matrices W_k , i.e.,

$$W = \alpha \sum_{k=1}^{p} W_k$$ where

α is the proportionality or normalizing constant.

- **Retrieval or recollection**

  After memorization, the process of retrieving a stored pattern, given an input pattern, is called **decoding.**

  Given an input pattern **X**, the decoding or recollection is accomplished by:

  first compute the net **input** to the output units using

$$\textbf{input}_j = \sum_{j=1}^{m} \mathbf{x_i}\ \mathbf{w}_{ij}$$

where **input** $_j$ is weighted sum of the input or activation value of node **j** , for **j = 1, 2, ..., n.**

then determine the units **output** using a bipolar output function:

$$\mathbf{Y}_j = \begin{cases} \mathbf{+1} \text{ if } \textbf{input } \mathbf{j} \geq \boldsymbol{\theta}_j \\ \mathbf{-1} \quad \text{other wise} \end{cases}$$

where $\boldsymbol{\theta}_{\mathbf{j}}$ is the threshold value of output neuron **j** .

- **Errors and noise**

  The input pattern may contain errors and noise, or may be an incomplete version of some previously encoded pattern.

  When a corrupted input pattern is presented, the network will retrieve the stored pattern that is closest to actual input pattern.

  The presence of noise or errors results only in a mere decrease rather than total degradation in the performance of the network.

  Thus, associative memories are robust and fault tolerant because of many processing elements performing highly parallel and distributed computations.

- **Performance Measures**

  The memory capacity and content-addressability are the measures of associative memory performance for correct retrieval. These two performance measures are related to each other.

  Memory capacity refers to the maximum number of associated pattern pairs that can be stored and correctly retrieved.

  Content-addressability is the ability of the network to retrieve the correct stored pattern.

  If input patterns are mutually orthogonal - perfect retrieval is possible.

  If the stored input patterns are not mutually orthogonal - non-perfect retrieval can happen due to crosstalk among the patterns.

2. Associative Memory Models

An associative memory is a system which stores mappings of specific input representations to specific output representations.

- An associative memory "associates" two patterns such that when one is encountered, the other can be reliably recalled.

- Most associative memory implementations are realized as connectionist networks.

The simplest associative memory model is **Linear associator**, which is a feed-forward type of network. It has very low memory capacity and therefore not much used.

The popular models are **Hopfield Model** and **Bi-directional Associative Memory (BAM) model**.

The Network Architecture of these models are presented in this section.

13

2.1 Associative Memory Models

The simplest and among the first studied associative memory models is **Linear associator.** It is a feed-forward type of network where the output is produced in a single feed-forward computation. It can be used as an auto-associator as well as a hetero-associator, but it possesses a very low memory capacity and therefore not much used.

The popular associative memory models are **Hopfield Model** and **Bi-directional Associative Memory (BAM) model.**

- The Hopfield model is an auto-associative memory, proposed by John Hopfield in 1982. It is an ensemble of simple processing units that have a fairly complex collective computational abilities and behavior. The Hopfield model computes its output recursively in time until the system becomes stable. Hopfield networks are designed using bipolar units and a learning procedure.

- The Bi-directional associative memory (BAM) model is similar to linear associator, but the connections are bi-directional and therefore allows forward and backward flow of information between the layers. The BAM model can perform both auto-associative and hetero-associative recall of stored information.

The network architecture of these three models are described in the next few slides.

2.2 Network Architectures of AM Models

The neural associative memory models follow different neural network architectures to memorize information. The network architectures

are either single layer or two layers .

- The **Linear associator model**, is a feed forward type network, consists, two layers of processing units, one serving as the input layer while the other as the output layer.

- The **Hopfield model**, is a single layer of processing elements where each unit is connected to every other unit in the network other than itself.

- The **Bi-directional associative memory (BAM) model** is similar to

  that of linear associator but the connections are bidirectional.

In this section, the neural network architectures of these models and the construction of the corresponding connection weight matrix **W** of the associative memory are illustrated.

3. Linear Associator Model (two layers)

It is a feed-forward type network where the output is produced in a single feed-forward computation. The model consists of two layers of processing units, one serving as the input layer while the other as the output layer. The inputs are directly connected to the outputs, via a series of weights. The links carrying weights connect every input to every output. The sum of the products of the weights and the inputs is calculated in each neuron node. The network architecture of the linear associator is as shown below.

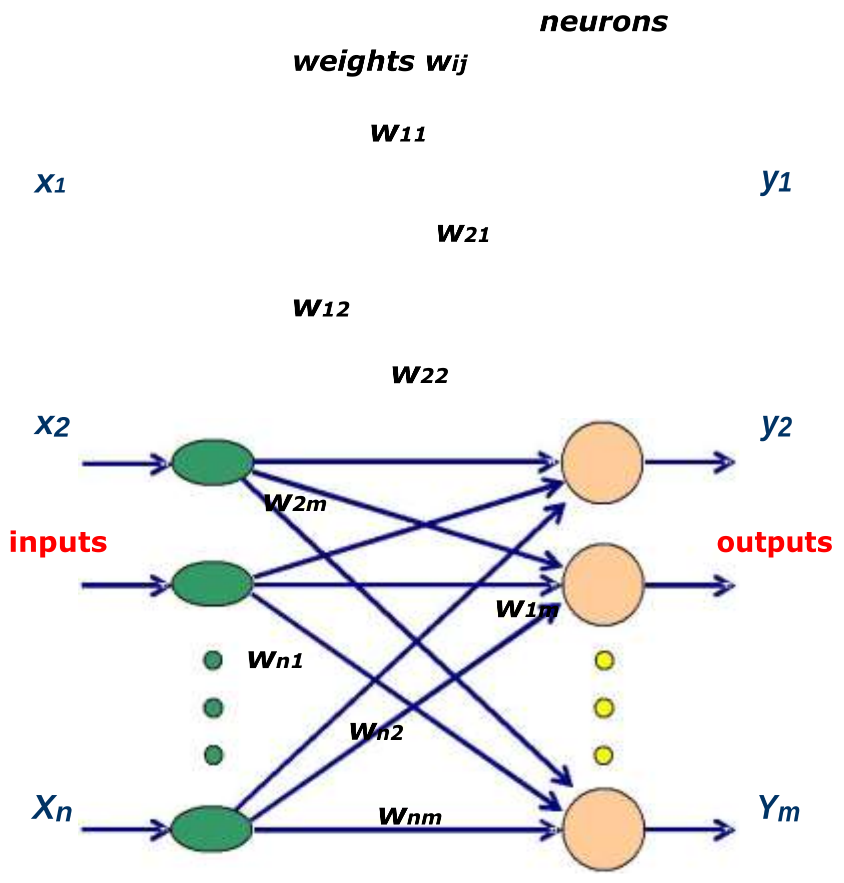


Fig. Linear associator model

- all **n** input units are connected to all **m** output units via connection

  weight matrix **W** = $[W_{ij}]_{n \times m}$ where W_{ij} denotes the strength of the unidirectional connection from the **i** th input unit to the **j** th output unit.

- the connection weight matrix stores the **p** different associated

  pattern pairs $\{(X_k, Y_k) \mid k = 1, 2, ..., p\}$.

  - building an associative memory is constructing the connection weight matrix **W** such that when an input pattern is presented,

  then the stored pattern associated with the input pattern is retrieved.

- **Encoding** : The process of constructing the connection weight matrix is called encoding. During encoding the weight values of correlation matrix $\mathbf{W_k}$ for an associated pattern pair $(\mathbf{X_k}, \mathbf{Y_k})$ are computed as:

  $(w_{ij})_k = (x_i)_k \ (y_j)_k$ where

  $(x_i)_k$ is the i th component of pattern $\mathbf{X_k}$ for $i = 1, 2, \ldots, m$, and

  $(y_j)_k$ is the j th component of pattern $\mathbf{Y_k}$ for $j = 1, 2, \ldots, n$.

- **Weight matrix** : Construction of weight matrix **W** is accomplished by summing those individual correlation matrices $\mathbf{W_k}$, ie, $\mathbf{W} = \alpha \sum_{k=1}^{p} \mathbf{W_k}$ where α is the constant of proportionality, for normalizing, usually set to **1/p** to store **p** different associated pattern pairs.

- **Decoding** : After memorization, the network can be used for retrieval;

  the process of retrieving a stored pattern, is called decoding; given an input pattern **X**, the decoding or retrieving is accomplished by computing, first the net *Input* as $\textbf{input}_j = \sum_{j=1}^{m} x_i \, w_{ij}$ where

  $\textbf{input}_j$ stands for the weighted sum of the input or activation value of node j , for $j = 1, 2, \ldots, n$. and x_i is the i th component of pattern $\mathbf{X_k}$, and then determine the units *Output* using a bipolar output function:

$$Y_j = \begin{cases} +1 & \text{if input}_j \geq \theta_j \\ -1 & \text{other wise} \end{cases}$$

where **θ_j** is the threshold value of output neuron **j** .

Note: The output units behave like linear threshold units; that compute a weighted sum of the input and produces a **-1** or **+1** depending whether the weighted sum is below or above a certain threshold value.

- **Performance** : The input pattern may contain errors and noise, or an incomplete version of some previously encoded pattern. When such corrupt input pattern is presented, the network will retrieve the stored pattern that is closest to actual input pattern. Therefore, the linear associator is robust and fault tolerant. The presence of noise or error results in a mere decrease rather than total degradation in the performance of the network.

2 Auto-associative Memory Model - Hopfield model (single layer)

Auto-associative memory means patterns rather than associated pattern pairs, are stored in memory. Hopfield model is one-layer

unidirectional auto-associative memory.

- the model consists, a single layer of processing elements where each

  unit is connected to every other unit in the network but not to itself.

- connection weight between or from neuron **j** to **i** is given by a

  number w_{ij}. The collection of all such numbers are represented

  by the weight matrix **W** which is square and symmetric, ie, $w_{ij} = w_{ji}$

  for $i, j = 1, 2, \ldots, m$.

- each unit has an external input **I** which leads to a modification

  in the computation of the net input to the units as

  $$\text{input}_j = \sum_{i=1}^{m} x_i w_{ij} + I_j \quad \text{for } j = 1, 2, \ldots, m.$$

  and x_i is the **i** th component of pattern X_k

- each unit acts as both input and output unit. Like linear associator,

  a single associated pattern pair is stored by computing the weight

  matrix as $W_k = X_k^T Y_k$ where $X_K = Y_K$

- **Weight Matrix** : Construction of weight matrix **W** is accomplished by

  summing those individual correlation matrices, ie, $W = \alpha \sum_p W_k$ where

$k=1$

α is the constant of proportionality, for normalizing, usually set to **1/p** to store **p** different associated pattern pairs. Since the Hopfield model is an auto-associative memory model, it is the patterns rather than associated pattern pairs, are stored in memory.

– **Decoding** : After memorization, the network can be used for retrieval; the process of retrieving a stored pattern, is called decoding; given an

input pattern **X**, the decoding or retrieving *m*is accomplished by computing, first the net *Input* as $\text{input}_j = \sum_{j=1}^{m} x_i \, w_{ij}$ where input_j

stands for the weighted sum of the input or activation value of node **j** , for **j = 1, 2, ..., n.** and x_i is the **i** th component of pattern X_k , and then determine the units *Output* using a bipolar output function:

$$Y_j = \begin{cases} +1 & \text{if input}_j \geq \theta_j \\ -1 & \text{other wise} \end{cases}$$

where θ_j is the threshold value of output neuron **j** .

Note: The output units behave like linear threshold units; that compute a weighted sum of the input and produces a **-1** or **+1** depending whether the weighted sum is below or above a certain threshold value.

Decoding in the Hopfield model is achieved by a collective and recursive relaxation search for a stored pattern given an initial stimulus pattern. Given an input pattern **X**, decoding is accomplished by computing the net input to the units and determining the output of those units using the output function to produce the pattern **X'**. The pattern **X'** is then fed back to the units as an input pattern to produce the pattern **X''**. The pattern **X''** is again fed back to the units to produce the pattern **X'''**. The process is repeated until the network stabilizes on a stored pattern where further computations do not change the output of the units.

In the next section, the working of an auto-correlator : how to store patterns, recall a pattern from the stored patterns and how to recognize a noisy pattern are explained.

■ **Bidirectional Associative Memory** (two-layer)

Kosko (1988) extended the Hopfield model, which is single layer, by incorporating an additional layer to perform recurrent auto-associations as well as hetero-associations on the stored memories. The network structure of the bidirectional associative memory model is similar to that of the linear associator but the connections are bidirectional; i.e.,

$w_{ij} = w_{ji}$, for **$i = 1, 2, \ldots ,$ n** and **$j = 1, 2, \ldots ,$ m.**

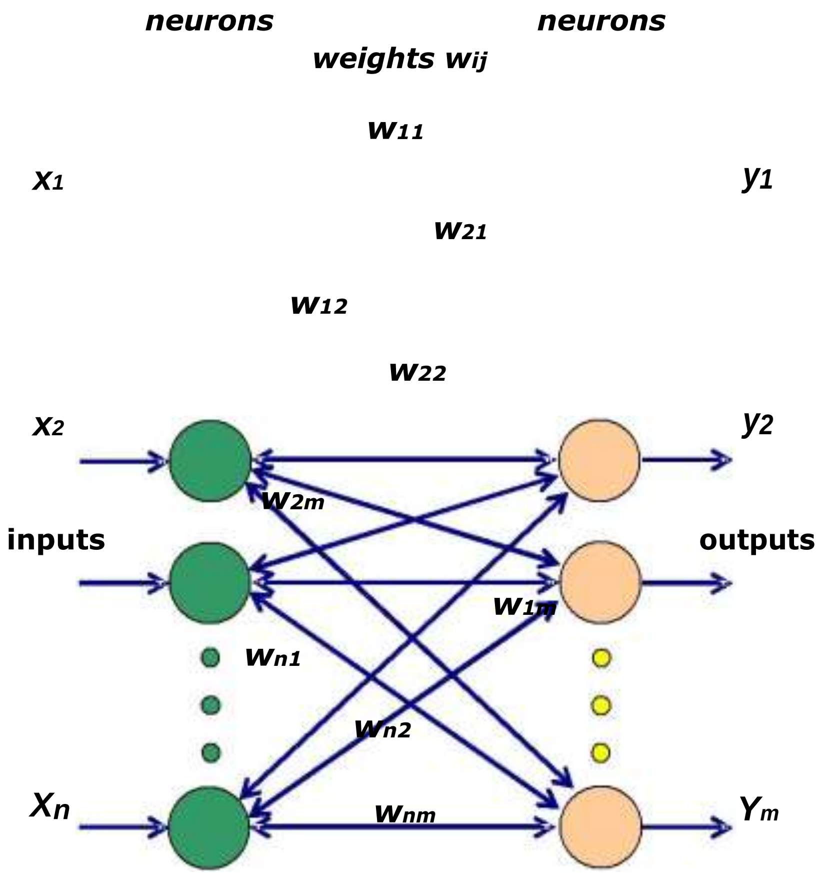


Fig. Bidirectional Associative Memory model

- In the bidirectional associative memory, a single associated pattern pair is stored by computing the weight matrix as $\mathbf{W_k} = X_k^T \mathbf{Y_k}$.

- the construction of the connection weight matrix **W**, to store **p** different associated pattern pairs simultaneously, is accomplished by summing up the individual correlation matrices $\mathbf{W_k}$,

i.e., $\mathbf{W} = \alpha \sum_{k=1}^{p} \mathbf{W_k}$

where α is the proportionality or normalizing constant.

- **Auto-associative Memory (auto-correlators)**

In the previous section, the structure of the Hopfield model has been explained. It is an auto-associative memory model which means patterns, rather than associated pattern pairs, are stored in memory. In this section, the working of an auto-associative memory (auto-correlator) is illustrated using some examples.

Working of an auto-correlator :

– how to store the patterns,

– how to retrieve / recall a pattern from the stored patterns, – and how to recognize a noisy pattern

21

- **How to Store Patterns : Example**

Consider the three bipolar patterns A_1 , A_2, A_3 to be stored as an auto-correlator.

$$A_1 = (-1, 1 , -1 , \ 1)$$

$$A_2 = (1, 1 , \quad 1 , -1)$$

$$A_3 = (-1, -1 , -1 , \ 1)$$

Note that the outer product of two vectors **U** and **V** is

$$U \otimes V = U^T V = \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} \begin{bmatrix} V_1 & V_2 & V_3 \end{bmatrix} = \begin{bmatrix} U_1V_1 & U_1V_2 & U_1V_3 \\ U_2V_1 & U_2V_2 & U_2V_3 \\ U_3V_1 & U_3V_2 & U_3V_3 \\ U_4V_1 & U_4V_2 & U_4V_3 \end{bmatrix}$$

Thus, the outer products of each of these three A_1 , A_2, A_3 bipolar patterns are

$$[A_1]^T_{4x1} \ [A_1]_{1x4} = \begin{Bmatrix} 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \end{Bmatrix}$$

j

$$[A_2]^{T}_{4x1}\;[A_2]_{1x4} \;=\; \downarrow \left\{ \begin{array}{cccc} 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{array} \right\}$$

e

$$[A_3]^{T}_{4x1}\;[A_3]_{1x4} \;=\; \downarrow \left\{ \begin{array}{cccc} & & j & \\ 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{array} \right\}$$

•

Therefore the connection matrix is

$$T = [t_{ij}] = \sum_{i=1}^{3} [A_i]^{T}_{4x1}\;[A_i]_{1x4} = \begin{array}{c} \\ \\ \\ i\downarrow \end{array} \left\{ \begin{array}{cccc} & & j & \\ 3 & 1 & 3 & -3 \\ 1 & 3 & 1 & -1 \\ 3 & 1 & 3 & -3 \\ -3 & -1 & -3 & 3 \end{array} \right\}$$

This is how the patterns are stored .

22

- **Retrieve a Pattern from the Stored Patterns** (ref. previous slide)

The previous slide shows the connection matrix **T** of the three bipolar patterns **A1 , A2, A3** stored as

$$\mathbf{T} = [t_{ij}] = \sum_{i=1}^{3} [A_i]^T_{4x1} [A_i]_{1x4} = \begin{array}{c|cccc} & & i & & \\ \hline & 3 & 1 & 3 & -3 \\ & 1 & 3 & 1 & -1 \\ & 3 & 1 & 3 & -3 \\ j \downarrow & -3 & -1 & -3 & 3 \end{array}$$

and one of the three stored pattern is $\mathbf{A_2} = \left\{ (1, \; 1 , \; 1 , -1) \right\}$ a_i

– Retrieve or recall of this pattern **A2** from the three stored patterns.

– The recall equation is

$$a_{new\,j} = f\,(a_i\, t_{ij} \,, a_j^{old}) \text{ for } \forall\, j = 1 , 2 , . . . , p$$

Computation for the recall equation **A2** yields $\alpha = \Sigma\, a_i\, t_{ij}$ and then find β

$i = \longrightarrow$

	1	2	3	4	α	β
$\alpha = \Sigma\, a_i\, t_{i,j=1}$	1x3	+ 1x1	+ 1x3	+ -1x-3	= 10	1
$\alpha = \Sigma\, a_i\, t_{i,j=2}$	1x1	+ 1x3	+ 1x1	+ -1x-1	= 6	1
$\alpha = \Sigma\, a_i\, t_{i,j=3}$						
$\alpha = \Sigma\, a_i\, t_{i,j=4}$						

$$\downarrow \left\{ \begin{array}{l} 1x3 \;+\; 1x1 \;+\; 1x3 \;+\; -1x-3 \\ 1x-3 \;+\; 1x-1 + 1x-3 \;+\; -1x3 \end{array} \right\} \begin{array}{ll} = 10 & 1 \\ = -1 & -1 \end{array}$$

Therefore $a\ new_j = f\,(a_i\ t_{ij}\ , a_j^{old})$ **for** $\forall$ **j = 1 , 2 , . . . , p** is $f\,(\alpha\ , \beta\ )$

$a_{new1} = f\ (10\ , 1)$

$a_{new2} = f\ (6\ , 1)$

$a_{new3} = f\ (10\ , 1)$

$a^{new}{}_4 = f\ (-1\ , -1)$

The values of β is the vector pattern **(1, 1 , 1 , -1)** which is **A2 .**

This is how to retrieve or recall a pattern from the stored patterns.

Similarly, retrieval of vector pattern **A3** as

$$(\ a^{new}_1\ ,\ a^{new}_2\ ,\ a^{new}_3\ ,\ a^{new}_4\ ,\ ) = (\ -1, -1\ , -1\ , 1\ ) = A_3$$

23

- **Recognition of Noisy Patterns** (ref. previous slide)

Consider a vector **A'** = **(1, 1 , 1 , 1)** which is a noisy presentation of one among the stored patterns.

– find the proximity of the noisy vector to the stored patterns using Hamming distance measure.

– note that the Hamming distance (HD) of a vector **X** from **Y**, where **X = (x1 , x2 , . . . , xn)** and **Y = (y1, y2 , . . . , yn)** is given by **HD (x , y) =** $\sum_{i=1}^{m} | (x_i - y_i) |$

The HDs of **A'** from each of the stored patterns A_1 , A_2, A_3 are

$$HD(A', A_1) = \Sigma\ |(x_1 - y_1)|,\ |(x2 - y2)|,\ |(x_3 - y_3)|,\ |(x_4 - y_4)|$$

$$= \Sigma\ |(1 - (-1))|,\ |(1 - 1)|,\ |(1 - (-1))|,\ |(1 - 1)|$$

$$= 4$$

$$HD(A', A_2) = 2$$

$$HD(A', A_3) = 6$$

Therefore the vector **A'** is closest to A_2 and so resembles it.

In other words the vector **A'** is a noisy version of vector A_2.

Computation of recall equation using vector **A'** yields :

$i = \longrightarrow$

$\alpha = \Sigma$ *ai* **t** *i , j=1* $\alpha = \Sigma$ *ai* **t** *i , j=2* $\alpha = \Sigma$ *ai* **t** *i , j=3* $\alpha = \Sigma$ *ai* **t** *i , j=4*

1	2	3	4	α	β
1x3	+ 1x1	+ 1x3	+ 1x-3	= 4	1
1x1	+ 1x3	+ 1x1	+ 1x-1	= 4	1
1x3	+ 1x1	+ 1x3	+ 1x-3	= 4	1
1x-3	+ 1x-1	+ 1x-3	+ 1x3	= -4	-1

Therefore $a^{new}{}_j = f\,(a_i\ t_{ij}\ , a_j^{old})$ for $\forall$ j = 1 , 2 , . . . , p is $f\,(\alpha\ , \beta\ )$

$a_{new1} = f\,(4\ ,\ 1)$

$a_{new2} = f\,(4\ ,\ 1)$

$a_{new3} = f\,(4\ ,\ 1)$

$a^{new}{}_4 = f\,(-4 , -1)$

The values of β is the vector pattern **(1, 1 , 1 , -1)** which is **A2 .**

Note : In presence of noise or in case of partial representation of vectors, an autocorrelator results in the refinement of the pattern or removal of noise to retrieve the closest matching stored pattern.

24

4. Bidirectional Hetero-associative Memory

The Hopfield one-layer unidirectional auto-associators have been discussed in previous section. Kosko (1987) extended this network to two-layer bidirectional structure called Bidirectional Associative Memory (BAM) which can achieve hetero-association. The important performance attributes of the BAM is its ability to recall stored pairs particularly in the presence of noise.

Definition : If the associated pattern pairs **(X, Y)** are different and if the model recalls a pattern **Y** given a pattern **X** or vice-versa, then it is termed as hetero-associative memory.

This section illustrates the bidirectional associative memory :

- Operations (retrieval, addition and deletion) ,
- Energy Function (Kosko's correlation matrix, incorrect recall of pattern),
- Multiple training encoding strategy (Wang's generalized correlation matrix).

4.1 Bidirectional Associative Memory (BAM) Operations

BAM is a two-layer nonlinear neural network.

Denote one layer as field **A** with elements A_i and the other layer as field **B** with elements B_i.

The basic coding procedure of the discrete BAM is as follows.

Consider **N** training pairs $\{ (A_1 , B_1) , (A_2 , B_2), \ldots , (A_i , B_i), \ldots (A_N , B_N) \}$

where $A_i = (a_{i1} , a_{i2} , \ldots , a_{in})$ and $B_i = (b_{i1} , b_{i2} , \ldots , b_{ip})$ and

a_{ij} , b_{ij} are either in **ON** or **OFF** state.

- in binary mode , **ON = 1** and **OFF = 0** and

  in bipolar mode, **ON = 1** and **OFF = -1**

- the original correlation matrix of the BAM is $M_0 = \sum_{i=1}^{N} [X_i^T] [Y_i]$

  where $X_i = (x_{i1} , x_{i2} , \ldots , x_{in})$ and $Y_i = (y_{i1} , y_{i2} , \ldots , y_{ip})$

  and $x_{ij}(y_{ij})$ is the bipolar form of $a_{ij}(b_{ij})$

The energy function **E** for the pair (α , β) and correlation matrix **M** is

$E = - \alpha M \beta^T$

With this background, the decoding processes, means the operations to **retrieve** nearest pattern pairs, and the **addition** and **deletion** of the pattern pairs are illustrated in the next few slides.

- **Retrieve the Nearest of a Pattern Pair, given any pair**
  (ref : previous slide)

  Example

  Retrieve the nearest of **(A_i , B_i)** pattern pair, given any pair **(α , β)** .

  The methods and the equations for retrieve are :

  – start with an initial condition which is any given pattern pair **(α , β)**,

  – determine a finite sequence of pattern pairs **$(\alpha ' , \beta ')$** , **$(\alpha '' , \beta '')$ . . . .**

  until an equilibrium point **(α_f , β_f)** is reached, where

$$\beta ' = \Phi\ (\alpha\ M) \quad \text{and} \quad \alpha ' = \Phi\ (\beta '\ M^T)$$

$$\beta '' = \Phi\ (\alpha '\ M) \quad \text{and} \quad \alpha '' = \Phi\ (\beta ''\ M^T)$$

$$\Phi\ (F) = G = g_1 , g_2 , \ldots , g_r ,$$

$$F = (f_1 , f_2 , \ldots , f_r)$$

Mis correlation matrix

$$\left\{ \begin{cases} 1 & \text{if } f_i > 0 \\ 0 \text{ (binary)} & \end{cases} \right.$$

gi = **,** **f i < 0**

-1 (bipolar)

previous g i , **f i = 0**

Kosko has proved that this process will converge for any correlation matrix **M**.

27

■ Addition and Deletion of Pattern Pairs

Given a set of pattern pairs **(X_i , Y_i) , for i = 1 , 2, . . . , n** and a set of correlation matrix **M** :

– a new pair **(X' , Y')** can be added or

– an existing pair **(X_j , Y_j)** can be deleted from the memory model.

Addition : add a new pair **(X' , Y')** , to existing correlation matrix **M** , them the new correlation matrix **M_{new}** is given by

$$M_{new} = X_1^T Y_1 + X_1^T Y_1 + \dots + X_n^T Y_n + X'^T Y'$$

Deletion : subtract the matrix corresponding to an existing pair **(X_j , Y_j)** from the correlation matrix **M** , them the new correlation matrix **M_{new}** is given by

$$M_{new} = M - (X_j^T Y_j)$$

Note : The addition and deletion of information is similar to the functioning of the system as a human memory exhibiting learning and forgetfulness.

4.2 Energy Function for BAM

Note : A system that changes with time is a dynamic system. There are two types of dynamics in a neural network. During training phase it iteratively update weights and during production phase it asymptotically converges to the solution patterns. State is a collection of qualitative and qualitative items that characterize the system e.g., weights, data flows. The Energy function (or Lyapunov function) is a bounded function of the system state that decreases with time and the system solution is the minimum energy.

Let a pair **(A , B)** defines the state of a BAM.

- to store a pattern, the value of the energy function for that pattern

  has to occupy a minimum point in the energy landscape.

- also adding a new patterns must not destroy the previously

  stored patterns.

The stability of a BAM can be proved by identifying the energy function **E** with each state **(A , B) .**

- For auto-associative memory : the energy function is

  E(A) = - AM A^T

- For bidirecional hetero associative memory : the energy function is

  E(A, B) = - AM B_T ; for a particular case **A = B** , it corresponds to Hopfield auto-associative function.

We wish to retrieve the nearest of **(A_i , B_i)** pair, when any **(α , β)** pair is presented as initial condition to BAM. The neurons change

their states until a bidirectional stable state **(A_f , B_f)** is reached. Kosko has shown that such stable state is reached for any matrix **M** when it corresponds to local minimum of the energy function. Each cycle of

decoding lowers the energy **E** if the energy function for any point **(** α **,** β **)** is given by $E = \alpha M \beta^T$

If the energy $E = A_i M B_i^T$ evaluated using coordinates of the pair **(A_i , B_i)** does not constitute a local minimum, then the point cannot be recalled, even though one starts with $\alpha = A_i$. Thus Kosko's encoding method does not ensure that the stored pairs are at a local minimum.

29

- **Example : Kosko's BAM for Retrieval of Associated Pair**

  The working of Kosko's BAM for retrieval of associated pair.

  Start with X_3, and hope to retrieve the associated pair Y_3 .

  Consider **N = 3** pattern pairs (A_1, B_1), (A_2, B_2), (A_3, B_3) given by

  $A_1 = (\,1\ 0\ 0\ 0\ 0\ 1\,)$ $B_1 = (\,1\ 1\ 0\ 0\ 0\,)$

  $A_2 = (\,0\ 1\ 1\ 0\ 0\ 0\,)$ $B_2 = (\,1\ 0\ 1\ 0\ 0\,)$

  $A_3 = (\,0\ 0\ 1\ 0\ 1\ 1\,)$ $B_3 = (\,0\ 1\ 1\ 1\ 0\,)$

  Convert these three binary pattern to bipolar form replacing **0s** by **-1s**.

  $X_1 = (\,1\ -1\ -1\ -1\ -1\ 1\,)$ $Y_1 = (\,1\ 1\ -1\ -1\ -1\,)$

  $X_2 = (\,-1\ 1\ 1\ -1\ -1\ -1\,)$ $Y_2 = (\,1\ -1\ 1\ -1\ -1\,)$

  $X_3 = (\,-1\ -1\ 1\ -1\ 1\ 1\,)$ $Y_3 = (\,-1\ 1\ 1\ 1\ -1\,)$

  The correlation matrix **M** is calculated as 6x5 matrix

$$M = X_1^T\ Y_1 + X_2^T\ Y_2 + X_3^T\ Y_3 = \begin{matrix} 1 & 1 & -3 & -1 & 1 \\ 1 & -3 & 1 & -1 & 1 \\ -1 & -1 & 3 & 1 & -1 \\ -1 & -1 & -1 & 1 & 3 \\ -3 & 1 & 1 & 3 & 1 \\ -1 & 3 & -1 & 1 & -1 \end{matrix}$$

  Suppose we start with α = **X₃**, and we hope to retrieve the associated pair **Y₃** . The calculations for the retrieval of **Y₃** yield :

$$\alpha\ \mathbf{M} = (\,-1\ -1\ 1\ -1\ 1\ 1\,)\ (\,\mathbf{M}\,) = (\,-6\ 6\ 6\ 6\ -6\,)$$

$$\Phi\ (\alpha\ \mathbf{M}) = \beta\ ' = (\,-1\ 1\ 1\ 1\ -1\,)$$

$$\beta' M^T = (-5\ -5\ 5\ -3\ 7\ 5)$$

$$\Phi\ (\beta' M^T) = (-1\ -1\ 1\ -1\ 1\ 1) = \alpha'$$

$$\alpha' \mathbf{M} = (-1\ -1\ 1\ -1\ 1\ 1)\ \mathbf{M} = (-6\ 6\ 6\ 6\ -6)$$

$$.\ (\alpha' \mathbf{M}) = \beta'' = (-1\ 1\ 1\ 1\ 1\ -1)$$

$$= \beta'$$

This retrieved patern β' is same as $\mathbf{Y_3}$.

Hence, $(\alpha_f , \beta_f) = (\mathbf{X_3} , \mathbf{Y_3})$ is correctly recalled, a desired result .

30

- **Example : Incorrect Recall by Kosko's BAM**

  The Working of incorrect recall by Kosko's BAM.

  Start with **X_2**, and hope to retrieve the associated pair **Y_2** . Consider **N = 3** pattern pairs **(A_1 , B_1) , (A_2 , B_2) , (A_3 , B_3)** given by

  $A_1 = (1\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0)$ $\quad B_1 = (1\ 1\ 1\ 0\ 0\ 0\ 0\ 1\ 0)$

  $A_2 = (0\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 1)$ $\quad B_2 = (1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1)$

  $A_3 = (1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 1)$ $\quad B_3 = (0\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1)$

  Convert these three binary pattern to bipolar form replacing **0s** by **-1s**.

  $X_1 = (1\ -1\ -1\ 1\ 1\ 1\ -1\ -1\ -1)$ $\quad Y_1 = (1\ 1\ 1\ -1\ -1\ -1\ -1\ 1\ -1)$

  $X_2 = (-1\ 1\ 1\ 1\ -1\ -1\ 1\ 1\ 1)$ $\quad Y_2 = (1\ -1\ -1\ -1\ -1\ -1\ -1\ -1\ 1)$

  $X_3 = (1\ -1\ 1\ -1\ 1\ 1\ -1\ 1\ 1)$ $\quad Y_3 = (-1\ 1\ -1\ 1\ -1\ -1\ 1\ 0\ 1)$

  The correlation matrix **M** is calculated as 9 x 9 matrix

$$M = X_1^T\,Y_1 + X_2^T\,Y_2 + X_3^T\,Y_3$$

$$= \begin{matrix} -1 & 3 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \\ 1 & -3 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ -1 & -1 & -3 & 1 & -1 & -1 & 1 & -3 & 3 \\ 3 & -1 & 1 & -3 & -1 & -1 & -3 & 1 & -1 \\ -1 & 3 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \\ -1 & 3 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \\ 1 & -3 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ -1 & -1 & -3 & 1 & -1 & -1 & 1 & -3 & 3 \end{matrix}$$

$$\begin{pmatrix} -1 & -1 & -3 & 1 & -1 & -1 & 1 & -3 & 3 \end{pmatrix}$$

(Continued in next slide)

31

[Continued from previous slide]

Let the pair (**X2** , **Y2**) be recalled.

$$X_2 = (-1\ 1\ 1\ 1\ -1\ -1\ 1\ 1\ 1) \qquad Y_2 = (1\ -1\ -1\ -1\ -1\ -1\ -1\ -1\ 1)$$

Start with α **= X2**, and hope to retrieve the associated pair **Y2** .

The calculations for the retrieval of **Y2** yield :

$$\alpha\ M = (5\ -19\ -13\ -5\ 1\ 1\ -5\ -13\ 13)$$

$$\Phi\ (\alpha\ M) = (1\ -1\ -1\ -1\ 1\ 1\ -1\ -1\ 1) = \beta\ '$$

$$\beta\ ' M^T = (-11\ 11\ 5\ 5\ -11\ -11\ 11\ 5\ 5)$$

$$\Phi\ (\beta\ ' M^T) = (-1\ 1\ 1\ 1\ -1\ -1\ 1\ 1\ 1) = \alpha\ '$$

$$\alpha\ ' M = (5\ -19\ -13\ -5\ 1\ 1\ -5\ -13\ 13)$$

$$\Phi\ (\alpha\ ' M) = (1\ -1\ -1\ -1\ 1\ 1\ -1\ -1\ 1) = \beta\ ''$$

$$= \beta\ '$$

Here β **'' =** β **'** . Hence the cycle terminates with

$$\alpha_F = \alpha\ ' = (-1\ 1\ 1\ 1\ -1\ -1\ 1\ 1\ 1) = X_2$$

$$\beta_F = \beta\ ' = (1\ -1\ -1\ -1\ 1\ 1\ -1\ -1\ 1) \neq Y_2$$

But β **'** is not **Y2** . Thus the vector pair **(X2 , Y2)** is not recalled correctly by Kosko's decoding process.

(Continued in next slide)

[Continued from previous slide]

Check with Energy Function : Compute the energy functions

for the coordinates of pair **(X_2, Y_2)** , the energy $\mathbf{E_2 = - X_2\, M\, Y_2^T = -71}$

for the coordinates of pair **(α_F, β_F)** , the energy $\mathbf{E_F = - \alpha_F\, M\, \beta_F^T = -75}$

However, the coordinates of pair **(X_2, Y_2)** is not at its local minimum can be shown by evaluating the energy **E** at a point which is "one Hamming distance" way from **Y_2** . To do this consider a point

$$Y_2' = (\; 1 \; -1 \; -1 \; -1 \quad 1 \; -1 \; -1 \; -1 \; 1 \;)$$

where the fifth component **-1** of **Y_2** has been changed to **1**. Now

$$\mathbf{E = - X_2 \; M \; Y_2'^T = - 73}$$

which is lower than **E_2** confirming the hypothesis that **(X_2, Y_2)** is not at its local minimum of **E**.

Note : The correlation matrix **M** used by Kosko does not guarantee that the energy of a training pair is at its local minimum. Therefore , a pair **Pi** can be recalled if and only if this pair is at a local minimum of the energy surface**.**

4.3 Multiple Training Encoding Strategy

Note : (Ref. example in previous section). Kosko extended the unidirectional auto-associative to bidirectional associative processes, using correlation matrix $\mathbf{M} = \sum X_i^T Y_i$ computed from the pattern pairs. The system proceeds to retrieve the nearest pair given any pair (α, β), with the help of recall equations. However, Kosko's encoding method does not ensure that the stored pairs are at local minimum and hence, results in incorrect recall.

Wang and other's, introduced **multiple training** encoding strategy which ensures the correct recall of pattern pairs. This encoding strategy is an enhancement / generalization of Kosko's encoding strategy. The Wang's generalized correlation matrix is $\mathbf{M} = \sum \mathbf{q}_i X_i^T Y_i$ where $\mathbf{q}_i$ is viewed as pair weight for $X_i^T Y_i$ as positive real numbers. It denotes the minimum number of times for using a pattern pair (X_i, Y_i) for training to guarantee recall of that pair.

To recover a pair (A_i, B_i) using multiple training of order **q**, let us augment or supplement matrix **M** with a matrix **P** defined as

$\mathbf{P} = (\mathbf{q} - 1)\ X_i^T Y_i$ where (X_i, Y_i) are the bipolar form of (A_i, B_i).

The augmentation implies adding **(q - 1)** more pairs located at (A_i, B_i) to the existing correlation matrix. As a result the energy **E'** can reduced to an arbitrarily low value by a suitable choice of **q**. This also ensures that the energy at **(A_i , B_i)** does not exceed at points which are one Hamming distance away from this location.

The new value of the energy function **E** evaluated at **(A_i , B_i)** then becomes

$$E'(A_i , B_i) = - A_i M B_i^T - (q - 1) A_i X_i^T Y_i B_i^T$$

The next few slides explains the step-by-step implementation of Multiple training encoding strategy for the recall of three pattern pairs **(X_1 , Y_1) , (X_1 , Y_1) , (X_1 , Y_1)** using one and same augmentation matrix **M** . Also an algorithm to summarize the complete process of multiple training encoding is given.

34

■ **Example : Multiple Training Encoding Strategy**

The working of multiple training encoding strategy which ensures the correct recall of pattern pairs.

Consider **N = 3** pattern pairs **(A_1 , B_1) , (A_2 , B_2) , (A_3 , B_3)** given by

A_1 = (1 0 0 1 1 1 0 0 0) B_1 = (1 1 1 0 0 0 0 1 0)

A_2 = (0 1 1 1 0 0 1 1 1) B_2 = (1 0 0 0 0 0 0 0 1)

A_3 = (1 0 1 0 1 1 0 1 1) B_3 = (0 1 0 1 0 0 1 0 1)

Convert these three binary pattern to bipolar form replacing

X_1 = (1 -1 -1 1 1 1 -1 -1 -1) Y_1 = (1 1 1 -1 -1

X_2 = (-1 1 1 1 -1 -1 1 1 1) Y_2 = (1 -1 -1 -1 -1

X_3 = (1 -1 1 -1 1 1 -1 1 1) Y_3 = (-1 1 -1 1 -1

Let the pair **(X_2 , Y_2)** be recalled.

X_2 = (-1 1 1 1 -1 -1 1 1 1) Y_2 = (1 -1 -1 -1 -1

Choose **q=2**, so that **P = X_2^T Y_2** , the augmented correlation m

becomes **M = X_1^T Y_1 +2X_2^T Y_2 + X_3^T Y_3**

```
        4   2   2            0   0   2     2  -2
   2   -4  -2  -2            0       0  -2  -2   2
   0                    -2  -4   0  -2  -2   0  -4   4
=                        4  -2   0  -4  -2  -2  -4   0   0
      -2   4   2   2         0       0   2   2  -2
      -2   4   2   2         0       0   2   2  -2
```

$$\left(\begin{array}{cccc} 2 & -4 & -2 & -2 \\ 0 & -2 & -4 & 0 \\ 0 & -2 & -4 & 0 \end{array}\right) \qquad \begin{array}{ccccc} 0 & 0 & -2 & -2 & 2 \\ -2 & -2 & 0 & -4 & 4 \\ -2 & -2 & 0 & -4 & 4 \end{array}$$

(Continued in next slide)

35

[Continued from previous slide]

Now give α = **X2**, and see that the corresponding pattern pair β = **Y2** is correctly recalled as shown below.

$$\alpha \mathbf{M} = (\; 14 \; -28 \; -22 \; -14 \; -8 \; -8 \; -14 \; -22 \; 22 \;)$$

$$\Phi(\alpha \mathbf{M}) = (\; 1 \; -1 \; -1 \; -1 \; -1 \; -1 \; -1 \; -1 \; 1 \;) = \beta'$$

$$\beta' M^T = (\; -16 \; 16 \; 18 \; 18 \; -16 \; -16 \; 16 \; 18 \; 18 \;)$$

$$\Phi(\beta' M^T) = (\; -1 \; 1 \; 1 \; 1 \; -1 \; -1 \; 1 \; 1 \; 1 \;) = \alpha'$$

$$\alpha' \mathbf{M} = (\; 14 \; -28 \; -22 \; -14 \; -8 \; -8 \; -14 \; -22 \; 23 \;)$$

$$(\alpha' \mathbf{M}) = (\; 1 \; -1 \; -1 \; -1 \; 1 \; 1 \; -1 \; -1 \; 1 \;) = \beta''$$

Here $\beta'' = \beta'$. Hence the cycle terminates with

$$\alpha_F = \alpha' = (\; -1 \; 1 \; 1 \; 1 \; -1 \; -1 \; 1 \; 1 \; 1 \;) = X_2$$

$$\beta_F = \beta' = (\; 1 \; -1 \; -1 \; -1 \; 1 \; 1 \; -1 \; -1 \; 1 \;) = Y_2$$

Thus, **(X2 , Y2)** is correctly recalled, using augmented correlation matrix **M** . But, it is not possible to recall **(X1 , Y1)** using the same matrix **M** as shown in the next slide.

(Continued in next slide)

[Continued from previous slide]

Note : The previous slide showed that the pattern pair **(X_2, Y_2)** is correctly recalled, using augmented correlation matrix

$$\mathbf{M} = X_1^T\ Y_1 + 2\ X_2^T\ Y_2 + X_3^T\ Y_3$$

but then the same matrix **M** can not recall correctly the other pattern pair **(X_1, Y_1)** as shown below.

X_1 = (1 -1 -1 1 1 1 -1 -1 -1) **Y_1 = (1 1 1 -1 -1 -1 -1 1 -1)**

Let α **= X_1** and to retrieve the associated pair **Y_1** the calculation shows

α **M = (-6 24 22 6 4 4 6 22 -22)**

Φ **(α M) = (-1 1 1 1 1 1 1 1 -1) =** β **'**

β **' M^T = (16 -16 -18 -18 16 16 -16 -18 -18)**

Φ **(β ' M^T) = (1 -1 -1 -1 1 1 -1 -1 -1) =** α **'**

α **' M = (-14 28 22 14 8 8 14 22 -22)**

Φ **(α ' M) = (-1 1 1 1 1 1 1 1 -1) =** β **"**

Here β **" =** β **'** . Hence the cycle terminates with

α **F =** α **' = (1 -1 -1 -1 1 1 -1 -1 -1) = X_1**

β **F =** β **' = (-1 1 1 1 1 1 1 1 -1) ≠ Y_1**

Thus, the pattern pair **(X_1, Y_1)** is not correctly recalled, using augmented correlation matrix **M**.

To tackle this problem, the correlation matrix **M** needs to be further augmented by **multiple training** of **(X_1 , Y_1)** as shown in the next slide.

(Continued in next slide)

[Continued from previous slide]

The previous slide shows that pattern pair **(X_1 , Y_1)** cannot be recalled under the same augmentation matrix **M** that is able to recall **(X_2 , Y_2)**.

However, this problem can be solved by **multiple training** of **(X_1 , Y_1)** which yields a further change in **M** to values by defining

$$M = 2\, X_1^T\, Y_1 + 2\, X_2^T\, Y_2 + X_3^T\, Y_3$$

$$= \begin{pmatrix} -1 & 5 & 3 & 1 & -1 & -1 & 1 & 3 & -3 \\ 1 & -5 & -3 & -1 & 1 & 1 & -1 & -3 & 3 \\ -1 & -3 & -5 & 1 & -1 & -1 & 1 & -5 & 5 \\ 5 & -1 & 1 & -5 & -3 & -3 & -5 & 1 & -1 \\ -1 & 5 & 3 & 1 & -1 & -1 & 1 & 3 & -3 \\ -1 & 5 & 3 & 1 & -1 & -1 & 1 & 3 & -3 \\ 1 & -5 & -3 & -1 & 1 & 1 & -1 & -3 & 3 \\ -1 & -3 & -5 & 1 & -1 & -1 & 1 & -5 & 5 \\ -1 & -3 & -5 & 1 & -1 & -1 & 1 & -5 & 5 \end{pmatrix}$$

Now observe in the next slide that all three pairs can be correctly recalled.

