

# OEC552-SOFT COMPUTING (SC)

## UNIT-I

### Introduction to Soft Computing

#### Text book

1. J.S.R.Jang, C.T. Sun and E.Mizutani, “**Neuro-Fuzzy and Soft Computing**”, PHI / Pearson Education 2004
2. S.N. Sivanandam & S.N. Deepa, “**Principles of Soft Computing, 2<sup>nd</sup> Edition**”  
by Copyright © 2011 Wiley India Pvt. Ltd. All rights reserved

## SOFT COMPUTING (SC)

- Solutions for a complex problems –obtained by incorporating certain processes resembling biological and nature inspired phenomena.
- used intelligently where solutions in polynomial time and remain intractable to conventional mathematical and analytical methods.

### Deals with

Imprecision, uncertainty, partial truth and approximation to achieve practicability, robustness and low solution cost.

**SC techniques** includes Expert systems, Artificial Neural Networks, Fuzzy Logic systems and evolutionary computations.

**Evolutionary computation techniques** include Evolutionary algorithms, metaheuristics and Swarm intelligent techniques.

**Swarm intelligent techniques** such as ant colony optimisation, Particle swarm optimisation, bees algorithms and cuckoo search.

# SOFT COMPUTING (SC)

## Soft Computing (SC):

The symbiotic use of many emerging problem-solving disciplines.

## According to Prof. Zadeh:

*"...in contrast to traditional hard computing, soft computing exploits the tolerance for imprecision, uncertainty, and partial truth to achieve tractability, robustness, low solution-cost, and better rapport with reality"*

## Soft Computing Main Components:

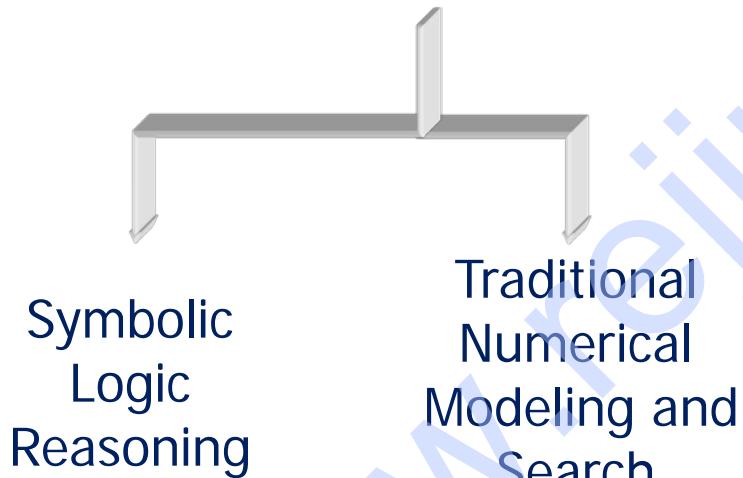
- Approximate Reasoning
- Search & Optimization
  - ✓ Neural Networks, Fuzzy Logic, Evolutionary Algorithms

# PROBLEM SOLVING TECHNIQUES

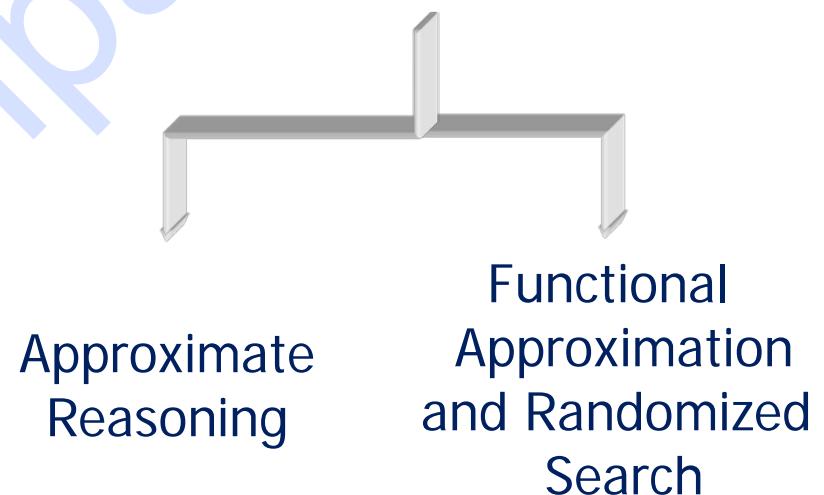
## HARD COMPUTING

## SOFT COMPUTING

### Precise Models



### Approximate Models



## Hard computing vs Soft computing

Hard Computing	Soft computing
Precisely stated analytical model required	Imprecision is tolerable
More Computation time required	As it involves intelligent computational steps, computational time required is less
It involves binary logic crisp systems and numerical analysis	It involves nature inspired systems such as neural networks, fuzzy logic systems and swarm intelligent system.
Precision is observed within the computation	Approximation is obtained in the computation
Imprecision and uncertainty are undesirable properties	Tolerance for imprecision and uncertainty is exploited to achieve tractability, lower cost, high Machine intelligence quotient and economy of communication.
It produces precise answer	It can produce approximate answers

## Hard computing vs Soft computing

Hard Computing	Soft computing
Programs are written which follow standard rules of programming	Programs are evolved which require new laws and theories to be created and justified while programming
The outcome is deterministic(i.e., Every trial run, the output is same)	The outcome is stochastic or random in nature and need not be deterministic
It requires exact input data	It can deal with ambiguous and noisy input data
It strictly follows sequential computations	It allows parallel computations
It produces precise answer	It can produce approximate answers

# OVERVIEW OF TECHNIQUES IN SOFT COMPUTING

- **Neural Networks**
- **Fuzzy Logic**
- **Genetic Algorithm**
- **Hybrid Systems**

## NEURAL NETWORKS

**DARPA Neural Network Study (1988, AFCEA International Press, p. 60):**

... a neural network is a system composed of many simple processing elements operating in parallel whose function is determined by network structure, connection strengths, and the processing performed at computing elements or nodes.

**According to Haykin (1994), p. 2:**

A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

- Knowledge is acquired by the network through a learning process.
- Interneuron connection strengths known as synaptic weights are used to store the knowledge

**According to Nigrin (1993), p. 11:**

A neural network is a circuit composed of a very large number of simple processing elements that are neurally based. Each element operates only on local information.

Furthermore each element operates asynchronously; thus there is no overall system clock.

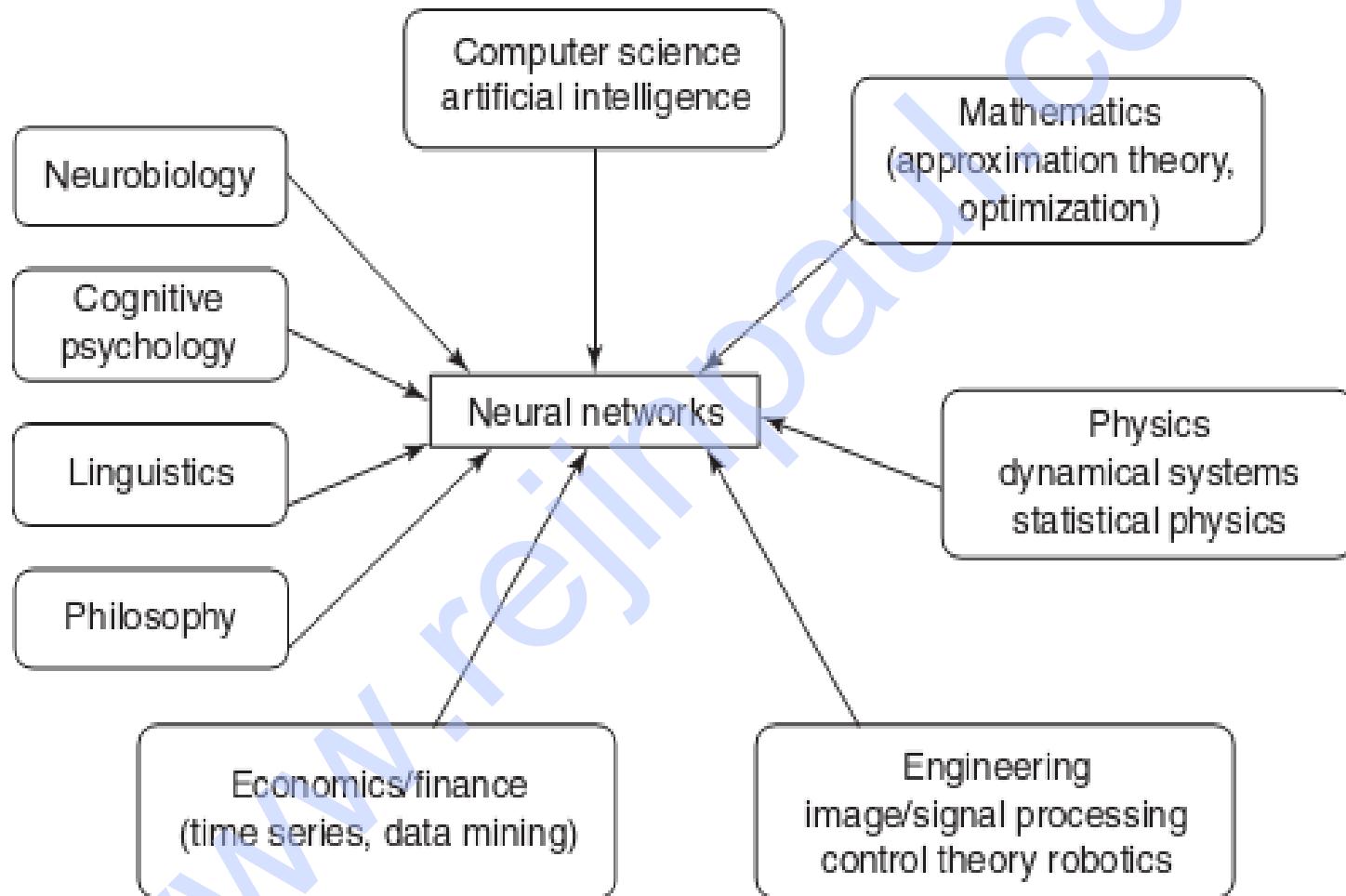
**According to Zurada (1992):**

Artificial neural systems, or neural networks, are physical cellular systems which can acquire, store and utilize experiential knowledge.

# Advantages of Neural Networks

- **Adaptive learning-** -based on data given for training / initial experience
- **Self organization** – create own representation of information it receives during training
- **Real time operation** - computation carried in parallel. Special H/W designed and manufactured
- **Fault tolerance via redundant information coding** - partial destruction leads to corresponding degradation of performance. Some network capabilities may be retained even after major network damage.

# MULTIDISCIPLINARY VIEW OF NEURAL NETWORKS



# Conventional Computing vs Neuro Computing

www.rejiinpaul.com

Conventional Computing	Neuro Computing
Computational process is sequential and deterministic	Computational process is not sequential and necessarily deterministic
A single processing unit is present-complex central processor	Many simple processing unit-present. They only takes the weighted sum of their inputs from other processors.
Respond to any programmed instruction	Do not respond to programmed instruction. But respond in parallel such as simulated or actual responses for the pattern of inputs present to it.
Separate addresses for storing data	No separate memory addresses for storing data; however information is contained in the overall activation state of the network
Knowledge is centrally located : if certain parts of data is lost, retrieval is not possible	Knowledge is distributed, data retrieval may be possible if a certain part of data is lost.(similar to the memory retrieval in human brain)
Not suited in situations where there are no clear cut algorithmic solutions	Well suited to situations where algorithmic solutions are not possible.
Cannot handle noisy imprecise data	Can manage noisy imprecise data.

# FUZZY LOGIC

- **Origins:** Multivalued Logic for treatment of imprecision and vagueness
  - **1930s:** **Post, Kleene, and Lukasiewicz** attempted to represent undetermined, unknown, and other possible intermediate truth-values.
  - **1937:** **Max Black** suggested the use of a consistency profile to represent vague (ambiguous) concepts.
  - **1965:** **Zadeh** proposed a complete theory of fuzzy sets (and its isomorphic fuzzy logic), to represent and manipulate ill-defined concepts.

## FUZZY LOGIC – LINGUISTIC VARIABLES

- Fuzzy logic gives us a language (with syntax and local semantics) in which we can translate our qualitative domain knowledge.
- Linguistic variables to model dynamic systems
- These variables take linguistic values that are characterized by:
  - a label - a sentence generated from the syntax
  - a meaning - a membership function determined by a local semantic procedure

## FUZZY LOGIC – REASONING METHODS

- The meaning of a linguistic variable may be interpreted as an elastic constraint on its value.
- These constraints are propagated by fuzzy inference operations, based on the generalized modus-ponens.
- An FL Controller (FLC) applies this reasoning system to a Knowledge Base (KB) containing the problem domain heuristics.
- The inference is the result of interpolating among the outputs of all relevant rules.
- The outcome is a membership distribution on the output space, which is defuzzified to produce a crisp output.

# GENETIC ALGORITHM

# EVOLUTIONARY PROCESS



**Definition of GA :** The genetic algorithm is a **probabilistic search algorithm** that iteratively transforms a set (called a population) of mathematical objects (typically fixed-length binary character strings), each with an associated fitness value, into a new population of offspring objects using the **Darwinian principle of natural selection** and using **operations** that are patterned after naturally occurring genetic operations, such as crossover (sexual recombination) and mutation.

# STEPS INVOLVED IN GENETIC ALGORITHM

The genetic algorithms follow the evolution process in the nature to find the better solutions of some complicated problems. Foundations of genetic algorithms are given in Holland (1975) and Goldberg (1989) books.

Genetic algorithms consist the following steps:

- Initialization
- Selection
- Reproduction with crossover and mutation

Selection and reproduction are repeated for each generation until a solution is reached.

During this procedure a certain strings of symbols, known as chromosomes, evaluate toward better solution.

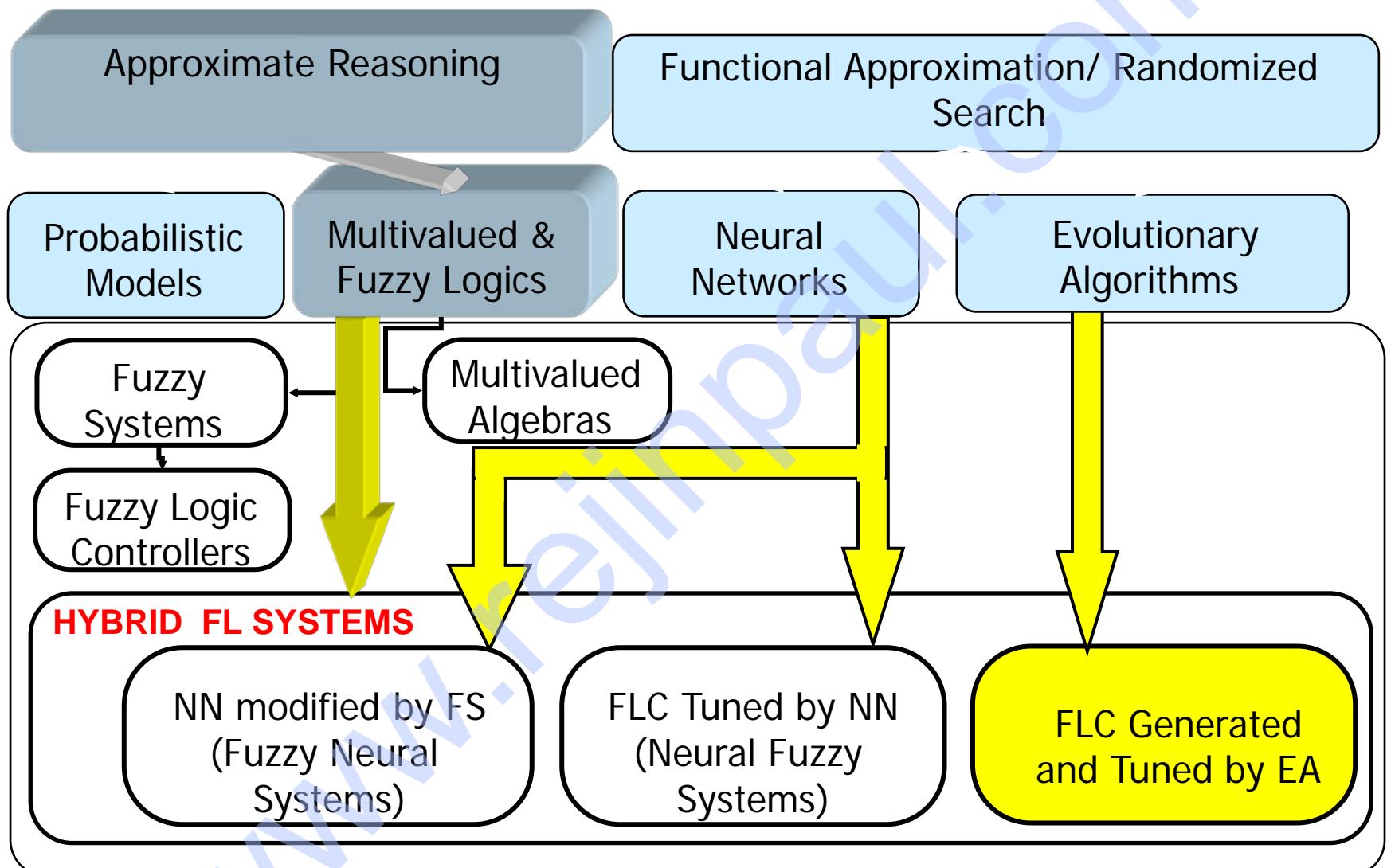
## HYBRID SYSTEMS

Hybrid systems enables one to combine various soft computing paradigms and result in a best solution.

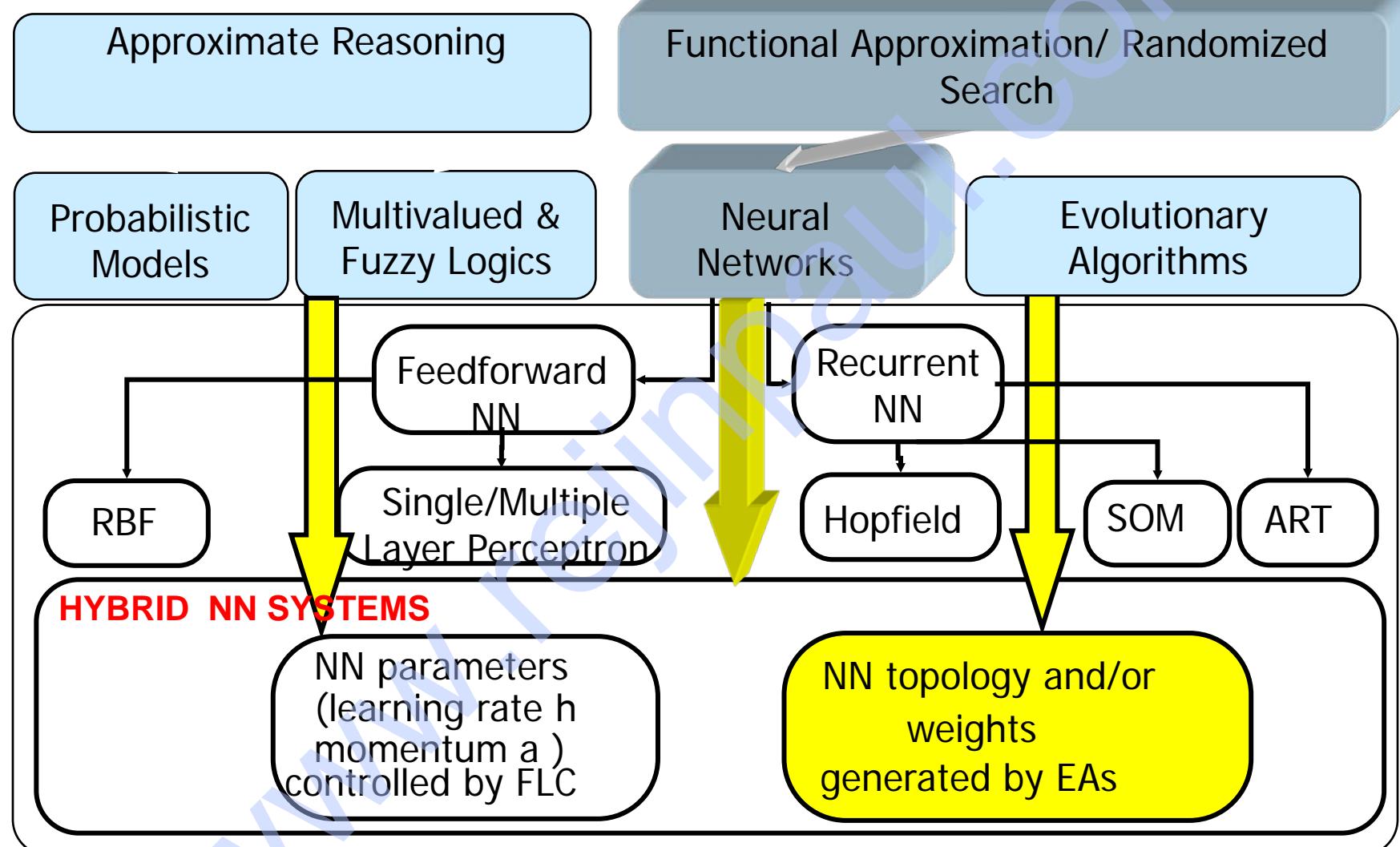
The major **three hybrid systems** are as follows:

- Hybrid Fuzzy Logic (FL) Systems
- Hybrid Neural Network (NN) Systems
- Hybrid Evolutionary Algorithm (EA) Systems

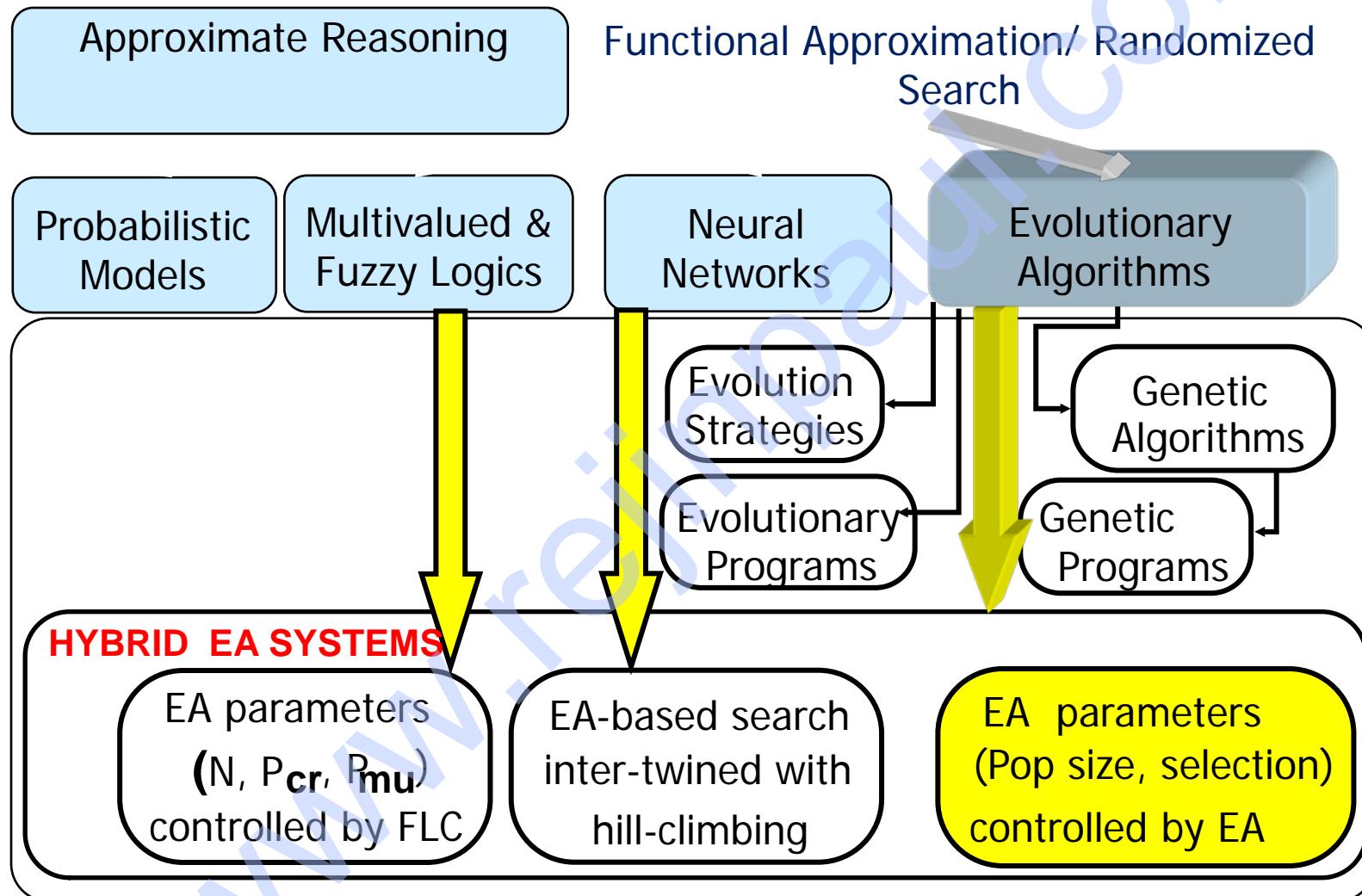
# SOFT COMPUTING: HYBRID FL SYSTEMS



# SOFT COMPUTING: HYBRID NN SYSTEMS



# SOFT COMPUTING: HYBRID EA SYSTEMS



## PROPERTIES OF SOFT COMPUTING

- Learning from experimental data
- Soft computing techniques derive their power of generalization from approximating or interpolating to produce outputs from previously unseen inputs by using outputs from previous learned inputs
- Generalization is usually done in a high dimensional space.

## ADVANTAGES OF SOFT COMPUTING

- Models based on human reasoning.
- Models can be
  - linguistic,
  - simple (no number crunching),
  - comprehensible (no black boxes),
  - fast when computing,
  - good in practice.

# APPLICATIONS OF SOFT COMPUTING

- Handwriting Recognition
- Image Processing and Data Compression
- Automotive Systems and Manufacturing
- Soft Computing to Architecture
- Decision-support Systems
- Soft Computing to Power Systems
- Neuro Fuzzy systems
- Fuzzy Logic Control
- Machine Learning Applications
- Speech and Vision Recognition Systems
- Process Control and So on

# SOFT COMPUTING APPLICATIONS: CONTROL



- Heavy industry (Matsushita, Siemens, Stora-Enso).
- Home appliances (Canon, Sony, Goldstar, Siemens).
- Automobiles (Nissan, Mitsubishi, Daimler-Chrysler, BMW, Volkswagen).
- Spacecrafts (NASA).

# SOFT COMPUTING APPLICATIONS: BUSINESS

- supplier evaluation for sample testing,
- customer targeting,
- sequencing,
- scheduling,
- optimizing R&D, projects,
- knowledge-based prognosis,
- fuzzy data analysis
- hospital stay prediction,
- TV commercial slot evaluation,
- address matching,
- fuzzy cluster analysis,
- sales prognosis for mail order house,
- multi-criteria optimization, etc.

## **SOFT COMPUTING APPLICATIONS: FINANCE**

- fuzzy scoring for mortgage applicants,
- creditworthiness assessment,
- fuzzy-enhanced score card for lease risk assessment,
- risk profile analysis,
- insurance fraud detection,
- cash supply optimization,
- foreign exchange trading,
- insider trading,
- trading surveillance,
- investor classification, etc.

## **SOFT COMPUTING APPLICATIONS: OTHERS**

Statistics, Social sciences, Behavioral sciences, Robotics, Biology, Medicine

## DEFINITION OF NEURAL NETWORKS

**According to the DARPA Neural Network Study (1988, AFCEA International Press, p. 60):**

- ... a neural network is a system composed of many simple processing elements operating in parallel whose function is determined by network structure, connection strengths, and the processing performed at computing elements or nodes.

**According to Haykin (1994), p. 2:**

A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

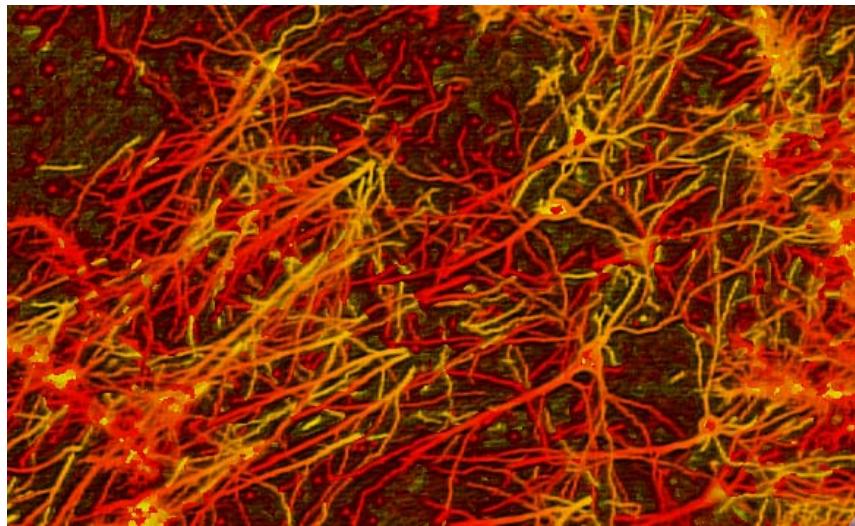
- Knowledge is acquired by the network through a learning process.
- Interneuron connection strengths known as synaptic weights are used to store the knowledge.

# BRAIN COMPUTATION

The **human brain** contains about 10 billion nerve cells, or neurons. On average, each neuron is connected to other neurons through approximately 10,000 synapses.

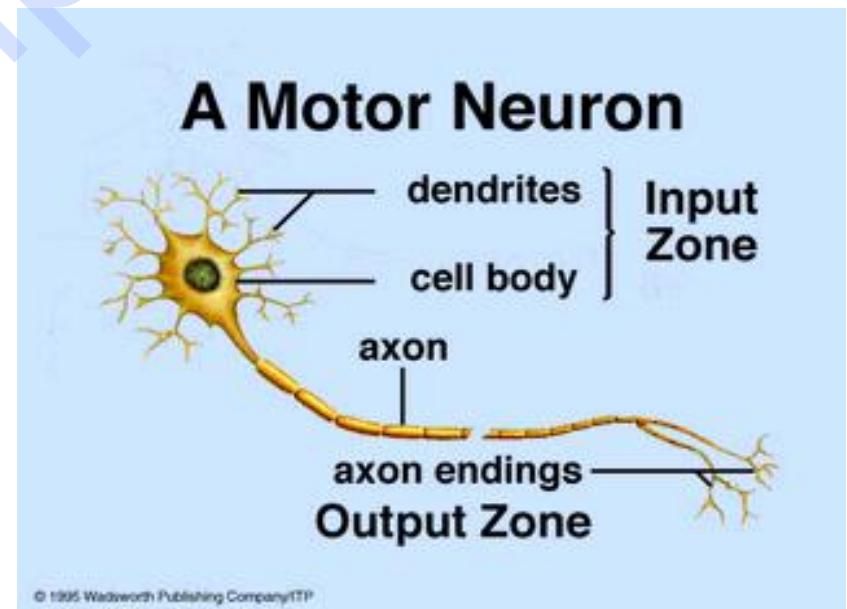


	processing elements	element size	energy use	processing speed	style of computation	fault tolerant	learns	intelligent, conscious
	$10^{14}$ synapses	$10^{-6}$ m	30 W	100 Hz	parallel, distributed	yes	yes	usually
	$10^8$ transistors	$10^{-6}$ m	30 W (CPU)	$10^9$ Hz	serial, centralized	no	a little	not (yet)



## BIOLOGICAL (MOTOR) NEURON

## INTERCONNECTIONS IN BRAIN



## ARTIFICIAL NEURAL NET

- Information-processing system.
- Neurons process the information.
- The signals are transmitted by means of connection links.
- The links possess an associated weight.
- The output signal is obtained by applying activations to the net input.

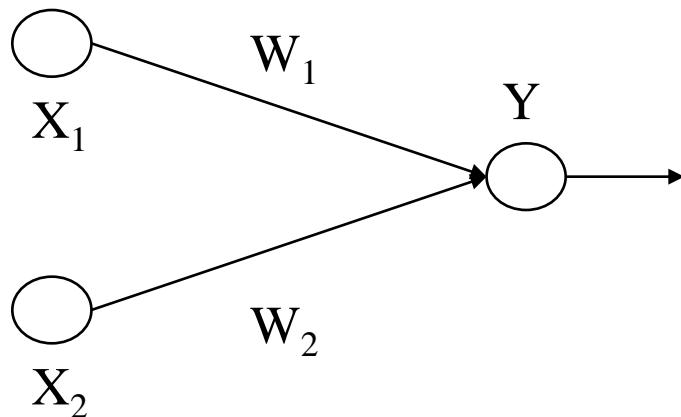
## MOTIVATION FOR NEURAL NET

- Scientists are challenged to use machines more effectively for tasks currently solved by humans.
- Symbolic rules don't reflect processes actually used by humans.
- Traditional computing excels in many areas, but not in others.

## The major areas being:

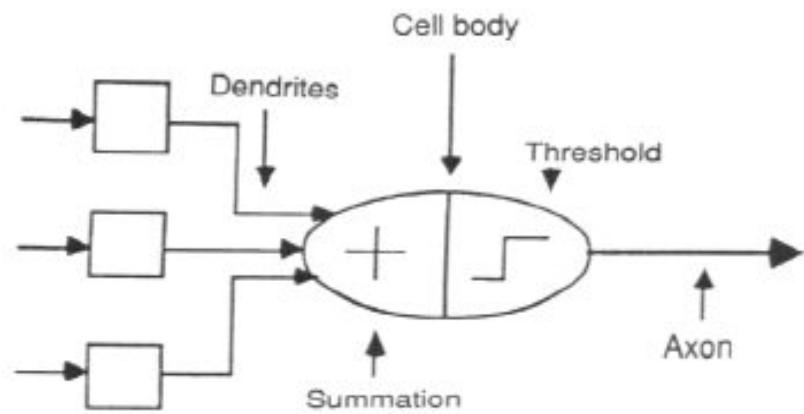
- Massive parallelism
- Distributed representation and computation
- Learning ability
- Generalization ability
- Adaptivity
- Inherent contextual information processing
- Fault tolerance
- Low energy consumption.

# ARTIFICIAL NEURAL NET



Simple artificial neural net with two input neurons ( $X_1, X_2$ ) and one output neuron ( $Y$ ). The interconnected weights are given by  $W_1$  and  $W_2$ .

## ASSOCIATION OF BIOLOGICAL NET WITH ARTIFICIAL NET



# PROCESSING OF AN ARTIFICIAL NET

The neuron is the basic information processing unit of a NN. It consists of:

1. A set of links, describing the neuron inputs, with weights  $W_1, W_2, \dots, W_m$ .
2. An adder function (linear combiner) for computing the weighted sum of the inputs (real numbers):
3. Activation function for limiting the amplitude of the neuron output.

$$\mathbf{u} = \sum_{j=1}^m \mathbf{W}_j \mathbf{X}_j$$

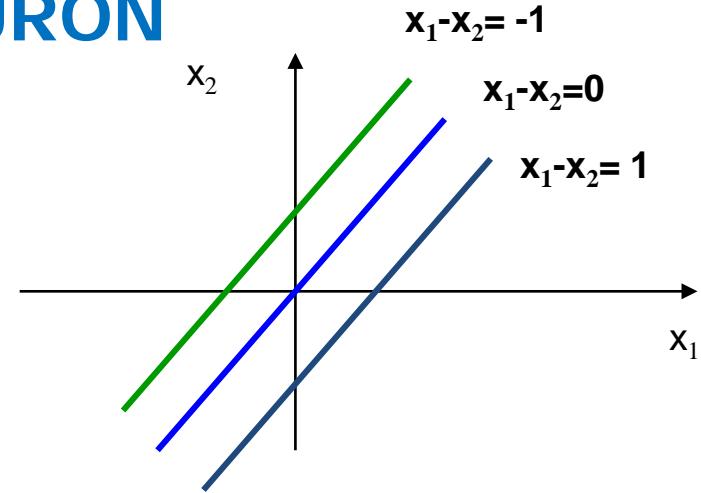
$$y = \varphi(u + b)$$

## BIAS OF AN ARTIFICIAL NEURON

The bias value is added to the weighted sum  $\sum w_i x_i$  so that we can transform it from the origin.

$$Y_{in} = \sum w_i x_i + b,$$

where  $b$  is the bias



# MULTI LAYER ARTIFICIAL NEURAL NET

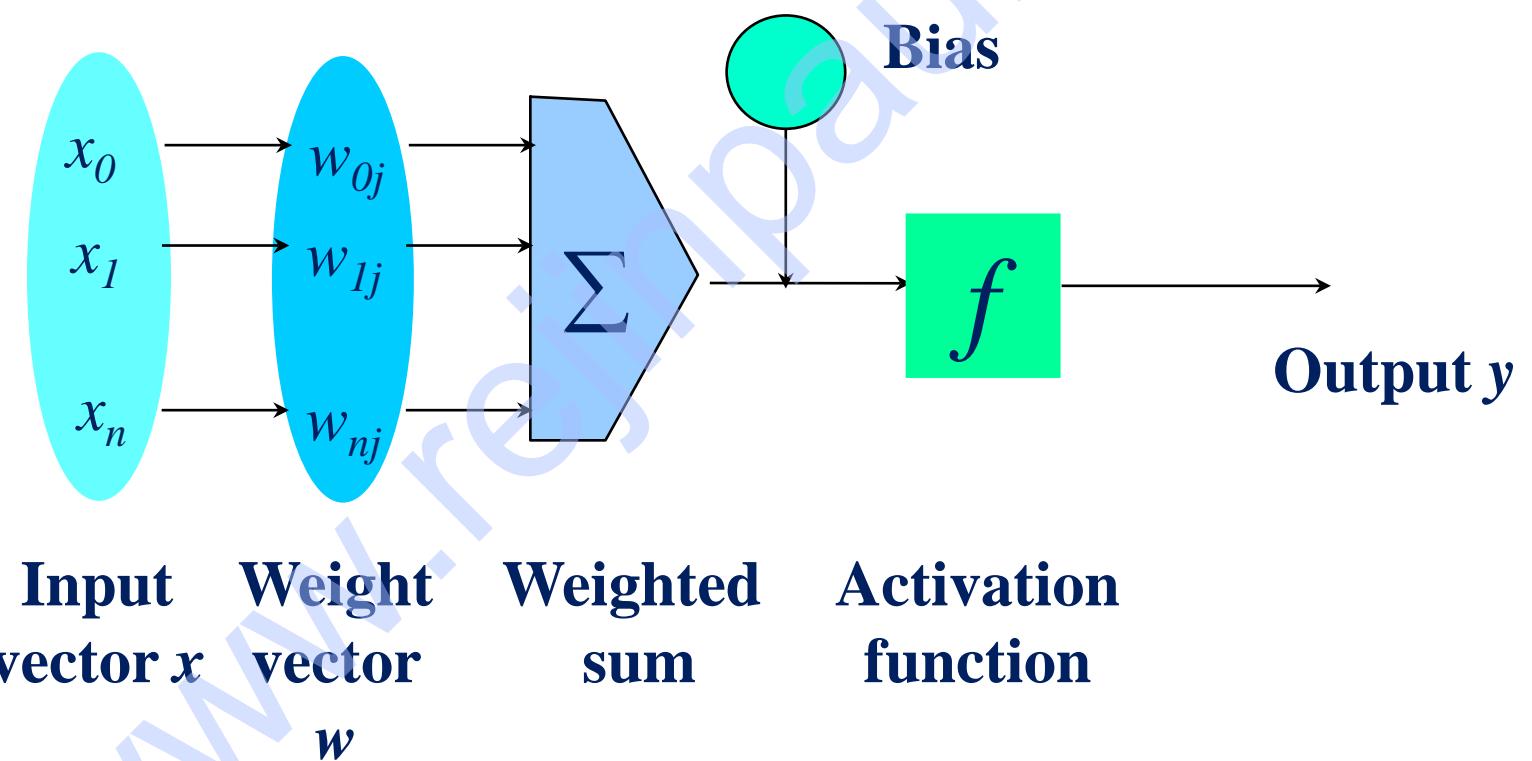
**INPUT:** records without class attribute with normalized attributes values.

**INPUT VECTOR:**  $X = \{x_1, x_2, \dots, x_n\}$  where n is the number of (non-class) attributes.

**INPUT LAYER:** there are as many nodes as non-class attributes, i.e. as the length of the input vector.

**HIDDEN LAYER:** the number of nodes in the hidden layer and the number of hidden layers depends on implementation.

## OPERATION OF A NEURAL NET



# WEIGHT AND BIAS UPDATION

## Per Sample Updating

- updating weights and biases after the presentation of each sample.

## Per Training Set Updating (Epoch or Iteration)

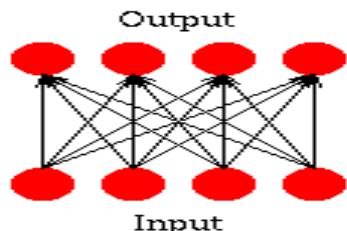
- weight and bias increments could be accumulated in variables and the weights and biases updated after all the samples of the training set have been presented.

## STOPPING CONDITION

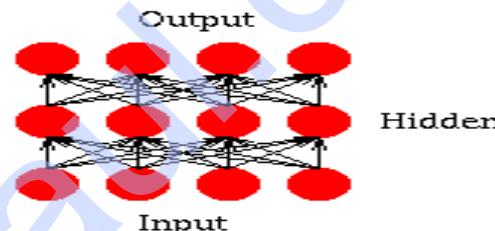
- All change in weights ( $\Delta w_{ij}$ ) in the previous epoch are below some threshold, or
- The percentage of samples misclassified in the previous epoch is below some threshold, or
- A pre-specified number of epochs has expired.
- In practice, several hundreds of thousands of epochs may be required before the weights will converge.

# BUILDING BLOCKS OF ARTIFICIAL NEURAL NET

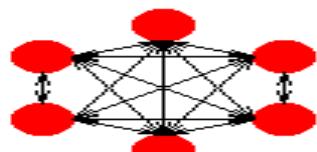
- Network Architecture (Connection between Neurons)
- Setting the Weights (Training)
- Activation Function



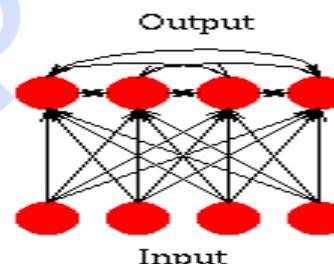
Single Layer Feedforward



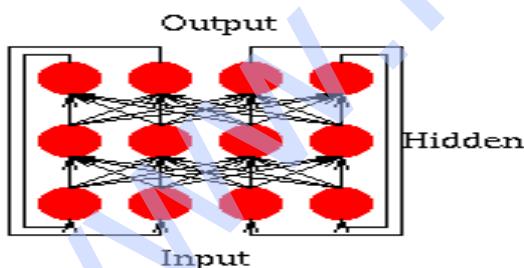
Multi Layer Feedforward



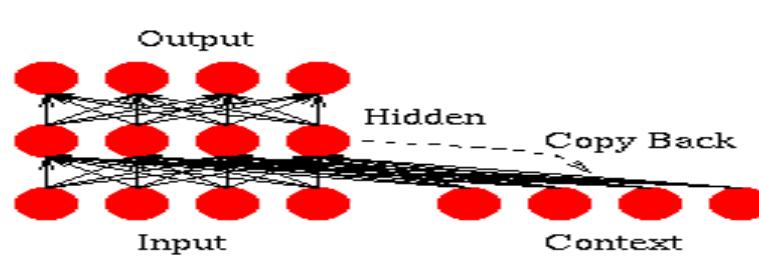
Fully Recurrent Network



Competitive Network



Jordan Network



Simple Recurrent Network

## LAYER PROPERTIES

- **Input Layer:** Each input unit may be designated by an attribute value possessed by the instance.
- **Hidden Layer:** Not directly observable, provides nonlinearities for the network.
- **Output Layer:** Encodes possible values.

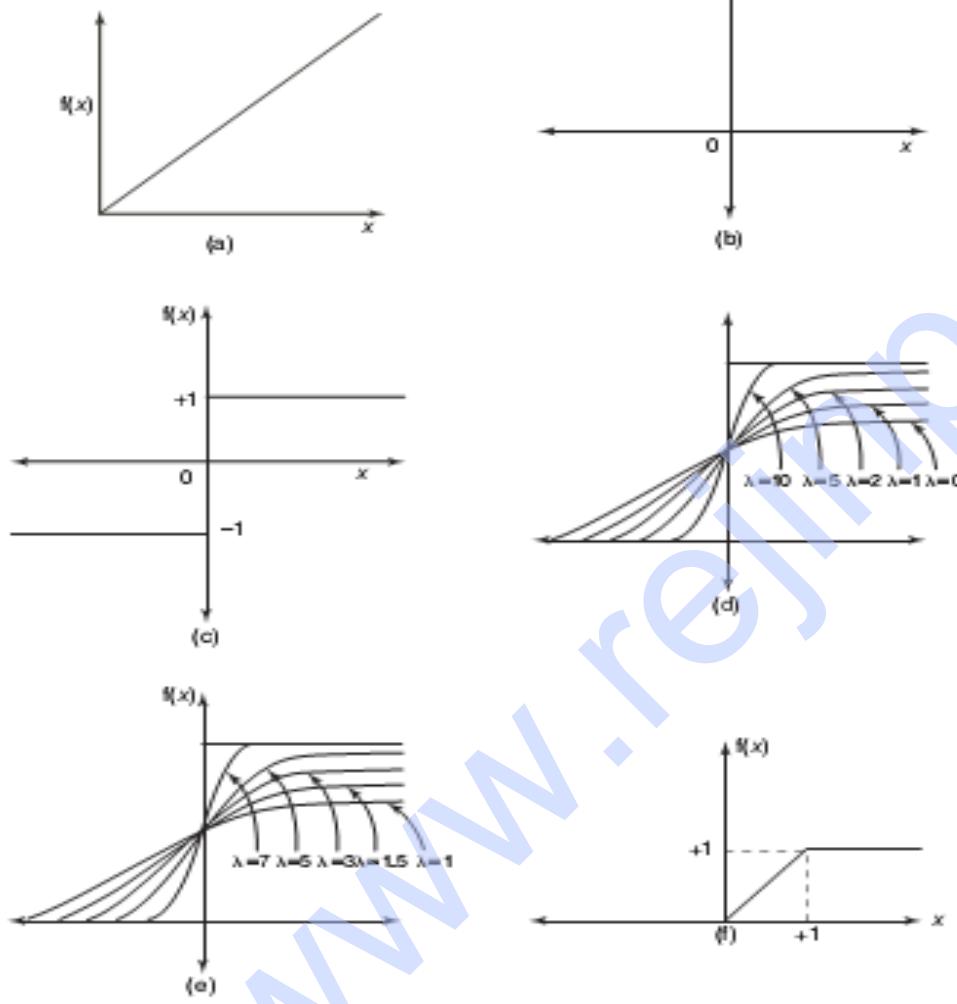
## TRAINING PROCESS

- **Supervised Training** - Providing the network with a series of sample inputs and comparing the output with the expected responses.
- **Unsupervised Training** - Most similar input vector is assigned to the same output unit.
- **Reinforcement Training** - Right answer is not provided but indication of whether 'right' or 'wrong' is provided.

# ACTIVATION FUNCTION

- **ACTIVATION LEVEL – DISCRETE OR CONTINUOUS**
- **HARD LIMIT FUNCTION (DISCRETE)**
  - Binary Activation function
  - Bipolar activation function
  - Identity function
- **SIGMOIDAL ACTIVATION FUNCTION (CONTINUOUS)**
  - Binary Sigmoidal activation function
  - Bipolar Sigmoidal activation function

# ACTIVATION FUNCTION



**Activation functions:**

- (A) Identity
- (B) Binary step
- (C) Bipolar step
- (D) Binary sigmoidal
- (E) Bipolar sigmoidal
- (F) Ramp

# CONSTRUCTING ANN

- Determine the network properties:
  - Network topology
  - Types of connectivity
  - Order of connections
  - Weight range
- Determine the node properties:
  - Activation range
- Determine the system dynamics
  - Weight initialization scheme
  - Activation – calculating formula
  - Learning rule

## PROBLEM SOLVING

- Select a suitable NN model based on the nature of the problem.
- Construct a NN according to the characteristics of the application domain.
- Train the neural network with the learning procedure of the selected model.
- Use the trained network for making inference or solving problems.

## NEURAL NETWORKS

- **Neural Network** learns by adjusting the weights so as to be able to correctly classify the training data and hence, after testing phase, to classify unknown data.
- **Neural Network** needs long time for training.
- **Neural Network** has a high tolerance to noisy and incomplete data.

## SALIENT FEATURES OF ANN

- Adaptive learning
- Self-organization
- Real-time operation
- Fault tolerance via redundant information coding
- Massive parallelism
- Learning and generalizing ability
- Distributed representation

## FEW APPLICATIONS OF NEURAL NETWORKS

- Aerospace
- Automotive
- Banking
- Credit Card Activity Checking
- Defense
- Electronics
- Entertainment
- Financial
- Industrial
- Insurance
- Insurance
- Manufacturing
- Medical
- Oil and Gas
- Robotics
- Speech
- Securities
- Telecommunications
- Transportation

# INTRODUCTION TO FUZZY LOGIC, CLASSICAL SETS AND FUZZY SETS

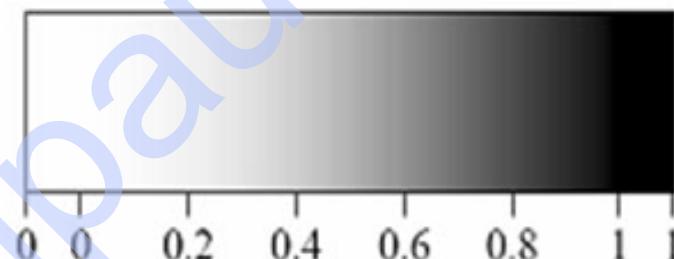
## FUZZY LOGIC

- Fuzzy logic is **the logic** underlying **approximate**, rather than exact, **modes of reasoning**.
- It is an extension of multivalued logic: **Everything**, including truth, **is a matter of degree**.
- It contains as special cases **not only** the classical two-value logic and multivalue logic systems, **but also** probabilistic logic.
- A proposition  $p$  has a **truth value**
  - 0 or 1 in two-value system,
  - element of a set  $T$  in multivalue system,
  - **Range over the fuzzy subsets of  $T$**  in fuzzy logic.

- Boolean logic uses sharp distinctions.
- Fuzzy logic reflects how people think.
- Fuzzy logic is a set of mathematical principles for



(a) Boolean Logic.



(b) Multi-valued Logic.

- Fuzzy logic is a set of mathematical principles for knowledge representation and reasoning based on degrees of membership.

# TYPES AND MODELING OF UNCERTAINTY

## Stochastic Uncertainty:

- ❖ The probability of hitting the target is 0.8

## Lexical Uncertainty:

- ❖ "Tall Men", "Hot Days", or "Stable Currencies"
- ❖ We will probably have a successful business year.
- ❖ The experience of expert A shows that B is Likely to Occur. However, expert C is convinced This Is Not True.

**Example:** One finds in desert two bottles of fluids with the following labels:

- ✓ bottle 1: there is a **probability of 5% that this bottle is poisoned.**
- ✓ bottle 2: this bottle contains a liquid which belongs to the set of drinkable water with membership function value of 0.95.

## FUZZY vs PROBABILITY

- • Let  $L = \text{set of all liquids}$
- –  $\mathbb{F}$  be the subset  $= \{\text{all drinkable liquids}\}$
- Suppose you had been in desert (you must drink!) and you come up with two bottles marked C and A.
- **Bottle C is labeled  $\mu_{\mathbb{F}}(C)=0.95$  and bottle A is labeled  $\Pr[A \in \mathbb{F}]=0.95$**
- C could contain swamp water, but would not contain any poison. Membership of 0.95 means that the contents of C are fairly similar to perfectly drinkable water.
- The probability that A is drinkable is 0.95, means that over a long run of experiments, the contents of A are expected to be drinkable in about 95% of the trials. In other cases it may contain poison.

# NEED OF FUZZY LOGIC

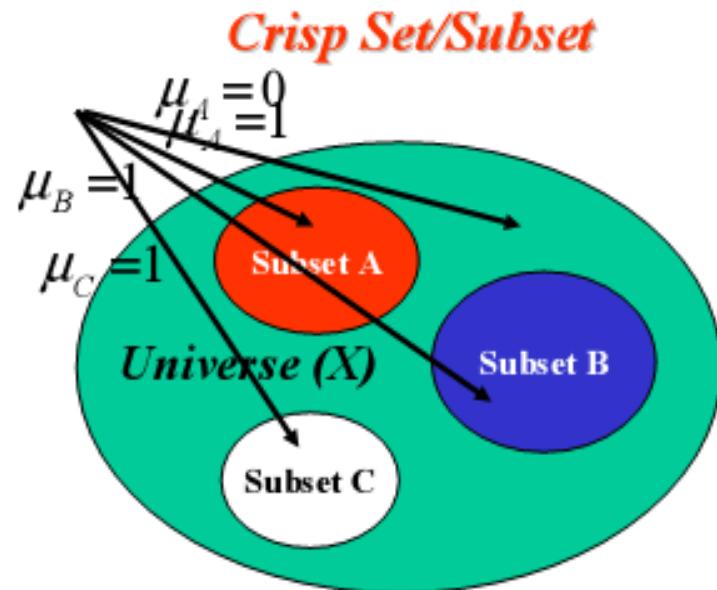
- Based on intuition and judgment.
- No need for a mathematical model.
- Provides a smooth transition between members and nonmembers.
- Relatively simple, fast and adaptive.
- Less sensitive to system fluctuations.
- Can implement design objectives, difficult to express mathematically, in linguistic or descriptive rules.

## CLASSICAL SETS (CRISP SETS)

Conventional or crisp sets are binary. An element either belongs to the set or does not.

{True, False}

{1, 0}



# OPERATIONS ON CRISP SETS

- UNION:

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

- INTERSECTION:

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

- COMPLEMENT:

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

- DIFFERENCE:

$$\begin{aligned} A \setminus B \text{ or } (A - B) &= \{x | x \in A \text{ and } x \notin B\} \\ &= A - (A \cap B) \end{aligned}$$

# PROPERTIES OF CRISP SETS

The various properties of crisp sets are as follows:

## 1. Commutativity

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

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

## 2. Associativity

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

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

## 3. Distributivity

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

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

## 4. Idempotency

$$A \cup A = A$$

$$A \cap A = A$$

## 5. Transitivity

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

## PROPERTIES OF CRISP SETS

6. Identity

$$\begin{aligned} A \cup \phi &= A, & A \cap \phi &= \phi \\ A \cap X &= A, & A \cup X &= X \end{aligned}$$

7. Involution (double negation)

$$\bar{\bar{A}} = A$$

8. Law of excluded middle

$$A \cup \bar{A} = X$$

9. Law of contradiction

$$A \cap \bar{A} = \phi$$

10. DeMorgan's law

$$\begin{aligned} \overline{A \cap B} &= \overline{A} \cup \overline{B} \\ \overline{A \cup B} &= \overline{A} \cap \overline{B} \end{aligned}$$

Rules of thumb frequently stated in “fuzzy” linguistic terms.

John is *tall*.

If someone is *tall and well-built*  
**then** his basketball skill is good.

### Fuzzy Sets

$0 \leq \mu_S(x) \leq 1$  -----  $\mu_S(x)$  (or  $\mu(S, x)$ ) is the **degree of membership** of  $x$  in set  $S$

$\mu_S(x) = 0$        $x$  is not at all in  $S$

$\mu_S(x) = 1$        $x$  is fully in  $S$ .

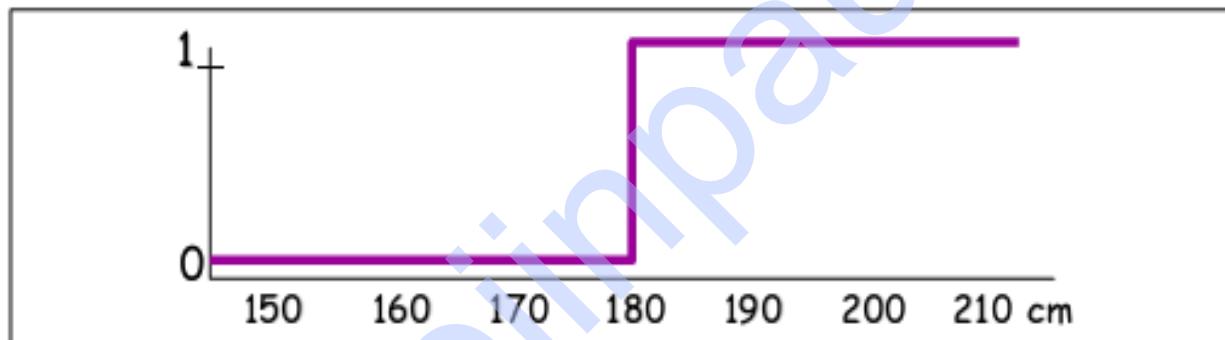
If  $\mu_S(x) = 0$  or  $1$ , then the set  $S$  is **crisp**.



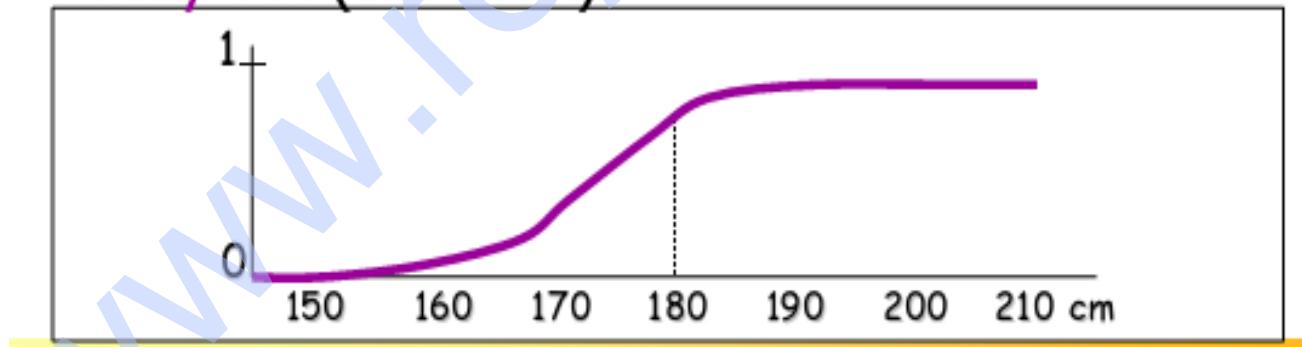
## Fuzzy set

■ Is a function  $f: \text{domain} \rightarrow [0,1]$

Crisp set (tall men):

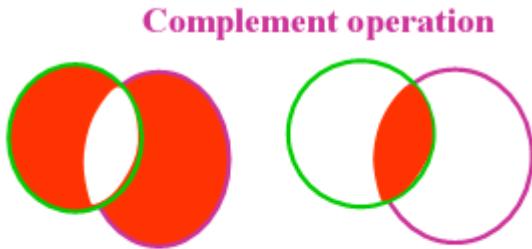


Fuzzy set (tall men):



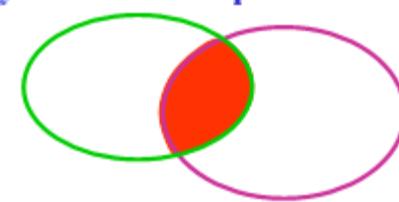
# OPERATIONS ON FUZZY SETS

- Union:  $\mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x))$
- Intersection:  $\mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x))$
- Complement:  $\mu_{\neg A}(x) = 1 - \mu_A(x)$



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

Fuzzy intersection operation or fuzzy AND



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

# PROPERTIES OF FUZZY SETS

The same as for crisp sets

- Commutativity
- Associativity
- Distributivity
- Idempotency
- Identity
- De Morgan's Laws
- ...

1. Commutativity

$$\begin{aligned}\tilde{A} \cup \tilde{B} &= \tilde{B} \cup \tilde{A} \\ \tilde{A} \cap \tilde{B} &= \tilde{B} \cap \tilde{A}\end{aligned}$$

2. Associativity

$$\begin{aligned}\tilde{A} \cup (\tilde{B} \cup \tilde{C}) &= (\tilde{A} \cup \tilde{B}) \cup \tilde{C} \\ \tilde{A} \cap (\tilde{B} \cap \tilde{C}) &= (\tilde{A} \cap \tilde{B}) \cap \tilde{C}\end{aligned}$$

3. Distributivity

$$\begin{aligned}\tilde{A} \cup (\tilde{B} \cap \tilde{C}) &= (\tilde{A} \cup \tilde{B}) \cap (\tilde{A} \cup \tilde{C}) \\ \tilde{A} \cap (\tilde{B} \cup \tilde{C}) &= (\tilde{A} \cap \tilde{B}) \cup (\tilde{A} \cap \tilde{C})\end{aligned}$$

4. Idempotency

$$\begin{aligned}\tilde{A} \cup \tilde{A} &= \tilde{A} \\ \tilde{A} \cap \tilde{A} &= \tilde{A}\end{aligned}$$

5. Identity

$$\begin{aligned}\tilde{A} \cup \phi &= \tilde{A} \text{ and } \tilde{A} \cup U = U \text{ (universal set)} \\ \tilde{A} \cap \phi &= \phi \text{ and } \tilde{A} \cap U = \tilde{A}\end{aligned}$$

6. Involution (double negation)

$$\bar{\bar{A}} = A$$

7. Transitivity

If  $\tilde{A} \subseteq \tilde{B} \subseteq \tilde{C}$ , then  $\tilde{A} \subseteq \tilde{C}$

8. Demorgan's law

$$\begin{aligned}\overline{\tilde{A} \cup \tilde{B}} &= \overline{\tilde{A}} \cap \overline{\tilde{B}} \\ \overline{\tilde{A} \cap \tilde{B}} &= \overline{\tilde{A}} \cup \overline{\tilde{B}}\end{aligned}$$

# CLASSICAL RELATIONS AND FUZZY RELATIONS

## RELATIONS

- Relations represent mappings between sets and connectives in logic.
- A classical binary relation represents the presence or absence of a connection or interaction or association between the elements of two sets.
- Fuzzy binary relations are a generalization of crisp binary relations, and they allow various degrees of relationship (association) between elements.

## CRISP CARTESIAN PRODUCT

Lets consider properties of crisp relations first and then extend the mechanism to fuzzy sets.

**Definition of (crisp) Product set:** Let  $A$  and  $B$  be two non-empty sets, the product set or Cartesian product  $A \times B$  is defined as follows,

$$A \times B = \{(a, b) \mid a \in A, b \in B\}$$

(a set of ordered pairs a,b)

# CRISP RELATIONS

**Cartesian product of  $n$  sets**

$$A_1 \times A_2 \times \dots \times A_n = \prod_{i=1}^n A_i = \{(a_1, \dots, a_n) \mid a_1 \in A_1, \dots, a_n \in A_n\}$$

**Definition of Binary Relation**

If  $A$  and  $B$  are two sets and there is a specific property between elements  $x$  of  $A$  and  $y$  of  $B$ , this property can be described using the ordered pair  $(x, y)$ . A set of such  $(x, y)$  pairs,  $x \in A$  and  $y \in B$ , is called a relation  $R$ .

$$R = \{ (x, y) \mid x \in A, y \in B \}$$

**Definition of  $n$ -ary relation**

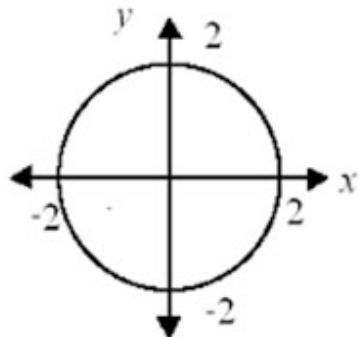
For sets  $A_1, A_2, A_3, \dots, A_n$ , the relation among elements  $x_1 \in A_1, x_2 \in A_2, x_3 \in A_3, \dots, x_n \in A_n$  can be described by  $n$ -tuple  $(x_1, x_2, \dots, x_n)$ . A collection of such  $n$ -tuples  $(x_1, x_2, x_3, \dots, x_n)$  is a relation  $R$  among  $A_1, A_2, A_3, \dots, A_n$ .

$$(x_1, x_2, x_3, \dots, x_n) \in R, \\ R \subseteq A_1 \times A_2 \times A_3 \times \dots \times A_n$$

# CRISP BINARY RELATIONS

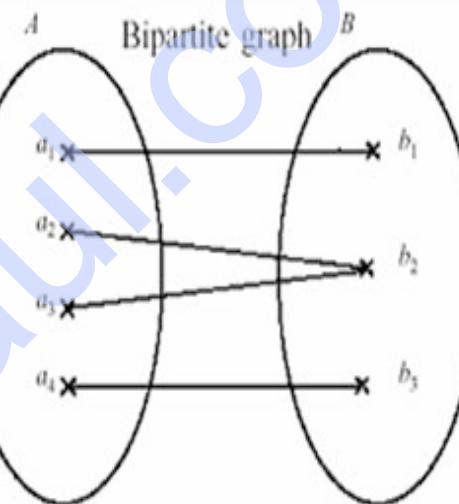
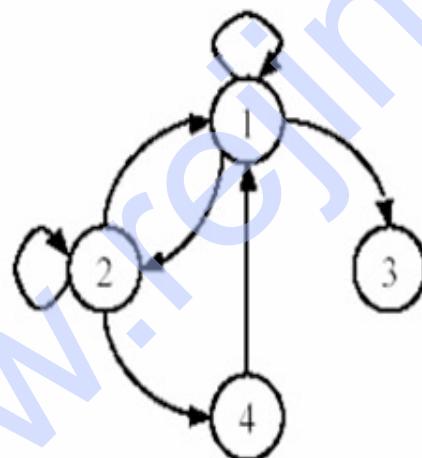
## Examples of binary relations

Coordinate diagram



Relation of  $x^2 + y^2 = 4$

Binary relation from  $A$  to  $B$



Relation matrix

$R$	$b_1$	$b_2$	$b_3$
$a_1$	1	0	0
$a_2$	0	1	0
$a_3$	0	1	0
$a_4$	0	0	1

# OPERATIONS ON CRISP RELATIONS

Function-theoretic operations for the two crisp relations ( $R, S$ ) are defined as follows:

1. Union

$$R \cup S \rightarrow \chi_{R \cup S}(x, y) : \chi_{R \cup S}(x, y) = \max [\chi_R(x, y), \chi_S(x, y)]$$

2. Intersection

$$R \cap S \rightarrow \chi_{R \cap S}(x, y) : \chi_{R \cap S}(x, y) = \min [\chi_R(x, y), \chi_S(x, y)]$$

3. Complement

$$\overline{R} \rightarrow \chi_{\overline{R}}(x, y) : \chi_{\overline{R}}(x, y) = 1 - \chi_R(x, y)$$

4. Containment

$$R \subset S \rightarrow \chi_R(x, y) : \chi_R(x, y) \leq \chi_S(x, y)$$

5. Identity

$$\phi \rightarrow \phi_R \text{ and } X \rightarrow E_R$$

# PROPERTIES OF CRISP RELATIONS

www.rejinpaul.com

The properties of crisp sets (given below) hold good for crisp relations as well.

- Commutativity,
- Associativity,
- Distributivity,
- Involution,
- Idempotency,
- DeMorgan's Law,
- Excluded Middle Laws.

## COMPOSITION ON CRISP RELATIONS

The composition operations are of two types:

1. Max-min composition
2. Max-product composition.

The max-min composition is defined by the function theoretic expression as

$$T = R \circ S$$
$$\chi_T(x, z) = \vee_{y \in Y} [\chi_R(x, y) \wedge \chi_S(y, z)]$$

The max-product composition is defined by the function theoretic expression as

$$T = R \circ S$$
$$\chi_T(x, z) = \vee_{y \in Y} [\chi_R(x, y) \cdot \chi_S(y, z)]$$

## FUZZY CARTESIAN PRODUCT

Let R be a fuzzy subset of M and S be a fuzzy subset of N. Then the Cartesian product  $R \times S$  is a fuzzy subset of  $N \times M$  such that

$$\forall \vec{x} \in M, \vec{y} \in N \quad \mu_{R \times S}(\vec{x}, \vec{y}) = \min(\mu_R(\vec{x}), \mu_S(\vec{y}))$$

### Example:

Let R be a fuzzy subset of  $\{a, b, c\}$  such that  $R = a/1 + b/0.8 + c/0.2$  and S be a fuzzy subset of  $\{1, 2, 3\}$  such that  $S = 1/1 + 3/0.8 + 2/0.5$ . Then  $R \times S$  is given by

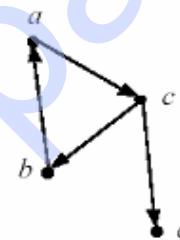
	1	2	3
a	1	0.5	0.9
b	0.8	0.5	0.8
c	0.2	0.2	0.2

A fuzzy relation  $R$  is a mapping from the Cartesian space  $X \times Y$  to the interval  $[0,1]$ , where the strength of the mapping is expressed by the membership function of the relation  $\mu_R(x,y)$

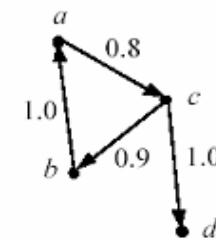
$$\mu_R : A \times B \rightarrow [0, 1]$$

$$R = \{((x, y), \mu_R(x, y)) \mid \mu_R(x, y) \geq 0, x \in A, y \in B\}$$

Crisp relation vs. Fuzzy relation



(a) Crisp relation



(b) Fuzzy relation

Corresponding fuzzy  
relation matrix

		a	b	c	d	
		a	0.0	0.0	0.8	0.0
		b	1.0	0.0	0.0	0.0
		c	0.0	0.9	0.0	1.0
		d	0.0	0.0	0.0	0.0

# OPERATIONS ON FUZZY RELATION

The basic operation on fuzzy sets also apply on fuzzy relations.

1. Union:

$$\mu_{\tilde{R} \cup \tilde{S}}(x, y) = \max [\mu_{\tilde{R}}(x, y), \mu_{\tilde{S}}(x, y)]$$

2. Intersection:

$$\mu_{\tilde{R} \cap \tilde{S}}(x, y) = \min [\mu_{\tilde{R}}(x, y), \mu_{\tilde{S}}(x, y)]$$

3. Complement:

$$\mu_{\tilde{R}}^c(x, y) = 1 - \mu_{\tilde{R}}(x, y)$$

4. Containment:

$$\tilde{R} \subset \tilde{S} \Rightarrow \mu_{\tilde{R}}(x, y) \leq \mu_{\tilde{S}}(x, y)$$

5. Inverse:

The inverse of a fuzzy relation  $R$  on  $X \times Y$  is denoted by  $R^{-1}$ . It is a relation on  $Y \times X$  defined by  $R^{-1}(y, x) = R(x, y)$  for all pairs  $(y, x) \in Y \times X$ .

6. Projection:

For a fuzzy relation  $R(X, Y)$ , let  $[R \downarrow Y]$  denote the projection of  $R$  onto  $Y$ . Then  $[R \downarrow Y]$  is a fuzzy relation in  $Y$  whose membership function is defined by

$$\mu_{[R \downarrow Y]}(x, y) = \max_x \mu_{\tilde{R}}(x, y)$$

The projection concept can be extended to an  $n$ -ary relation  $R(x_1, x_2, \dots, x_n)$ .

# PROPERTIES OF FUZZY RELATIONS

The properties of fuzzy sets (given below) hold good for fuzzy relations as well.

- Commutativity,
- Associativity,
- Distributivity,
- Involution,
- Idempotency,
- DeMorgan's Law,
- Excluded Middle Laws.

## COMPOSITION OF FUZZY RELATIONS

Two fuzzy relations  $R$  and  $S$  are defined on sets  $A$ ,  $B$  and  $C$ . That is,  $R \subseteq A \times B$ ,  $S \subseteq B \times C$ . The composition  $S \bullet R = SR$  of two relations  $R$  and  $S$  is expressed by the relation from  $A$  to  $C$ :

For  $(x, y) \in A \times B$ ,  $(y, z) \in B \times C$ ,

$$\begin{aligned}\mu_{S \bullet R}(x, z) &= \max_y [\min(\mu_R(x, y), \mu_S(y, z))] \\ &= \vee_y [\mu_R(x, y) \wedge \mu_S(y, z)]\end{aligned}$$

$M_{S \bullet R} = M_R \bullet M_S$  (matrix notation)  
(max-min composition)

Example:

$$X = \{x_1, x_2\}, Y = \{y_1, y_2\}, \text{ and } Z = \{z_1, z_2, z_3\}$$

Consider the following fuzzy relations:

$$\tilde{R} = \begin{matrix} & y_1 & y_2 \\ x_1 & 0.7 & 0.5 \\ x_2 & 0.8 & 0.4 \end{matrix} \quad \text{and} \quad \tilde{S} = \begin{matrix} & z_1 & z_2 & z_3 \\ y_1 & 0.9 & 0.6 & 0.5 \\ y_2 & 0.1 & 0.7 & 0.5 \end{matrix}$$

Using max-min composition,

$$\begin{aligned} \mu_{\tilde{T}}(x_1, z_1) &= \vee_{y \in Y} (\mu_{\tilde{R}}(x_1, y) \wedge \mu_{\tilde{S}}(y, z_1)) \\ &= \max[\min(0.7, 0.9), \min(0.5, 0.1)] \\ &= 0.7 \end{aligned} \quad \left\{ \begin{matrix} & z_1 & z_2 & z_3 \\ \tilde{T} = & \begin{matrix} 0.7 & 0.6 & 0.5 \\ 0.8 & 0.6 & 0.4 \end{matrix} \\ & x_1 & x_2 \end{matrix} \right.$$

Two fuzzy relations  $R$  and  $S$  are defined on sets  $A$ ,  $B$  and  $C$ . That is,  $R \subseteq A \times B$ ,  $S \subseteq B \times C$ . The composition  $S \bullet R = SR$  of two relations  $R$  and  $S$  is expressed by the relation from  $A$  to  $C$ :

For  $(x, y) \in A \times B$ ,  $(y, z) \in B \times C$ ,

$$\begin{aligned} \mu_{S \bullet R}(x, z) &= \max_y [\mu_R(x, y) \bullet \mu_S(y, z)] \\ &= \vee_y [\mu_R(x, y) \bullet \mu_S(y, z)] \end{aligned}$$

$M_{S \bullet R} = M_R \bullet M_S$  (matrix notation)  
(max-product composition)

Max-product example:

$$X = \{x_1, x_2\}, Y = \{y_1, y_2\}, \text{ and } Z = \{z_1, z_2, z_3\}$$

Consider the following fuzzy relations:

$$\tilde{R} = \begin{matrix} y_1 \\ x_1 & [0.7 & 0.5] \\ x_2 & [0.8 & 0.4] \end{matrix} \quad \text{and} \quad \tilde{S} = \begin{matrix} z_1 \\ y_1 & [0.9 & 0.6 & 0.5] \\ y_2 & [0.1 & 0.7 & 0.5] \end{matrix}$$

Using max-product composition,

$$\left. \begin{aligned} \mu_{\tilde{T}}(x_2, z_2) &= \vee_{y \in Y} (\mu_{\tilde{R}}(x_2, y) * \mu_{\tilde{S}}(y, z_2)) \\ &= \max[(0.8, 0.6), (0.4, 0.7)] \\ &= 0.48 \end{aligned} \right\} \quad \tilde{T} = \begin{matrix} z_1 & z_2 & z_3 \\ x_1 & [.63 & .42 & .25] \\ x_2 & [.72 & .48 & .20] \end{matrix}$$

# CLASSICAL EQUIVALENCE RELATION

Let relation  $R$  on universe  $X$  be a relation from  $X$  to  $X$ . Relation  $R$  is an equivalence relation if the following three properties are satisfied.

1. Reflexivity
2. Symmetry
3. Transitivity

The function theoretic forms of representation of these properties are as follows:

1. Reflexivity:

$$\chi_R(x_i, x_i) = 1 \text{ or } (x_i, x_i) \in R$$

2. Symmetry:

$$\begin{aligned}\chi_R(x_i, x_j) &= \chi_R(x_j, x_i) \\ \text{i.e., } (x_i, x_j) \in R &\Rightarrow (x_j, x_i) \in R\end{aligned}$$

3. Transitivity:

$$\begin{aligned}\chi_R(x_i, x_j) \text{ and } \chi_R(x_j, x_k) &= 1, \text{ so } \chi_R(x_i, x_k) = 1 \\ \text{i.e., } (x_i, x_j) \in R, (x_j, x_k) \in R, &\text{ so } (x_i, x_k) \in R\end{aligned}$$

# CLASSICAL TOLERANCE RELATION

A tolerance relation  $R_1$  on universe  $X$  is one where the only the properties of reflexivity and symmetry are satisfied. The tolerance relation can also be called proximity relation. An equivalence relation can be formed from tolerance relation  $R_1$  by  $(n - 1)$  compositions within itself, where  $n$  is the cardinality of the set that defines  $R_1$ , here it is  $X$ , i.e.

$$\underbrace{R_1^{n-1}}_{\substack{\text{Tolerance} \\ \text{relation}}} = R_1 \circ R_1 \circ \cdots \circ R_1 = \underbrace{R}_{\substack{\text{Equivalence} \\ \text{relation}}}$$

# FUZZY EQUIVALENCE RELATION

Let  $\tilde{R}$  be a fuzzy relation on universe  $X$ , which maps elements from  $X$  to  $X$ . Relation  $\tilde{R}$  will be a fuzzy equivalence relation if all the three properties – reflexive, symmetry and transitivity – are satisfied. The membership function theoretic forms for these properties are represented as follows:

1. Reflexivity:

$$\mu_{\tilde{R}}(x_i, x_i) = 1 \quad \forall x \in X$$

If this is not the case for few  $x \in X$ , then  $R(X, X)$  is said to be irreflexive.

2. Symmetry:

$$\mu_{\tilde{R}}(x_i, x_j) = \mu_{\tilde{R}}(x_j, x_i) \text{ for all } x_i, x_j \in X$$

If this is not satisfied for few  $x_i, x_j \in X$ , then  $R(X, X)$  is called asymmetric.

3. Transitivity:

$$\mu_{\tilde{R}}(x_i, x_j) = \lambda_1 \text{ and } \mu_{\tilde{R}}(x_j, x_k) = \lambda_2$$

$$\Rightarrow \mu_{\tilde{R}}(x_i, x_k) = \lambda$$

where

$$\lambda = \min [\lambda_1, \lambda_2]$$

$$\text{i.e., } \mu_{\tilde{R}}(x_i, x_k) \geq \max_{x_j \in X} \min[\mu_{\tilde{R}}(x_i, x_j), \mu_{\tilde{R}}(x_j, x_k)] \quad \forall (x_i, x_k) \in X^2$$

## FUZZY TOLERANCE RELATION

A binary fuzzy relation that possesses the properties of reflexivity and symmetry is called fuzzy tolerance relation or resemblance relation.

The equivalence relations are a special case of the tolerance relation. The fuzzy tolerance relation can be reformed into fuzzy equivalence relation in the same way as a crisp tolerance relation is reformed into crisp equivalence relation, i.e.,

$$\underbrace{R_1^{n-1}}_{\text{Fuzzy tolerance relation}} = R_1 \circ R_1 \circ \dots \circ R_1 = \underbrace{R}_{\text{Fuzzy equivalence relation}}$$

where 'n' is the cardinality of the set that defines  $R_1$ .

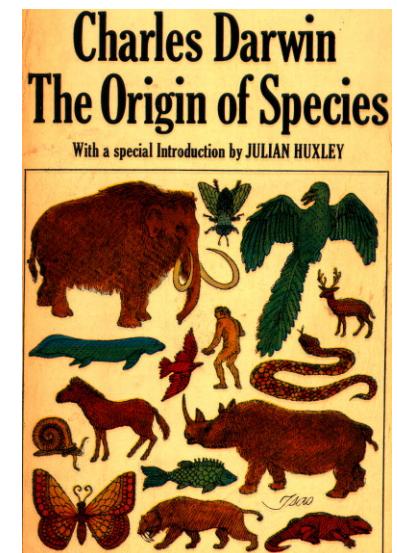
# GENETIC ALGORITHM

## General Introduction to GAs

- Genetic algorithms (GAs) are a technique to solve problems which need optimization.
- GAs are a subclass of **Evolutionary Computing** and are random search algorithms.
- GAs are based on Darwin's theory of evolution.

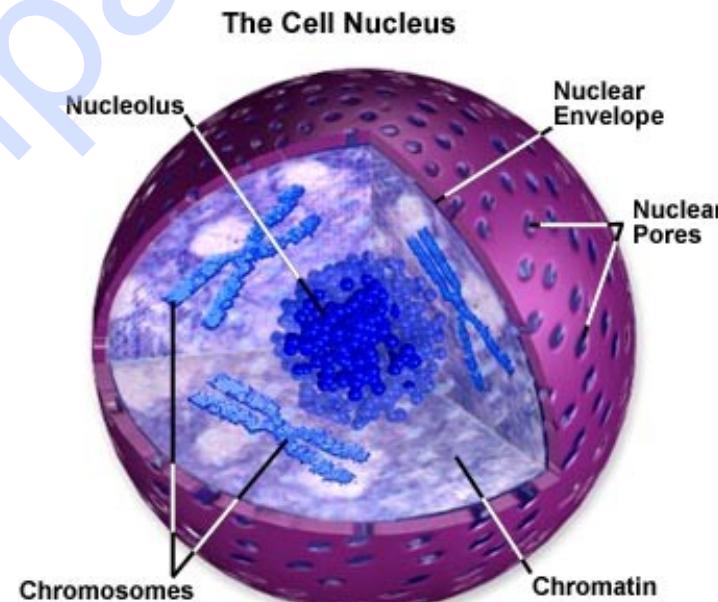
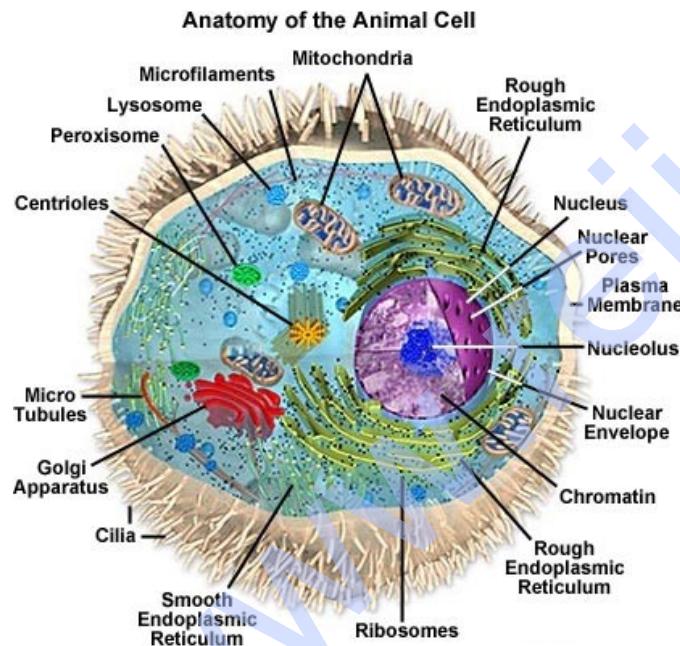
### History of GAs:

- Evolutionary computing evolved in the 1960s.
- GAs were created by John Holland in the mid-1970s.



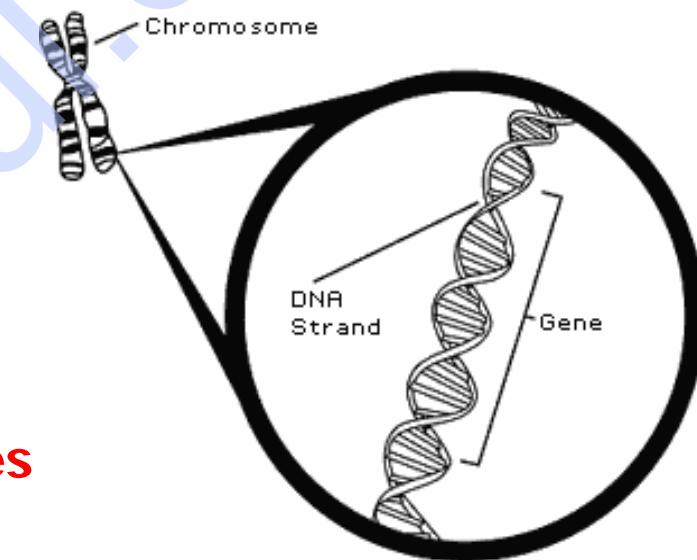
# Biological Background (1) – The Cell

- Every cell is a complex of many small “factories” working together.
- The center of this all is the **cell nucleus**.
- The nucleus contains the genetic information.



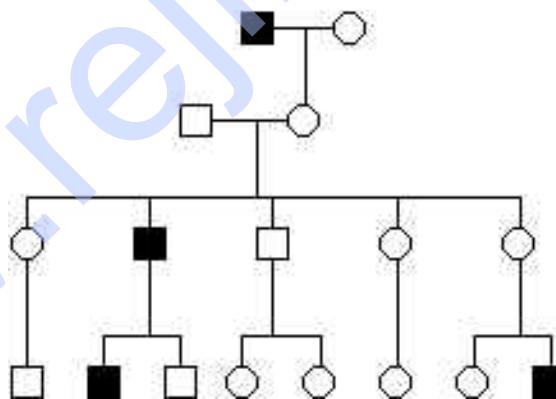
## Biological Background (2) – Chromosomes

- Genetic information is stored in the **chromosomes**.
- Each chromosome is build of **DNA**.
- Chromosomes in humans form pairs.
- There are **23 pairs**.
- The chromosome is divided in parts: **genes**
- Genes code for properties.
- The possibilities of the genes for one property is called: **allele**.
- Every gene has an unique position on the chromosome: **locus**.



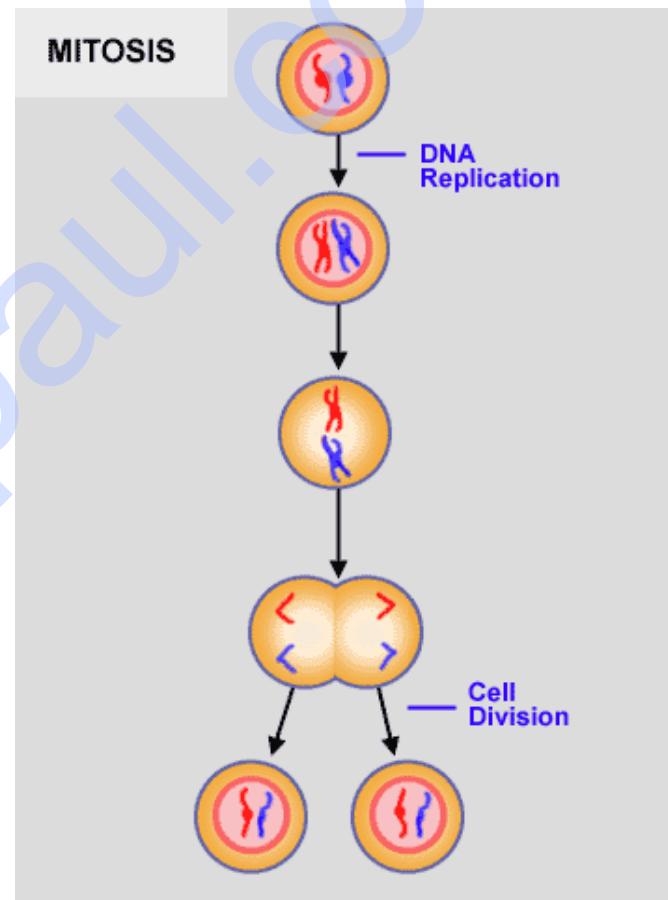
## Biological Background (3) – Genetics

- The entire combination of genes is called **genotype**.
- A genotype develops into a **phenotype**.
- **Alleles** can be either dominant or recessive.
- Dominant alleles will always express from the genotype to the phenotype.
- Recessive alleles can survive in the population for many generations without being expressed.



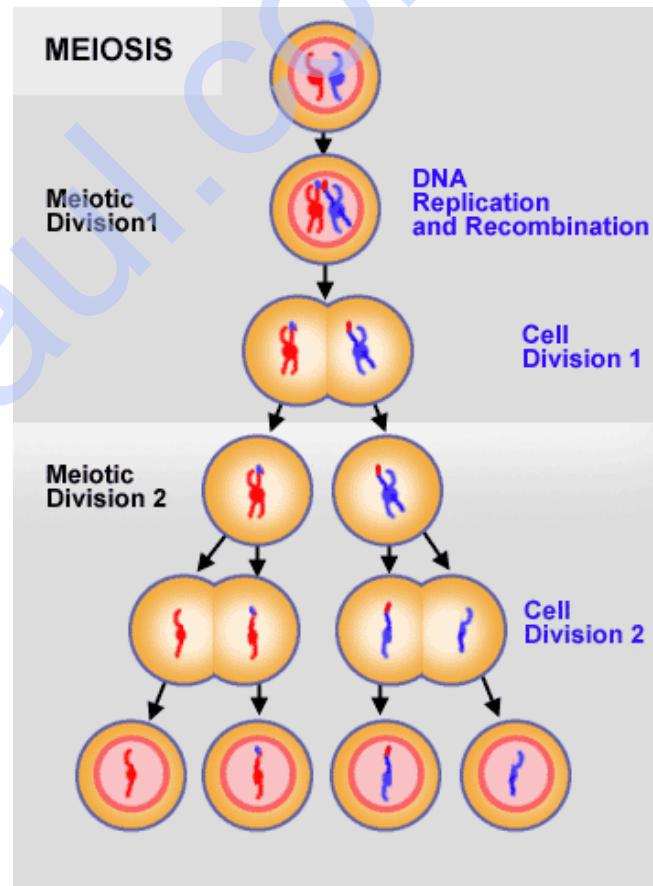
## Biological Background (4) – Reproduction

- Reproduction of genetical information:
  - Mitosis,
  - Meiosis.
- Mitosis is copying the same genetic information to new offspring: there is no exchange of information.
- Mitosis is the normal way of growing of multicell structures, like organs.



## Biological Background (5) – Reproduction

- Meiosis is the basis of sexual reproduction.
- After meiotic division 2 **gametes** appear in the process.
- In reproduction two gametes conjugate to a **zygote** which will become the new individual.
- Hence genetic information is shared between the parents in order to create new offspring.

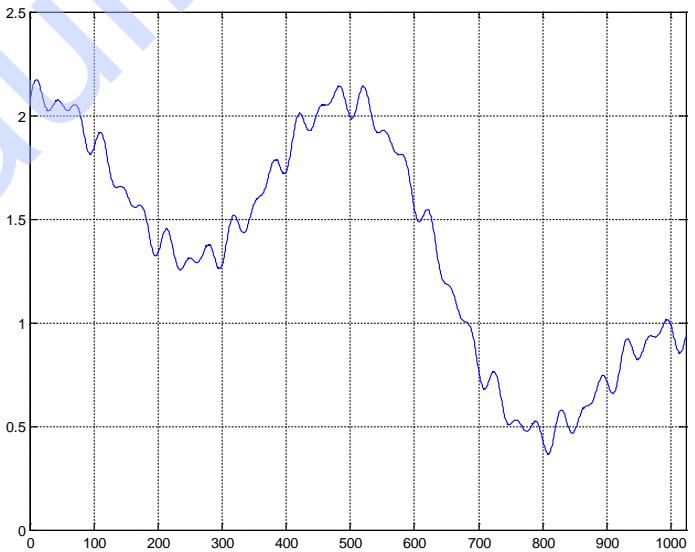


## Biological Background (6) – Natural Selection

- The origin of species: “Preservation of favorable variations and rejection of unfavorable variations.”
- There are more individuals born than can survive, so there is a continuous **struggle for life**.
- Individuals with an advantage have a greater chance for survival: **survival of the fittest**. For example, Giraffes with long necks.
- Genetic variations due to crossover and mutation.

# Genetic Algorithm (1) – Search Space

- Most often one is looking for the best solution in a specific subset of solutions.
- This subset is called the **search space** (or state space).
- Every point in the search space is a possible solution.
- Therefore every point has a **fitness** value, depending on the problem definition.
- GAs are used to search the search space for the best solution, e.g. a minimum.
- Difficulties are the local minima and the starting point of the search.



## Genetic Algorithm (2) – Basic Algorithm

- Starting with a subset of n randomly chosen solutions from the search space (i.e. chromosomes).  
This is the **population**.
- This population is used to produce a next **generation** of individuals by reproduction.
- Individuals with a higher **fitness** have more chance to reproduce (i.e. natural selection).

## Comparison of Natural and GA Terminology

Natural	Genetic Algorithm
Chromosome	String
Gene	Feature or character
Allele	Feature value
Locus	String position
Genotype	Structure
Phenotype	Parameter set, a decoded structure

## UNIT-III

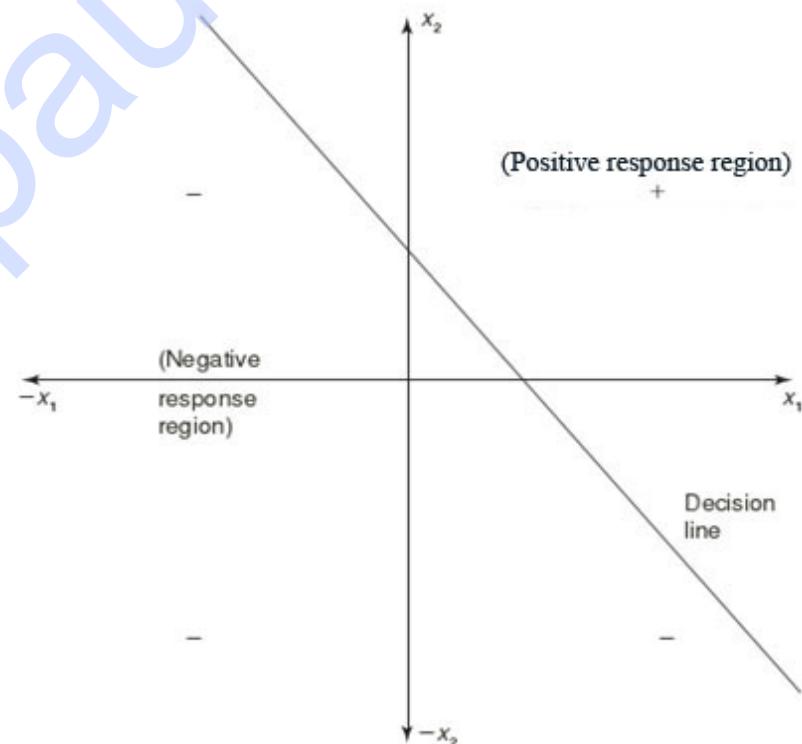
# ARTIFICIAL NEURAL NETWORKS

### McCULLOCH–PITTS NEURON

- Neurons are sparsely and randomly connected
- Firing state is binary (1 = firing, 0 = not firing)
- All but one neuron are excitatory (tend to increase voltage of other cells)
  - One inhibitory neuron connects to all other neurons
  - It functions to regulate network activity (prevent too many firings)

# LINEAR SEPARABILITY

- Linear separability is the concept wherein the separation of the input space into regions is based on whether the network response is positive or negative.
- Consider a network having positive response in the first quadrant and negative response in all other quadrants (AND function) with either binary or bipolar data, then the decision line is drawn separating the positive response region from the negative response region.