(Continued in next slide)

[Continued from previous slide]

Recall of pattern pair (X1 , Y1)

$X_1 = (1\ -1\ -1\ 1\ 1\ -1\ -1\ -1)$ $Y_1 = (1\ 1\ 1\ -1\ -1\ -1\ -1\ 1\ -1)$

Let $\alpha = X_1$ and to retrieve the associated pair Y_1 the calculation shows

$$\alpha M = (3\ 33\ 31\ -3\ -5\ -5\ -3\ 31\ -31)$$

$$\Phi(\alpha M) = (1\ 1\ 1\ -1\ -1\ -1\ -1\ 1\ -1) = \beta'$$

$$(\beta' M^T) = (13\ -13\ -19\ 23\ 13\ 13\ -13\ -19\ -19)$$

$$\Phi(\beta' M^T) = (1\ -1\ -1\ 1\ 1\ 1\ -1\ -1\ -1) = \alpha'$$

$$\alpha' M = (3\ 33\ 31\ -3\ -5\ -5\ -3\ 31\ -31)$$

$$\Phi(\alpha' M) = (1\ 1\ 1\ -1\ -1\ -1\ -1\ 1\ -1) = \beta''$$

Here $\beta'' = \beta'$. Hence the cycle terminates with

$$\alpha_F = \alpha' = (1\ -1\ -1\ 1\ 1\ 1\ -1\ -1\ -1) = X_1$$

$$\beta_F = \beta' = (1\ 1\ 1\ -1\ -1\ -1\ -1\ 1\ -1) = Y_1$$

Thus, the pattern pair ***(X₁ , Y₁)*** is correctly recalled

Recall of pattern pair (X2 , Y2)

$X_2 = (-1\ 1\ 1\ 1\ -1\ -1\ 1\ 1\ 1)$ $Y_2 = (1\ -1\ -1\ -1\ -1\ -1\ -1\ -1\ 1)$

Let $\alpha = X_2$ and to retrieve the associated pair Y_2 the calculation shows

$$\alpha M = (7\ -35\ -29\ -7\ -1\ -1\ -7\ -29\ 29)$$

$$\Phi(\alpha M) = (1\ -1\ -1\ -1\ -1\ -1\ -1\ -1\ 1) = \beta'$$

$$(\beta' M^T) = (-15\ 15\ 17\ 19\ -15\ -15\ 15\ 17\ 17)$$

$$\Phi(\beta' M^T) = (-1\ 1\ 1\ 1\ -1\ -1\ 1\ 1\ 1) = \alpha'$$

$$\alpha' M = (7\ -35\ -29\ -7\ -1\ -1\ -7\ -29\ 29)$$

$\phi(\alpha' M) = (1\ -1\ -1\ -1\ -1\ -1\ -1\ -1\ 1) = \beta''$

Here $\beta'' = \beta'$. Hence the cycle terminates with

$\alpha_F = \alpha' = (-1\ 1\ 1\ 1\ -1\ -1\ 1\ 1\ 1) = X_2$

$\beta_F = \beta' = (1\ -1\ -1\ -1\ -1\ -1\ -1\ -1\ 1) = Y_2$

Thus, the pattern pair ***(X_2, Y_2)*** is correctly recalled

Recall of pattern pair (X3 , Y3)

$X_3 = (1\ -1\ 1\ -1\ 1\ 1\ -1\ 1\ 1)$ $Y_3 = (-1\ 1\ -1\ 1\ -1\ -1\ 1\ 0\ 1)$

Let α = ***X_3*** and to retrieve the associated pair ***Y_3*** the calculation shows

$\alpha M = (-13\ 17\ -1\ 13\ -5\ -5\ 13\ -1\ 1)$

$\Phi(\alpha M) = (-1\ 1\ -1\ 1\ -1\ -1\ 1\ -1\ 1) = \beta'$

$(\beta' M^T) = (11\ -11\ 27\ -63\ 11\ 11\ -11\ 27\ 27)$

$\Phi(\beta' M^T) = (1\ -1\ 1\ -1\ 1\ 1\ -1\ 1\ 1) = \alpha'$

$\alpha' M = (-13\ 17\ -1\ 13\ -5\ -5\ 13\ -1\ 1)$

$\phi(\alpha' M) = (-1\ 1\ -1\ 1\ -1\ -1\ 1\ -1\ 1) = \beta''$

Here $\beta'' = \beta'$. Hence the cycle terminates with

$\alpha_F = \alpha' = (1\ -1\ 1\ -1\ 1\ 1\ -1\ 1\ 1) = X_3$

$\beta_F = \beta' = (-1\ 1\ -1\ 1\ -1\ -1\ 1\ 0\ 1) = Y_3$

Thus, the pattern pair ***(X_3, Y_3)*** is correctly recalled

(Continued in next slide)

[Continued from previous slide]

Thus, the multiple training encoding strategy ensures the correct recall of a pair for a suitable augmentation of **M** . The generalization of the correlation matrix, for the correct recall of all training pairs, is written as

$$\mathbf{M} = \sum_{i=1}^{N} \mathbf{q_i} \, X_i^T \, Y_i$$ where $\mathbf{q_i}$ **'s** are **+ve** real numbers.

This modified correlation matrix is called generalized correlation matrix. Using one and same augmentation matrix **M**, it is possible to recall all the training pattern pairs .

40

- **Algorithm** (for the Multiple training encoding strategy)

  To summarize the complete process of multiple training encoding an algorithm is given below.

  Algorithm Mul_Tr_Encode **(N ,** X_i **,** Y_i **,** q_i **)** where

  ■ : Number of stored patterns set

  $X_{,i}$

  Y_i

  $X =$

  : the bipolar pattern pairs

  $x\ i$ x

  $Y =$

  (X_1, X_2, ... , X_N**)** where X_i **= (** $_1$ $'x\ i\ 2$, . . . $i\ n$ **)**

  ■

  (: Y_1, Y_2, ... , Y_N**)** where Y_j **= (** $x\ j\ 1$, $x\ j\ 2$, . . . $x\ j\ n$ **)**

  is the weight vector **(**q_1 , q_2 , ... , q_N **)**

```
Step 1  Initialize correlation matrix M to null matrix   $M \leftarrow [0]$
Step 2  Compute the correlation matrix M as
        For $i \leftarrow 1$ to $N$
        $M \leftarrow M \oplus [\, q_i * \text{Transpose}(X_i) \otimes (X_i)\,]$   end
        (symbols $\oplus$ matrix addition, $\otimes$ matrix multiplication, and
                 $*$ scalar multiplication)
Step 3  Read input bipolar pattern $A$
Step 4  Compute A_M where $A\_M \leftarrow A \otimes M$
Step 5  Apply threshold function $\Phi$ to A_M to get $B'$
         ie $B' \leftarrow \Phi(A\_M)$
        where $\Phi$ is defined as $\Phi(F) = G = g_1, g_2, \ldots, g_n$
Step 6  Output $B'$ is the associated pattern pair
end
```

Adaptive Resonance Theory (ART)

What is ART ?

f ART stands for "Adaptive Resonance Theory", invented by Stephen Grossberg in 1976. ART represents a family of neural networks.

g ART encompasses a wide variety of neural networks.

The basic ART System is an unsupervised learning model.

- The term "resonance" refers to resonant state of a neural network in

  which a category prototype vector matches close enough to the current input vector. ART matching leads to this resonant state, which permits learning. The network learns only in its resonant state.

- ART neural networks are capable of developing stable clusters of arbitrary sequences of input patterns by self-organizing.

  ART-1 can cluster binary input vectors. ART-2
  can cluster real-valued input vectors.

- ART systems are well suited to problems that require online learning of large and evolving databases.

1. Adaptive Resonance Theory (ART)

Real world is faced with a situations where data is continuously changing.

In such situation, every learning system faces plasticity-stability dilemma.

This dilemma is about *:*

> "A system that must be able to learn to adapt to a changing environment (ie it must be plastic) but the constant change can make the system unstable, because the system may learn new information only by forgetting every thing it has so far learned."

This phenomenon, a contradiction between plasticity and stability, is called **plasticity - stability dilemma.**

The back-propagation algorithm suffer from such stability problem.

- Adaptive Resonance Theory (ART) is a new type of neural network, designed by Grossberg in 1976 to solve plasticity-stability dilemma.

- ART has a self regulating control structure that allows autonomous recognition and learning.

- ART requires no supervisory control or algorithmic implementation.

- **Recap** *(Supervised , Unsupervised and BackProp Algorithms)*

  Neural networks learn through supervised and unsupervised means.

  The hybrid approaches are becoming increasingly common as well.

  - In **supervised learning**, the input and the expected output of the system are provided, and the ANN is used to model the relationship between the two. Given an input set **x**, and a corresponding output

    set **y**, an optimal rule is determined such that: **y = f(x) + e** where, **e** is an approximation error that needs to be minimized. Supervised learning is useful when we want the network to reproduce the characteristics of a certain relationship.

  - In **unsupervised learning**, the data and a cost function are provided. The ANN is trained to minimize the cost function by finding a suitable

    input-output relationship. Given an input set **x**, and a cost function **g(x, y)** of the input and output sets, the goal is to minimize the cost function through a proper selection of **f** (the relationship between **x**, and **y**). At each training iteration, the trainer provides the input to the network, and the network produces a result. This result is put into the cost function, and the total cost is used to update the weights. Weights are continually updated until the system output produces a minimal cost. Unsupervised learning is useful in situations where a cost function is known, but a data set is not know that minimizes that cost function over a particular input space.

- In **backprop network learning,** a set of input-output pairs are given and the network is able to learn an appropriate mapping. Among the

  supervised learning, BPN is most used and well known for its ability to attack problems which we can not solve explicitly. However there are several technical problems with back-propagation type algorithms.

  They are not well suited for tasks where input space changes and are often slow to learn, particularly with many hidden units. Also the semantics of the algorithm are poorly understood and not biologically plausible, restricting its usefulness as a model of neural learning.

Most learning in brains is completely unsupervised.

05

■ Competitive Learning Neural Networks

While no information is available about desired outputs the network updated weights only on the basis of input patterns. The Competitive Learning network is unsupervised learning for categorizing inputs. The neurons (or units) compete to fire for particular inputs and then learn to respond better for the inputs that activated them by adjusting their weights.

– For an output unit j , the input vector $X = [x_1 , x_2 , x_3]^T$ and the weight vector $W_j = [w_{1j} , w_{1j} , w_{1j}]^T$ are normalized to unit length.

– The activation value a_j of the output unit j is calculated by the inner product of the weight vectors

$$a_j = \sum_{i=1}^{3} x_i \; w_{ij} = X_T \, W_j = W_j \; X_T$$

and then the output unit with the highest activation is selected for further processing; **this implied competitive.**

– Assuming that output unit k has the maximal activation, the weights leading to this unit are updated according to the competitive, called **winner-take-all (WTA)** learning rule

$$W_k (t + 1) = \frac{w_k (t) + \eta \{x (t) + w_k (t)\}}{||w_k (t) + \eta \{x (t) + w_k (t)\}||}$$

which is normalized to ensure that $w_k\ (t + 1)$ is always of unit length; only the weights at the winner output unit **k** are updated and all other weights remain unchanged.

– Alternately, Euclidean distance as a dissimilarity measure is a more general scheme of competitive learning, in which the activation of output unit j is as

$$a_j = \left\{ \sum_{i=1}^{3} (x_i - w_{ij})^2 \right\}^{1/2} = || x_i - w_{ij} ||$$

the weights of the output units with the smallest activation are updated according to

$$w_k\ (t + 1) = w_k\ (t) + \eta\ \{x\ (t) + w_k\ (t)\}$$

A competitive network, on the input patterns, performs an on-line clustering process and when complete the input data are divided into disjoint clusters such that similarity between individuals in the same cluster are larger than those in different clusters. Stated above, two metrics of similarity measures: one is Inner product and the other Euclidean distance. Other metrics of similarity measures can be used. The selection of different metrics lead to different clustering.

Limitations of Competitive Learning :

– Competitive learning lacks the capability to add new clusters when deemed necessary.

– Competitive learning does not guarantee stability in forming clusters.

If the learning rate η is constant, then the winning unit that responds to a pattern may continue changing during training.

■ If the learning rate η is decreasing with time, it may become too small to update cluster centers when new data of different probability are presented.

Carpenter and Grossberg (1998) referred such occurrence as the stability-plasticity dilemma which is common in designing intelligent learning systems. In general, a learning system should be plastic, or adaptive in reacting to changing environments, and should be stable to preserve knowledge acquired previously.

- **Stability-Plasticity Dilemma (SPD)**

Every learning system faces the plasticity-stability dilemma.

The plasticity-stability dilemma poses few questions :

– How can we continue to quickly learn new things about the environment and yet not forgetting what we have already learned?

– How can a learning system remain plastic (adaptive) in response to significant input yet stable in response to irrelevant input?

- How can a neural network can remain plastic enough to learn new patterns and yet be able to maintain the stability of the already learned patterns?

- How does the system know to switch between its plastic and stable modes.

- What is the method by which the system can retain previously learned information while learning new things.

Answer to these questions, about plasticity-stability dilemma in learning systems is the Grossberg's Adaptive Resonance Theory (ART).

- ART has been developed to avoid stability-plasticity dilemma in competitive networks learning.

- The stability-plasticity dilemma addresses how a learning system can preserve its previously learned knowledge while keeping its ability to learn new patterns.

- ART is a family of different neural architectures. ART architecture can self-organize in real time producing stable recognition while getting input patterns beyond those originally stored.

2. Adaptive Resonance Theory (ART) Networks

An adaptive clustering technique was developed by Carpenter and Grossberg in 1987 and is called the Adaptive Resonance Theory (ART) .

The Adaptive Resonance Theory (ART) networks are self-organizing competitive neural network. ART includes a wide variety of neural networks. ART networks follow both supervised and unsupervised algorithms.

- The **unsupervised ARTs** named as ART1, ART2 , ART3, . . and are similar

to many iterative clustering algorithms where the terms "nearest" and "closer" are modified by introducing the concept of "resonance".

Resonance is just a matter of being within a certain threshold of a second similarity measure.

The **supervised ART** algorithms that are named with the suffix "MAP", as ARTMAP. Here the algorithms cluster both the inputs and targets and associate two sets of clusters.

The basic ART system is unsupervised learning model. It typically consists of

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

– a reset module

Each of these are explained in the next slide.

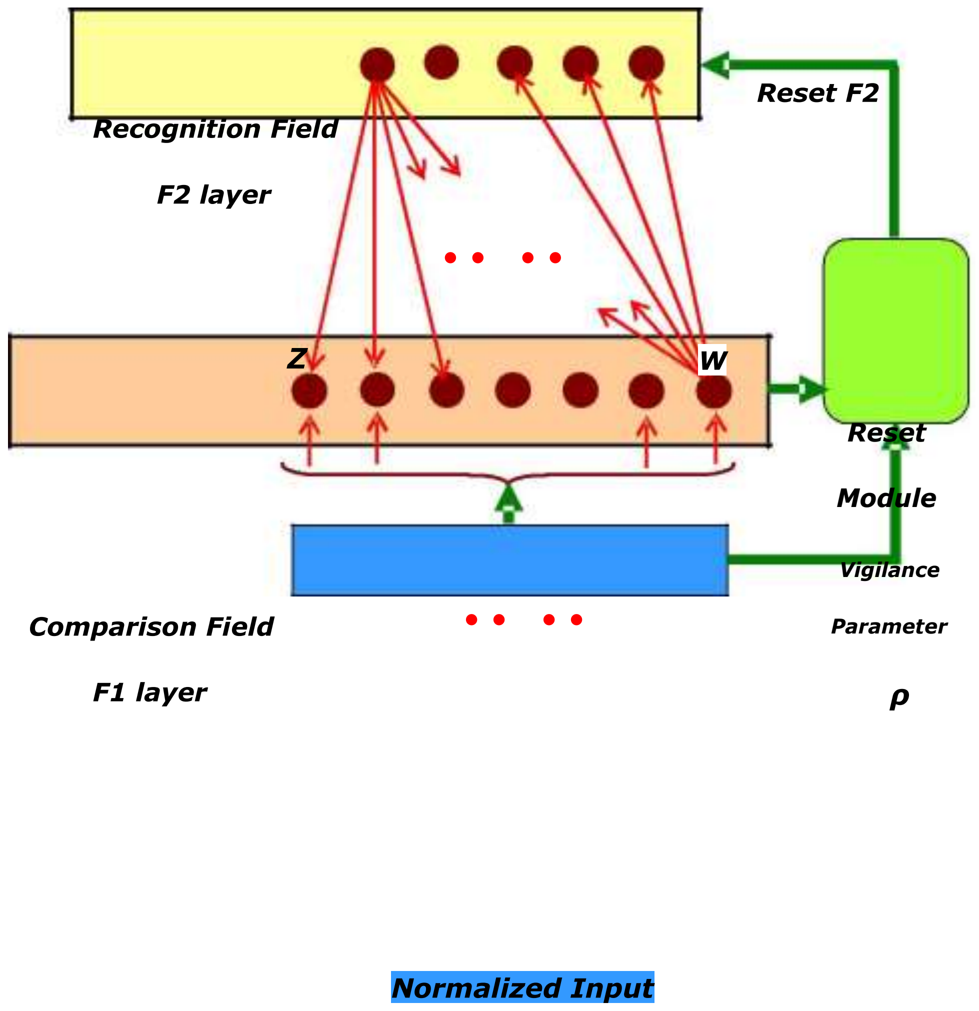


Fig Basic ART Structure

10

- **Comparison field**

The comparison field takes an input vector (a one-dimensional array of values) and transfers it to its best match in the recognition field. Its best match is the single neuron whose set of weights

(weight vector) most closely matches the input vector.

= **Recognition field**

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. In this way the recognition field exhibits lateral inhibition, allowing each neuron in it to represent a category to which input vectors are classified.

= **Vigilance parameter**

After the input vector is classified, a reset module compares the

strength of the recognition match to a vigilance parameter. The vigilance parameter has considerable influence on the system:

- Higher vigilance produces highly detailed memories (many, fine-grained categories), and

- Lower vigilance results in more general memories (fewer, more-general categories).

- **Reset Module**

The reset module compares the strength of the recognition match to the vigilance parameter.

– If the vigilance threshold is met, then training commences.

– Otherwise, if the match level does not meet the vigilance

parameter, then the firing recognition neuron is inhibited until a new input vector is applied;

Training commences only upon completion of a search procedure.

In the 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.

2.1 Simple ART Network

ART includes a wide variety of neural networks. ART networks follow both supervised and unsupervised algorithms. The unsupervised ARTs as ART1, ART2, ART3, ... are similar to many iterative clustering algorithms.

The simplest ART network is a vector classifier. It accepts as input a vector and classifies it into a category depending on the stored pattern it most closely resembles. Once a pattern is found, it is modified (trained) to resemble the input vector. If the input vector does not match any stored pattern within a certain tolerance, then a new category is created by storing a new pattern similar to the input vector. Consequently, no stored pattern is ever modified unless it matches the input vector within a certain tolerance.

This means that an ART network has

- both plasticity and stability;
- new categories can be formed when the environment does not match any of the stored patterns, and
- the environment cannot change stored patterns unless they are sufficiently similar.

2.2 General ART Architecture

The general structure, of an ART network is shown below.

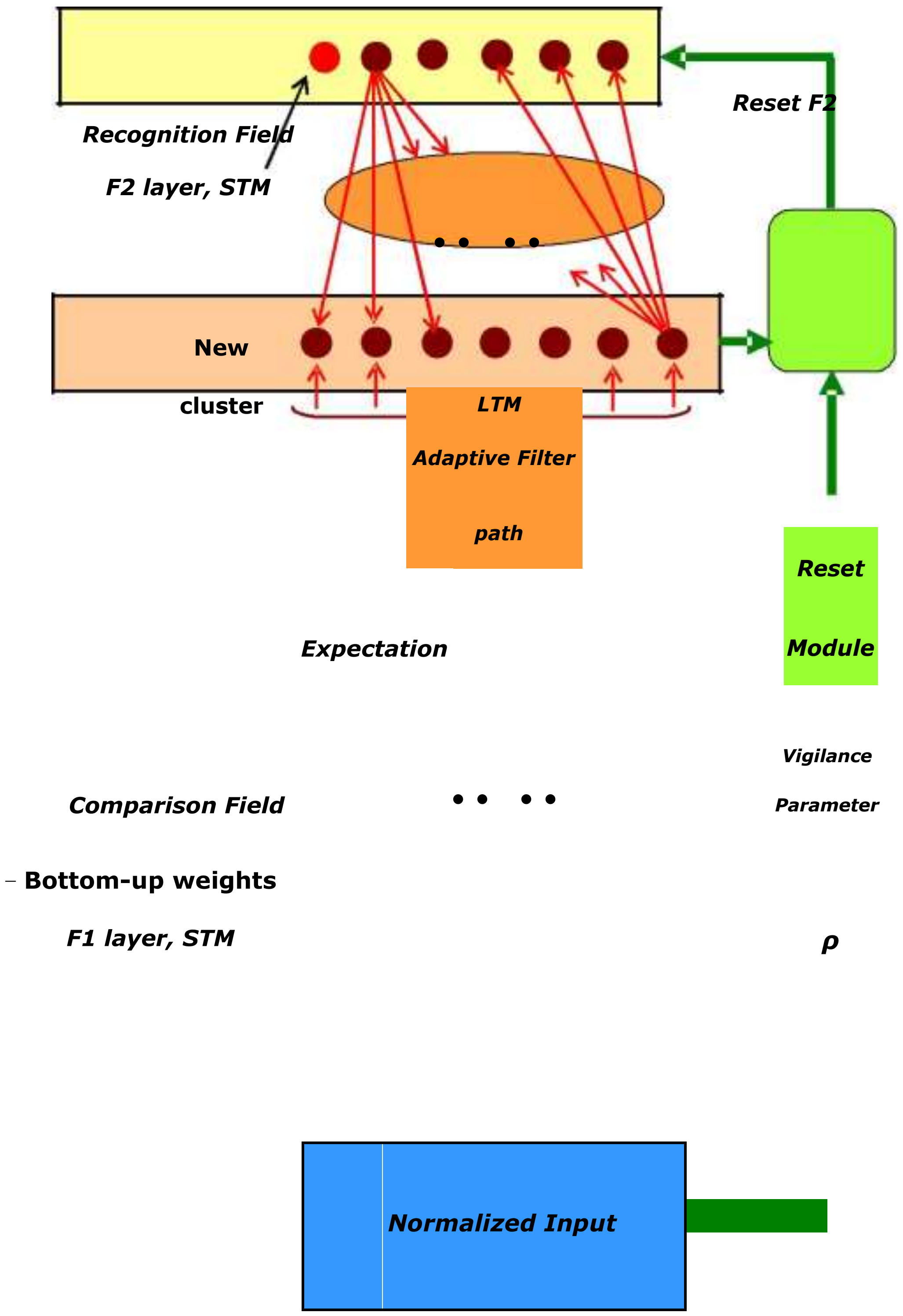


Fig Simplified ART Architecture

There are two layers of neurons and a reset mechanism.

- **F1 layer** : an input processing field; also called **comparison layer**.

- **F2 layer** : the cluster units ; also called **competitive layer**.

- **Reset mechanism** : to control the degree of similarity of patterns placed on the same cluster; takes decision whether or not to allow cluster unit to learn.

There are two sets of connections, each with their own weights, called :

from each unit of layer **F1** to all units of layer **F2** .

- **Top-down weights** from each unit of layer **F2** to all units of layer **F1** .

14

2.3 Important ART Networks

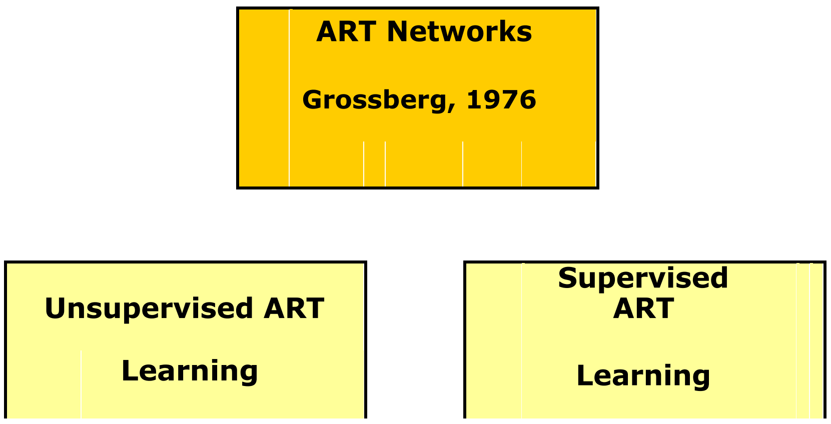
The ART comes in several varieties. They belong to both unsupervised and supervised form of learning.

Unsupervised ARTs are named as ART1, ART2 , ART3, . . and are similar to many iterative clustering algorithms.

- ART1 model (1987) designed to cluster binary input patterns.
- ART2 model (1987) developed to cluster continuous input patterns.
- ART3 model (1990) is the refinement of these two models.

Supervised ARTs are named with the suffix "MAP", as ARTMAP, that combines two slightly modified ART-1 or ART-2 units to form a supervised learning model where the first unit takes the input data and the second unit takes the correct output data. The algorithms cluster both the inputs and targets, and associate the two sets of clusters. **Fuzzy ART** and **Fuzzy ARTMAP** are generalization using fuzzy logic.

A taxonomy of important ART networks are shown below.



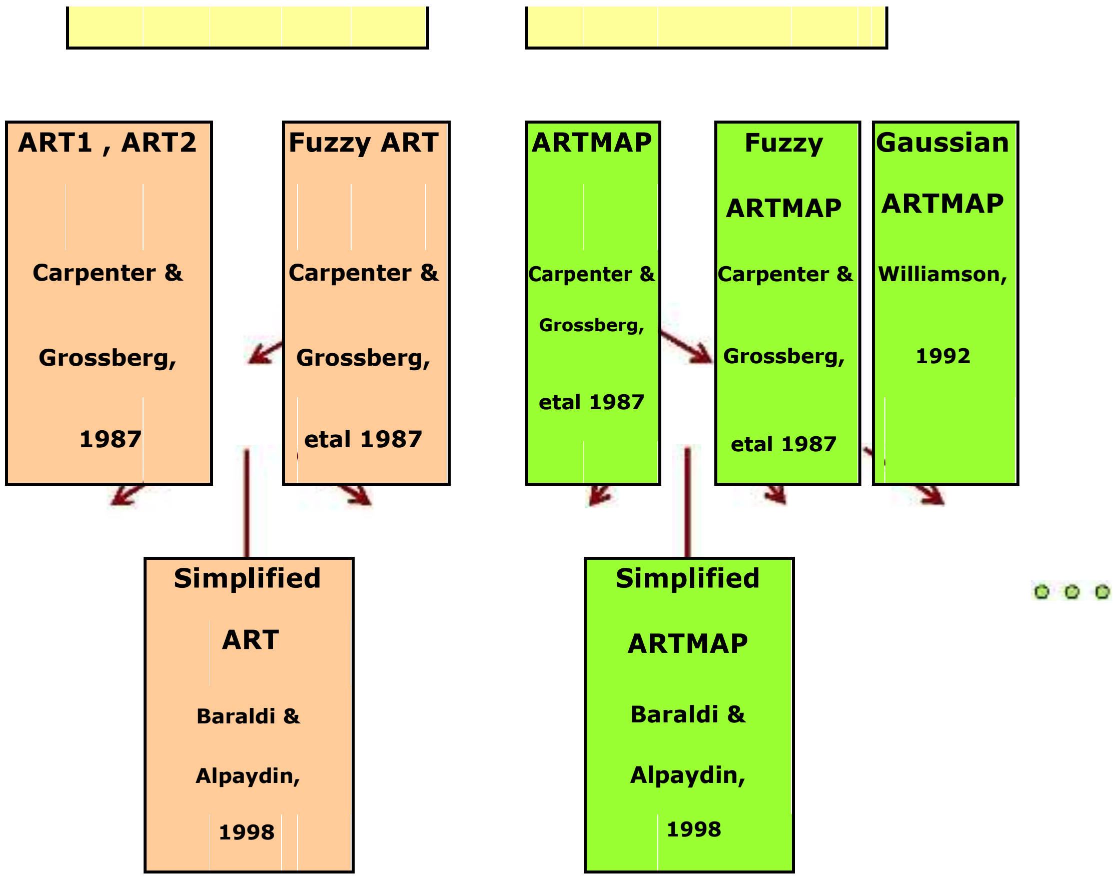


Fig. Important ART Networks

Note : Only the unsupervised ARTs are presented in what follows in the remaining slides.

15

2.4 Unsupervised ARTs – Discovering Cluster Structure

Human has ability to learn through classification. Human learn new concepts by relating them to existing knowledge and if unable to relate to something already known, then creates a new structure. The unsupervised ARTs named as ART1, ART2 , ART3*, . .* represent such human like learning ability.

ART is similar to many iterative clustering algorithms where each pattern is processed by

- Finding the "nearest cluster" seed/prototype/template to that pattern and then updating that cluster to be "closer" to the pattern;
- Here the measures "nearest" and "closer" can be defined in different ways in n-dimensional Euclidean space or an n-space.

How ART is different from most other clustering algorithms is that it is capable of determining number of clusters through adaptation.

- ART allows a training example to modify an existing cluster only if the cluster is sufficiently close to the example (the cluster is said to "resonate" with the example); otherwise a new cluster is formed to handle the example
- To determine when a new cluster should be formed, ART uses a vigilance parameter as a threshold of similarity between patterns and clusters.

ART networks can "discover" structure in the data by finding how the data is clustered. The ART networks are capable of developing stable clusters of arbitrary sequences of input patterns by self-organization.

Note : For better understanding, in the subsequent sections, first the iterative clustering algorithm (a non-neural approach) is presented then the ART1 and ART2 neural networks are presented.

16

- **Iterative Clustering** - **Non Neural Approach**

Organizing data into sensible groupings is one of the most fundamental mode of understanding and learning.

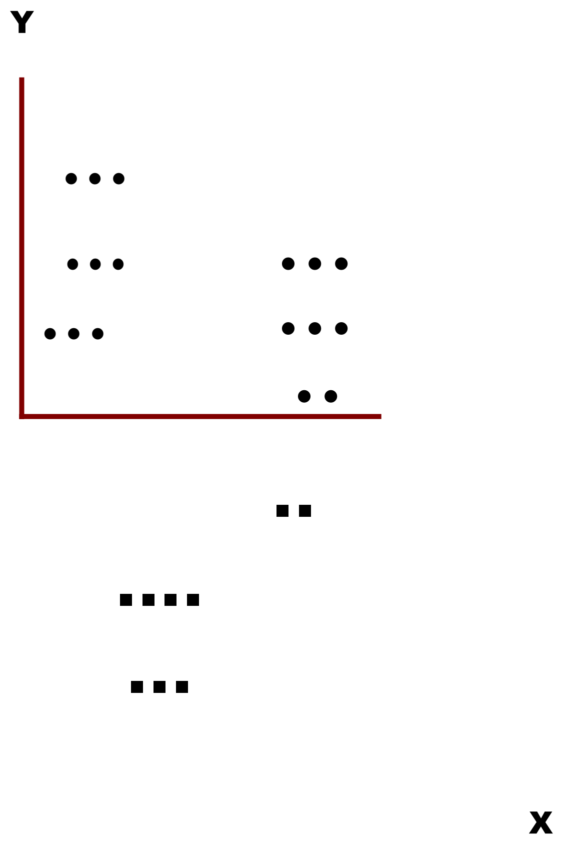
Clustering is a way to form 'natural groupings' or clusters of patterns. Clustering is often called an unsupervised learning.

- cluster analysis is the study of algorithms and methods for grouping, or clustering, objects according to measured or perceived intrinsic characteristics or similarity.

- Cluster analysis does not use category labels that tag objects with prior identifiers, i.e., class labels.
- The absence of category information, distinguishes the clustering (unsupervised learning) from the classification or discriminant analysis (supervised learning).
- The aim of clustering is exploratory in nature to find structure in data.

- **Example :**

Three natural groups of data points, that is three natural clusters.



In clustering, the task is to learn a classification from the data represented in an n-dimensional Euclidean space or an n-space.

- the data set is explored to find some intrinsic structures in them;
- no predefined classification of patterns are required;

The K-mean, ISODATA and Vector Quantization techniques are some of the decision theoretic approaches for cluster formation among unsupervised learning algorithms.

(Note : a recap of distance function in n-space is first mentioned and then vector quantization clustering is illustrated.)

18

■ Recap : Distance Functions

Vector Space Operations

Let **R** denote the field of real numbers.

For any non-negative integer **n**, the space of all n-tuples of real numbers forms an n-dimensional vector space over ***R***, denotes R^n.

An element of R_n is written as $X = (x_1, x_2, \ldots x_i \ldots, x_n)$, where x_i is a real number. Similarly the other element $Y = (y_1, y_2, \ldots y_i \ldots, y_n)$

The vector space operations on R^n are defined by

$X + Y = (x_1 + y_1, X_2 + y_2, \ldots, x_n + y_n)$ and

$aX = (ax_1, ax_2, \ldots, ax_n)$

The standard inner product, called dot product, on R_n, is given by

$X \bullet Y = \sum^{n}_{i=1} (x_1\, y_1 + x_2\, y_2 + \ldots + xn\, yn)$ is a real number.

The dot product defines a **distance function** (or metric) on R^n by

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

The (interior) angle ***θ*** between **x** and **y** is then given by

$$\theta = \cos^{-1}\left(\frac{X \bullet Y}{||X||\ ||Y||}\right)$$

The dot product of **X** with itself is always non negative, is given by

$$||X|| = \sum^{n}_{i=1} (x_i - x_i)^2$$

[Continued in next slide]

19

- **Euclidean Distance**

  It is also known as Euclidean metric, is the "ordinary" distance between two points that one would measure with a ruler.

  The Euclidean distance between two points

  $\mathbf{P = (p_1 , p_2 , . . p_i . . , x_n)}$ and

  $\mathbf{Q = (q_1 , q_2 , . . q_i . . , q_n)}$

  in Euclidean n-space , is defined as :

  $$\sqrt{(p_1 – q_1)^2 + (p_2 – q_2)^2 + . . + (p_n – q_n)^2}$$

  $$\sqrt{\quad}$$

  - $\sum^{n}_{i=1} (p_i - q_i)^2$

  Example : Three-dimensional distance

  For two 3D points,

  $\mathbf{P = (p_x, p_y, . . p_z)}$ and

  $\mathbf{Q = (q_x, q_y, . . q_z)}$

  The Euclidean 3-space , is computed as :

$$\sqrt{(p_x - q_x)^2 + (p_y - q_y)^2 + (p_z - q_z)^2}$$

20

3.1 Vector Quantization Clustering

The goal is to "discover" structure in the data by finding how the data is clustered. One method for doing this is called vector quantization for grouping feature vectors into clusters.

The Vector Quantization (VQ) is non-neural approach to dynamic allocation of cluster centers.

VQ is a non-neural approach for grouping feature vectors into clusters.

It works by dividing a large set of points (vectors) into groups having approximately the same number of points closest to them. Each group is represented by its centroid point, as in k-means and some other clustering algorithms.

- **Algorithm for vector quantization**

  - To begin with, in VQ no cluster has been allocated; first pattern would hold itself as the cluster center.

  - When ever a new input vector X^p as pth pattern appears, the Euclidean distance d between it and the jth cluster center C_j is calculated as

$$\mathbf{d} = | X^p - C_j | = \left[\sum_{i=1}^{N} (X_i^p - C_{ji})^2 \right]^{1/2}$$

  - The cluster closest to the input is determined as

$$| X^p - C_k | < | X^p - C_j | = \left\{ \begin{array}{l} j = 1, .., M \\ \textit{minimum} \\ j \neq k \end{array} \right.$$

where **M** is the number of allotted clusters.

– Once the closest cluster **k** is determined, the distance $| X^p - C_k |$ must be tested for the threshold distance ρ as

◇ $| X^p - C_k | < \rho$ pattern assigned to **k**th cluster

2. $| X^p - C_k | > \rho$ a new cluster is allocated to pattern **p**

– update that cluster centre where the current pattern is assigned

$$C_x = (1/N_x) \sum_{x \in S_n} X$$

where set **X** represents all patterns coordinates **(x , y)** allocated to that cluster (ie S_n) and **N** is number of such patterns.

21

◇ **Example 1 :** *(Ref previous slide)*

Consider 12 numbers of pattern points in Euclidean space.

Their coordinates **(X , Y)** are indicated in the table below.

Table Input pattern - coordinates of 12 points

Points	X	Y	Points	X	Y
1	2	3	7	6	4
2	3	3	8	7	4
3	2	6	9	2	4
4	3	6	10	3	4
5	6	3	11	2	7
6	7	3	12	3	7

Determine clusters using VQ, assuming the threshold distance = 2.0.

- Take a new pattern, find its distances from all the clusters identified,
- Compare distances w.r.t the threshold distance and accordingly decide cluster allocation to this pattern,
- Update the cluster center to which this new pattern is allocated,
- Repeat for next pattern.

Computations to form clusters

	Determining cluster closest to input pattern						Cluster no
Input	Cluster 1		Cluster 2		Cluster 3		assigned to
Pattern	Distance	center	Distance	center	Distance	center	i/p pattern
1, (2,3)	0	(2 , 3)					1
2, (3,3)	1	(2.5 , 3)					1

3,	(2,6)	3.041381		0	(2 , 6)			2
4,	(3,6)	3.041381		1	(2.5 , 6)			2
5,	(6,3)	4.5		5.408326		0	(6 , 3)	3
6,	(7,3)	5.5		6.264982		1	(6.5 , 3)	3
7,	(6,4)	3.640054		4.031128		1.118033	(6.333333, 3.3333)	3
8,	(7,4)	4.609772		4.924428		0.942809	(6.5, 3.5)	3
9,	(2,4)	1.11803	(2.33333, 3.33333)	2.06155		4.527692		1
10,	(3,4)	0.942808	(2.5, 3.5)	2.0615528		3.5355339		1
11,	(2,7)	3.5355339		1.1180339	(2.333333, 6.333333)	7.2629195		2
12,	(3,7)	3.5707142		0.9428089	(2.5, 6.5)	4.9497474		2

The computations illustrated in the above table indicates :

– No of clusters **3**

– Cluster centers **C$_1$ = (2.5, 3.5) ; C$_2$ = (2.5, 6.5); C$_3$ = (6.5, 3.5).**

– Clusters Membership **S(1) = {P1, P2, P9, P10}; S(2) = {P3, P4, P11, P12};**

S(3) = {P5, P6, P7, P8};

These results are graphically represented in the next slide

22

[Continued from previous slide]

Graphical Representation of Clustering

(Ref - Example -1 in previous slide)

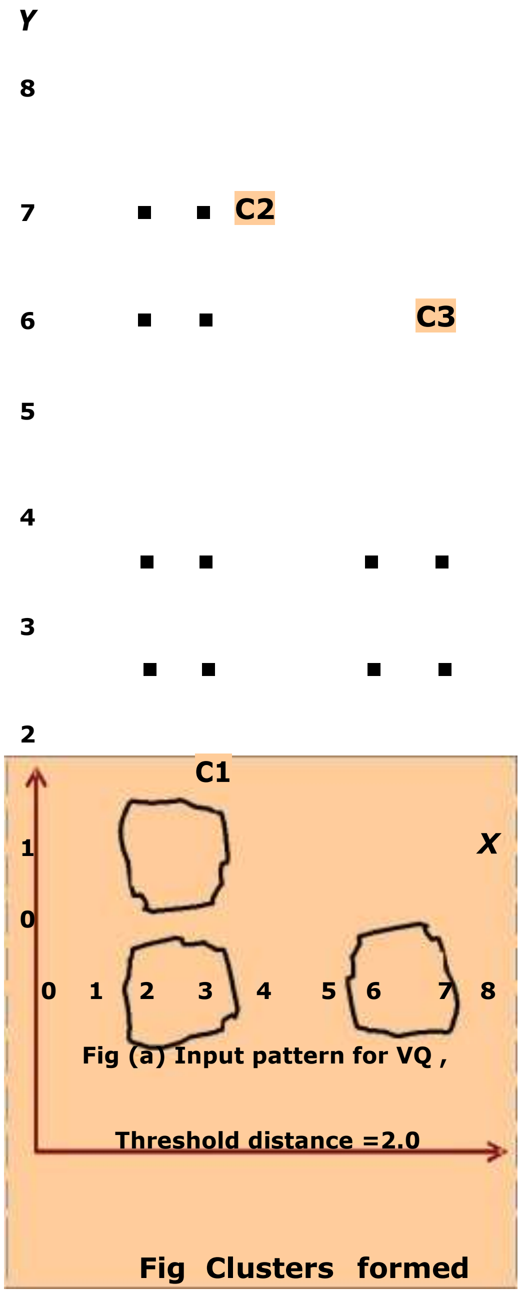


Fig (a) Input pattern for VQ ,

Threshold distance =2.0

Fig Clusters formed

Results of vector quantization :

Clusters formed

- Number of input patterns : 12
- Threshold distance assumed : 2.0
- No of clusters : 3
- Cluster centers :

  C1 = (2.5, 3.5) ;

  C2 = (2.5, 6.5);

  C3 = (6.5, 3.5).
- Clusters Membership :

  S(1)= {P1, P2, P9, P10};

  S(2) = {P3, P4, P11, P12};

  S(1) = {P5, P6, P7, P8};

Note : About threshold distance

- large threshold distance may obscure meaningful categories.
- low threshold distance may increase more meaningful categories.
- See next slide, clusters for threshold distances as **3.5** and **4.5** .

23

- **Example 2**

The input patterns are same as of Example 1.

Determine the clusters, assuming the threshold distance = 3.5 and 4.5. – follow the same procedure as of Example 1 ;

– do computations to form clusters, assuming the threshold distances as **3.5** and **4.5**.

– The results are shown below.

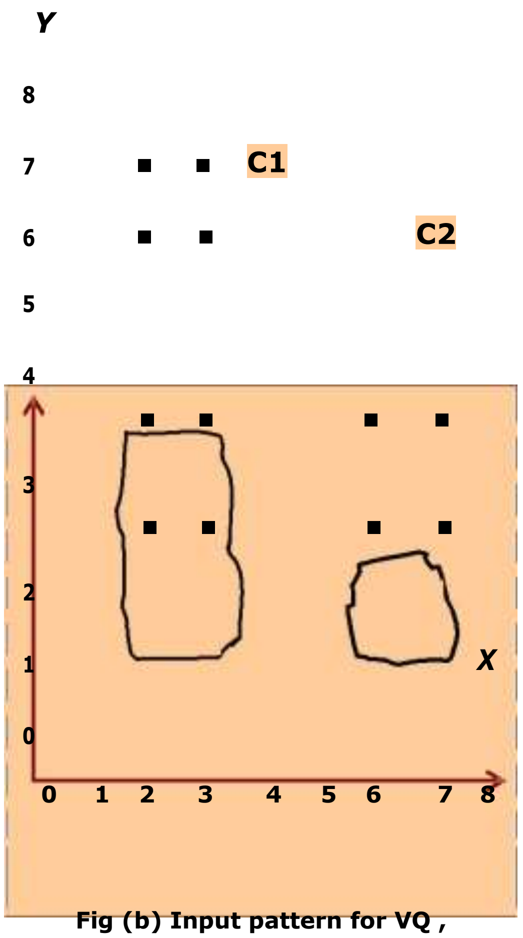


Fig (b) Input pattern for VQ ,

Threshold distance = 3.5

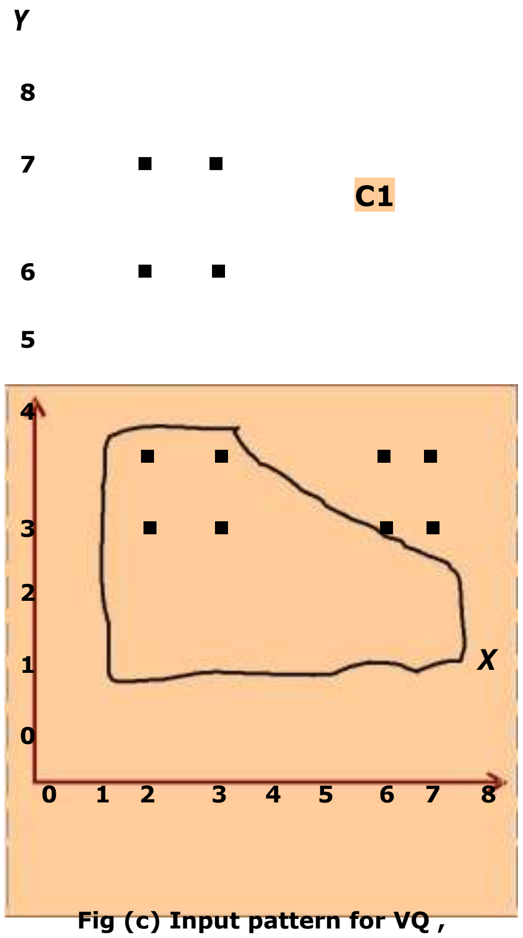


Fig (c) Input pattern for VQ ,

Threshold distance = 4.5

Fig Clusters formed

- Fig (b) for the threshold distance = **3.5** , two clusters formed.
- Fig (c) for the threshold distance = **4.5** , one cluster formed.

24

☐ Unsupervised ART Clustering

The taxonomy of important ART networks, the basic ART structure, and the general ART architecture have been explained in the previous slides. Here only Unsupervised ART (ART1 and ART2) Clustering are presented.

ART1 is a clustering algorithm can learn and recognize binary patterns. Here

– similar data are grouped into cluster

– reorganizes clusters based upon the changes

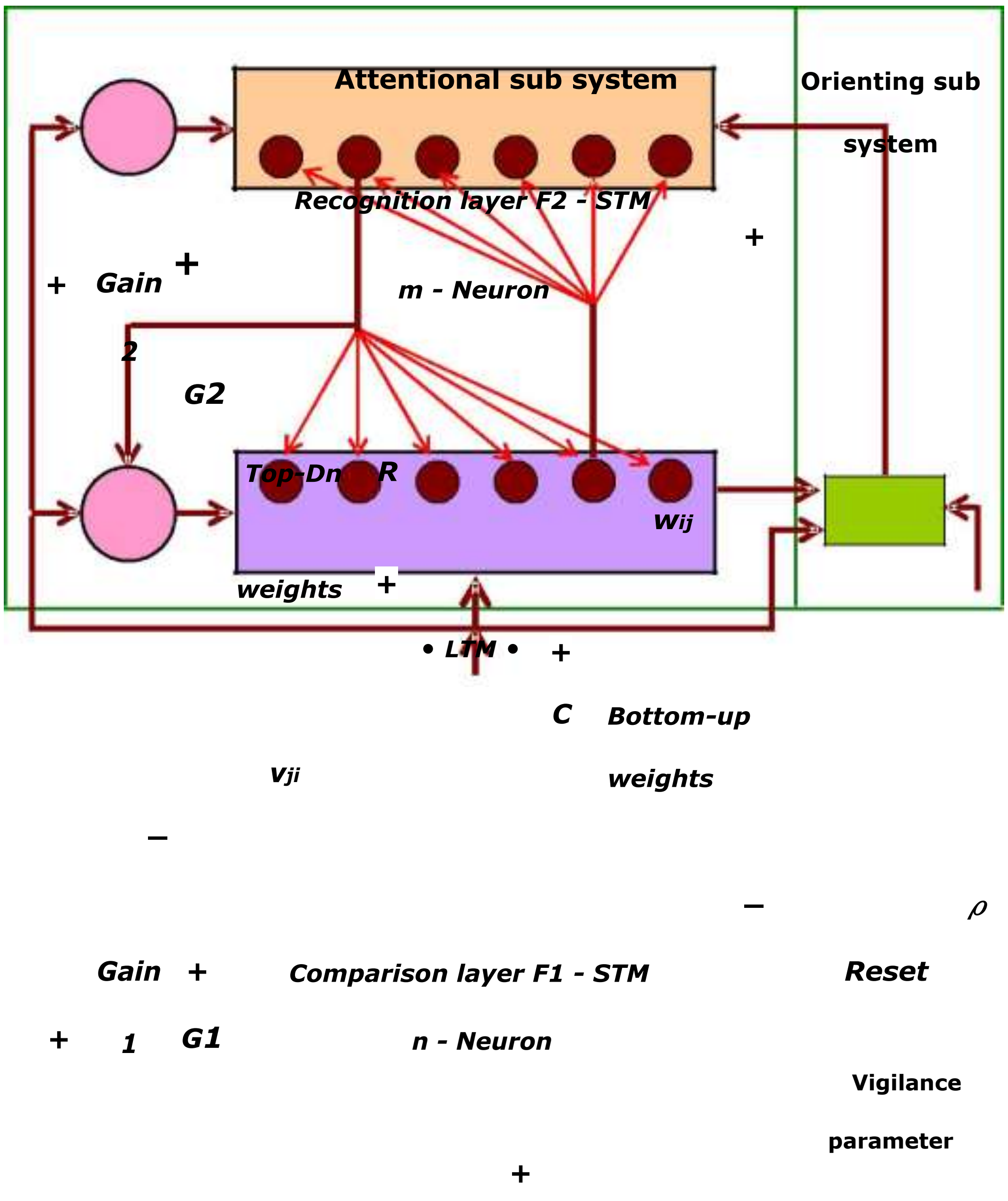
– creates new cluster when different data is encountered

ART2 is similar to ART1, can learn and recognize arbitrary sequences of analog input patterns.