# HEBB NETWORK

Donald Hebb stated in 1949 that in the brain, the learning is performed by the change in the synaptic gap. **Hebb explained it:**

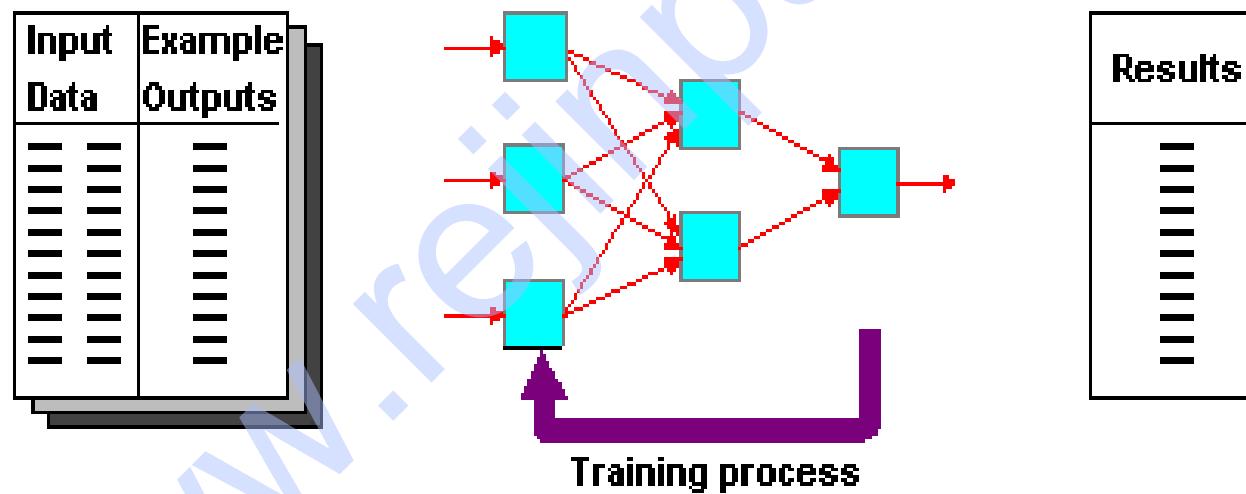
"When an axon of cell A is near enough to excite cell B, and repeatedly or permanently takes place in firing it, some growth process or metabolic change takes place in one or both the cells such that A's efficiency, as one of the cells firing B, is increased."

## HEBB LEARNING

- The weights between neurons whose activities are positively correlated are increased:  
$$\frac{dw_{ij}}{dt} \sim \text{correlation}(x_i, x_j)$$
- Associative memory is produced automatically
- The Hebb rule can be used for pattern association, pattern categorization, pattern classification and over a range of other areas.

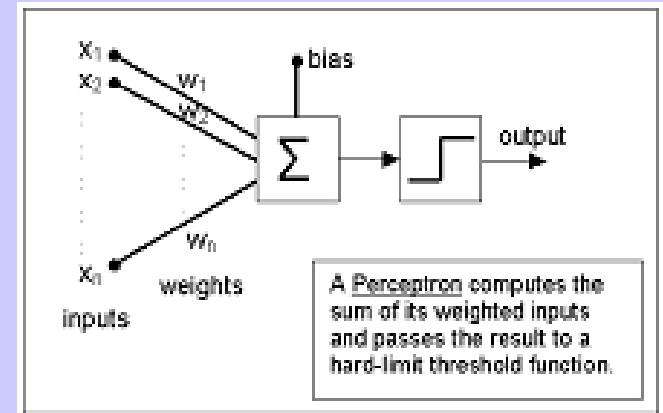
## SUPERVISED LEARNING NETWORKS

- Training and test data sets
- Training set; input & target are specified

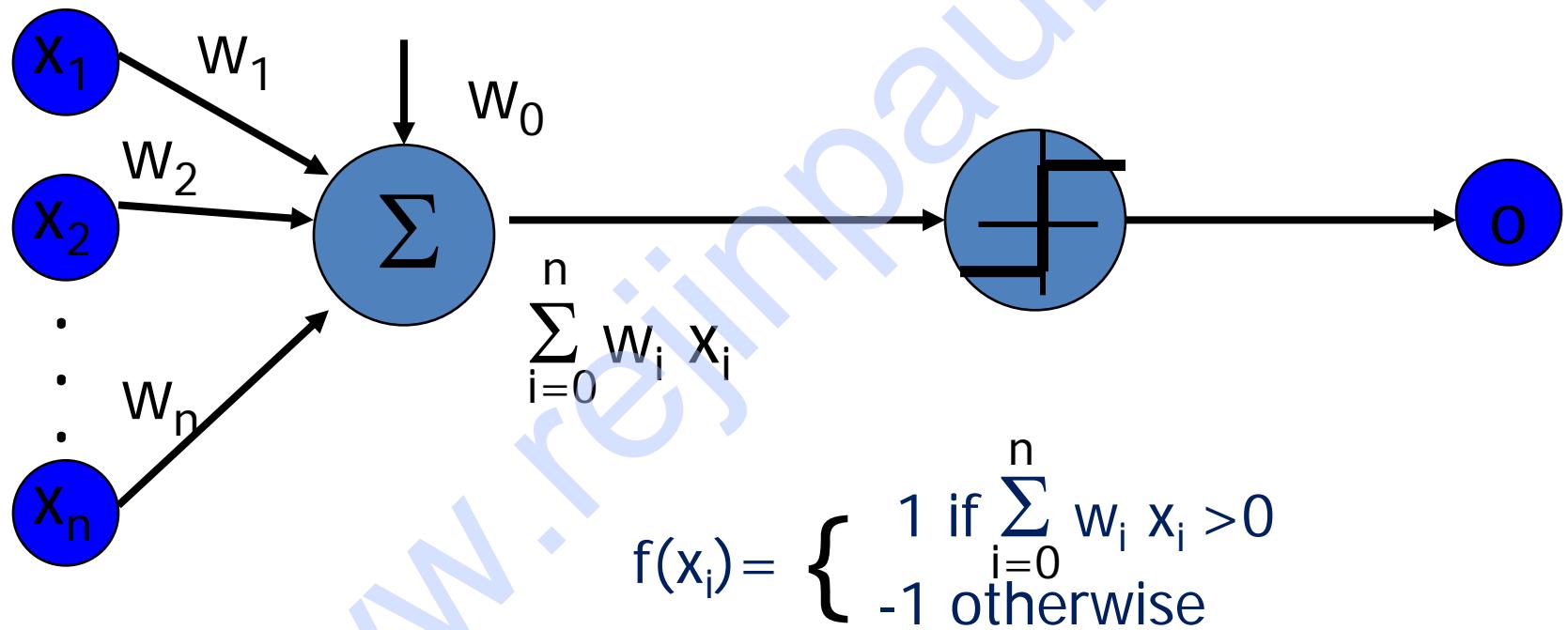


# PERCEPTRON NETWORKS

- One type of NN system is based on the “perceptron”.
- A perceptron computes a sum of weighted combination of its inputs, if the sum is greater than a certain threshold (bias), then it outputs a “1”, else a “-1”.



- Linear threshold unit (LTU)



# PERCEPTRON LEARNING

$$w_i = w_i + \Delta w_i$$

$$\Delta w_i = \eta (t - o) x_i$$

where

$t = c(x)$  is the target value,

$o$  is the perceptron output,

$\eta$  Is a small constant (e.g., 0.1) called **learning rate**.

- If the output is correct ( $t = o$ ) the weights  $w_i$  are not changed
- If the output is incorrect ( $t \neq o$ ) the weights  $w_i$  are changed such that the output of the perceptron for the new weights is closer to  $t$ .
- The algorithm **converges** to the correct classification
  - if the training data is linearly separable
  - $\eta$  is sufficiently small

# LEARNING ALGORITHM

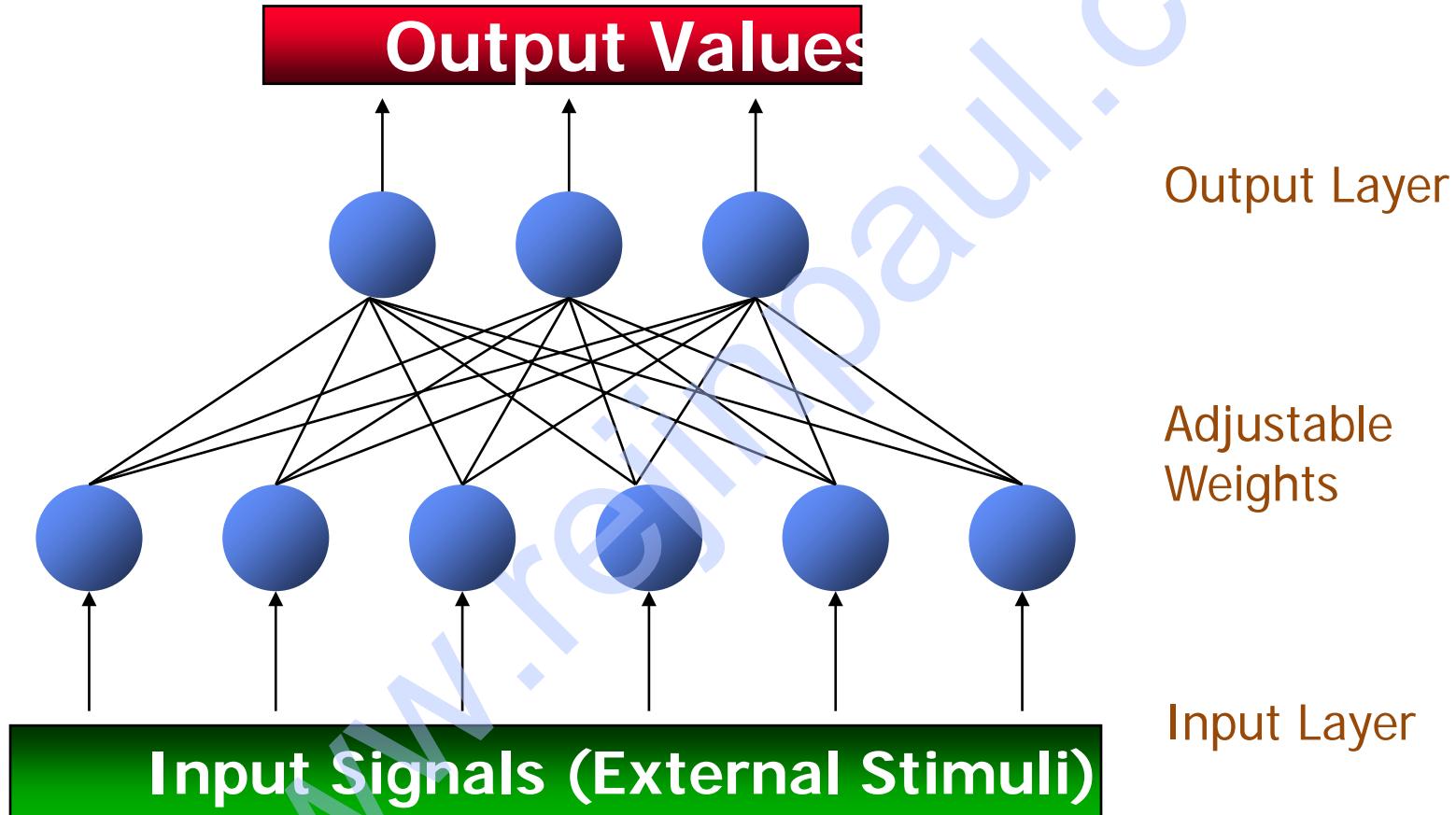
- **Epoch** : Presentation of the entire training set to the neural network.
- In the case of the **AND function**, an epoch consists of **four sets of inputs** being presented to the network (i.e. [0,0], [0,1], [1,0], [1,1]).
- **Error**: The error value is the amount by which the value output by the network differs from the target value. For example, if we required the network to output 0 and it outputs 1, then Error = -1.
- **Target Value, T** : When we are training a network we not only present it with the input but also with a value that we require the network to produce. For example, if we present the network with [1,1] for the AND function, the training value will be 1.

- **O** : The output value from the neuron.
- **I<sub>j</sub>** : Inputs being presented to the neuron.
- **W<sub>j</sub>** : Weight from input neuron ( $I_j$ ) to the output neuron.
- **LR** : The learning rate. This dictates how quickly the network converges. It is set by a matter of experimentation. It is typically 0.1.

## TRAINING ALGORITHM

- Adjust neural network weights to map inputs to outputs.
- Use a set of sample patterns where the desired output (given the inputs presented) is known.
- The purpose is to learn to
  - Recognize features which are common to good and bad exemplars

# MULTILAYER PERCEPTRON



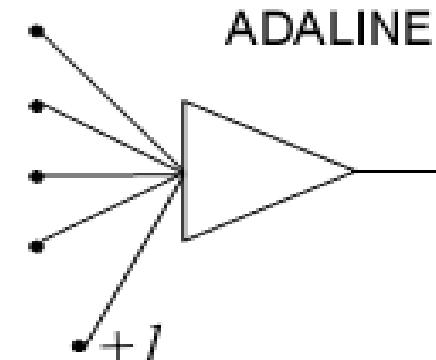
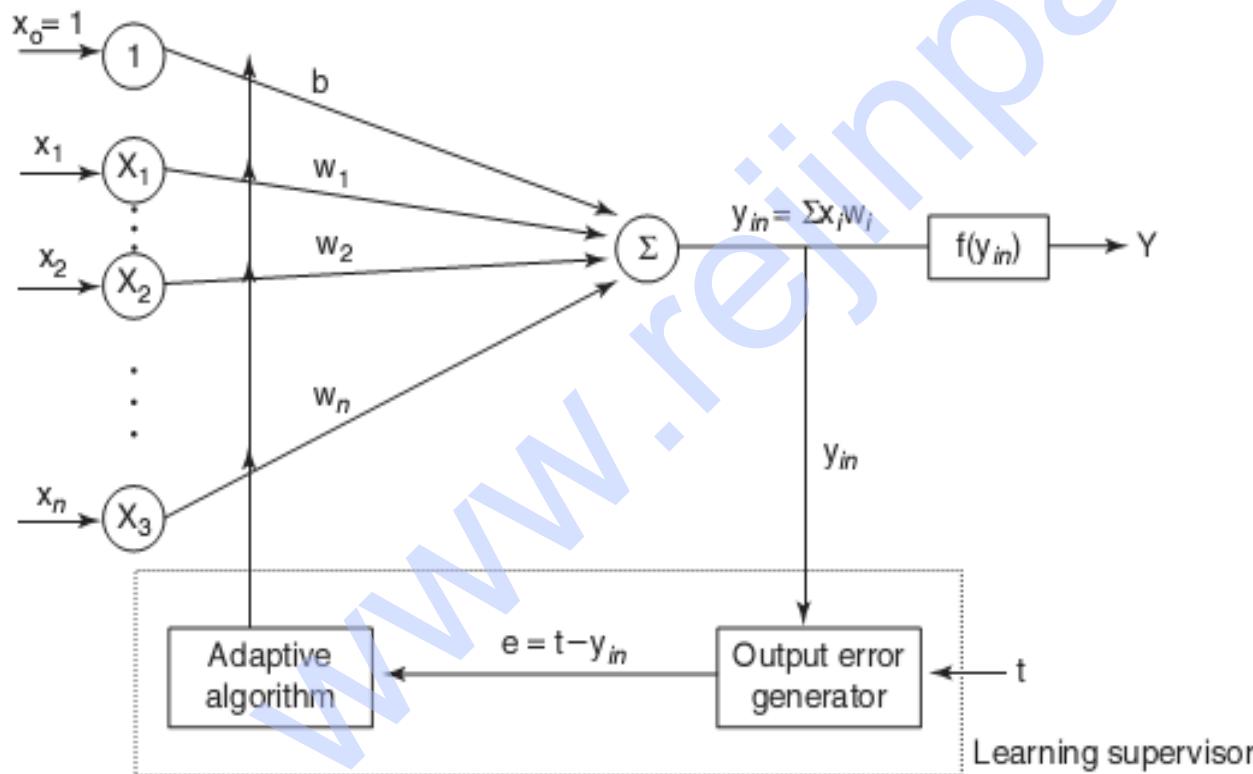
# LAYERS IN NEURAL NETWORK

- **The input layer:**
  - Introduces input values into the network.
  - No activation function or other processing.
- **The hidden layer(s):**
  - Performs classification of features.
  - Two hidden layers are sufficient to solve any problem.
  - Features imply more layers may be better.
- **The output layer:**
  - Functionally is just like the hidden layers.
  - Outputs are passed on to the world outside the neural network.

# ADAPTIVE LINEAR NEURON (ADALINE)

In 1959, Bernard Widrow and Marcian Hoff of Stanford developed models they called ADALINE (Adaptive Linear Neuron) and MADALINE (Multilayer ADALINE). These models were named for their use of **Multiple ADAptive LINear Elements**. MADALINE was the first neural network to be applied to a real world problem. It is an adaptive filter which eliminates echoes on phone lines.

## ADALINE MODEL



## ADALINE LEARNING RULE

Adaline network uses Delta Learning Rule. This rule is also called as Widrow Learning Rule or Least Mean Square Rule. The delta rule for adjusting the weights is given as ( $i = 1$  to  $n$ ):

$$\Delta w_i = \alpha(t - y_{in})x_i$$

$\Delta w_i$  = weight change

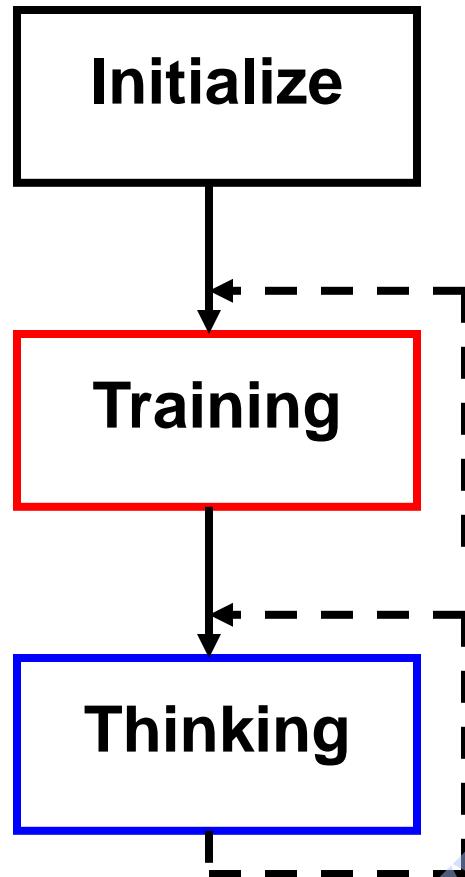
$\alpha$  = learning rate

$x$  = vector of activation of input unit

$y_{in}$  = net input to output unit, i.e.,  $Y = \sum_{i=1}^n x_i w_i$

$t$  = target output

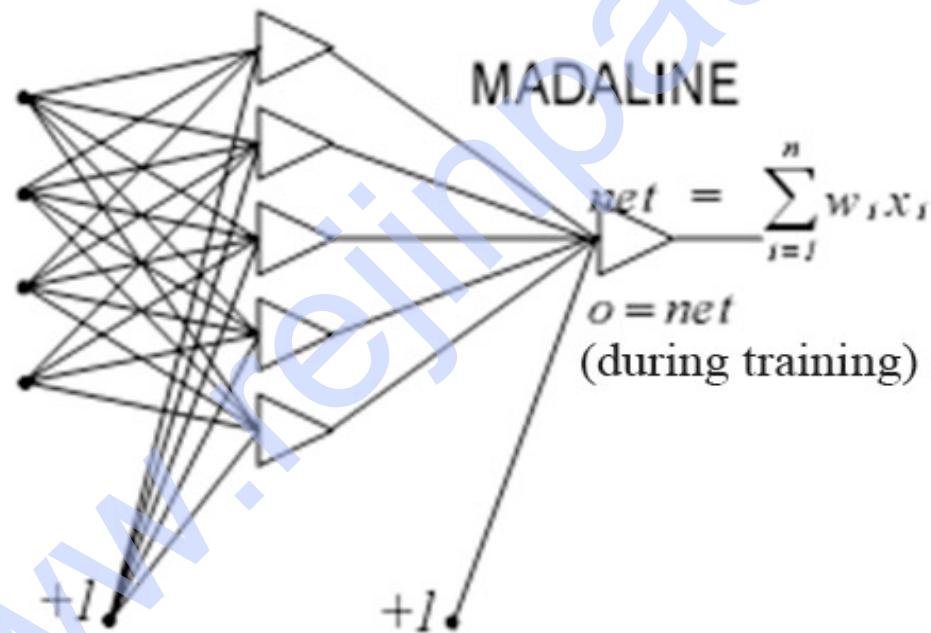
# USING ADALINE NETWORKS



- Initialize
  - Assign random weights to all links
- Training
  - Feed-in known inputs in random sequence
  - Simulate the network
  - Compute error between the input and the output (Error Function)
  - Adjust weights (Learning Function)
  - Repeat until total error <  $\epsilon$
- Thinking
  - Simulate the network
  - Network will respond to any input
  - Does not guarantee a correct solution even for trained inputs

# MADALINE NETWORK

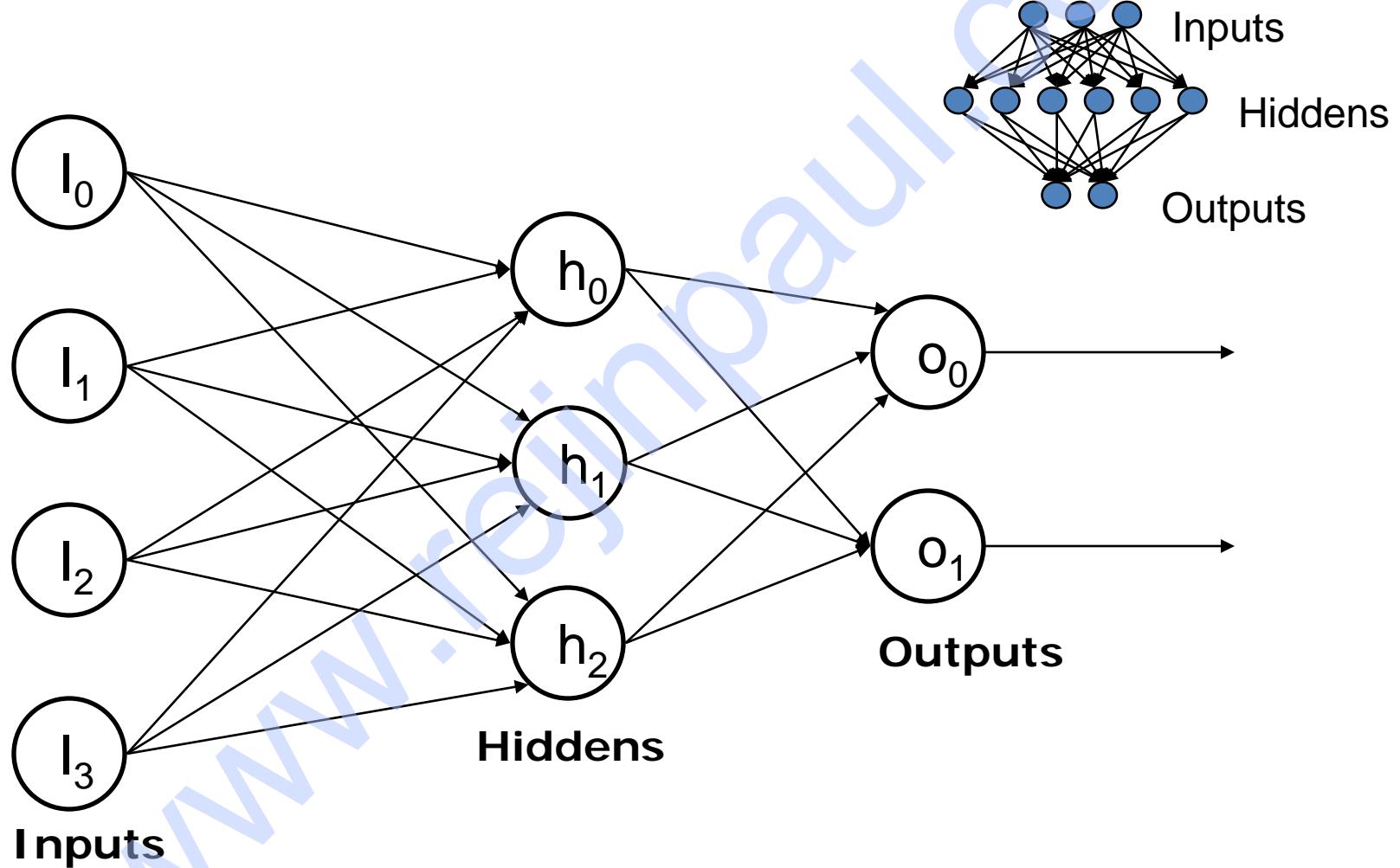
MADALINE is a Multilayer Adaptive Linear Element. MADALINE was the first neural network to be applied to a real world problem. It is used in several adaptive filtering process.



# BACK PROPAGATION NETWORK

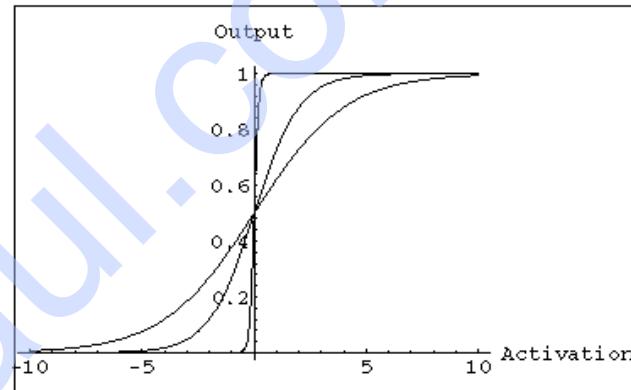
- Backprop implements a gradient descent search through the space of possible network weights, iteratively reducing the error E, between training example target values and the network outputs.
  - Guaranteed to converge only towards some local minima.
- 
- A training procedure which allows multilayer feed forward Neural Networks to be trained.
  - Can theoretically perform “any” input-output mapping.
  - Can learn to solve linearly inseparable problems.

# MULTILAYER FEEDFORWARD NETWORK



# MULTILAYER FEEDFORWARD NETWORK: ACTIVATION AND TRAINING

- For feed forward networks:
  - A continuous function can be
  - differentiated allowing
  - gradient-descent.
  - Back propagation is an example of a gradient-descent technique.
  - Uses sigmoid (binary or bipolar) activation function.



In multilayer networks, the activation function is usually more complex than just a threshold function, like  $1/[1+\exp(-x)]$  or even  $2/[1+\exp(-x)] - 1$  to allow for inhibition, etc.

# GRADIENT DESCENT

- Gradient-Descent(training\_examples,  $\eta$ )
- Each training example is a pair of the form  $\langle(x_1, \dots, x_n), t\rangle$  where  $(x_1, \dots, x_n)$  is the vector of input values, and  $t$  is the target output value,  $\eta$  is the learning rate (e.g. 0.1)
- Initialize each  $w_i$  to some small random value
- Until the termination condition is met, Do
  - Initialize each  $\Delta w_i$  to zero
  - For each  $\langle(x_1, \dots, x_n), t\rangle$  in training\_examples Do
    - ✓ Input the instance  $(x_1, \dots, x_n)$  to the linear unit and compute the output  $o$
    - ✓ For each linear unit weight  $w_i$  Do
      - $\Delta w_i = \Delta w_i + \eta (t - o) x_i$
      - For each linear unit weight  $w_i$  Do
      - $w_i = w_i + \Delta w_i$

## MODES OF GRADIENT DESCENT

- Batch mode : gradient descent

$w = w - \eta \nabla E_D[w]$  over the entire data D

$$E_D[w] = \frac{1}{2} \sum_d (t_d - o_d)^2$$

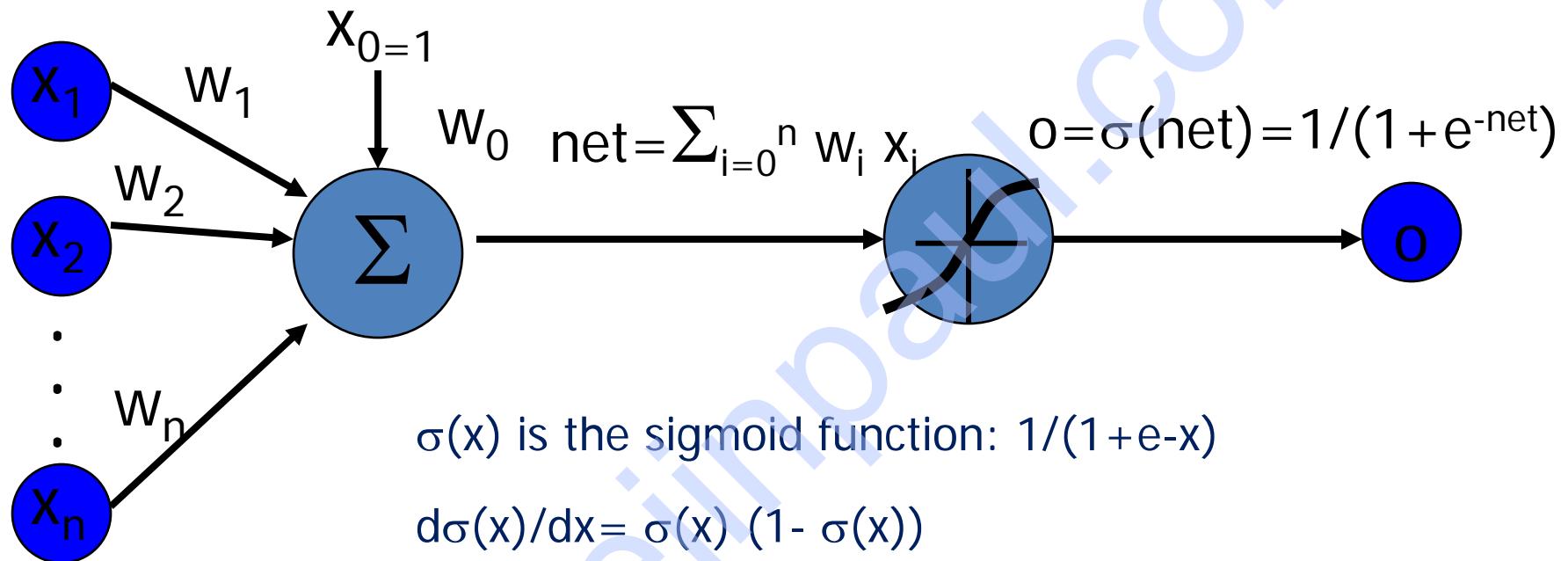
- Incremental mode: gradient descent

$w = w - \eta \nabla E_d[w]$  over individual training examples d

$$E_d[w] = \frac{1}{2} (t_d - o_d)^2$$

- Incremental Gradient Descent can approximate Batch Gradient Descent arbitrarily closely if  $\eta$  is small enough.

# SIGMOID ACTIVATION FUNCTION



Derive gradient decent rules to train:

- one sigmoid function
- $$\frac{\partial E}{\partial w_i} = -\sum d(td - od) od(1 - od) x_i$$
- Multilayer networks of sigmoid units backpropagation

# BACKPROPAGATION TRAINING ALGORITHM

- Initialize each  $w_i$  to some small random value.
- Until the termination condition is met, Do
  - For each training example  $\langle (x_1, \dots, x_n), t \rangle$  Do
  - Input the instance  $(x_1, \dots, x_n)$  to the network and compute the network outputs  $o_k$
  - For each output unit  $k$ 
    - $\delta_k = o_k(1 - o_k)(t_k - o_k)$
  - For each hidden unit  $h$ 
    - $\delta_h = o_h(1 - o_h) \sum_k w_{h,k} \delta_k$
  - For each network weight  $w_{i,j}$  Do
  - $w_{i,j} = w_{i,j} + \Delta w_{i,j}$  where
    - $\Delta w_{i,j} = \eta \delta_j x_{i,j}$

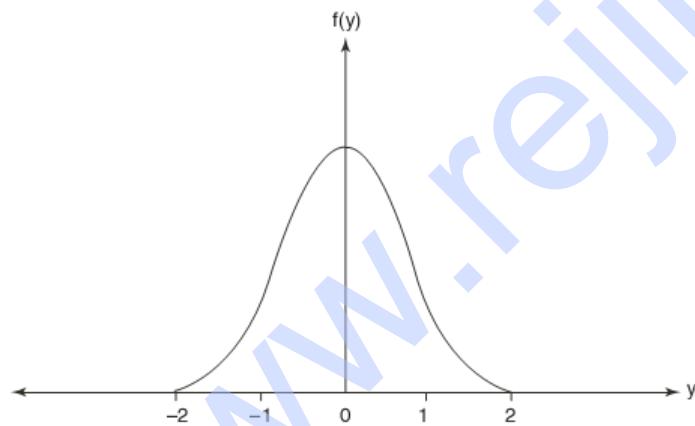
- Gradient descent over entire network weight vector
- Easily generalized to arbitrary directed graphs
- Will find a local, not necessarily global error minimum -in practice often works well (can be invoked multiple times with different initial weights)
- Often include weight momentum term  
$$\Delta w_{i,j}(t) = \eta \delta_j x_{i,j} + \alpha \Delta w_{i,j}(t-1)$$
- Minimizes error training examples
- Will it generalize well to unseen instances (over-fitting)?
- Training can be slow typical 1000-10000 iterations (use Levenberg-Marquardt instead of gradient descent)

## APPLICATIONS

- Load forecasting problems in power systems.
- Image processing.
- Fault diagnosis and fault detection.
- Gesture recognition, speech recognition.
- Signature verification.
- Bioinformatics.
- Structural engineering design (civil).

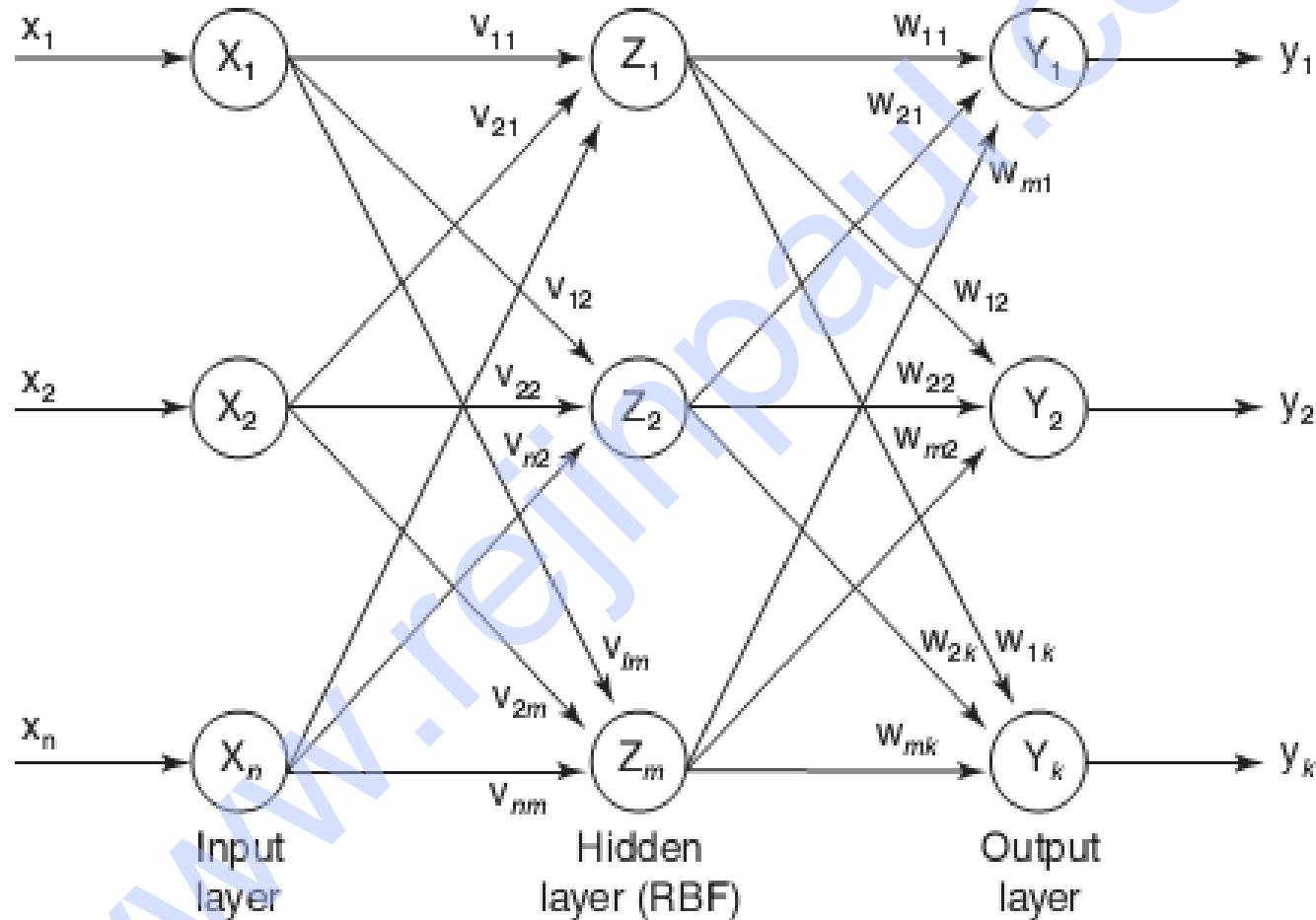
# RADIAL BASIS FUNCTION NETWORK

- The radial basis function (RBF) is a classification and functional approximation neural network developed by M.J.D. Powell.
- The network uses the most common nonlinearities such as sigmoidal and Gaussian kernel functions.
- The Gaussian functions are also used in regularization networks.
- The Gaussian function is generally defined as



$$f(y) = e^{-y^2}$$

# RADIAL BASIS FUNCTION NETWORK



Discussed on the several supervised learning networks like

- Perceptron,
- Adaline,
- Madaline,
- Backpropagation Network,
- Radial Basis Function Network.

Apart from these mentioned above, there are several other supervised neural networks like tree neural networks, wavelet neural network, functional link neural network and so on.

# ASSOCIATIVE MEMORY NETWORKS

## PATTERN ASSOCIATION

- **Associating patterns which are**
  - similar,
  - contrary,
  - in close proximity (spatial),
  - in close succession (temporal).
  
- **Associative recall**
  - evoke associated patterns,
  - recall a pattern by part of it,
  - evoke/recall with incomplete/noisy patterns.

# ASSOCIATIVE MEMORY (AM) NETWORK

- **Two types of associations exist. For two patterns  $s$  and  $t$** 
  - hetero-association ( $s \neq t$ ): relating two different patterns ( $s$  – input,  $t$  – target).
  - auto-association ( $s = t$ ): relating parts of a pattern with other parts.
- **Architectures of NN associative memory:**
  - single layer (with/out input layer),
  - two layers (for bidirectional association)
- **Learning algorithms for AM:**
  - Hebbian learning rule and its variations,
  - gradient descent.

# ASSOCIATIVE MEMORY NETWORK

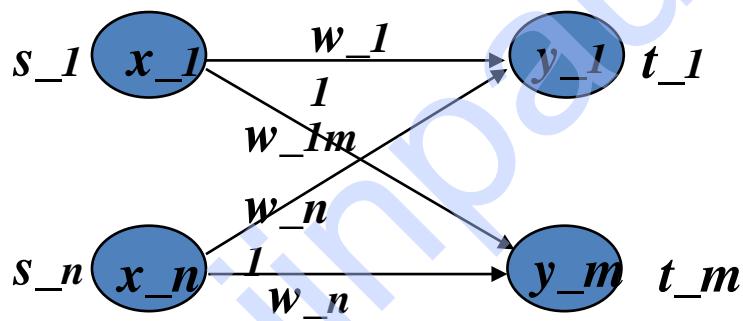
## ➤ WORKING PROCESS

- Recall a stored pattern by a noisy input pattern.
- Using the weights that capture the association.
- Stored patterns are viewed as “attractors”, each has its “attraction basin”.
- Often call this type of NN “associative memory” (recall by association, not explicit indexing/addressing).

# TRAINING ALGORITHM FOR ASSOCIATIVE MEMORY NETWORK

## ➤ Network structure: single layer

- one output layer of non-linear units and one input layer.



## ➤ Goal of learning:

- to obtain a set of weights  $w_{jj}$  from a set of training pattern pairs  $\{s:t\}$  such that when  $s$  is applied to the input layer,  $t$  is computed at the output layer,
- for all training pairs  $s:t$ ,  $t_j = f(sTw)$  for all  $j$ .

# HEBB RULE FOR PATTERN ASSOCIATION

## ➤ Algorithm (bipolar or binary patterns):

- For each training samples s:t:  $\Delta w_{ij} = s_i \cdot t_j$
- $\Delta w_{ij}$  increases if both  $s_i$  and  $t_j$  are ON (binary) or have the same sign (bipolar).

If  $\Delta w_{ij} = 0$ , then, after updates for all  $P$  training patterns,

$$w_{ij} = \sum_{p=1}^P s_i(p)t_j(p), \quad W = \{ w_{ij} \}$$

- Instead of obtaining  $W$  by iterative updates, it can be computed from the training set by calculating the outer product of  $s$  and  $t$ .

# OUTER PRODUCT FOR PATTERN ASSOCIATION

Let  $s$  and  $t$  be row vectors.

Then for a particular training pair  $s:t$

$$\Delta W(p) = s^T(p) \cdot t(p) = \begin{bmatrix} s_1 \\ \vdots \\ s_n \end{bmatrix} \begin{bmatrix} t_1, \dots, t_m \end{bmatrix} = \begin{bmatrix} s_1 t_1 \dots s_1 t_m \\ s_2 t_1 \dots s_2 t_m \\ \vdots \\ s_n t_1 \dots s_n t_m \end{bmatrix} = \begin{bmatrix} \Delta w_{11} \dots \Delta w_{1m} \\ \vdots \\ \Delta w_{n1} \dots \Delta w_{nm} \end{bmatrix}$$

and

$$W(p) = \sum_{p=1}^P s^T(p) \cdot t(p)$$

# HETERO-ASSOCIATIVE MEMORY NETWORK

- Binary pattern pairs  $s:t$  with  $|s| = 4$  and  $|t| = 2$ .
  - Total weighted input to output units:  $y\_in_j = \sum x_i w_{ij}$
  - Activation function: threshold  $y_j = \begin{cases} 1 & \text{if } y\_in_j^i > 0 \\ 0 & \text{if } y\_in_j \leq 0 \end{cases}$
  - Weights are computed by **Hebbian rule** (sum of outer products of all training pairs)
  - Training samples:
- $$W = \sum_{p=1}^P s_i^T(p) t_j(p)$$

	s(p)	t(p)
p=1	(1 0 0 0)	(1, 0)
p=2	(1 1 0 0)	(1, 0)
p=3	(0 0 0 1)	(0, 1)
p=4	(0 0 1 1)	(0, 1)

## COMPUTING THE WEIGHTS

$$s^T(1) \cdot t(1) = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

$$s^T(2) \cdot t(2) = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

$$s^T(3) \cdot t(3) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}$$

$$s^T(4) \cdot t(4) = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}$$

$$W = \begin{pmatrix} 2 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 2 \end{pmatrix}$$

## TEST/ RECALL THE NETWORK

$$x = [1 \ 0 \ 0 \ 0]$$

$$(1 \ 0 \ 0 \ 0) \begin{pmatrix} 2 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 2 \end{pmatrix} = (2 \ 0)$$

$$y_1 = 1, \quad y_2 = 0$$

$$x = [0 \ 1 \ 1 \ 0]$$

$$(0 \ 1 \ 1 \ 0) \begin{pmatrix} 2 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 2 \end{pmatrix} = (1 \ 1)$$

$$y_1 = 1, \quad y_2 = 1$$

$$x = [0 \ 1 \ 0 \ 0] \text{ similar to } s(1) \text{ and } s(2)$$

$$(0 \ 1 \ 0 \ 0) \begin{pmatrix} 2 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 2 \end{pmatrix} = (1 \ 0)$$

$$y_1 = 1, \quad y_2 = 0$$

$(1 \ 0 \ 0 \ 0), (1 \ 1 \ 0 \ 0)$  class  $(1, 0)$

$(0 \ 0 \ 0 \ 1), (0 \ 0 \ 1 \ 1)$  class  $(0, 1)$

$(0 \ 1 \ 1 \ 0)$  is not sufficiently similar to any class

# AUTO-ASSOCIATIVE MEMORY NETWORK

- Same as hetero-associative nets, except  $t(p) = s(p)$ .
- Used to recall a pattern by its noisy or incomplete version.  
(pattern completion/pattern recovery)
- A single pattern  $s = (1, 1, 1, -1)$  is stored (weights computed by Hebbian rule or outer product rule).

$$W = \begin{bmatrix} 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix}$$

training pattern

$$(111-1) \cdot W = (4\ 4\ 4-4) \rightarrow (111-1)$$

noisy pattern

$$(-111-1) \cdot W = (2\ 2\ 2-2) \rightarrow (111-1)$$

missing info

$$(0\ 0\ 1-1) \cdot W = (2\ 2\ 2-2) \rightarrow (111-1)$$

more noisy

$$(-1-11-1) \cdot W = (0\ 0\ 0) \text{ not recognized}$$

# AUTO-ASSOCIATIVE MEMORY NETWORK – DIAGONAL ELEMENTS

- Diagonal elements will dominate the computation when multiple patterns are stored ( $= P$ ).
- When  $P$  is large,  $\mathbf{W}$  is close to an identity matrix. This causes output = input, which may not be any stored pattern. The pattern correction power is lost.
- Replace diagonal elements by zero.

$$\mathbf{W}_0 = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}$$

$$(1 \ 1 \ 1 \ -1)\mathbf{W}' = (3 \ 3 \ 3 \ -3) \rightarrow (1 \ 1 \ 1 \ -1)$$

$$(-1 \ 1 \ 1 \ -1)\mathbf{W}' = (3 \ 1 \ 1 \ -1) \rightarrow (1 \ 1 \ 1 \ -1)$$

$$(0 \ 0 \ 1 \ -1)\mathbf{W}' = (2 \ 2 \ 1 \ -1) \rightarrow (1 \ 1 \ 1 \ -1)$$

$$(-1 \ -1 \ 1 \ -1)\mathbf{W}' = (1 \ 1 \ -1 \ 1) \rightarrow \text{wrong}$$

# STORAGE CAPACITY

- Number of patterns that can be correctly stored & recalled by a network.
- More patterns can be stored if they are not similar to each other (e.g., orthogonal).
- Non-orthogonal

$$(1 \ -1 \ -1 \ 1) \rightarrow \\ (1 \ 1 \ -1 \ 1)$$

$$W_0 = \begin{bmatrix} 0 & 0 & -2 & 2 \\ 0 & 0 & 0 & 0 \\ -2 & 0 & 0 & -2 \\ 2 & 0 & -2 & 0 \end{bmatrix}$$

$$(1 \ -1 \ -1 \ 1) \cdot W_0 = (1 \ 0 \ -1 \ 1)$$

It is not stored correctly

- Orthogonal

$$(1 \ 1 \ -1 \ -1) \\ (-1 \ 1 \ 1 \ -1) \rightarrow \\ (-1 \ 1 \ -1 \ 1)$$

$$W_0 = \begin{bmatrix} 0 & -1 & -1 & -1 \\ -1 & 0 & -1 & -1 \\ -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}$$

All three patterns can be correctly recalled

# BIDIRECTIONAL ASSOCIATIVE MEMORY (BAM) NETWORK

## Architecture:

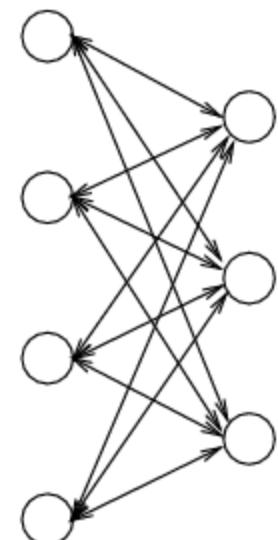
- Two layers of non-linear units: X-layer, Y-layer.
- Units: discrete threshold, continuing sigmoid (can be either binary or bipolar).

## Weights:

$$W_{n \times m} = \sum_{p=1}^P s^T(p) \cdot t(p) \quad (\text{Hebbian/outerproduct})$$

Symmetric:  $w_{ij} = w_{ji}$

Convert binary patterns to bipolar when constructing  $W$ .



## RECALL OF BAM NETWORK

Bidirectional, either by X (to recall Y) or by Y (to recall X).

Recurrent:

$$y(t) = [f(y\_in_1(t), \dots, f(y\_in_m(t))]$$

$$\text{where } y\_in_j(t) = \sum_{i=1}^n w_{ij} \cdot x_i(t-1)$$

$$x(t+1) = [f(x\_in_1(t+1), \dots, f(x\_in_n(t+1))]$$

$$\text{where } x\_in_i(t+1) = \sum_{j=1}^m w_{ij} \cdot y_j(t)$$

Update can be either asynchronous (as in hetero-associative memory) or synchronous (change all Y units at one time, then all X units the next time).

# ACTIVATION FUNCTIONS IN BAM

The activation function is based on whether the input target vector pairs used are binary or bipolar.

Activation function for the Y-layer	Activation function for the X-layer
With binary input vectors $y_j = \begin{cases} 1 & \text{if } y_{inj} > 0 \\ y_j & \text{if } y_{inj} = 0 \\ 0 & \text{if } y_{inj} < 0 \end{cases}$	With binary input vectors $x_i = \begin{cases} 1 & \text{if } x_{ini} > 0 \\ x & \text{if } x_{ini} = 0 \\ -1 & \text{if } x_{ini} < 0 \end{cases}$
With bipolar input vectors $y_j = \begin{cases} 1 & \text{if } y_{inj} > \theta_j \\ y_j & \text{if } y_{inj} = \theta_j \\ -1 & \text{if } y_{inj} < \theta_j \end{cases}$	With bipolar input vectors $x_i = \begin{cases} 1 & \text{if } x_{ini} > \theta_i \\ x & \text{if } x_{ini} = \theta_i \\ -1 & \text{if } x_{ini} < \theta_i \end{cases}$

# DISCRETE HOPFIELD NETWORK (DHN)

- A single-layer network
  - each node as both input and output units.
- More than an Associative Memory, Discrete Hopfield Network can be used in several applications.
  - Other applications such as combinatorial optimization.
- Different forms: discrete & continuous.
- Major contribution of John Hopfield to NN:
  - Treating a network as a dynamic system.
  - Introduction of energy function into Neural Network Research.

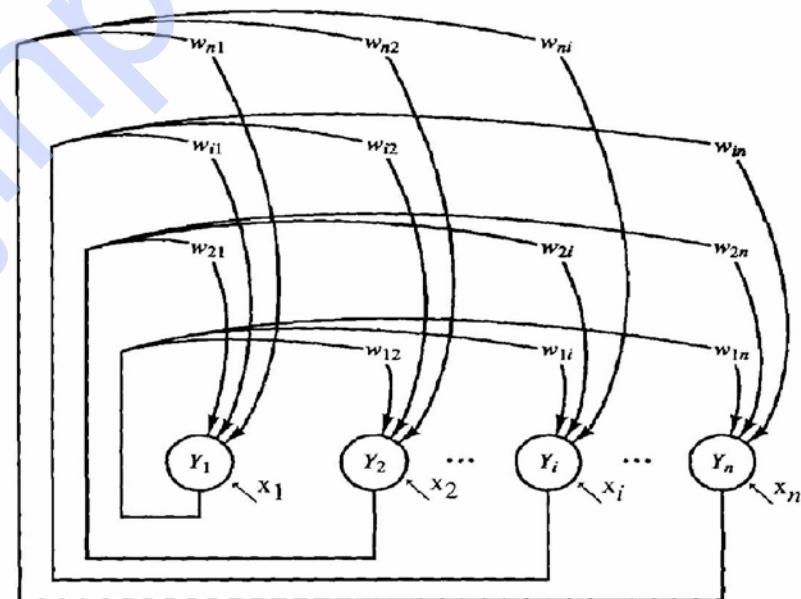
# ARCHITECTURE OF DHN

## ➤ Architecture

- Single-layer (units serve as both input and output):
  - ✓ nodes are threshold units (binary or bipolar).
  - ✓ weights: fully connected, symmetric, and zero diagonal.

$$\begin{aligned}w_{ij} &= w_{ji} \\w_{ii} &= 0\end{aligned}$$

$x_i$  are external inputs, which may be transient or permanent.



- **Weights:**

- To store patterns  $s(p)$ ,  $p=1,2,\dots,P$

**bipolar:**  $w_{ij} = \sum_p s_i(p)s_j(p) \quad i \neq j$

$$w_{ii} = 0$$

same as ~~Hebbian~~ rule (with zero diagonal)

**binary:**  $w_{ij} = \sum_p (2s_i(p)-1)(2s_j(p)-1) \quad i \neq j$

$$w_{ii} = 0$$

converting  $s(p)$  to bipolar when constructing  $W$ .

- Recall

- Use an input vector to recall a stored vector (book calls the application of DHM)
- Each time, randomly select a unit for update

### Recall Procedure

1. Apply recall input vector  $x$  to the network:  $y_i := x_i \quad i=1, 2, \dots, n$
2. While convergence = fails do
  - 2.1. Randomly select a unit
  - 2.2. Compute  $y\_in_i = x_i + \sum_{j \neq i} y_j w_{ji}$
  - 2.3. Determine activation of  $Y_i$ 
$$y_i = \begin{cases} 1 & \text{if } y\_in_i > \theta_i \\ y_i & \text{if } y\_in_i = \theta_i \\ -1 & \text{if } y\_in_i < \theta_i \end{cases}$$
  - 2.4. Periodically test for convergence.

## STORAGE CAPACITY OF DHN

$P$ : maximum number of random patterns of dimension  $n$  can be stored in a DHM of  $n$  nodes

Hopfield's observation:  $P \approx 0.15n$ ,  $\frac{P}{n} \approx 0.15$

Theoretical analysis:  $P \approx \frac{n}{2\log_2 n}$ ,  $\frac{P}{n} \approx \frac{1}{2\log_2 n}$

$P/n$  decreases because larger  $n$  leads to more interference between stored patterns.

Some work to modify  $HM$  to increase its capacity to close to  $n$ ,  $W$  is trained (not computed by Hebbian rule).

# CONTINUOUS HOPFIELD NET

## ➤ Architecture

- Continuous node output, and continuous time
- Fully connected with symmetric weights  $w_{ij} = w_{ji}$ ,  $w_{ii} = 0$
- Internal activation  $u_i$ : with  $\frac{du_i(t)}{dt} = \sum_{j=1}^n w_{ij}x_j(t) + \theta_i = net_i(t)$
- Output (state)  $x_i(t) = f(u_i(t))$

where  $f$  is a sigmoid function to ensure binary/bipolar output. E.g. for bipolar, use hyperbolic tangent function:

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

# CONTINUOUS HOPFIELD NET

Computation: all units change their output (states) at the same time, based on states of all others.

- Compute net:  $net_i(t) = \sum_{j=1}^n w_{ij}x_j(t) + \theta_i$
- Compute internal activation  $u_i(t)$  by first-order Taylor expansion

$$u_i(t + \delta) = \int in_i(t)dt \approx u_i(t) + \frac{du_i(t)}{dt} \cdot \delta + \dots = u_i(t) + net_i \cdot \delta$$

- Compute output

$$x_i(t) = f(u_i(t))$$

# ITERATIVE ASSOCIATIVE MEMORY NETWORK

Example

$$x = (1, 1, 1, -1) \quad W = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}$$

Output units are  
threshold units

An incomplete recall input :  $x' = (1, 0, 0, 0)$

$$Wx' = (0, 1, 1, -1) = x''$$

$$Wx'' = (3, 2, 2, -2) \rightarrow (1, 1, 1, -1) = x$$

In general: using current output as input of the next iteration

$x(0)$  = initial recall input

$x(I) = S(Wx(I-1)), \quad I = 1, 2, \dots$

until  $x(N) = x(K)$  for some  $K < N$

Dynamic System: State vector  $x(I)$

If  $K = N-1$ ,  $x(N)$  is a stable state (fixed point)

$$f(Wx(N)) = f(Wx(N-1)) = x(N)$$

If  $x(K)$  is one of the stored pattern, then  $x(K)$  is called a **genuine memory**

Otherwise,  $x(K)$  is a **spurious memory** (caused by cross-talk/interference between genuine memories)

Each fixed point (genuine or spurious memory) is an **attractor** (with different attraction basin)

If  $K \neq N-1$ , **limit-circle**,

The network will repeat

$x(K), x(K+1), \dots, x(N) = x(K)$  when iteration continues.

Iteration will eventually stop because the total number of distinct state is finite ( $3^n$ ) if threshold units are used. If patterns are continuous, the system may continue evolve forever (**chaos**) if no such  $K$  exists.

# UNSUPERVISED LEARNING NETWORKS

- No help from the outside.
- No training data, no information available on the desired output.
- Learning by doing.
- Used to pick out structure in the input:
  - Clustering,
  - Reduction of dimensionality → compression.
- Example: Kohonen's Learning Law.

There exists several networks under this category, such as

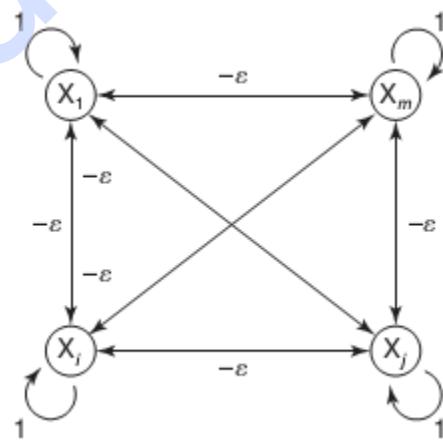
- Max Net,
- Mexican Hat,
- Kohonen Self-organizing Feature Maps,
- Learning Vector Quantization,
- Counterpropagation Networks,
- Hamming Network,
- Adaptive Resonance Theory.

## COMPETITIVE LEARNING

- Output units compete, so that eventually only one neuron (the one with the most input) is active in response to each output pattern.
- The total weight from the input layer to each output neuron is limited. If some connections are strengthened, others must be weakened.
- A consequence is that the winner is the output neuron whose weights best match the activation pattern.

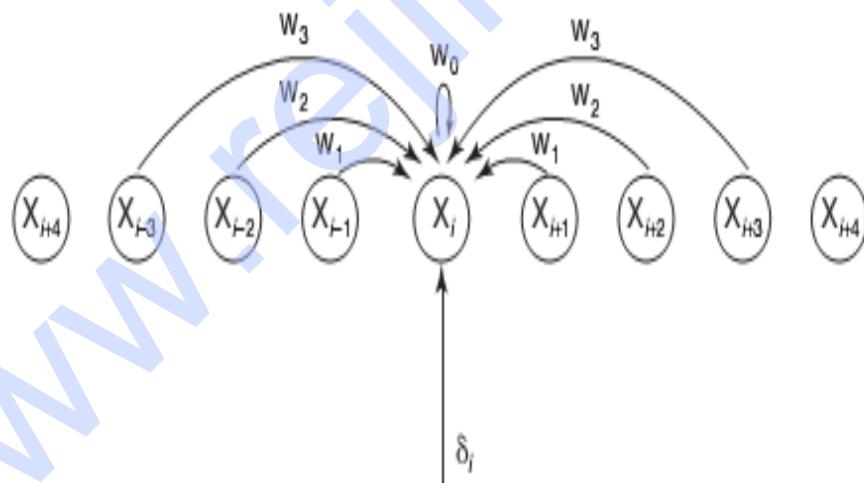
## MAX NET

- Max Net is a fixed weight competitive net.
- Max Net serves as a subnet for picking the node whose input is larger. All the nodes present in this subnet are fully interconnected and there exist symmetrical weights in all these weighted interconnections.
- The weights between the neurons are inhibitory and fixed.
- The architecture of this net is as shown:



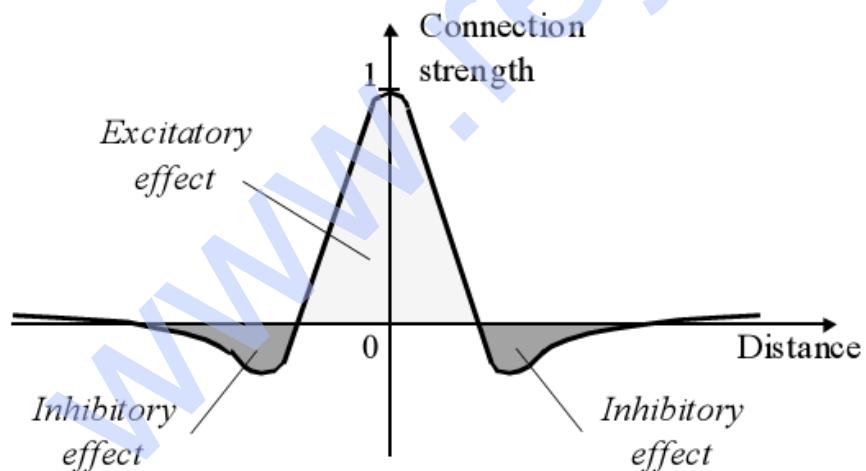
## MEXICAN HAT NETWORK

- Kohonen developed the Mexican hat network which is a more generalized contrast enhancement network compared to the earlier Max Net.
- Here, in addition to the connections within a particular layer of neural net, the neurons also receive some other external signals. This interconnection pattern is repeated for several other neurons in the layer. The architecture for the network is as shown below:



# MEXICAN HAT NETWORK

- The lateral connections are used to create a competition between neurons. The neuron with the largest activation level among all neurons in the output layer becomes the winner. This neuron is the only neuron that produces an output signal. The activity of all other neurons is suppressed in the competition.
  
- The lateral feedback connections produce excitatory or inhibitory effects, depending on the distance from the winning neuron. This is achieved by the use of a ***Mexican Hat function*** which describes synaptic weights between neurons in the output layer.

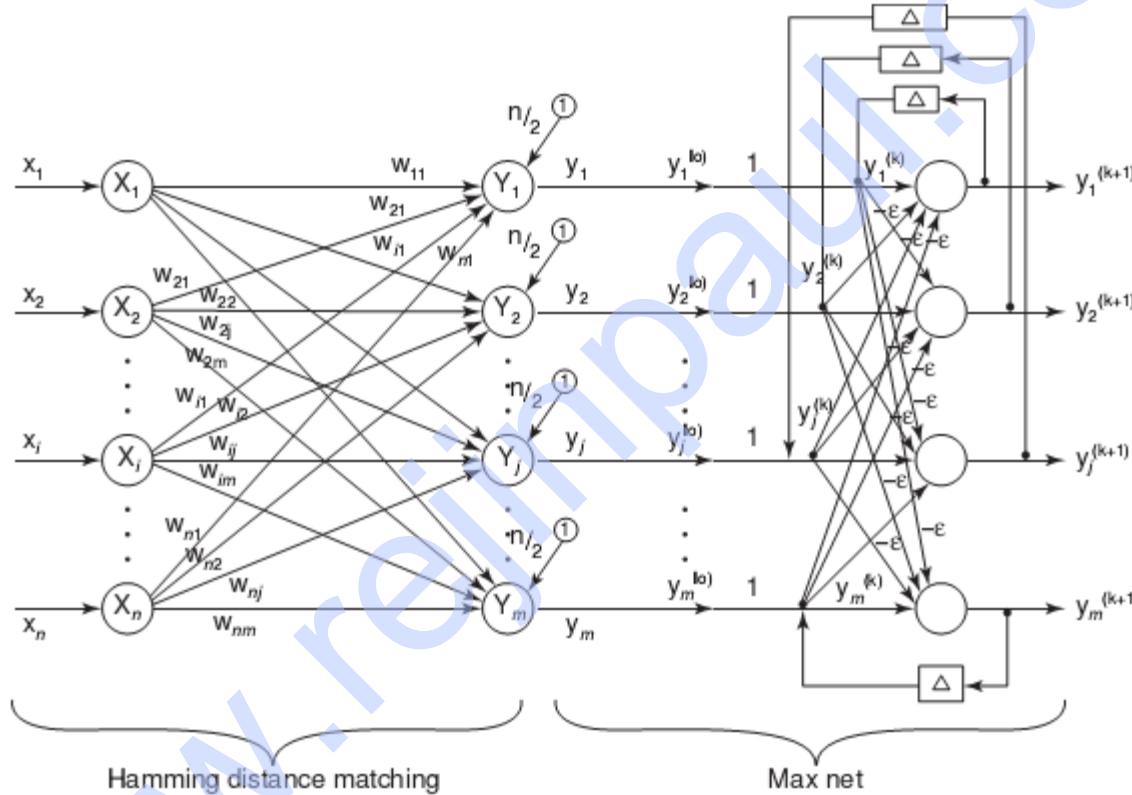


**MEXICAN HAT FUNCTION OF LATERAL CONNECTION**

## HAMMING NETWORK

- The Hamming network selects stored classes, which are at a maximum Hamming distance ( $H$ ) from the noisy vector presented at the input.
- The **Hamming distance** between the two vectors is the number of components in which the vectors differ.
- The Hamming network consists of **two layers**.
  - The first layer computes the difference between the total number of components and Hamming distance between the input vector  $x$  and the stored pattern of vectors in the feed forward path.
  - The second layer of the Hamming network is composed of Max Net (used as a subnet) or a winner-take-all network which is a recurrent network.

# ARCHITECTURE OF HAMMING NET



## SELF-ORGANIZATION

- Network Organization is fundamental to the brain
  - Functional structure.
  - Layered structure.
  - Both parallel processing and serial processing require organization of the brain.

## SELF-ORGANIZING FEATURE MAP

Our brain is dominated by the cerebral cortex, a very complex structure of billions of neurons and hundreds of billions of synapses. The cortex includes areas that are responsible for different human activities (motor, visual, auditory, etc.) and associated with different sensory inputs. One can say that each sensory input is mapped into a corresponding area of the cerebral cortex. *The cortex is a self-organizing computational map in the human brain.*

# SELF-ORGANIZING NETWORKS

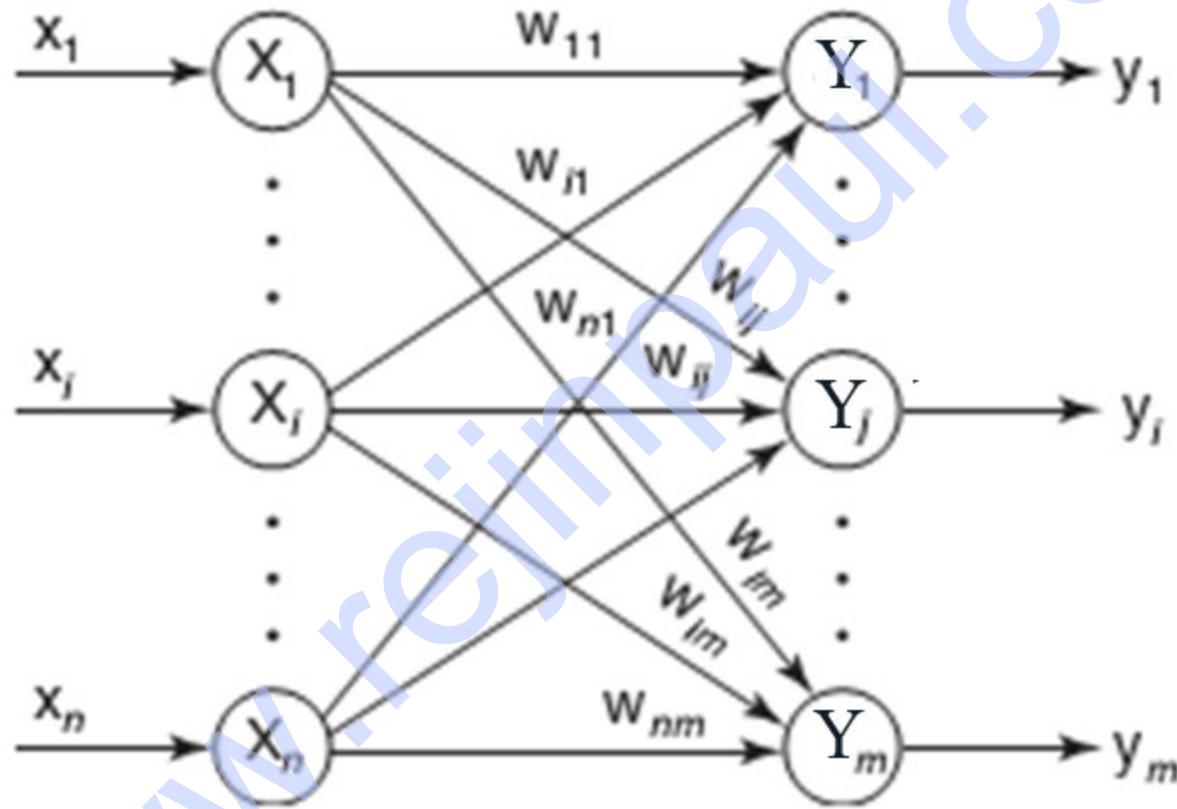
www.rejinpaul.com

- Discover significant patterns or features in the input data.
- Discovery is done without a teacher.
- Synaptic weights are changed according to local rules.
- The changes affect a neuron's immediate environment until a final configuration develops.

## KOHONEN SELF-ORGANIZING FEATURE MAP (KSOFM)

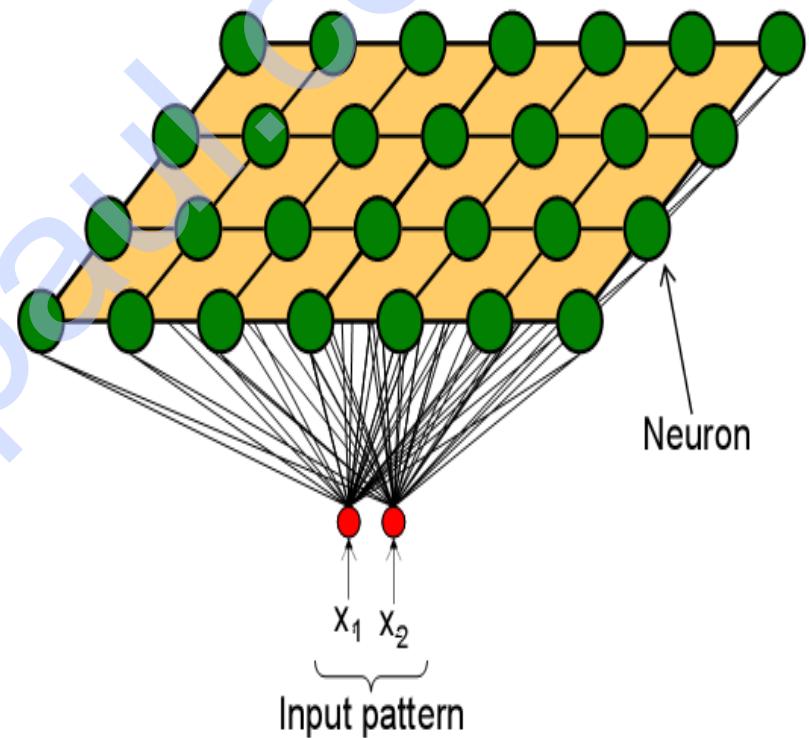
- The Kohonen model provides a **topological mapping**.
- It places a fixed number of input patterns from the input layer into a higher dimensional output or Kohonen layer.
- Training in the Kohonen network begins with the winner's neighborhood of a fairly large size. Then, as training proceeds, the neighborhood size gradually decreases.
- Kohonen SOMs result from the synergy of **three basic processes**
  - Competition,
  - Cooperation,
  - Adaptation.

## ARCHITECTURE OF KSOFM



## COMPETITION OF KSOFM

- Each neuron in an SOM is assigned a weight vector with the same dimensionality N as the input space.
- Any given input pattern is compared to the weight vector of each neuron and the closest neuron is declared the winner.
- The Euclidean norm is commonly used to measure distance.



## CO-OPERATION OF KSOFM

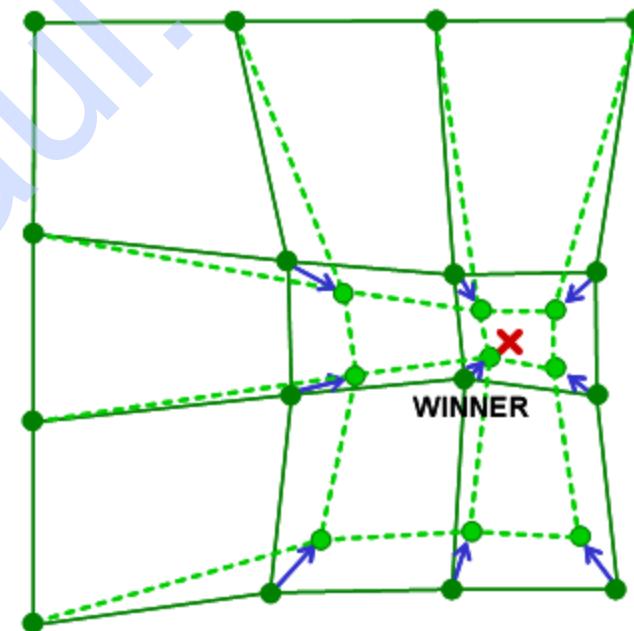
- The activation of the winning neuron is spread to neurons in its immediate neighborhood.
  - This allows topologically close neurons to become sensitive to similar patterns.
- The winner's neighborhood is determined on the lattice topology.
  - Distance in the lattice is a function of the number of lateral connections to the winner.
- The size of the neighborhood is initially large, but shrinks over time.
  - An initially large neighborhood promotes a topology-preserving mapping.
  - Smaller neighborhoods allow neurons to specialize in the latter stages of training.

## ADAPTATION OF KSOFM

During training, the winner neuron and its topological neighbors are adapted to make their weight vectors more similar to the input pattern that caused the activation.

Neurons that are closer to the winner will adapt more heavily than neurons that are further away.

The magnitude of the adaptation is controlled with a learning rate, which decays over time to ensure convergence of the SOM.



# KSOFM ALGORITHM

## Step 1: Initialization

Set initial synaptic weights to small random values, say in an interval [0, 1], and assign a small positive value to the learning rate parameter  $\alpha$ .

## Step 2: Activation and Similarity Matching.

Activate the Kohonen network by applying the input vector  $\mathbf{X}$ , and find the winner-takes-all (best matching) neuron  $j_{\mathbf{X}}$  at iteration  $p$ , using the minimum-distance Euclidean criterion

$$j_{\mathbf{X}}(p) = \min_j \|\mathbf{X} - \mathbf{W}_j(p)\| = \left\{ \sum_{i=1}^n [x_i - w_{ij}(p)]^2 \right\}^{1/2},$$

where  $n$  is the number of neurons in the input layer, and  $m$  is the number of neurons in the Kohonen layer.

**Step 3: Learning.**

Update the synaptic weights

$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$

where  $\Delta w_{ij}(p)$  is the weight correction at iteration  $p$ .

The weight correction is determined by the competitive learning rule:

$$\Delta w_{ij}(p) = \begin{cases} \alpha [x_i - w_{ij}(p)], & j \in \Lambda_j(p) \\ 0, & j \notin \Lambda_j(p) \end{cases}$$

where  $\alpha$  is the *learning rate* parameter, and  $\Lambda_j(p)$  is the neighbourhood function centred around the winner-takes-all neuron  $j_X$  at iteration  $p$ .

**Step 4: Iteration.**

Increase iteration  $p$  by one, go back to Step 2 and continue until the minimum-distance Euclidean criterion is satisfied, or no noticeable changes occur in the feature map.

## EXAMPLE OF KSOFM

Suppose, for instance, that the 2-dimensional input vector  $\mathbf{X}$  is presented to the three-neuron Kohonen network,

$$\mathbf{X} = \begin{bmatrix} 0.52 \\ 0.12 \end{bmatrix}$$

The initial weight vectors,  $\mathbf{W}_j$ , are given by

$$\mathbf{W}_1 = \begin{bmatrix} 0.27 \\ 0.81 \end{bmatrix}$$

$$\mathbf{W}_2 = \begin{bmatrix} 0.42 \\ 0.70 \end{bmatrix}$$

$$\mathbf{W}_3 = \begin{bmatrix} 0.43 \\ 0.21 \end{bmatrix}$$

Find the winning neuron using the Euclidean distance:

$$d_1 = \sqrt{(x_1 - w_{11})^2 + (x_2 - w_{21})^2} = \sqrt{(0.52 - 0.27)^2 + (0.12 - 0.81)^2} = 0.73$$

$$d_2 = \sqrt{(x_1 - w_{12})^2 + (x_2 - w_{22})^2} = \sqrt{(0.52 - 0.42)^2 + (0.12 - 0.70)^2} = 0.59$$

$$d_3 = \sqrt{(x_1 - w_{13})^2 + (x_2 - w_{23})^2} = \sqrt{(0.52 - 0.43)^2 + (0.12 - 0.21)^2} = 0.13$$

Neuron 3 is the winner and its weight vector  $W_3$  is updated according to the competitive learning rule:

$$\Delta w_{13} = \alpha (x_1 - w_{13}) = 0.1 (0.52 - 0.43) = 0.01$$

$$\Delta w_{23} = \alpha (x_2 - w_{23}) = 0.1 (0.12 - 0.21) = -0.01$$

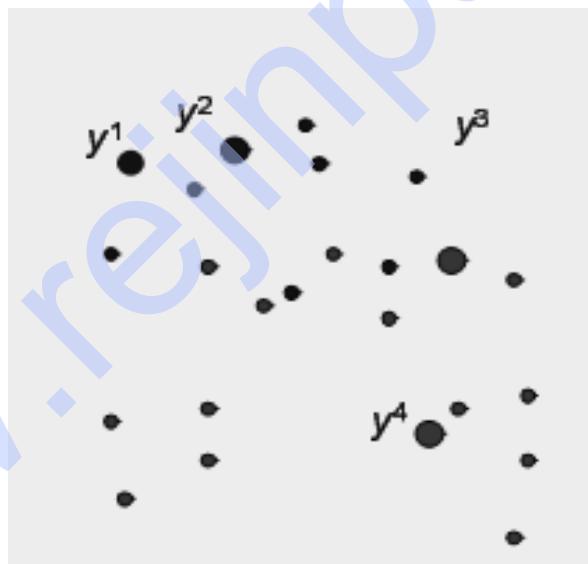
The updated weight vector  $W_3$  at iteration  $(p+1)$  is determined as:

$$W_3(p+1) = W_3(p) + \Delta W_3(p) = \begin{bmatrix} 0.43 \\ 0.21 \end{bmatrix} + \begin{bmatrix} 0.01 \\ -0.01 \end{bmatrix} = \begin{bmatrix} 0.44 \\ 0.20 \end{bmatrix}$$

The weight vector  $W_3$  of the winning neuron 3 becomes closer to the input vector  $X$  with each iteration.

# LEARNING VECTOR QUANTIZATION (LVQ)

- This is a supervised version of vector quantization. Classes are predefined and we have a set of labeled data.
- The goal is to determine a set of prototypes that best represent each class.



## BASIC SCHEME OF LVQ

**Step 1:** Initialize prototype vectors for different classes.

**Step 2:** Present a single input.

**Step 3:** Identify the closest prototype, i.e., the so-called winner.

**Step 4:** Move the winner

- closer toward the data (same class),
- away from the data (different class).

## VARIANTS OF LVQ

- LVQ 1
- LVQ 2
- LVQ 2.1
- LVQ 3

# COUNTERPROPAGATION NETWORK

- Another variant of the BPN is the **counterpropagation network** (CPN).
- Although this network uses **linear** neurons, it can learn **nonlinear** functions by means of a hidden layer of **competitive units**.
- Moreover, the network is able to learn a function and its **inverse** at the same time.

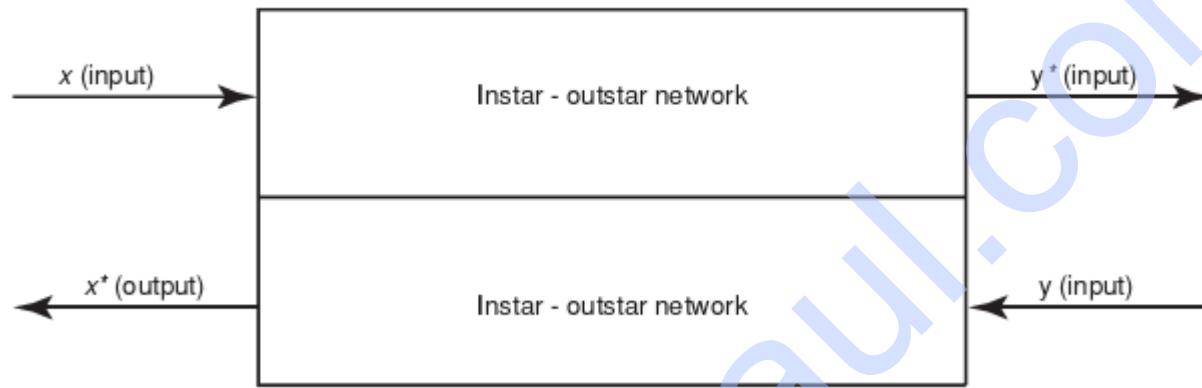
## TYPES OF COUNTERPROPAGATION NETWORK

- **FULL COUNTERPROPAGATION NET**
  - Full counterpropagation net (full CPN) efficiently represents a large number of vector pairs  $x:y$  by adaptively constructing a look-up-table.

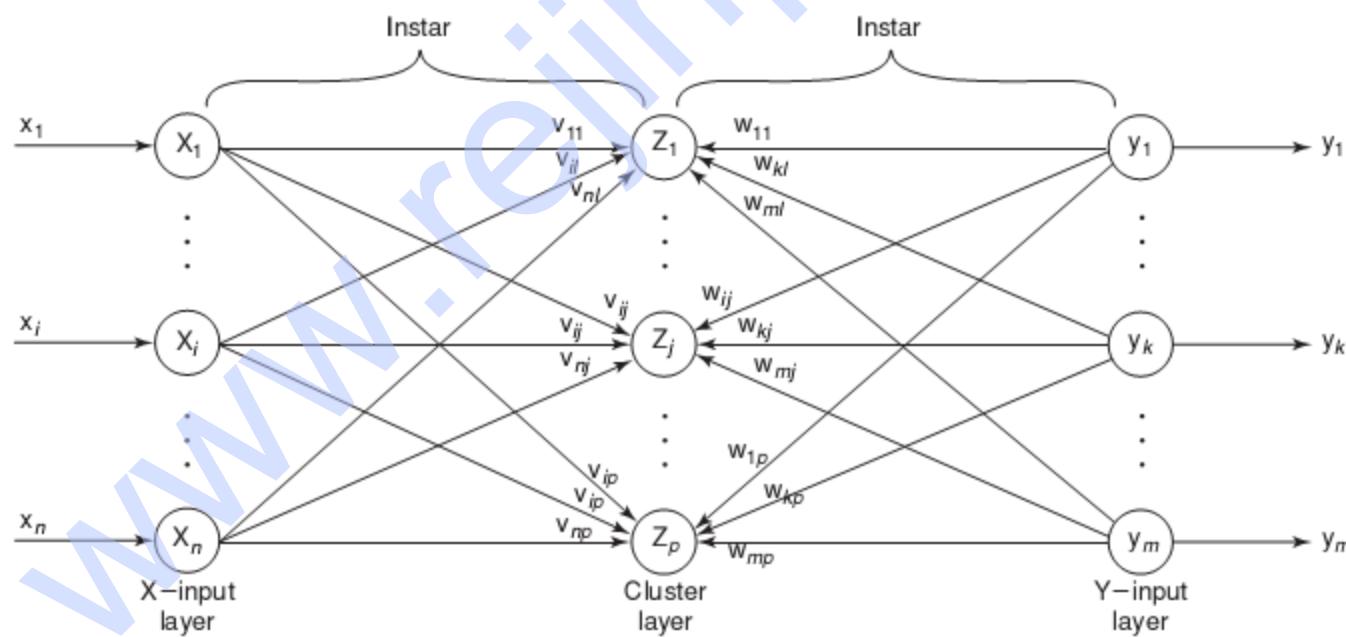
## FORWARD-ONLY COUNTERPROPAGATION NET

- A simplified version of full CPN is the forward-only CPN. The approximation of the function  $y = f(x)$  but not of  $x = f(y)$  can be performed using forward-only CPN.
- In forward-only CPN only the  $x$ -vectors are used to form the clusters on the Kohonen units.

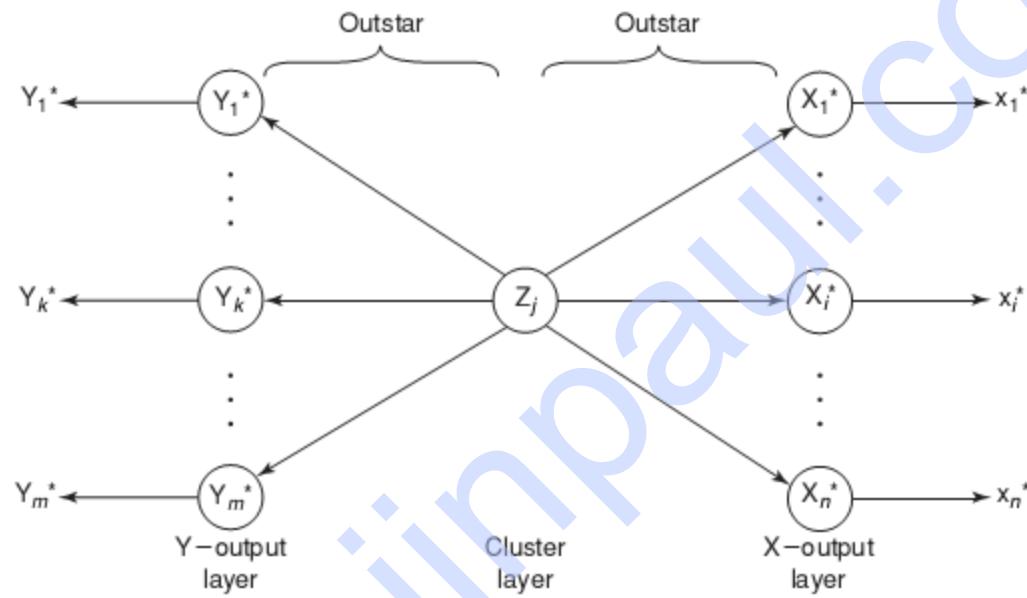
# BASIC STRUCTURE OF FULL CPN



## FIRST PHASE OF CPN



## SECOND PHASE OF CPN



## CPN LEARNING PROCESS

**The CPN learning process (general form for n input units and m output units):**

1. Randomly select a vector pair  $(x, y)$  from the training set.
2. Normalize (shrink/expand to “length” 1) the input vector  $x$  by dividing every component of  $x$  by the magnitude  $\|x\|$ , where

$$\|x\| = \sqrt{\sum_{j=1}^n x_j^2}$$

3. Initialize the input neurons with the normalized vector and compute the activation of the **linear** hidden-layer units.
4. In the hidden (competitive) layer, determine the unit W with the largest activation (the winner).
5. Adjust the connection weights between W and all N input-layer units according to the formula:

$$w_{Wn}^H(t+1) = w_{Wn}^H(t) + \alpha(x_n - w_{Wn}^H(t))$$

6. Repeat steps 1 to 5 until all training patterns have been processed once.

7. Repeat step 6 until each input pattern is consistently associated with the same competitive unit.
8. Select the first vector pair in the training set (the current pattern).
9. Repeat steps 2 to 4 (normalization, competition) for the current pattern.
10. Adjust the connection weights between the winning hidden-layer unit and all M output layer units according to the equation:

$$w_{mW}^o(t+1) = w_{mW}^o(t) + \beta(y_m - w_{mW}^o(t))$$

11. Repeat steps 9 and 10 for each vector pair in the training set.
12. Repeat steps 8 through 11 until the difference between the desired and the actual output falls below an acceptable threshold.

# COUNTERPROPAGATION NETWORK

- After the **first phase** of the training, each hidden-layer neuron is associated with a subset of input vectors.
- The training process minimized the average angle difference between the weight vectors and their associated input vectors.
- In the **second phase** of the training, we adjust the weights in the network's output layer in such a way that, for any winning hidden-layer unit, the network's output is as close as possible to the desired output for the winning unit's associated input vectors.
- The idea is that when we later use the network to compute functions, the output of the winning hidden-layer unit is 1 and the output of all other hidden-layer units is 0.

- In the first training phase, if a hidden-layer unit **does not win** for a long period of time, its weights should be set to **random** values to give that unit a chance to win subsequently.
- There is **no need for normalizing** the training output vectors.
- After the training has finished, the network maps the training vectors onto output vectors that are **close** to the desired ones. The **more** hidden units, the **better** the mapping.
- CPN can be widely used in **data compression** and **image compression** applications.

# ADAPTIVE RESONANCE THEORY (ART) NETWORK

- **Adaptive Resonance Theory** (ART) is a family of algorithms for unsupervised learning developed by Carpenter and Grossberg.
- **ART** is similar to many iterative clustering algorithms where each pattern is processed by
  - finding the "nearest" cluster (a.k.a. prototype or template) to that exemplar (desired).
  - updating that cluster to be "closer" to the exemplar.

## ARCHITECTURES OF ART NETWORK

- **ART1**, designed for binary features.
- **ART2**, designed for continuous (analog) features.
- **ARTMAP**, a supervised version of ART.

# FUNDAMENTAL ALGORITHM OF ART NETWORK

**Step 0:** Initialize the necessary parameters.

**Step 1:** Perform Steps 2–9 when stopping condition is false.

**Step 2:** Perform Steps 3–8 for each input vector.

**Step 3:**  $F_1$  layer processing is done.

**Step 4:** Perform Steps 5–7 when reset condition is true.

**Step 5:** Find the victim unit to learn the current input pattern. The victim unit is going to be the  $F_2$  unit (that is not inhibited) with the largest input.

**Step 6:**  $F_1(b)$  units combine their inputs from  $F_1(a)$  and  $F_2$ .

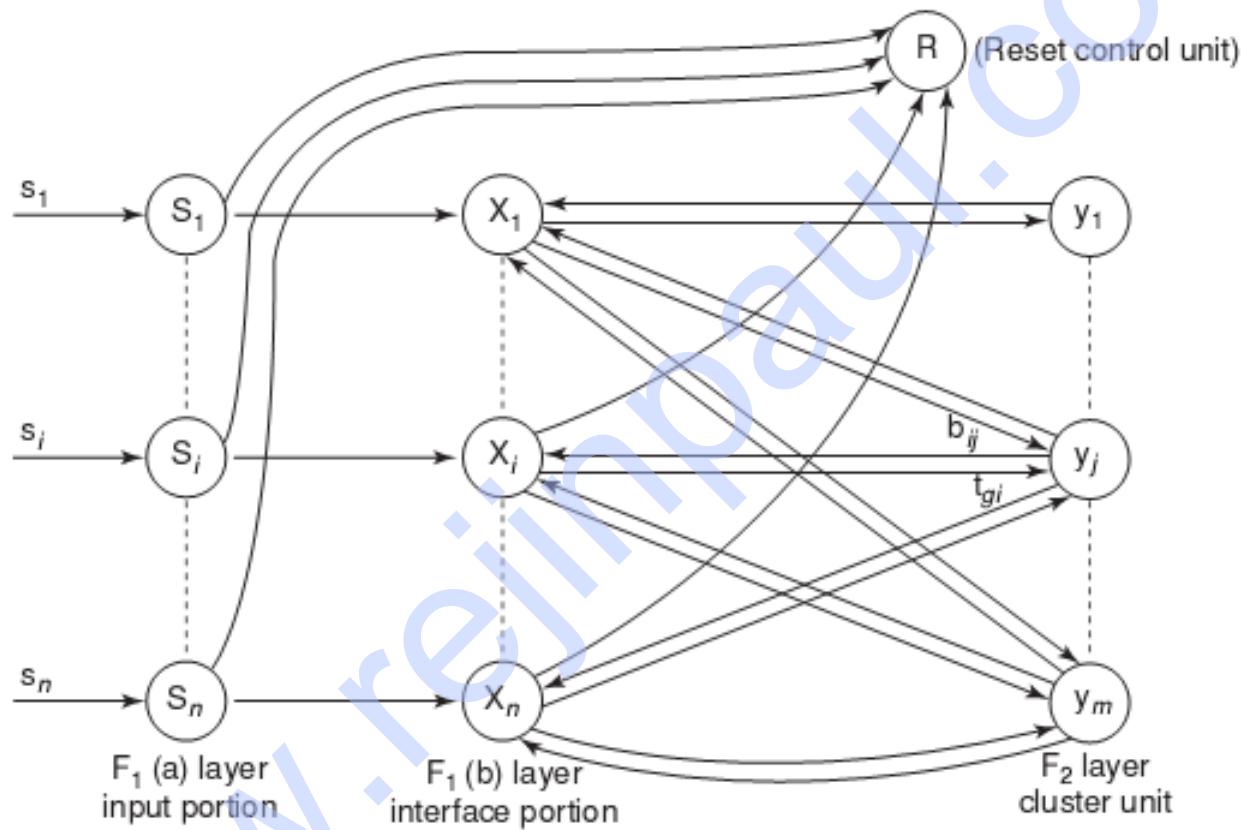
**Step 7:** Test for reset condition.

If reset is true, then the current victim unit is rejected (inhibited); go to Step 4. If reset is false, then the current victim unit is accepted for learning; go to next step (Step 8).

**Step 8:** Weight updation is performed.

**Step 9:** Test for stopping condition.

# BASIC ARCHITECTURE OF ART1



## ART1 UNITS

ART1 Network is made up of two units

➤ Computational units

- Input unit (F1 unit – input and interface).
- Cluster unit (F2 unit – output).
- Reset control unit (controls degree of similarity).

➤ Supplemental units

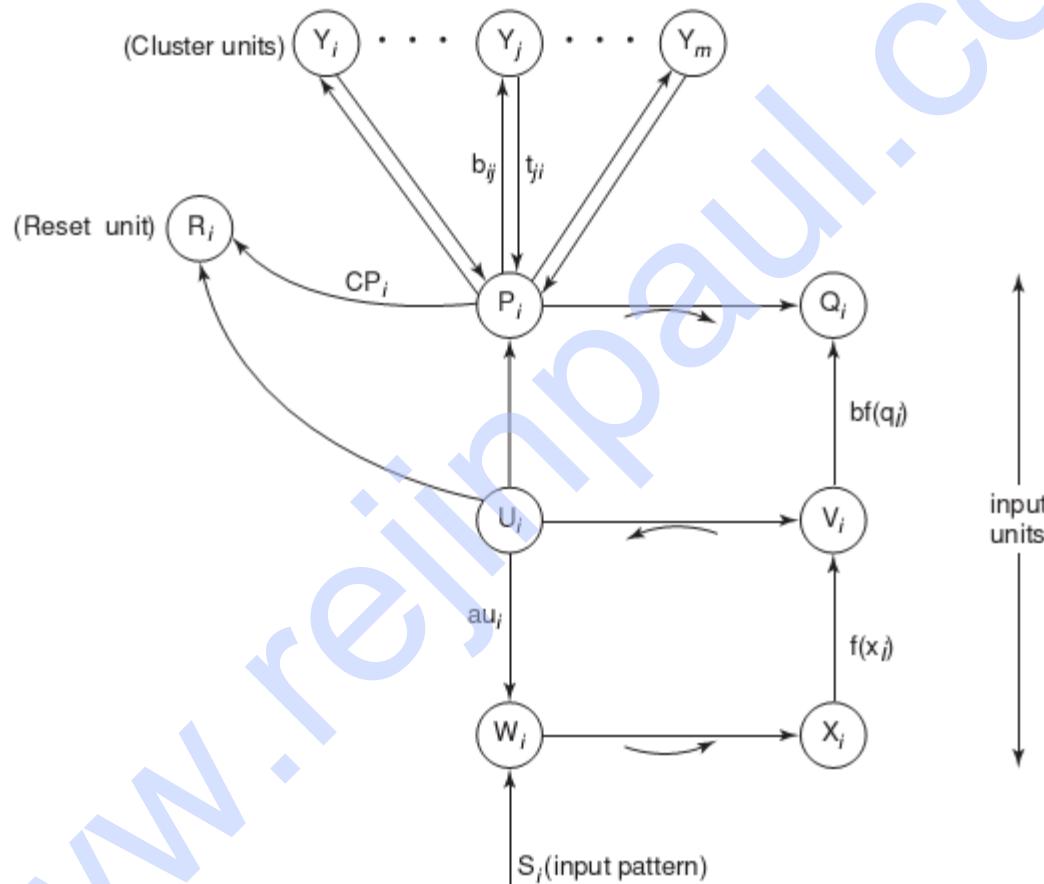
- One reset control unit.
- Two gain control units.