The ART1 architecture, the model description, the pattern matching cycle, and the algorithm - clustering procedure, and a numerical example is presented in this section.

4.1 ART1 Architecture

The Architecture of ART1 neural network consist of two layers of neurons.



Pattern Vector (P_i = 0 or 1)

$I_{H=1}$ [1 1 0 0 P_i 0]

- - - - - - - - - - - - - - - - - -

$I_{H=h}$ [1 0 0 1 P_i 0]

Fig. ART1 Network architecture

ATR1 model consists an "Attentional" and an "Orienting" subsystem.

The **Attentional sub-system** consists of :

- two competitive networks, as Comparison layer **F1** and Recognition layer **F2**, fully connected with top-down and bottom-up weights;
- two control gains, as **Gain1** and **Gain2**.

- Reset layer for controlling the attentional sub-system overall dynamics based on vigilance parameter.
- Vigilance parameter ρ determines the degree of mismatch to be tolerated between the input pattern vectors and the weights connecting **F1** and **F2**.

The nodes at **F2** represent the clusters formed. Once the network stabilizes, the top-down weights corresponding to each node in **F2** represent the prototype vector for that node.

26

4.2 ART1 Model Description

The ART1 system consists of two major subsystem, an attentional subsystem and an orienting subsystem, described below. The system does pattern matching operation during which the network structure tries to determine whether the input pattern is among the patterns previously stored in the network or not.

- **Attentional Subsystem**

(a) **F1 layer** of neurons/nodes called or input layer or comparison layer; short term memory (**STM**).

(b) **F2 layer** of neurons/nodes called or output layer or recognition layer; short term memory (**STM**).

(c) **Gain control unit** , Gain1 and Gain2, one for each layer.

(d) **Bottom-up connections** from F1 to F2 layer ; traces of long term memory (**LTM**).

(e) **Top-down connections** from F2 to F1 layer; traces of long term memory (**LTM**).

(f) **Interconnections** among the nodes in each layer are not shown.

(g) **Inhibitory connection** (-ve weights) from F2 layer to gain control.

(h) **Excitatory connection** (+ve weights) from gain control to F1 and F2.

- **Orienting Subsystem**

(h) **Reset layer** for controlling the attentional subsystem overall dynamics.

(i) **Inhibitory connection** (-ve weights) from F1 layer to Reset node.

(j) **Excitatory connection** (+ve weights) from Reset node to F2 layer

27

- **Comparison F1 and Recognition F2 layers**

The comparison layer **F1** receives the binary external input, then **F1** passes the external input to recognition layer **F2** for matching it to a

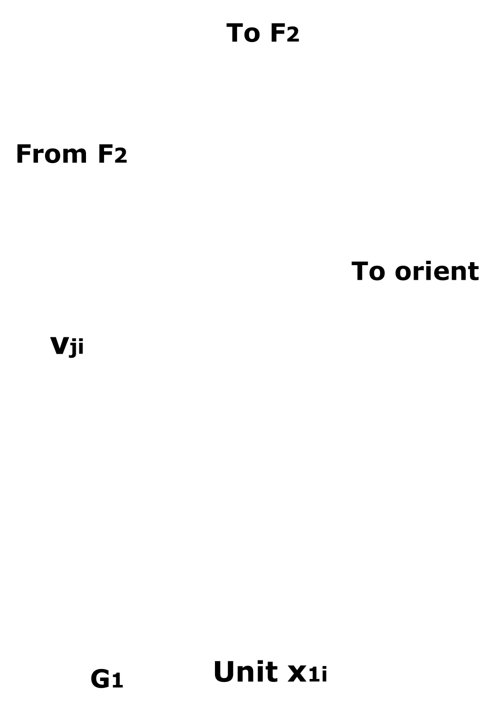
classification category. The result is passed back to **F1** to find :

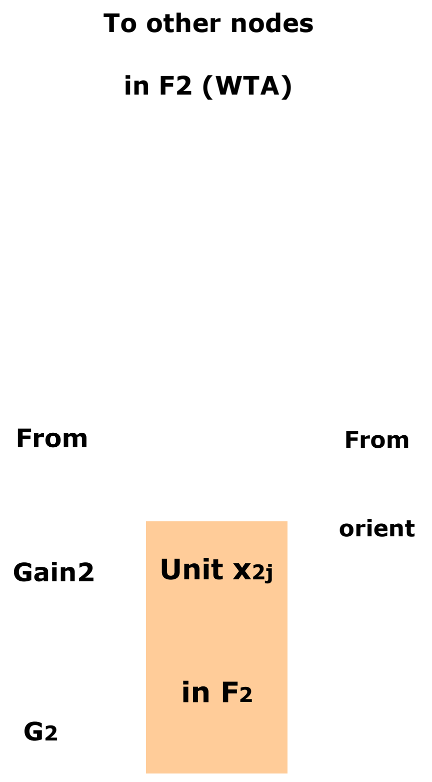
If the category matches to that of input, then

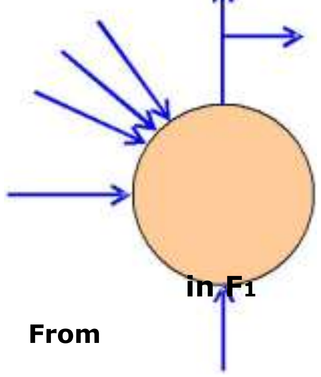
- If Yes (match) then a new input vector is read and the cycle starts again
- If No (mismatch) then the orienting system inhibits the previous category to get a new category match in **F2** layer.

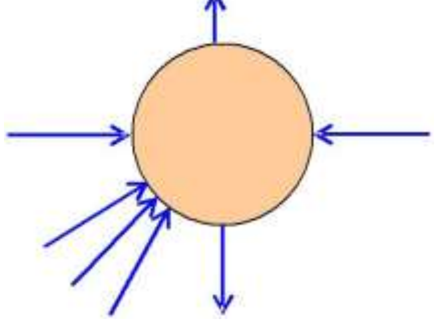
The two gains, control the activities of **F1** and **F2** layer, respectively.

Processing element x_{1i} in layer F1 **Processing element x_{2i} in layer F2**









1. A processing element X_{1i} in F_1 receives input from three sources:

(a) External input vector I_i,

(b) Gain control signal G_1

(c) Internal network input V_{ji} made of the output from F_2 multiplied appropriate connections weights.

2. There is no inhibitory input to the neuron

3. The output of the neuron is fed to the F_2 layer as well as the orienting sub-system.

1. A processing element X_{2j} in F_2 receives input from three sources:

(a) Orienting sub-system,

(b) Gain control signal **G2**

(c) Internal network input w_{ij} made of the output from F_1 multiplied appropriate connections weights.

2. There is no inhibitory input to the neuron.

3. The output of the neuron is fed to the F_1 layer as well as G_1 control.

28

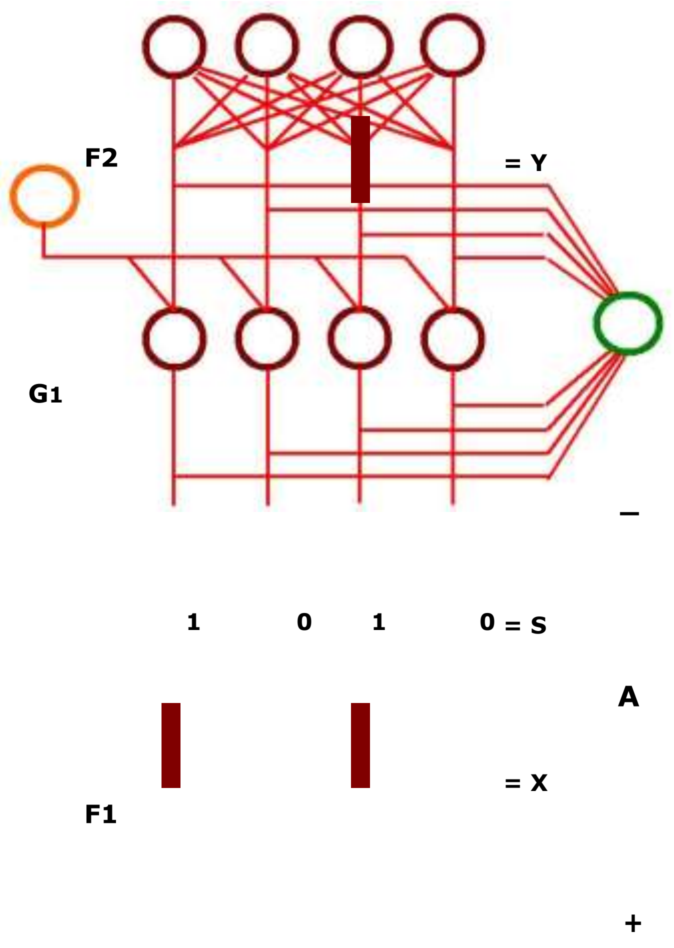
,

4.3 ART1 Pattern Matching Cycle

The ART network structure does pattern matching and tries to determine whether an input pattern is among the patterns previously stored in the network or not.

Pattern matching consists of : Input pattern presentation, Pattern matching attempts, Reset operations, and the Final recognition. The step-by-step pattern matching operations are described below.

- **Fig (a) show the input pattern presentation.** The sequence of effects are :



► Input pattern **I** presented to the units in **F1** layer. A pattern of activation **X** is produced across **F1**.

► Same input pattern **I** also excites the orientation sub-system **A** and gain control **G1**.

► Output pattern **S** (which is

1 **0** **1** **0 = I**

Fig (a) Input Pattern

inhibitory signal) is sent to **A**. It cancels the excitatory effect of signal **I** so that **A** remains inactive.

► Gain control **G1** sends an excitatory signal to **F1**. The same signal is applied to each node in **F1** layer. It is known as nonspecific signal.

► Appearance of **X** on **F1** results an output pattern **S**. It is sent through connections to **F2** which receives entire output vector **S**.

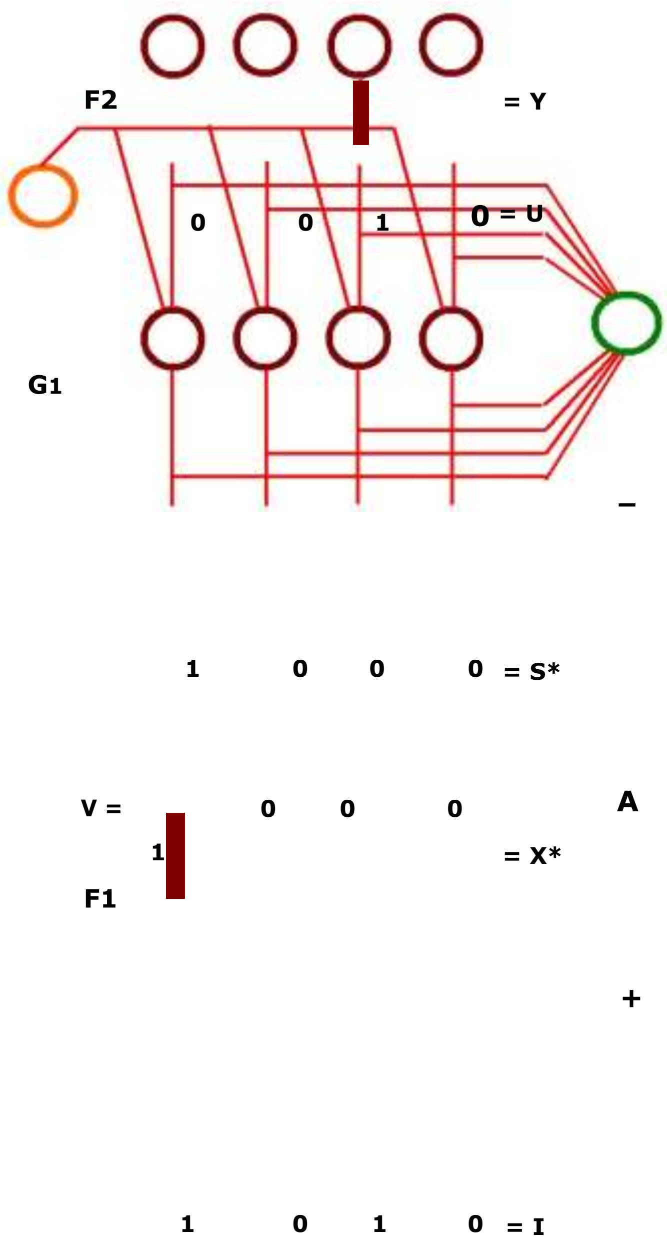
► Net values calculated in the **F2** units, as the sum the product of the input values and the connection weights.

► Thus, in response to inputs from **F1**, a pattern of activity **Y** develops across the nodes of **F2** which is a competitive layer that performs a contrast enhancement on the input signal.

29

- **Fig (b) show the Pattern Matching Attempts.**

  The sequence of operations are :



► Pattern of activity **Y** results an output **U** from **F2** which is an inhibitory signal sent to **G1**. If it receives any inhibitory signal from **F2**, it ceases activity.

► Output **U** becomes second input pattern for **F1** units. Output **U** is transformed to pattern **V**, by **LTM** traces on the top-down connections from **F2** to **F1**.

Fig (b) Pattern matching

► Activities that develop over the nodes in **F1** or **F2** layers are the **STM** traces not shown in the fig.

- **The 2/3 Rule**

► Among the three possible sources of input to **F1** or **F2**, only two are used at a time. The units on **F1** and **F2** can become active only if two out of the possible three sources of input are active. This feature is called the 2/3 rule.

► Due to the 2/3 rule, only those **F1** nodes receiving signals from both

I and **V** will remain active. So the pattern that remains on **F1** is $\mathbf{I} \cap \mathbf{V}$.

► The Fig shows patterns mismatch and a new activity pattern **X*** develops on **F1**. As the new output pattern **S*** is different from the original **S** , the inhibitory signal to **A** no longer cancels the excitation coming from input pattern **I**.

30

- **Fig (c) show the Reset Operations.**
  The sequence of effects are :

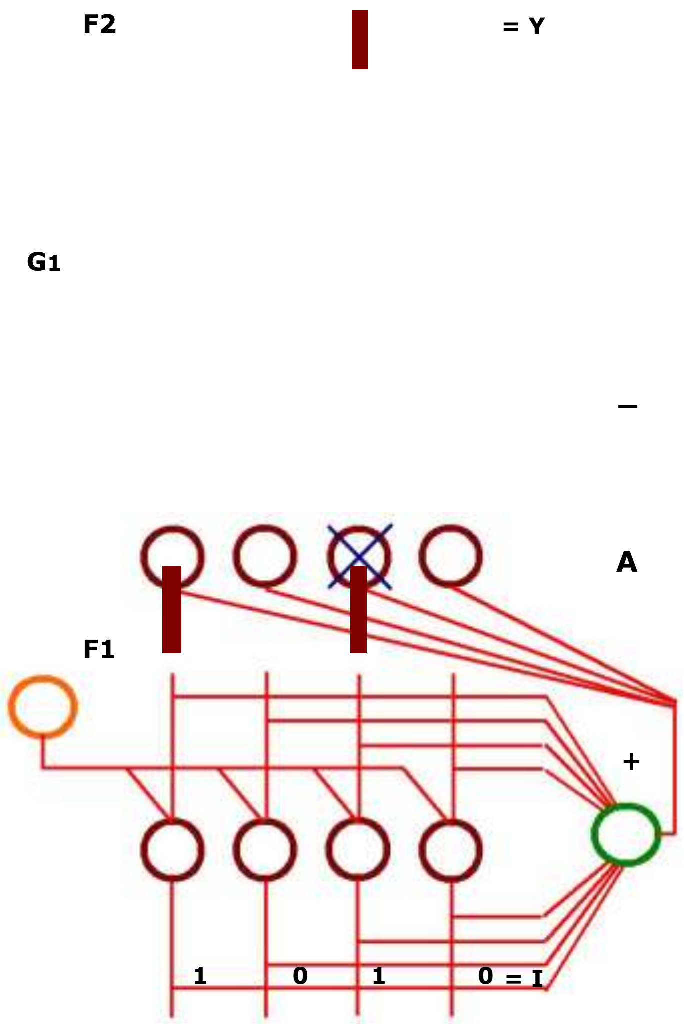


Fig (c) Reset

► Orientation sub-system **A** becomes active due to mismatch of patterns on **F1**.

► Sub-system **A** sends a non-specific reset signal to all nodes on **F2**.

► Nodes on **F2** responds according to their present state. If nodes are inactive, nodes do not respond; If nodes are active, nodes become inactive and remain for an extended period of time. This sustained inhibition prevents the same node winning the competition during the next cycle.

► Since output **Y** no longer appears, the top-down output and the inhibitory signal to **G1** also disappears.

31

- **Fig (d) show the final Recognition.**
  The sequence of operations are :

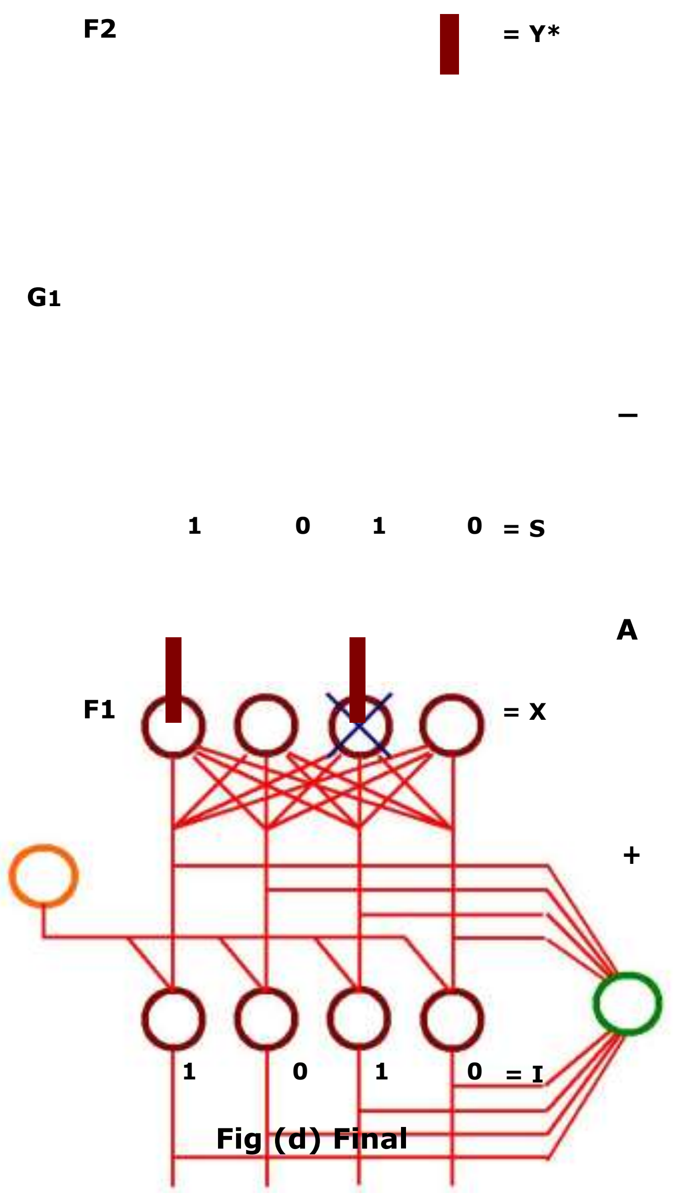


Fig (d) Final

► The original pattern **X** is reinstated on **F1** and a new cycle of pattern matching begins. Thus a new pattern **Y*** appears on **F2**.

► The nodes participating in the original pattern **Y** remains inactive due to long term effects of the reset signal from **A**.

► This cycle of pattern matching will continue until a match is found, or until **F2** runs out of previously stored patterns. If no match is found, the network will assign some uncommitted node or nodes on **F2** and will begin to learn the new pattern. Learning takes through **LTM** traces (modification of weights). This learning process does not start or stop but continue while the pattern matching process takes place. When ever signals are sent over connections, the weights associated with those connections are subject to modification.

► The mismatches do not result in loss of knowledge or learning of

incorrect association because the time required for significant changes in weights is very large compared to the time required for a complete matching cycle. The connection participating in mismatch are not active long enough to effect the associated weights seriously.

► When a match occurs, there is no reset signal and the network settles down into a resonate state. During this stable state, connections remain active for sufficiently long time so that the weights are strengthened. This resonant state can arise only when a pattern match occurs or during enlisting of new units on **F2** to store an unknown pattern.

32

4.4 ART1 Algorithm - Clustering Procedure

(Ref: Fig ART1 Architecture, Model and Pattern matching explained before)

Example **$t = 4$, $x(4) = \{1\ 0\ 0\}^T$**

- **Notations**

- ***I(X)*** is input data set ( ) of the form ***$I(X) = \{ x(1), x(2), . . , x(t) \}$*** where ***t*** represents time or number of vectors. Each **$x(t)$** has ***n*** elements;

is the 4th vector that has 3 elements .

- ***$W(t) = (w_{ij}(t))$*** is the Bottom-up weight matrix of type ***$n \times m$*** where

$i = 1, n$; $j = 1, m$; and its each column is a column vector of the form

$w_j(t) = [(w_{1j}(t) . . . . w_{ij}(t) . . . w_{nj}(t)]^T$, **T** is transpose; Example :

Each column is a column vectors of the form

$$W(t) = (w_{ij}(t)) = \begin{array}{cc} W_{j=1} & W_{j=2} \\ \downarrow & \downarrow \\ \end{array} \left[\begin{array}{cc} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{array}\right]$$

$V(t) = (v_{ji}(t))$ is the Top-down weight matrix of type ***$m \times n$*** where

$j = 1, m$; $i = 1, n$; and its each line is a column vector of the form

$v_j(t) = [(v_{j1}(t) . . . . v_{ji}(t) . . . v_{jn}(t)]^T$, **T** is transpose; Example :

Each line is a column vector of the form

$v_{j=1}$ ***$v_{j=2}$***

$$v(t) = (v_{ji}(t)) = \begin{bmatrix} v11 & v12 & v13 \\ v21 & v22 & v23 \end{bmatrix}$$

- For any two vectors ***u*** and ***v*** belong to the same vector space ***R***, say ***u, v*** $\in$ ***R*** the notation $< u , v > = u \cdot v = u^T \cdot v$ is scalar product; *and* $u \times v = (u_1 v_1 , . . . u_i v_i . . u_n v_n)^T \in R$, is piecewise product , that is

  component by component.

- The $u \wedge v \in R^n$ means component wise minimum, that is the minimum on each pair of components $min \{ u_i ; v_i \}$, $i = 1, n ;$

- The 1-norm of vector ***u*** is $||u||_1 = ||u|| = \sum_{i=1}^{n} | u_i |$

- The vigilance parameter is real value $\rho \in (0 , 1)$,
  The learning rate is real value $\alpha \in (0 , 1)$,

33

- **Step-by-Step Clustering Procedure**

Input: Feature vectors

ƒ Feature vectors $\mathbf{I}_{H=1 \text{ to } h}$, each representing input pattern to layer $\mathbf{F_1}$.

ƒ Vigilance parameter ρ ; select value between **0.3** and **0.5**.

Assign values to control gains G_1 and G_2

$$G_1 = \begin{cases} 1 & \text{if input } I_H \neq 0 \text{ and output from } F_2 \text{ layer} = 0 \\ 0 & \text{otherwise} \end{cases}$$

$$G_2 = \begin{cases} 1 & \text{if input } I_H \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Output: Clusters grouped according to the similarity is determined by ρ. Each neuron at the output layer represents a cluster, and the top-down (or backward) weights represents temp plates or prototype of the cluster.

Step - 1 (Initialization)

- Initially, no input vector **I** is applied, making the control gains,

  G1 = 0, G2 = 0. Set nodes in **F1** layer and **F2** layer to zero.

- Initialize bottom-up $w_{ij}(t)$ and top-down $v_{ji}(t)$ weights for time **t.**
  Weight w_{ij} is from neuron **i** in **F1** layer to neuron **j** in **F2** layer; where $i = 1, n$; $j = 1, m$; and weight matrix $W(t) = (w_{ij}(t))$ is of

  type **n x m.**

  Each column in **W(t)** is a column vector $w_j(t), j = 1, m$;

  $w_j(t) = [(w_{1j}(t) \ldots w_{ij}(t) \ldots w_{nj}(t)]^T$, **T** is transpose and

  $w_{ij} = 1/(n+1)$ where **n** is the size of input vector;

  Example : If **n = 3**; then $w_{ij} = 1/4$

  $$\begin{array}{lcc} \text{column vectors} & W_{j=1} & W_{j=2} \\ & W_{11} & W_{12} \\ W(t) = (w_{ij}(t)) = & W_{21} & W_{22} \\ & W_{31} & W_{32} \end{array}$$

  The v_{ji} is weight from neuron **j** in **F2** layer to neuron **i** in **F1** layer;

  where $j = 1, m$; $i = 1, n$; Weight matrix $V(t) = (v_{ji}(t))$ is of

  type **m x n.**

  Each line in **V(t)** is a column vector $v_j(t), j = 1, m$;

  $v_j(t) = [(v_{j1}(t) \ldots v_{ji}(t) \ldots v_{jn}(t)]^T$, **T** is transpose and $v_{ji} = 1$.

  Each line is a column vector $v_{j=1}$ $v_{j=2}$

v11 v12 v13

v(t) = (vji (t)) =

- Initialize the vigilance parameter, usually $\boldsymbol{0.3 \le \rho \le 0.5}$
- Learning rate α ***= 0, 9***

- Special Rule : Example

  "While indecision, then the winner is second between equal".

35

Step - 2 (Loop back from step 8)

Repeat steps 3 to 10 for all input vectors $\mathbf{I_H}$ presented to the **F1** layer; that is **I(X) = { x(1), x(2), x(3), . . . , x(t), }**

Step – 3 (Choose input pattern vector)

Present a randomly chosen input data pattern, in a format as input vector.

Time t =1,

The First the binary input pattern say **{ 0 0 1 }** is presented to the network. Then

– As input **I ≠ 0 ,** therefore node **G1 = 1** and thus activates all nodes in **F1**.

– Again, as input **I ≠ 0** and from **F2** the output $\mathbf{X_2 = 0}$ means not producing any output, therefore node **G2 = 1** and thus activates all nodes in **F2**, means recognition in **F2** is allowed.

Step - 4 (Compute input for each node in F2)

Compute input **y j** for each node in **F2** layer using :

$$y_j = \sum_{i=1}^{n} I_i \times W_{ij}$$, If **j = 1 , 2** then **y j=1** and **y j=1** are

$$y_{j=1} = \begin{bmatrix} I_1 & I_2 & I_3 \end{bmatrix} \begin{bmatrix} W_{11} \\ W_{21} \\ W_{31} \end{bmatrix} \qquad y_{j=2} = \begin{bmatrix} I_1 & I_2 & I_3 \end{bmatrix} \begin{bmatrix} W_{12} \\ W_{22} \\ W_{32} \end{bmatrix}$$

Step – 5 (Select Winning neuron)

Find **k** , the node in **F2**, that has the largest **y k** calculated in step 4.

$$y_k = \sum_{j=1}^{\text{no of nodes in } F_2} \max (y_j)$$

If an Indecision tie is noticed, then follow note stated below.

Else go to step 6.

Note :

Calculated in step 4, ***y j=1* = 1/4** and ***y j=2* = 1/4**, are equal, means an indecision tie then go by some defined special rule.

Let us say the winner is the second node between the equals, i.e., ***k* = *2***.

Perform vigilance test, for the $\mathbf{F2_k}$ output neuron , as below:

$$r = \frac{< V_k \;,\; X(t) >}{||X(t)||} = \frac{V_k^{T} \cdot X(t)}{||X(t)||}$$

If $\mathbf{r} > \rho = \mathbf{0.3}$, means resonance exists and learning starts as :

The input vector ***X(t)*** is accepted by $\mathbf{F2_{k=2}}$.

Go to step 6.

37

Step – 6 (Compute activation in F1)

For the winning node **K** in **F2** in step 5, compute activation in **F1** as

$X^*_k = (x^*_1 , x^*_2 , \cdots , x^*_{i=n})$ where $x^*_i = v_{ki} \times I_i$ is the

piecewise product component by component and $i = 1, 2, \ldots, n$. ; i.e.,

$$X^*_K = (v_{k1} I_1 , \ldots , v_{ki} I_i \ldots, v_{kn} I_n)^T$$

Step – 7 (Similarity between activation in F1 and input)

Calculate the similarity between X^*_k and input I_H using :

$$\frac{\| X^*_k \|}{\| I_H \|} = \frac{\sum_{i=1}^{n} X^*_i}{\sum_{i=1}^{n} I_i}$$

Example : If $X^*_{K=2} = \{0\ 0\ 1\}$, $I_{H=1} = \{0\ 0\ 1\}$

then similarity between X^*_k and input I_H is

$$\| X^*_{K=2} \| = \sum_{i=1}^{n} X^*_i = 1$$

$$\left\| I_{H=1} \right\| \qquad \sum_{i=1}^{n} I_i$$

38

Step – 8 (Test similarity with vigilance parameter)

Test the similarity calculated in Step 7 with the vigilance parameter:

The similarity $\dfrac{\left\| X^{*}_{K=2} \right\|}{\left(\left\| I_{H=1} \right\| \right)} = \mathbf{1}$ is $> \rho$

It means the similarity between $X^{*}_{K=2}$, $I_{H=1}$ is **true**. Therefore,

Associate Input $I_{H=1}$ with **F2** layer node **$m = k$**

(a) Temporarily disable node k by setting its activation to **0**

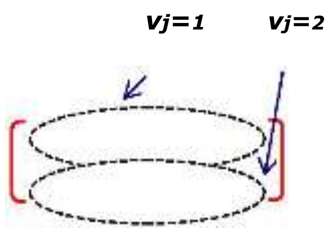
(b) Update top-down weights , $v_j(t)$ of node $j = k = 2$, from **F2** to **F1**

$$v_{k\,i}\,(new) = v_{k\,i}\,(t) \times I_i \quad \text{where } i = 1, 2, \ldots, n ,$$

(c) Update bottom-up weights , $w_j(t)$ of node $j = k$, from **F2** to **F1**

$$w_{k\,i}\,(new) = \frac{v_{k\,i}\,(new)}{0.5 + \|\, v_{k\,i}\,(new) \,\|} \quad \text{where } i = 1, 2, \ldots, n$$

(d) Update weight matrix $W(t)$ and **$V(t)$** for next input vector, time $t = 2$



$$v(t) = v(2) = (v_{ji}\,(2)) = \begin{array}{ccc} v11 & v12 & v13 \\ v21 & v22 & v23 \end{array}$$

$$W(t) = W(2) = (w_{ij}\,(t)) = \begin{array}{cc} W_{j=1} & W_{j=2} \\ \downarrow & \downarrow \\ W_{11} & W_{12} \\ W_{21} & W_{22} \\ W_{31} & W_{32} \end{array}$$

If done with all input pattern vectors t (1, n) then **STOP.**

else Repeat step 3 to 8 for next Input pattern

39

4.5 ART1 Numerical Example

- **Example : Classify in even or odd the numbers** **1, 2, 3, 4, 5, 6, 7**

Input:

The decimal numbers **1, 2, 3, 4, 5, 6, 7** given in the BCD format.

This input data is represented by the set() of the form

I(X) = { x(1), x(2), x(3), x(4), x(5), x(6), x(7) } where

Decimal nos	BCD format	Input vectors x(t)
1	0 0 1	$x(1) = \{ 0\ 0\ 1\}^T$
2	0 1 0	$x(2) = \{ 0\ 1\ 0\}^T$
3	0 1 1	$x(3) = \{ 0\ 1\ 1\}^T$
4	1 0 0	$x(4) = \{ 1\ 0\ 0\}^T$
5	1 0 1	$x(5) = \{ 1\ 0\ 1\}^T$
6	1 1 0	$x(6) = \{ 1\ 1\ 0\}^T$
7	1 1 1	$x(7) = \{ 1\ 1\ 1\}^T$

– The variable **t** is time, here the natural numbers which vary from 1 to 7, is expressed as **t = 1 , 7** .

– The **x(t)** is input vector; **t = 1, 7** represents **7** vectors.

– Each **x(t)** has 3 elements, hence input layer **F1** contains **n= 3** neurons;

– let class **A1** contains even numbers and **A2** contains odd numbers, this means , two clusters, therefore output layer **F2** contains **m = 2** neurons.

40

Step - 1 **(Initialization)**

- Initially, no input vector **I** is applied, making the control gains, **G1 = 0, G2 = 0.** Set nodes in **F1** layer and **F2** layer to zero.

- Initialize bottom-up w_{ij} **(t)** and top-down v_{ji} **(t)** weights for time **t**. Weight w_{ij} is from neuron ***i*** in **F1** layer to neuron ***j*** in **F2** layer;

  where ***i = 1, n ; j = 1, m ;*** and

  weight matrix $W(t) = (w_{ij}(t))$ is of type **n x m.**

  Each column in ***W(t)*** is a column vector $w_j(t)$, ***j = 1, m ;***

  $w_j(t) = [(w_{1j}(t) \ldots w_{ij}(t) \ldots w_{nj}(t)]^T$, **T** is transpose and

  w_{ij} **= 1/(n+1)** where ***n*** is the size of input vector;

  here ***n = 3***; so $w_{ij} = 1/4$

  column vectors $W_{j=1}$ $W_{j=2}$

$$W(t) = (w_{ij}(t)) = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \\ W_{31} & W_{32} \end{bmatrix} = \begin{bmatrix} 1/4 & 1/4 \\ 1/4 & 1/4 \\ 1/4 & 1/4 \end{bmatrix} \quad \text{where } t=1$$

  The v_{ji} is weight from neuron ***j*** in **F2** layer to neuron ***i*** in **F1** layer;

  where ***j = 1, m; i = 1, n ;***

  weight matrix $V(t) = (v_{ji}(t))$ is of type **m x n.**

  Each line in ***V(t)*** is a column vector $V_j(t)$, ***j = 1, m ;***

  $v_j(t) = [(v_{j1}(t) \ldots v_{ji}(t) \ldots v_{jn}(t)]^T$, **T** is transpose and $v_{ji} = 1$.

  Each line is a column vector $v_{j=1}$ $v_{j=2}$

$$v(t) = (v_{ji}(t)) = \begin{matrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \end{matrix} = \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} \quad \text{where } t=1$$

- Initialize

the vigilance parameter ρ = **0.3**, usually **0.3 ≤ ρ ≤ 0.5**

- Learning rate α **= 0, 9**

- Special Rule : While indecision , then the winner is second between equal.

Step - 2 **(Loop back from step 8)**

Repeat steps 3 to 10 for all input vectors $\boldsymbol{I_{H = 1 \text{ to } h=7}}$ presented to the **F1** layer; that is ***I(X) = { x(1), x(2), x(3), x(4), x(5), x(6), x(7) }***

Step – 3 **(Choose input pattern vector)**

Present a randomly chosen input data in B C D format as input vector.

Let us choose the data in natural order, say ***x(t)*** = ***x(1) = { 0 0 1 }^{T}***

Time t =1, the binary input pattern **{ 0 0 1 }** is presented to network.

– As input ***I ≠ 0 ,*** therefore node ***G1 = 1*** and thus activates all nodes in **F1**.

– Again, as input ***I ≠ 0*** and from **F2** the output ***X_2 = 0*** means not producing any output, therefore node ***G2 = 1*** and thus activates all nodes in **F2**, means recognition in **F2** is allowed.

42

Step - 4 (Compute input for each node in F2)

Compute input y_j for each node in **F2** layer using :

$$y_j = \sum_{i=1}^{n} I_i \times w_{ij}$$

$$y_{j=1} = \begin{array}{ccc} I_1 & I_2 & I_3 \end{array} \begin{array}{c} W_{11} \\ W_{21} \\ W_{31} \end{array} = \begin{array}{ccc} 0 & 0 & 1 \end{array} \begin{array}{c} 1/4 \\ 1/4 \\ 1/4 \end{array} = 1/4$$

$$y_{j=2} = \begin{array}{ccc} I_1 & I_2 & I_3 \end{array} \begin{array}{c} W_{12} \\ W_{22} \\ W_{32} \end{array} = \begin{array}{ccc} 0 & 0 & 1 \end{array} \begin{array}{c} 1/4 \\ 1/4 \\ 1/4 \end{array} = 1/4$$

Step – 5 (Select winning neuron)

Find k , the node in ***F2***, that has the largest y_k calculated in step 4.

$$y_k = \sum_{j=1}^{\text{no of nodes in } F_2} \max (y_j)$$

If an indecision tie is noticed, then follow note stated below.

Else go to step 6.

Note :

Calculated in step 4, $y_{j=1} = 1/4$ and $y_{j=2} = 1/4$, are equal, means an indecision tie. *[Go by Remarks mentioned before, how to deal with the tie]*.

Let us say the winner is the second node between the equals, i.e., ***k* = *2***.

Perform vigilance test, for the **$F2_k$** output neuron , as below:

$$r = \frac{< V_k\ ,\ X(t) >}{||X(t)||} = \frac{V_k^T \cdot X(t)}{||X(t)||} = \frac{\begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}}{\sum_{i=1}^{n} |X(t)|} = \frac{1}{1} = 1$$

Thus $r > \rho = 0.3$ **, means resonance exists and learning starts as** :

The input vector ***X(t=1)*** is accepted by **$F2_{k=2}$** , ie ***x(1)*** $\in$ **A2** cluster.

Go to Step 6.

43

Step – 6 (Compute activation in F1)

For the winning node **K** in **F2** in step 5, compute activation in **F1** as

$* =$ $x*$ $*$ . . ., $*$ where x^*_i $_{ki}$ is the

X_k ($_1$, x_2, $x_{i=n}$) $= v \; x \; I$

piecewise product component by component and $i = 1, 2, . . . , n$. ; i.e.,

$$X^*_K = (v_{k1} I_1 , . . . , v_{ki} I_i . ., v_{kn} I_n)^T$$

Accordingly $X^*_{K=2}$ = **{1 1 1} x {0 0 1}** = **{0 0 1}**

Step – 7 (Similarity between activation in F1 and input)

Calculate the similarity between X^*_k and input I_H using :

$$\frac{\| X^*_k \|}{\| I_H \|} = \frac{\sum_{i=1}^{n} X^*_i}{\sum_{i=1}^{n} I_i} \qquad \text{here } n = 3$$

Accordingly , while $X^*_{K=2}$ = {0 0 1} , $I_{H=1}$ = **{0 0 1}**

Similarity between X^*_k and input I_H is

$$\frac{\| X^*_{K=2} \|}{} = \frac{\sum_{i=1}^{n} X^*_i}{} = 1$$

$$\left\| I_{H=1} \right\| \qquad \sum_{i=1}^{n} I_i$$

44

Step – 8 (Test similarity with vigilance parameter)

Test the similarity calculated in Step 7 with the vigilance parameter:

The similarity $\frac{\| X^*_{K=2} \|}{\left(\| I_{H=1} \| \right)} = 1$ is $> \rho$

It means the similarity between $X^*_{K=2}$, $I_{H=1}$ is **true**. Therefore,

Associate Input $I_{H=1}$ with **F2** layer node $m = k = 2$, i.e., **Cluster 2**

(a) Temporarily disable node *k* by setting its activation to **0**

(b) Update top-down weights , $v_j\ (t)$ of node $j = k = 2$, from **F2** to **F1**

$v_{k\,i}\ (new) = v_{k\,i}\ (t=1) \times I_i$ where $i = 1, 2, \ldots, n = 3$,

$$v_{k=2}\ (t=2) = v_{k=2,\,i}\ (t=1) \times I_i = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$$

(c) Update bottom-up weights , $w_j\ (t)$ of node $j = k = 2$, from ***F2*** to ***F1***

$$w_{k\,i}\ (new) = \frac{v_{k\,i}\ (new)}{0.5 + \|\ v_{k\,i}\ (new)\ \|}$$ where $i = 1, 2, \ldots, n = 3$,

$w_{k=2}\ (t=2) \quad v_{k=2,\,i}\ (t=2) = 0\ 0\ 1$

$$= \frac{\qquad}{0.5 + \|v_{k=2,\,i}\,(t=2)\|} \qquad \frac{\qquad}{0.5 + \| \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \|}$$

$$= \begin{bmatrix} 0 & 0 & 2/3 \end{bmatrix}^T$$

(d) Update weight matrix *W(t)* and *V(t)* for next input vector, time *t* =2

$$\boldsymbol{v(t) = v(2) = (v_{ji}\,(2))} = \begin{array}{c|ccc} & & & \\ v_{j=1} & v_{11} & v_{12} & v_{13} \\ v_{j=2} & v_{21} & v_{22} & v_{23} \end{array} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{W(t) = W(2) = (w_{ij}\,(t))} = \begin{array}{cc} W_{j=1} & W_{j=2} \\ \hline W_{11} & W_{12} \\ W_{21} & W_{22} \\ W_{31} & W_{32} \end{array} = \begin{bmatrix} 1/4 & 0 \\ 1/4 & 0 \\ 1/4 & 2/3 \end{bmatrix}$$

If done with all input pattern vectors t (1, 7) then stop.

else Repeat step 3 to 8 for next input pattern

45

- **Present Next Input Vector and Do Next Iteration (step 3 to 8)**

Time t =2; $I_H =$ $I_2 = \{ 0\ 1\ 0 \}$;

$$W(t=2) = \begin{array}{cc} W_{j=1} & W_{j=2} \\ 1/4 & 0 \\ 1/4 & 0 \\ 1/4 & 2/3 \end{array} \qquad v(t=2) = \begin{array}{ccc} & v_{j=1} & v_{j=2} \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{array}$$

$$y_{j=1} = \begin{matrix} 0 & 1 & 0 \end{matrix} \begin{matrix} 1/4 \\ 1/4 \\ 1/4 \end{matrix} = 1/4 = 0.25$$

$$y_{j=2} = \begin{matrix} 0 & 1 & 0 \end{matrix} \begin{matrix} 0 \\ 0 \\ 2/3 \end{matrix} = 0 = 0$$

Find winning neuron, in node in F2 that has ***max(y*** $_{j=1,m}$***)*** ;

Assign ***k = j ;*** i.e., y_k = y_j **= max(1/4, 0)** ;

Decision $y_{j=1}$ is maximum, so **K = 1**

Do vigilance test , for output neuron **F2**$_{k=1}$**,**

$$r = \frac{V^T_{k=1} \cdot X(t=2)}{||X(t=2)||} = \frac{\begin{matrix} 1 & 1 & 1 \end{matrix} \begin{matrix} 0 \\ 1 \\ 0 \end{matrix}}{\sum^{n} |X(t=2)|} = \frac{1}{1} = 1$$

$[\quad]\left[\quad\right]_{i=1}$

$r >$

Resonance , since ρ = ***0.3*** , resonance exists ; So **Start learning**;

Input vector ***x(t=2)*** is accepted by ***F2*k=1** , means ***x(2)*** $\in$ **A1 Cluster.**

Compute activation in **F1,** for winning node **k = 1**, piecewise product component by component

$$X^*_{K=1} = V_{k=1,\,i} \times I_{H=2,\,i} = (v_{k1}\, I_{H1}\,,\, .\, .\, v_{ki}\, I_{Hi}\, .\,,\, v_{kn}\, I_{Hn})^T$$

$$= \{1\ 1\ 1\} \times \{0\ 1\ 0\} = \{\ 0\ 1\ 0\ \}$$

Find similarity between $X^*_{K=1} = \{0\ 1\ 0\}$ and $I_{H=2} = \{0\ 1\ 0\}$ as

$$\frac{\|X^*_{K=1}\|}{\|I_{H=2}\|} = \frac{\sum_{i=1}^{n} X^*_i}{\sum_{i=1}^{n} I_{H=2,\,i}} = 1$$

[Continued in next slide]

46

[Continued from previous slide : Time t =2]

Test the similarity calculated with the vigilance parameter:

$$\textbf{Similarity} \quad \frac{\left\| X^{*}_{K=1} \right\|}{\left\| I_{H=2} \right\|} = 1 \quad \text{is} \quad > \rho$$

It means the similarity between $X^{*}_{K=1}$, $I_{H=2}$ is **true**.

So Associate input $I_{H=2}$ with **F2** layer node $m = k = 1$, i.e., **Cluster 1**

(a) Temporarily disable node *k* = 1 by setting its activation to **0**

(b) Update top-down weights, $v_j\ (t=2)$ of node $j = k = 1$, from **F2** to **F1**

$$v_{k=1,\,i}\ (new) = v_{k=1,\,i}\ (t=2) \times I_{H=2,\,i} \quad \text{where } i = 1, 2, \ldots, n = 3\,,$$

$$v_{k=1,}\ (t=3) = v_{k=1,\,i}\ (t=2) \times I_{H=2,\,i} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T$$

(c) Update bottom-up weights, $w_j\ (t=2)$ of node $j = k = 1$, from $F1$ to $F2$

$$w_{k=1,\,i}\ (new) = \frac{v_{k=1,\,i}\ (new)}{0.5 + ||\, v_{k=1,\,i}\ (new)\, ||} \quad \text{where } i = 1, 2, \ldots, n = 3\,,$$

$$w_{k=1,}\ (t=3) = \frac{v_{k=1,\,i}\ (t=3)}{} = \frac{0\ 1\ 0}{}$$

$$0.5 + \|v_{k=1,\,i}\,(t=3)\| \qquad 0.5 + \| [\, 0\ 1\ 0 \,] \|$$

$$= [\, 0 \;\; 2/3 \;\; 0 \,]^T$$

(d) Update weight matrix *W(t)* and ***V(t)*** for next input vector, time ***t =3***

$$V(t) = v(3) = (v_{ji}\,(3)) = \begin{array}{c} \\ v_{j=1} \\ v_{j=2} \end{array} \begin{bmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$W(t) = W(3) = (w_{ij}\,(3)) = \begin{array}{cc} W_{j=1} & W_{j=2} \\ \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} \end{array} = \begin{bmatrix} 0 & 0 \\ 2/3 & 0 \\ 0 & 2/3 \end{bmatrix}$$

If done with all input pattern vectors t (1, 7) then stop.

else Repeat step 3 to 8 for next input pattern

47

- **Present Next Input Vector and Do Next Iteration (step 3 to 8)**

Time t = 3; **I**$_H$ = **I**$_3$ = **{ 0 1 1 }** ;

$$W(t=3) = \begin{array}{cc} W_{j=1} & W_{j=2} \\ \left[\begin{matrix} 0 & 0 \\ 2/3 & 0 \\ 0 & 2/3 \end{matrix}\right] \end{array} \qquad v(t=3) = \begin{array}{c} v_{j=1} \; v_{j=2} \\ \left[\begin{matrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}\right] \end{array}$$

$$y_{j=1} = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 2/3 \\ 0 \end{bmatrix} = 2/3 = 0.666$$

$$y_{j=2} = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 2/3 \end{bmatrix} = 2/3 = 0.666$$

Find winning neuron, in node in F2 that has ***max(y*** $_{j=1,\,m}$***)*** ;

Assign ***k = j ;*** i.e., ***y*** $_k$ = ***y*** $_j$ **= max(2/3, 2/3)** *;* indecision tie*;* take winner as second; **j = K = 2**

Decision **K = 2**

Do vigilance test , for output neuron **F2**$_{k=2}$**,**

0 0 1 0

$$r = \frac{V^T_{k=2} \cdot X(t=3)}{||X(t=3)||} = \frac{\begin{matrix}1\\1\end{matrix}}{\sum_{i=1}^{n} |X(t=3)|} = \frac{1}{2} = 0.5$$

Resonance , since $r > \rho = \mathbf{0.3}$, resonance exists ; So **Start learning**;

Input vector ***x(t=3)*** is accepted by ***F2*k=2** , means ***x(3)*** $\in$ **A2 Cluster**.