## ART2 NETWORK

ART was motivated by the “stability-plasticity dilemma”, a term coined by Grossberg that describes the problems endemic to competitive learning

- The network's adaptability or plasticity causes prior learning to be eroded by exposure to more recent input patterns
- ART resolves this problem by creating a new cluster every time an example is very dissimilar from the existing clusters
  - Stability: previous learning is preserved since the existing clusters are not altered and
  - Plasticity: the new example is incorporated by creating a new cluster

# BASIC ARCHITECTURE OF ART2



# ART2 ALGORITHM

Let:  $\alpha$ : positive number  $\alpha \leq 1/\sqrt{N_{EX}}$   
 $\beta$ : small positive number  
 $\theta$ : normalization parameter  $0 < \theta < 1/\sqrt{N_{EX}}$   
 $\rho$ : vigilance parameter  $0 \leq \rho < 1$

0. For each example  $x^{(n)}$  in the database
  - 0a. Normalize  $x^{(n)}$  to have magnitude 1
  - 0b. Replace coordinates of  $x^{(n)}$  that are  $<\theta$  by 0 (remove small noise signals)
  - 0c. Re-normalize  $x^{(n)}$
1. Start with no prototype vectors (clusters)
2. Perform iterations until no example causes any change. At this point quit because stability has been achieved. For each iteration, choose the next example  $x^{(n)}$  in cyclic order
3. Find the prototype  $w_k$  (cluster) not yet tried during this iteration that maximizes  $w_k^T x^{(n)}$   
 (inner product of two normal vectors is equal to the cosine of the angle between the vectors)
4. Test whether  $w_k$  is sufficiently similar to  $x^{(n)}$

$$w_k^T x^{(n)} \geq \alpha \sum_{j=1}^{N_{EX}} x^{(n)}(j)$$

- 4a. If not then
  - 4a1. Make a new cluster with prototype set to  $x^{(n)}$
  - 4a2. End this iteration and return to step 2 for the next example
- 4b. If sufficiently similar, then test for vigilance acceptability

$$w_k^T x^{(n)} \geq \rho$$

- 4b1. If acceptable then  $x^{(n)}$  belongs to  $w_k$ . Modify  $w_k$  to be more like  $x^{(n)}$

$$w_k = \frac{(1-\beta)w_k + \beta x^{(n)}}{\|(1-\beta)w_k + \beta x^{(n)}\|}$$

and go to step 2 for the next iteration with the next example

- 4b2. If not acceptable, then make a new cluster with prototype set to  $x^{(n)}$

# APPLICATIONS OF NEURAL NETWORKS

- in signature analysis:
  - as a mechanism for comparing signatures made (e.g. in a bank) with those stored. This is one of the first large-scale applications of neural networks in the USA, and is also one of the first to use a neural network chip.
- in process control:
  - there are clearly applications to be made here: most processes cannot be determined as computable algorithms.
- in monitoring:
  - networks have been used to monitor
    - the state of aircraft engines. By monitoring vibration levels and sound, early warning of engine problems can be given.
    - British Rail have also been testing a similar application monitoring diesel engines.
  - Pen PC's
    - PC's where one can write on a tablet, and the writing will be recognised and translated into (ASCII) text.
  - Speech and Vision recognition systems
    - Not new, but Neural Networks are becoming increasingly part of such systems. They are used as a system component, in conjunction with traditional computers.

## NEURAL NETWORK TOOLBOX

- The Matlab neural network toolbox provides a complete set of functions and a graphical user interface for the design, implementation, visualization, and simulation of neural networks.
- It supports the most commonly used supervised and unsupervised network architectures and a comprehensive set of training and learning functions.

## KEY FEATURES

- Graphical user interface (GUI) for creating, training, and simulating your neural networks.
- Support for the most commonly used supervised and unsupervised network architectures.
- A comprehensive set of training and learning functions.
- A suite of Simulink blocks, as well as documentation and demonstrations of control system applications.
- Automatic generation of Simulink models from neural network objects.
- Routines for improving generalization.

# GENERAL CREATION OF NETWORK

net = network

net=

network(numInputs,numLayers,biasConnect,inputConnect,layerConnec,  
outputConnect,targetConnect)

## Description

NETWORK creates new custom networks. It is used to create networks that are then customized by functions such as NEWP, NEWLIN, NEWFF, etc.

**NETWORK takes these optional arguments (shown with default values):**

numInputs - Number of inputs, 0.

numLayers - Number of layers, 0.

biasConnect - numLayers-by-1 Boolean vector, zeros.

inputConnect - numLayers-by-numInputs Boolean matrix, zeros.

layerConnect - numLayers-by-numLayers Boolean matrix, zeros.

outputConnect - 1-by-numLayers Boolean vector, zeros.

targetConnect - 1-by-numLayers Boolean vector, zeros, and returns,

NET - New network with the given property values.

## TRAIN AND ADAPT

1. **Incremental training** : updating the weights after the presentation of each single training sample.
2. **Batch training** : updating the weights after each presenting the complete data set.

When using adapt, both incremental and batch training can be used . When using train on the other hand, only batch training will be used, regardless of the format of the data. The big plus of train is that it gives you a lot more choice in training functions (gradient descent, gradient descent w/ momentum, Levenberg-Marquardt, etc.) which are implemented very efficiently .

The difference between train and adapt: the difference between passes and epochs. When using adapt, the property that determines how many times the complete training data set is used for training the network is called net.adaptParam.passes. Fair enough. But, when using train, the exact same property is now called net.trainParam.epochs.

```
>> net.trainFcn = 'traingdm';
>> net.trainParam.epochs = 1000;
>> net.adaptFcn = 'adaptwb';
>> net.adaptParam.passes = 10;
```

## TRAINING FUNCTIONS

There are several types of training functions:

1. Supported training functions,
2. Supported learning functions,
3. Transfer functions
4. Transfer derivative functions,
5. Weight and bias initialize functions,
6. Weight derivative functions.

# SUPPORTED TRAINING FUNCTIONS

**trainb**

**trainbfg**

**trainbr**

**trainc**

**traincgb**

**traincfg**

**traincgp**

**traingd**

**traingda**

**traingdm**

**traingdx**

**trainlm**

**trainoss**

**trainr**

**trainrp**

**trains**

**trainscg**

- Batch training with weight and bias learning rules
- BFGS quasi-Newton backpropagation
- Bayesian regularization
- Cyclical order incremental update
- Powell-Beale conjugate gradient backpropagation
- Fletcher-Powell conjugate gradient backpropagation
- Polak-Ribiere conjugate gradient backpropagation
- Gradient descent backpropagation
- Gradient descent with adaptive learning rate backpropagation
- Gradient descent with momentum backpropagation
- Gradient descent with momentum & adaptive linear ackpropagation
- Levenberg-Marquardt backpropagation
- One step secant backpropagations
- Random order incremental update
- Resilient backpropagation (Rprop)
- Sequential order incremental update
- Scaled conjugate gradient backpropagation

# SUPPORTED LEARNING FUNCTIONS

**learncon**

– Conscience bias learning function

**learngd**

– Gradient descent weight/bias learning function

**learngdm**

– Gradient descent with momentum weight/bias learning function

**learnh**

– Hebb weight learning function

**learnhd**

– Hebb with decay weight learning rule

**learnis**

– Instar weight learning function

**learnk**

– Kohonen weight learning function

**learnlv1**

– LVQ1 weight learning function

**learnlv2**

– LVQ2 weight learning function

**learnos**

– Outstar weight learning function

**learnp**

– Perceptron weight and bias learning function

**learnpn**

– Normalized perceptron weight and bias learning function

**learnsom**

– Self-organizing map weight learning function

**learnwh**

– Widrow-Hoff weight and bias learning rule

# TRANSFER FUNCTIONS

**compet**

- Competitive transfer function.

**hardlim**

- Hard limit transfer function.

**hardlims**

- Symmetric hard limit transfer function.

**logsig**

- Log sigmoid transfer function.

**poslin**

- Positive linear transfer function.

**purelin**

- Linear transfer function.

**radbas**

- Radial basis transfer function.

**satlin**

- Saturating linear transfer function.

**satlins**

- Symmetric saturating linear transfer function.

**softmax**

- Soft max transfer function.

**tansig**

- Hyperbolic tangent sigmoid transfer function.

**tribas**

- Triangular basis transfer function.

# TRANSFER DERIVATIVE FUNCTIONS

**Dhardlim**

- Hard limit transfer derivative function.

**dhardlms**

- Symmetric hard limit transfer derivative function

**dlogsig**

- Log sigmoid transfer derivative function.

**dposlin**

- Positive linear transfer derivative function.

**dpurelin**

- Hard limit transfer derivative function.

**dradbas**

- Radial basis transfer derivative function.

**dsatlin**

- Saturating linear transfer derivative function.

**dsatlins**

- Symmetric saturating linear transfer derivative function.

**dtansig**

- Hyperbolic tangent sigmoid transfer derivative function.

**dtribas**

- Triangular basis transfer derivative function.

# WEIGHT AND BIAS INITIALIZATION FUNCTIONS

**initcon**

- Conscience bias initialization function.

**initzero**

- Zero weight/bias initialization function.

**midpoint**

- Midpoint weight initialization function.

**randnc**

- Normalized column weight initialization function.

**randnr**

- Normalized row weight initialization function.

**rands**

- Symmetric random weight/bias initialization function.

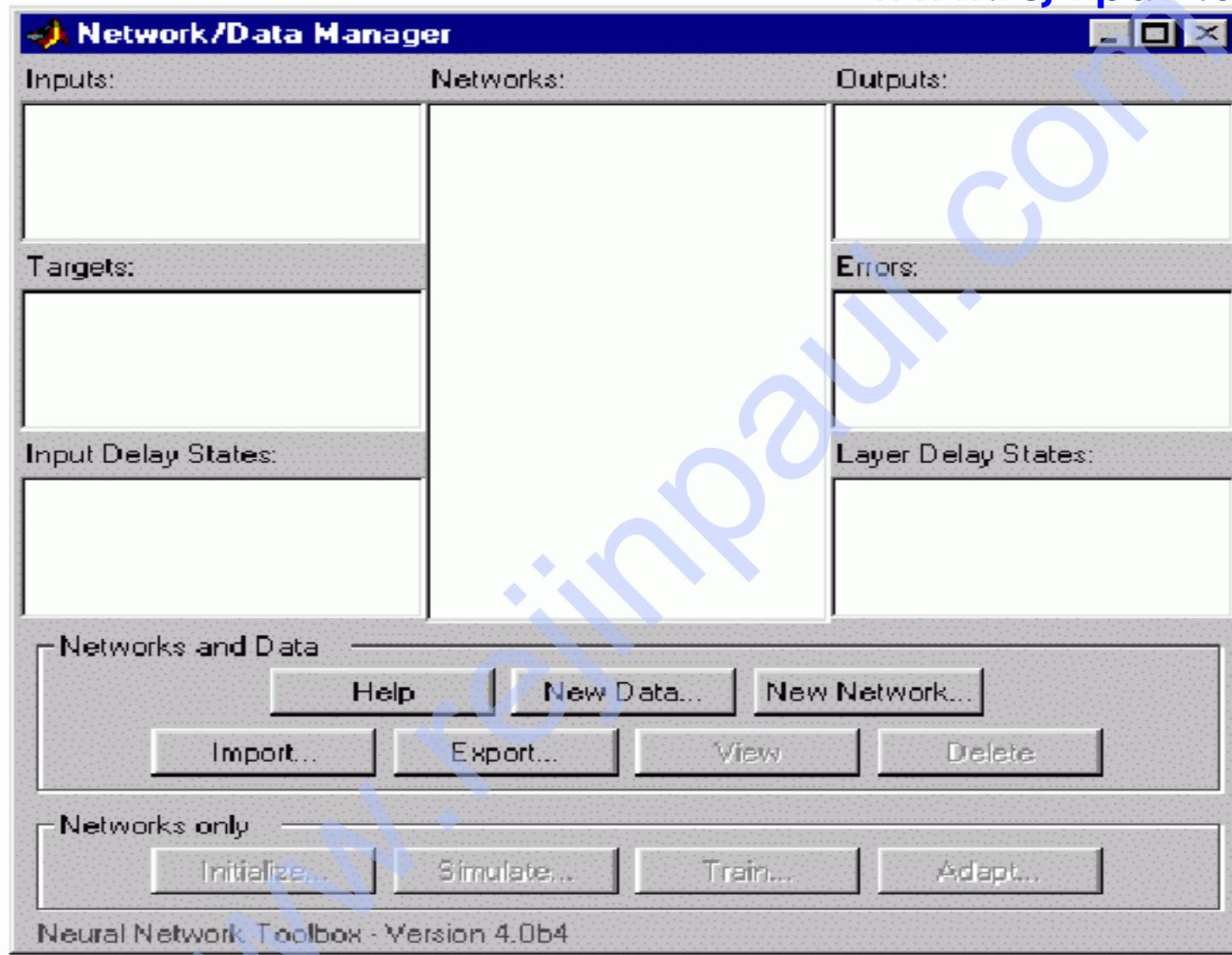
# WEIGHT DERIVATIVE FUNCTIONS

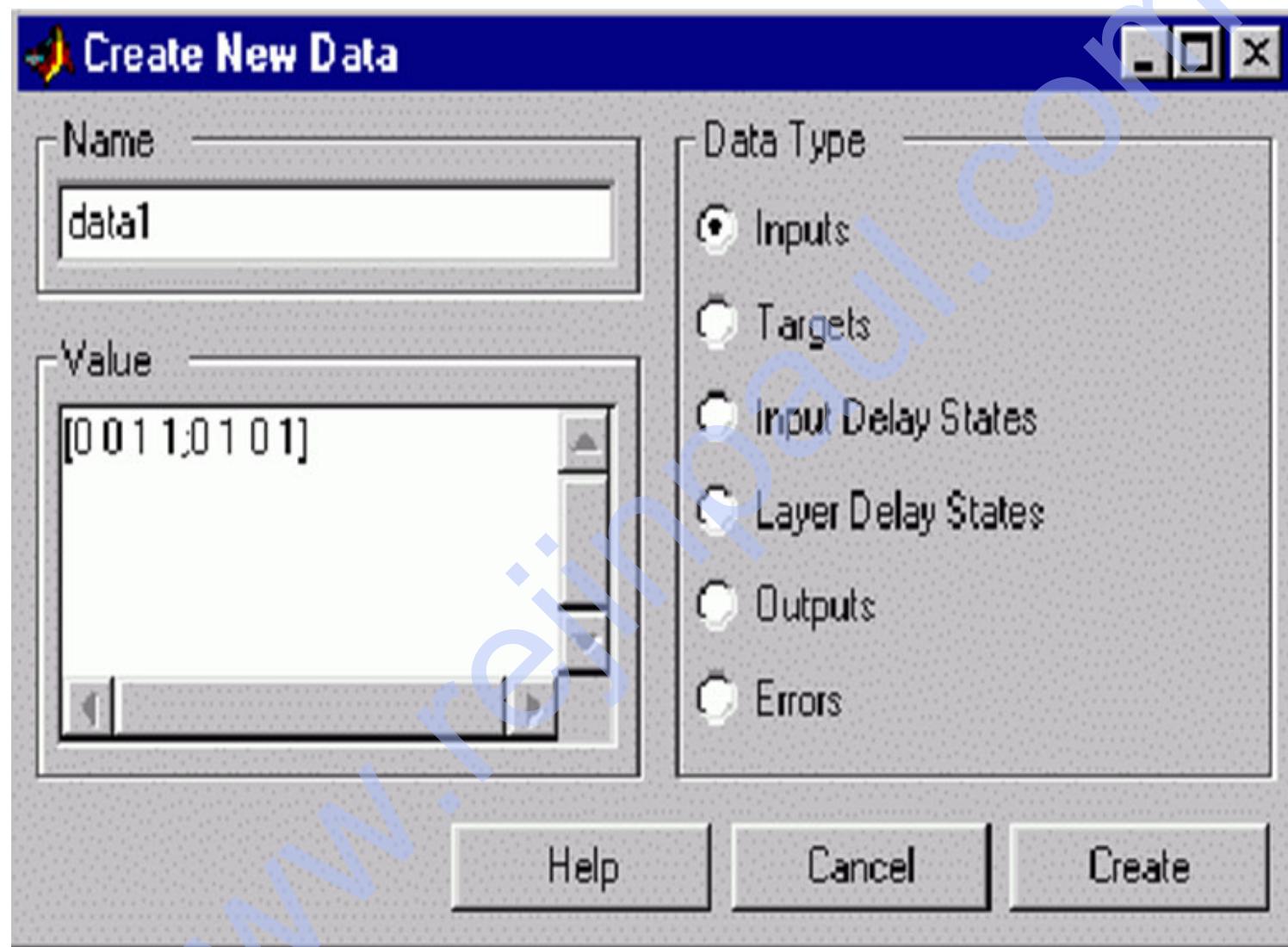
**ddotprod**

- Dot product weight derivative function.

## NEURAL NETWORK TOOLBOX GUI

1. The graphical user interface (GUI) is designed to be simple and user friendly. This tool lets you import potentially large and complex data sets.
2. The GUI also enables you to create, initialize, train, simulate, and manage the networks. It has the GUI Network/Data Manager window.
3. The window has its own work area, separate from the more familiar command line workspace. Thus, when using the GUI, one might "export" the GUI results to the (command line) workspace. Similarly to "import" results from the command line workspace to the GUI.
4. Once the Network/Data Manager is up and running, create a network, view it, train it, simulate it and export the final results to the workspace. Similarly, import data from the workspace for use in the GUI.





 **Create New Network**

Network Name: ANDNet

Network Type: Perceptron

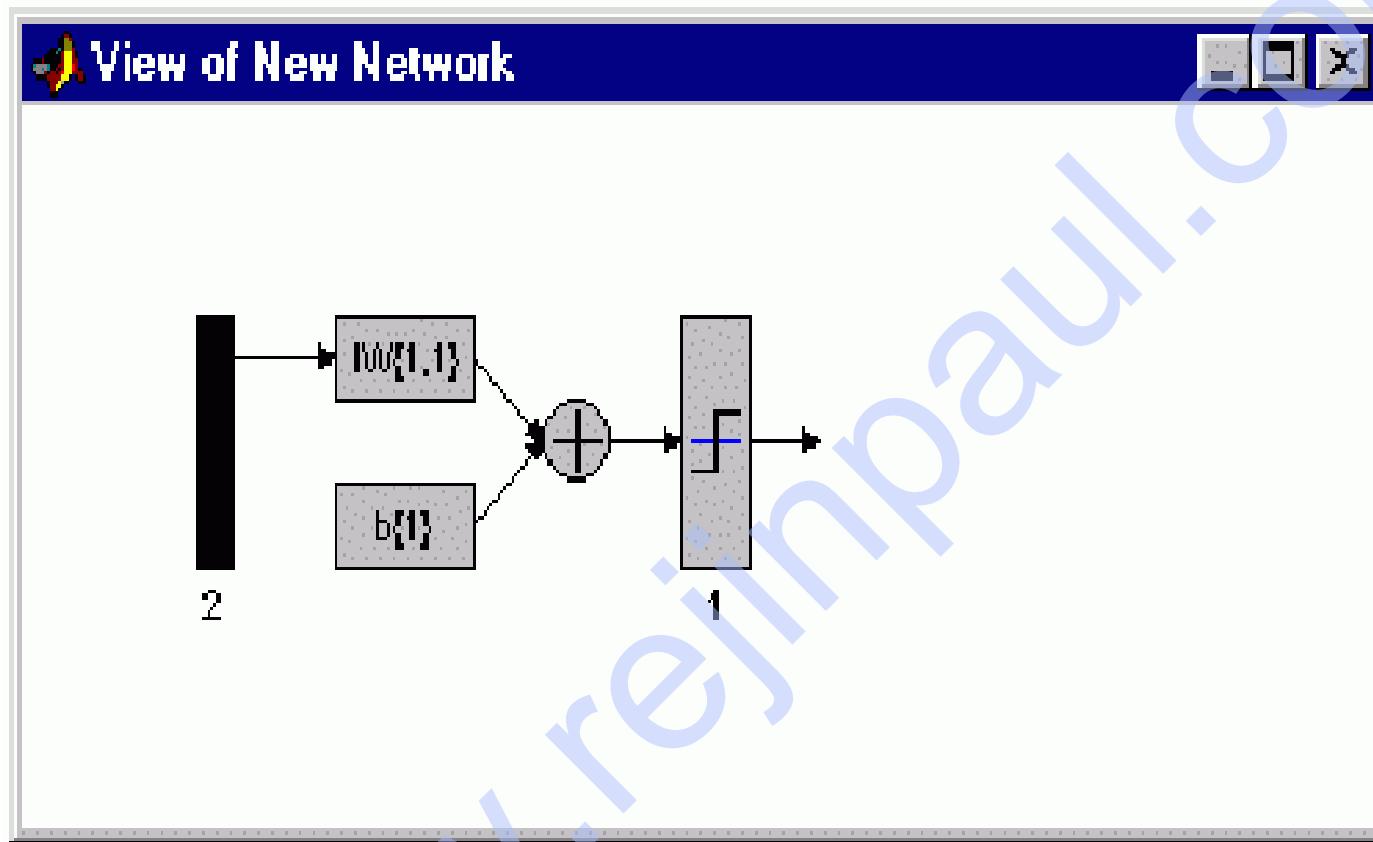
Input ranges: [0 1;0 1] Get from input:

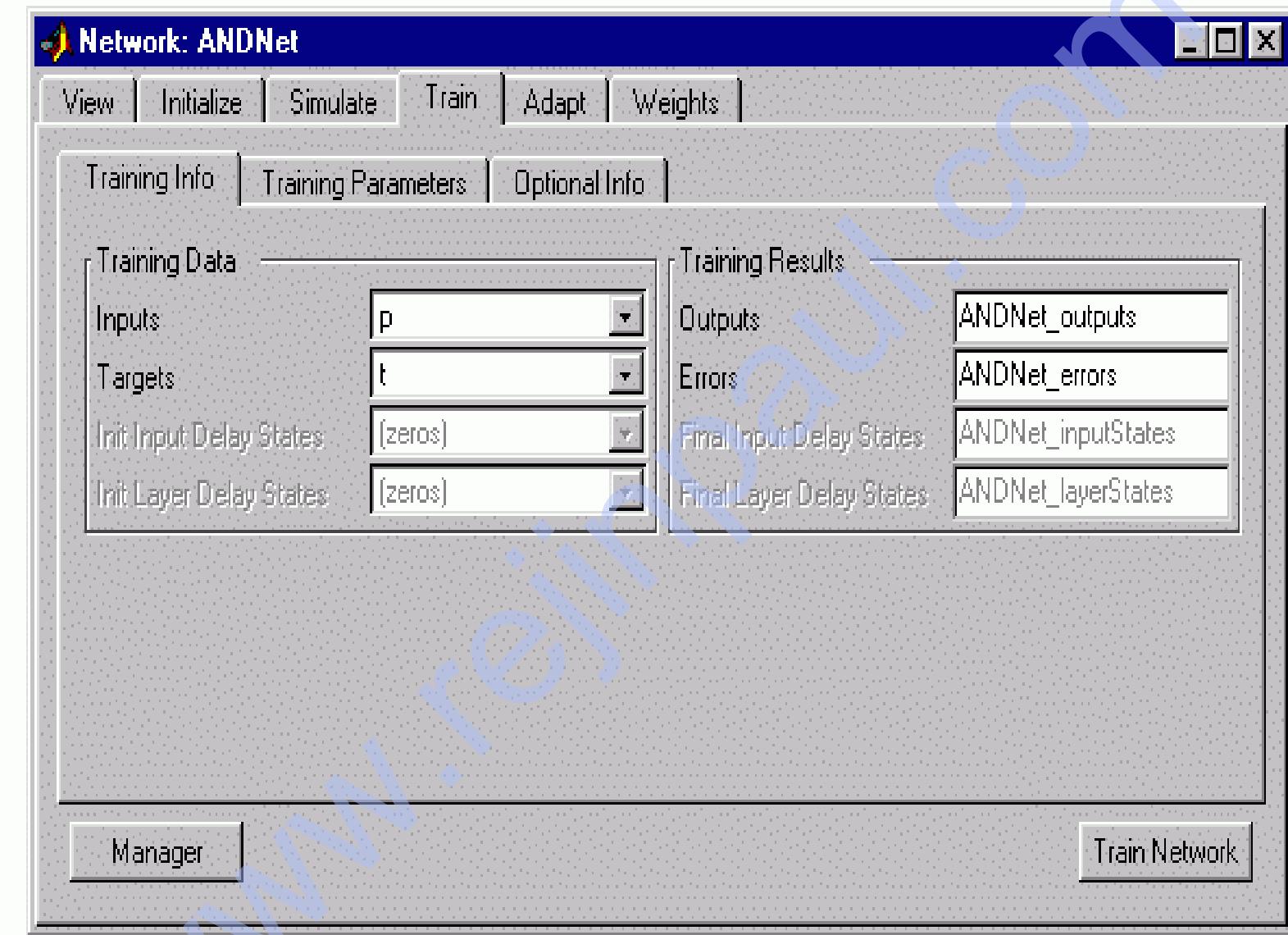
Number of neurons: 1

Transfer function: HARDLIM

Learning function: LEARNP

**View** **Defaults** **Cancel** **Create**





A graphical user interface can thus be used to

1. Create network,
2. Create data,
3. Train the networks,
4. Export the networks,
5. Export the data to the command line workspace.

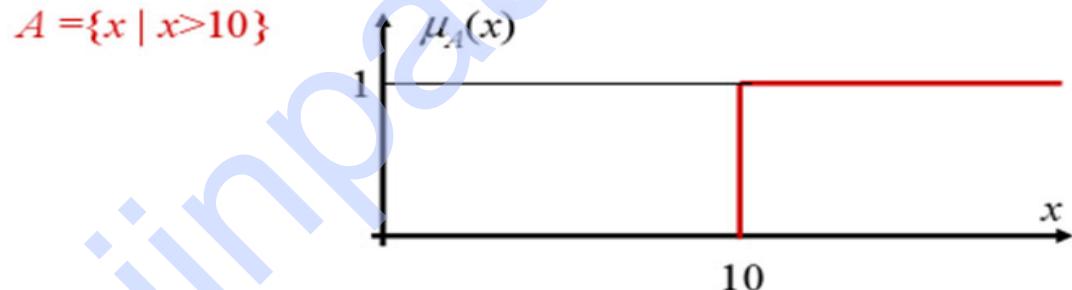
# UNIT-III FUZZY LOGIC

[www.rejinpaul.com](http://www.rejinpaul.com)

## MEMBERSHIP FUNCTIONS

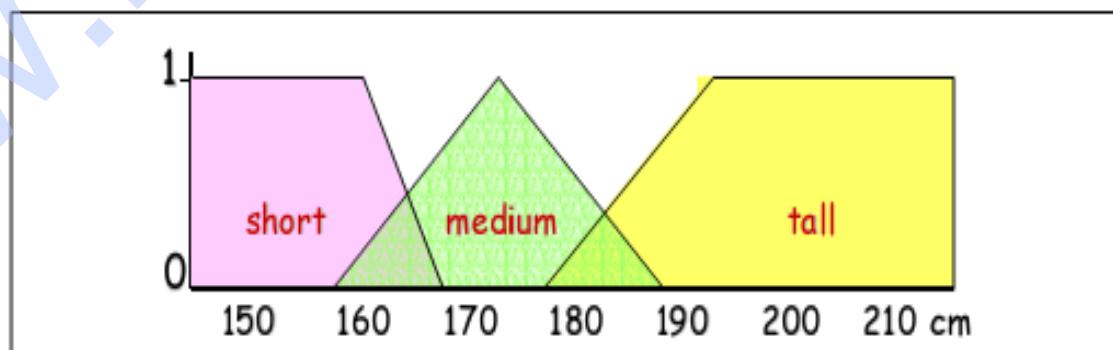
### CRISP MEMBERSHIP FUNCTIONS

- Crisp membership functions ( $\mu$ ) are either one or zero.
- Consider the example: Numbers greater than 10. The membership curve for the set A is given by



### REPRESENTING A DOMAIN IN FUZZY LOGIC

Fuzzy sets (men's height):



# FUZZY MEMBERSHIP FUNCTIONS

- Categorization of element  $x$  into a set  $A$  described through a membership function  $\mu_A(x)$
- Formally, given a fuzzy set  $A$  of universe  $X$

$\mu_A(x): X \rightarrow [0,1]$ , where

$\mu_A(x) = 1$  if  $x$  is totally in  $A$

$\mu_A(x) = 0$  if  $x$  is totally not in  $A$

$0 < \mu_A(x) < 1$  if  $x$  is partially in  $A$

$$\mu_{\text{Tall}}(200) = 1$$

$$\mu_{\text{Tall}}(160) = 0$$

$$0 < \mu_{\text{Tall}}(180) < 1$$

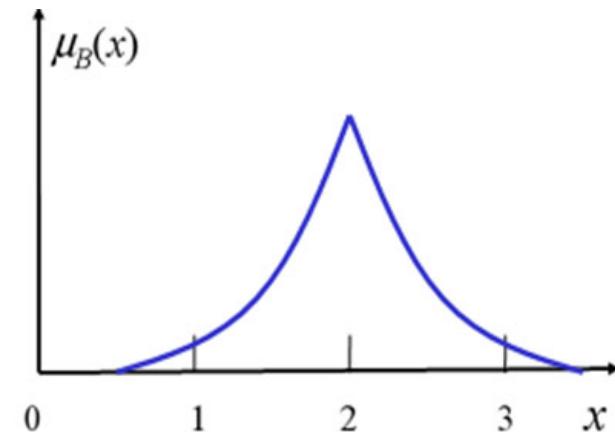
- (Discrete) Fuzzy set  $A$  is represented as:

$$A = \{\mu_A(x_1)/x_1, \mu_A(x_2)/x_2, \dots, \mu_A(x_n)/x_n\}$$

$$\text{Tall} = \{0/160, 0.2/170, 0.8/180, 1/190\}$$

The set  $B$  of numbers approaching 2 can be represented by the membership function

$$\mu_B(x) = e^{-|x-2|}$$



## FUZZINESS vs PROBABILITY

- When first exposed to fuzzy logic, humans associate **membership functions** with **density functions**.
- This is not so, since:
  - Probability density is an **abstraction from empirical frequency**.  
⇒ an aggregate property.
    - **how often** events occur in different ways.
    - ways that are quite **crisp** and **mutually exclusive** after occurrence.
  - Fuzzy relations, by contrast, **are properties of single events** that **are always there**, and not different from occurrence to occurrence.

# LINGUISTIC VARIABLE

- A **linguistic variable** associates words or sentences with a measure of **belief functions**, also called **membership function**.
- The set of values that it can take is called **term set**.
- Each value in the set is a **fuzzy variable** defined over a **base variable**.
- The base variable defines the **Universe of discourse** for all the fuzzy variables in the term set.

A **linguistic variable** is a quintuple  $[X, T(X), U, G, M]$  where

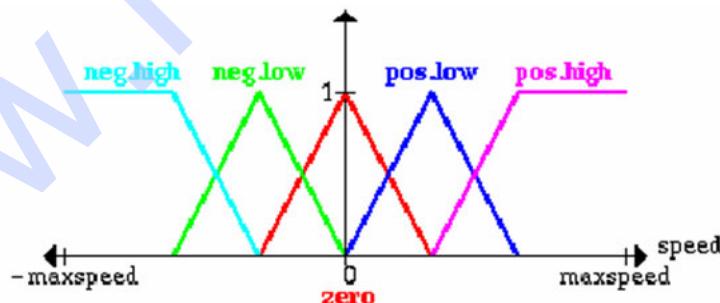
- $X$  is the name of the variable,
- $T(X)$  is the term set, i.e. the set of names of linguistic values of  $X$ ,
- $U$  is the universe of discourse,
- $G$  is the grammar to generate the names and
- $M$  is a set of semantic rules for associating each  $X$  with its meaning.

# LINGUISTIC VARIABLE

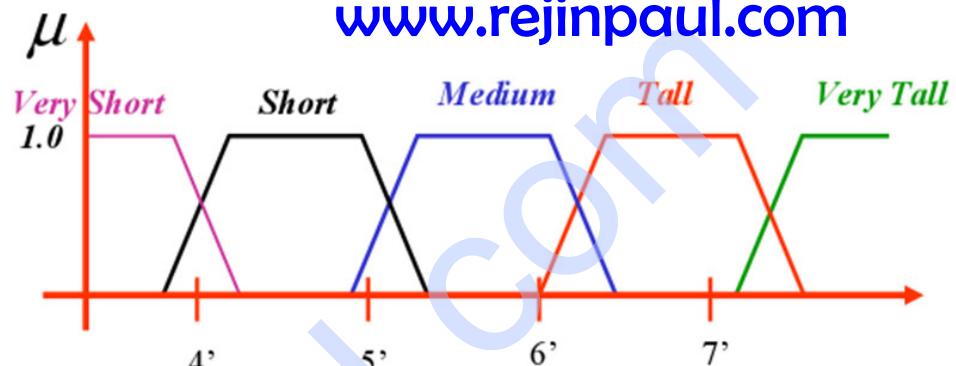
- Let  $x$  be a linguistic variable with the label "speed".
- Terms of  $x$ , which are fuzzy sets, could be "positive low", "negative high" from the term set  $T$ :

$$T = \{PositiveHigh, PositiveLow, NegativeLow, \\ NegativeHigh, Zero\}$$

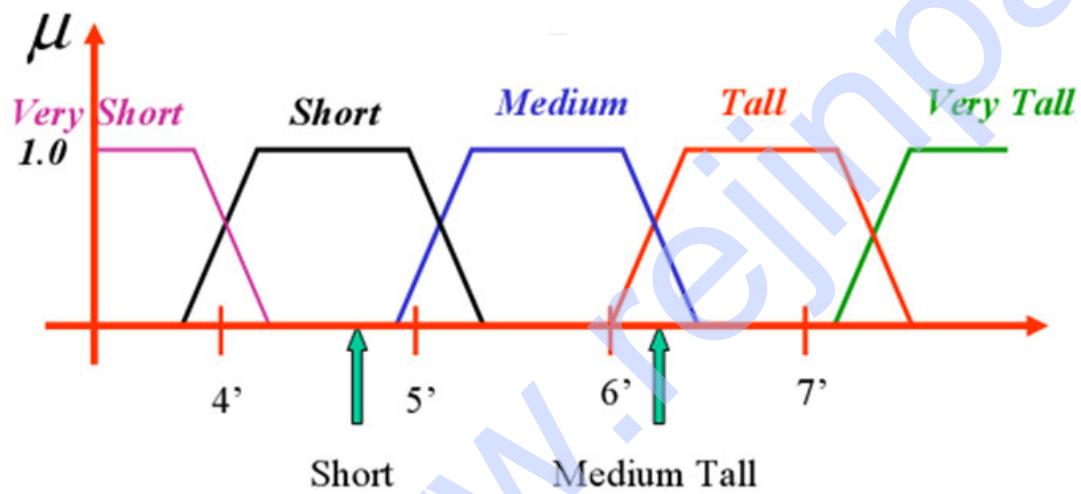
- Each term is a fuzzy variable defined on the base variable which might be the scale of all relevant velocities.



# MEMBERSHIP FUNCTIONS



$$\mu = [\mu_{vs}, \mu_s, \mu_m, \mu_t, \mu_{vt}]$$

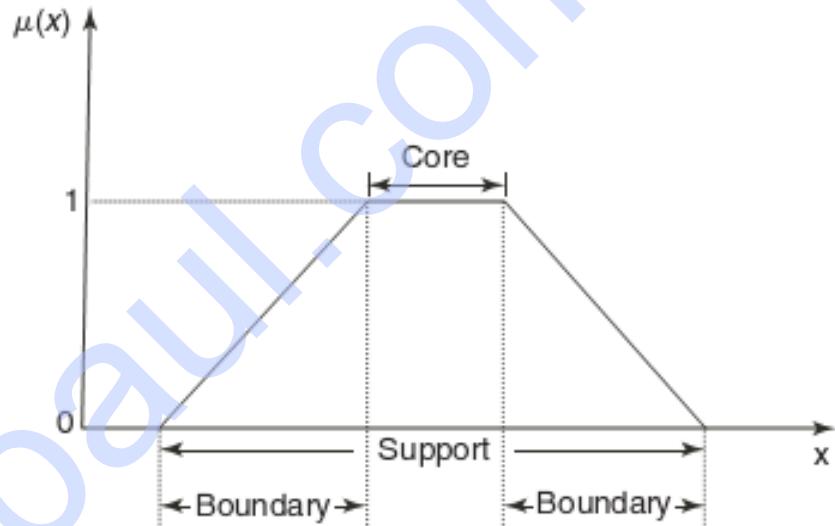


$$\mu = [0, 1, 0, 0, 0]$$

$$\mu = [0, 0, 0.5, 0.5, 0]$$

# FEATURES OF MEMBERSHIP FUNCTIONS

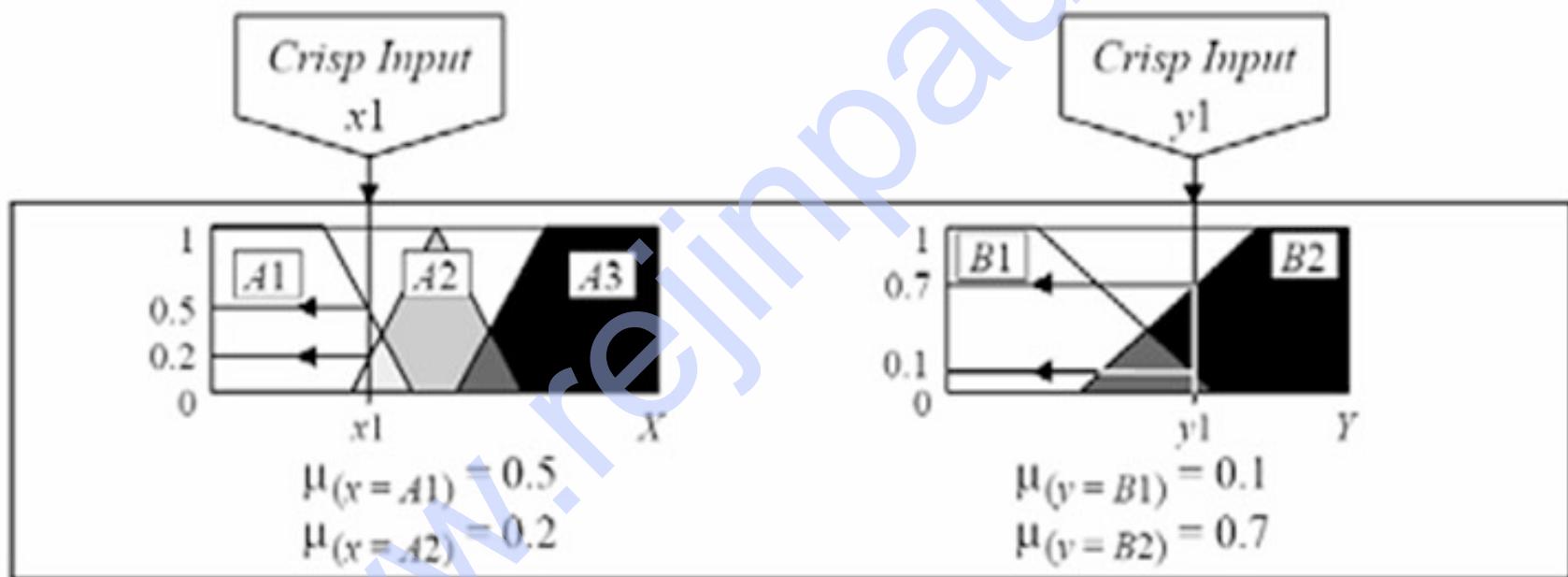
- CORE:  $\mu_{\tilde{A}}(x) = 1$
- SUPPORT:  $\mu_{\tilde{A}}(x) > 0$
- BOUNDARY:  $0 < \mu_{\tilde{A}}(x) < 1$



## FUZZIFICATION

- Fuzzifier converts a crisp input into a fuzzy variable.
- Definition of the membership functions must
  - reflects the designer's knowledge
  - provides smooth transition between member and nonmembers of a fuzzy set
  - simple to calculate
- Typical shapes of the membership function are Gaussian, trapezoidal and triangular.

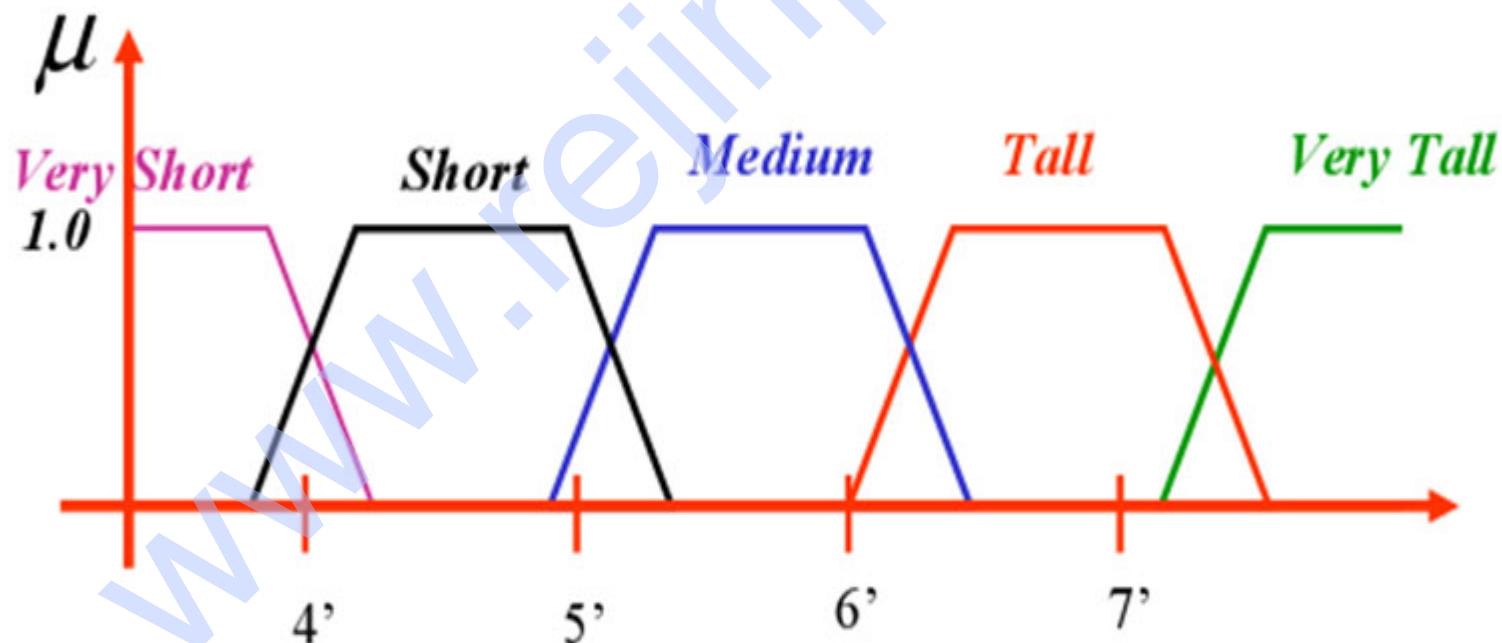
- Use crisp inputs from the user.
- Determine membership values for all the relevant classes (i.e., in right Universe of Discourse).

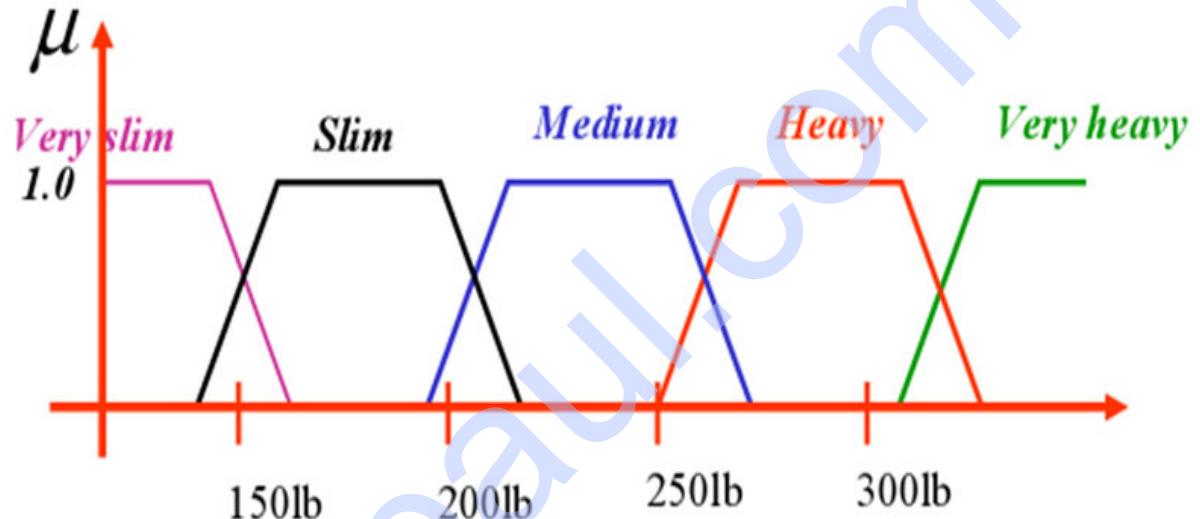


## EXAMPLE - FUZZIFICATION

- Assume we want to evaluate the health of a person based on his height and weight.
- The input variables are the crisp numbers of the person's height and weight.
- Fuzzification is a process by which the numbers are changes into linguistic words

### FUZZIFICATION OF HEIGHT





## METHODS OF MEMBERSHIP VALUE ASSIGNMENT

The various methods of assigning membership values are:

- Intuition,
- Inference,
- Rank ordering,
- Angular fuzzy sets,
- Neural networks,
- Genetic algorithm,
- Inductive reasoning.

# DEFUZZIFICATION

- Defuzzification is a mapping process from a space of fuzzy control actions defined over an output universe of discourse into a space of crisp (nonfuzzy) control actions.
- Defuzzification is a process of converting output fuzzy variable into a unique number.
- Defuzzification process has the capability to reduce a fuzzy set into a crisp single-valued quantity or into a crisp set; to convert a fuzzy matrix into a crisp matrix; or to convert a fuzzy number into a crisp number.

## LAMBDA CUT FOR FUZZY SETS

Consider a fuzzy set  $\underline{A}$ . The set  $A_\lambda$  ( $0 < \lambda < 1$ ) called the lambda ( $\lambda$ )-cut (or alpha [ $\alpha$ ]-cut) set is a crisp set of the fuzzy set and is defined as follows:

$$A_\lambda = \{x \mid \mu_{\underline{A}}(x) \geq \lambda\}; \quad \lambda \in [0, 1]$$

The properties of  $\lambda$ -cut sets are as follows:

1.  $(\underline{A} \cup \underline{B})_\lambda = A_\lambda \cup B_\lambda$
2.  $(\underline{A} \cap \underline{B})_\lambda = A_\lambda \cap B_\lambda$
3.  $(\bar{\underline{A}})_\lambda \neq (\bar{A}_\lambda)$  except when  $\lambda = 0.5$
4. For any  $\lambda \leq \beta$ , where  $0 \leq \beta \leq 1$ , it is true that  $A_\beta \subseteq A_\lambda$ , where  $A_0 = \mathcal{X}$ .

## LAMBDA CUT FOR FUZZY RELATIONS

Let  $\tilde{R}$  be a fuzzy relation where each row of the relational matrix is considered a fuzzy set. The  $j$ th row in a fuzzy relation matrix  $\tilde{R}$  denotes a discrete membership function for a fuzzy set  $\tilde{R}_j$ . A fuzzy relation can be converted into a crisp relation in the following manner:

$$R_\lambda = \{(x, y) | \mu_{\tilde{R}}(x, y) \geq \lambda\}$$

where  $R_\lambda$  is a  $\lambda$ -cut relation of the fuzzy relation  $\tilde{R}$ .