Compute activation in **F1,** for winning node **k = 2**, piecewise product component by component

$$X^*_{K=2} = V_{k=2,\,i} \ \text{x}\ I_{H=3,\,i} = (v_{k1}\, I_{H1}\, , \, . \, . \; v_{ki}\, I_{Hi}\, . \, , \; v_{kn}\, I_{Hn})^T$$

$$= \{0\ 0\ 1\}\ \text{x}\ \{0\ 1\ 1\} = \{\ 0\ 0\ 1\ \}$$

Find similarity between $X^*_{K=2} = \{0\ 1\ 0\}$ and $I_{H=3} = \{0\ 1\ 1\}$ as

$$\frac{\| X^*_{K=2} \|}{\| I_{H=3} \|} = \frac{\sum_{i=1}^{n} X^*_i}{\sum_{i=1}^{n} I_{H=3,\,i}} = 1/2 = 0.5$$

[Continued in next slide]

48

[Continued from previous slide : Time t =3]

Test the similarity calculated with the vigilance parameter:

$$\textbf{Similarity} \quad \frac{\left\| \overset{*}{X}_{K=1} \right\|}{\left[\left\| I_{H=2} \right\|\right]} = \textbf{0.5} \quad \textbf{is} \quad > \rho$$

It means the similarity between $\overset{*}{X}_{K=2}$, $I_{H=3}$ is **true**.

So Associate input $I_{H=3}$ with **F2** layer node $m = k = 2$, i.e., **Cluster 2**

(a) Temporarily disable node *k* = 2 by setting its activation to **0**

(b) Update top-down weights, $v_j\ (t=3)$ of node $j = k = 2$, from **F2** to **F1**

$$v_{k=2,\,i}\ (new) = v_{k=2,\,i}\ (t=3) \times I_{H=3,\,i} \quad \text{where } i = 1, 2, \ldots, n = 3\,,$$

$$v_{k=2,}\ (t=4) = v_{k=2,\,i}\ (t=3) \times I_{H=3,\,i} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$$

(c) Update bottom-up weights, $w_j\ (t=3)$ of node $j = k = 2$, from $F1$ to $F2$

$$w_{k=2,\,i}\ (new) = \frac{v_{k=2,\,i}\ (new)}{0.5 + ||\ v_{k=2,\,i}\ (new)\ ||} \quad \text{where } i = 1, 2, \ldots, n = 3\,,$$

$$w_{k=2,}\ (t=4) = \frac{v_{k=2,\,i}\ (t=4)}{} = \frac{0\ 0\ 1}{}$$

$0.5 + ||v_{k=2,\, i}\ (t=4)||$ $\quad$ $0.5 + ||[0\ 0\ 1]||$

$= [0\ \ 0\ \ 2/3]^T$

(d) Update weight matrix *W(t)* and ***V(t)*** for next input vector, time ***t =4***

$$V(t) = v(3) = (v_{ji}\,(3)) = \begin{array}{c|ccc} & & & \\ v_{j=1} & v_{11} & v_{12} & v_{13} \\ v_{j=2} & v_{21} & v_{22} & v_{23} \end{array} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$W(t) = W(3) = (w_{ij}\,(3)) = \begin{bmatrix} W_{j=1} & W_{j=2} \\ W_{11} & W_{12} \\ W_{21} & W_{22} \\ W_{31} & W_{32} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 2/3 & 0 \\ 0 & 2/3 \end{bmatrix}$$

If done with all input pattern vectors t (1, 7) then stop.

else Repeat step 3 to 8 for next input pattern

49

- **Present Next Input Vector and Do Next Iteration (step 3 to 8)**

Time t = 4; $I_H = I_4 = \{ 1\ 0\ 0 \}$;

$$W(t=3) = \begin{array}{cc} W_{j=1} & W_{j=2} \\ 0 & 0 \\ 2/3 & 0 \\ 0 & 2/3 \end{array} \qquad V(t=3) = \begin{array}{ccc} & v_{j=1} & v_{j=2} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}$$

$$y_{j=1} = \begin{array}{ccc} 1 & 0 & 0 \end{array} \begin{array}{c} 0 \\ 2/3 \\ 0 \end{array} = 0 \qquad y_{j=2} = \begin{array}{ccc} 1 & 0 & 0 \end{array} \begin{array}{c} 0 \\ 0 \\ 2/3 \end{array} = 0$$

Find winning neuron, in node in F2 that has ***max(y*** $_{j=1,m}$***)*** ;

Assign ***k = j*** for ***y*** $_j$ **= max(0, 0)** *;* Indecision tie*;* Analyze both cases

Case 1 : Take winner as first; **j = K = 1**; Decision **K = 1**

Do vigilance test , for output neuron **F2**$_{k=1}$,

$$r = \frac{V^T_{k=1} \cdot X(t=4)}{||X(t=4)||} = \frac{\begin{array}{ccc} 0 & 1 & 0 \end{array} \begin{array}{c} 1 \\ 0 \\ 0 \end{array}}{\sum_{i=1}^{n} |X(t=4)|} = \frac{0}{1} = 0$$

$r < \rho$

Resonance , since = ***0.3*** , no resonance exists *;*

Input vector ***x(t=4)*** is not accepted by ***F2*****k=1**, means ***x(4)*** ∉ **A1** Cluster.

Put Output **O_1(t = 4) = 0**.

Case 2 : Take winner as second ; **j = K = 2**; Decision **K = 2**

Do vigilance test , for output neuron **F2k=2**,

$$r = \frac{V^T_{k=2} \cdot X(t=4)}{||X(t=4)||} = \frac{\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}}{\sum_{i=1}^{n} |X(t=4)|} = \frac{0}{1} = 0$$

Resonance , since **r <** ρ = ***0.3*** , no resonance exists *;*

Input vector ***x(t=4)*** is not accepted by ***F2*****k=2**, means ***x(4)*** ∉ **A2** Cluster.

Put Output **O_2(t = 4) = 0**.

Thus Input vector ***x(t=4)*** is **Rejected** by **F2** layer.

[Continued in next slide]

[Continued from previous slide : Time t =4]

Update weight matrix *W(t)* and ***V(t)*** for next input vector, time ***t =5***

$$W(4) = W(3)\ ;\ \ V(4) = V(3)\ ;\ \ O_{(t\,=\,4)} = \{\ 1\ \ \ 1\ \}^T$$

vj=1 ***vj=2***

$$V(t) = v(4) = (v_{ji}\ (4)) = \begin{matrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \end{matrix} = \begin{matrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$$

$$W(t) = W(4) = (w_{ij}\ (4)) = \begin{array}{cc} W_{j=1} & W_{j=2} \\ W_{11} & W_{12} \\ W_{21} & W_{22} \\ W_{31} & W_{32} \end{array} = \begin{bmatrix} 0 & 0 \\ 2/3 & 0 \\ 0 & 2/3 \end{bmatrix}$$

If done with all input pattern vectors t (1, 7) then stop.

else Repeat step 3 to 8 for next input pattern

51

- **Present Next Input Vector and Do Next Iteration (step 3 to 8)**

Time t =5; **I**$_H$ = **I**$_5$ = **{ 1 0 1 }** ;

$$W(t=5) = \begin{array}{cc} W_{j=1} & W_{j=2} \\ 0 & 0 \\ 2/3 & 0 \\ 0 & 2/3 \end{array} \qquad v(t=5) = \begin{array}{ccc|c} 0 & 1 & 0 & v_{j=1} \\ 0 & 0 & 1 & v_{j=2} \end{array}$$

$$y_{j=1} = \begin{pmatrix} 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 2/3 \\ 0 \end{pmatrix} = 0 \qquad y_{j=2} = \begin{pmatrix} 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 2/3 \end{pmatrix} = 2/3$$

Find winning neuron, in node in F2 that has ***max(y*** $_{j=1,m}$***)*** ;

Assign ***k = j ;*** i.e., $y_k = y_j$ **= max(0, 2/3)**

Decision $y_{j=2}$ is maximum, so **K = 2**

Do vigilance test , for output neuron **F2**$_{k=2}$,

$$r = \frac{V^T_{k=2} \cdot X(t=5)}{||X(t=5)||} = \frac{\begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}}{\sum_{i=1}^{n} |X(t=5)|} = \frac{1}{2} = 0.5$$

Resonance , since **r >** ρ = ***0.3*** , resonance exists *;* So **Start learning**;

Input vector $\boldsymbol{x(t=5)}$ is accepted by $\boldsymbol{F2_{k=2}}$, means $\boldsymbol{x(5)} \in$ **A2 Cluster.**

Compute activation in **F1,** for winning node **k = 2,** piecewise product component by component

$$X^*_{K=2} = V_{k=2,\,i} \times I_{H=5,\,i} = (v_{k1}\, I_{H1}\, , \ldots\, v_{ki}\, I_{Hi}\, . , \, v_{kn}\, I_{Hn})^T$$

$$= \{0\ 0\ 1\} \times \{1\ 0\ 1\} = \{\ 0\ 0\ 1\ \}$$

Find similarity between $X^*_{K=2} = \{0\ 0\ 1\}$ and $I_{H=5} = \{1\ 0\ 1\}$ as

$$\frac{\left\| X^*_{K=2} \right\|}{\left\| I_{H=5} \right\|} = \frac{\sum_{i=1}^{n} X^*_i}{\sum_{i=1}^{n} I_{H=5,\,i}} = 1/2 = 0.5$$

[Continued in next slide]

52

[Continued from previous slide : Time t =5]

Test the similarity calculated with the vigilance parameter:

$$\textbf{Similarity} \quad \frac{\| X^*_{K=1} \|}{\| I_{H=2} \|} = \mathbf{0.5} \quad \textbf{is} \quad > \rho$$

It means the similarity between $X^*_{K=2}$, $I_{H=5}$ is **true**.

So Associate input $I_{H=5}$ with **F2** layer node ***m = k = 2,*** i.e., **Cluster 2**

(a) Temporarily disable node *k* = 2 by setting its activation to **0**

(b) Update top-down weights, ***vj (t=5)*** of node ***j = k* = 2**, from **F2** to **F1**

$$v_{k=2,\, i}\ (new) = v_{k=2,\, i}\ (t=5) \times I_{H=5,\, i} \quad \text{where } i = 1, 2, \ldots, n = 3,$$

$$v_{k=2,}\ (t=6) = v_{k=2,\, i}\ (t=5) \times I_{H=5,\, i} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$$

(c) Update bottom-up weights, ***wj (t=5)*** of node ***j = k* = 2**, from ***F1*** to ***F2***

$$w_{k=2,\, i}\ (new) = \frac{v_{k=2,\, i}\ (new)}{0.5 + \| v_{k=2,\, i}\ (new) \|} \quad \text{where } i = 1, 2, \ldots, n = 3,$$

$$w_{k=2,}\ (t=6) = \frac{v_{k=2,\, i}\ (t=6)}{} = \frac{0\ 0\ 1}{}$$

$$= \frac{\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}}{0.5 + ||v_{k=2,i}(t=5)||} = \frac{\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}}{0.5 + ||\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}||} = \begin{bmatrix} 0 & 0 & 2/3 \end{bmatrix}^T$$

(d) Update weight matrix *W(t)* and *V(t)* for next input vector, time ***t =6***

$$V(t) = v(6) = (v_{ji}(6)) = \begin{array}{c|ccc} & & & \\ v_{j=1} & v_{11} & v_{12} & v_{13} \\ v_{j=2} & v_{21} & v_{22} & v_{23} \end{array} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$W(t) = W(6) = (w_{ij}(6)) = \begin{array}{cc} W_{j=1} & W_{j=2} \\ \hline W_{11} & W_{12} \\ W_{21} & W_{22} \\ W_{31} & W_{32} \end{array} = \begin{bmatrix} 0 & 0 \\ 2/3 & 0 \\ 0 & 2/3 \end{bmatrix}$$

If done with all input pattern vectors t (1, 7) then stop.

else Repeat step 3 to 8 for next input pattern

53

- **Present Next Input Vector and Do Next Iteration (step 3 to 8)**

Time t =6; $I_H = I_6 = \{ 1\ 1\ 0 \}$;

$$W(t=6) = \begin{array}{cc} W_{j=1} & W_{j=2} \\ 0 & 0 \\ 2/3 & 0 \\ 0 & 2/3 \end{array} \qquad v(t=6) = \begin{array}{ccc} & v_{j=1} & v_{j=2} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}$$

$$y_{j=1} = \begin{pmatrix} 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 2/3 \\ 0 \end{pmatrix} = 2/3 \qquad y_{j=2} = \begin{pmatrix} 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 2/3 \end{pmatrix} = 0$$

Find winning neuron, in node in F2 that has $\max(y_{j=1,m})$;

Assign $k = j$; i.e., $y_k = y_j$ = **max(2/3 , 0)**

Decision $y_{j=1}$ is maximum, so **K = 1**

Do vigilance test , for output neuron **F2**$_{k=1}$,

$$r = \frac{V^T_{k=1} \cdot X(t=6)}{||X(t=6)||} = \frac{\begin{pmatrix} 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}}{\sum_{i=1}^{n} |X(t=6)|} = \frac{1}{2} = 0.5$$

Resonance , since **r >** $\rho = 0.3$, resonance exists ; So **Start learning**;

Input vector $\boldsymbol{x(t=6)}$ is accepted by $\boldsymbol{F2_{k=1}}$, means $\boldsymbol{x(6)} \in$ **A1 Cluster.**

Compute activation in **F1,** for winning node **k = 1,** piecewise product component by component

$$X^*_{K=1} = V_{k=1,\,i} \times I_{H=6,\,i} = (V_{k1}\, I_{H1}\, , \ldots V_{ki}\, I_{Hi} \ldots , V_{kn}\, I_{Hn})^T$$

$$= \{0\ 1\ 0\} \times \{1\ 1\ 0\} = \{\ 0\ 1\ 0\ \}$$

Find similarity between $X^*_{K=1} = \{0\ 1\ 0\}$ and $I_{H=6} = \{1\ 1\ 0\}$ as

$$\frac{\| X^*_{K=1} \|}{\| I_{H=6} \|} = \frac{\sum_{i=1}^{n} X^*_i}{\sum_{i=1}^{n} I_{H=5,\,i}} = 1/2 = 0.5$$

[Continued in next slide]

54

[Continued from previous slide : Time t =6]

Test the similarity calculated with the vigilance parameter:

$$\textbf{Similarity} \quad \frac{\| X^*_{K=1} \|}{\| I_{H=6} \|} = 0.5 \quad \textbf{is} \; > \; \rho$$

It means the similarity between $X^*_{K=1}$, $I_{H=6}$ is **true**.

So Associate input $I_{H=6}$ with **F2** layer node $m = k = 1$, i.e., **Cluster 1**

(a) Temporarily disable node *k* = 1 by setting its activation to **0**

(b) Update top-down weights, $v_j\ (t=6)$ of node $j = k = 2$, from **F2** to **F1**

$$v_{k=1,\,i}\ (new) = v_{k=1,\,i}\ (t=6) \times I_{H=6,\,i} \quad \text{where } i = 1, 2, \ldots, n = 3,$$

$$v_{k=1,}\ (t=7) = v_{k=1,\,i}\ (t=6) \times I_{H=6,\,i} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T$$

(c) Update bottom-up weights, $w_j\ (t=2)$ of node $j = k = 1$, from $F1$ to $F2$

$$w_{k=1,\,i}\ (new) = \frac{v_{k=1,\,i}\ (new)}{0.5 + \|\ v_{k=1,\,i}\ (new)\ \|} \quad \text{where } i = 1, 2, \ldots, n = 3,$$

$$w_{k=1,}\ (t=7) = \frac{v_{k=1,\,i}\ (t=7)}{\quad} = \frac{0\ 1\ 0}{\quad}$$

$$= \frac{v_{k=1,i}(t=7)}{0.5 + ||v_{k=1,i}(t=7)||} = \frac{[0\ 1\ 0]}{0.5 + ||[0\ 1\ 0]||}$$

$$= \begin{bmatrix} 0 & 2/3 & 0 \end{bmatrix}^T$$

- **Update weight matrix *W(t)*** and ***V(t)*** for next input vector, time ***t =7***

$$V(t) = v(7) = (v_{ji}(7)) = \begin{array}{c|ccc} v_{j=1} & v_{11} & v_{12} & v_{13} \\ v_{j=2} & v_{21} & v_{22} & v_{23} \end{array} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$W(t) = W(7) = (w_{ij}(7)) = \begin{array}{cc} W_{j=1} & W_{j=2} \\ \hline W_{11} & W_{12} \\ W_{21} & W_{22} \\ W_{31} & W_{32} \end{array} = \begin{bmatrix} 0 & 0 \\ 2/3 & 0 \\ 0 & 2/3 \end{bmatrix}$$

If done with all input pattern vectors t (1, 7) then stop.

else Repeat step 3 to 8 for next input pattern

55

- **Present Next Input Vector and Do Next Iteration (step 3 to 8)**

Time t =7; $I_H = I_7 = \{ 1\ 1\ 1 \}$;

$$W(t=7) = \begin{array}{cc} W_{j=1} & W_{j=2} \\ \hline 0 & 0 \\ 2/3 & 0 \\ 0 & 2/3 \end{array} \qquad v(t=7) = \begin{array}{ccc} & v_{j=1} & v_{j=2} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}$$

$$y_{j=1} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 2/3 \\ 0 \end{bmatrix} = 2/3 \qquad y_{j=2} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 2/3 \end{bmatrix} = 2/3$$

Find winning neuron, in node in F2 that has $\max(y_{j=1,m})$;

Assign $k = j$; i.e., $y_k = y_j = \max(2/3\ ,\ 2/3)$; indecision tie;

take winner as second; **j = K = 2**

Decision **K = 2**

Do vigilance test , for output neuron $F2_{k=1}$,

$$r = \frac{V^T_{k=2} \cdot X(t=7)}{||X(t=7)||} = \frac{\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}}{n} = \frac{1}{3} = 0.333$$

Σ $|X(t=7)|$

$i=1$

r >

Resonance , since ρ = ***0.3*** , resonance exists ; So **Start learning**;

Input vector ***x(t=7)*** is accepted by ***F2*k=2** , means ***x(7)*** $\in$ **A2 Cluster.**

Compute activation in **F1,** for winning node **k = 2,** piecewise product component by component

$X^*_{K=2} = V_{k=2,\, i} \times I_{H=7,\, i} \quad = \quad (V_{k1}\, I_{H1}, \ldots V_{ki}\, I_{Hi} \ldots, V_{kn}\, I_{Hn})^T$

$= \{0\ 0\ 1\} \times \{1\ 1\ 1\} = \{\ 0\ 0\ 1\ \}$

Find similarity between $X^*_{K=2} = \{0\ 0\ 1\}$ and $I_{H=7} = \{1\ 1\ 1\}$ as

$$\frac{\| X^*_{K=2} \|}{\| I_{H=7} \|} = \frac{\sum_{i=1}^{n} X^*_i}{\sum_{i=1}^{n} I_{H=7,\, i}} = 1/3 = 0.333$$

[Continued in next slide]

56

[Continued from previous slide : Time t =7]

Test the similarity calculated with the vigilance parameter:

$$\textbf{Similarity} \quad \frac{\| X^*_{K=2} \|}{\left(\| I_{H=7} \| \right)} = \mathbf{0.333} \quad \textbf{is} \quad > \rho$$

It means the similarity between $X^*_{K=2}$, $I_{H=7}$ is **true**.

So Associate input $I_{H=7}$ with **F2** layer node $m = k = 2$, i.e., **Cluster 2**

- **Temporarily disable node *k* = 2** by setting its activation to **0**
- **Update top-down weights**, $v_j\ (t=7)$ of node $j = k = 2$, from **F2** to **F1**

$$v_{k=2,\, i}\ (new) = v_{k=2,\, i}\ (t=7) \times I_{H=7,\, i} \quad \text{where } i = 1, 2, \ldots, n = 3,$$

$$v_{k=2,\, (t=8)} = v_{k=2,\, i}\ (t=7) \times I_{H=7,\, i} = \begin{bmatrix} 0\ 0\ 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 0\ 0\ 1 \end{bmatrix}^T$$

- **Update bottom-up weights**, $w_j\ (t=7)$ of node $j = k = 1$, from $F1$ to $F2$

$$w_{k=2,\, i}\ (new) = \frac{v_{k=2,\, i}\ (new)}{0.5 + \| v_{k=2,\, i}\ (new) \|} \quad \text{where } i = 1, 2, \ldots, n = 3,$$

$$w_{k=2,\, (t=8)} = \frac{v_{k=2,\, i}\ (t=8)}{} = \frac{0\ 0\ 1}{}$$

$$0.5 + ||v_{k=2,\,i}\,(t=8)|| \qquad 0.5 + ||\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}||$$

$$= \begin{bmatrix} 0 & 0 & 2/3 \end{bmatrix}^T$$

- **Update weight matrix *W(t)*** and ***V(t)*** for next input vector, time ***t =8***

$$V(t) = v(8) = (v_{ji}\,(8)) = \begin{array}{c|ccc} & & & \\ v_{j=1} & v_{11} & v_{12} & v_{13} \\ v_{j=2} & v_{21} & v_{22} & v_{23} \end{array} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$W(t) = W(8) = (w_{ij}\,(8)) = \begin{array}{cc} W_{j=1} & W_{j=2} \\ \hline W_{11} & W_{12} \\ W_{21} & W_{22} \\ W_{31} & W_{32} \end{array} = \begin{bmatrix} 0 & 0 \\ 2/3 & 0 \\ 0 & 2/3 \end{bmatrix}$$

If done with all input pattern vectors t (1, 7) then STOP.

else Repeat step 3 to 8 for next input pattern

57

[Continued from previous slide]

■ **Remarks**

The decimal numbers
format (patterns) have **1, 2, 3, 4, 5, 6, 7** given in the BCD
as even or odd.
been classified into two clusters (classes)

Cluster Class **A_1 = { X(t=2), X(t=2) }**

Cluster Class **A_2 = { X(t=1), X(t=3) , X(t=3) , X(t=3) }**

The network failed to classify **X(t=4)** and rejected it.

The network has learned by the :

– Top down weight matrix ***V(t)*** and

– Bottom up weight matrix ***W(t)***

These two weight matrices, given below, were arrived after all, 1 to 7, patterns were one-by-one input to network that adjusted the weights following the algorithm presented.

$$\begin{array}{cc} v_{j=1} & v_{j=2} \end{array}$$

$$V(t) = v(8) = (v_{ji}(8)) = \begin{array}{ccc} v_{11} & v_{12} & v_{13} \end{array} = \begin{array}{ccc} 0 & 1 & 0 \end{array}$$

$$\begin{array}{ccc} v_{21} & v_{22} & v_{23} \\ \multicolumn{3}{c}{\left[W_{j=1} \quad W_{j=2} \right]} \end{array} \qquad \begin{array}{ccc} 0 & 0 & 1 \\ \multicolumn{3}{c}{\left[\qquad\qquad \right]} \end{array}$$

$$W(t) = W(8) = (w_{ij}(8)) = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 2/3 & 0 \\ 0 & 2/3 \end{pmatrix}$$

58

4.3 ART2

The Adaptive Resonance Theory (ART) developed by Carpenter and Grossberg designed for clustering binary vectors, called ART1 have been illustrated in the previous section.

They later developed ART2 for clustering continuous or real valued vectors. The capability of recognizing analog patterns is significant enhancement to the system. The differences between ART2 and ART1 are :

- The modifications needed to accommodate patterns with continuous-valued components.
- The F1 field of ART2 is more complex because continuous-valued input vectors may be arbitrarily close together. The F1 layer is split into several sunlayers.
- The F1 field in ART2 includes a combination of normalization and noise suppression, in addition to the comparison of the bottom-up and top-down signals needed for the reset mechanism.
- The orienting subsystem also to accommodate real-valued data.

The learning laws of ART2 are simple though the network is complicated.

Fuzzy Set Theory

What is Fuzzy Set ?

- The word "fuzzy" means "vagueness". Fuzziness occurs when the boundary of a piece of information is not clear-cut.

- Fuzzy sets have been introduced by Lotfi A. Zadeh (1965) as an extension of the classical notion of set.

- Classical set theory allows the membership of the elements in the set in binary terms, a bivalent condition - an element either belongs or does not belong to the set.

  Fuzzy set theory permits the gradual assessment of the membership of elements in a set, described with the aid of a membership function valued in the real unit interval [0, 1].

- **Example:**

  Words like young, tall, good, or high are fuzzy.

  - There is no single quantitative value which defines the term young.
  - For some people, age 25 is young, and for others, age 35 is young.
  - The concept young has no clean boundary.
  - Age 1 is definitely young and age 100 is definitely not young;
  - Age 35 has some possibility of being young and usually depends on the context in which it is being considered.

3. Introduction

In real world, there exists much fuzzy knowledge;

Knowledge that is vague, imprecise, uncertain, ambiguous, inexact, or probabilistic in nature.

Human thinking and reasoning frequently involve fuzzy information, originating from inherently inexact human concepts. Humans, can give satisfactory answers, which are probably true.

However, our systems are unable to answer many questions. The reason is, most systems are designed based upon classical set theory and two-valued logic which is unable to cope with unreliable and incomplete information and give expert opinions.

We want, our systems should also be able to cope with unreliable and incomplete information and give expert opinions. Fuzzy sets have been able provide solutions to many real world problems.

Fuzzy Set theory is an extension of classical set theory where elements have degrees of membership.

- **Classical Set Theory**

A Set is any well defined collection of objects. An object in a set is called an element or member of that set.

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

– Classical set theory enumerates all its elements using

$$\mathbf{A = \{ a_1 , a_2 , a_3 , a_4 , ... a_n \}}$$

If the elements $\mathbf{a_i}$ $\mathbf{(i = 1, 2, 3, . . . n)}$ of a set **A** are subset of universal set **X**, then set **A** can be represented for all elements $\mathbf{x} \in \mathbf{X}$ by its **characteristic function**

$$\propto_{\mathbf{A}} \mathbf{(x)} = \begin{cases} \mathbf{1} \quad \textbf{if} \quad \mathbf{x} \in \mathbf{X} \\ \\ \textbf{otherwise} \end{cases}$$

– A set **A** is well described by a function called characteristic function.

This function, defined on the universal space **X**, assumes :

a value of **1** for those elements **x** that belong to set **A**, and

- value of **0** for those elements **x** that do not belong to set **A**.

The notations used to express these mathematically are

$$\left.\begin{array}{l} : X \rightarrow [0, 1] \\ A(x) = 1 , x \text{ is a member of } A \end{array}\right\} \quad \text{Eq.(1)}$$

$$A(x) = 0 , x \text{ is not a member of } A$$

Alternatively, the set **A** can be represented for all elements $x \in X$

by its characteristic function $\propto_A (x)$ defined as

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

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

05

■ Fuzzy Set Theory

Fuzzy set theory is an extension of classical set theory where elements have varying degrees of membership. A logic based on the two truth values, *True* and *False*, is sometimes inadequate when describing human reasoning. Fuzzy logic uses the whole interval between **0** (false) and **1** (true) to describe human reasoning.

- A **Fuzzy Set** is any set that allows its members to have different degree of membership, called **membership function**, in the interval **[0 , 1]**.

- The **degree of membership** or truth is not same as probability;

  0 fuzzy truth is not likelihood of some event or condition.

  1 fuzzy truth represents membership in vaguely defined sets;

- **Fuzzy logic** is derived from fuzzy set theory dealing with reasoning that is approximate rather than precisely deduced from classical predicate logic.

- Fuzzy logic is capable of handling inherently imprecise concepts.

- Fuzzy logic allows in linguistic form the set membership values to imprecise concepts like **"slightly", "quite" and "very".**

- Fuzzy set theory defines Fuzzy Operators on Fuzzy Sets.

- **Crisp and Non-Crisp Set**

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

  Such sets are **crisp sets.**

  – For Non-crisp sets the characteristic function $\propto_A(x)$ can be defined.

  - The characteristic function $\propto_A(x)$ of Eq. (2) for the crisp set is generalized for the Non-crisp sets.
  - This generalized characteristic function $\propto_A(x)$ 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.

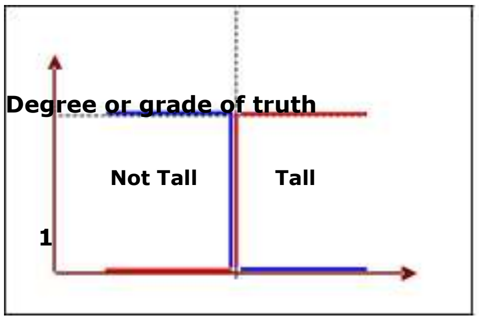
- **Representation of Crisp and Non-Crisp Set**

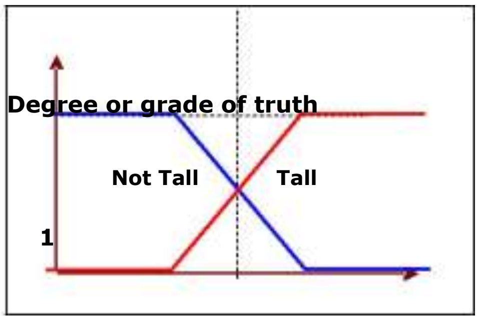
  Example : Classify students for a basketball team
  This example explains the grade of truth value.

  - **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.





0 0

1.8 m Height x 1.8 m Height x

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 ∝A (x)** of **A.**

  This membership function associates to each element **x**σ ∈ **X** a

  number as ∝**A (x**σ **)** in the closed unit interval **[0, 1].**

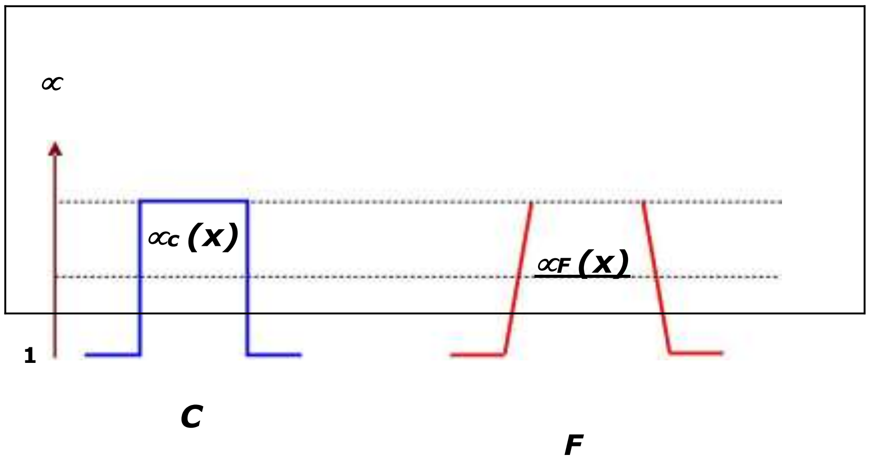
  The number ∝**A (x**σ **)** represents the **degree of membership** of **x**σ in **A.**

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

  : X → [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



0.5

0 x

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 ⊆ Fuzzy Sets**

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

[Continued in next slide]

■ **Examples of Crisp and Non-Crisp Set**

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

If we consider space **X** consisting of natural numbers $\leq$ **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.**

2. Fuzzy Set

A Fuzzy Set is any set that allows its members to have different degree of membership, called membership function, in the interval **[0 , 1]**.

- **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(u) = 0 for u >= 9.**
SMALL(6) = 0.3,	
SMALL(7) = 0.2,	
SMALL(8) = 0.1,	

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**.

11

- **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}]

12

2.1 Fuzzy Membership

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)}**.

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 **degree of membership** of the element **x** in a fuzzy set **A**.

The Graphic Interpretation of fuzzy membership for the fuzzy sets : Small, Prime Numbers, Universal-space, Finite and Infinite UniversalSpace, and Empty are illustrated in the next few slides.

- **Graphic Interpretation of Fuzzy Sets SMALL**

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

X = { xi } = {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"}]

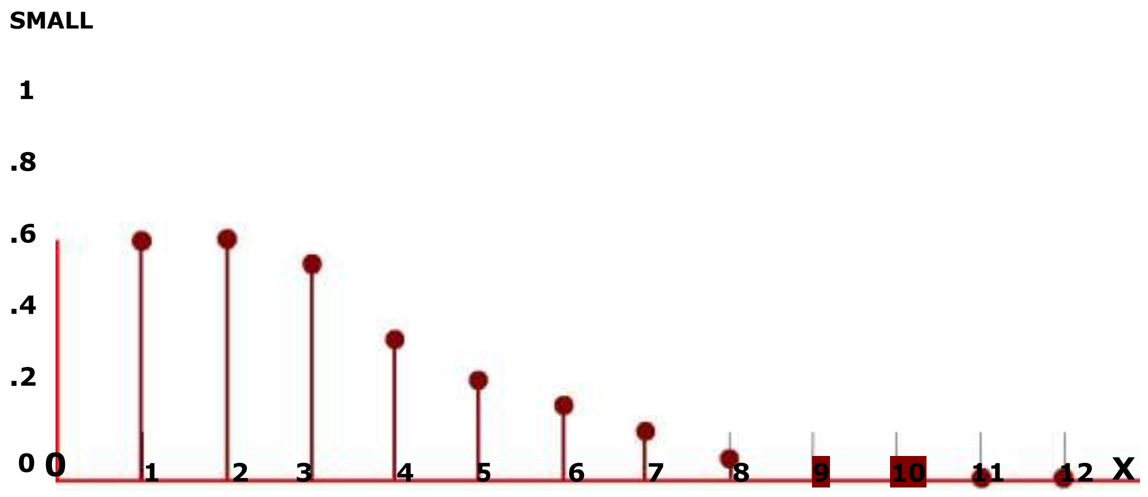


Fig Graphic Interpretation of Fuzzy Sets SMALL

14

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

The fuzzy set PRIME numbers, defined in the universal space

X = { xi } = {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"}]

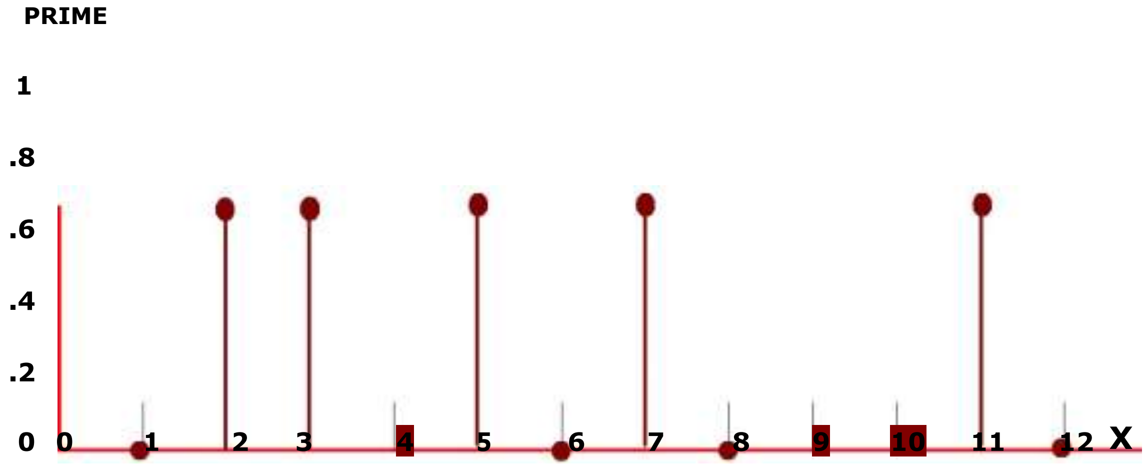


Fig Graphic Interpretation of Fuzzy Sets PRIME

15

- **Graphic Interpretation of Fuzzy Sets UNIVERSALSPACE**

In any application of sets or fuzzy sets theory, all sets are subsets of

i 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 = { xi } = {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 :

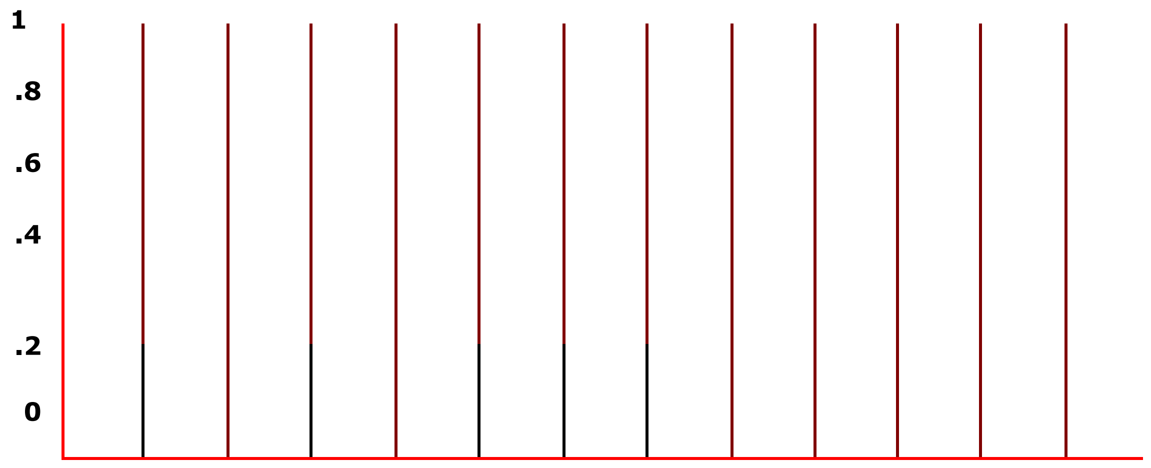
UNIVERSALSPACE = FuzzySet {{1, 1}, {2, 1}, {3, 1}, {4, 1}, {5, 1}, {6, 1}, {7, 1}, {8, 1}, {9, 1}, {10, 1}, {11, 1}, {12, 1}}

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 "}]

UNIVERSAL SPACE



0 1 2 3 4 5 6 7 8 9 10 11 12 X

Fig Graphic Interpretation of Fuzzy Set UNIVERSALSPACE

16

- **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 "}]

EMPTY

1

.8

.6

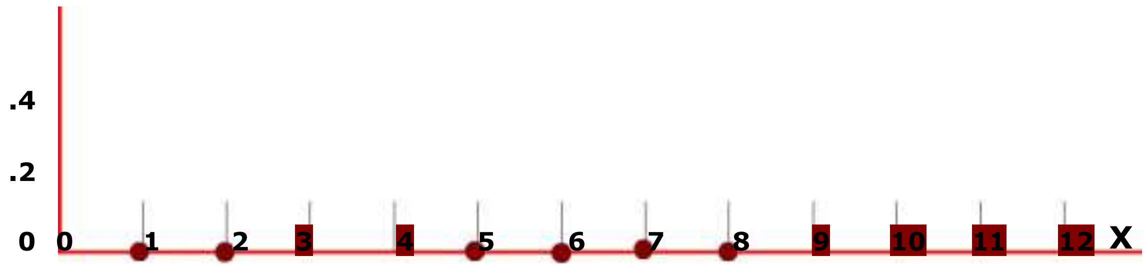


Fig Graphic Interpretation of Fuzzy Set EMPTY

18

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 $\cup$ MEDIUM]**
Intersection :	**FUZZYINTERSECTON = [SMALL $\cap$ 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]

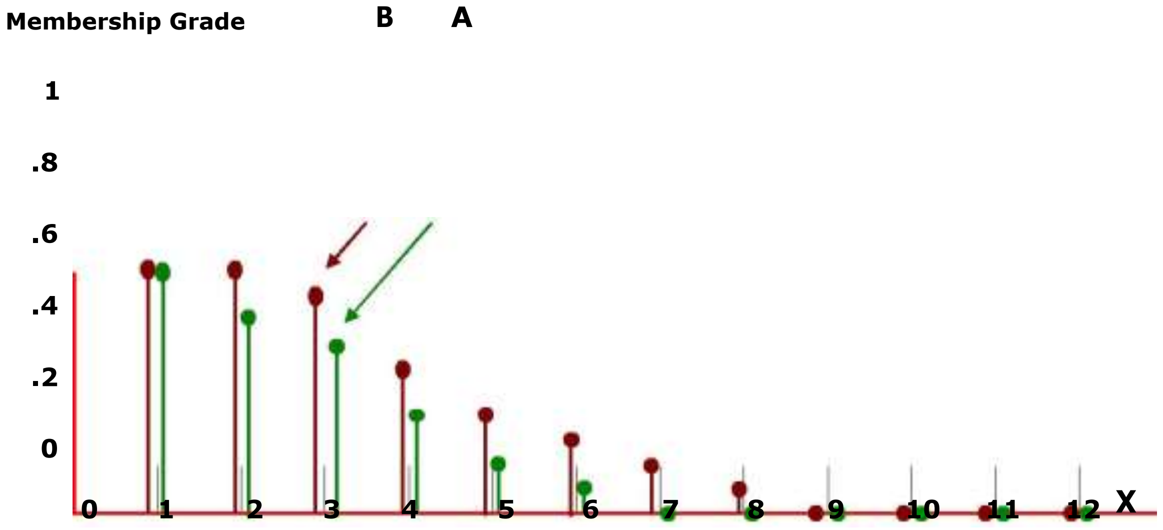


Fig Graphic Interpretation of Fuzzy Inclusion

FuzzyPlot [SMALL, VERYSMALL]

20

- **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)** $\subset$ **B(x)** $\subset$ **C(x), then accordingly A** $\subset$ **C.**

21

- **Equality**

Let **A** and **B** be fuzzy sets defined in the same space **X**.
Then **A** and **B** if and only if are equal, which is denoted **X = Y**

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]

Membership Grade **B A**

1

.8

.6

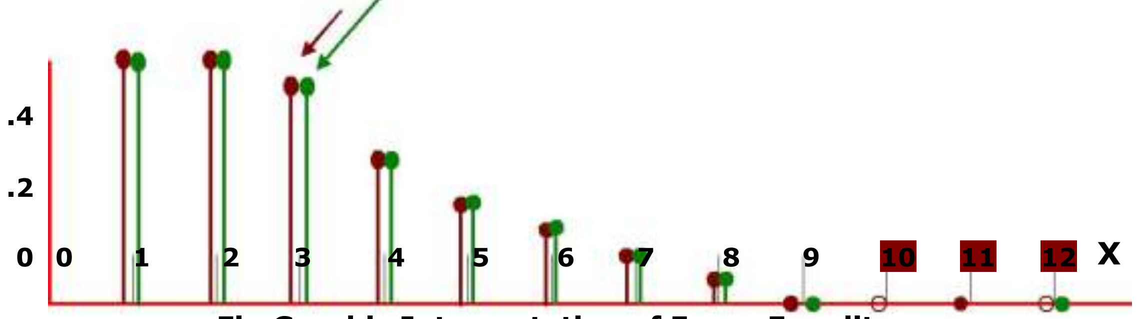


Fig Graphic Interpretation of Fuzzy Equality

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.**

22

- **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 $\overline{A}$

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]

Membership Grade **A** **Ac**

1

.8

.6

.4

.2

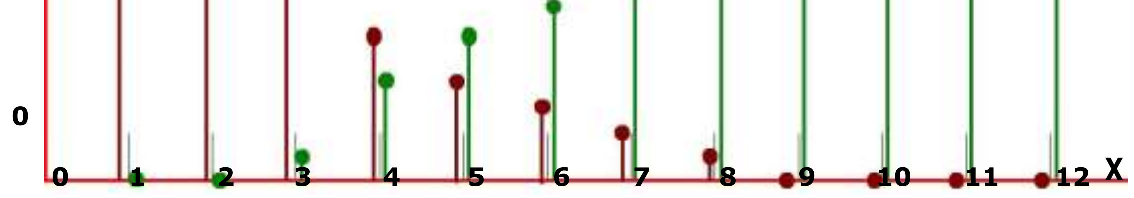


Fig Graphic Interpretation of Fuzzy Compliment

FuzzyPlot [SMALL, NOTSMALL]

23

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]

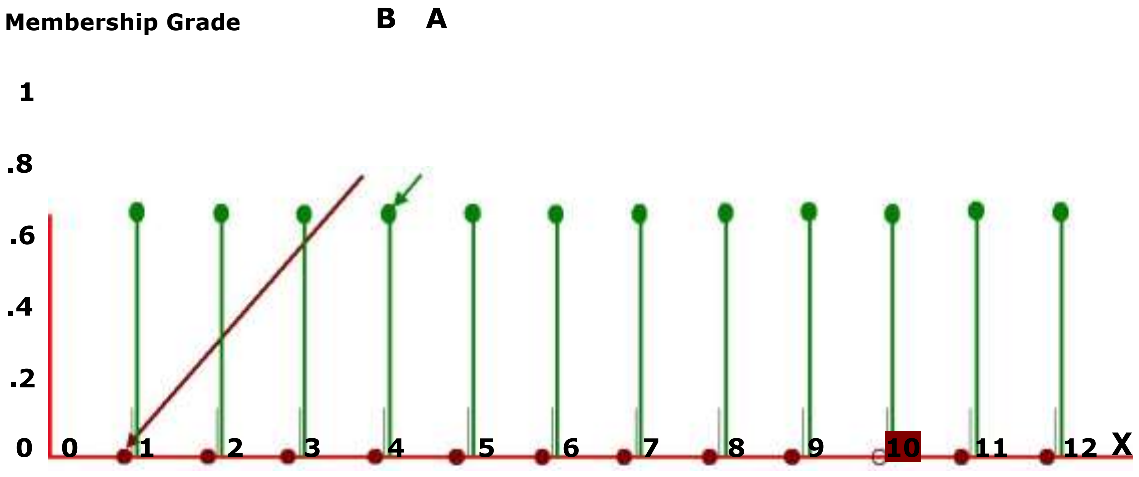


Fig Graphic Interpretation of Fuzzy Compliment

FuzzyPlot [EMPTY, UNIVERSALSPACE]

24

- **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}]

Membership Grade **FUZZYUNION = [SMALL ∪ MEDIUM]**

1

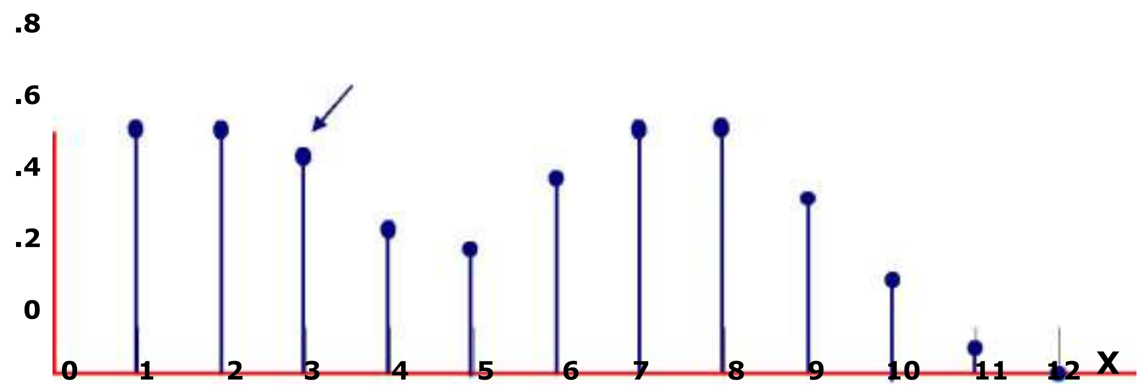


Fig Graphic Interpretation of Fuzzy Union

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**.