For two fuzzy relations  $\tilde{R}$  and  $\tilde{S}$  the following properties should hold:

1.  $(\tilde{R} \cup \tilde{S})_\lambda = R_\lambda \cup S_\lambda$
2.  $(\tilde{R} \cap \tilde{S})_\lambda = R_\lambda \cap S_\lambda$
3.  $(\tilde{R})_\lambda \neq (\tilde{R}_\lambda)$  except when  $\lambda = 0.5$
4. For any  $\lambda \leq \beta$ , where  $0 \leq \beta \leq 1$ , it is true that  $R_\beta \subseteq R_\lambda$ .

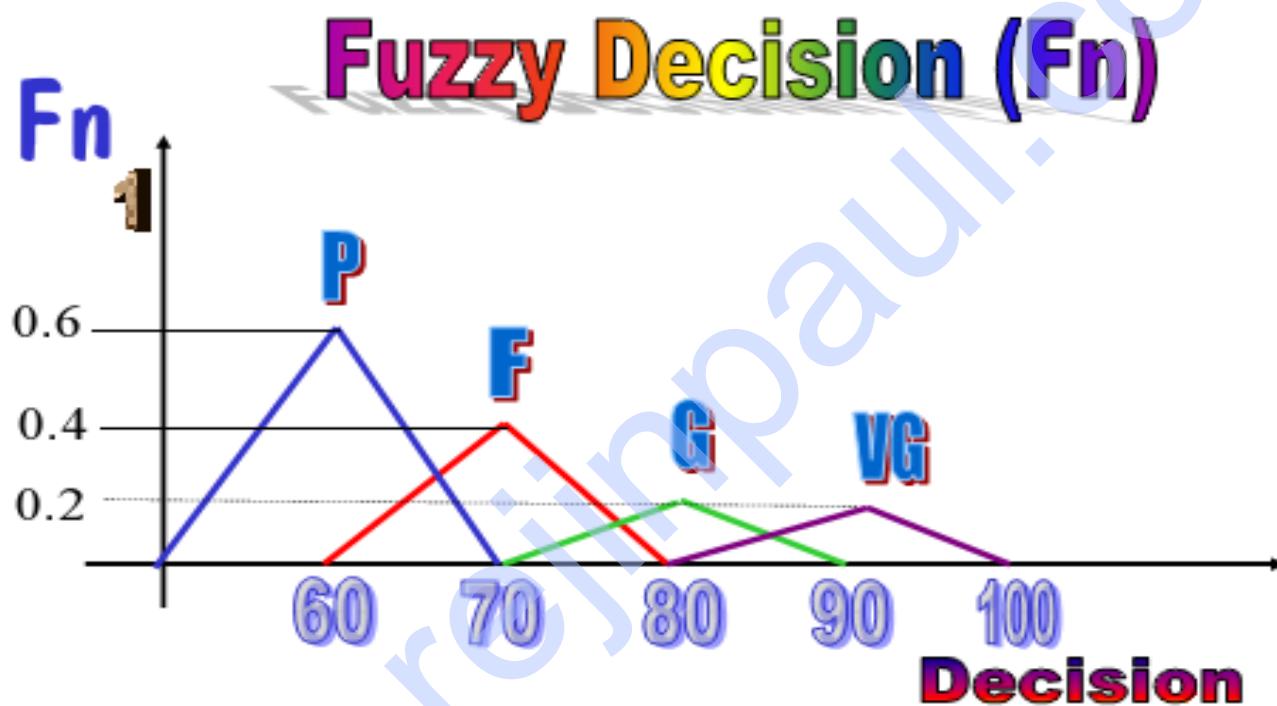
# METHODS OF DEFUZZIFICATION

Defuzzification is the process of conversion of a fuzzy quantity into a precise quantity.

Defuzzification methods include:

- Max-membership principle,
- Centroid method,
- Weighted average method,
- Mean-max membership,
- Center of sums,
- Center of largest area,
- First of maxima, last of maxima.

## FUZZY DECISION

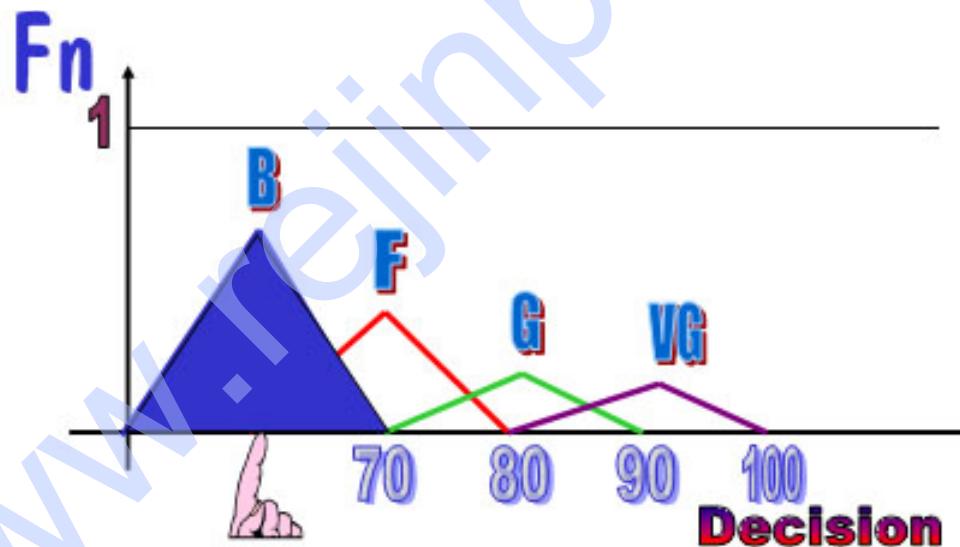


$$F_n = \{P, F, G, VG, E\}$$

$$F_n = \{0.6, 0.4, 0.2, 0.2, 0\}$$

## MAX MEMBERSHIP METHOD

- Fuzzy set with the largest membership value is selected.
- Fuzzy decision:  $F_n = \{P, F, G, VG, E\}$
- $F_n = \{0.6, 0.4, 0.2, 0.2, 0\}$
- Final decision (FD) = Poor Student
- If two decisions have same membership max, use the average of the two.



## CENTROID METHOD

This method is also known as center-of-mass, center-of-area, or center-of-gravity method. It is the most commonly used defuzzification method. The defuzzified output  $x^*$  is defined as

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

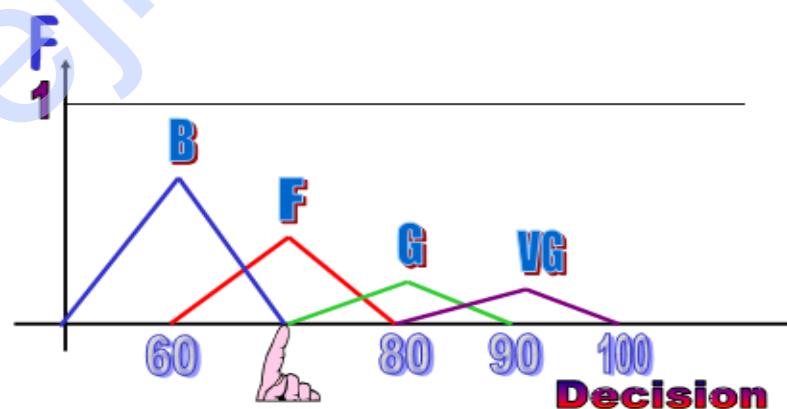
where the symbol  $\int$  denotes an algebraic integration.

## WEIGHTED AVERAGE METHOD

$$FD = \frac{\sum_i \mu_f f_n}{\sum_i \mu_f} = \frac{\mu_E \times E + \mu_{VG} \times VG + \dots}{\mu_E + \mu_{VG} + \dots}$$

$$FD = \frac{0 \times 100 + 0.2 \times 90 + 0.2 \times 80 + 0.4 \times 70 + 0.6 \times 60}{0.2 + 0.2 + 0.4 + 0.6} = 70$$

Final Decision (FD) = Fair Student



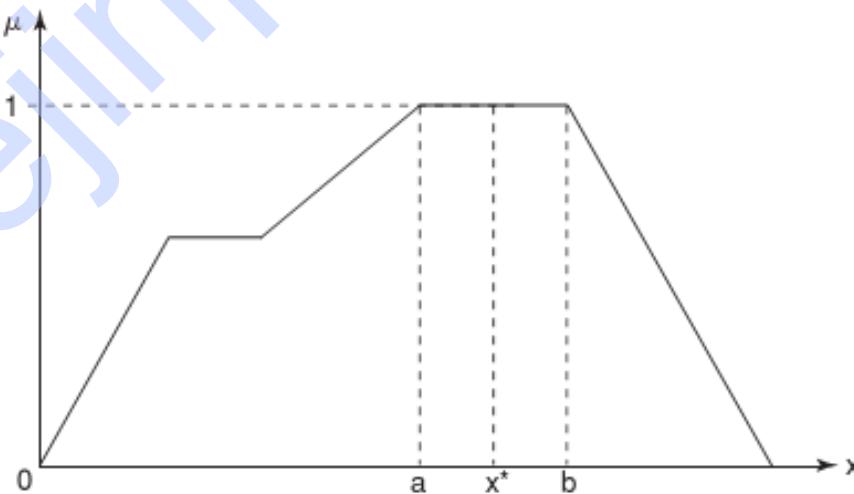
# MEAN MAX MEMBERSHIP METHOD

- The *mean of maximum defuzzification* yields the mean value of all local control actions whose membership functions reach the maximum. The crisp  $z$  value is given by in the above example:

$$z = \sum_{k=1}^2 \alpha_k H_k W_k / \sum_{k=1}^2 \alpha_k H_k,$$

- With  $W_k$  being the crisp support value at which the membership function reaches maximum  $H_k$  (most usually 1 for normalized membership functions).

$$x^* = \frac{a + b}{2}$$



## CENTER OF SUMS

This method employs the algebraic sum of the individual fuzzy subsets instead of their unions. The calculations here are very fast

but the **main drawback** is that the intersecting areas are added twice.

The defuzzified value  $x^*$  is given by

$$x^* = \frac{\int_x x \sum_{i=1}^n \mu_{C_i}(x) dx}{\int_x \sum_{i=1}^n \mu_{C_i}(x) dx}$$

## CENTER OF LARGEST AREA

This method can be adopted when the output consists of at least two convex fuzzy subsets which are not overlapping. The output in this case is biased towards a side of one membership function. When output fuzzy set has at least two convex regions then the center-of-gravity of the convex fuzzy subregion having the largest area is used to obtain the defuzzified value  $x^*$ .

This value is given by

$$x^* = \frac{\int \mu_{c_i}(x) \cdot x dx}{\int \mu_{c_i}(x) dx}$$

where  $c_j$  is the convex subregion that has the largest area making up  $c_i$

# FIRST OF MAXIMA (LAST OF MAXIMA)

The steps used for obtaining crisp values are as follows:

1. Initially, the maximum height in the union is found:

$$\text{hgt}(\underline{c}_i) = \sup_{x \in X} \mu_{\underline{c}_i}(x)$$

where sup is supremum, i.e., the least upper bound.

2. Then the first of maxima is found:

$$x^* = \inf_{x \in X} \{x \in X \mid \mu_{\underline{c}_i}(x) = \text{hgt}(\underline{c}_i)\}$$

where inf is the infimum, i.e., the greatest lower bound.

3. After this the last maxima is found:

$$x^* = \sup_{x \in X} \{x \in X \mid \mu_{\underline{c}_i}(x) = \text{hgt}(\underline{c}_i)\}$$

where

sup = supremum, i.e., the least upper bound

inf = infimum, i.e., the greatest lower bound

- Defuzzification process is essential because some engineering applications need exact values for performing the operation.
- Example: If speed of a motor has to be varied, we cannot instruct to raise it "slightly", "high", etc., using linguistic variables; rather, it should be specified as raise it by 200 rpm or so, i.e., a specific amount of raise should be mentioned.
- The method of defuzzification should be assessed on the basis of the output in the context of data available.

# FUZZY ARITHMETIC AND FUZZY MEASURES

## FUZZY ARITHMETIC

- Distributions do not have to be precise.
- Requires no assumption about correlations.
- Fuzzy measures are upper bounds on probability.
- Fuzzy arithmetic might be a conservative way to do risk assessments.

## FUZZY NUMBERS AND THEIR ARITHMETIC

- **Fuzzy numbers**
  - Fuzzy sets of the real line,
  - Unimodal,
  - Reach possibility level one.
- **Fuzzy arithmetic**
  - Interval arithmetic at each possibility level.

# FEATURES OF FUZZY ARITHMETIC

www.rejinpaul.com

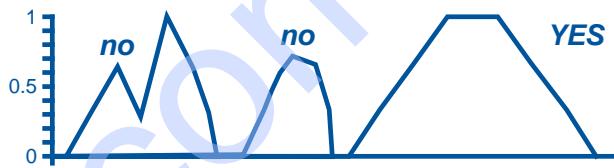
- Fully developed arithmetic and logic
  - Addition, subtraction, multiplication, division, min, max;
  - Log, exp, sqrt, abs, powers, and, or, not;
  - Backcalculation, updating, mixtures, etc.
- Very fast calculation and convenient software.
- Very easy to explain.
- Distributional answers (not just worst case).
- Results robust to choice about shape.

## TYPES OF NUMBERS

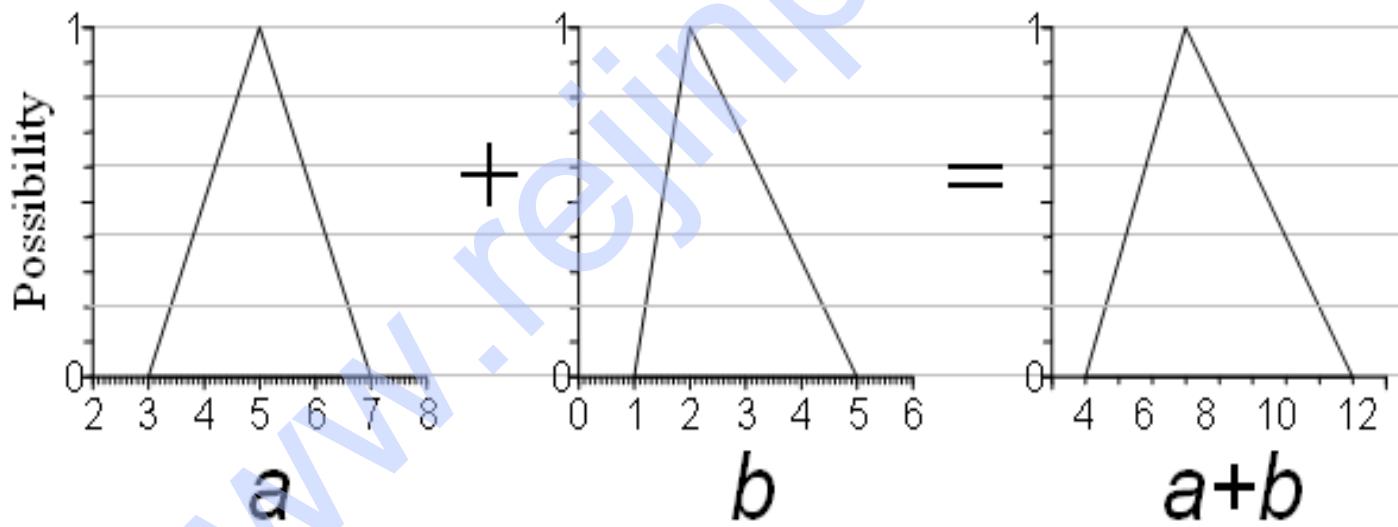
- **Scalars** are well-known or mathematically defined integers and real numbers.
- **Intervals** are numbers whose values are not known with certainty but about which bounds can be established.
- **Fuzzy** numbers are uncertain numbers for which, in addition to knowing a range of possible values, one can say that some values are more plausible than others.

# FUZZY NUMBERS

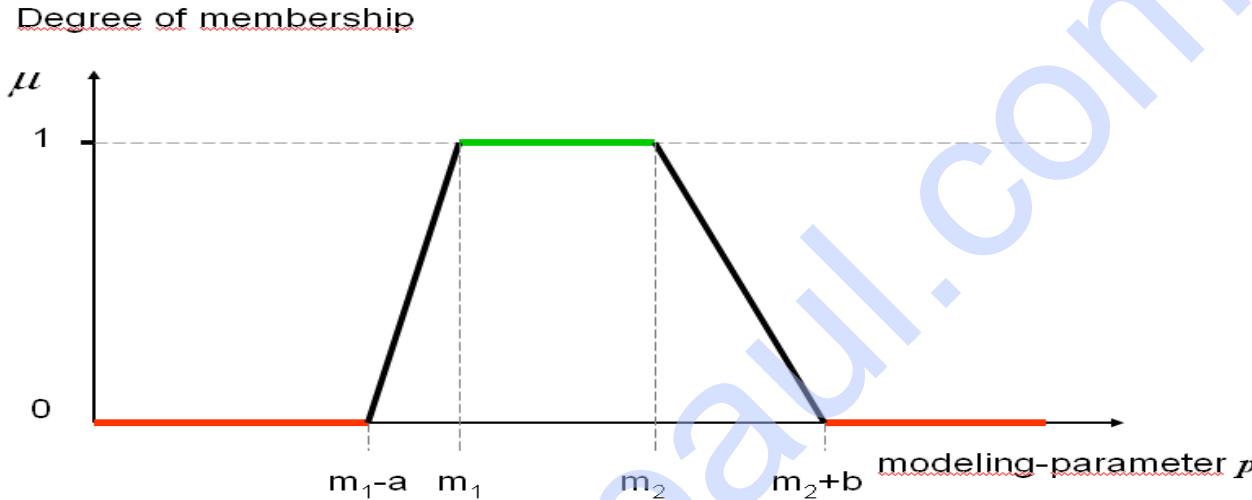
- Fuzzy set that is unimodal and reaches 1.
- Nested stack of intervals.



## LEVEL-WISE INTERVAL ARITHMETIC

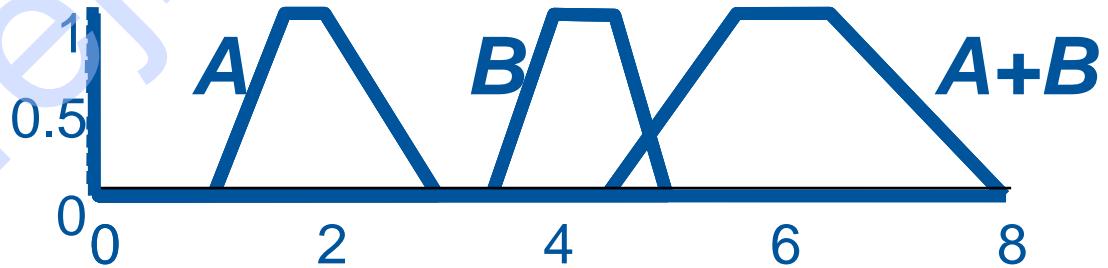


# FUZZY INTERVALS BY TRAPEZOIDAL SHAPES



Fuzzy Interval:  $P = [m_1, m_2, a, b]$

## FUZZY ADDITION



- Subtraction, multiplication, division, minimum, maximum, exponentiation, logarithms, etc. are also defined.
- If distributions are multimodal, possibility theory (rather than just simple fuzzy arithmetic) is required.

# FUZZY ARITHMETIC

## Interval Arithmetic

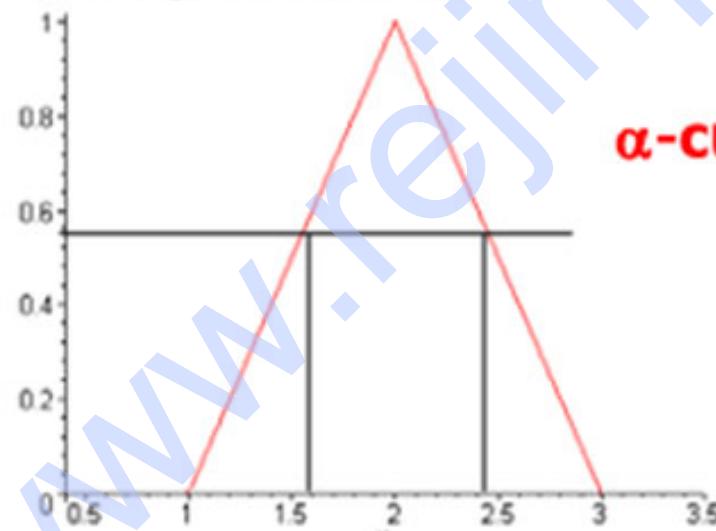
$$[a, b] + [d, e] = [a+d, b+e]$$

$$[a, b] - [d, e] = [a-e, b-d]$$

$$[a, b] \times [d, e] = [\min(a.d, a.e, b.d, b.e), \max(a.d, a.e, b.d, b.e)]$$

$$[a, b]/[d, e] = [\min(a/d, a/e, b/d, b/e), \max(a/d, a/e, b/d, b/e)]$$

## Fuzzy Numbers



**α-cuts are intervals**

**Let A and B be two fuzzy numbers  
then if \* = +, -, × or / then**

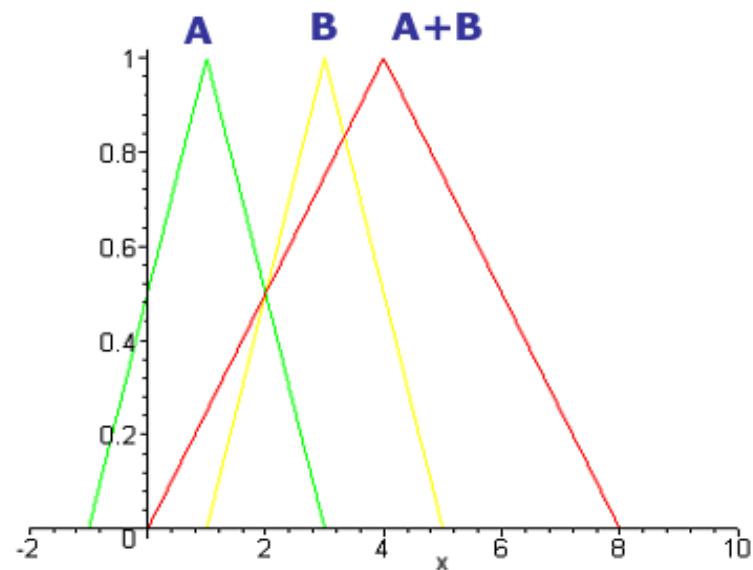
$$(A * B)_\alpha = A_\alpha * B_\alpha$$

**Example**

$$\mu_A = \begin{cases} 0 & \text{for } x \leq -1 \text{ and } x > 3 \\ (x+1)/2 & \text{for } -1 < x \leq 1 \\ (3-x)/2 & \text{for } 1 < x \leq 3 \end{cases} \quad A_\alpha = [2\alpha - 1, 3 - 2\alpha]$$

$$\mu_B = \begin{cases} 0 & \text{for } x \leq 1 \text{ and } x > 5 \\ (x-1)/2 & \text{for } 1 < x \leq 3 \\ (5-x)/2 & \text{for } 3 < x \leq 5 \end{cases} \quad B_\alpha = [2\alpha + 1, 5 - 2\alpha]$$

$$\begin{aligned} (A + B)_\alpha &= [2\alpha - 1, 3 - 2\alpha] + [2\alpha + 1, 5 - 2\alpha] \\ &= [4\alpha, 8 - 4\alpha] \end{aligned}$$



## EXTENSION PRINCIPLE

**A is a fuzzy set on X :**

$$A = \mu_A(x_1)/x_1 + \mu_A(x_2)/x_2 + \dots + \mu_A(x_n)/x_n$$

**The image of A under  $f()$  is a fuzzy set B:**

$$B = \mu_B(y_1)/y_1 + \mu_B(y_2)/y_2 + \dots + \mu_B(y_n)/y_n$$

**where  $y_i = f(x_i)$ ,  $i = 1$  to  $n$ .**

**If  $f()$  is a many-to-one mapping, then**

$$\mu_B(y) = \max_{x=f^{-1}(y)} \mu_A(x)$$

## VARIOUS FUZZY MEASURES

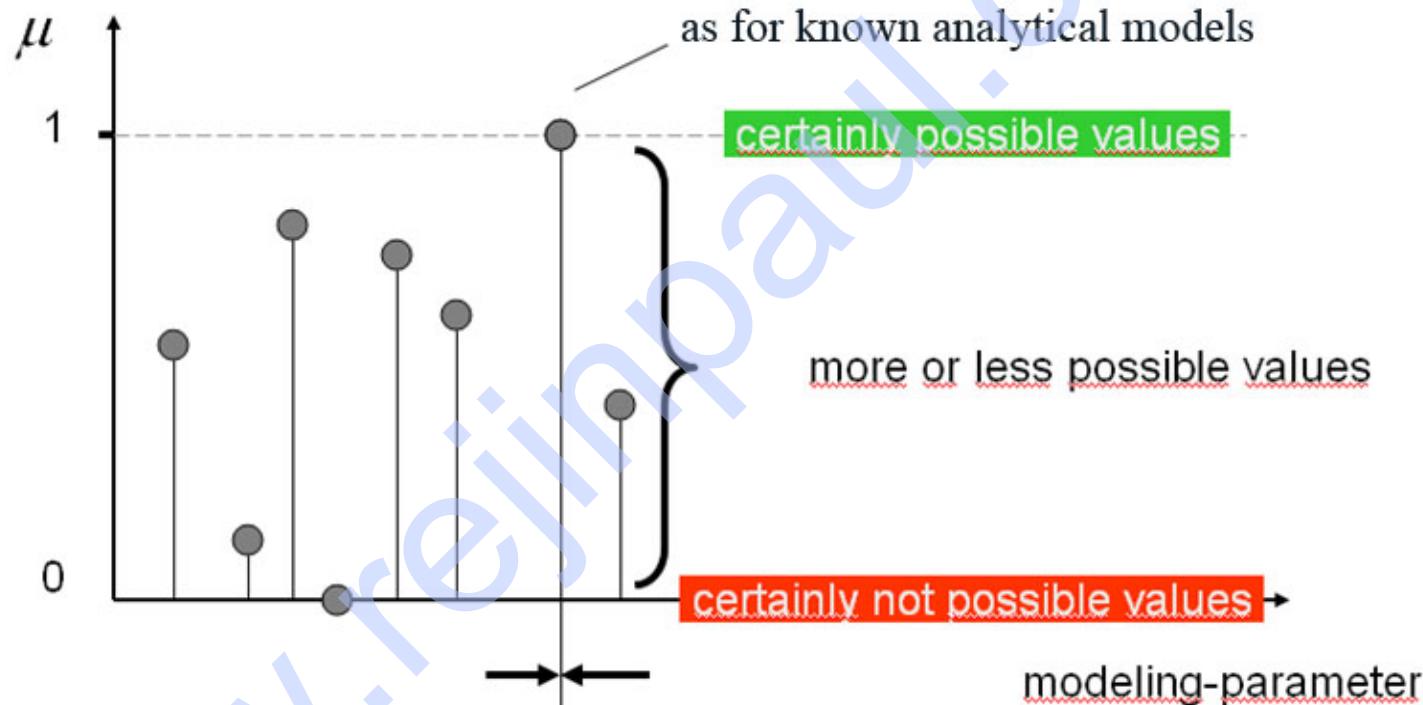
- Belief and Plausibility Measure.
- Probability Measure.
- Possibility and Necessity Measure.

## POSSIBILITY MEASURE

- No single definition.
- Many definitions could be used:
  - Subjective assessments.
  - Social consensus.
  - Measurement error.
  - Upper betting rates (Giles)
  - Extra-observational ranges (Gaines).

# FUZZY SET AS POSSIBILITY MEASURE

Degree of membership to a fuzzy set



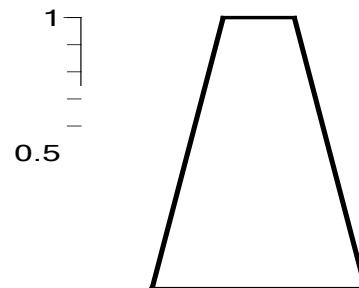
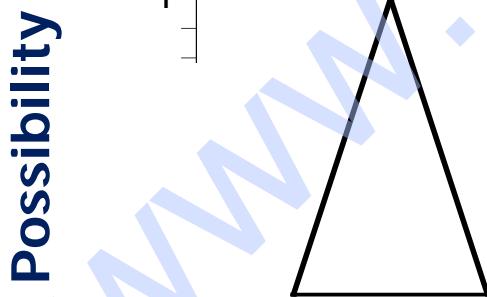
Exactly known parameter value, e. g.:

13.21345678953142.....

- **Subjective assignments:**
  - Make them up from highest, lowest and best-guess estimates.
- **Objective consensus:**
  - Stack up consistent interval estimates or bridge inconsistent ones.
- **Measurement error:**
  - Infer from measurement protocols.

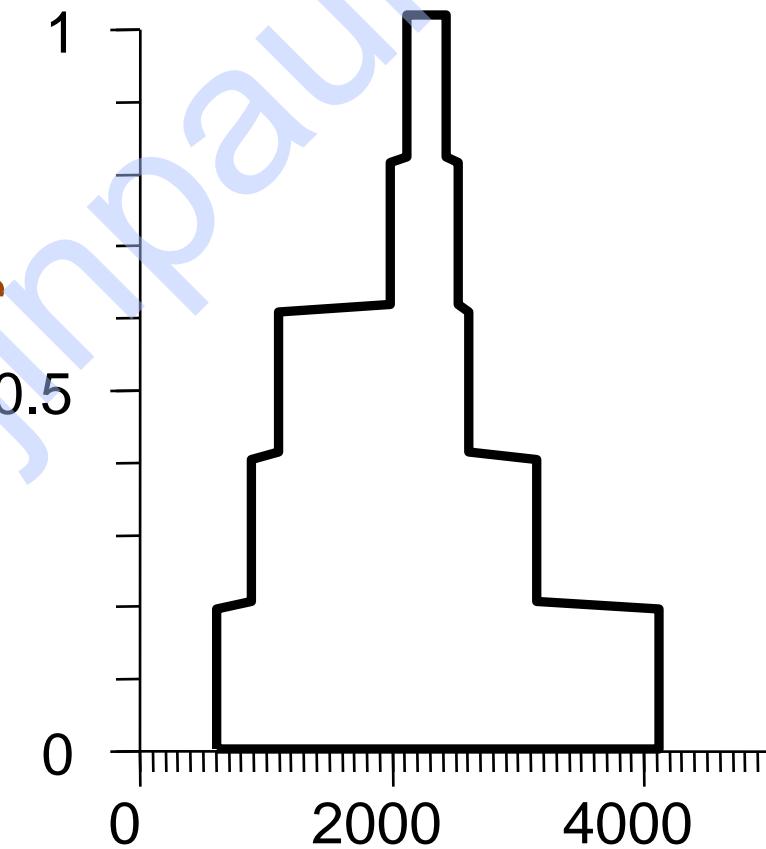
## SUBJECTIVE ASSIGNMENTS

- **Triangular fuzzy numbers, e.g. [1,2,3].**
- **Trapezoidal fuzzy numbers, e.g. [1,2,3,4].**



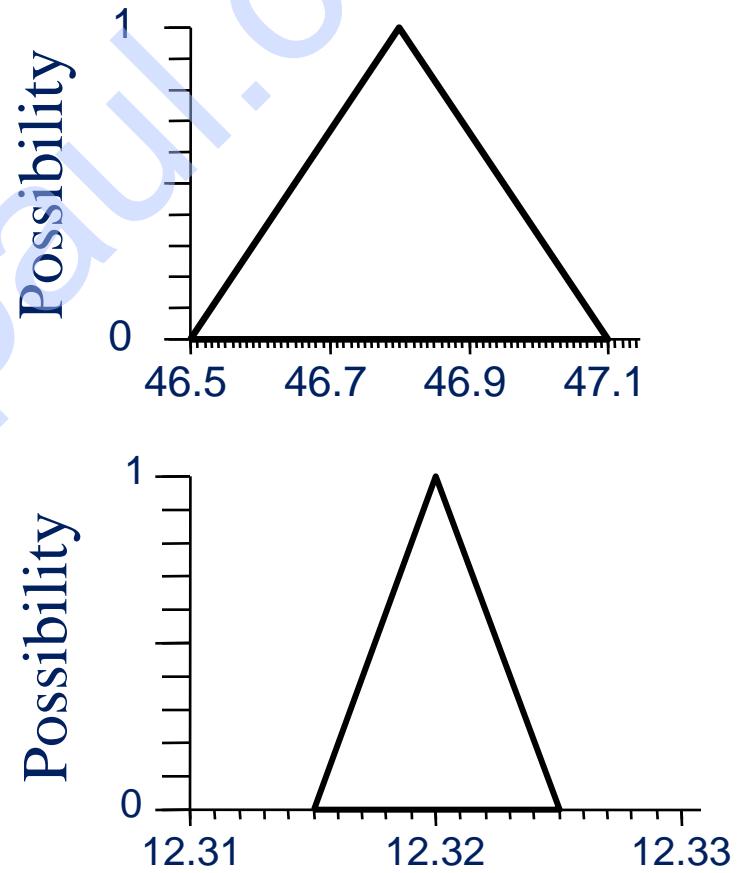
# OBJECTIVE CONSENSUS

- [1000, 3000]
- [2000, 2400]
- [500, 2500]
- [800, 4000]
- [1900, 2300]

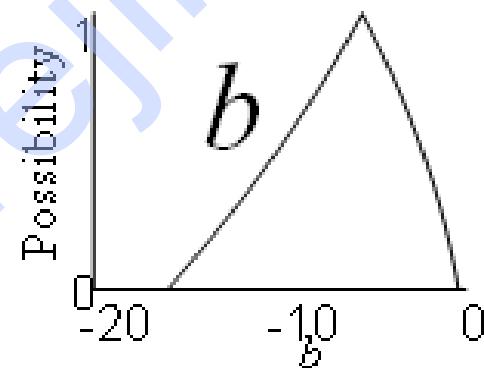
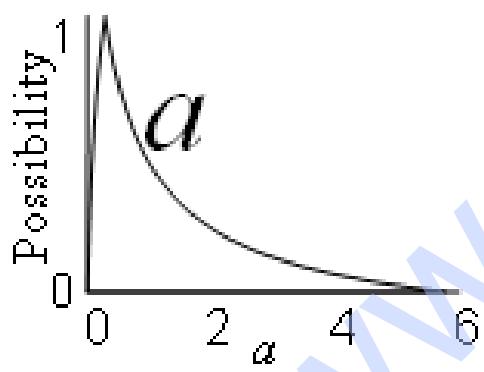
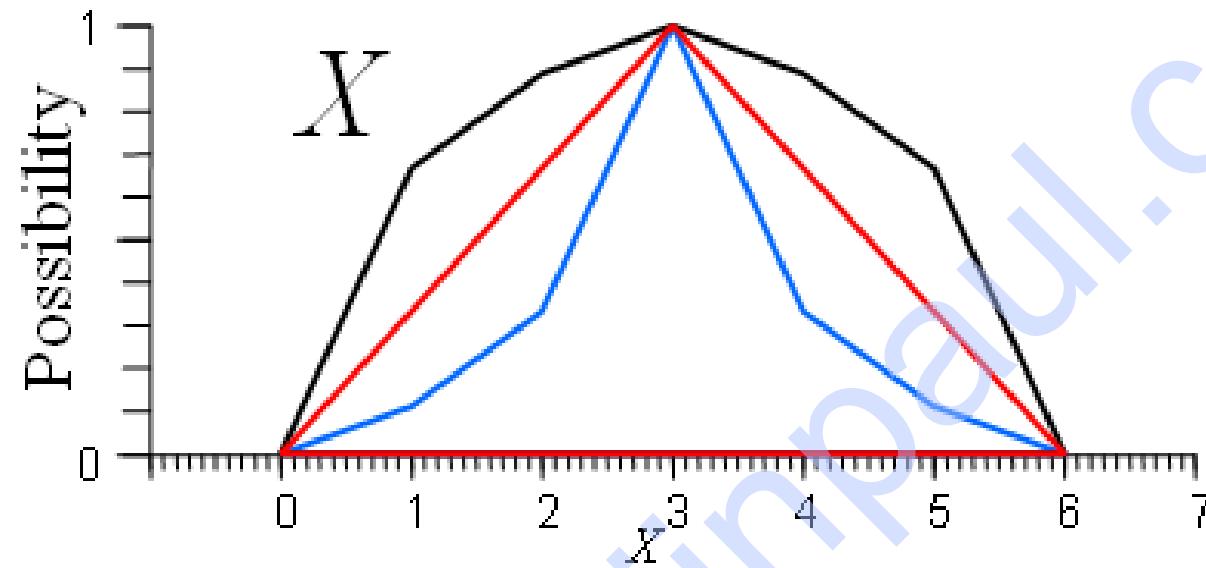


# MEASUREMENT ERROR

- $46.8 \pm 0.3$
- $[46.5, 46.8, 47.1]$
  
- $[12.32]$
- $[12.315, 12.32, 12.325]$



## SHAPE OF X AFFECTING $aX+b$



$$a = (d * e) / (h + g)$$

$$b = -f * e$$

where

$$d = [0.3, 1.7, 3]$$

$$e = [0.4, 1, 1.5]$$

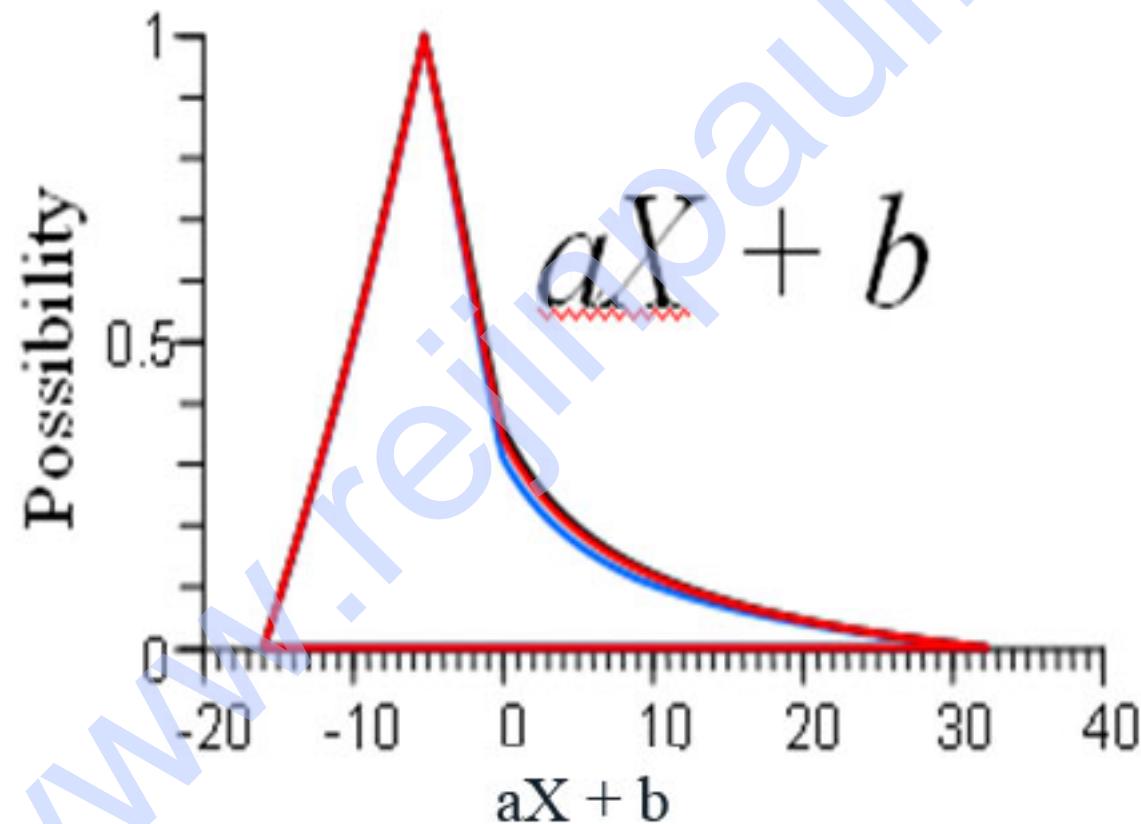
$$f = [0.8, 6, 10]$$

$$g = [0.2, 2, 5]$$

$$h = [0.6, 3, 6]$$

## ROBUSTNESS OF THE ANSWER

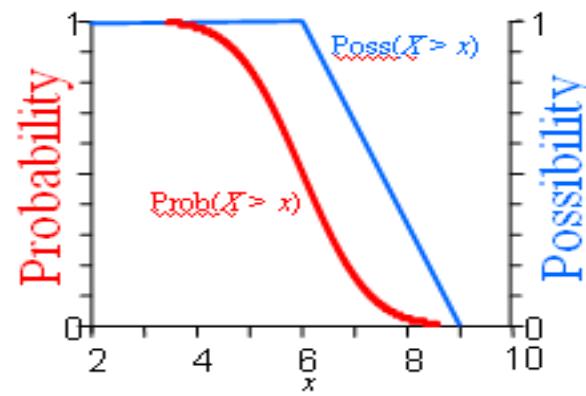
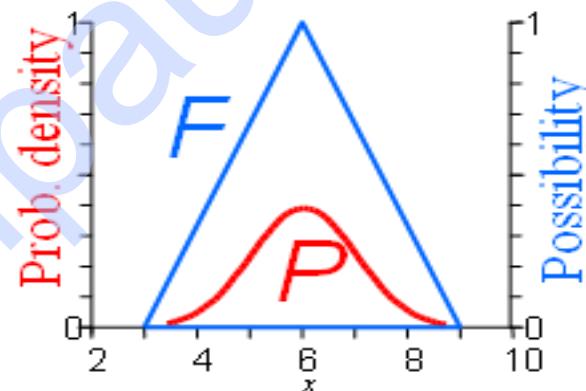
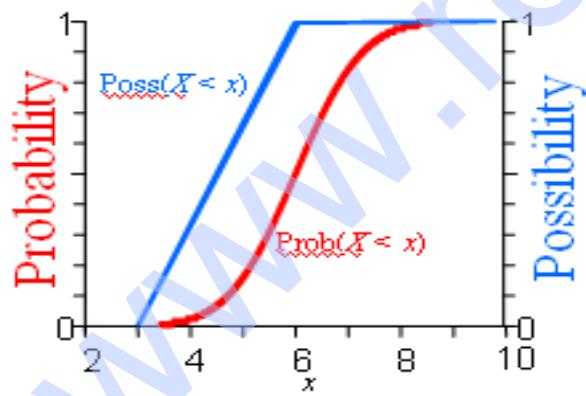
Different choices for the fuzzy number  $X$  all yield very similar distributions for  $aX + b$



# PROBABILITY MEASURE

- A fuzzy number  $F$  is said to “enclose” a probability distribution  $P$  if,
  - the left side of  $F$  is larger than  $P(x)$  for each  $x$ ,
  - the right side of  $F$  is larger than  $1-P(x)$  for each  $x$ .
- For every event  $X < x$  and  $x < X$ , possibility is larger than the probability, so it is an upper bound.

$F$  encloses  $P$



## ADVANTAGES OF FUZZY ARITHMETIC

- Requires little data.
- Applicable to all kinds of uncertainty.
- Fully comprehensive.
- Fast and easy to compute.
- Doesn't require information about correlations.
- Conservative, but not hyper-conservative
- In between worst case and probability.
- Backcalculations easy to solve.

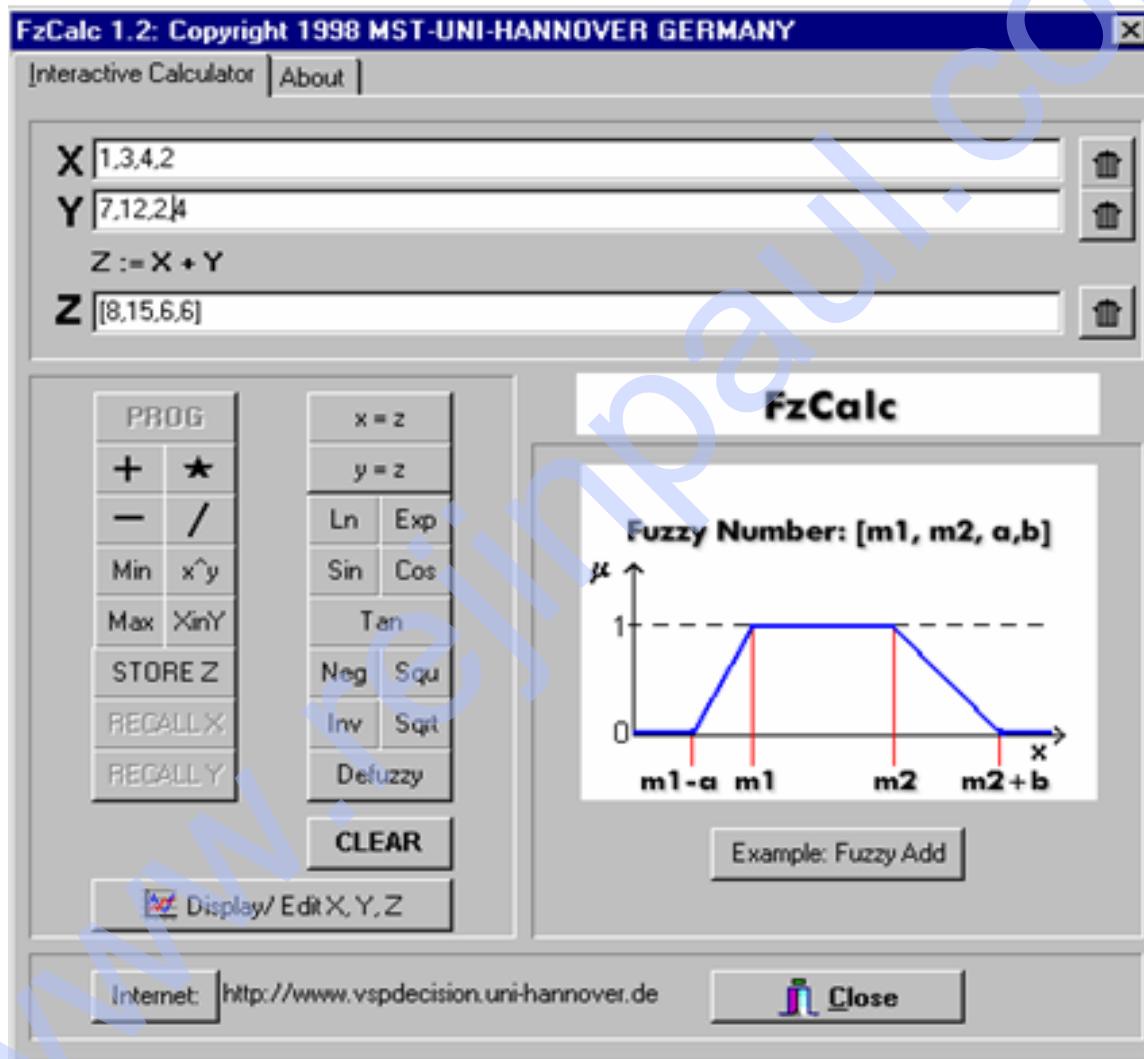
## LIMITATIONS OF FUZZY ARITHMETIC

- Controversial.
- Are alpha levels comparable for different variables?
- Not optimal when there're a lot of data.
- Can't use knowledge of correlations to tighten answers.
- Not conservative against all possible dependencies.
- Repeated variables make calculations cumbersome.

## SOFTWARES FOR FUZZY ARITHMETIC

- FuziCalc
  - (Windows 3.1) FuziWare, 800-472-6183
- Fuzzy Arithmetic C + + Library
  - (C code) anonymous ftp to mathct.dipmat.unict.it and get \fuzzy\fznum\*.\*
- Cosmet (Phaser)
  - (DOS, soon for Windows) [acooper@sandia.gov](mailto:acooper@sandia.gov)
- Risk Calc
  - (Windows) 800-735-4350; [www.ramas.com](http://www.ramas.com)

# SCREEN SHOT OF FUZZY CALCULATOR - FzCalc



# FUZZY RULE BASE AND APPROXIMATE REASONING

## FUZZY RULES AND REASONING

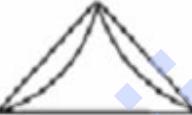
The degree of an element in a fuzzy set corresponds to the truth value of a proposition in fuzzy logic systems.