25

- **Intersection**

Let **A** and **B** be fuzzy sets defined in the space **X**. Intersection is defined as the greatest fuzzy set that include both **A** and **B**. Intersection of **A** and **B** is denoted by **A ∩ B.** The following relation must be satisfied for the

intersection 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 $x \in X$

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}]

Membership Grade **FUZZYINTERSECTON = [SMALL ∩ MEDIUM]**

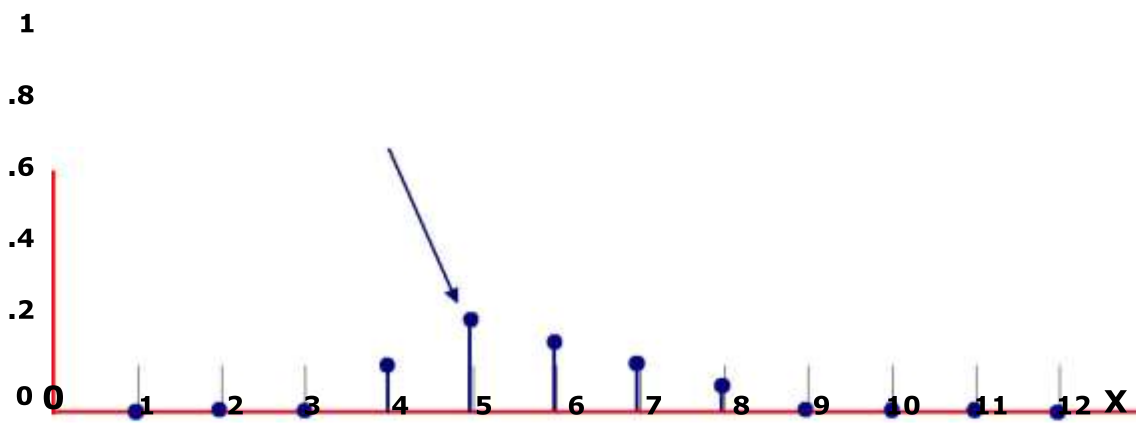


Fig Graphic Interpretation of Fuzzy Union

FuzzyPlot [INTERSECTION]

26

- **Difference**

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

The difference of **A** and **B** is denoted by **A** ∩ **B'.**

Fuzzy Difference : **(A - B)(x) = min [A(x), 1- B(x)]** **for all** $x \in X$

Example : Difference of **MEDIUM and SMALL**

The fuzzy set A 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 set B SMALL

MEDIUM = 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}}

Fuzzy Complement : Bc(x) = 1 – B(x)

The fuzzy set Bc 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 : Difference by the definition of **Difference**

FUZZYDIFFERENCE = [MEDIUM ∩ SMALL']

SetMediumDIFFERECESmall = **FuzzySet [{{1,0},{2,0}, {3,0}, {4,0.2}, {5,0.5}, {6,0.7}, {7,0.8}, {8, 0.9}, {9, 0.7}, {10, 0.4}, {11, 0.1}, {12, 0}}** , **UniversalSpace** → **{1, 12, 1}]**

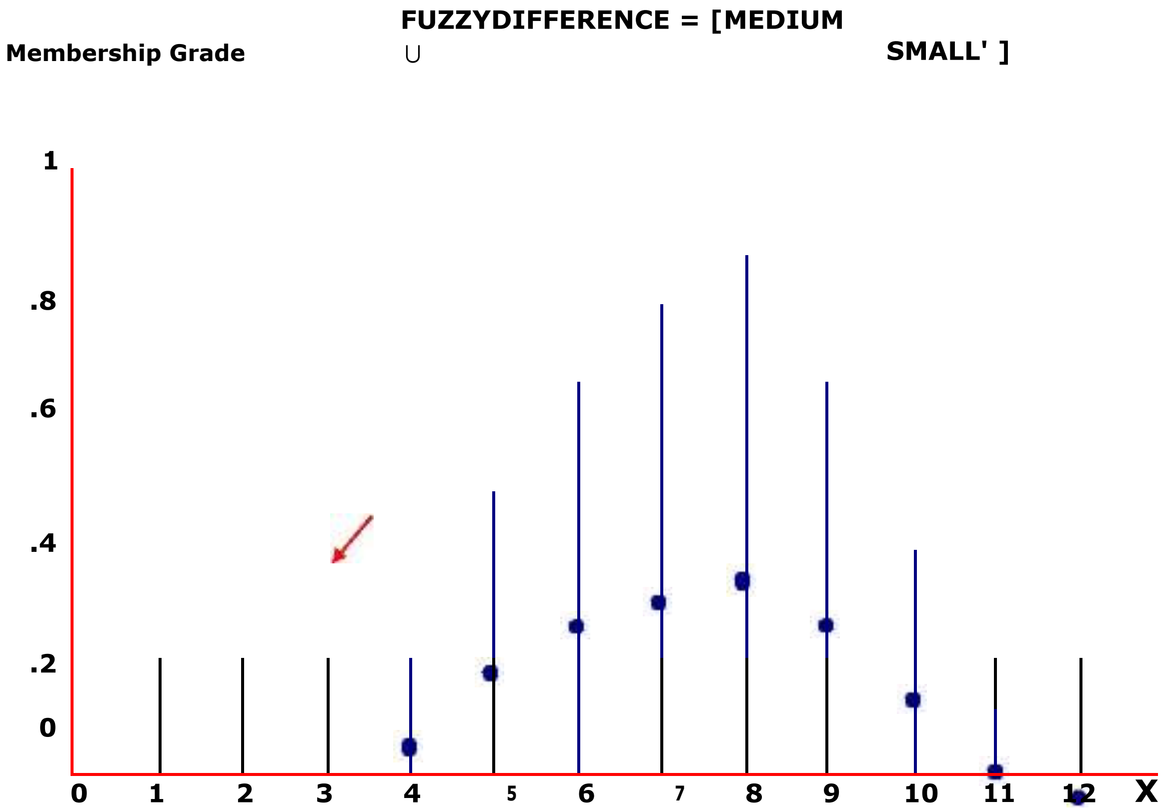


Fig Graphic Interpretation of Fuzzy Union

FuzzyPlot [UNION]

27

2.3 Fuzzy Properties

Properties related to Union, Intersection, Differences are illustrated below.

- **Properties Related to Union**

The properties related to union are :

Identity, Idempotence, Commutativity and Associativity.

- **Identity:**

$A \cup \Phi = A$

input = Equality [SMALL ∪ EMPTY , SMALL]
output = True

$A \cup X = X$

input = Equality [SMALL ∪ UnivrsalSpace , UnivrsalSpace]

output = True

- **Idempotence :**

$A \cup A = A$

input = Equality [SMALL ∪ SMALL , SMALL]

output = True

- **Commutativity :**

  A ∪ B = B ∪ A

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

  output = True

28

[Continued from previous slide]

- **Associativity:**

A ∪ (B∪ C) = (A∪ B) ∪ C

input = Equality [Small ∪ (Medium ∪ Big) , (Small ∪ Medium) ∪ Big]

output = True

Fuzzy Set Small , Medium , Big

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}}]

Calculate Fuzzy relations :

- **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}]**

- **Small** ∪ **(Medium** ∪ **Big)** = **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, 0.8}, {11, 1}, {12, 1}]**

- **(Small** ∪ **Medium)** ∪ **Big] = 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, 0.8}, {11, 1},{12, 1}]**

Fuzzy set (3) and (4) proves Associativity relation

29

- **Properties Related to Intersection**

  Absorption, Identity, Idempotence, Commutativity, Associativity.

  - **Absorption by Empty Set :**

    A ∩ Φ = Φ

    input = Equality [Small ∩ Empty , Empty]

    output = True

  - **Identity :**

    A ∩ X = A

    input = Equality [Small ∩ UnivrsalSpace , Small]
    output = True

  - **Idempotence :**

    A ∩ A = A

    input = Equality [Small ∩ Small , Small]
    output = True

  - **Commutativity :**

    A ∩ B = B ∩ A

    input = Equality [Small ∩ Big , Big ∩ Small]

    output = True

- **Associativity :**

A ∩ **(B** ∩ **C) = (A** ∩ **B)** ∩ **C**

input = Equality [Small ∩ (Medium ∩ Big), (Small ∩ Medium) ∩ Big]

output = True

30

- **Additional Properties**

Related to Intersection and Union

- **Distributivity:**

  A ∩ (B ∪ C) = (A ∩ B) ∪ (A ∩ C)

  input = Equality [Small ∩ (Medium ∪ Big) ,

  (Small ∩ Medium) ∪ (Small ∩ Big)]

  output = True

- **Distributivity:**

  A ∪ (B ∩ C) = (A ∪ B) ∩ (A ∪ C)

  input = Equality [Small ∪ (Medium ∩ Big) ,

  (Small ∪ Medium) ∩ (Small ∪ Big)]

  output = True

- **Law of excluded middle :**

  A ∪ A' = X

  input = Equality [Small ∪ NotSmall , UnivrsalSpace]
  output = True

= **Law of contradiction**

$\mathbf{A \cap A' = \Phi}$

input = Equality [Small ∩ NotSmall , EmptySpace]
output = True

31

aa Cartesian Product Of Two Fuzzy Sets

Cartesian Product of two Crisp Sets

Let **A** and **B** be two crisp sets in the universe of discourse **X** and **Y**.. The Cartesian product of **A** and **B** is denoted by **A x B**

Defined as $\mathbf{A \times B = \{ (a , b) \mid a \in A , b \in B \}}$

Note : Generally $\mathbf{A \times B \neq B \times A}$

Example :

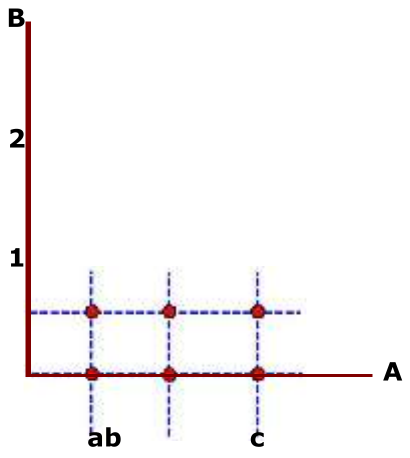
Let **A = {a, b, c}** and **B = {1, 2}**

then **A x B = { (a , 1) , (a , 2) ,**

(b , 1) , (b , 2) ,

(c , 1) , (c , 2) }

Graphic representation of A x B



- ### Cartesian product of two Fuzzy Sets

Let **A** and **B** be two fuzzy sets in the universe of discourse **X** and **Y**.

The Cartesian product of **A** and **B** is denoted by **A x B**

Defined by their membership function $\propto_A (x)$ and $\propto_B (y)$ as

$$\propto_{A \times B} (x , y) = \min [\propto_A (x) , \propto_B (y)] = \propto_A (x) \wedge \propto_B (y)$$

or $\propto_{A \times B} (x , y) = \propto_A (x) \propto_B (y)$

for all $x \in X$ and $y \in Y$

Thus the Cartesian product **A x B** is a fuzzy set of ordered pair

(x , y) for all **x** $\in$ **X** and **y** $\in$ **Y**, with grade membership of **(x , y)** in

- **x Y** given by the above equations .

In a sense Cartesian product of two Fuzzy sets is a Fuzzy Relation.

32

- Fuzzy Relations

Fuzzy Relations describe the degree of association of the elements;
Example : **"x is approximately equal to y".**

– Fuzzy relations offer the capability to capture the uncertainty and vagueness in relations between sets and elements of a set.

– Fuzzy Relations make the description of a concept possible.

Fuzzy Relations were introduced to supersede classical crisp relations; It describes the total presence or absence of association of elements.

In this section, first the fuzzy relation is defined and then expressing fuzzy relations in terms of matrices and graphical visualizations. Later the properties of fuzzy relations and operations that can be performed with fuzzy relations are illustrated.

33

3.1 Definition of Fuzzy Relation

Fuzzy relation is a generalization of the definition of fuzzy set from 2-D space to 3-D space.

- **Fuzzy relation definition**

Consider a Cartesian product

$$\mathbf{A \times B} = \{ (x, y) \mid x \in A, y \in B \}$$

where **A** and **B** are subsets of universal sets $\mathbf{U_1}$ and $\mathbf{U_2}$.

Fuzzy relation on **A x B** is denoted by **R** or **R(x , y)** is defined as the set

$$\mathbf{R} = \{ ((x, y), \propto_R (x, y)) \mid (x, y) \in A \times B, \propto_R (x, y) \in [0,1] \}$$

where $\propto_R$ **(x , y)** is a function in two variables called membership function.

- It gives the degree of membership of the ordered pair **(x , y)** in **R** associating with each pair **(x , y)** in **A x B** a real number in the interval **[0 , 1]**.
- The degree of membership indicates the degree to which **x** is in relation to **y**.

Note :

- Definition of fuzzy relation is a generalization of the definition of fuzzy set from the 2-D space **$(x , , \propto_R (x))$** to 3-D space **$((x , y) , \propto_R (x , y))$**.

- Cartesian product **A x B** is a relation by itself between **x** and **y** .

- A fuzzy relation **R** is a sub set of **R^3** namely

$\{ ((x , y) , \propto_R (x , y)) \mid \in \quad A \times B \times [0,1] \in \quad U_1 \times U_2 \times [0,1] \}$

34

- **Example of Fuzzy Relation**

- $= \{ ((x_1, y_1), 0)), ((x_1, y_2), 0.1)), ((x_1, y_3), 0.2)),$
$((x_2, y_1), 0.7)), ((x_2, y_2), 0.2)), ((x_2, y_3), 0.3)),$

$((x_3, y_1), 1)), ((x_3, y_2), 0.6)), ((x_3, y_3), 0.2)),$

The relation can be written in matrix form as

$$R \triangleq \begin{array}{c|ccc} y & y_1 & Y_2 & Y_3 \\ x & & & \\ \hline x_1 & 0 & 0.1 & 0.2 \\ X_2 & 0.7 & 0.2 & 0.3 \\ X_3 & 1 & 0.6 & 0.2 \end{array}$$

where symbol $\triangleq$ means ' is defined as' and

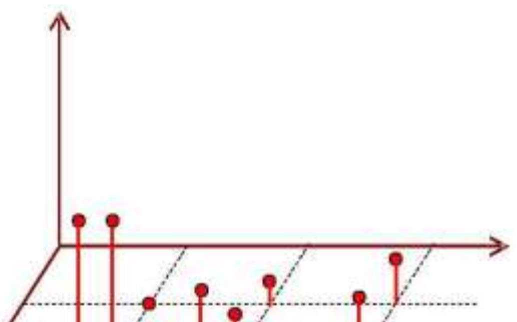
the values in the matrix are the values of membership function:

$\propto_R (x_1, y_1) = 0$ $\quad \propto_R (x_1, y_2) = 0.1$ $\quad \propto_R (x_1, y_3) = 0.2$

$\propto_R (x_2, y_1) = 0.7$ $\quad \propto_R (x_2, y_2) = 0.2$ $\quad \propto_R (x_2, y_3) = 0.3$

$\propto_R (x_3, y_1) = 1$ $\quad \propto_R (x_3, y_2) = 0.6$ $\quad \propto_R (x_3, y_3) = 0.2$

Assuming $\mathbf{x_1 = 1}$, $\mathbf{x_2 = 2}$, $\mathbf{x_3 = 3}$
the relation can be graphically **(X, Y, $\propto$)** as :



and $\mathbf{y_1 = 1}$, $\mathbf{y_2 = 2}$, $\mathbf{y_3 = 3}$, represented
by points in 3-D space

Note : Since the values of the membership function **0.7, 1, 0.6** are in the direction of **x** below the major diagonal **(0, 0.2, 0.2)** in the matrix are grater than those **0.1, 0.2, 0.3** in the direction of **y**, we therefore say that the relation **R** describes **x** is grater than **y**.

35

3.2 Forming Fuzzy Relations

Assume that **V** and **W** are two collections of objects.

A fuzzy relation is characterized in the same way as it is in a fuzzy set.

– The first item is a list containing element and membership grade pairs,

$$\{\{v_1, w_1\}, R_{11}\}, \{\{ v_1, w_2\}, R_{12}\}, \dots , \{\{ v_n, w_m\}, R_{nm}\}\}.$$

where $\{ v_1, w_1\}, \{ v_1, w_2\}, \dots , \{ v_n, w_m\}$ are the elements of the relation are defined as ordered pairs, and $\{ R_{11} , R_{12} , \dots , R_{nm}\}$ are the membership grades of the elements of the relation that range from 0 to 1, inclusive.

– The second item is the universal space; for relations, the universal space consists of a pair of ordered pairs,

$$\{\{ V_{min}, V_{max}, C_1\}, \{ W_{min}, W_{max}, C_2\}\}.$$

where the first pair defines the universal space for the first set and the second pair defines the universal space for the second set.

Example showing how fuzzy relations are represented

Let V = {1, 2, 3} and W = {1, 2, 3, 4}.

A fuzzy relation **R** is, a function defined in the space **V x W**, which takes values from the interval [0, 1] , expressed as **R : V x W → [0, 1]**

■ **= FuzzyRelation [{{{1, 1}, 1}, {{1, 2}, 0.2}, {{1, 3}, 0.7}, {{1, 4}, 0}, {{2, 1}, 0.7}, {{2, 2}, 1}, {{2, 3}, 0.4}, {{2, 4}, 0.8}, {{3, 1}, 0}, {{3, 2}, 0.6}, {{3, 3}, 0.3}, {{3, 4}, 0.5}, UniversalSpace → {{1, 3, 1}, {1, 4, 1}}]**

This relation can be represented in the following two forms shown below

Elements of fuzzy relation are ordered pairs **{v_i , w_j}**, where **v_i** is first and **w_j** is second element. The membership grades of the elements are represented by the heights of the vertical lines.

36

3.3 Projections of Fuzzy Relations

Definition : A fuzzy relation on **A x B** is denoted by **R** or **R(x , y)** is defined as the set

$$\mathbf{R = \{ ((x , y) , \propto_R (x , y)) \mid (x , y) \in A \times B , \propto_R (x , y) \in [0,1] \}}$$

where $\propto_R (x , y)$ is a function in two variables called membership function. The first, the second and the total projections of fuzzy relations are stated below.

■ **First Projection** $\underset{X}{\max}$ **of R :** defined as

$$\mathbf{R^{(1)} = \{(x) , \propto_{R^{(1)}} (x , y))\}}$$

$$\mathbf{= \{(x) , \underset{Y}{\max} \propto_R (x , y)) \mid (x , y) \in A \times B \}}$$

$\underset{Y}{\max}$

$\underset{X}{\max}$ ■ **Second Projection of R :** defined as

$$\mathbf{R^{(2)} = \{(y) , \propto_{R^{(2)}} (x , y))\}}$$

$$\mathbf{= \{(y) , \propto_R (x , y)) \mid (x , y) \in A \times B \}}$$

• **Total Projection of R :** defined as

$\mathbf{R_{(T)}} =$

$$\max_{X} \max_{Y} \{\propto_R (x , y) \mid (x , y) \in A \times B \}$$

Note : In all these three expression

means **max** with respect to **y** while **x** is considered fixed

means **max** with respect to **x** while **y** is considered fixed

The Total Projection is also known as Global projection

37

■ Example : Fuzzy Projections

The Fuzzy Relation **R** together with First, Second and Total Projection of **R** are shown below.

R =

y / x	y_1	y_2	y_3	y_4	Y_5	$R_{(1)}$		
x_1	**0.1**	**0.3**	**1**	**0.5**	**0.3**	**1**		
x_2	**0.2**	**0.5**	**0.7**	**0.9**	**0.6**	**0.9**		
x_3	**0.3**	**0.6**	**1**	**0.8**	**0.2**	**1**		
$R_{(2)}$	**0.3**	**0.6**	**1**	**0.9**	**0.6**	**1**	=	$R_{(T)}$

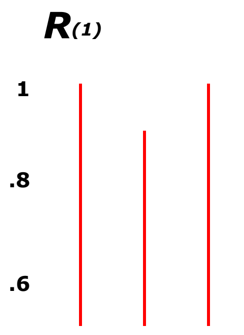
Note :

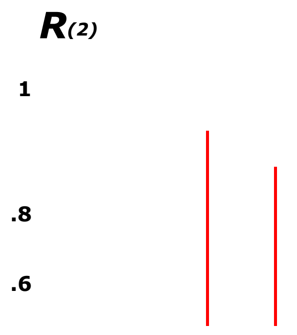
For $R^{(1)}$ select $\max_{Y}$ means **max** with respect to **y** while **x** is considered fixed

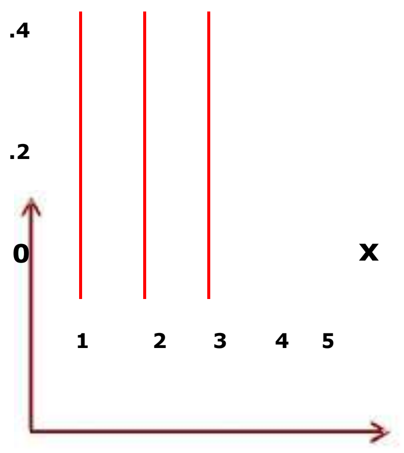
For $R^{(2)}$ select $\max_{x}$ means **max** with respect to **x** while **y** is considered fixed

For $R_{(T)}$ select **max** with respect to $R_{(1)}$ and $R_{(2)}$

The Fuzzy plot of these projections are shown below.







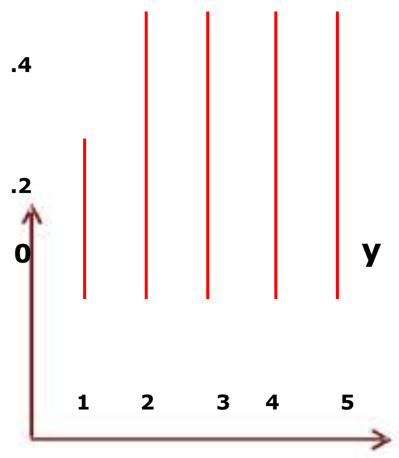


Fig Fuzzy plot of 1st projection $R^{(1)}$ **Fig Fuzzy plot of 2nd projection $R^{(2)}$**

38

3.4 Max-Min and Min-Max Composition

The operation composition combines the fuzzy relations in different

variables, say **(x , y)** and **(y , z)** ; **x** ∈ **A , y** ∈ **B ,** **z** ∈ **C** .

Consider the relations :

$R_1(x , y) = \{ ((x , y) , \propto_{R1} (x , y)) \mid (x , y) \in A \times B \}$

$R_2(y , z) = \{ ((y , y) , \propto_{R1} (y , z)) \mid (y , z) \in B \times C \}$

The domain of **R_1** is **A x B** and the domain of **R_2** is **B x C**

◇ Max-Min Composition

Definition : The Max-Min composition denoted by **$R_1 \circ R_2$** with membership function $\propto_{R1 \circ R2}$ defined as

$$R_1 \circ R_2 = \{ ((x , z) , \max_{Y}(\min (\propto_{R1} (x , y) , \propto_{R2} (y , z))))\} ,$$

$$(x , z) \in A \times C , y \in B$$

Thus $R_1 \circ R_2$ is relation in the domain **A x C**

An example of the composition is shown in the next slide.

◊ **Example : Max-Min Composition**

Consider the relations **$R_1(x , y)$** and **$R_2(y , z)$** as given below.

$$R_1 = \begin{array}{c|ccc} y & y_1 & y_2 & y_3 \\ x & & & \\ \hline x_1 & 0.1 & 0.3 & 0 \\ x_2 & 0.8 & 1 & 0.3 \end{array} \qquad R_2 = \begin{array}{c|ccc} z & z_1 & z_2 & z_3 \\ y & & & \\ \hline y_1 & 0.8 & 0.2 & 0 \\ y_2 & 0.2 & 1 & 0.6 \\ y_3 & 0.5 & 0 & 0.4 \end{array}$$

Note : Number of columns in the first table and second table are equal.

Compute max-min composition denoted by **$R_1 \circ R_2$** :

Step -1 Compute **min** operation (definition in previous slide).

Consider row **x_1** and column **z_1** , means the pair **(x_1 , z_1)** for all **y_j** , **$j = 1, 2, 3$**, and perform **min** operation

$\min (\propto_{R1} (x_1 , y_1) , \propto_{R2} (y_1 , z_1)) = \min (0.1, 0.8) = 0.1,$

$\min (\propto_{R1} (x_1 , y_2) , \propto_{R2} (y_2 , z_1)) = \min (0.3, 0.2) = 0.2,$

$\min (\propto_{R1} (x_1 , y_3) , \propto_{R2} (y_3 , z_1)) = \min (0, 0.5) = 0,$

Step -2 Compute **max** operation (definition in previous slide).

For **$x = x_1$** , **$z = z_1$** , **$y = y_j$, $j = 1, 2, 3$**,

Calculate the grade membership of the pair **(x_1 , z_1)** as

$\{ (x_1 , z_1) , \max ((\min (0.1, 0.8), \min (0.3, 0.2), \min (0, 0.5))$

i.e. **$\{ (x_1 , z_1) , \max(0.1, 0.2, 0) \}$**

i.e. **$\{ (x_1 , z_1) , 0.2 \}$**

Hence the grade membership of the pair **(x_1 , z_1)** is **0.2** .

Similarly, find all the grade membership of the pairs

$(x_1 , z_2) , (x_1 , z_3) , (x_2 , z_1) , (x_2 , z_2) , (x_2 , z_3)$

The final result is

$$R_1 \circ R_2 = \begin{array}{c|ccc} z & z_1 & z_2 & z_3 \\ x & & & \\ \hline x_1 & 0.1 & 0.3 & 0 \\ x_2 & 0.8 & 1 & 0.3 \end{array}$$

Note : If tables **R_1** and **R_2** are considered as matrices, the operation composition resembles the operation multiplication in matrix calculus linking row by columns. After each cell is occupied max-min value (the product is replaced by min, the sum is replaced by max).

40

■ **Example : Min-Max Composition**

The min-max composition is similar to max-min composition with the difference that the roll of max and min are interchanged.

Definition : The max-min composition denoted by $\mathbf{R_1}\ \mathbf{R_2}$ with membership function $\propto_{\mathbf{R1}\ \mathbf{R2}}$ is defined by

$$\mathbf{R_1}\quad \mathbf{R_2} = \{ ((x , z) , \overset{min}{}{}_{y} (\max (\propto_{R1} (x , y) , \propto_{R2} (y , z))))\} ,$$

$$(x , z) \in A \times C , y \in B$$

Thus $\mathbf{R_1}\quad \mathbf{R_2}$ is relation in the domain $\mathbf{A \times C}$

Consider the relations $\mathbf{R_1(x , y)}$ and $\mathbf{R_2(y , z)}$ relation of previous example of max-min composition, as given by the same that is

$$\mathbf{R_1} \triangleq \begin{array}{c|ccc} {}_{x}\ {}^{y} & y_1 & y_2 & y_3 \\ \hline x_1 & 0.1 & 0.3 & 0 \\ x_2 & 0.8 & 1 & 0.3 \end{array} \qquad \mathbf{R_2} \triangleq \begin{array}{c|ccc} {}_{y}\ {}^{z} & z_1 & z_2 & z_3 \\ \hline y_1 & 0.8 & 0.2 & 0 \\ y_2 & 0.2 & 1 & 0.6 \\ y_3 & 0.5 & 0 & 0.4 \end{array}$$

After computation in similar way as done in the case of max-min composition,   the final result is

$$\boldsymbol{R_1 \quad R_2} = \begin{array}{c|ccc} z & & & \\ x & z_1 & z_2 & z_3 \\ \hline x_1 & 0.3 & 0 & 0.1 \\ x_2 & 0.5 & 0.4 & 0.4 \end{array}$$

◇ **Relation between Max-Min and Min-Max Compositions**

The Max-Min and Min-Max Compositions are related by the formula

$$\overline{R_1} \circ \overline{R_2} = \overline{R_1 \quad R_2}$$

41

Fuzzy Systems

What are Fuzzy Systems ?

- Fuzzy Systems include Fuzzy Logic and Fuzzy Set Theory.
- Knowledge exists in two distinct forms :
  - the Objective knowledge that exists in mathematical form is used in engineering problems; and
  - the Subjective knowledge that exists in linguistic form, usually impossible to quantify.

  Fuzzy Logic can coordinate these two forms of knowledge in a logical way.
- Fuzzy Systems can handle simultaneously the numerical data and linguistic knowledge.
- Fuzzy Systems provide opportunities for modeling of conditions which are inherently imprecisely defined.
- Many real world problems have been modeled, simulated, and replicated with the help of fuzzy systems.
- The applications of Fuzzy Systems are many like : Information retrieval systems, Navigation system, and Robot vision.
- Expert Systems design have become easy because their domains are inherently fuzzy and can now be handled better;

examples : Decision-support systems, Financial planners, Diagnostic system, and Meteorological system.

03

- **Introduction**

Any system that uses Fuzzy mathematics may be viewed as Fuzzy system.

The Fuzzy Set Theory - membership function, operations, properties and the relations have been described in previous lectures. These are the prerequisites for understanding Fuzzy Systems. The applications of Fuzzy set theory is Fuzzy logic which is covered in this section.

Here the emphasis is on the design of fuzzy system and fuzzy controller in a

closed–loop. The specific topics of interest are :

- Fuzzification of input information,
- Fuzzy Inferencing using Fuzzy sets ,
- De-Fuzzification of results from the Reasoning process, and
- Fuzzy controller in a closed–loop.

Fuzzy Inferencing, is the core constituent of a fuzzy system. A block schematic of Fuzzy System is shown in the next slide. Fuzzy Inferencing combines the facts obtained from the Fuzzification with the fuzzy rule base and conducts the Fuzzy Reasoning Process.

04

- **Fuzzy System**

A block schematic of Fuzzy System is shown below.

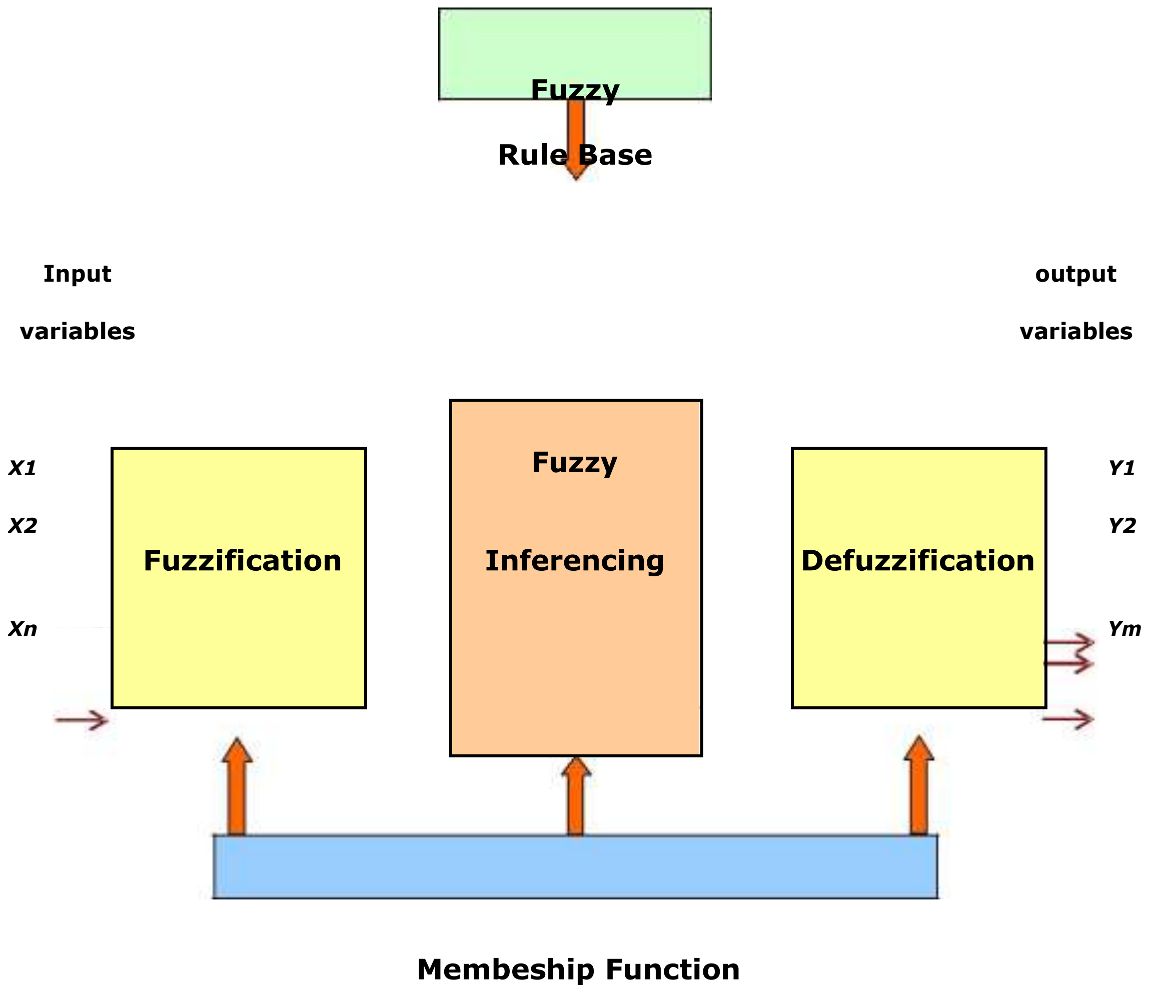


Fig. Elements of Fuzzy System

Fuzzy System elements

– **Input Vector** : $\mathbf{X = [x_1 , x_2, . . . x_n]^T}$ are crisp values**,** which are transformed into fuzzy sets in the fuzzification block.

– **Output Vector** : $\mathbf{Y = [y_1 , y_2, . . . y_m]^T}$ comes out from the defuzzification block, which transforms an output fuzzy set back to a crisp value**.**

- **Fuzzification** : a process of transforming crisp values into grades of membership for linguistic terms, "far", "near", "small" of fuzzy sets.
- **Fuzzy Rule base** : a collection of propositions containing linguistic variables; the rules are expressed in the form:

  If (x is A) AND (y is B) . . . . . . THEN (z is C)

  where **x, y** and **z** represent variables (e.g. distance, size) and **A, B** and **Z** are linguistic variables (e.g. `far', `near', `small').
- **Membership function** : provides a measure of the degree of similarity of elements in the universe of discourse **U** to fuzzy set.
- **Fuzzy Inferencing** : combines the facts obtained from the Fuzzification with the rule base and conducts the Fuzzy reasoning process.
- **Defuzzyfication:** Translate results back to the real world values.

05

1. Fuzzy Logic

A simple form of logic, called a two-valued logic is the study of "truth tables" and logic circuits. Here the possible values are true as **1**, and false as **0**.

This simple two-valued logic is generalized and called **fuzzy logic** which treats "truth" as a continuous quantity ranging from **0** to **1**.

Definition : Fuzzy logic (FL) is derived from fuzzy set theory dealing with reasoning that is approximate rather than precisely deduced from classical two-valued logic.

- FL is the application of Fuzzy set theory.
- FL allows set membership values to range (inclusively) between **0** and **1**.
- FL is capable of handling inherently imprecise concepts.
- FL allows in linguistic form, the set membership values to imprecise concepts like "**slightly**", "**quite**" and "**very**".

06

2.1 Classical Logic

Logic is used to represent simple facts. Logic defines the ways of putting symbols together to form **sentences** that represent facts. Sentences are either true or false but not both are called **propositions**.

Examples :

Sentence	Truth value	Is it a Proposition ?
"Grass is green"	"true"	Yes
"2 + 5 = 5"	"false"	Yes
"Close the door"	-	No
"Is it hot out side ?"	-	No
"x > 2"	-	No (since x is not defined)
"x = x"	-	No

(don't know what is "x" and "=" mean; "3 = 3" or say "air is equal to air" or "Water is equal to water" has no meaning)

- **Propositional Logic (PL)**

A proposition is a statement - which in English is a declarative sentence and Logic defines the ways of putting symbols together to form sentences that represent facts. Every proposition is either true or false. Propositional logic is also called boolean algebra.

Examples: (a) The sky is blue., (b) Snow is cold. , (c) 12 * 12=144

Propositional logic : It is fundamental to all logic.

- Propositions are “Sentences”; either true or false but not both.
- A sentence is smallest unit in propositional logic
- If proposition is true, then truth value is "true"; else “false”

‡ **Example ;**

Sentence	**"Grass is green";**
Truth value	**“ true”;**
Proposition	**“yes”**

07

ƒ Statement, Variables and Symbols

Statement : A simple statement is one that does not contain any other statement as a part. A compound statement is one that has two or more simple statements as parts called components.

Operator or connective : Joins simple statements into compounds, and joins compounds into larger compounds.

Symbols for connectives

assertion	**P**					**"p is true"**
nagation	**¬p**	**~**	**!**		**NOT**	**"p is false"**
conjunction	**p ∧ q**	**·**	**&&**	**&**	**AND**	**"both p and q are true"**
disjunction	**P v q**	**||**	**|**		**OR**	**"either p is true, or q is true, or both "**
implication	**p → q**	⊃	⇒		**if . . then**	**"if p is true, then q is true" " p implies q "**
equivalence	↔	≡	⇔		**if and only if**	**"p and q are either both true or both false"**

- **Truth Value**

The truth value of a statement is its truth or falsity ,

- is either true or false,

~pis either true or false,

p v q is either true or false, and so on.

"**T**" or "**1**" means "**true**". and

"**F**" or "**0**" means "**false**"

Truth table is a convenient way of showing relationship between several propositions. The truth table for negation, conjunction, disjunction, implication and equivalence are shown below.

p	q	¬p	¬q	p ∧ q	p v q	p→ q	p ↔ q	q→ p
T	T	F	F	T	T	T	T	T
T	F	F	T	F	T	F	F	T
F	T	T	F	F	T	T	F	F
F	F	T	T	F	F	T	T	T

- **Tautology**

A Tautology is proposition formed by combining other propositions **(p, q, r, . . .)** which is true regardless of truth or falsehood of **p, q, r, ...**

The important tautologies are :

$(p \to q) \Leftrightarrow \neg [p \wedge (\neg q)]$ and $(p \to q) \Leftrightarrow (\neg p) \vee q$

A proof of these tautologies, using the truth tables are given below.

Tautologies $(p \to q) \Leftrightarrow \neg [p \wedge (\neg q)]$ and $(p \to q) \Leftrightarrow (\neg p) \vee q$

Table 1: Proof of Tautologies

p	q	p→ q	¬q	p ∧ (¬q)	¬ [p ∧ (¬q)]	¬p	(¬p) ∨ q
T	T	T	F	F	T	F	T
T	F	F	T	T	F	F	F
F	T	T	F	F	T	T	T
F	F	T	T	F	T	T	T

Note :

- The entries of two columns **p→ q** and ← **[p ∧ (←q)]** are identical, proves the tautology. Similarly, the entries of two columns **p→ q** and **(←p)** ∨ **q** are identical, proves the other tautology.

- The importance of these tautologies is that they express the membership function for **p**→ **q** in terms of membership functions of either propositions **p** and ←**q** or ←**p** and **q.**

10

- **Equivalences**

Between Logic , Set theory and Boolean algebra.

Some mathematical equivalence between Logic and Set theory and the correspondence between Logic and Boolean algebra **(0, 1)** are given below.

Logic	Boolean Algebra (0, 1)	Set theory
T	1	
F	0	
∧	x	∩ , ∩
∨	+	∪ , U
¬	’ ie complement	(‾)
↔	=	
p, q, r	a, b, c	

11

- **Membership Functions obtain from facts**

Consider the facts (the two tautologies)

$(\mathbf{p} \rightarrow \mathbf{q}) \leftrightarrow \; \leftarrow [\mathbf{p} \wedge (\leftarrow \mathbf{q})]$ and $(\mathbf{p} \rightarrow \mathbf{q}) \leftrightarrow \; (\leftarrow \mathbf{p}) \vee \mathbf{q}$

Using these facts and the equivalence between logic and set theory, we can obtain membership functions for $\propto_{p \rightarrow q} (x , y)$.

From 1st fact : $\propto_{p \rightarrow q} (x , y) = 1 - \propto_{p \cap \bar{q}} (x , y)$

$= 1 - \min [\propto_p(x) , 1 - \propto_q (y)]$ **Eq (1)**

From 2nd fact : $\propto_{p \rightarrow q} (x , y) = 1 - \propto_{\bar{p} \cup q} (x , y)$

$= \max [1 - \propto_p (x) , \propto_q (y)]$ **Eq (2)**

Boolean truth table below shows the validation membership functions

Table-2 : Validation of Eq (1) and Eq (2)

$\propto_p(x)$	$\propto_q(y)$	$1 - \propto_p (x)$	$1 - \propto_q (y)$	$\max [1 - \propto_p (x) , \propto_q (y)]$	$1 - \min [\propto_p(x) , 1 - \propto_q (y)]$
1	1	0	0	1	1
1	0	0	1	0	0
0	1	1	0	1	1
0	0	1	1	1	1

Note :

■ Entries in last two columns of this table-2 agrees with the entries in table-1 for **p→ q** , the proof of tautologies, read **T** as **1** and **F** as **0**.

■ The implication membership functions of Eq.1 and Eq.2 are not the only ones that give agreement with **p→ q**. The others are :

$$\propto_{p \to q}(x, y) = 1 - \propto_p(x)\,(1 - \propto_q(y)) \qquad \textbf{Eq (3)}$$

$$\propto_{p \to q}(x, y) = \min [\, 1, 1 - \propto_p(x) + \propto_q(y)] \qquad \textbf{Eq (4)}$$

12

- **Modus Ponens and Modus Tollens**

In traditional propositional logic there are two important inference rules, Modus Ponens and Modus Tollens**.**

Modus Ponens

Premise 1 : **" x is A "**

Premise 2 : **" if x is A then y is B "** ; Consequence : **" y is B "**

Modus Ponens is associated with the implication **" A implies B " [A→ B]** In terms of propositions **p** and **q**, the Modus Ponens is expressed as

$$(p \land (p \rightarrow q)) \rightarrow q$$

Modus Tollens

Premise 1 : **" y is not B "**

Premise 2 : **" if x is A then y is B "** ; Consequence : **" x is not A "** In terms of propositions **p** and **q**, the Modus Tollens is expressed as

$$(\neg q \land (p \rightarrow q)) \rightarrow \neg p$$

13

2.2 Fuzzy Logic

Like the extension of crisp set theory to fuzzy set theory, the extension of crisp logic is made by replacing the bivalent membership functions of the crisp logic with the fuzzy membership functions.

In crisp logic, the truth value acquired by the proposition are 2-valued, namely true as **1** and false as **0**.

In fuzzy logic, the truth values are multi-valued, as absolute true, partially true, absolute false etc represented numerically as real value between

- to **1**.

Note : The fuzzy variables in fuzzy sets, fuzzy propositions, fuzzy relations etc are represented usually using symbol **~** as $\tilde{\mathbf{P}}$ but for the purpose of easy to write it is always represented as **P** .

- Recaps

01 Membership function $\propto_{\mathbf{A}}$ **(x)** describes the membership of the elements **x** of the base set **X** in the fuzzy set **A** .

- **Fuzzy Intersection operator** ∩ (AND connective) applied to two fuzzy sets **A** and **B** with the membership functions $\propto_{\mathbf{A}}$ **(x)** and $\propto_{\mathbf{B}}$ **(x)** based on min/max operations is $\propto_{\mathbf{A \cap B}}$ **= min [** $\propto_{\mathbf{A}}$ **(x) ,** $\propto_{\mathbf{B}}$ **(x)] ,** **x** ∈ **X (Eq. 01)**

- **Fuzzy Intersection operator** ∩ (AND connective) applied to two fuzzy sets **A** and **B** with the membership functions $\propto_{\mathbf{A}}$ **(x)** and $\propto_{\mathbf{B}}$ **(x)** based on algebraic product is $\propto_{\mathbf{A \cap B}}$ **=** $\propto_{\mathbf{A}}$ **(x)** $\propto_{\mathbf{B}}$ **(x) , x** ∈ **X (Eq. 02)**

4. Fuzzy Union operator U (OR connective) applied to two fuzzy sets **A** and **B** with the membership functions $\propto_{\mathbf{A}}$ **(x)** and $\propto_{\mathbf{B}}$ **(x)** based on min/max operations is $\propto_{\mathbf{A \cup B}}$ **= max [** $\propto_{\mathbf{A}}$ **(x) ,** $\propto_{\mathbf{B}}$ **(x)] ,** **x** ∈ **X (Eq. 03)**

- **Fuzzy Union operator U** (OR connective) applied to two fuzzy sets **A** and **B** with the membership functions $\propto_{\mathbf{A}}$ **(x)** and $\propto_{\mathbf{B}}$ **(x)** based on algebraic sum is

$\propto_{\mathbf{A \cup B}}$ **=** $\propto_{\mathbf{A}}$ **(x) +** $\propto_{\mathbf{B}}$ **(x) -** $\propto_{\mathbf{A}}$ **(x)** $\propto_{\mathbf{B}}$ **(x) , x** ∈ **X (Eq. 04)**

06 Fuzzy Compliment operator (¯) (NOT operation) applied to fuzzy set **A** with the membership function $\propto_{\mathbf{A}}$ **(x)** is $\propto_{\bar{A}}$ **= 1 -** $\propto_{\mathbf{A}}$ **(x) , x** ∈ **X (Eq. 05)**

4. Fuzzy relations combining two fuzzy sets by connective "min operation" is an operation by cartesian product **R : X x Y→ [0 , 1].**

$\propto_{\mathbf{R(x,y)}}$ **= min[**$\propto_{\mathbf{A}}$ **(x),** $\propto_{\mathbf{B}}$ **(y)] (Eq. 06)** or

$\propto_{\mathbf{R(x,y)}}$ **=** $\propto_{\mathbf{A}}$ **(x)** $\propto_{\mathbf{B}}$ **(y) (Eq. 07)**

Y / x	V	h-m	m
G	1	0.5	0.0

Example : Relation **R** between fruit colour **x**

	R		
Y	0.3	1	0.4
R	0	0.2	1

≜

and maturity grade **y** characterized by base set

linguistic colorset **X = {green, yellow, red}**

maturity grade as **Y = {verdant, half-mature, mature}**

- **Max-Min Composition -** combines the fuzzy relations variables, say **(x , y)** and **(y , z)** ; **x** ∈ **A , y** ∈ **B , z** ∈ **C** . consider the relations :

  $$R_1(x , y) = \{ ((x , y) , \propto_{R1} (x , y)) \mid (x , y) \in A \times B \}$$

  $$R_2(y , z) = \{ ((y , y) , \propto_{R1} (y , z)) \mid (y , z) \in B \times C \}$$

  The domain of R_1 is **A x B** and the domain of R_2 is **B x C**

  max-min composition denoted by $R_1 \circ R_2$ with membership function $\propto_{R1 \circ R2}$

  $$R_1 \circ R_2 = \{ ((x , z) , \max_y (\min (\propto_{R1} (x , y) , \propto_{R2} (y , z))))\} ,$$
  $$(x , z) \in A \times C , y \in B \qquad \textbf{(Eq. 08)}$$

  Thus $R_1 \circ R_2$ is relation in the domain **A x C**

15

- **Fuzzy Propositional**

A fuzzy proposition is a statement **P** which acquires a fuzzy truth value **T(P)** .