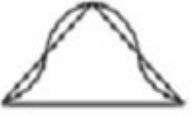
## LINGUISTIC VARIABLES

- A linguistic variable is a fuzzy variable.
  - The linguistic variable speed ranges between 0 and 300 km/h and includes the fuzzy sets slow, very slow, fast, ...
  - Fuzzy sets define the linguistic values.
- Hedges are qualifiers of a linguistic variable.
  - All purpose: very, quite, extremely
  - Probability: likely, unlikely
  - Quantifiers: most, several, few
  - Possibilities: almost impossible, quite possible

# LINGUISTIC HEDGES (LINGUISTIC QUANTIFIERS)

- Hedges modify the shape of a fuzzy set.

<i>Hedge</i>	<i>Mathematical Expression</i>	<i>Graphical Representation</i>
A little	$[\mu_A(x)]^{1.3}$	
Slightly	$[\mu_A(x)]^{1.7}$	
Very	$[\mu_A(x)]^2$	
Extremely	$[\mu_A(x)]^3$	

<i>Hedge</i>	<i>Mathematical Expression</i>	<i>Graphical Representation</i>
Very very	$[\mu_A(x)]^4$	
More or less	$\sqrt{\mu_A(x)}$	
Somewhat	$\sqrt{\mu_A(x)}$	
Indeed	$2 [\mu_A(x)]^2$ if $0 \leq \mu_A \leq 0.5$ $1 - 2 [1 - \mu_A(x)]^2$ if $0.5 < \mu_A \leq 1$	

## TRUTH TABLES

Truth tables define logic functions of two propositions. Let X and Y be two propositions, either of which can be true or false.

The operations over the propositions are:

1. Conjunction ( $\wedge$ ): X AND Y.
2. Disjunction ( $\vee$ ): X OR Y.
3. Implication or conditional ( $\Rightarrow$ ): IF X THEN Y.
4. Bidirectional or equivalence ( $\Leftrightarrow$ ): X IF AND ONLY IF Y.

## FUZZY RULES

A fuzzy rule is defined as the conditional statement of the form

If x is A THEN y is B

where x and y are linguistic variables and A and B are linguistic values determined by fuzzy sets on the universes of discourse X and Y.

- The decision-making process is based on rules with sentence conjunctives **AND**, **OR** and **ALSO**.
- Each rule corresponds to a fuzzy relation.
- Rules belong to a **rule base**.
- Example: If (Distance x to second car is **SMALL**) **OR** (Distance y to obstacle is **CLOSE**) **AND** (speed v is **HIGH**) **THEN** (perform **LARGE** correction to steering angle  $\theta$ ) **ALSO** (make **MEDIUM** reduction in speed v).
- Three antecedents (or premises) in this example give rise to two outputs (consequences).

## FUZZY RULE FORMATION

**IF** height is tall **THEN** weight is heavy.

Here the fuzzy classes height and weight have a given range (i.e., the universe of discourse).

range (height) = [140, 220]      range (weight) = [50, 250]

# FORMATION OF FUZZY RULES

Three general forms are adopted for forming fuzzy rules. They are:

- Assignment statements,
- Conditional statements,
- Unconditional statements.

## Assignment Statements

$y = \text{small}$

Orange color = orange

$a = s$

Paul is not tall and not very short

Climate = autumn

Outside temperature = normal

## Unconditional Statements

Goto sum.

Stop.

Divide by  $a$ .

Turn the pressure low.

## Conditional Statements

IF  $y$  is very cool THEN stop.

IF A is high THEN B is low ELSE B is not low.

IF temperature is high THEN climate is hot.

# DECOMPOSITION OF FUZZY RULES

www.rejinpaul.com

A compound rule is a collection of several simple rules combined together.  
Multiple conjunctive antecedent, Multiple disjunctive antecedent,  
Conditional statements (with ELSE and UNLESS).

## Multiple Conjunctive antecedents

IF  $x$  is  $\tilde{A}_1, \tilde{A}_2, \dots, \tilde{A}_n$  THEN  $y$  is  $\tilde{B}_m$ .

Assume a new fuzzy subset  $\tilde{A}_m$  defined as

$$\tilde{A}_m = \tilde{A}_1 \cap \tilde{A}_2 \cap \dots \cap \tilde{A}_n$$

and expressed by means of membership function

$$\mu_{\tilde{A}_m}(x) = \min [\mu_{\tilde{A}_1}(x), \mu_{\tilde{A}_2}(x), \dots, \mu_{\tilde{A}_n}(x)].$$

## Multiple disjunctive antecedent

IF  $x$  is  $\tilde{A}_1$  OR  $x$  is  $\tilde{A}_2, \dots$  OR  $x$  is  $\tilde{A}_n$  THEN  $y$  is  $\tilde{B}_m$ .

This can be written as

IF  $x$  is  $\tilde{A}_n$  THEN  $y$  is  $\tilde{B}_m$ ,

where the fuzzy set  $\tilde{A}_m$  is defined as

$$\tilde{A}_m = \tilde{A}_1 \cup \tilde{A}_2 \cup \tilde{A}_3 \cup \dots \cup \tilde{A}_n$$

## Conditional Statements ( With Else and Unless)

IF  $\tilde{A}_1$  (THEN  $\tilde{B}_1$ ) UNLESS  $\tilde{A}_2$

can be decomposed as

IF  $\tilde{A}_1$  THEN  $\tilde{B}_1$

OR

IF  $\tilde{A}_2$  THEN NOT  $\tilde{B}_1$

IF  $\tilde{A}_1$  THEN ( $\tilde{B}_1$ ) ELSE IF  $\tilde{A}_2$  THEN ( $\tilde{B}_2$ )

can be decomposed into the form

IF  $\tilde{A}_1$  THEN  $\tilde{B}_1$

OR

IF NOT  $\tilde{A}_1$  AND IF  $\tilde{A}_2$  THEN  $\tilde{B}_2$

# AGGREGATION OF FUZZY RULES

Aggregation of rules is the process of obtaining the overall consequents from the individual consequents provided by each rule.

- Conjunctive system of rules.
- Disjunctive system of rules.

## Conjunctive system of rules

*Conjunctive system of rules:* For a system of rules to be jointly satisfied, the rules are connected by “and” connectives. Here, the aggregated output,  $y$ , is determined by the fuzzy intersection of all individual rule consequents,  $y_i$ , where  $i = 1$  to  $n$ , as

$$y = y_1 \text{ and } y_2 \text{ and } \dots \text{ and } y_n$$

or

$$y = y_1 \cap y_2 \cap y_3 \cap \dots \cap y_n.$$

This aggregated output can be defined by the membership function

$$\mu_y(y) = \min [\mu_{y_1}(y), \mu_{y_2}(y), \dots, \mu_{y_n}(y)] \text{ for } y \in Y.$$

## Disjunctive system of rules

Disjunctive system of rules: In this case, the satisfaction of at least one rule is required. The rules are connected by “or” connectives. Here, the fuzzy union of all individual rule contributions determines the aggregated output, as

$$y = y_1 \text{ or } y_2 \text{ or } \dots \text{ or } y_n$$

or

$$y = y_1 \cup y_2 \cup y_3 \cup \dots \cup y_n.$$

Again it can be defined by the membership function

$$\mu_y(y) = \max [\mu_{y_1}(y), \mu_{y_2}(y), \dots \mu_{y_n}(y)] \text{ for } y \in Y.$$

## FUZZY RULE - EXAMPLE

**Rule 1:** If height is short then weight is light.

**Rule 2:** If height is medium then weight is medium.

**Rule 3:** If height is tall then weight is heavy.

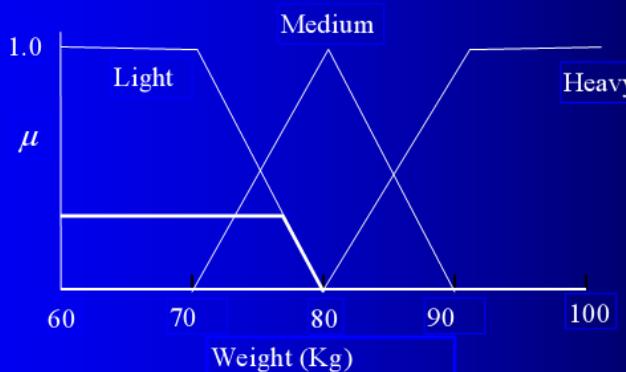
**Problem:** Given

- (a) membership functions for short, medium-height, tall, light, medium-weight and heavy;
- (b) The three fuzzy rules;
- (c) the fact that John's height is 6'1" estimate John's weight.

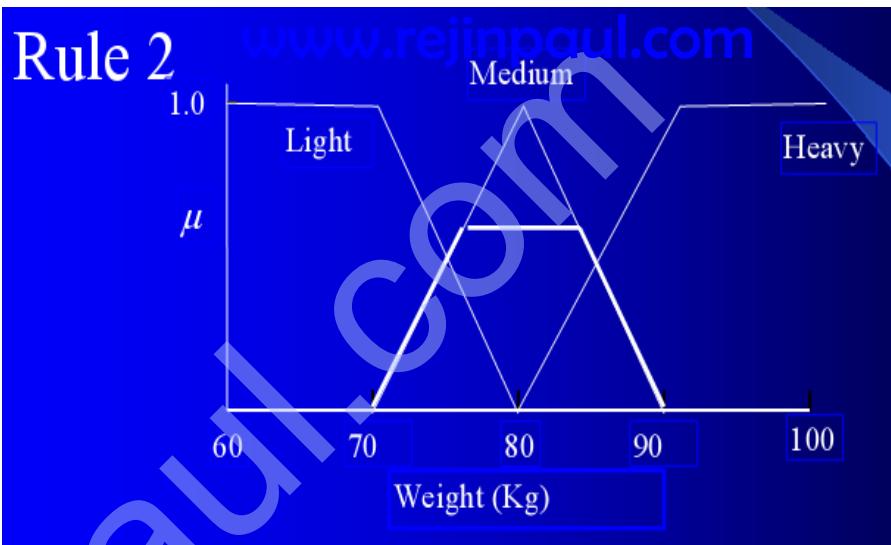
**Solution:**

- (1) From John's height we know that
  - John is short (degree 0.3)
  - John is of medium height (degree 0.6).
  - John is tall (degree 0.2).
- (2) Each rule produces a fuzzy set as output by truncating the consequent membership function at the value of the antecedent membership.

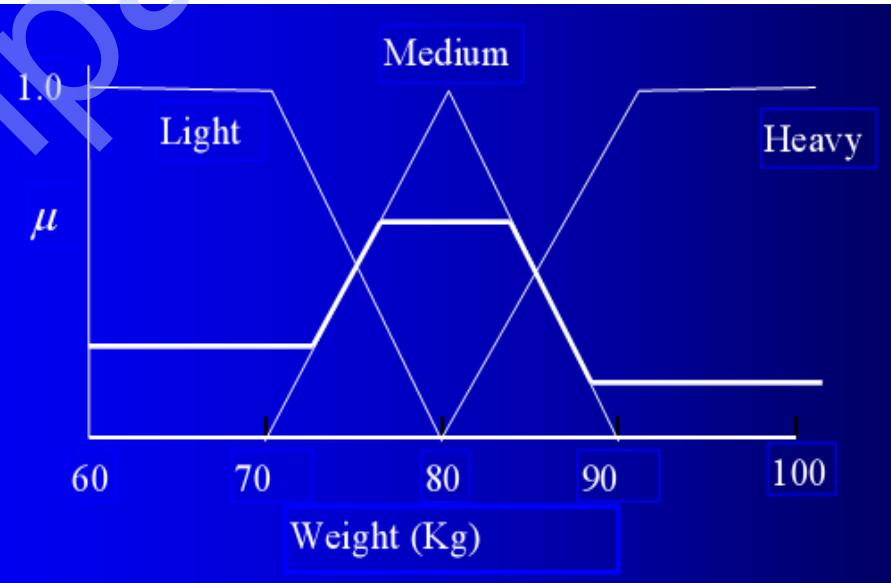
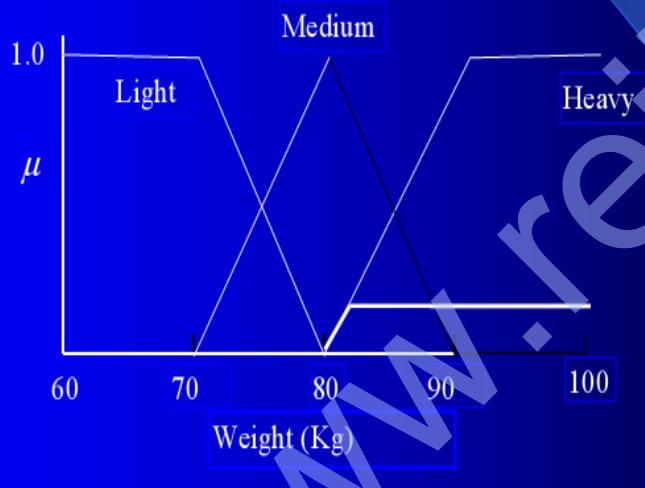
### Rule 1



### Rule 2



### Rule 3



- The cumulative fuzzy output is obtained by OR-ing the output from each rule. Cumulative fuzzy output (weight at 6'1").

1. De-fuzzify to obtain a numerical estimate of the output.
2. Choose the middle of the range where the truth value is maximum.
3. John's weight = 80 Kg.

There exist four modes of fuzzy approximate reasoning, which include:

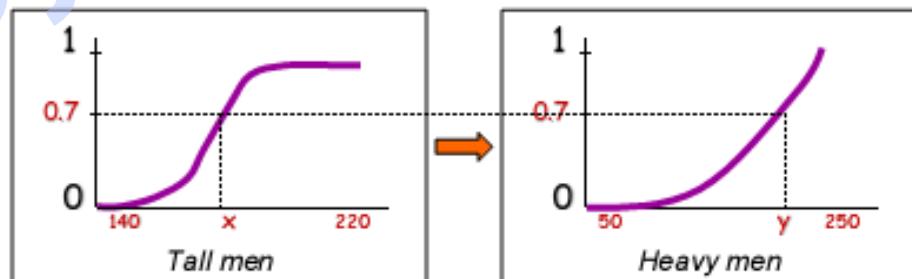
1. Categorical reasoning,
2. Qualitative reasoning,
3. Syllogistic reasoning,
4. Dispositional reasoning.

## REASONING WITH FUZZY RULES

- In classical systems, rules with true antecedents fire.
- In fuzzy systems, truth (i.e., membership in some class) is relative, so all rules fire (to some extent).

■ If the antecedent is true to some degree, the consequent is true to the same degree.

IF length is tall THEN weight is heavy



$$\mu_{\text{Tall}}(x) = 0.7 \rightarrow \mu_{\text{Heavy}}(y) = 0.7$$

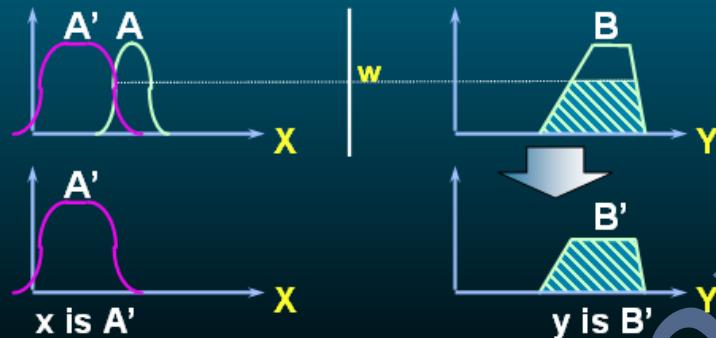
# SINGLE RULE WITH SINGLE ANTECEDANT

Rule: if  $x$  is  $A$  then  $y$  is  $B$

Fact:  $x$  is  $A'$

Conclusion:  $y$  is  $B'$

Graphic Representation:

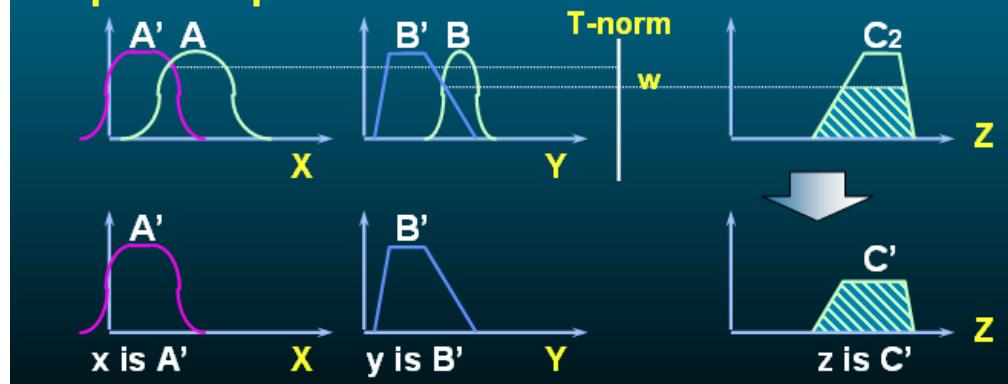


Rule: if  $x$  is  $A$  and  $y$  is  $B$  then  $z$  is  $C$

Fact:  $x$  is  $A'$  and  $y$  is  $B'$

Conclusion:  $z$  is  $C'$

Graphic Representation:



## MULTIPLE ANTECEDANTS

IF x is A AND y is B THEN z is C

IF x is A OR y is B THEN z is C

Use unification (OR) or intersection (AND) operations to calculate a membership value for the whole antecedent.

$$\text{AND: } \mu_C(z) = \min(\mu_A(x), \mu_B(y))$$

$$\text{OR: } \mu_C(z) = \max(\mu_A(x), \mu_B(y))$$

E.g. If rain is heavy AND wind is strong THEN weather is bad

$$((\mu_{\text{heavy}}(\text{rain}) = 0.7) \wedge (\mu_{\text{strong}}(\text{wind}) = 0.4)) \rightarrow (\mu_{\text{bad}}(\text{weather}) = 0.4)$$

# MULTIPLE RULE WITH MULTIPLE ANTECEDENTS

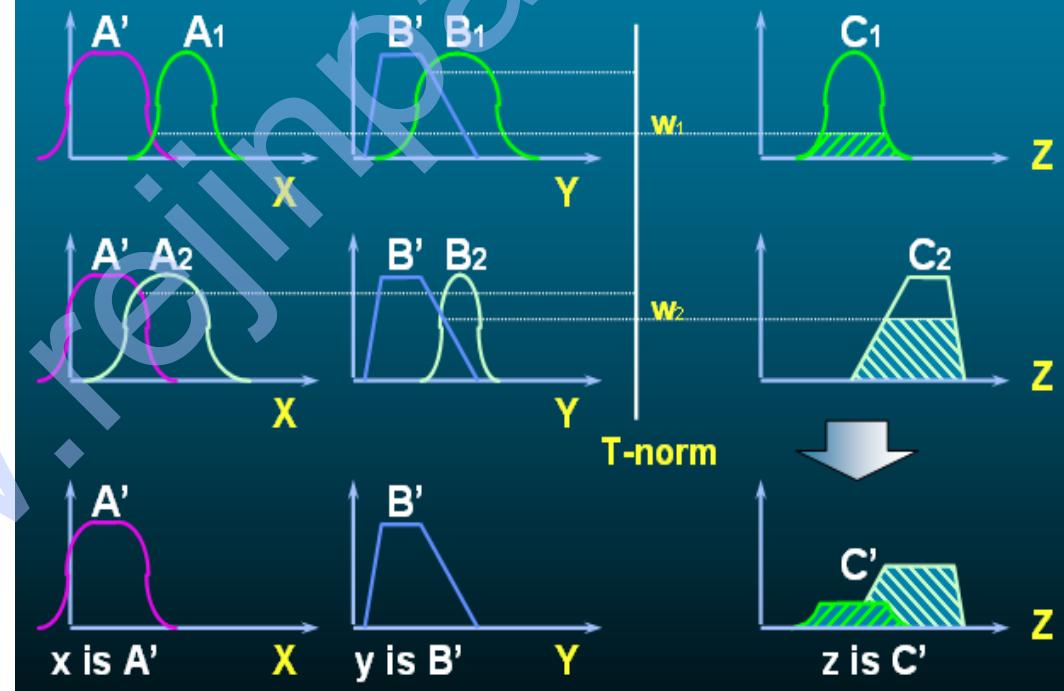
Rule 1: if  $x$  is  $A_1$  and  $y$  is  $B_1$  then  $z$  is  $C_1$

Rule 2: if  $x$  is  $A_2$  and  $y$  is  $B_2$  then  $z$  is  $C_2$

Fact:  $x$  is  $A'$  and  $y$  is  $B'$

Conclusion:  $z$  is  $C'$

Graphics representation:



## MULTIPLE CONSEQUENTS

IF x is A THEN y is B AND z is C

Each consequent is affected equally by the membership in the antecedent class(es).

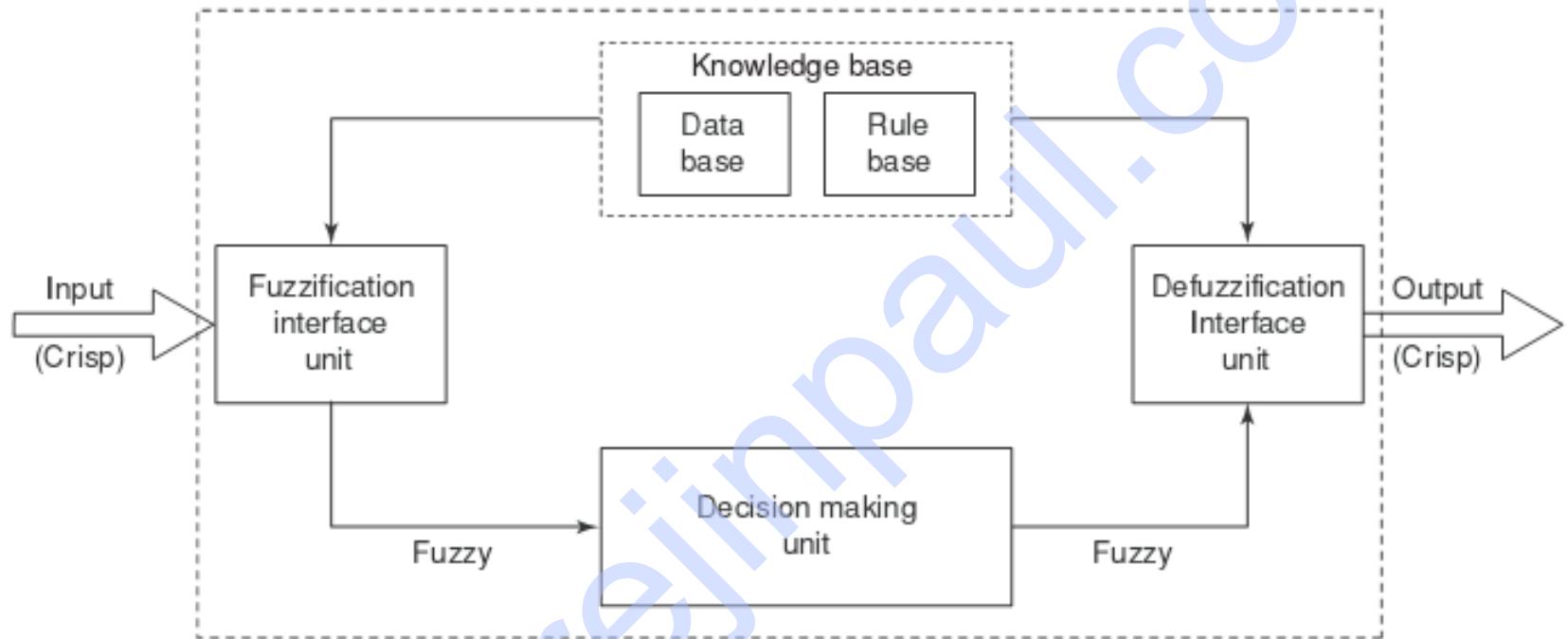
E.g., IF x is tall THEN x is heavy AND x has large feet.

$$\mu_{\text{Tall}}(x) = 0.7 \rightarrow \mu_{\text{Heavy}}(y) = 0.7 \wedge \mu_{\text{LargeFeet}}(y) = 0.7$$

## FUZZY INFERENCE SYSTEMS (FIS)

- Fuzzy rule based systems, fuzzy models, and fuzzy expert systems are also known as fuzzy inference systems.
- The key unit of a fuzzy logic system is FIS.
- The primary work of this system is decision-making.
- FIS uses "IF...THEN" rules along with connectors "OR" or "AND" for making necessary decision rules.
- The input to FIS may be fuzzy or crisp, but the output from FIS is always a fuzzy set.
- When FIS is used as a controller, it is necessary to have crisp output.
- Hence, there should be a defuzzification unit for converting fuzzy variables into crisp variables along FIS.

# BLOCK DIAGRAM OF FIS



## TYPES OF FIS

There are **two types** of Fuzzy Inference Systems:

- Mamdani FIS(1975)
- Sugeno FIS(1985)

# MAMDANI FUZZY INFERENCE SYSTEMS (FIS)

- Fuzzify input variables:
  - Determine membership values.
- Evaluate rules:
  - Based on membership values of (composite) antecedents.
- Aggregate rule outputs:
  - Unify all membership values for the output from all rules.
- Defuzzify the output:
  - COG: Center of gravity (approx. by summation).

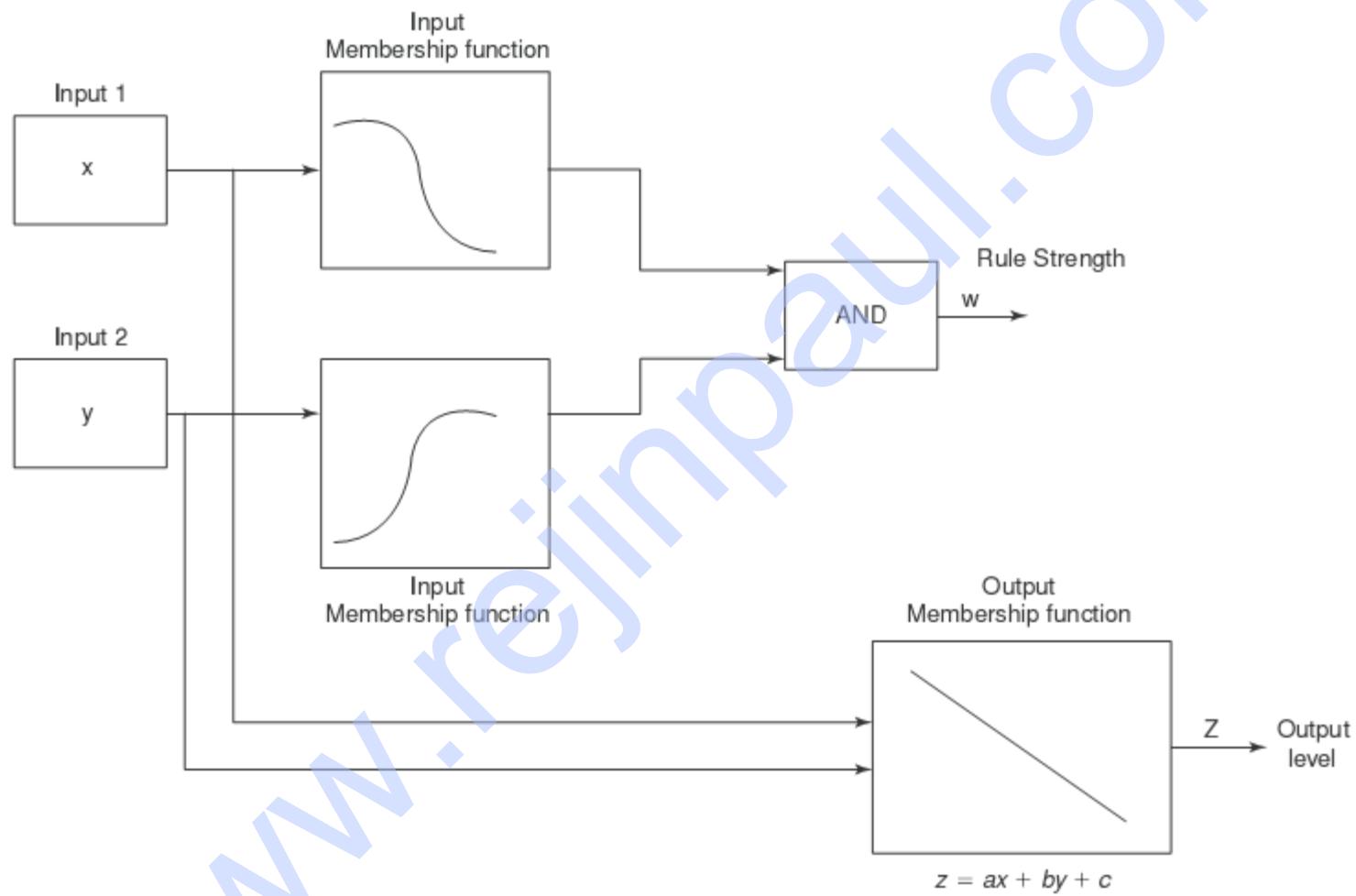
# SUGENO FUZZY INFERENCE SYSTEMS (FIS)

The **main steps** of the fuzzy inference process namely,

1. fuzzifying the inputs and
2. applying the fuzzy operator are exactly the same as in MAMDANI FIS.

The main **difference between Mamdani's and Sugeno's methods** is that Sugeno output membership functions are either linear or constant.

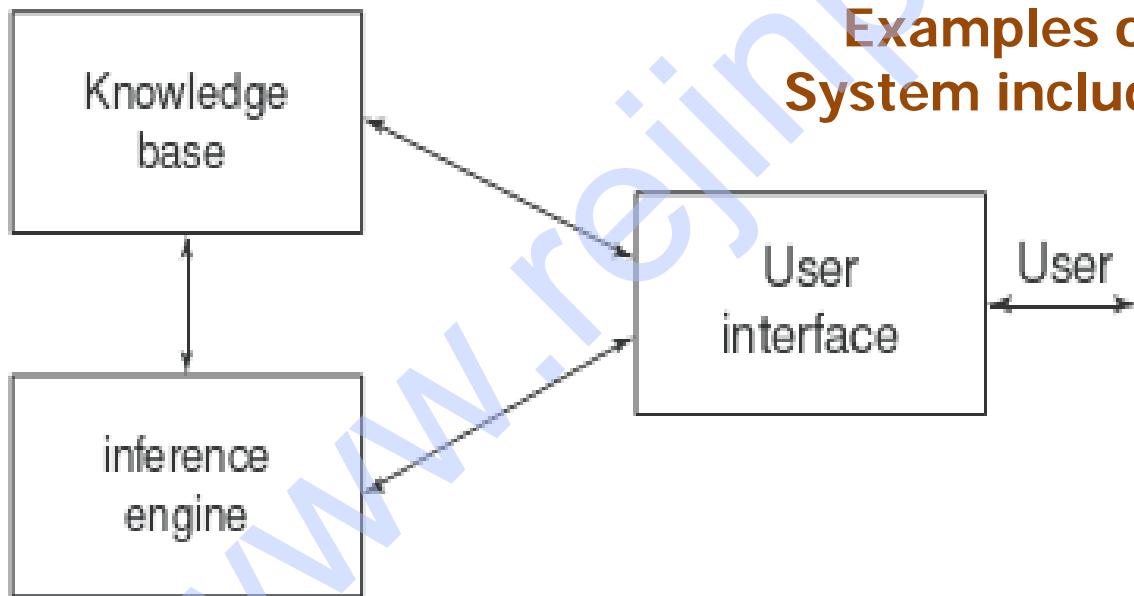
# SUGENO FIS



# FUZZY EXPERT SYSTEMS

An expert system contains **three major blocks**:

- Knowledge base that contains the knowledge specific to the domain of application.
- Inference engine that uses the knowledge in the knowledge base for performing suitable reasoning for user's queries.
- User interface that provides a smooth communication between the user and the system.



**Examples of Fuzzy Expert System include Z-II, MILORD.**

**BLOCK DIAGRAM OF FUZZY EXPERT SYSTEMS**

- **Advantages** of fuzzy logic
  - Allows the use of vague linguistic terms in the rules.
- **Disadvantages** of fuzzy logic
  - Difficult to estimate membership function
  - There are many ways of interpreting fuzzy rules, combining the outputs of several fuzzy rules and de-fuzzifying the output.

# FUZZY DECISION MAKING

Decision making is a very important social, economical and scientific endeavor.

The decision-making process involves **three steps**:

1. Determining the set of alternatives,
2. Evaluating alternatives,
3. Comparison between alternatives.

Certain fuzzy decision-making process are as follows:

- Individual decision making,
- Multiperson decision making,
- Multiobjective decision making,
- Multiattribute decision making,
- Fuzzy Bayesian decision making.

# INDIVIDUAL DECISION MAKING

A decision-making model in this situation is characterized by the following:

1. set of possible actions;
2. set of goals  $G_i (i \in X_n)$ ;
3. set of constraints  $C_j (j \in X_m)$ .

The goals and constraints are expressed in terms of fuzzy sets. These fuzzy sets in individual decision making are not defined directly on the set of actions, but by means of other sets that characterize relevant states of nature.

Consider a set  $A$ , then the goal and constraint for this set are given by

$$G_i(a) = \text{Composition}[G_i(a)] = G_i^1(G_i(a)) \text{ with } G_i^1$$

$$C_j(a) = \text{Composition}[C_j(a)] = C_j^1(C_j(a)) \text{ with } C_j^1$$

for  $a \in A$ . The fuzzy decision in this case is given by

$$F_D = \min \left[ \inf_{i \in X_n} G_i(a), \inf_{j \in X_m} C_j(a) \right]$$

# MULTIPERSON DECISION MAKING

In multiperson decision making, the decision makers have access to different information upon which to base their decision.

Here, each member of a group of ' $n$ ' individual decision makers has a preference ordering  $\text{PO}_k, k \in x_n$ , which totally or partially orders a set  $X$ . A social choice (sc)function has to be found, given the individual preference ordering. The fuzzy relation for a social choice preference function is given by,

$$\text{SC} : X \times X \rightarrow [0, 1]$$

which has a membership of  $\text{SC}(X_i, X_j)$ , which indicates the preference of alternative  $X_i$  over  $X_j$ . Let, Number of persons preferring  $X_i$  to  $X_j = N(X_i, X_j)$

Total Number of decision makers =  $n$

$$\text{Then, } \text{SC}(x_i, x_j) = \frac{N(x_i, x_j)}{n}$$

The multiperson decision making is also given by,

$$\text{SC}(x_i, x_j) = \begin{cases} 1, & \text{if } x_i >_k x_j \text{ for some } k \\ 0, & \text{otherwise} \end{cases}$$

## MULTIOBJECTIVE DECISION MAKING

In making a decision when there are several objectives to be realized, then the decision making is called **multiobjective decision making**.

Many decision processes may be based on single objectives such as **cost minimization, time consumption, profit maximization** and so on.

The **main issues** in multiobjective decision making are:

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

## MULTIATTRIBUTE DECISION MAKING

- The evaluation of alternatives can be carried out based on several attributes of the object called multiattribute decision making.
- The attributes may be classified into numerical data, linguistic data and qualitative data.

# MULTIATTRIBUTE DECISION MAKING

The problem of decision-making structure in multiatributes deals with determination of an evaluation structure for the multiattribute decision making from the available multiattribute data  $x_i$  ( $i = 1$  to  $n$ ) and alternative evaluations  $y$  shown in the table.

Alternative number	Alternative evaluation	Evaluation of alternative attributes
$j$	$y$	$x_1 \dots x_i \dots x_r$
1	$y_1$	$x_{11} \dots x_{i1} \dots x_{r1}$
2	$y_2$	$x_{12} \dots x_{i2} \dots x_{r2}$
$\vdots$	$\vdots$	$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$
$j$	$y_j$	$x_{1j} \dots x_{ij} \dots x_{rj}$
$\vdots$	$\vdots$	$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$
$n$	$y_n$	$x_{1n} \dots x_{in} \dots x_{rn}$

# FUZZY BAYESIAN DECISION MAKING

- Probability theory.
- Bayesian inference:
  - Use probability theory and information about independence.
  - Reason diagnostically [from evidence (effects) to conclusions (causes)] or causally (from causes to effects).
- Bayesian networks:
  - Compact representation of probability distribution over a set of propositional random variables.
  - Take advantage of independence relationships.

# FUZZY BAYESIAN DECISION MAKING

## Bayes' Theorem

$$P(A,B) = P(A|B) P(B)$$

$$P(B,A) = P(B|A) P(A)$$

**The theorem:**

---

$$P(B|A) = P(A|B) * P(B) / P(A)$$

---

## Expected Utility

The expected value of a variable is:

$$E[X] = \sum v_i \Pr(X = v_i)$$

i.e. it weights the value of events by their likelihood.

When the fuzzy events on the new information universe are orthogonal, we extend the Bayesian approach for considering fuzzy information. The fuzzy equivalents for the posterior probability, maximum expected utility and the marginal probability are given by, for a fuzzy event  $\tilde{E}_t$ ,

$$E(u_j | \tilde{E}_t) = \sum_{i=1}^n u_{ji} P(S_i | \tilde{E}_t)$$

$$E = (u^* | \tilde{E}_t) = \max_j E(u_j | \tilde{E}_t)$$

$$E(u_\phi^*) = \sum_{t=1}^g E(u^* | \tilde{E}_t) P(\tilde{E}_t)$$

The value of fuzzy information can be determined as

$$v(\phi) = E(u_\phi^*) - E(u^*)$$

# APPLICATIONS OF FUZZY LOGIC

- **Fuzzy Control**
  - Neuro-Fuzzy System
  - Intelligent Control
  - Hybrid Control
- **Fuzzy Pattern Recognition**
- **Fuzzy Modeling**

## Examples

- Ride smoothness control
- Camcorder auto-focus and jiggle control
- Braking systems
- Copier quality control
- Rice cooker temperature control
- High performance drives
- Air-conditioning systems

In the various decision-making process,

In many decision-making situations, the goals, constraints and consequences of the defined alternatives are known imprecisely, which is due to ambiguity and vagueness.

Methods for addressing this form of imprecision are important for dealing with many of the uncertainties we deal within human systems.

## INTRODUCTION TO FUZZY LOGIC TOOLBOX IN MATLAB

### FUZZY LOGIC TOOLBOX

- Fuzzy logic in Matlab can be dealt very easily due to the existing new Fuzzy Logic Toolbox.
- This provides a complete set of functions to design and implement various fuzzy logic processes.
- The major fuzzy logic operations include:
  - fuzzification,
  - defuzzification,
  - fuzzy inference.
- These all are performed by means of various functions and even can be implemented using the Graphical User Interface.

The **features** are:

- It provides tools to create and edit Fuzzy Inference Systems (FIS).  
It allows integrating fuzzy systems into simulation with SIMULINK.
- It is possible to create stand-alone C programs that call on fuzzy systems built with MATLAB.

The Toolbox provides three categories of tools:

- Command line functions,
- Graphical or interactive tools,
- Simulink blocks.

# COMMAND LINE FIS FUNCTIONS

**Addmf**

- Add membership function to FIS

**addrule**

- Add rule to FIS.

**addvar**

- Add variable to FIS.

**defuzz**

- Defuzzify membership function.

**evalfis**

- Perform fuzzy inference calculation

**evalmf**

- Generic membership function evaluation.

**gensurf**

- Generate FIS output surface.

**getfis**

- Get fuzzy system properties.

**mfstrch**

- Stretch membership function.

**newfis**

- Create new FIS.

**plotfis**

- Display FIS input-output diagram.

**plotmf**

- Display all membership functions for one variable.

**readfis**

- Load FIS from disk.

**rmmf**

- Remove membership function from FIS

**rmvar**

- Remove variable from FIS.

**Setfis**

- Set fuzzy system properties.

**showfis**

- Display annotated FIS.

**Showrule**

- Display FIS rules

**writefis**

- Save FIS to disk.

# MEMBERSHIP FUNCTIONS

**Dsigmf**

**gauss2mf**

**gaussmf**

**gbellmf**

**pimf**

**psigmf**

**smf**

**sigmf**

**trapmf**

**trimf**

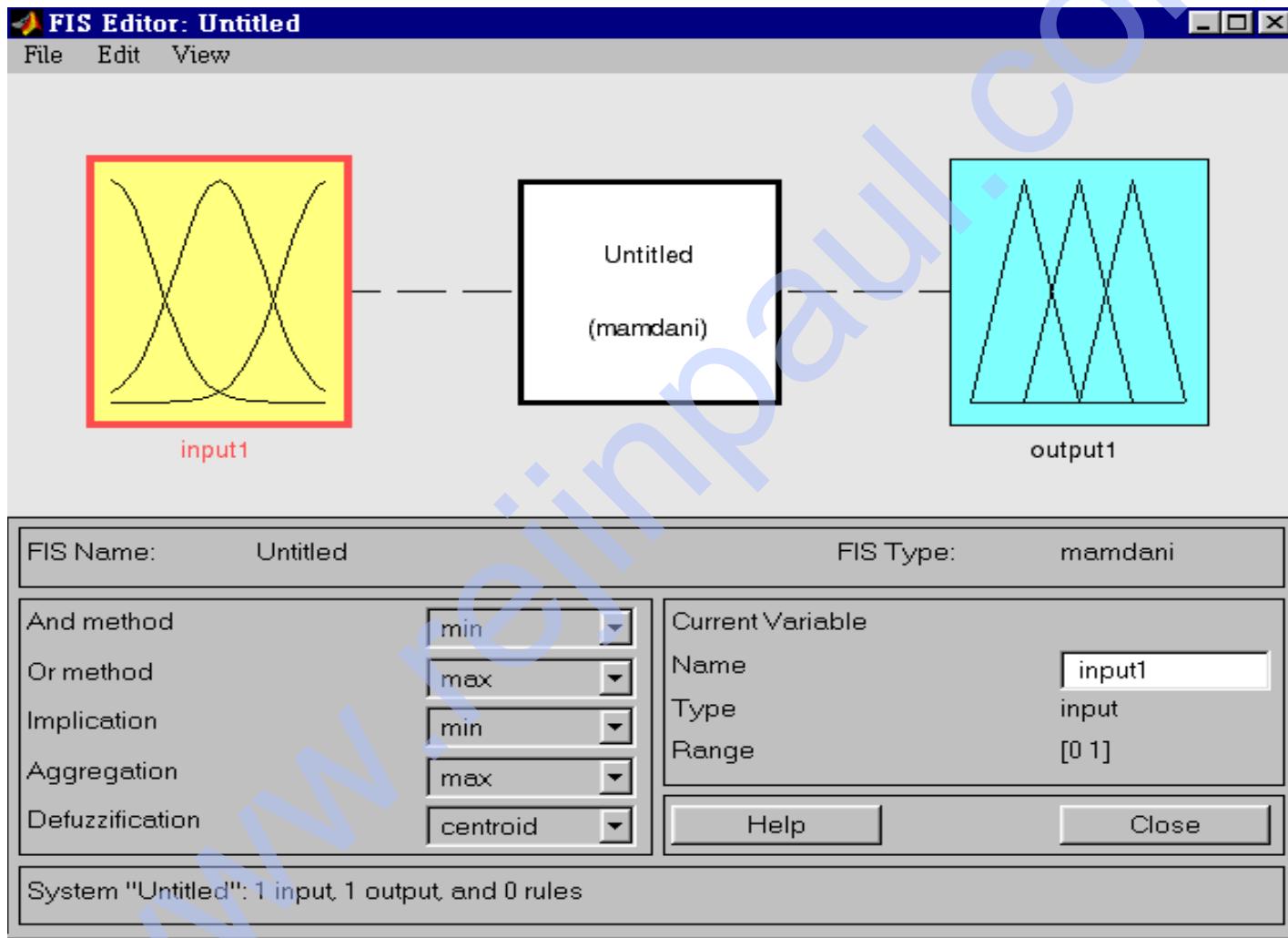
**zmf**

- Difference of two sigmoid membership functions.
- Two-sided Gaussian curve membership function.
- Gaussian curve membership function.
- Generalized bell curve membership function.
- Pi-shaped curve membership function.
- Product of two sigmoid membership functions.
- S-shaped curve membership function.
- Sigmoid curve membership function.
- Trapezoidal membership function.
- Triangular membership function.
- Z-shaped curve membership function.

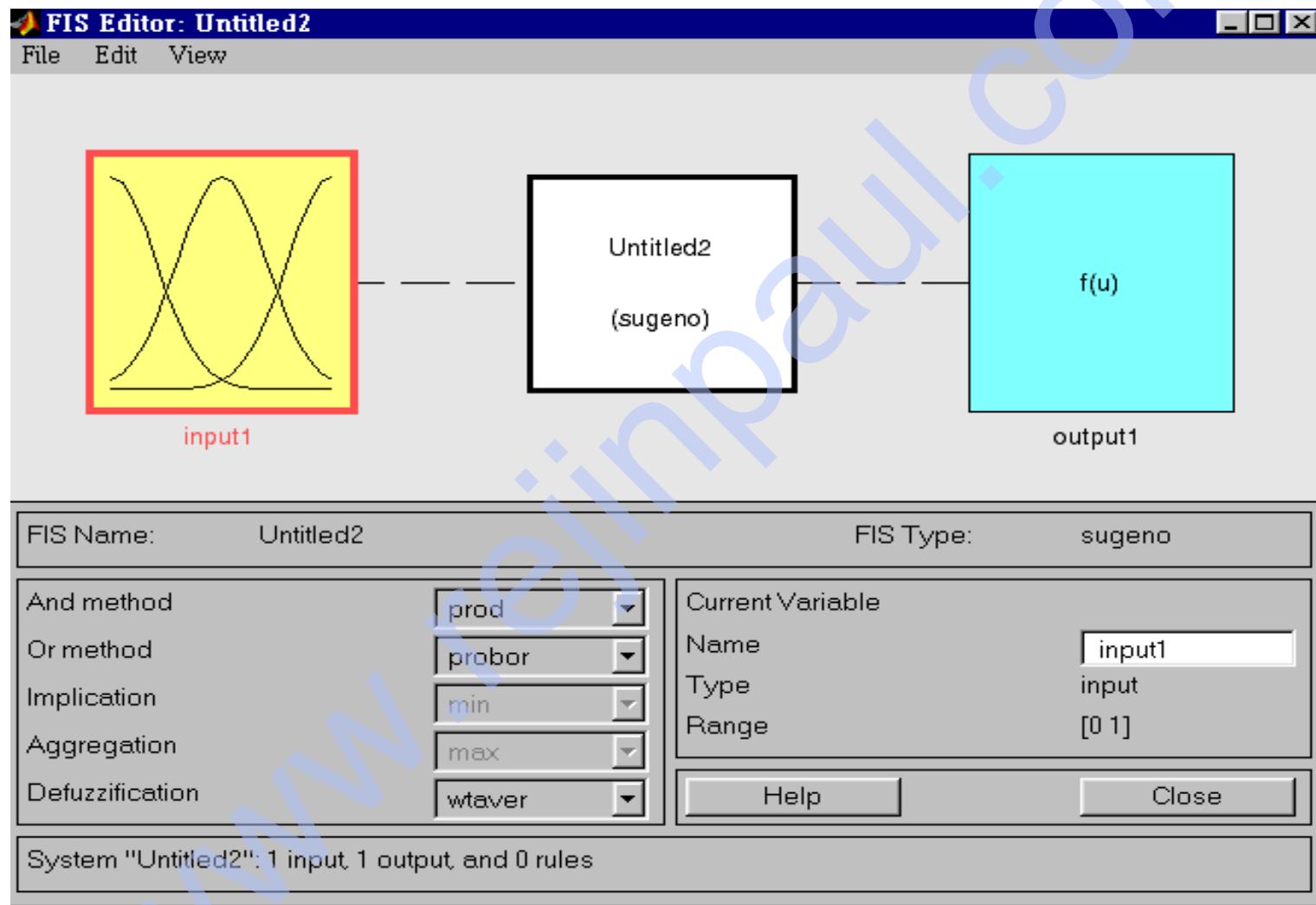
# GRAPHICAL USER INTERFACE EDITORS (GUI TOOLS)

<b>anfisedit</b>	-	ANFIS training and testing UI tool.
<b>findcluster</b>	-	Clustering UI tool.
<b>fuzzy</b>	-	Basic FIS editor.
<b>mfedit</b>	-	Membership function editor.
<b>ruleedit</b>	-	Rule editor and parser.
<b>ruleview</b>	-	Rule viewer and fuzzy inference diagram.
<b>surfview</b>	-	Output surface viewer.