Example :

P: Ram is honest

T(P) = 0.8 , means **P** is partially true. **T(P)** = 1 , means **P** is absolutely true.

■ Fuzzy Connectives

The fuzzy logic is similar to crisp logic supported by connectives.

Table below illustrates the definitions of fuzzy connectives.

Table : Fuzzy Connectves

Connective	Symbols	Usage	Definition
Nagation	¬	**¬ P**	**1 – T(P)**
Disjuction	∨	**P ∨ Q**	**Max[T(P) , T(Q)]**
Conjuction	∧	**P ∧ Q**	**min[T(P) , T(Q)]**
Implication	⇒	**P ⇒ Q**	**¬P ∨ Q = max (1-T(P), T(Q)]**

Here **P , Q** are fuzzy proposition and **T(P) , T(Q)** are their truth values.

- the **P** and **Q** are related by the ⇒ operator are known as antecedents and consequent respectively.

- as crisp logic, here in fuzzy logic also the operator ⇒ represents

  IF-THEN statement like,

  IF x is A THEN y is B, is equivalent to

  R = (A x B) U (¬ A x Y)

  the membership function of **R** is given by

  $$\propto_R (x , y) = \max [\min (\propto_A (x) , \propto_B (y)) , 1 - \propto_A (x)]$$

– For the compound implication statement like

IF x is A THEN y is B, ELSE y is C is equivalent to

R = (A x B) ∪ (¬ A x C)

the membership function of **R** is given by

$$\propto_R (x , y) = \max [\min (\propto_A (x) , \propto_B (y)) , \min (1 - \propto_A (x), \propto_C (y))]$$

17

Example 1 : (Ref : Previous slide)

P : Mary is efficient , $T(P) = 0.8$,

Q : Ram is efficient , $T(Q) = 0.65$,

¬ P : Mary is efficient , $T(\neg P) = 1 - T(P) = 1 - 0.8 = 0.2$

$P \wedge Q$: Mary is efficient and so is Ram, i.e.

$$T(P \wedge Q) = \min (T(P), T(Q)) = \min (0.8, 0.65)) = 0.65$$

$P \vee Q$: Either Mary or Ram is efficient i.e.

$$T(P \vee Q) = \max (T(P), T(Q)) = \max (0.8, 0.65)) = 0.8$$

$P \Rightarrow Q$: If Mary is efficient then so is Ram, i.e.

$$T(P \Rightarrow Q) = \max (1 - T(P), T(Q)) = \max (0.2, 0.65)) = 0.65$$

18

Example 2 : (Ref : Previous slide on fuzzy connective)

Let **X = {a, b, c, d}** ,

A = {(a, 0) (b, 0.8) (c, 0.6) (d, 1)}

B = {(1, 0.2) (2, 1) (3, 0.8) (4, 0)}

C = {(1, 0) (2, 0.4) (3, 1) (4, 0.8)}

= { 1, 2, 3, 4} the universe of discourse could be viewed as

{ (1, 1) (2, 1) (3, 1) (4, 1) }

i.e., a fuzzy set all of whose elements ***x*** have ***∝(x) = 1***

Determine the implication relations

- **If x is A** **for all x in the set X** **THEN y is B**
- **If x is A THEN y is B Else y is C**

Solution

To determine implication relations (i) compute :

The operator ⇒ represents **IF-THEN** statement like,

IF x is A THEN y is B, is equivalent to **R = (A x B) U (¬ A x Y)** and

the membership function **R** is given by

$$\propto_R (x , y) = \max [\min (\propto_A (x) , \propto_B (y)) , 1 - \propto_A (x)]$$

Fuzzy Intersection A x B is defined as : **Fuzzy Intersection ¬*A* x Y is defined as :**

for all x in the set X,

(A ∩ B)(x) = min [A(x), B(x)],

(¬A ∩ Y)(x) = min [A(x), Y(x)],

A x B =

B	1	2	3	4
A				
a	0	0	0	0
b	0.2	0.8	0.8	0
c	0.2	0.6	0.6	0
d	0.2	1	0.8	0

¬A x Y =

y	1	2	3	4
A				
a	1	1	1	1
b	0.2	0.2	0.2	0.2
c	0.4	0.4	0.4	0.4
d	0	0	0	0

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

Therefore **R = (A x B) U (¬ A x Y)** gives

R =

y	1	2	3	4
x				
a	1	1	1	1
b	0.2	0.8	0.8	0
c	0.4	0.6	0.6	0.4
d	0.2	1	0.8	0

This represents **If x is A THEN y is B** ie **T(A ⇒ B) = max (1- T(A), T(B))**

19

To determine implication relations (ii) compute : (Ref : Previous slide)

Given **X = {a, b, c, d}** ,

A = {(a, 0) (b, 0.8) (c, 0.6) (d, 1)}

B = {(1, 0.2) (2, 1) (3, 0.8) (4, 0)}

C = {(1, 0) (2, 0.4) (3, 1) (4, 0.8)}

Here, the operator ⇒ represents **IF-THEN-ELSE** statement like,
IF x is A THEN y is B Else y is C, is equivalent to

R = (A x B) U (¬ A x C) and

the membership function of **R** is given by

$$\propto_R (x , y) = \max [\min (\propto_A (x) , \propto_B (y)) , \min(1 - \propto_A (x), \propto_C (y)]$$

Fuzzy Intersection A x B is defined as :

for all x in the set X,

(A ∩ B)(x) = min [A(x), B(x)],

Fuzzy Intersection ¬A x Y is defined as :

for all x in the set X

(¬A ∩ C)(x) = min [A(x), C(x)],

A x B =

A \ B	1	2	3	4
a	0	0	0	0
b	0.2	0.8	0.8	0
c	0.2	0.6	0.6	0
d	0.2	1	0.8	0

¬A x C =

A \ y	1	2	3	4
a	0	0.4	1	0.8
b	0.2	0.2	0.2	0.2
c	0.4	0.4	0.4	0.4
d	0	0	0	0

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

Therefore **R = (A x B) U (¬ A x C)** **gives**

$$R = \begin{array}{c|cccc} x \backslash y & 1 & 2 & 3 & 4 \\ \hline a & 1 & 1 & 1 & 1 \\ b & 0.2 & 0.8 & 0.8 & 0 \\ c & 0.4 & 0.6 & 0.6 & 0.4 \\ d & 0.2 & 1 & 0.8 & 0 \end{array}$$

This represents **If x is A THEN y is B Else y is C**

20

3 Fuzzy Quantifiers

In crisp logic, the predicates are quantified by quantifiers. Similarly, in fuzzy logic the propositions are quantified by quantifiers. There are two classes of fuzzy quantifiers :

- Absolute quantifiers

and - Relative quantifiers

Examples :

Absolute quantifiers	**Relative quantifiers**
round about 250	almost
much greater than 6	about
some where around 20	most

21

☐ Fuzzification

The fuzzification is a process of transforming crisp values into grades of membership for linguistic terms of fuzzy sets.

The purpose is to allow a fuzzy condition in a rule to be interpreted.

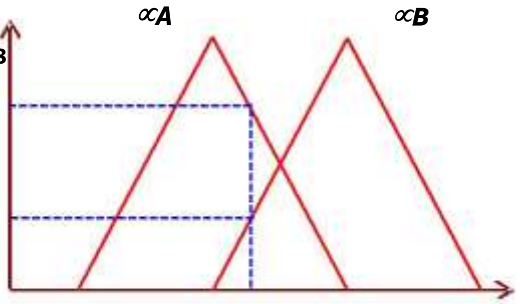
Fuzzification of the car speed

Example 1 : Speed X_0 = 70km/h

Fig below shows the fuzzification of low and a medium speed fuzzy set.

Low Medium

1 $\propto$



.8

.6

.4

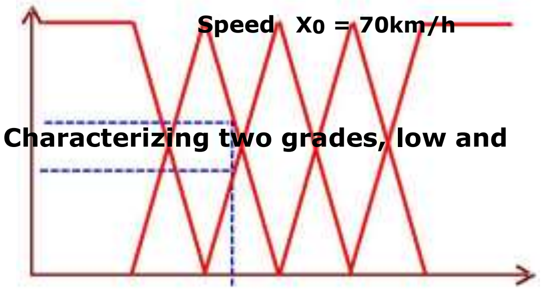
.2

0

20 40 60 80 100 120 140

Speed X_0 = 70km/h

Characterizing two grades, low and medium speed fuzzy set



Example 2 : Speed X_0 = 40km/h

Medium

$\propto$ **V Low Low High V High**

1

.8

.6

.4

.2

0

10 20 30 40 50 60 70 80 90 00

Speed X_0 = 40km/h

the car speed to characterize a

Given car speed value **X_0=70km/h** : grade **$\propto_A(x_0)$ = 0.75** belongs to fuzzy low, and grade **$\propto_B(x_0)$ = 0.25** belongs to fuzzy medium

Given car speed value **X_0=40km/h** : grade **$\propto_A(x_0)$ = 0.6** belongs to fuzzy low, and grade **$\propto_B(x_0)$ = 0.4** belongs to fuzzy medium.

Characterizing five grades, Very low, low, medium, high and very high speed fuzzy set

22

Fuzzy Inference

Fuzzy Inferencing is the core element of a fuzzy system.

Fuzzy Inferencing combines - the facts obtained from the fuzzification with the rule base, and then conducts the fuzzy reasoning process.

Fuzzy Inference is also known as **approximate reasoning.**

Fuzzy Inference is computational procedures used for evaluating linguistic descriptions. Two important inferring procedures are

– Generalized Modus Ponens (GMP)
– Generalized Modus Tollens (GMT)

23

- **Generalized Modus Ponens (GMP)**

This is formally stated as

If x is A THEN y is B

- **is ¬A**

- **is ¬B**

where **A , B , ¬A , ¬B** are fuzzy terms.

Note : Every fuzzy linguistic statements above the line is analytically known and what is below the line is analytically unknown.

To compute the membership function **¬B** , the max-min composition of fuzzy set **¬A** with **R(x , y)** which is the known implication relation (**IF-THEN**) is used. i.e. **¬B = ¬A o R(x, y)** In terms of membership function

$\propto_{\neg B}(y) = \max(\min(\propto_{\neg A}(x), \propto_R(x, y)))$ where

$\propto_{\neg A}(x)$ is the membership function of **¬A** ,

$\propto_R(x, y)$ is the membership function of the implication relation and $\propto_{\neg B}(y)$ is the membership function of **¬B**

24

- **Generalized Modus Tollens (GMT)**

This is formally stated as

If x is A THEN y is B

• is ¬B

x is ¬A

where **A , B , ¬A** , **¬B** are fuzzy terms.

Note : Every fuzzy linguistic statements above the line is analytically known and what is below the line is analytically unknown.

To compute the membership function **¬A** , the max-min composition of fuzzy set **¬B** with **R(x , y)** which is the known implication relation (**IF-THEN**) is used. i.e. $\neg A = \neg B \circ R(x, y)$

In terms of membership function

$$\propto_{\neg A}(y) = \max(\min(\propto_{\neg B}(x), \propto_R(x, y)))$$ where

$\square_{\neg B}(x)$ is the membership function of **¬B** ,

$\propto_R(x, y)$ is the membership function of the implication relation and $\propto_{\neg A}(y)$ is the membership function of **¬A**

Example :

Apply the fuzzy Modus Ponens rules to deduce Rotation is quite slow?

Given **:**

- If the temperature is high then then the rotation is slow.
- The temperature is very high.

Let **H (High) , VH (Very High) , S (Slow) and QS (Quite Slow)** indicate the associated fuzzy sets.

Let the set for temperatures be **X = {30, 40, 50, 60, 70, 80, 90, 100}** , and Let the set of rotations per minute be **Y = {10, 20, 30, 40, 50, 60}** and

H = {(70, 1) (80, 1) (90, 0.3)}

VH = {(90, 0.9) (100, 1)}

QS = {10, 1) (20, 08) }

S = {(30, 0.8) (40, 1) (50, 0.6)

To derive **R(x, y)** representing the implication relation (i) above, compute

R (x, y) = max (H x S , ¬ H x Y)

	10	20	30	40	50	60
30	0	0	0	0	0	0
40	0	0	0	0	0	0
50	0	0	0	0	0	0

	10	20	30	40	50	60
30	1	1	1	1	1	1
40	1	1	1	1	1	1
50	1	1	1	1	1	1

$$H \times S = \begin{array}{c|cccccc} 60 & 0 & 0 & 0 & 0 & 0 & 0 \\ 70 & 0 & 0 & 0.8 & 1 & 0.6 & 0 \\ 80 & 0 & 0 & 0.8 & 1 & 0.6 & 0 \\ 90 & 0 & 0 & 0.3 & 0.3 & 0.3 & 0 \\ 100 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \qquad H \times Y = \begin{array}{c|cccccc} \mathit{60} & 1 & 1 & 1 & 1 & 1 & 1 \\ \mathit{70} & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathit{80} & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathit{90} & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 \\ \mathit{100} & 1 & 1 & 1 & 1 & 1 & 1 \end{array}$$

$$R(x,Y) = \begin{array}{c|cccccc} & 10 & 20 & 30 & 40 & 50 & 60 \\ \hline 30 & 1 & 1 & 1 & 1 & 1 & 1 \\ 40 & 1 & 1 & 1 & 1 & 1 & 1 \\ 50 & 1 & 1 & 1 & 1 & 1 & 1 \\ 60 & 1 & 1 & 1 & 1 & 1 & 1 \\ 70 & 0 & 0 & 0.8 & 1 & 0.6 & 0 \\ 80 & 0 & 0 & 0.8 & 1 & 0.6 & 0 \\ 90 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 \\ 100 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}$$

To deduce Rotation is quite slow, we make use of the composition rule

$$QS = VH \circ R(x, y)$$

$$\begin{array}{c|cccccc} & 10 & 20 & 30 & 40 & 50 & 60 \\ \hline 30 & 1 & 1 & 1 & 1 & 1 & 1 \\ 40 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}$$

$$= [0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0.9 \;\; 1] \;\; \mathbf{x} \;\; \begin{array}{c|cccccc} 50 & 1 & 1 & 1 & 1 & 1 & 1 \\ 60 & 1 & 1 & 1 & 1 & 1 & 1 \\ 70 & 0 & 0 & 0 & 0 & 0 & 0 \\ 80 & 0 & 0 & 0 & 0 & 0 & 0 \\ 90 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 \\ 100 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}$$

$$= [1 \; 1 \; 1 \; 1 \; 1 \; 1 \;]$$

- **Fuzzy Rule Based System**

The fuzzy linguistic descriptions are formal representation of systems made through fuzzy IF-THEN rule. They encode knowledge about a system in statements of the form :

IF (a set of conditions) are satisfied THEN (a set of consequents) can be inferred.

IF (x_1 is A_1, x_2 is A_2, xn is An) THEN (y_1 is B_1, y_2 is B_2, yn is Bn)

where linguistic variables $\mathbf{x_i}$, $\mathbf{y_j}$ take the values of fuzzy sets $\mathbf{A_i}$ and $\mathbf{B_j}$ respectively.

Example :

IF **there is "*heavy*" rain and "*strong*" winds**

THEN **there must "*severe*" flood warnings.**

Here, **heavy** , **strong** , and **severe** are fuzzy sets qualifying the variables *rain, wind,* and *flood* warnings respectively.

A collection of rules referring to a particular system is known as a fuzzy

rule base. If the conclusion **C** to be drawn from a rule base **R** is the conjunction of all the individual consequents **C** $_i$ of each rule , then

$\mathbf{C = C_1 \cap C_2 \cap . . . \cap C_n}$ where

$\propto_c (y) = \min (\propto_{c1}(y), \propto_{c2}(y), \propto_{cn}(y)), \forall y \in Y$

where **Y** is universe of discourse.

On the other hand, if the conclusion **C** to be drawn from a rule base **R** is the disjunction of the individual consequents of each rule, then

$\mathbf{C = C_1 \cup C_2 \cup . . . \cup C_n}$ where

$\propto_c (y) = \max (\propto_{c1}(y), \propto_{c2}(y), \propto_{cn}(y)), \forall y \in Y$ where

- is universe of discourse.

28

- **Defuzzification**

In many situations, for a system whose output is fuzzy, it is easier to take a crisp decision if the output is represented as a single quantity. This conversion of a single crisp value is called Defuzzification.

Defuzzification is the reverse process of fuzzification.

The typical Defuzzification methods
are – Centroid method,

– Center of sums, –
Mean of maxima.

Centroid method

It is also known as the "center of gravity" of area method. It obtains the centre of area **(x*)** occupied by the fuzzy set .
For discrete membership function, it is given by

$$\mathbf{x^*} = \frac{\sum_{i=1}^{n} \mathbf{xi} \propto \mathbf{(xi)}}{\sum^{n} \propto \mathbf{(xi)}} \quad \text{where}$$

- represents the number elements in the sample,

and x_i are the elements, and

$\propto (x_i)$ is the membership function.

Genetic Algorithms & Modeling

What are GAs ?

- 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.

- **Introduction**

Solving problems mean looking for solutions, which is best among others. Finding the solution to a problem is often thought :

– In computer science and AI, as a process of **search** through the space of possible solutions. The set of possible solutions defines the search space (also called state space) for a given problem. Solutions or partial solutions are viewed as points in the search space.

– In engineering and mathematics, as a process of **optimization**. The problems are first formulated as mathematical models expressed in terms of functions and then to find a solution, discover the parameters that optimize the model or the function components that provide optimal system performance.

- **Why Genetic Algorithms ?**

It is better than conventional AI ; It is more robust.

j unlike older AI systems, the GA's do not break easily even if the inputs changed slightly, or in the presence of reasonable noise.

- while performing search in large state-space, multi-modal state-space, or n-dimensional surface, a genetic algorithms offer significant benefits over many other typical search optimization techniques like - linear programming, heuristic, depth-first, breath-first.

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

1.1 Optimization

Optimization is a process that finds a best, or optimal, solution for a problem. The Optimization problems are centered around three factors :

- **An objective function :** which is to be minimized or maximized; Examples:

  In manufacturing, we want to maximize the profit or minimize the cost .

  In designing an automobile panel, we want to maximize the strength.

- **A set of unknowns or variables :** that affect the objective function, Examples:

  In manufacturing, the variables are amount of resources used or the time spent.

  In panel design problem, the variables are shape and dimensions of the panel.

- **A set of constraints :** that allow the unknowns to take on certain values but exclude others;
  Examples:

  1. In manufacturing, one constrain is, that all "time" variables to be non-negative.

  - In the panel design, we want to limit the weight and put constrain on its shape.

An optimization problem is defined as : Finding values of the variables that minimize or maximize the objective function while satisfying the constraints.

- **Optimization Methods**

Many optimization methods exist and categorized as shown below. The suitability of a method depends on one or more problem characteristics to be optimized to meet one or more objectives like :

- low cost,
- high performance,
- low loss

These characteristics are not necessarily obtainable, and requires knowledge about the problem.

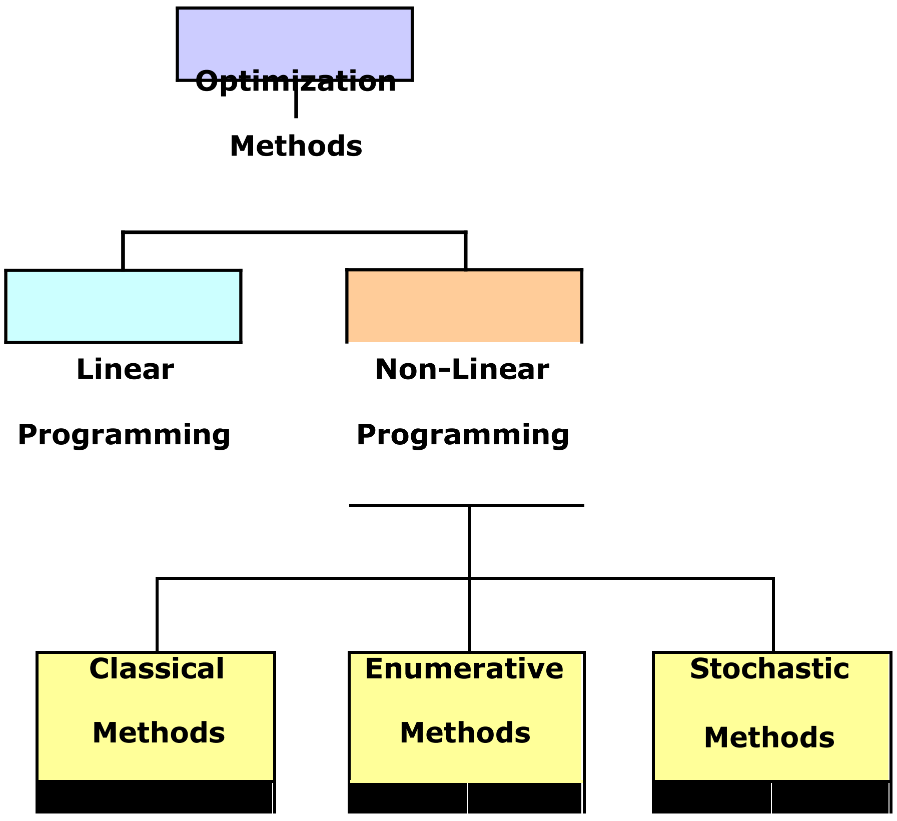


Fig. Optimization Methods

Each of these methods are briefly discussed indicating the nature of the problem they are more applicable.

- **Linear Programming**

Intends to obtain the optimal solution to problems that are perfectly represented by a set of linear equations; thus require a priori knowledge of the problem. Here the

- the functions to be minimized or maximized, is called *objective functions*,
- the set of linear equations are called *restrictions*.
- the *optimal solution*, is the one that minimizes (or maximizes) the objective function.

Example : "Traveling salesman", seeking a minimal traveling distance.

- **Non- Linear Programming**

Intended for problems described by non-linear equations.
The methods are divided in three large groups:

Classical, Enumerative and Stochastic.

Classical search uses deterministic approach to find best solution. These methods requires knowledge of gradients or higher order derivatives. In many practical problems, some desired information

are not available, means deterministic algorithms are inappropriate.

The techniques are subdivide into:

- Direct methods, e.g. Newton or Fibonacci – Indirect methods.

Enumerative search goes through every point (one point at a time) related to the function's domain space. At each point, all possible solutions are generated and tested to find optimum solution. It is easy to implement but usually require significant computation. In the field of artificial intelligence, the enumerative methods are subdivided into two categories:

- Uninformed methods, e.g. Mini-Max algorithm
- Informed methods, e.g. Alpha-Beta and A* ,

Stochastic search deliberately introduces randomness into the search process. The injected randomness may provide the necessary impetus to move away from a local solution when searching for a global optimum. e.g., a gradient vector criterion for “smoothing” problems. Stochastic methods offer robustness quality to optimization process. Among the stochastic techniques, the most widely used are :

- Evolutionary Strategies (ES),
- Genetic Algorithms (GA), and
- Simulated Annealing (SA).

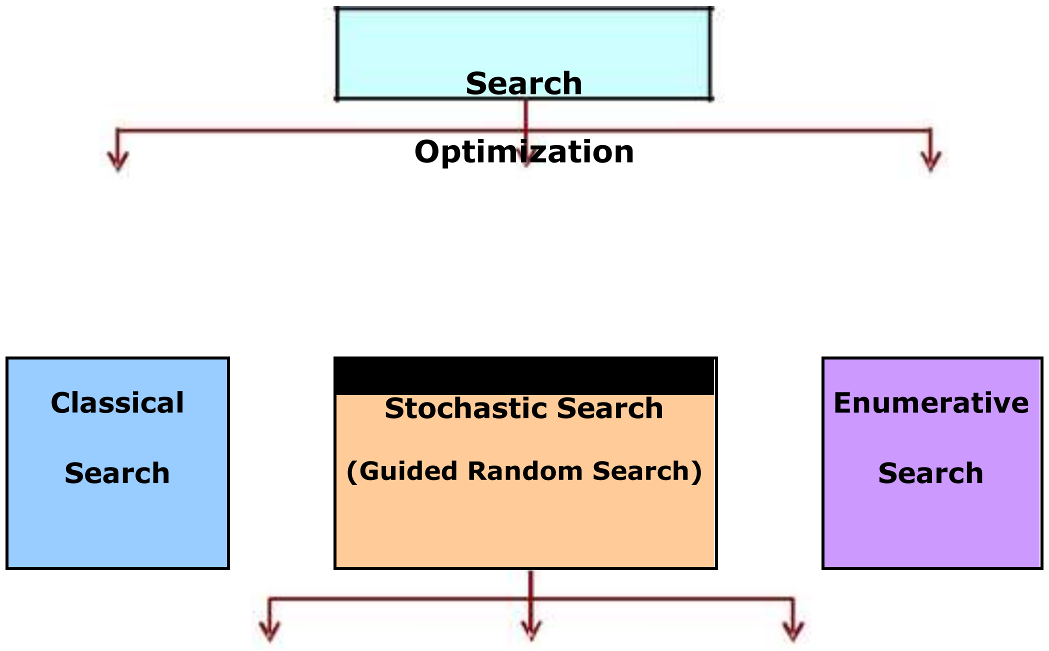
The ES and GA emulate nature’s evolutionary behavior, while SA is based on the physical process of annealing a material.

1.2 Search Optimization

Among the three Non-Linear search methodologies, just mentioned in the previous slide, our immediate concern is **Stochastic search** which means

– Evolutionary Strategies (ES), – Genetic Algorithms (GA), and – Simulated Annealing (SA).

The two other search methodologies, shown below, the *Classical* and the *Enumerative* methods, are first briefly explained. Later the *Stochastic methods* are discussed in detail. All these methods belong to Non-Linear search.



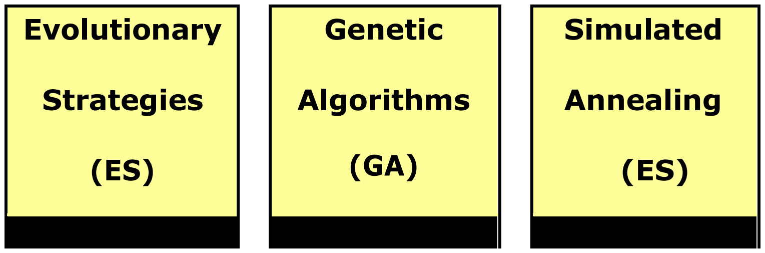


Fig Non- Linear search methods

10

- **Classical or Calculus based search**

  Uses deterministic approach to find best solutions of an optimization problem.

  - the solutions satisfy a set of necessary and sufficient conditions of the optimization problem.
  - the techniques are subdivide into **direct** and **indirect** methods.

    Direct or Numerical methods :

    - example : Newton or Fibonacci,
    - tries to find extremes by "hopping" around the search space and assessing the gradient of the new point, which guides the search.
    - applies the concept of "hill climbing", and finds the best

      local point by climbing the steepest permissible gradient.
    - used only on a restricted set of "well behaved" functions.

  - Indirect methods :

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

11

■ **Enumerative Search**

Here the search goes through every point (one point at a time) related to the function's domain space.

- At each point, all possible solutions are generated and tested to find optimum solution.
- It is easy to implement but usually require significant computation. Thus these techniques are not suitable for applications with large domain spaces.

In the field of artificial intelligence, the enumerative methods are subdivided into two categories : Uninformed and Informed methods.

- Uninformed or blind methods :
  - example: Mini-Max algorithm,
  - search all points in the space in a predefined order,
  - used in game playing.
- Informed methods :
  - example: Alpha-Beta and A* ,
  - does more sophisticated search
  - uses domain specific knowledge in the form of a cost function or heuristic to reduce cost for search.

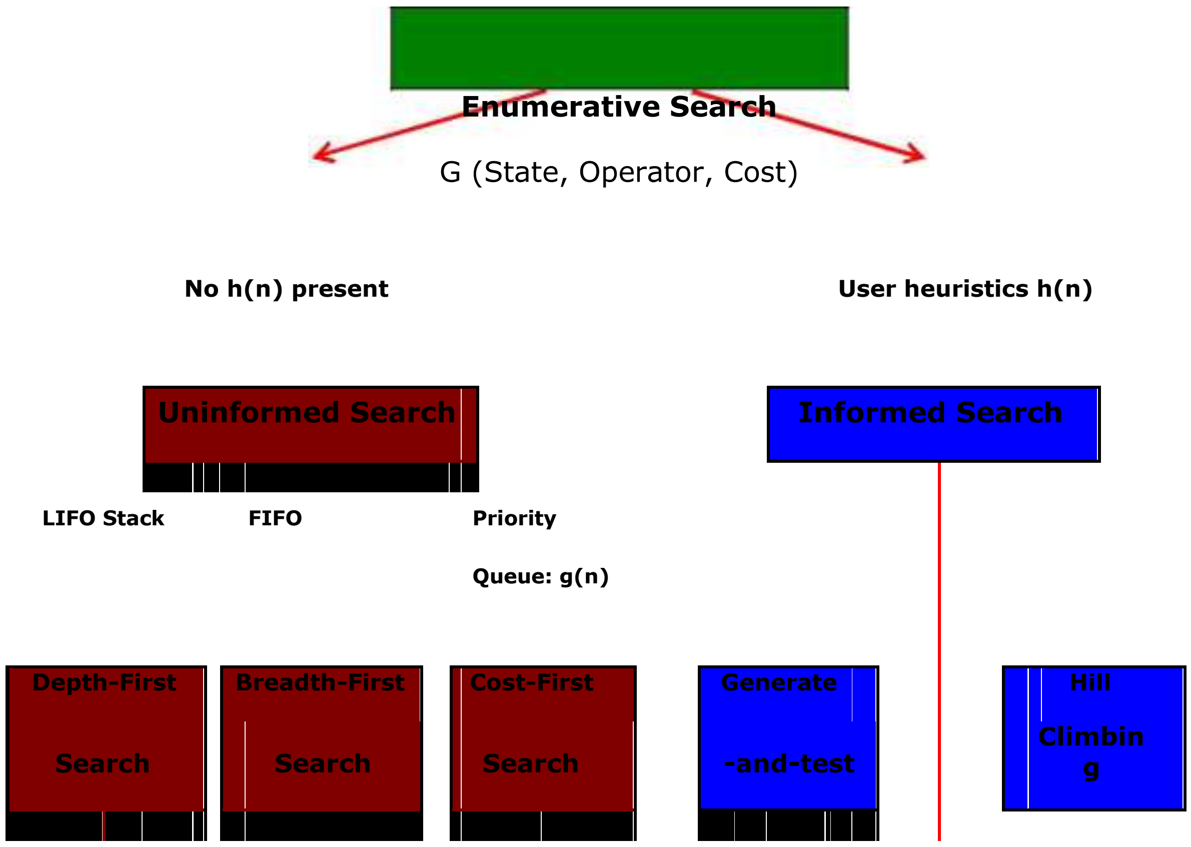
Next slide shows, the taxonomy of enumerative search in AI domain.

12

[Ref : previous slide Enumerative search]

The Enumerative search techniques follows, the traditional search and control strategies, in the domain of Artificial Intelligence.

- the search methods explore the search space "intelligently"; means evaluating possibilities without investigating every single possibility.
- there are many control structures for search; the depth-first search and breadth-first search are two common search strategies.
- the taxonomy of search algorithms in AI domain is given below.



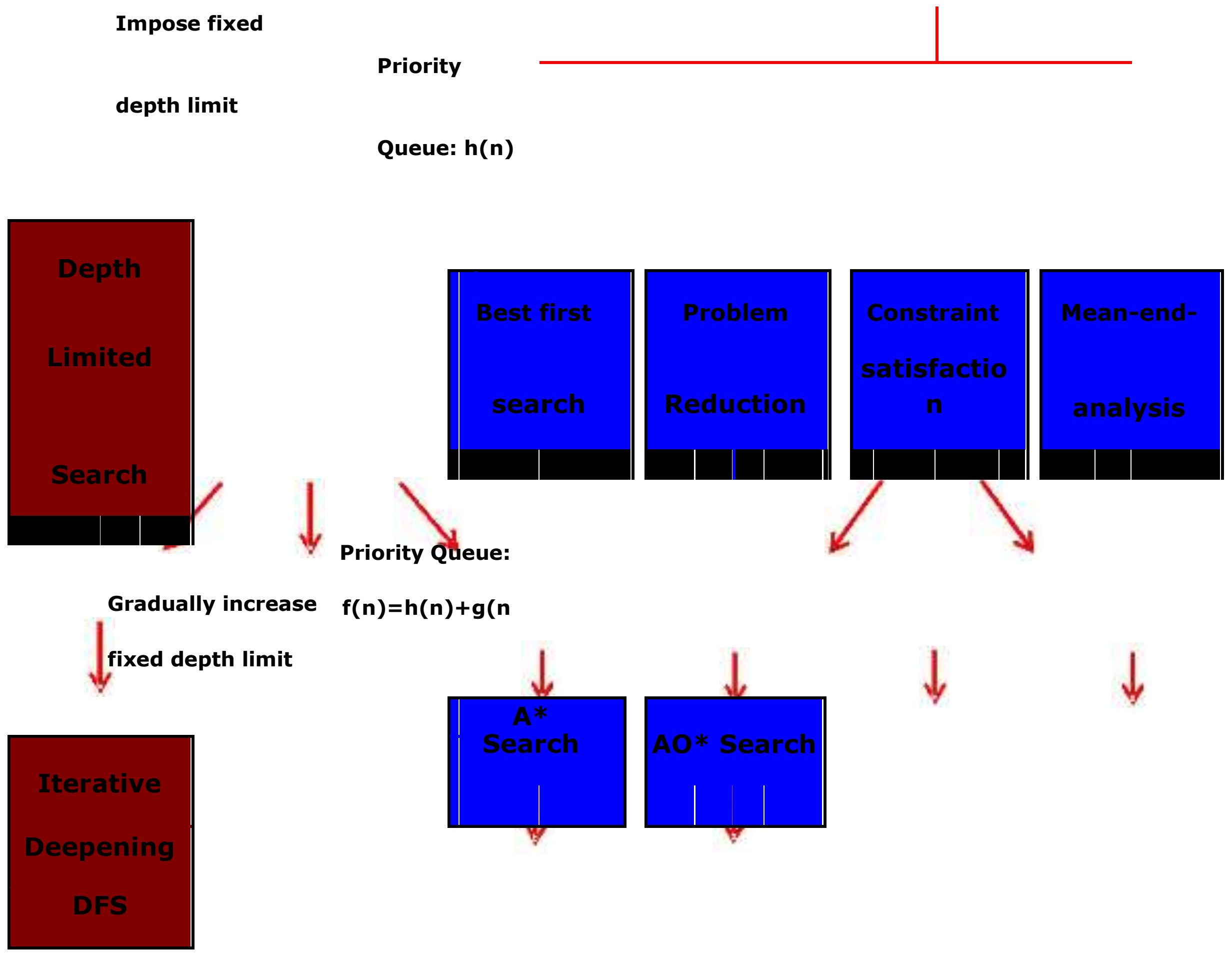


Fig. Enumerative Search Algorithms in AI Domain

13

- **Stochastic Search**

  Here the search methods, include heuristics and an element of randomness (non-determinism) in traversing the search space. Unlike

  the previous two search methodologies

  – the stochastic search algorithm moves from one point to another in

  the search space in a non-deterministic manner, guided by heuristics.

  – the stochastic search techniques are usually called Guided random search techniques.

  The stochastic search techniques are grouped into two major subclasses :

  – Simulated annealing and

  – Evolutionary algorithms.

  Both these classes follow the principles of evolutionary processes.

  - Simulated annealing (SAs)

    – uses a thermodynamic evolution process to search minimum energy states.

  - Evolutionary algorithms (EAs)

    – use natural selection principles.

– the 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.

The next slide shows, the taxonomy of evolutionary search algorithms. It includes the other two search, the Enumerative search and Calculus based techniques, for better understanding of Non-Linear search methodologies in its entirety.

14

- **Taxonomy of Search Optimization**

Fig. below shows different types of Search Optimization algorithms.

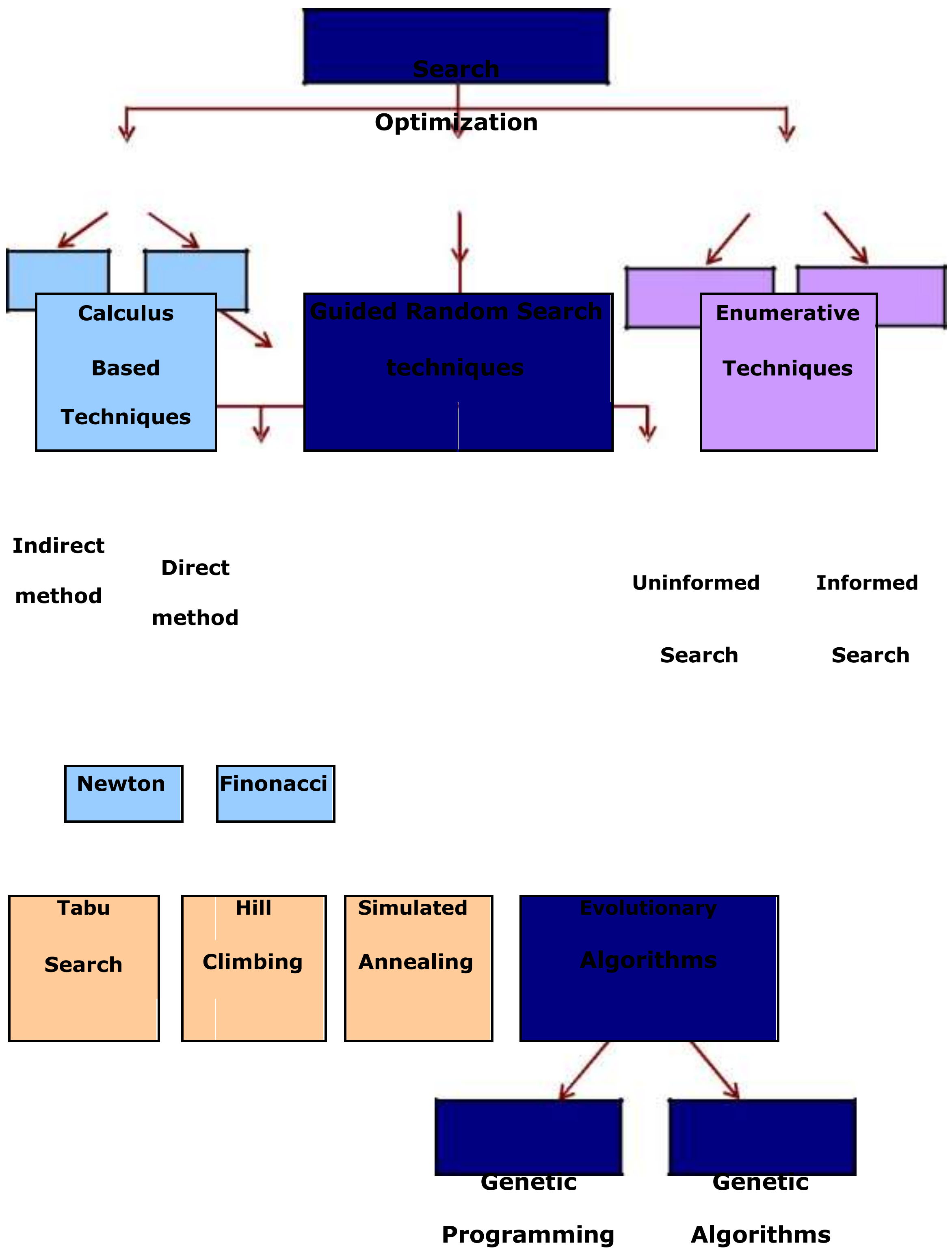


Fig. Taxonomy of Search Optimization techniques

We are interested in Evolutionary search algorithms.

Our main concern is to understand the evolutionary algorithms :

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

The Evolutionary Algorithms include :

- Genetic Algorithms and
- Genetic Programming

15

1.3 Evolutionary Algorithm (EAs)

Evolutionary Algorithm (EA) is a subset of Evolutionary Computation (EC) which is a subfield of Artificial Intelligence (AI).

Evolutionary Computation (EC) is a general term for several computational techniques. Evolutionary Computation represents powerful search and optimization paradigm influenced by biological mechanisms of evolution : that of natural selection and genetic.

Evolutionary Algorithms (EAs) refers to Evolutionary Computational

models using randomness and genetic inspired operations. EAs involve selection, recombination, random variation and competition of the individuals in a population of adequately represented potential solutions. The candidate solutions are referred as chromosomes or individuals.

Genetic Algorithms (GAs) represent the main paradigm of Evolutionary Computation.

- GAs simulate natural evolution, mimicking processes the nature uses : Selection, Crosses over, Mutation and Accepting.
- GAs simulate the survival of the fittest among individuals over consecutive generation for solving a problem.

Development History

$$\mathbf{EC} = \mathbf{GP} + \mathbf{ES} + \mathbf{EP} + \mathbf{GA}$$

Evolutionary Computing	**Genetic Programming**	**Evolution Strategies**	**Evolutionary Programming**	**Genetic Algorithms**
Rechenberg	Koza	Rechenberg	Fogel	Holland
1960	1992	1965	1962	1970

16

1.4 Genetic Algorithms (GAs) - Basic Concepts

Genetic algorithms (GAs) are the main paradigm of evolutionary computing. GAs are inspired by Darwin's theory about evolution – the "survival of the fittest". In nature, competition among individuals for scanty resources results in the fittest individuals dominating over the weaker ones.

- GAs are the ways of solving problems by mimicking processes nature uses; ie., Selection, Crosses over, Mutation and Accepting, to evolve a solution to a problem.

- GAs are adaptive heuristic search based on the evolutionary ideas of natural selection and genetics.

- GAs are intelligent exploitation of random search used in optimization problems.

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

The biological background (basic genetics), the scheme of evolutionary processes, the working principles and the steps involved in GAs are illustrated in next few slides.

■ Biological Background – Basic Genetics

Every **organism** has a set of rules, describing how that organism is built. All living organisms consist of **cells**.

In each cell there is same set of **chromosomes.** Chromosomes are strings of DNA and serve as a model for the whole organism.

A chromosome consists of **genes**, blocks of DNA.

Each gene encodes a particular protein that represents a **trait** (feature), e.g., color of eyes.

Possible settings for a trait (e.g. blue, brown) are called **alleles**.

Each gene has its own position in the chromosome called its **locus.**

Complete set of genetic material (all chromosomes) is called a **genome**.

Particular set of genes in a genome is called **genotype.**

The physical expression of the genotype (the organism itself after birth) is called the **phenotype**, its physical and mental characteristics, such as eye color, intelligence etc.

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. This process is called **recombination** (cross over) .

The new created offspring can then be mutated. **Mutation** means, that the elements of DNA are a bit changed. This changes are mainly caused by errors in copying genes from parents.

The **fitness** of an organism is measured by success of the organism in its life (survival).

18

[continued from previous slide - Biological background]

Below shown, the general scheme of evolutionary process in genetic along with pseudo-code.

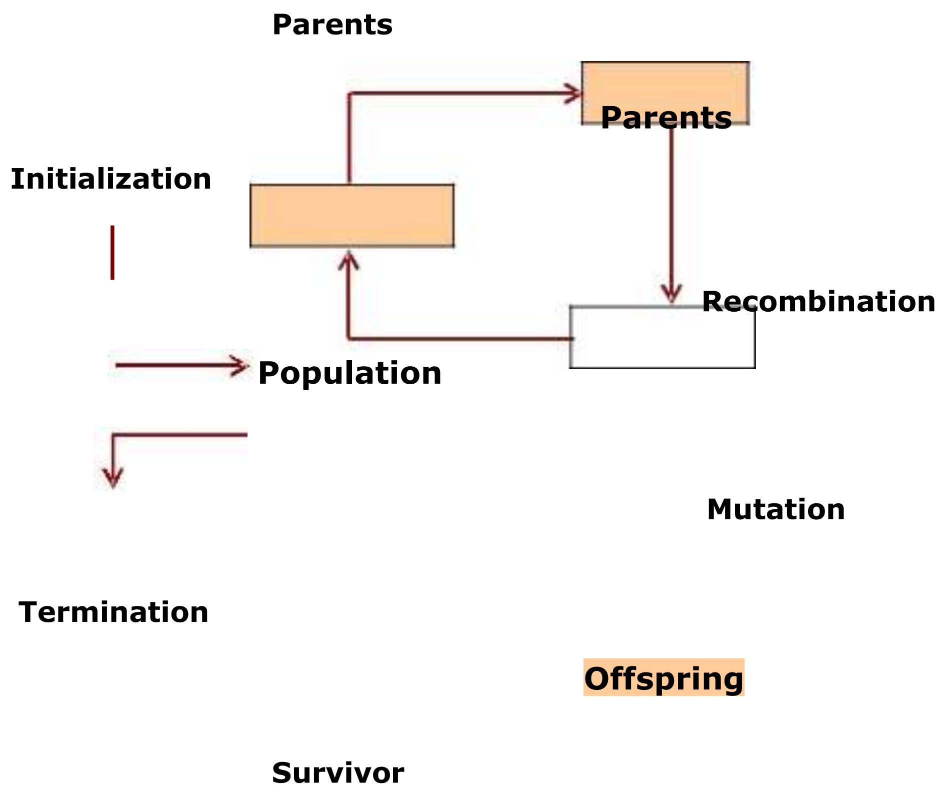


Fig. General Scheme of Evolutionary process

Pseudo-Code

```
BEGIN

INITIALISE  population with random candidate solution.

EVALUATE   each candidate;
```

```
REPEAT UNTIL  (termination condition ) is satisfied DO
    ■ SELECT parents;
    ■ RECOMBINE  pairs of parents;
    ■ MUTATE  the resulting offspring;
    ■ SELECT  individuals or the next generation;
END.
```

19

- **Search Space**

  In solving problems, some solution will be the best among others. The space of all feasible solutions (among which the desired solution

  resides) is called **search space** (also called state space).

  - Each point in the search space represents one **possible solution.**
  - Each possible solution can be "marked" by its value (or **fitness**) for the problem.
  - The GA looks for the **best solution** among a number of possible solutions represented by one point in the search space.
  - Looking for a solution is then equal to looking for some extreme value (minimum or maximum) in the search space.
  - At times the search space may be well defined, but usually only a few points in the search space are known.

  In using GA, the process of finding solutions generates other points (possible solutions) as evolution proceeds.

- **Working Principles**

Before getting into GAs, it is necessary to explain few terms.

- Chromosome : a set of genes; a chromosome contains the solution in form of genes.
- Gene : a part of chromosome; a gene contains a part of solution. It determines the solution. e.g. 16743 is a chromosome and 1, 6, 7, 4 and 3 are its genes.
- Individual : same as chromosome.
- Population: number of individuals present with same length of chromosome.
- Fitness : the value assigned to an individual based on how far or close a individual is from the solution; greater the fitness value better the solution it contains.
- Fitness function : a function that assigns fitness value to the individual. It is problem specific.
- Breeding : taking two fit individuals and then intermingling there chromosome to create new two individuals.
- Mutation : changing a random gene in an individual.
- Selection : selecting individuals for creating the next generation.

Working principles :

Genetic algorithm begins with a set of solutions (represented by chromosomes) called the population.

– Solutions from one population are taken and used to form a new population. This is motivated by the possibility that the new population will be better than the old one.

– Solutions are selected according to their fitness to form new solutions (offspring); more suitable they are, more chances they have to reproduce.

– This is repeated until some condition (e.g. number of populations or improvement of the best solution) is satisfied.

■ Outline of the Basic Genetic Algorithm

[Start] Generate random population of **n** chromosomes (i.e. suitable solutions for the problem).

[Fitness] Evaluate the fitness **f(x)** of each chromosome **x** in the population.

[New population] Create a new population by repeating following steps until the new population is complete.

[Selection] Select two parent chromosomes from a population according to their fitness (better the fitness, bigger the chance to be selected)

[Crossover] With a crossover probability, cross over the parents to form new offspring (children). If no crossover was performed, offspring is the exact copy of parents.

[Mutation] With a mutation probability, mutate new offspring at

each locus (position in chromosome).

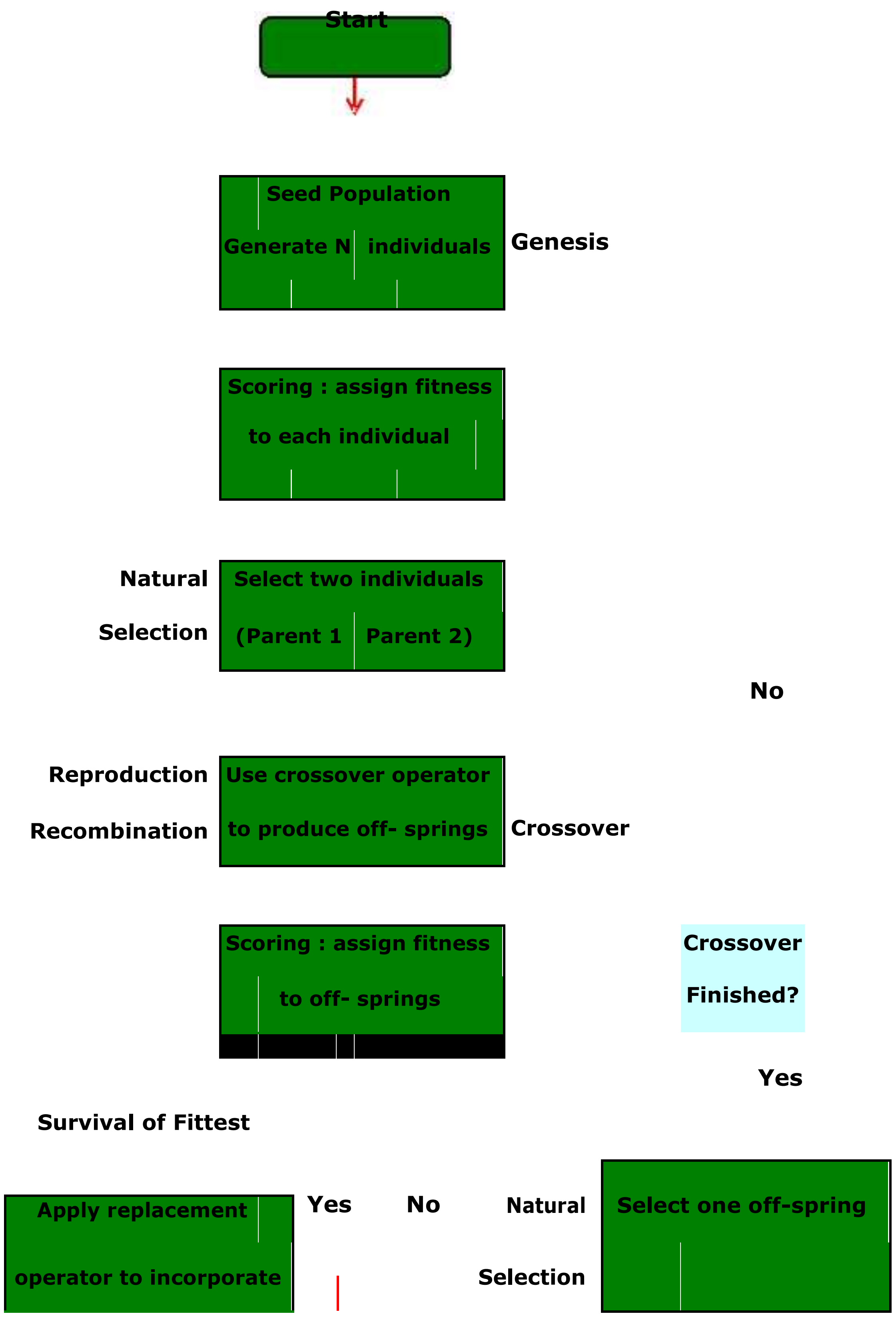
(d) [Accepting] Place new offspring in the new population

- [Replace] Use new generated population for a further run of the algorithm
- [Test] If the end condition is satisfied, stop, and return the best solution in current population
- [Loop] Go to step 2

Note : The genetic algorithm's performance is largely influenced by two operators called crossover and mutation. These two operators are the most important parts of GA.

22

- **Flow chart for Genetic Programming**



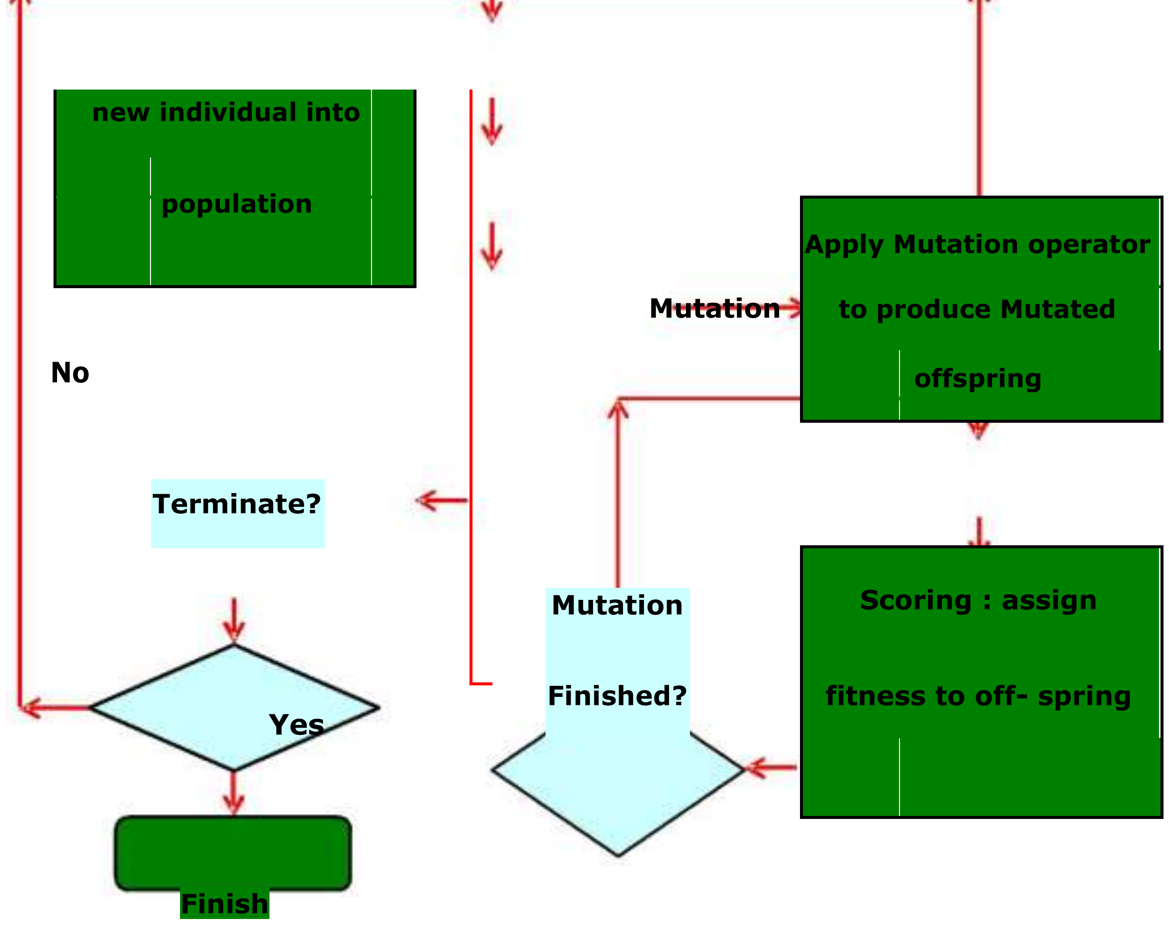


Fig. Genetic Algorithm – program flow chart

23

◇ Encoding

Before a genetic algorithm can be put to work on any problem, a method is needed to encode potential solutions to that problem in a form so that a computer can process.

- One common approach is to encode solutions as binary strings: sequences of **1's** and **0's**, where the digit at each position represents the value of some aspect of the solution.

  Example :

  A Gene represents some data (eye color, hair color, sight, etc.).

  a **Gene** looks like : **(11100010)**

  a **Chromosome** looks like:

Gene1	**Gene2**	**Gene3**	**Gene4**
(11000010,	**00001110,**	**001111010,**	**10100011)**

  A chromosome should in some way contain information about solution which it represents; it thus requires encoding. The most popular way of encoding is a **binary string** like :

  Chromosome 1 : 1101100100110110

  Chromosome 2 : 1101111000011110

  Each bit in the string represent some characteristics of the solution.

- There are many other ways of encoding, e.g., encoding values as integer or real numbers or some permutations and so on.

– The virtue of these encoding method depends on the problem to work on .

24

■ Binary Encoding

Binary encoding is the most common to represent information contained. In genetic algorithms, it was first used because of its relative simplicity.

– In binary encoding, every chromosome is a string of bits : **0** or **1**, like

Chromosome 1: 1 0 1 1 0 0 1 0 1 1 0 0 1 0 1 0 1 1 1 0 0 1 0 1

Chromosome 2: 1 1 1 1 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 1 1

– Binary encoding gives many possible chromosomes even with a small number of alleles ie possible settings for a trait (features).

– This encoding is often not natural for many problems and sometimes corrections must be made after crossover and/or mutation.

Example 1:

One variable function, say **0** to **15** numbers, numeric values, represented by 4 bit binary string.

Numeric value	4–bit string	Numeric value	4–bit string	Numeric value	4–bit string
0	0 0 0 0	6	0 1 1 0	12	1 1 0 0
1	0 0 0 1	7	0 1 1 1	13	1 1 0 1

2	0 0 1 0	8	1 0 0 0	14	1 1 1 0
3	0 0 1 1	9	1 0 0 1	15	1 1 1 1
4	0 1 0 0	10	1 0 1 0		
5	0 1 0 1	11	1 0 1 1		

25

[continued binary encoding]

Example 2 :

Two variable function represented by 4 bit string for each variable.

Let two variables **X_1 , X_2** as **(1011 0110)** .

Every variable will have both upper and lower limits as $X_i^L \le X_i \le X_i^U$

Because 4-bit string can represent integers from **0** to **15**,

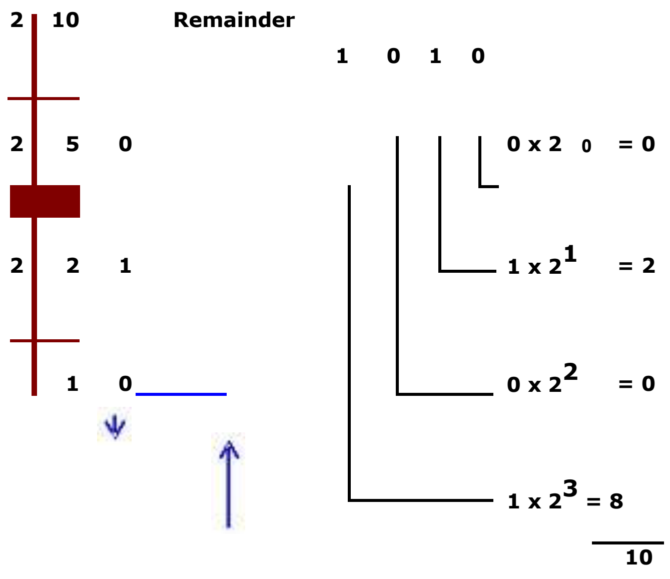
so **(0000 0000)** and **(1111 1111)** represent the points for **X_1 , X_2** as

(X_1^L, X_2^L) and (X_1^U, X_2^U) respectively.

Thus, an **n-bit** string can represent integers from

0 to **2^n -1**, i.e. **2^n** integers.

Binary Coding **Equivalent integer** **Decoded binary substring**



Let ***Xi*** is coded as a substring ***Si*** of length ***ni***. Then decoded binary substring ***Si*** is as

$$\sum_{k=0}^{K=ni-1} 2^k S_k$$

where ***Si*** can be **0** or **1** and the string ***S*** is represented as

S_{n-1} . . . . S_3 S_2 S_1 S_0

Example : Decoding value

Consider a 4-bit string **(0111)**,

– the decoded value is equal to

$$2^3 \times 0 + 2^2 \times 1 + 2^1 \times 1 + 2^0 \times 1 = 7$$

– Knowing X_i^L and X_i^U corresponding to **(0000)** and **(1111)** ,

the equivalent value for any 4-bit string can be obtained as

$$X_i = X_i^L + \frac{(X_i^U - X_i^L)}{(2^{n_i} - 1)} \times \text{(decoded value of string)}$$

– For e.g. a variable ***Xi*** ; let $X_i^L = 2$, and $X_i^U = 17$, find what value the 4-bit string ***Xi*** = **(1010)** would represent. First get decoded value for

$$S_i = 1010 = 2^3 \times 1 + 2^2 \times 0 + 2^1 \times 1 + 2^0 \times 0 = 10 \text{ then}$$

$$X_i = 2 + \frac{(17 - 2)}{(2^4 - 1)} \times 10 = 12$$

The accuracy obtained with a 4-bit code is **1/16** of search space.

By increasing the string length by 1-bit , accuracy increases to **1/32.**

26

- **Value Encoding**

The Value encoding can be used in problems where values such as real numbers are used. Use of binary encoding for this type of problems would be difficult.

In value encoding, every chromosome is a sequence of some values.

The Values can be anything connected to the problem, such as
: real numbers, characters or objects.

Examples :

Chromosome A 1.2324 5.3243 0.4556 2.3293 2.4545

Chromosome B ABDJEIFJDHDIERJFDLDFLFEGT

Chromosome C (back), (back), (right), (forward), (left)

- Value encoding is often necessary to develop some new types of crossovers and mutations specific for the problem.

- Permutation Encoding

Permutation encoding can be used in ordering problems, such as traveling salesman problem or task ordering problem.

In permutation encoding, every chromosome is a string of numbers that represent a position in a sequence.

Chromosome A 1 5 3 2 6 4 7 9 8

Chromosome B 8 5 6 7 2 3 1 4 9

☐ Permutation encoding is useful for ordering problems. For some problems, crossover and mutation corrections must be made to leave the chromosome consistent.

Examples :

1. The Traveling Salesman problem:

   There are cities and given distances between them. Traveling salesman has to visit all of them, but he does not want to travel more than necessary. Find a sequence of cities with a minimal traveled distance. Here, encoded chromosomes describe the order of cities the salesman visits.

2. The Eight Queens problem :

   There are eight queens. Find a way to place them on a chess board so that no two queens attack each other. Here, encoding describes the position of a queen on each row.

28

- **Tree Encoding**

Tree encoding is used mainly for evolving programs or expressions. For genetic programming :

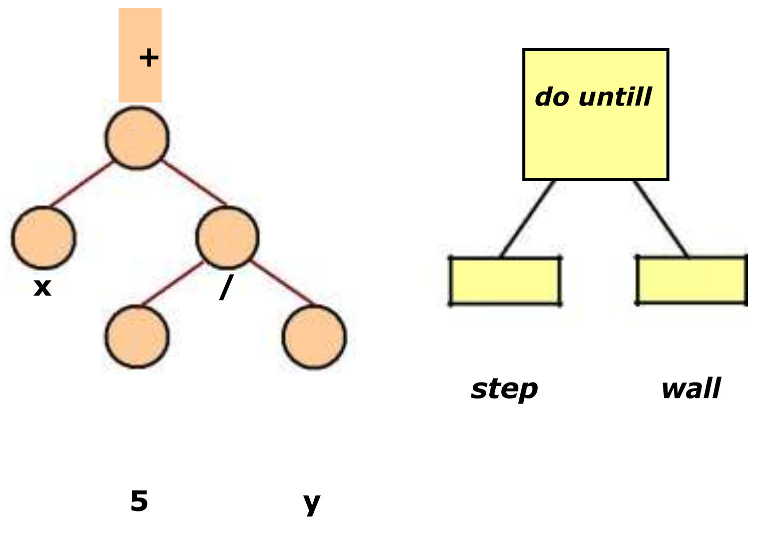
– In tree encoding, every chromosome is a tree of some objects, such as functions or commands in programming language.

– Tree encoding is useful for evolving programs or any other structures that can be encoded in trees.

– The crossover and mutation can be done relatively easy way .

Example :

Chromosome A **Chromosome B**



(+ x (/ 5 y)) **(do until step wall)**

Fig. Example of Chromosomes with tree encoding

Note : Tree encoding is good for evolving programs. The programming language LISP is often used. Programs in LISP can be easily parsed as a tree, so the crossover and mutation is relatively easy.

29

3. Operators of Genetic Algorithm

Genetic operators used in genetic algorithms maintain genetic diversity.

Genetic diversity or variation is a necessity for the process of evolution.

Genetic operators are analogous to those which occur in the natural world:

- **Reproduction** (or Selection) ;
- **Crossover** (or Recombination); and
- **Mutation**.

In addition to these operators, there are some parameters of GA.

One important parameter is **Population size.**

- Population size says how many chromosomes are in population (in one generation).
- If there are only few chromosomes, then GA would have a few possibilities to perform crossover and only a small part of search space is explored.
- If there are many chromosomes, then GA slows down.
- Research shows that after some limit, it is not useful to increase population size, because it does not help in solving the problem faster. The population size depends on the type of encoding and the problem.

30

3.1 Reproduction, or Selection

Reproduction is usually the first operator applied on population. From the population, the chromosomes are selected to be parents to crossover and produce offspring.

The problem is how to select these chromosomes ?

According to Darwin's evolution theory "survival of the fittest" – the best ones should survive and create new offspring.

- The Reproduction operators are also called Selection operators.
- Selection means extract a subset of genes from an existing population, according to any definition of quality. Every gene has a meaning, so one can derive from the gene a kind of quality measurement called **fitness function**. Following this quality (fitness value), selection can be performed.
- Fitness function quantifies the optimality of a solution (chromosome) so that a particular solution may be ranked against all the other solutions. The function depicts the closeness of a given 'solution' to the desired result.

Many reproduction operators exists and they all essentially do same thing. They pick from current population the strings of above average and insert their multiple copies in the mating pool in a probabilistic manner.

The most commonly used methods of selecting chromosomes for parents to crossover are :

- Roulette wheel selection,
- Boltzmann selection,
- Tournament selection,
- Rank selection
- Steady state selection.

The Roulette wheel and Boltzmann selections methods are illustrated next.

- **Example of Selection**

Evolutionary Algorithms is to maximize the function **$f(x) = x_2$** with **x** in the integer interval **[0 , 31]**, i.e., **x = 0, 1, . . . 30, 31.**

- The first step is encoding of chromosomes; use binary representation for integers; 5-bits are used to represent integers up to **31**.
- Assume that the population size is **4**.
- Generate initial population at random. They are chromosomes or genotypes; e.g., **01101, 11000, 01000, 10011**.
- Calculate fitness value for each individual.
  - Decode the individual into an integer (called phenotypes),

    01101 → 13; 11000 → 24; 01000 → 8; 10011 → 19;

  – Evaluate the fitness according to **$f(x) = x^2$** ,

    13 → 169; 24 → 576; 8 → 64; 19 → 361.

4. Select parents (two individuals) for crossover based on their fitness in **p_i**. Out of many methods for selecting the best chromosomes, if **roulette-wheel** selection is used, then the probability of the **i** th string in the population is $p_i = F_i \,/\, (\sum_{j=1}^{n} F_j)$, where

   F_i is fitness for the string **i** in the population, expressed as **f(x)**

   p_i is probability of the string **i** being selected,

4. is no of individuals in the population, is population size, **n=4**

n * p$_i$ is expected count

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

The string no 2 has maximum chance of selection.

32

- **Roulette wheel selection** (Fitness-Proportionate Selection)

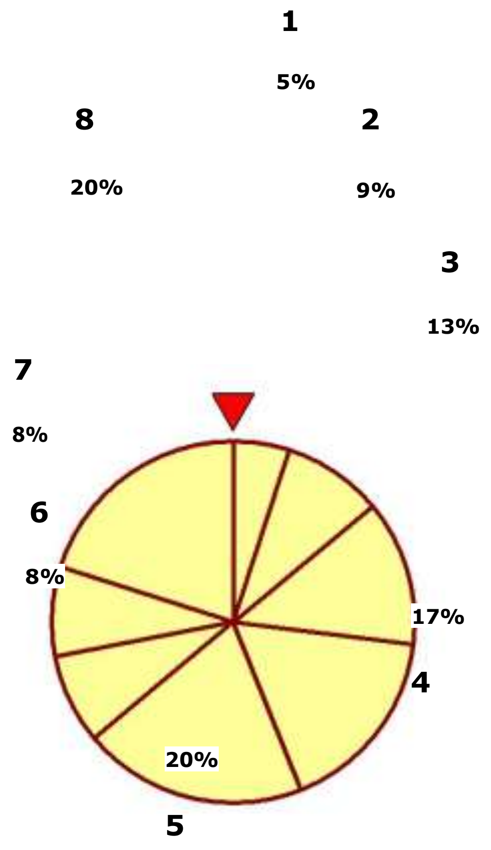
  Roulette-wheel selection, also known as Fitness Proportionate Selection, is a genetic operator, used for selecting potentially useful solutions for recombination.

  In fitness-proportionate selection :

  – the chance of an individual's being selected is proportional to its fitness, greater or less than its competitors' fitness.

  – conceptually, this can be thought as a game of Roulette.

Fig. Roulette-wheel Shows 8 individual with fitness



The Roulette-wheel simulates **8** individuals with fitness values **F**$_i$, marked at its circumference; e.g.,

– the **5**th individual has a higher fitness than others, so the wheel would choose the **5**th individual more than other individuals .

– the fitness of the individuals is calculated as the wheel is spun **n = 8** times, each time selecting an instance, of the string, chosen by the wheel pointer.

Probability of $\mathbf{i}^{th}$ string is $\mathbf{p_i = F_i / (\sum_{j=1}^{n} F_j)}$, where

• **= no of individuals**, called population size; **pi = probability** of $\mathbf{i}^{th}$ string being selected; **F**$_i$ **= fitness** for $\mathbf{i}^{th}$ string in the population.
Because the circumference of the wheel is marked according to a string's fitness, the Roulette-wheel mechanism is expected to make $\frac{F}{\bar{F}}$ copies of the $\mathbf{i}^{th}$ string.

Average fitness = $\bar{F}$ F_j / n ; **Expected count = (n =8) x pi**

Cumulative Probability$_5$ **=** $\sum_{i=1}^{N=5} p_i$

33

- **Boltzmann Selection**

  Simulated annealing is a method used to minimize or maximize a function.

  – This method simulates the process of slow cooling of molten metal to achieve the minimum function value in a minimization problem.

  – The cooling phenomena is simulated by controlling a temperature like parameter introduced with the concept of Boltzmann probability distribution.

  – The system in thermal equilibrium at a temperature **T** has its energy distribution based on the probability defined by

  $P(E) = \exp(-E / kT)$ were **k** is Boltzmann constant.

  – This expression suggests that a system at a higher temperature has

  almost uniform probability at any energy state, but at lower temperature it has a small probability of being at a higher energy state.

  – Thus, by controlling the temperature **T** and assuming that the search process follows Boltzmann probability distribution, the convergence of the algorithm is controlled.

3.2 Crossover

Crossover is a genetic operator that combines (mates) two chromosomes (parents) to produce a new chromosome (offspring). The idea behind crossover is that the new chromosome may be better than both of the parents if it takes the best characteristics from each of the parents. Crossover occurs during evolution according to a user-definable crossover probability. Crossover selects genes from parent chromosomes and creates a new offspring.

The Crossover operators are of many types.

– one simple way is, **One-Point crossover**.

– the others are **Two Point, Uniform, Arithmetic, and Heuristic crossovers.**

The operators are selected based on the way chromosomes are encoded.

- **One-Point Crossover**

  One-Point crossover operator randomly selects one crossover point and then copy everything before this point from the first parent and then everything after the crossover point copy from the second parent. The Crossover would then look as shown below.

  Consider the two parents selected for crossover.

  Parent 1 **1 1 0 1 1 | 0 0 1 0 0 1 1 0 1 1 0**

  Parent 2 **1 1 0 1 1 | 1 1 0 0 0 0 1 1 1 1 0**

  Interchanging the parents chromosomes after the crossover points -

  The Offspring produced are :

  Offspring 1 **1 1 0 1 1 | 1 1 0 0 0 0 1 1 1 1 0**

  Offspring 2 **1 1 0 1 1 | 0 0 1 0 0 1 1 0 1 1 0**

  Note : The symbol, a vertical line, **|** is the chosen crossover point.

► Two-Point Crossover

Two-Point crossover operator randomly selects two crossover points within a chromosome then interchanges the two parent chromosomes between these points to produce two new offspring.

Consider the two parents selected for crossover :

Parent 1	**1 1 0 1 1 | 0 0 1 0 0 1 1 | 0 1 1 0**
Parent 2	**1 1 0 1 1 | 1 1 0 0 0 0 1 | 1 1 1 0**

Interchanging the parents chromosomes between the crossover points -

The Offspring produced are :

Offspring 1	**1 1 0 1 1 | 0 0 1 0 0 1 1 | 0 1 1 0**
Offspring 2	**1 1 0 1 1 | 0 0 1 0 0 1 1 | 0 1 1 0**

37

► Uniform Crossover

Uniform crossover operator decides (with some probability – know as the mixing ratio) which parent will contribute how the gene values in the offspring chromosomes. The crossover operator allows the parent chromosomes to be mixed at the gene level rather than the segment level (as with one and two point crossover).

Consider the two parents selected for crossover.

Parent 1	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**0**
Parent 2	**1**	**1**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**0**

If the mixing ratio is **0.5** approximately, then half of the genes in the offspring will come from parent **1** and other half will come from parent **2**. The possible set of offspring after uniform crossover would be:

Offspring 1	1_1	1_2	0_2	1_1	1_1	1_2	1_2	0_2	0_1	0_1	0_2	1_1	1_2	1_1	1_1	0_2
Offspring 2	1_2	1_1	0_1	1_2	1_2	0_1	0_1	1_1	0_2	0_2	1_1	1_2	0_1	1_2	1_2	0_1

Note: The subscripts indicate which parent the gene came from.

- **Arithmetic**

  Arithmetic crossover operator linearly combines two parent chromosome vectors to produce two new offspring according to the equations:

  Offspring1 = a * Parent1 + (1- a) * Parent2

  Offspring2 = (1 – a) * Parent1 + a * Parent2

  where **a** is a random weighting factor chosen before each crossover operation.

  Consider two parents (each of 4 float genes) selected for crossover:

Parent 1	**(0.3)**	**(1.4)**	**(0.2)**	**(7.4)**
Parent 2	**(0.5)**	**(4.5)**	**(0.1)**	**(5.6)**

  Applying the above two equations and assuming the weighting factor **a = 0.7**, applying above equations, we get two resulting offspring. The possible set of offspring after arithmetic crossover would be:

Offspring 1	**(0.36)**	**(2.33)**	**(0.17)**	**(6.87)**
Offspring 2	**(0.402)**	**(2.981)**	**(0.149)**	**(5.842)**

ii Heuristic

Heuristic crossover operator uses the fitness values of the two parent chromosomes to determine the direction of the search.

The offspring are created according to the equations:

Offspring1 = BestParent + r * (BestParent – WorstParent)

Offspring2 = BestParent

where **r** is a random number between **0** and **1**.

It is possible that **offspring1** will not be feasible. It can happen if **r** is chosen such that one or more of its genes fall outside of the allowable upper or lower bounds. For this reason, heuristic crossover has a user defined parameter **n** for the number of times to try and find an **r** that results in a feasible chromosome. If a feasible chromosome is not produced after **n** tries, the worst parent is returned as offspring1.

3.3 Mutation

After a crossover is performed, mutation takes place.

Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of chromosomes to the next.

Mutation occurs during evolution according to a user-definable mutation probability, usually set to fairly low value, say **0.01** a good first choice.

Mutation alters one or more gene values in a chromosome from its initial state. This can result in entirely new gene values being added to the gene pool. With the new gene values, the genetic algorithm may be able to arrive at better solution than was previously possible.

Mutation is an important part of the genetic search, helps to prevent the population from stagnating at any local optima. Mutation is intended to prevent the search falling into a local optimum of the state space.

The Mutation operators are of many type.

- one simple way is, **Flip Bit**.
- the others are **Boundary, Non-Uniform, Uniform, and Gaussian**.

The operators are selected based on the way chromosomes are encoded .

41

■ **Flip Bit**

The mutation operator simply inverts the value of the chosen gene. i.e. **0** goes to **1** and **1** goes to **0**.

This mutation operator can only be used for binary genes.

Consider the two original off-springs selected for mutation.

Original offspring 1	**1 1 0 1 1 1 1 0 0 0 0 1 1 1 1 0**
Original offspring 2	**1 1 0 1 1 0 0 1 0 0 1 1 0 1 1 0**

Invert the value of the chosen gene as **0** to **1** and **1** to **0**

The Mutated Off-spring produced are :

Mutated offspring 1	**1 1 0 0 1 1 1 0 0 0 0 1 1 1 1 0**
Mutated offspring 2	**1 1 0 1 1 0 1 1 0 0 1 1 0 1 0 0**

- **Boundary**

  The mutation operator replaces the value of the chosen gene with either the upper or lower bound for that gene (chosen randomly).

  This mutation operator can only be used for integer and float genes.

- **Non-Uniform**

  The mutation operator increases the probability such that the amount of the mutation will be close to **0** as the generation number increases. This mutation operator prevents the population from stagnating in the early stages of the evolution then allows the genetic algorithm to fine tune the solution in the later stages of evolution.

  This mutation operator can only be used for integer and float genes.

- **Uniform**

  The mutation operator replaces the value of the chosen gene with a uniform random value selected between the user-specified upper and lower bounds for that gene.

  This mutation operator can only be used for integer and float genes.

- **Gaussian**

  The mutation operator adds a unit Gaussian distributed random value to the chosen gene. The new gene value is clipped if it falls outside of the user-specified lower or upper bounds for that gene.

  This mutation operator can only be used for integer and float genes.

■ **Basic Genetic Algorithm :**

Examples to demonstrate and explain : Random population, Fitness, Selection, Crossover, Mutation, and Accepting.

Example 1 :

Maximize the function **$f(x) = x^2$** over the range of integers from **0 . . . 31**.

Note : This function could be solved by a variety of traditional methods such as a hill-climbing algorithm which uses the derivative. One way is to :

- Start from any integer **x** in the domain of **f**
- Evaluate at this point **x** the derivative **f'**
- Observing that the derivative is **+ve**, pick a new **x** which is at a small distance in the **+ve** direction from current **x**
- Repeat until **x = 31**

See, how a genetic algorithm would approach this problem ?

Genetic Algorithm approach to problem - Maximize the function **$f(x) = x^2$**

1. Devise a means to represent a solution to the problem :

Assume, we represent **x** with five-digit unsigned binary integers.

2. Devise a heuristic for evaluating the fitness of any particular solution :

The function **f(x)** is simple, so it is easy to use the **f(x)** value itself to rate the fitness of a solution; else we might have considered a more simpler heuristic that would more or less serve the same purpose.

3. Coding - Binary and the String length :

GAs often process binary representations of solutions. This works well, because crossover and mutation can be clearly defined for binary solutions. A Binary string of length **5** can represents 32 numbers (0 to 31).

4. Randomly generate a set of solutions :

Here, considered a population of four solutions. However, larger populations are used in real applications to explore a larger part of the search. Assume, four randomly generated solutions as : **01101, 11000, 01000, 10011**. These are chromosomes or genotypes.

5. Evaluate the fitness of each member of the population :

The calculated fitness values for each individual are -

(a) Decode the individual into an integer (called phenotypes),

01101 → 13; 11000 → 24; 01000 → 8; 10011 → 19;

(b) Evaluate the fitness according to $\mathbf{f(x) = x^2}$,

13 → 169; 24 → 576; 8 → 64; 19 → 361.

(c) Expected count = **N * Prob i** , where **N** is the number of individuals in the population called population size, here **N = 4**.

Thus the evaluation of the initial population summarized in table below .

String No i	Initial Population (chromosome)	X value (Pheno types)	Fitness $f(x) = x^2$	Prob i (fraction of total)	Expected count N * Prob i
1	0 1 1 0 1	13	169	0.14	0.58
2	1 1 0 0 0	24	576	0.49	1.97
3	0 1 0 0 0	8	64	0.06	0.22
4	1 0 0 1 1	19	361	0.31	1.23
Total (sum)			1170	1.00	4.00
Average			293	0.25	1.00

Max		**576**	**0.49**	**1.97**

Thus, the string no 2 has maximum chance of selection.

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

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

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

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

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

7. Randomly pair the members of the new generation

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

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

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

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

Crossing these two strings at that point yields:

0 1 1 0 1 $\Rightarrow$ **0 1 1 0 |1** $\Rightarrow$ **0 1 1 0 0**

1 1 0 0 0 $\Rightarrow$ **1 1 0 0 |0** $\Rightarrow$ **1 1 0 0 1**

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

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

Crossing these two strings at that point yields:

1 1 0 0 0 $\Rightarrow$ **1 1 |0 0 0** $\Rightarrow$ **1 1 0 1 1**

1 0 0 1 1 $\Rightarrow$ **1 0 |0 1 1** $\Rightarrow$ **1 0 0 0 0**

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

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

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

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

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

Total (sum)		**1754**	**1.000**	**4.000**
Average		**439**	**0.250**	**1.000**
Max		**729**	**0.415**	**1.660**

Observe that :

1. Initial populations : At start step 5 were

0 1 1 0 1 , 1 1 0 0 0 , 0 1 0 0 0 , 1 0 0 1 1

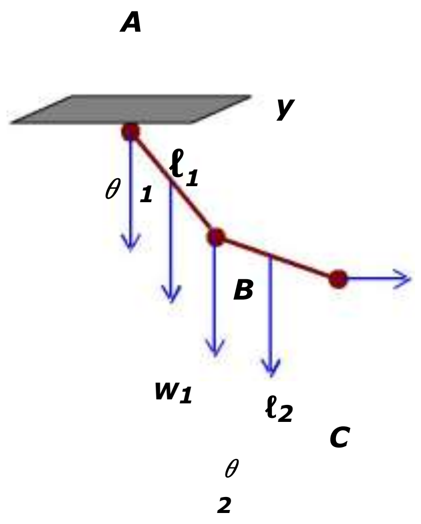
After one cycle, new populations, at step 10 to act as initial population

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

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

Example 2 : Two bar pendulum

Two uniform bars are connected by pins at ***A*** and ***B*** and supported at ***A***. Let a horizontal force ***P*** acts at ***C***.



Given : Force ***P* = 2**, Length of bars **ℓ_1 = 2 ,** **ℓ_2 = 2**, Bar weights ***W_1*= 2**, ***W_2* = 2** . angles **= *X_i***

Find : Equilibrium configuration of the system if fiction at all joints are neglected ?

P **Solution** : Since there are two unknowns θ_1 and θ_2, we use 4 – bit binary for each unknown.

x^U - x^L **90 - 0**

W_2

Accuracy = ----------- = --------- = 6^0

2^4 - 1   15

Fig. Two bar pendulum

Hence, the binary coding and the corresponding angles X_i are given as

$$X_i = X_i^L + \frac{X_i^U - X_i^L}{2^4 - 1} S_i$$

where ***Si*** is decoded Value of the ***i*** ***th*** chromosome.

e.g. the 6th chromosome binary code **(0 1 0 1)** would have the corresponding angle given by $S_i = 0\ 1\ 0\ 1 = 2^3 \times 0 + 2^2 \times 1 + 2^1 \times 0 + 2^0 \times 1 = 5$

$$X_i = 0 + \frac{90 - 0}{15} \times 5 = 30$$

The binary coding and the angles are given in the table below.

S. No.	***Binary code*** ***Si***	***Angle*** ***Xi***	*S. No.*	***Binary code*** ***Si***	***Angle*** ***Xi***
1	**0 0 0 0**	**0**	9	**1 0 0 0**	**48**
2	**0 0 0 1**	**6**	10	**1 0 0 1**	**54**
3	**0 0 1 0**	**12**	11	**1 0 1 0**	**60**
4	**0 0 1 1**	**18**	12	**1 0 1 1**	**66**
5	**0 1 0 0**	**24**	13	**1 1 0 0**	**72**
6	**0 1 0 1**	**30**	14	**1 1 0 1**	**78**
7	**0 1 1 0**	**36**	15	**1 1 1 0**	**84**
8	**0 1 1 1**	**42**	16	**1 1 1 1**	**90**

Note : The total potential for two bar pendulum is written as

$(c) = - P[(\ell_1 \sin\theta_1 + \ell_2 \sin\theta_2)] - (W_1 \ell_1 /2)\cos\theta_1 - W_2 [(\ell_2 /2) \cos\theta_2 + \ell_1 \cos\theta_1]$ (Eq.1)

Substituting the values for **P**, W_1, W_2, ℓ_1, ℓ_2 all as **2** , we get **,**

$\Pi(\theta_1, \theta_2) = - 4 \sin\theta_1 - 6 \cos\theta_1 - 4 \sin\theta_2 - 2 \cos\theta_2$ **= function *f*** (Eq. 2)

θ_1, θ_2 **lies** **between 0 and 90** **both inclusive ie** $0 \le \theta_1, \theta_2 \le 90$ (Eq. 3)

Equilibrium configuration is the one which makes **Π** a minimum .

Since the objective function is **–ve** , instead of minimizing the function ***f*** let us maximize $-f = f'$. The maximum value of $f' = 8$ when θ_1 and θ_2 are zero.

Hence the ***fitness function F*** is given by $F = -f - 7 = f' - 7$ (Eq. 4)

48

First randomly generate ***8*** population with ***8*** bit strings as shown in table below.

Population No.	***Population of 8 bit strings (Randomly generated)***		***Corresponding Angles (from table above)***		***F = – f – 7***
			θ 1 ,	***θ 2***	
1	**0 0 0 0**	**0 0 0 0**	**0**	**0**	**1**
2	**0 0 1 0**	**0 0 0 0**	**12**	**6**	**2.1**
3	**0 0 0 1**	**0 0 0 0**	**6**	**30**	**3.11**
4	**0 0 1 0**	**1 0 0 0**	**12**	**48**	**4.01**
5	**0 1 1 0**	**1 0 1 0**	**36**	**60**	**4.66**
6	**1 1 1 0**	**1 0 0 0**	**84**	**48**	**1.91**
7	**1 1 1 0**	**1 1 0 1**	**84**	**78**	**1.93**
8	**0 1 1 1**	**1 1 0 0**	**42**	**72**	**4.55**

These angles and the corresponding to fitness function are shown below.

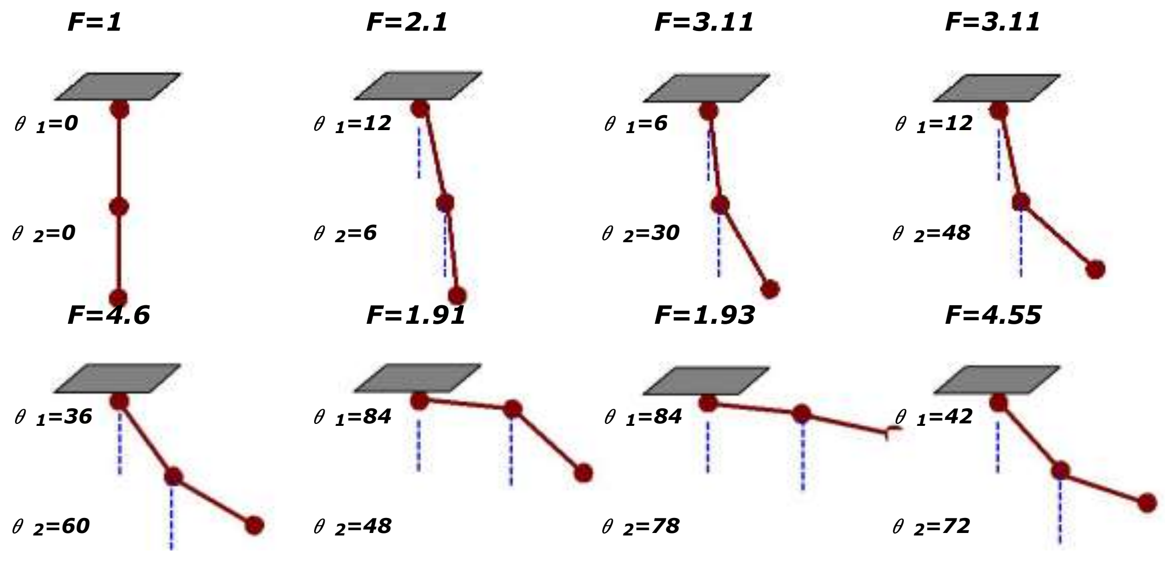


Fig.   Fitness function F for various population

The above Table and the Fig. illustrates that :

– GA begins with a population of random strings.

– Then, each string is evaluated to find the fitness value.
– The population is then operated by three operators –

– The new population is further evaluated tested for termination.

– If the termination criteria are not met, the population is iteratively operated by the three operators and evaluated until the termination criteria are met.

– One cycle of these operation and the subsequent evaluation procedure is known as a *Generation* in GA terminology.

Hybrid Systems

Integration of NN FL GA

What is Hybridization ?

- Hybrid systems employ more than one technology to solve a problem.
- Hybridization of technologies can have pitfalls and therefore need to be done with care.

— If one technology can solve a problem then a hybrid technology ought to be used only if its application results in a better solution.

- Hybrid systems have been classified as :
  - *Sequential hybrid system*: the technologies are used in pipelining fashion;
  - *Auxiliary hybrid system*: the one technology calls the other technology as subroutine;
  - *Embedded hybrid system* : the technologies participating appear to be fused totally.
- Hybridization of fuzzy logic, neural networks, genetic algorithms has led to creation of a perspective scientific trend known as soft computing.
  - *Neural networks* mimic our ability to adapt to circumstances and learn from past experience,

– *Fuzzy logic* addresses the imprecision or vagueness in input and output,
– *Genetic algorithms* are inspired by biological evolution, can systemize

random search and reach to optimum characteristics.

- Each of these technologies have provided efficient solution to wide range of problems belonging to different domains. However, each of these technologies has advantages and disadvantages. It is therefore appropriate that Hybridization of these three technologies are done so as to over come the weakness of one with the strength of other.

☐ **Introduction :**

Hybridization - Integration of NN , FL , and GA

Fuzzy logic, Neural networks and Genetic algorithms are soft computing methods which are inspired by biological computational processes and nature's problem solving strategies.

Neural Networks (NNs) are highly simplified model of human nervous system which mimic our ability to adapt to circumstances and learn from past experience. Neural Networks systems are represented by different architectures like single and multilayer feed forward network. The networks offers back proposition generalization, associative memory and adaptive resonance theory.

Fuzzy logic addresses the imprecision or vagueness in input and output description of the system. The sets have no crisp boundaries and provide a gradual transition among the members and non-members of the set elements.

Genetic algorithms are inspired by biological evolution, can systemize random search and reach to optimum characteristics.

Each of these technologies have provided efficient solution to wide range of problems belonging to different domains. However, each of these technologies suffer from advantages and disadvantages.

It is therefore appropriate that Hybridization of these three technologies are done so as to over come the weakness of one with the strength of other.

1.1 Hybrid Systems

Hybrid systems employ more than one technology to solve a problem.

Hybridization of technologies can have pitfalls and therefore need to be done with care. If one technology can solve a problem then a hybrid technology ought to be used only if its application results in a better solution. Hybrid systems have been classified as *Sequential , Auxiliary and Embedded*.

In *Sequential hybrid system*, the technologies are used in pipelining fashion.

In *Auxiliary hybrid system*, one technology calls the other technology as subroutine.

In *Embedded* hybrid system, the technologies participating appear to be fused totally.

- **Sequential Hybrid System**

In Sequential hybrid system, the technologies are used in pipelining fashion. Thus, one technology's output becomes another technology's input and it goes on. However, this is one of the weakest form of hybridization since an integrated combination of technologies is not present.

Example: A Genetic algorithm preprocessor obtains the optimal parameters for different instances of a problem and hands over the preprocessed data to a neural network for further processing.

- **Auxiliary Hybrid System**

  In Auxiliary hybrid system, one technology calls the other technology as subroutine to process or manipulate information needed. The second technology processes the information provided by the first and hands it over for further use. This type of hybridization is better than the sequential hybrids.

  Example : A neuron-genetic system in which a neural network employs a genetic algorithm to optimize its structural parameters that defines its architecture.

- **Embedded Hybrid System**

  In Embedded hybrid system, the technologies participating are integrated in such a manner that they appear intertwined. The fusion is so complete that it would appear that no technology can be used without the others for solving the problem.

  Example : A NN-FL hybrid system may have an NN which receives fuzzy inputs, processes it and extracts fuzzy outputs as well.

1.2 Neural Networks, Fuzzy Logic, and Genetic Algorithms Hybrids

Neural Networks, Fuzzy Logic, and Genetic Algorithms are three distinct technologies.

Each of these technologies has advantages and disadvantages. It is therefore appropriate that hybridization of these three technologies are done so as to over come the weakness of one with the strength of other.

- **Neuro-Fuzzy Hybrid**

Neural Networks and Fuzzy logic represents two distinct methodologies to deal with uncertainty. Each of these has its own merits and demerits.

Neural Networks :

- Merits : Neural Networks, can model complex nonlinear relationships and are appropriately suited for classification phenomenon into predetermined classes.

- Demerits : Neural Network's output, precision is often limited to least squares errors; the training time required is quite large; the training data has to be chosen over entire range where the variables are expected to change.

Fuzzy logic :

– Merits : Fuzzy logic system, addresses the imprecision of inputs and outputs defined by fuzzy sets and allow greater flexibility in formulating detail system description.

Integration of NN and FL, called Neuro-Fuzzy systems, have the potential to extend the capabilities of the systems beyond either of these two technologies applied individually. The integrated systems have turned out to be useful in :

– accomplishing mathematical relationships among many variables in a complex dynamic process,

– performing mapping with some degree of imprecision, and

– controlling nonlinear systems to an extent not possible with conventional linear control systems.

There are two ways to do hybridization :

– One, is to provide NNs with fuzzy capabilities, there by increasing the network's expressiveness and flexibility to adapt to uncertain environments.

– Second, is to apply neuronal learning capabilities to fuzzy systems so that the fuzzy systems become more adaptive to changing environments. This method is called NN driven fuzzy reasoning.

- **Neuro-Genetic Hybrids**

The Neural Networks and Genetic Algorithms represents two distinct methodologies.

Neural Networks : can learn various tasks from examples, classify phenomena and model nonlinear relationships.

Genetic Algorithms : have offered themselves as potential candidates for the optimization of parameters of NN.

Integration of GAs and NNs has turned out to be useful.

- Genetically evolved nets have reported comparable results against their conventional counterparts.
- The gradient descent learning algorithms have reported difficulties in

  leaning the topology of the networks whose weights they optimize.
- GA based algorithms have provided encouraging results especially with regard to face recognition, animal control, and others.
- Genetic algorithms encode the parameters of NNs as a string of properties of the network, i.e. chromosomes. A large population of chromosomes representing many possible parameters sets, for the given NN, is generated.
- GA-NN is also known as GANN have the ability to locate the neighborhood of the optimal solution quicker than other conventional search strategies.
- The drawbacks of GANN algorithms are : large amount of memory required to handle and manipulate chromosomes for a given network; the question is whether this problem scales as the size of the networks become large.

•

Fuzzy - Genetic Hybrids

Fuzzy systems have been integrated with GAs.

The fuzzy systems like NNs (feed forward) are universal approximator in the sense that they exhibit the capability to approximate general nonlinear functions to any desired degree of accuracy.

The adjustments of system parameters called for in the process, so that the system output matches the training data, have been tackled using GAs. Several parameters which a fuzzy system is involved with like input/output variables and the membership function that define the fuzzy systems, have been optimized using GAs.

1.3 Typical Hybrid Systems

The Systems considered are listed below.

1. Genetic algorithm based back propagation network (Neuro Genetic Hybrid)
2. Fuzzy back propagation network (Neuro – Fuzzy Hybrid with Multilayer Feed forward Network as the host architecture)
3. Simplified Fuzzy ARTMAP

   (Neuro – Fuzzy Hybrid with Recurrent Network as the host architecture)
4. Fuzzy Associative Memory

   (Neuro – Fuzzy Hybrid with single layer Feed forward architecture)
5. Fuzzy logic controlled Genetic algorithm (Fuzzy – Genetic Hybrid)

- **Genetic Algorithm (GA) based Back Propagation Network (BPN)**

Neural networks (NNs) are the adaptive system that changes its structure based on external or internal information that flows through the network. Neural network solve problems by self-learning and self-organizing.

Back Propagation Network (BPN) is a method of training multi-layer neural networks. Here learning occurs during this training phase.

The steps involved are:

- The pattern of activation arriving at the output layer is compared with the correct output pattern to calculate an error signal.
- The error signal is then back-propagated from output to input for adjusting the weights in each layer of the BPN.
- The Back-Propagation searches on the error surface using gradient descent method to minimize error $\mathbf{E = 1/2 \Sigma (T_j - O_j)^2}$ where $\mathbf{T_j}$ is target output and $\mathbf{O_j}$ is the calculated output by the network.

Limitations of BPN :

- BPN can recognize patterns similar to those they have learnt, but do not have the ability to recognize new patterns.
- BPN must be sufficiently trained to extract enough general features applicable to both seen and unseen; over training to network may have undesired effects.

Genetic Algorithms (GAs) are adaptive search and optimization algorithms, mimic the principles of nature.

- GAs are different form traditional search and
- Optimization exhibit simplicity, ease of operation, minimal requirements, and global perspective.

Hybridization of BPN and GAs

- The BPN determines its weight based on gradient search technique and therefore it may encounter a local minima problem.

– GAs do not guarantee to find global optimum solution, but are good in finding quickly good acceptable solution.

– Therefore, hybridization of BPN and GAs are expected to provide many advantages compare to what they alone can**.**

The GA based techniques for determining weights in a BPN are explained next.

15

2.1 GA based techniques for determining weights in a BPN

Genetic algorithms work with population of individual strings.

The steps involved in GAs are:

- each individual string represent a possible solution of the problem considered,
- each individual string is assigned a fitness value,
- high fit individuals participate in reproduction, yields new strings as offspring and they share some features with each parents,
- low fit individuals are kept out from reproduction and so die,
- a whole new population of possible solutions to the problem is generated by selecting high fit individuals from current generation,
- this new generation contains characteristics which are better than their ancestors,
- processing this way after many generation, the entire population inherits the best and fit solution.

However, before a GA is executed :

- a suitable **coding** for the problem is devised,

- a **fitness function** is formulated,

- parents have to be **selected** for reproduction and crossover to generate offspring.

All these aspects of GAs for determining weights of BPN are illustrated in next few slides.

16

- **Coding**

Assume a BPN configuration ***ℓ - m – n*** where

– ***ℓ*** is input , ***m*** is hidden and ***n*** is output neurons.

– number of weights to be determined are ***(ℓ + n) m.***

– each weight (gene) is a real number.

– assume number of digits (gene length) in weight are ***d*** .

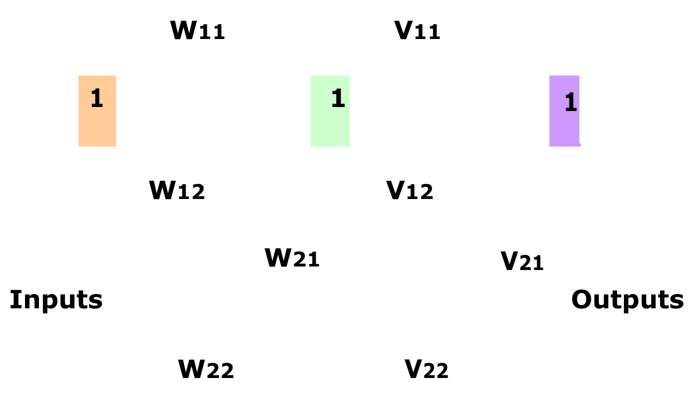
– a string **S** represents weight matrices of input-hidden and the hidden-output layers in a linear form arranged as row-major or column-major selected.

– population size is the randomly generated initial population of ***p*** chromosomes.

Example :

Consider a BPN configuration ***ℓ - m – n*** where ***ℓ* = 2** is input , ***m* = 2** is hidden and ***n* = 2** is output neuron.

Input neuron Hidden neurons output neurons



– number of weights is **(ℓ + n) m**

= (2 + 2) . 2 = 8

– each weight is real number and assume number of digits in weight are **d = 5**

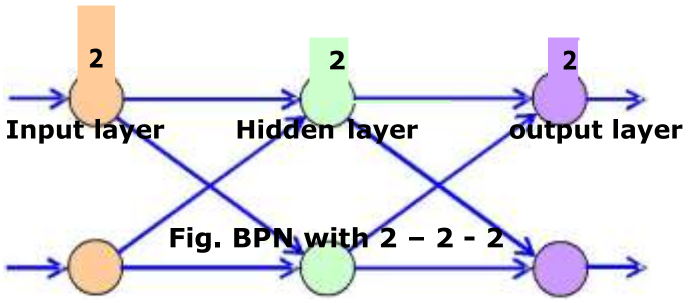


Fig. BPN with 2 – 2 - 2

– string **S** representing chromosome of weights is **8 x 5 = 40** in length

– Choose a population size **p = 40** ie choose **40** chromosomes

Gene ***Gene*** ***Gene*** ***Gene*** ***Gene*** ***Gene*** ***Gene*** ***Gene***

← **k=0** → ← **k=1** → ← **k=2** → ← ← **k=3** → ← **k=4** → **k=5** → ← **k=6** → ← **k=7** →

84321 46234 78901 32104 42689 63421 46421 87640

← **Chromosome** →

32478 76510 02461 84753 64321 14261 87654 12367

Chromosome

← →

Fig. Some randomly generated chromosome made of 8 genes representing 8 weights for BPN

17

- **Weight Extraction**

Extract weights from each chromosomes, later to determine the fitness values.

Let $x_1, \; x_2, \; \ldots \; x_d, \; \ldots \; x_L$ represent a chromosome and

Let $x_{kd+1}, x_{kd+2}, \ldots x_{(k+1)d}$ represent kth gene $(k \geq 0)$ in the chromosomes.

The actual weight w_k is given by

$$w_k = \begin{cases} + \dfrac{x_{kd+2}\,10^{d-2} + x_{kd+3}\,10^{d-3} + \ldots + x_{(k+1)d}}{10^{d-2}}, & \text{if } 5 \leq x_{kd+1} \leq 9 \\[2ex] - \dfrac{x_{kd+2}\,10^{d-2} + x_{kd+3}\,10^{d-3} + \ldots + x_{(k+1)d}}{10^{d-2}}, & \text{if } 0 \leq x_{kd+1} < 5 \end{cases}$$

Example : *[Ref Fig. BPN previous slide]*

The Chromosomes are stated in the Fig. The weights extracted from all the eight genes are :

- **Gene 0 : 84321** ,

  Here we have, $k = 0, d = 5$, and x_{kd+1} is x_1 such that

  $5 \leq x_1 = 8 \leq 9$. Hence, the weight extracted is

$$W_0 = + \frac{4 \times 10^3 + 3 \times 10^2 + 2 \times 10 + 1}{10^3} = +4.321$$

- **Gene 1 : 46234** ,

  Here we have, $k = 1, d = 5$, and x_{kd+1} is x_6 such that

$0 \leq x_6 = 4 \leq 5$. Hence, the weight extracted is

$$W_1 = -\frac{6 \times 10^3 + 2 \times 10^2 + 3 \times 10 + 4}{10^3} = -6.234$$

- Similarly for the remaining genes

Gene 2 :	**78901**	**yields**	**W2 = + 8.901**
Gene 3 :	**32104**	**yields**	**W3 = – 2.104**
Gene 4 :	**42689**	**yields**	**W4 = – 2.689**
Gene 5 :	**63421**	**yields**	**W5 = + 3.421**
Gene 6 :	**46421**	**yields**	**W6 = – 6.421**
Gene 7 :	**87640**	**yields**	**W7 = + 7.640**

18

- **Fitness Function :**

A fitness is devised for each problem.

Example :

The matrix on the right, represents a set of input **I** and output **T** for problem **P** to be solved.

$$\left\{\begin{matrix} (I_{11}, I21) & (T_{11}, T21) \\ (I_{12}, I22) & (T_{12}, T22) \\ (I_{13}, I23) & (T_{13}, T23) \end{matrix}\right\}$$

Generate initial population $\mathbf{P_0}$ of size $\mathbf{p = 40}$.

Let $\mathbf{C^0_1}$, $\mathbf{C^0_1}$, . . . , $\mathbf{C^0_{40}}$ represent the 40 chromosomes.

Let $\bar{W}^0{}_1$, $\bar{W}^0{}_2$, ... $\bar{W}^0{}_{40}$ be the weight sets extracted, using the Eq. in the previous slides, from each of the chromosome $\mathbf{C^0_i}$, $\mathbf{i = 1, 2, . . . , 40}$.

Let $\bar{O}^0{}_1$, $\bar{O}^0{}_2$, $\bar{O}^0{}_3$ be the calculated outputs of BPN.

Compute root mean square error :

$$\mathbf{E_1 = (T_{11} - O_{11})^2 + (T_{21} - O_{21})^2} ,$$

$$\mathbf{E_2 = (T_{12} - O_{12})^2 + (T_{22} - O_{22})^2}$$

$$\mathbf{E_3 = (T_{13} - O_{13})^2 + (T_{23} - O_{23})^2}$$

The root mean square of error is

$$\mathbf{E = [(E_1 + E_2 + E_3) / 3]^{1/2}}$$

Compute Fitness $\mathbf{F_1}$:

The fitness $\mathbf{F_1}$ for the chromosome $\mathbf{C_{01}}$ is given by

$$\mathbf{F_1 = 1 / E}$$

Similarly, find the fitness $\mathbf{F_2}$ for the chromosome $\mathbf{C_{02}}$ and so on the fitness $\mathbf{F_n}$ for the chromosome $\mathbf{C_{0n}}$

Algorithm

```
{
Let ( $\overline{I_i}$ , $\overline{T_i}$ ) , $i = 1, 2, ..., N$ represents the input-output pairs of the
problem to be solved by BPN with configuration $\ell - m - n$ ; where
    $\overline{I} = (I_{1i}, I_{2i}, ..., I_{\ell i})$ and
    $\overline{T}_i = (T_{1i}, T_{2i}, ..., T_{ni})$
For each chromosome $C_i$ , $i = 1, 2, ..., p$ belonging to current the
population $P_i$ whose size is $p$
 {
   Extract weights $\overline{w}_i$ form $C_i$ using Eq. 2.1 in previous slide;
   Keeping $\overline{w}_i$ as a fixed weight, train the BPN for the N input instances;
   Calculate error $E_i$ for each of the input instances using the formula below
   $E_i = \sum_j (T_{ji} - O_{ji})^2$ where $\overline{O}_i$ is the output vector calculated by BPN;
   Find the root mean square E of the errors $E_i$ , $i = 1, 2, ..., N$
   i.e. $E = ((\sum_i E_i) / N)^{1/2}$
   Calculate the Fitness value $F_i$ for each of the individual string of the
   population as $F_i = 1 / E$
 }
Output $F_i$ for each $C_i$ , $i = 1, 2, ..., p$ ;
```

}

Thus the Fitness values **Fi** for all chromosomes in the initial population are computed. The population size is **p = 40**, so **F i , i = 1**

▪ **2 , . . , 40** are computed.

A schematic for the computation of fitness values is illustrated below.

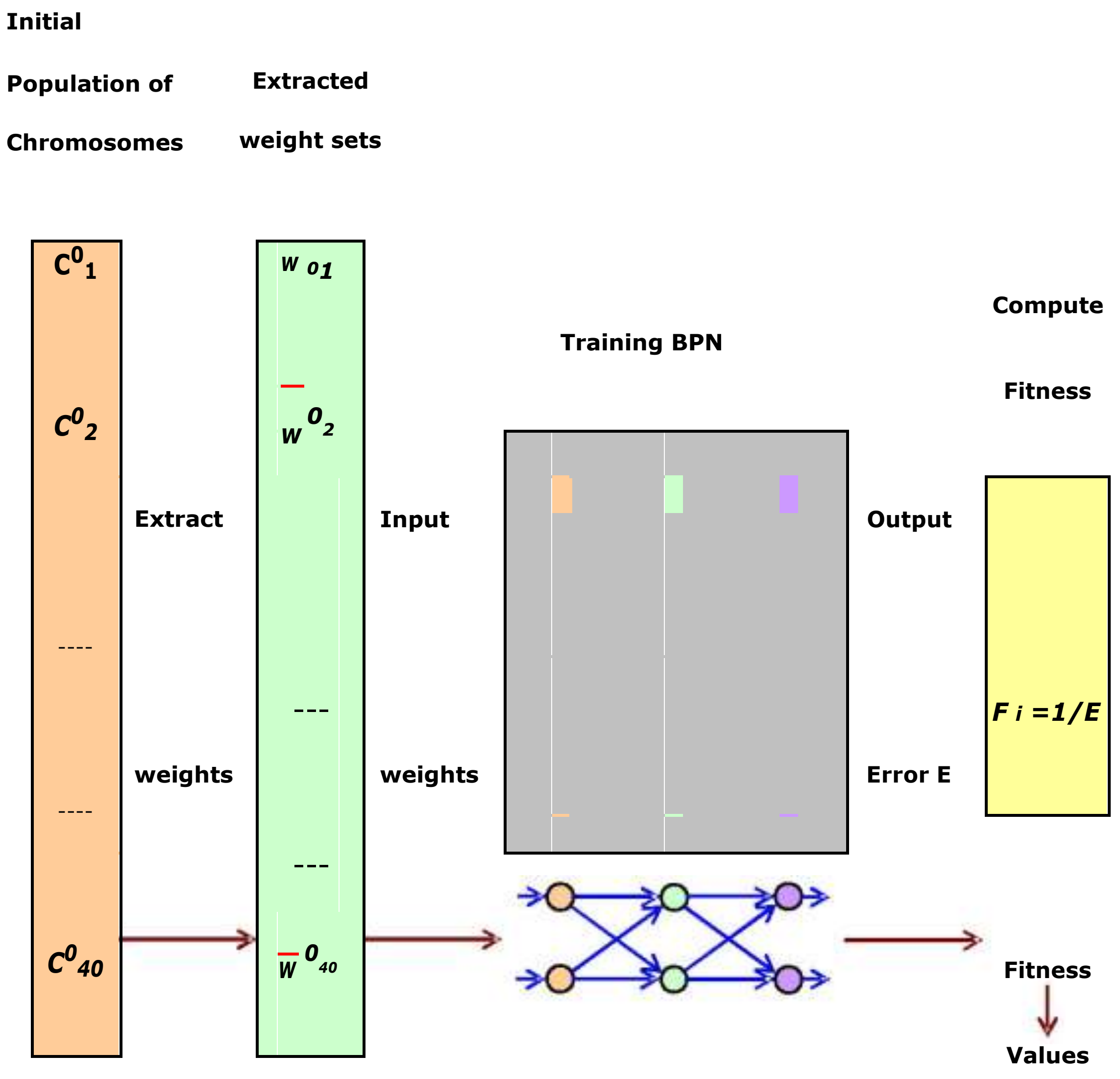


Fig. Computation of Fitness values for the population

21

- **Reproduction of Offspring**

  Before the parent chromosomes reproduce offspring :

  First, form a mating pool by excluding that chromosome $\boldsymbol{C_\ell}$ with least fitness $\mathbf{F_{min}}$ and then replacing it with a duplicate copy of $\mathbf{C_k}$ with highest fitness $\mathbf{F_{max}}$;

  i.e., the best fit individuals have multiple copies while worst fit individuals die off.

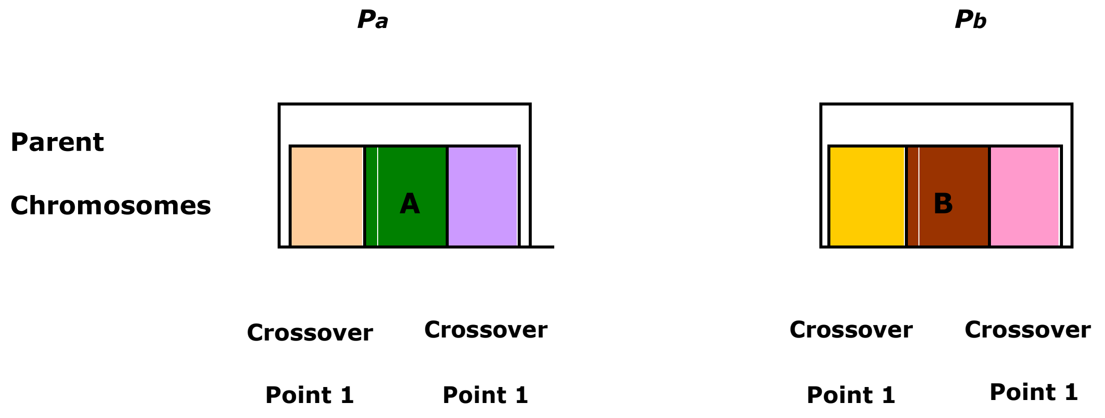
  Having formed the mating pool, select parent pair at random. Chromosomes of respective pairs are combined using crossover operator. Fig. below shows :

  – two parent chromosomes **Pa** and **Pb**,

  – the two point crossover,

  – exchange of gene segments by the parent pairs, and

  – the offspring **Oa** and **Ob** are produced.



Offspring

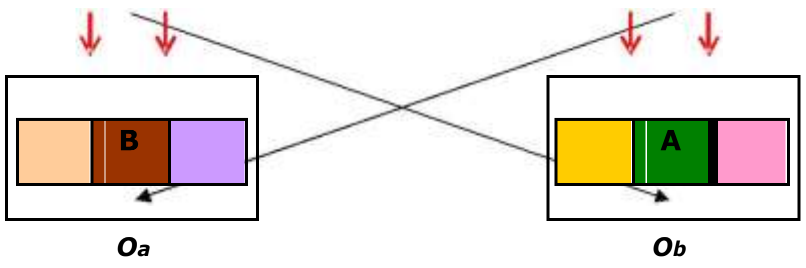


Fig. Two – point crossover operator

22

Example :

– Consider the initial population of chromosomes P_0 generated, with their fitness value F_i, *where* $i = 1, 2, \ldots, 40$,

– Let $F_{max} = F_k$ be maximum and $F_{min} = F_\ell$ be minimum fitness value for $1 \le \ell, k \le 40$ where $\ell \ne k$

– Replace all chromosomes having fitness value F^{min} with copies of chromosomes having fitness value F^{max}

Fig. below illustrates the Initial population of chromosomes and the formation of the mating pool.

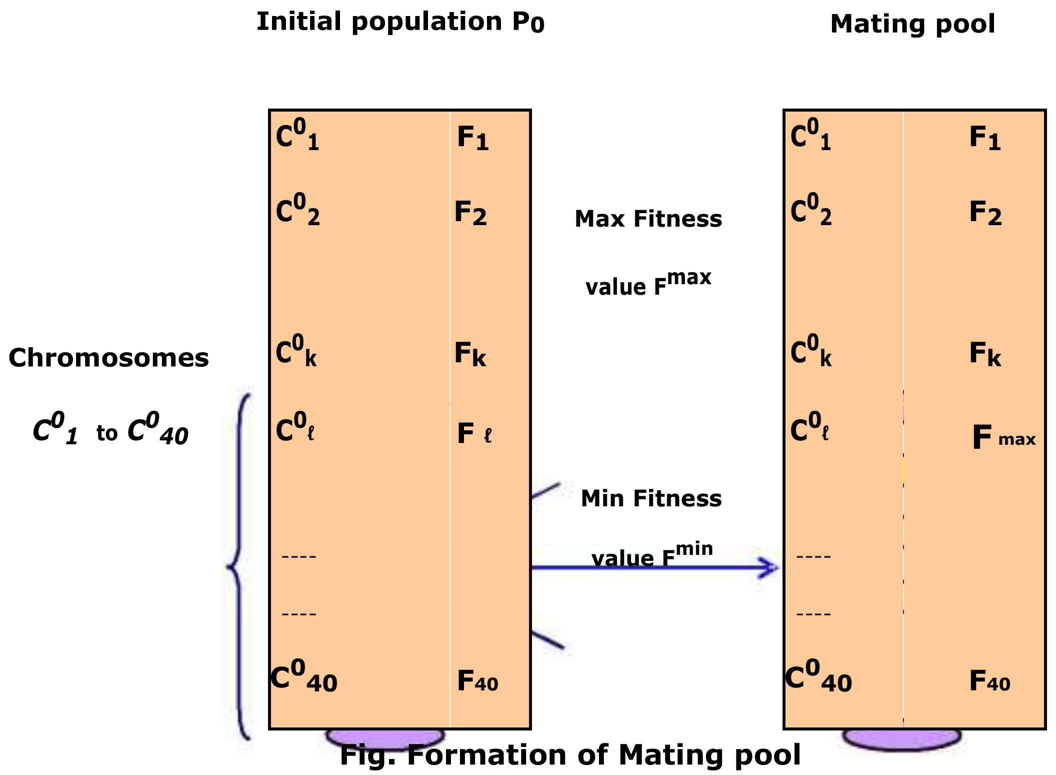


Fig. Formation of Mating pool

F_{min} is replaced by F^{max}

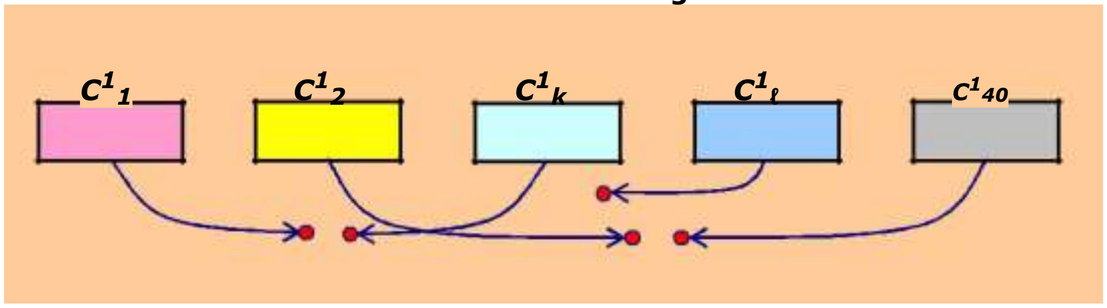
k Selection of Parent Chromosomes

The previous slide illustrated Reproduction of the Offspring.

Here, sample *"Selection Of Parents"* for the *"Two Points Crossover"* operator

to produce Offspring Chromosomes are illustrated.

Chromosomes - Mating Pool

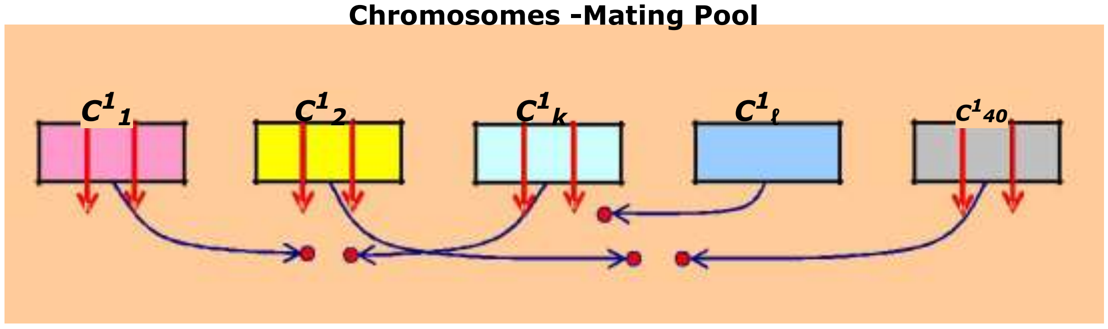


Selected Parent Pairs

Fig. Random Selection of Parent Chromosomes

The Crossover Points of the Chromosomes are randomly chosen for each parent pairs as shown in the Fig. below.

Chromosomes -Mating Pool



Crossover

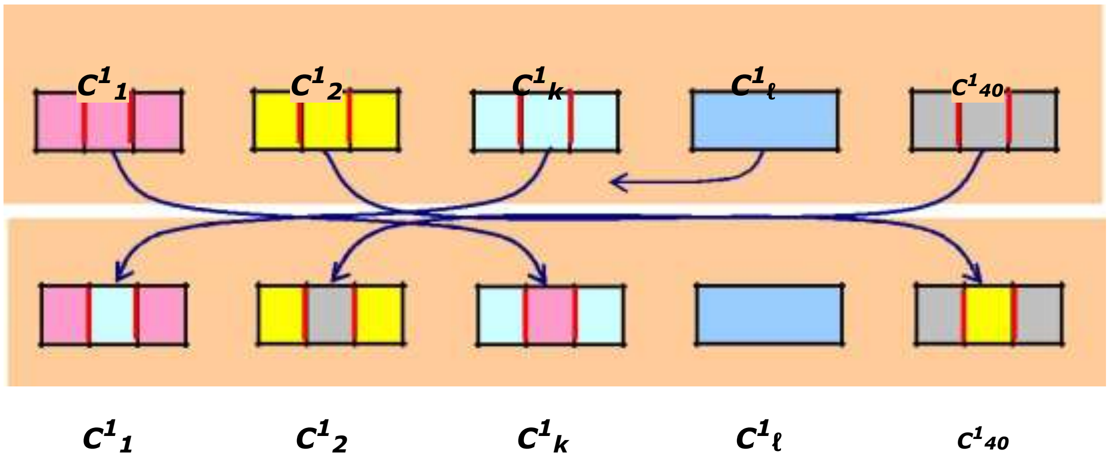
points

Selected Parent Pairs

Fig. Randomly chosen Crossover points of Parent Chromosomes

The Genes are exchanged for *Mutation* as shown in the Fig. below.

Chromosomes -Mating Pool



New Population P1

Fig. New population P_1 after application of two point Crossover operator

Thus new population $\mathbf{P_1}$ is created comprising **40** Chromosomes which are the Offspring of the earlier population generation $\mathbf{P_0}$.

24

- **Convergence**

  For any problem, if GA is correctly implemented, the population evolves over successive generations with fitness value increasing towards the global optimum.

  Convergence is the progression towards increasing uniformity.

  A population is said to have converged when 95% of the individuals constituting the population share the same fitness value.

  Example :

  Let a population $\mathbf{P_1}$ undergoes the process of selection, reproduction, and crossover.

  - the fitness values for the chromosomes in $\mathbf{P_1}$ are computed.
  - the best individuals replicated and the reproduction carried out using two-point crossover operators form the next generation $\mathbf{P_2}$ of the chromosomes.
  - the process of generation proceeds until at one stage 95% of the chromosomes in the population $\mathbf{P_i}$ converge to the same fitness value.
  - at that stage, the weights extracted from the population $\mathbf{P_i}$ are the final weights to be used by BPN.

- **Fuzzy Back Propagation Network**

Neural Networks and Fuzzy logic (NN-FL) represents two distinct methodologies and the integration of NN and FL is called Neuro-Fuzzy systems.

Back Propagation Network (BPN) is a method of training multi-layer neural networks where learning occurs during this training phase.

Fuzzy Back Propagation Network (Fuzzy-BPN) is a hybrid architecture. It is, Hybridization of BPN by incorporating fuzzy logic.

Fuzzy-BPN architecture, maps fuzzy inputs to crisp outputs. Here, the Neurons uses LR-type fuzzy numbers.

The Fuzzy-Neuron structure, the architecture of fuzzy BP, its learning mechanism and algorithms are illustrated in next few slides.

3.1 LR-type Fuzzy Numbers

The LR-type fuzzy number are special type of representation of fuzzy numbers. They introduce functions called **L** and **R**.

- **Definition**

A fuzzy member $\tilde{\mathbf{M}}$ is of **L-R** type if and only if

$$\mu_{\tilde{M}}(x) = \begin{cases} L \dfrac{m-x}{\alpha} & \text{for } x \le m, \quad \alpha > 0 \\ R \dfrac{m-x}{} & \text{for } x \le m, \quad \beta > 0 \end{cases}$$

β

where **L** is a left reference

R is a right reference, **~**

m , is called mean of **M** is a real number,

α , **β** are left and right spreads respectively.

$\mu_{\tilde{M}}$ is the membership function of fuzzy member $\tilde{M}$

The functions **L** and **R** are defined as follows:

$$L\left(\frac{m-x}{\alpha}\right) = \max\left(0, 1 - \frac{m-x}{\alpha}\right)$$

$$R\left(\frac{m-x}{\alpha}\right) = \max\left(0, 1 - \left[\frac{m-x}{\alpha}\right]\right)$$

LR-type fuzzy number $\tilde{M}$ can be represented as **(m, α, β)** $_{LR}$ shown below.

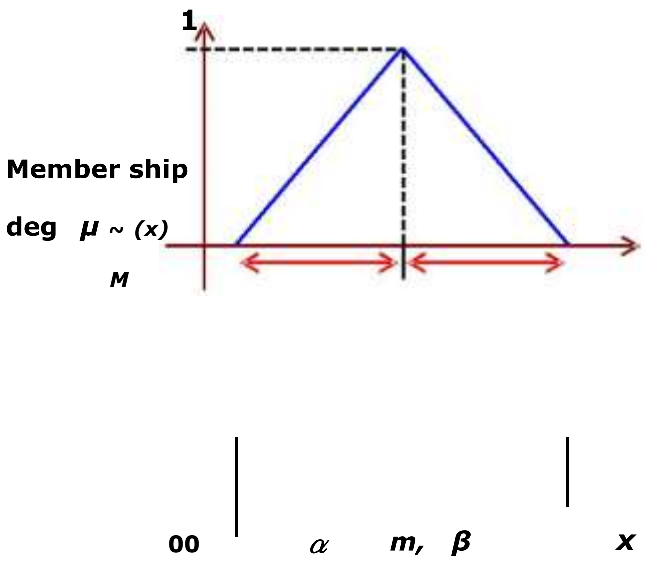


Fig. A triangular fuzzy number (m, α, β).

Note : If α and **β** are both zero, then **L-R** type function indicates a crisp value. The choice of **L** and **R** functions is specific to problem.

- **Operations on LR-type Fuzzy Numbers**

Let $\tilde{M} = (m, \alpha, \beta)_{LR}$ and $\tilde{N} = (n, \gamma, \delta)_{LR}$ be two L R-type fuzzy numbers. The basic operations are

- **Addition**

$$(m, \alpha, \beta)_{LR} \oplus (n, \gamma, \delta)_{LR} = (m + n, \alpha + \gamma, \beta + \delta)_{LR}$$

- **Substraction**

$$(m, \alpha, \beta)_{LR} \ominus (n, \gamma, \delta)_{LR} = (m - n, \alpha + \delta, \beta + \gamma)_{LR}$$

- **Multiplicaion**

$(m, \alpha, \beta)_{LR} \otimes (n, \gamma, \delta)_{LR} = (mn, m\gamma + n\alpha, m\delta + n\beta)_{LR}$ for $m \geq 0$, $n \geq 0$

$(m, \alpha, \beta)_{LR} \otimes (n, \gamma, \delta)_{LR} = (mn, m\alpha - m\delta, n\beta - m\gamma)_{RL}$ for $m<0$, $n \geq 0$

$(m, \alpha, \beta)_{LR} \otimes (n, \gamma, \delta)_{LR} = (mn, -n\beta - m\delta, -n\alpha - m\gamma)_{LR}$ for $m<0$, $n<0$

- **Scalar Multiplicaion**

$\lambda * (m, \alpha, \beta)_{LR} = (\lambda m, \lambda\alpha, \lambda\beta)_{LR}$, $\forall \lambda \geq 0$, $\lambda \in R$

$\lambda * (m, \alpha, \beta)_{LR} = (\lambda m, -\lambda\alpha, -\lambda\beta)_{RL}$, $\forall \lambda < 0$, $\lambda \in R$

■ Fuzzy Neuron

The fuzzy neuron is the basic element of Fuzzy BP network. Fig. below shows the architecture of the fuzzy neuron.

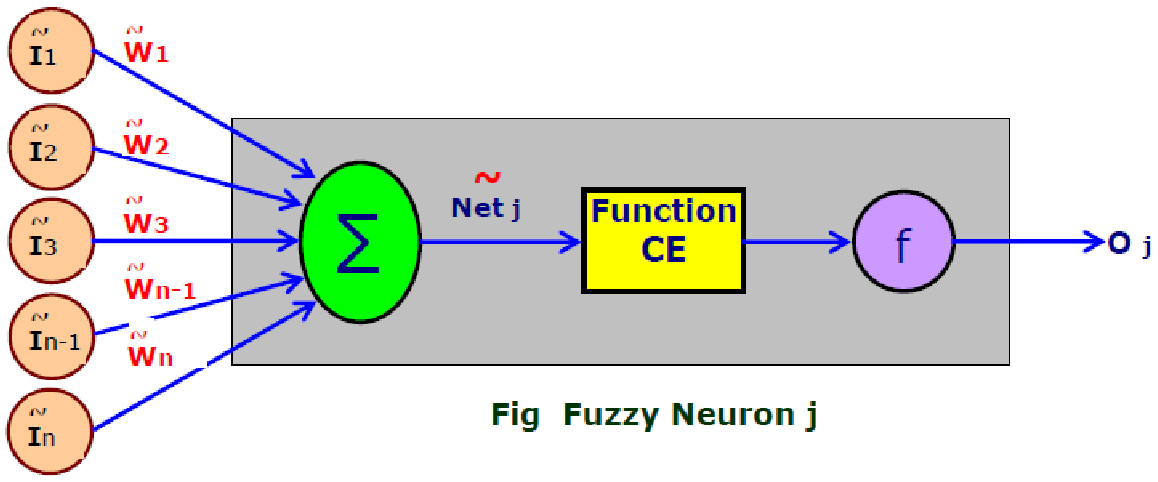


Fig Fuzzy Neuron j

The fuzzy neuron computes the crisp output given by

$$O = f\,(NET) = f\left(CE\left(\sum_{i=1}^{n} \tilde{W}_i \cdot \tilde{I}_i\right)\right)$$ where $\tilde{I}_0 = (1, 0, 0)$ is the bias.

Here, the fuzzy weighted summation is given by

$$\tilde{net} = \sum_{i=0}^{n} \tilde{W}_i \cdot \tilde{I}_i$$ is first computed and

$$NET = CE\,(\tilde{net})$$ is computed next

The function ***CE*** is the centroid of triangular fuzzy number, that has **m** as mean and α , **β** as left and right spreads explained before, can be treated as defuzzification operation, which maps fuzzy weighted summation to crisp value.

If $\tilde{net} = (\tilde{net}_m , \tilde{net}_\alpha , \tilde{net}_\beta)$ is the fuzzy weighted summation

Then function **CE** is given by

$$CE (\tilde{net}) = CE (\tilde{net}_m , \tilde{net}_\alpha , \tilde{net}_\beta) = \tilde{net}_m + 1/3 (\tilde{net}_\beta - \tilde{net}_\alpha) = NET$$

The function **f** is a sigmoidal function that performs nonlinear mapping between the input and output. The function **f** is obtained as :

f (NET) = 1 / (1 + exp (- NET)) = O is final crisp output value.

■ Architecture of Fuzzy BP

Fuzzy Back Propagation Network (BP) is a 3-layered feed forward architecture. The 3 layers are: input layer, hidden layer and output layer. Considering a configuration of **ℓ-input** neurons, **m-hidden** neurons and **n-output** neurons, the architecture of Fuzzy BP is shown below.

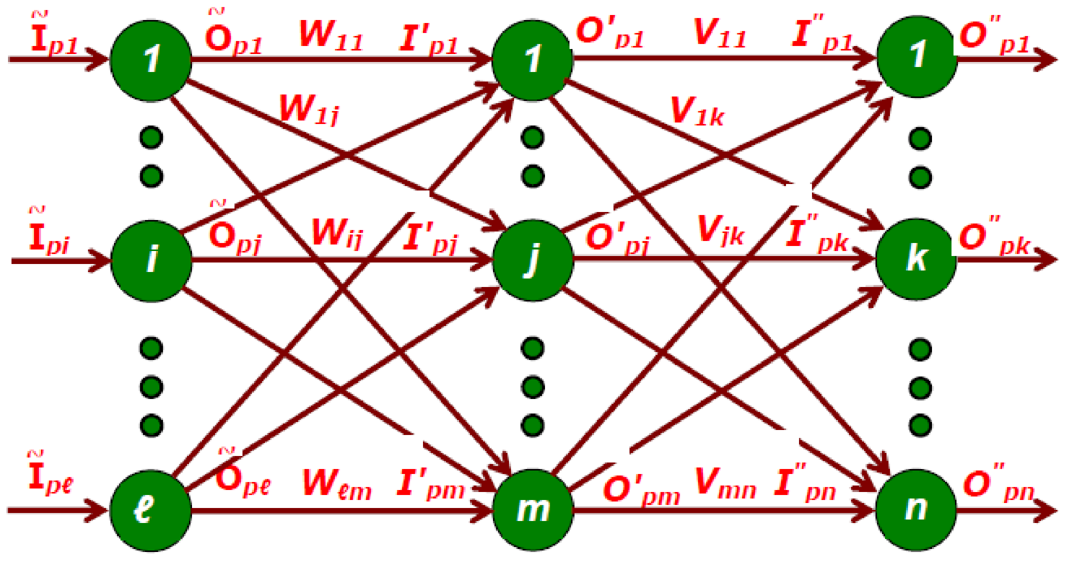


Fig. Three layer Fuzzy BP architecture.

- **Fuzzy Associative Memory**

A fuzzy logic system contains the sets used to categorize input data (i.e., fuzzification), the decision rules that are applied to each set, and then a way of generating an output from the rule results (i.e., defuzzification).

In the fuzzification stage, a data point is assigned a degree of membership (DOM) determined by a membership function. The membership function is often a triangular function centered at a given point. The Defuzzification is the name for a procedure to produce a real (non-fuzzy) output .

Associative Memory is a type of memory with a generalized addressing method. The address is not the same as the data location, as in traditional memory. An associative memory system stores mappings of specific input representations to specific output representations. Associative memory allows a fuzzy rule base to be stored. The inputs are the degrees of membership, and the outputs are the fuzzy system's output.

Fuzzy Associative Memory (FAM) consists of a single-layer feed-forward fuzzy neural network that stores fuzzy rules **"If x is X_k then y is Y_k"** by means of a fuzzy associative matrix.

FAM has many applications; one such application is modeling the operations of **washing machine**.

The problem indicates, that there are two inputs and one-output variables. The inference engineer is constructed based on fuzzy rule :

" If < input variable > AND < input variable >
THEN < output variable >"

According to the above fuzzy rule, the Fuzzy Associative Memory (FSM) of **X, Y,** and **T** variables are listed in the Table below.

Washing time (T)		**Weight (X)**		
		S	**M**	**L**
Stream (Y)	**S**	**M**	**L**	**L**
	M	**S**	**M**	**L**
	L	**S**	**S**	**L**

Table 1. Fuzzy associative memory (FSM) of Washing Machine

- **Operations :** To wash the clothes –
  Turn on the power,

  The machine automatically detects the weight of the clothes as **(X) = 3.2** K.g. ,

  – The machine adjusts the water stream **(Y)** to **32** liter/min.,

- **Fuzzy Representation :**

The fuzzy sets representation, while **X = 3.2 Kg** and **Y = 32 liter/min.**, according to the membership functions, are as follows:

The fuzzy set of $X_{3.2\text{ Kg}}$ = { 0.8/S, 0.2/M, 0/L }

The fuzzy set of $Y_{32\text{ liters/min.}}$ = { 0.4/S, 0.8/M, 0/L }

- **Defuzzification**

The real washing time is defuzzied by the **Center of gravity (COG)** defuzzification formula. The washing time is calculated as :

$$Z_{COG} = \sum_{j=1}^{n} \mu_c(Z_j)\, Z_j \;/\; \sum_{j=1}^{n} \mu_c(Z_j)$$ where

$j = 1, \ldots, n$, is the number of quantization levels of the output,

Z_j is the control output at the quantization level j ,

$\mu_c(Z_j)$ represents its membership value in the output fuzzy set.

Referring to Fig in the previous slide and the formula for COG, we get the fuzzy set of the washing time as ***w* = { 0.8/20, 0.4/35, 0.2/60 }** The calculated washing time using COG formula ***T* = 41.025 min.**

- **Simplified Fuzzy ARTMAP**

ART is a neural network topology whose dynamics are based on Adaptive Resonance Theory (ART). ART networks follow both supervised and unsupervised algorithms.

- The Unsupervised ARTs are similar to many iterative clustering algorithms where "nearest" and "closer" are modified slightly by introducing the concept of "**resonance**". *Resonance* is just a matter of being within a certain threshold of a second similarity measure.

- The Supervised ART algorithms that are named with the suffix "MAP", as ***ARTMAP***. Here the algorithms cluster both the inputs and targets and associate two sets of clusters.

ART1:

ART2 :

The basic ART system is an unsupervised learning model.

The ART systems have many variations : ART1, ART2, Fuzzy ART, ARTMAP.

The simplest variety of ART networks, accepting only binary inputs.

It extends network capabilities to support continuous inputs.

ARTMAP : Also known as Predictive ART. It combines two slightly modified ART-1 or ART-2 units into a supervised learning structure. Here, 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.

The Fuzzy ARTMAP model is fuzzy logic based computations incorporated in the ARTMAP model.

Fuzzy ARTMAP is neural network architecture for conducting supervised learning in a multidimensional setting. When Fuzzy ARTMAP is used on a learning problem, it is trained till it correctly classifies all training data. This feature causes Fuzzy ARTMAP to 'over-fit' some data sets, especially those in which the underlying pattern has to overlap. To avoid the problem of 'over-fitting' we must allow for error in the training process.

- **Supervised ARTMAP System**

ARTMAP is also known as predictive ART. The Fig. below shows a supervised ARTMAP system. Here, two ART modules are linked by an

inter-ART module called the Map Field. The Map Field forms predictive associations between categories of the ART modules and realizes a match tracking rule. If ARTa and ARTb are disconnected then each module would be of self-organize category, groupings their respective input sets.

Fig.Supervised ARTMAP system

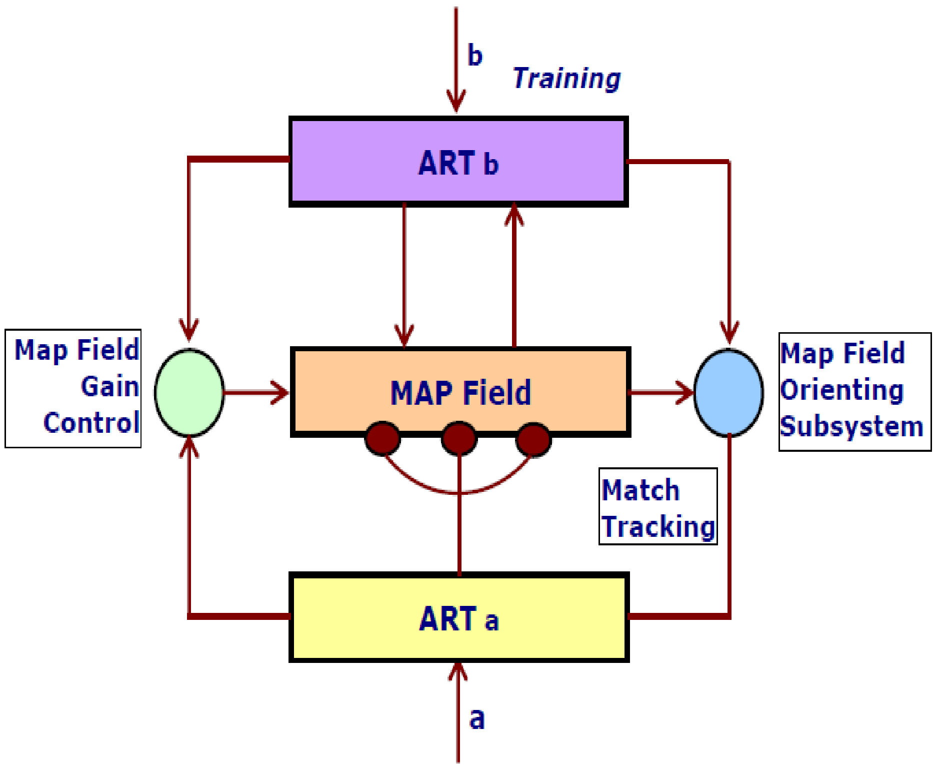


Fig. Supervised ARTMAP system

In supervised mode, the mappings are learned between input vectors ***a*** and ***b***. A familiar example of supervised neural networks are feed-forward networks with back-propagation of errors.

- **Comparing ARTMAP with Back-Propagation Networks**

ARTMAP networks are self-stabilizing, while in BP networks the new information gradually washes away old information. A consequence of this is that a BP network has separate training and performance phases while ARTMAP systems perform and learn at the same time

– ARTMAP networks are designed to work in real-time, while BP networks are typically designed to work off-line, at least during their training phase.

- ARTMAP systems can learn both in a fast as well as in a slow match configuration, while, the BP networks can only learn in slow mismatch configuration. This means that an ARTMAP system learns, or adapts its weights, only when the input matches an established category, while BP networks learn when the input does not match an established category.
- In BP networks there is always a danger of the system getting trapped in a local minimum while this is impossible for ART systems. However, the systems based on ART modules learning may depend upon the ordering of the input patterns.