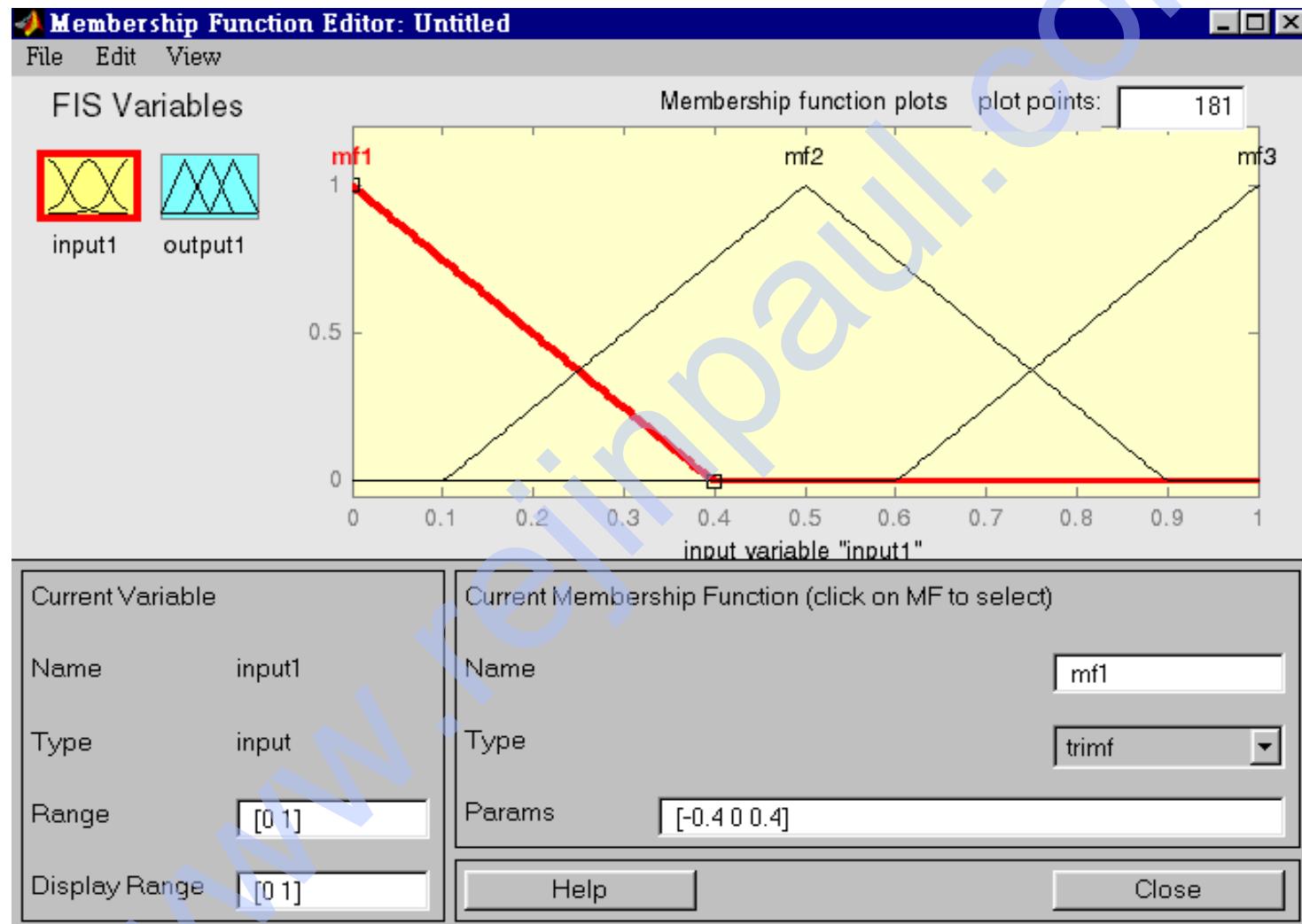
# FIS EDITOR (MAMDANI)



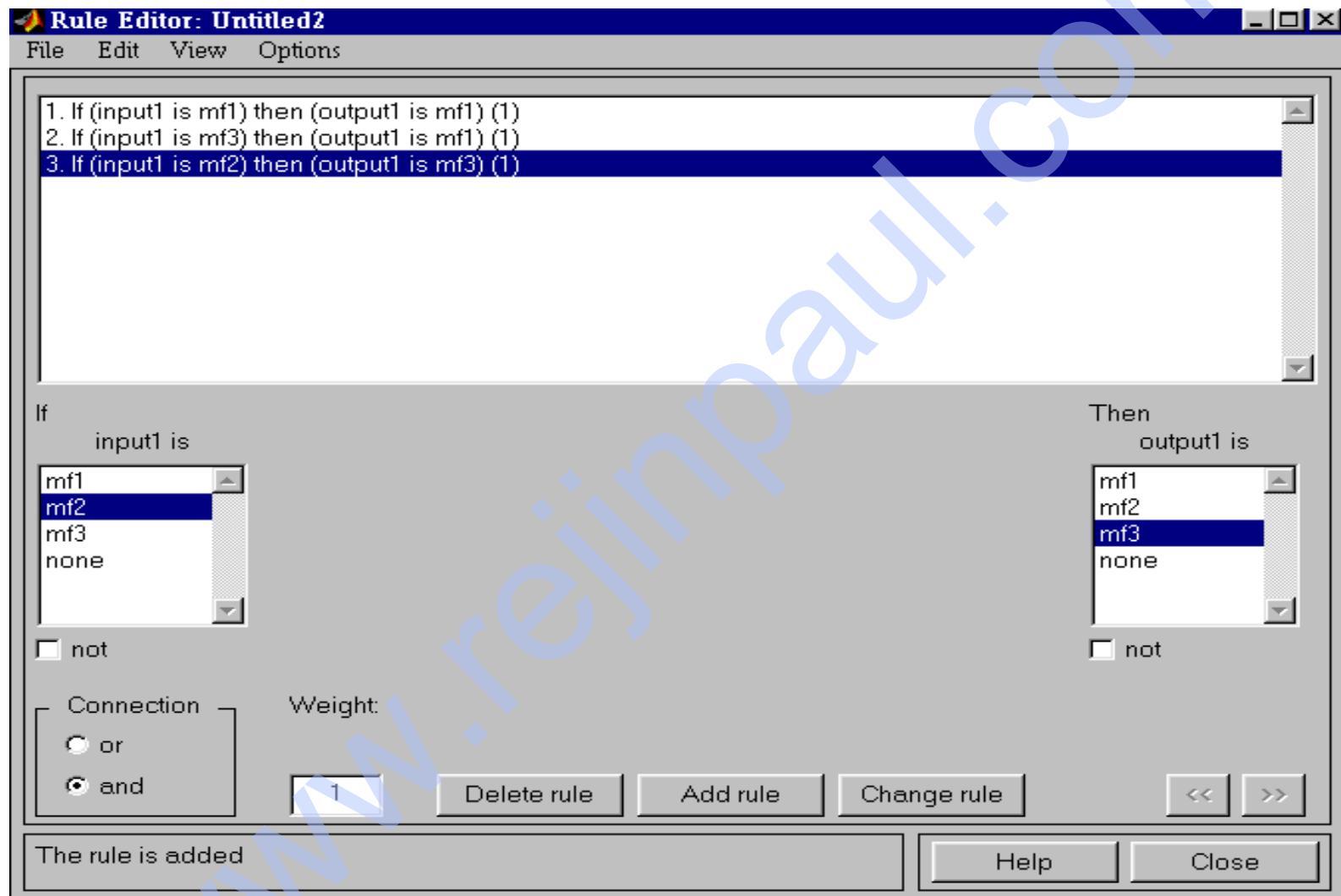
# FIS EDITOR (SUGENO)



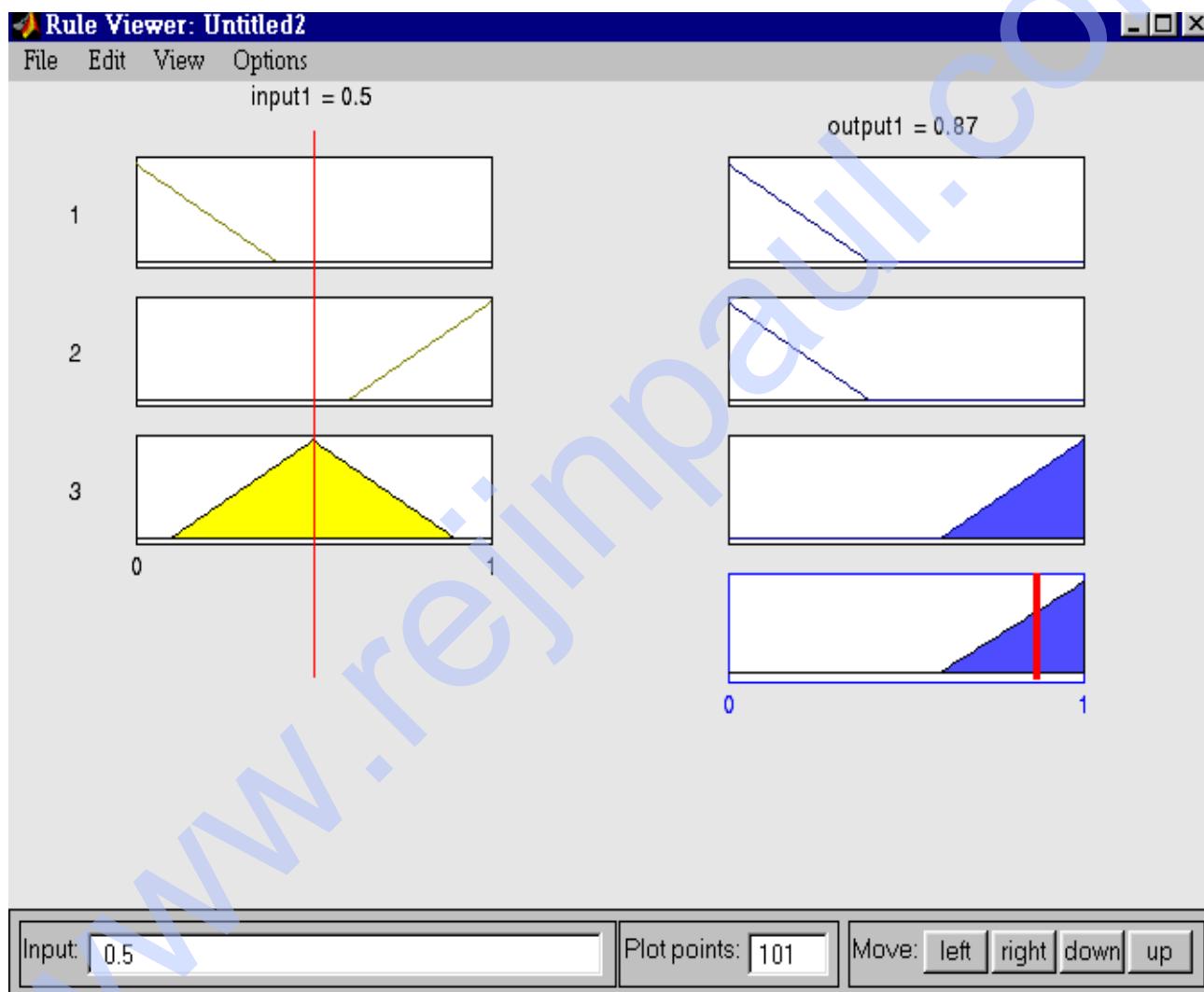
# FIS MEMBERSHIP FUNCTION EDITOR



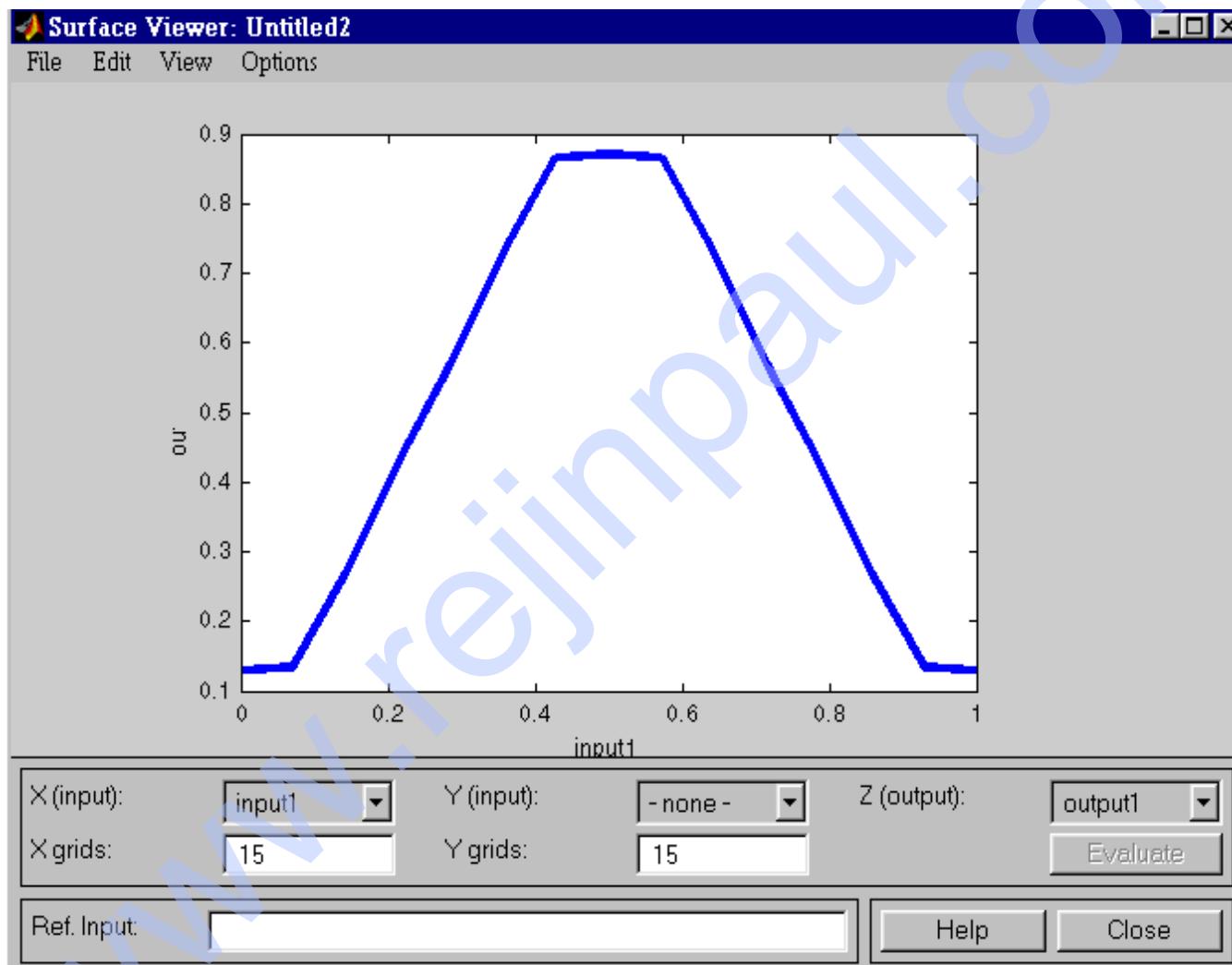
# FIS RULE EDITOR



# FIS RULE VIEWER



# FIS SURFACE VIEWER



# SIMULINK BLOCKS

Once fuzzy system is created using GUI tools or some other method, can be directly embedded into SIMULINK using the Fuzzy Logic Controller block.



## ADVANCED TECHNIQUES

**anfis**

- Training routine for Sugeno-type FIS (MEX only).

**fcm**

- Find clusters with fuzzy c-means clustering.

**genfis1**

- Generate FIS matrix using generic method.

**genfis2**

- Generate FIS matrix using subtractive clustering.

**subclust**

- Estimate cluster centers with subtractive clustering.

# UNIT -IV

# GENETIC ALGORITHM

## Genetic Algorithm (3) – Basic Algorithm

Outline of the basic algorithm

**0 START** : Create random population of n chromosomes

**1 FITNESS** : Evaluate fitness  $f(x)$  of each chromosome in the population

### **2 NEW POPULATION**

**1 REPRODUCTION/SELECTION** : Based on  $f(x)$

**2 CROSS OVER** : Cross-over chromosomes

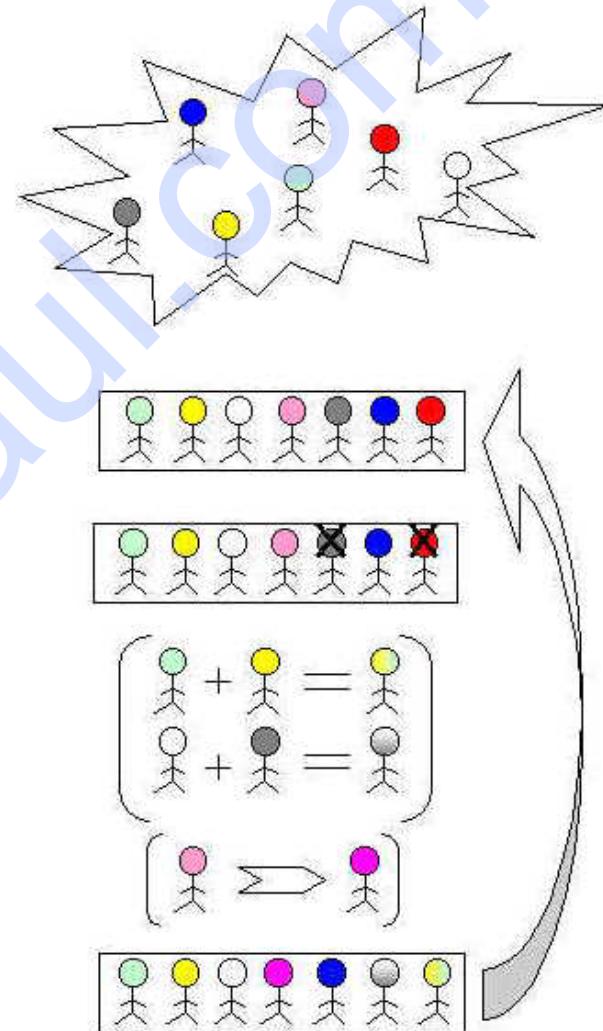
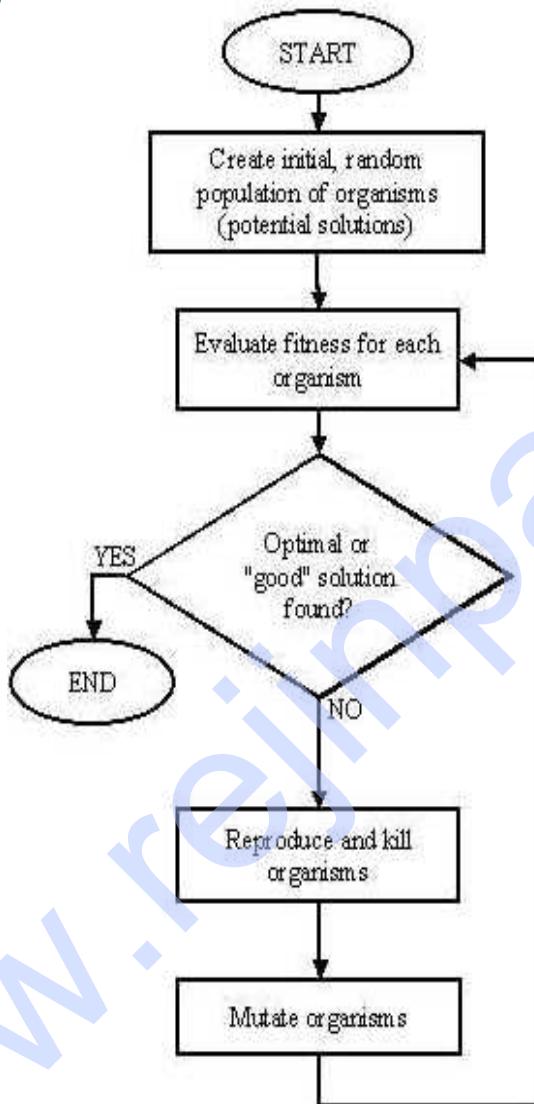
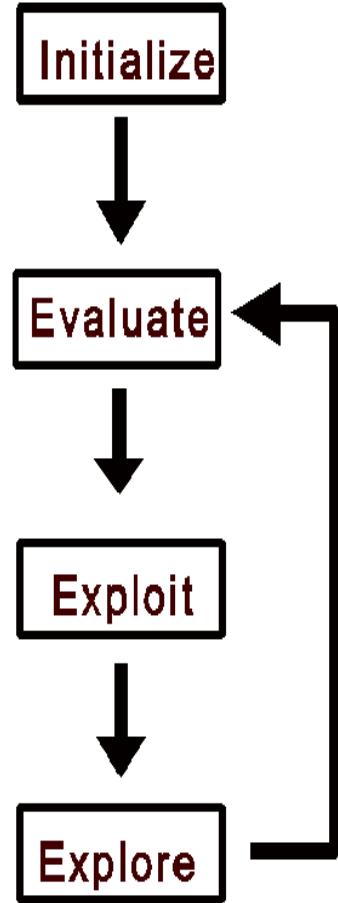
**3 MUTATION** : Mutate chromosomes

**3 REPLACE** : Replace old with new population: the new generation

**4 TEST** : Test problem criterium

**5 LOOP** : Continue step 1 – 4 untill criterium is satisfied

## Flowchart of GA



- All individuals in population evaluated by fitness function.
- Individuals allowed to reproduce (Selection), Crossover, mutate.

## Genetic Algorithm – Reproduction Cycle

1. Select parents for the mating pool (size of mating pool=population size).
2. Shuffle the mating pool.
3. For each consecutive pair apply crossover.
4. For each offspring apply mutation (bit-flip independently for each bit).
5. Replace the whole population with the resulting offspring.

## Genetic Algorithm – Coding

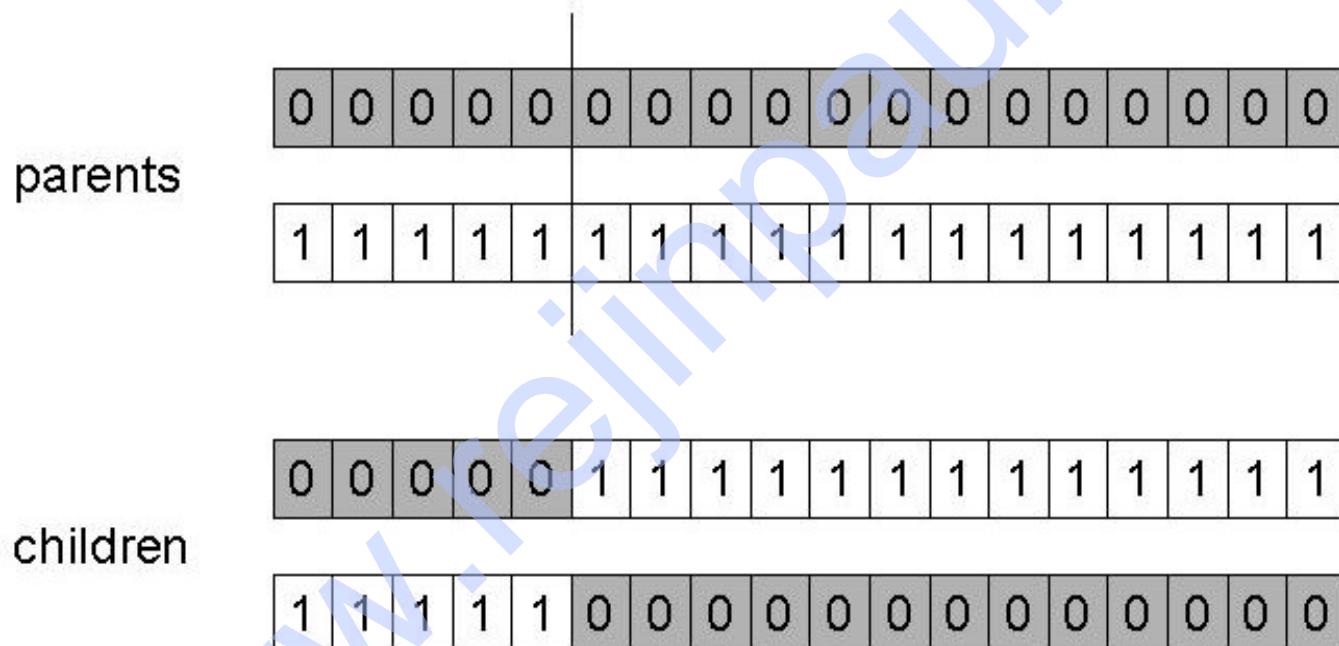
- Chromosomes are encoded by bitstrings.
- Every bitstring therefore is a solution but not necessarily the best solution.
- The way bitstrings can code differs from problem to problem.

Either sequence of on/off or the number 9

1
0
0
1

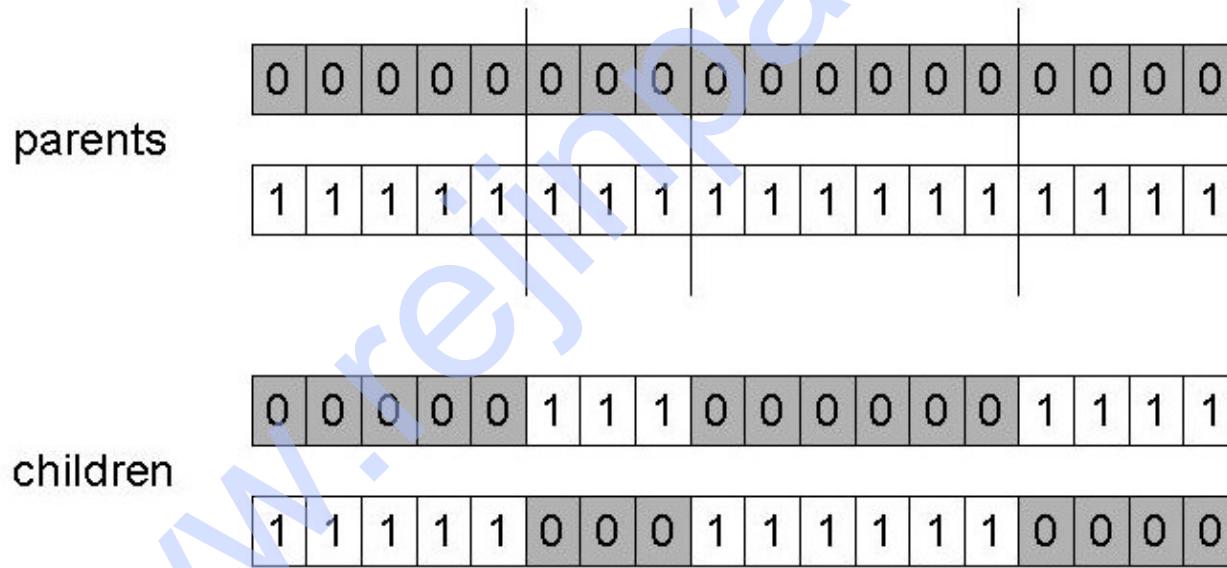
## Genetic Algorithm – Crossover (Single Point)

- Choose a random point on the two parents.
- Split parents at this crossover point.
- Create children by exchanging tails.



## Genetic Algorithm – Crossover (n Points)

- Choose n random crossover points.
- Split along those points.
- Glue parts, alternating between parents.
- Generalization of 1 point.



## Genetic Algorithm – Uniform Crossover

Generate uniformly random number.

X1 = 0 1 1 0 0 0 1 0 1 0

X2 = 1 1 0 0 0 0 0 1 1 1

Uniformly generated = 1 0 0 0 0 1 0 0 0

As a result, the new population becomes,

X1 = 1 1 1 0 0 0 0 0 1 0

X2 = 0 1 0 0 0 1 1 1 1

## Genetic Algorithm – Mutation

- Alter each gene independently with a probability  $p_m$
- $p_m$  is called the mutation rate
  - Typically between 1/pop\_size and 1/ chromosome\_length

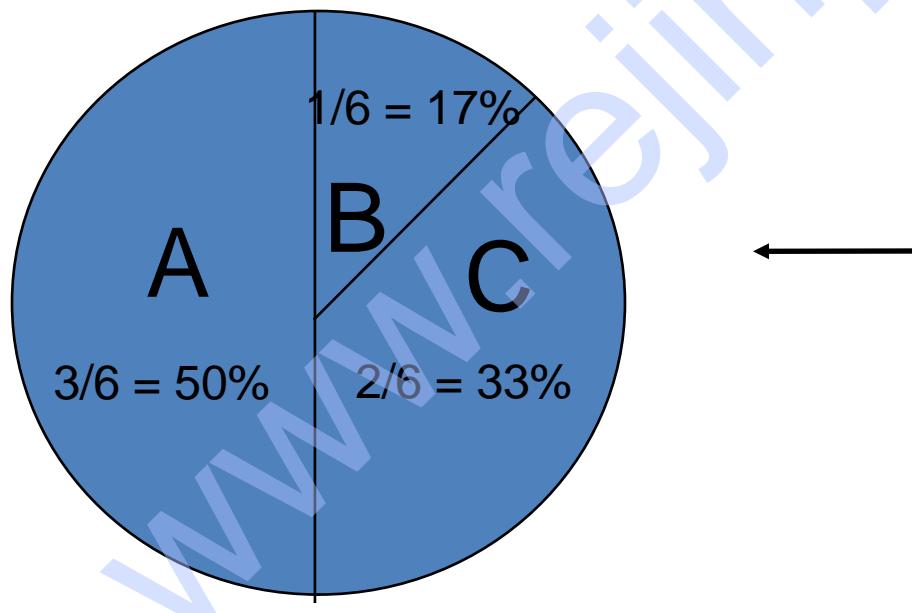
00010101 0011001 01111000  
00100100 10111010 11110000  
11000101 01011000 01101010  
11000101 01011000 01101010

00000000 00001000 00000010  
10000010 00000010 00000000  
00000000 00010000 00000000  
00010000 00000000 01000000

00010101 00110001 01111010  
10100110 10111000 11110000  
11000101 01111000 01101010  
11010101 01011000 00101010

## Genetic Algorithm – Selection

- Main idea: Better individuals get higher chance:
  - Chances are proportional to fitness.
  - Implementation: Roulette wheel technique
    - » Assign to each individual a part of the roulette wheel.
    - » Spin the wheel n times to select n individuals.



$$\text{fitness}(A) = 3$$

$$\text{fitness}(B) = 1$$

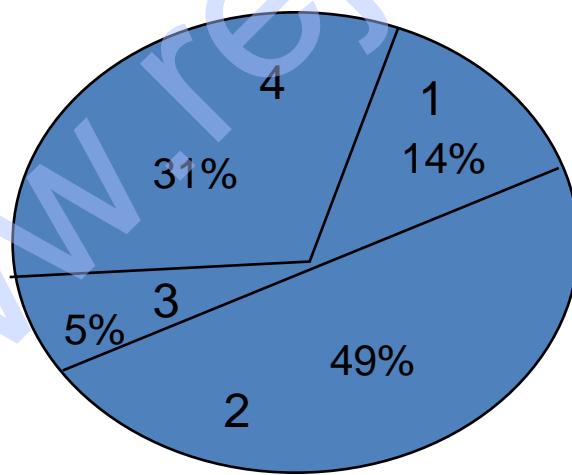
$$\text{fitness}(C) = 2$$

## Genetic Algorithm – An Example

- Simple problem: max  $x_2$  over  $\{0, 1, \dots, 31\}$
- GA approach:
  - Representation: binary code, e.g.  $01101 \leftrightarrow 13$
  - Population size: 4
  - 1-point xover, bitwise mutation
  - Roulette wheel selection
  - Random initialisation
- One generational cycle performed manually is shown here.

## Example : Selection

String no.	Initial population	$x$ Value	Fitness $f(x) = x^2$	$Prob_i$	Expected count	Actual count
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4
Average			293	0.25	1.00	1
Max			576	0.49	1.97	2



## Example : Crossover

String no.	Mating pool	Crossover point	Offspring after xover	$x$ Value	Fitness $f(x) = x^2$
1	0 1 1 0   1	4	0 1 1 0 0	12	144
2	1 1 0 0   0	4	1 1 0 0 1	25	625
2	1 1   0 0 0	2	1 1 0 1 1	27	729
4	1 0   0 1 1	2	1 0 0 0 0	16	256
Sum					1754
Average					439
Max					729

## Example : Mutation

String no.	Offspring after xover	Offspring after mutation	$x$ Value	Fitness $f(x) = x^2$
1	0 1 1 0 0	1 1 1 0 0	26	676
2	1 1 0 0 1	1 1 0 0 1	25	625
2	1 1 0 1 1	1 1 0 1 1	27	729
4	1 0 0 0 0	1 0 1 0 0	18	324
Sum				2354
Average				588.5
Max				729

## Simple Genetic Algorithm

- Has been subject of many (early) studies
  - still often used as benchmark for novel Gas.
- Shows many **shortcomings**, e.g.
  - Representation is too restrictive.
  - Mutation & crossovers only applicable for bit-string & integer representations.
  - Selection mechanism sensitive for converging populations with close fitness values.

## Comparison of GA with Traditional Optimization Techniques

- GA works with the coding of solution set and not with the solution itself.
- GA uses population of solutions rather than a single solution for searching.
- GA uses fitness function for evaluation rather the derivatives.
- GA uses probabilistic transition and not deterministic rules.

# APPLICATIONS OF GENETIC ALGORITHM

Job shop scheduling (Davern 1994)

Manufacturing cell design (Joiner 1996)

Fixture design (Gold 1998)

- Several Scheduling Problems,
- Traveling Salesman Problem,
- Energy Optimization and so on.

## References

- Holland, J. (1992), *Adaptation in natural and artificial systems* , 2nd Ed. Cambridge: MIT Press.
- Davis, L. (Ed.) (1991), *Handbook of genetic algorithms*. New York: Van Nostrand Reinhold.
- Goldberg, D. (1989), *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley.
- Fogel, D. (1995), *Evolutionary computation: Towards a new philosophy of machine intelligence*. Piscataway: IEEE Press.
- Bäck, T., Hammel, U., and Schwefel, H. (1997), 'Evolutionary computation: Comments on the history and the current state', *IEEE Trans. On Evol. Comp.* 1, (1)

## Online Resources

- <http://www.spectroscopynow.com>
- <http://www.cs.bris.ac.uk/~colin/evollect1/evollect0/index.htm\>
- IlliGAL (<http://www-illigal.ge.uiuc.edu/index.php3>)
- GAlib (<http://lancet.mit.edu/ga/>)

# GENETIC ALGORITHM MATLAB TOOLBOX

- The genetic algorithm is a method for solving both constrained and unconstrained optimization problems that is based on natural selection, the process that drives biological evolution.
- The genetic algorithm repeatedly modifies a population of individual solutions. At each step, the genetic algorithm selects individuals at random from the current population to be parents and uses them to produce the children for the next generation.
- Over successive generations, the population "evolves" toward an optimal solution.

The **Genetic Algorithm and Direct Search Toolbox** is a collection of functions that extend the capabilities of the Optimization Toolbox and the MATLAB numeric computing environment. The Genetic Algorithm and Direct Search Toolbox includes routines for solving optimization problems using

- GENETIC ALGORITHM
- DIRECT SEARCH

These algorithms enable you to solve a variety of optimization problems that lie outside the scope of the Optimization Toolbox.

The genetic algorithm uses **three main types of rules** at each step to create the next generation from the current population:

- **Selection** rules select the individuals, called parents, that contribute to the population at the next generation.
- **Crossover** rules combine two parents to form children for the next generation.
- **Mutation** rules apply random changes to individual parents to form children.

The genetic algorithm at the command line, call the genetic algorithm function ga with the syntax

**[*x fval*] = ga(@fitnessfun, *nvars*, *options*)** where  
**@fitnessfun** is a handle to the fitness function.

- **nvars** is the number of independent variables for the fitness function.
- **options** is a structure containing options for the genetic algorithm. If you do not pass in this argument, ga uses its default options.

The results are given by

- **x** — Point at which the final value is attained.
- **fval** — Final value of the fitness function.

# COMMAND LINE FUNCTIONS

**Bin2int**

- BINary string to INTeger string conversion

**bin2real**

- BINary string to REAL vector conversion

**bindecod**

- BINary DECODing to binary, integer or real numbers

**compdiv**

- COMPuter DIVerse things of GEA Toolbox

**compdiv2**

- COMPuter DIVerse things of GEA Toolbox

**compete**

- COMPETition between subpopulations

**comploc**

- COMPuter LOCal model things of toolbox

**compplot**

- COMPuter PLOT things of GEA Toolbox

**geemain2**

- MAIN function for Genetic and Evolutionary Algorithm toolbox for matlab

**initbp**

- CReaTe an initial Binary Population

**initip**

- CReaTe an initial (Integer value) Population

**mutate**

- high level MUTATION function

**mutbin**

- MUTation for BINary representation

**mutbmd**

- real value Mutation like Discrete Breeder genetic algorithm

**mutcomb**

- MUTation for combinatorial problems

**mutes1**

- MUTation by Evolutionary Strategies 1, derandomized Self Adaption

**mutexch**

- MUTation by eXChange

**mutint**

- MUTation for INTeger representation

**mutinvert**

- MUTation by INVERTing variables

**mutmove**

- MUTation by MOVEing variables

**mutrand**

- MUTation RANDOM

**mutrandbin**

- MUTation RANDOM of binary variables

**Mutrandbin**

- MUTation RANDOM of binary variables

**mutrandint**

- MUTation RANDOM of integer variables

**mutrandperm**

- MUTation RANDOM of binary variables

**Mutrandreal**

- MUTation RANDOM of real variables

**Rankgoal**

- perform goal preference calculation between multiple objective values

**ranking**

- RANK-based fitness assignment, single and multi objective, linear and nonlinear

**rankplt**

- RANK two multi objective values Partially Less Than

**rankshare**

- SHARING between individuals

**recdis**

- RECombination DIScrete

**recdp**

- RECombination Double Point

**recdprs**

- RECombination Double Point with Reduced Surrogate

**recgp**

- RECombination Generalized Position

**recint**

- RECombination extended INTermediate

**reclin**

- RECombination extended LIne

**reclinex**

- EXtended LIne RECombination

**recmp**

- RECombination Multi-Point, low level

**function recombin**

- high level RECOMBINation function

**recpm**

- RECombination Partial Matching

**recsh**

- RECombination SHuffle

**recsp**

- RECombination Single Point

# SELECTION FUNCTIONS

[www.rejinpaul.com](http://www.rejinpaul.com)

**selection**

- high level SELECTION function

**sellocal**

- SELECTION in a LOCAL neighbourhood

**selrws**

- SELECTION by Roulette Wheel Selection

**selsus**

- SELECTION by Stochastic Universal Sampling

**seltour**

- SELECTION by TOURNAMENT

**seltrunc**

- SELECTION by TRUNCATION

# OBJECTIVE FUNCTIONS

**initdopi**

- INITialization function for DOuble Integrator objdopi

**initfun1**

- INITialization function for de jong's FUNction 1

**mopfonseca1**

- MultiObjective Problem: FONSECA's function 1

**mopfonseca2**

- MultiObjective Problem: FONSECA's function 1

**moptest**

- MultiObjective function TESTing

**obj4wings**

- OBjective function FOUR-WINGS.

**objbran**

- OBjective function for BRANin rcos function

**objdopi**

- OBjective function for DOuble Integrator

**objeaso**

- OBjective function for EASom function

**objfletwell**

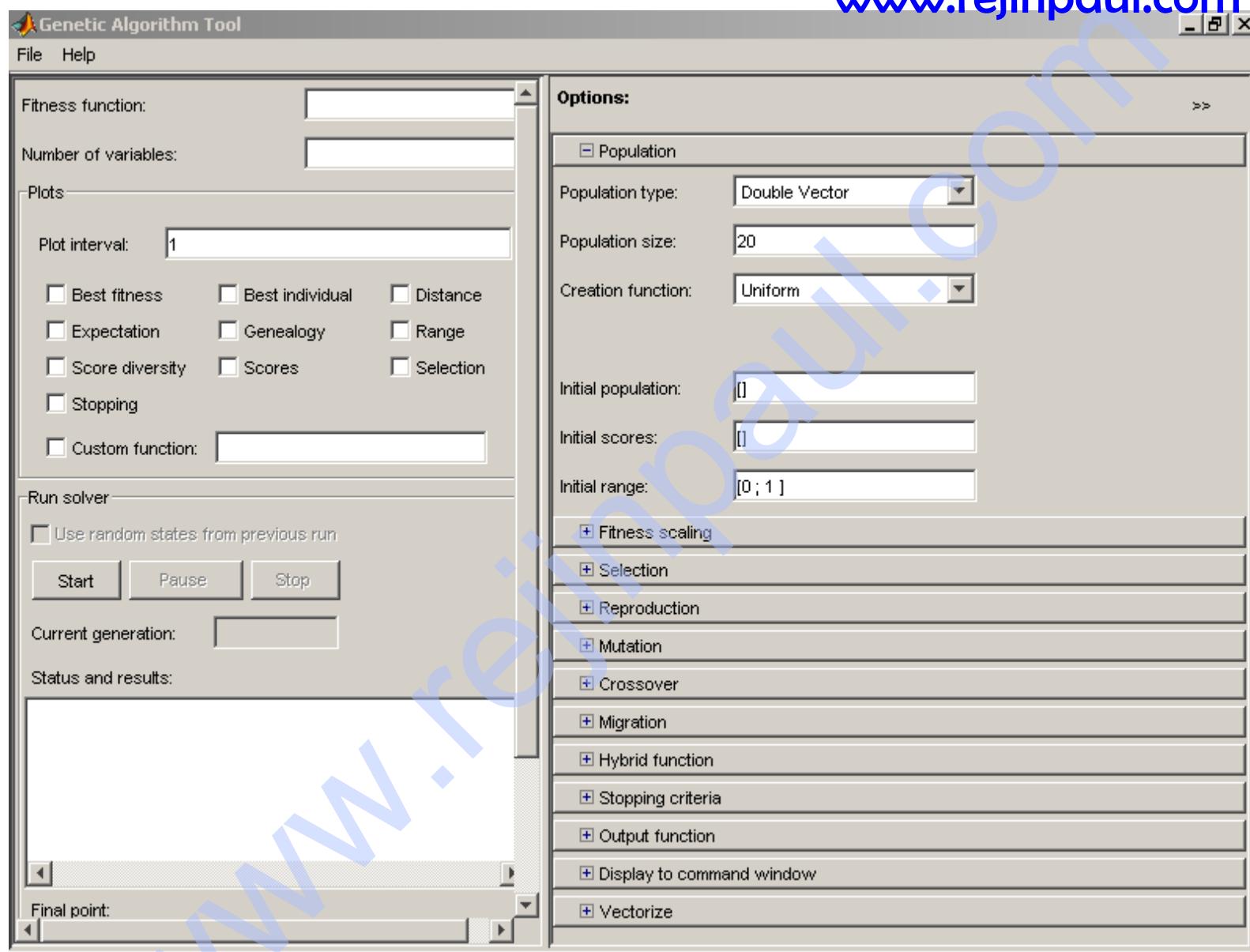
- OBjective function after FLETcher and PoWELL

## GA GRAPHICAL USER INTERFACE

The Genetic Algorithm Tool is a **graphical user interface** that enables you to use the genetic algorithm without working at the command line. To open the Genetic Algorithm Tool, enter

**Gatool**

at the MATLAB command prompt.



To use the Genetic Algorithm Tool, you must first enter the following information:

**Fitness function** — The objective function you want to minimize. Enter the fitness function in the form @fitnessfun, where fitnessfun.m is an M-file that computes the fitness function.

**Number of Variables** – The number of variables in the given fitness function should be given.

## PLOT OPTIONS

1. Best fitness
2. Best individual
3. Distance
4. Expectation
5. Genealogy
6. Range
7. Score Diversity
8. Scores
9. Selection
10. Stopping

## POPULATION

In this case population type, population size and creation function may be selected.

The initial population and initial score may be specified, if not, the 'Ga tool' creates them. The initial range should be given.

## FITNESS SCALING

The fitness scaling should be any of the following

- a. Rank
- b. Proportional
- c. Top
- d. Shift Linear
- e. Custom

# SELECTION

Selection function:

Stochastic uniform

Stochastic uniform

Remainder

Uniform

Roulette

Tournament

Custom

■ Reproduction

■ Mutation

# REPRODUCTION

Elite count:

2

Crossover fraction:

0.8

# MUTATION

Mutation

Mutation function: Gaussian

Scale: Gaussian

Uniform

Custom

Shrink: 1.0

Crossover

Crossover function: Scattered

Scattered

Single point

Two point

Intermediate

Heuristic

Custom

Migration

Hybrid function

Stopping criteria

Output & Logging

# CROSSOVER

# Migration



## Hybrid function

Hybrid function:

None

None

fminsearch

patternsearch

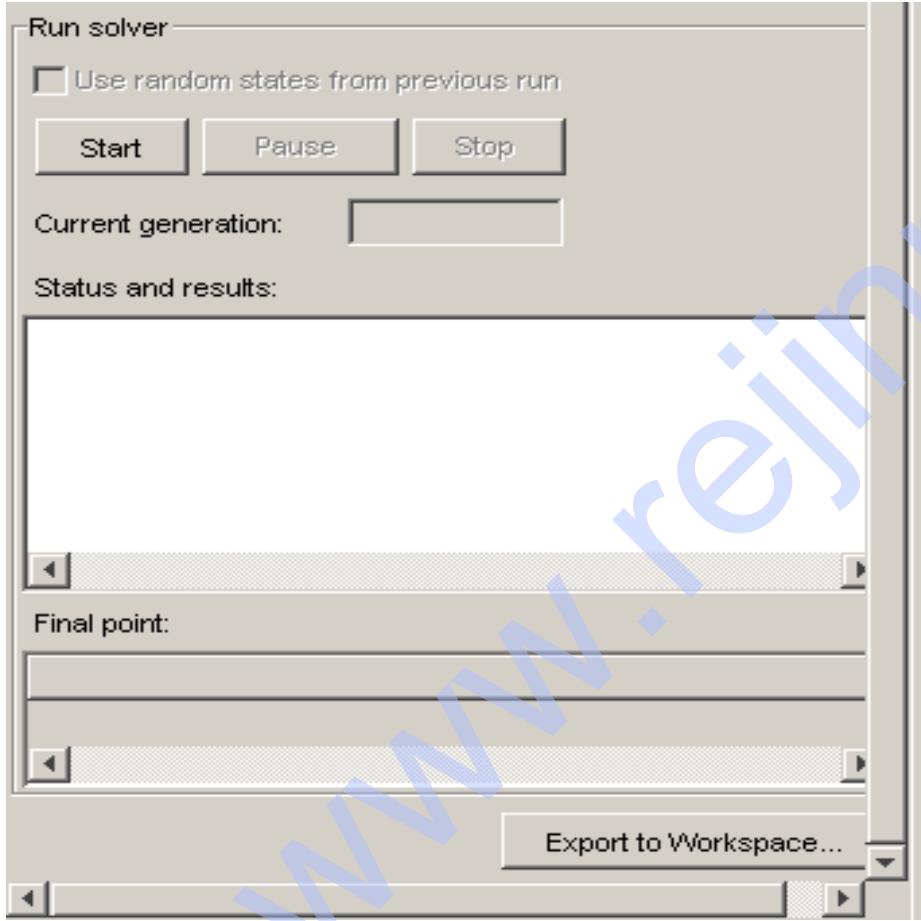
fminunc

## Stopping criteria

## Output function

**HYBRID FUCNTION**

# STOPPING CONDITION



www.rejinpaul.com

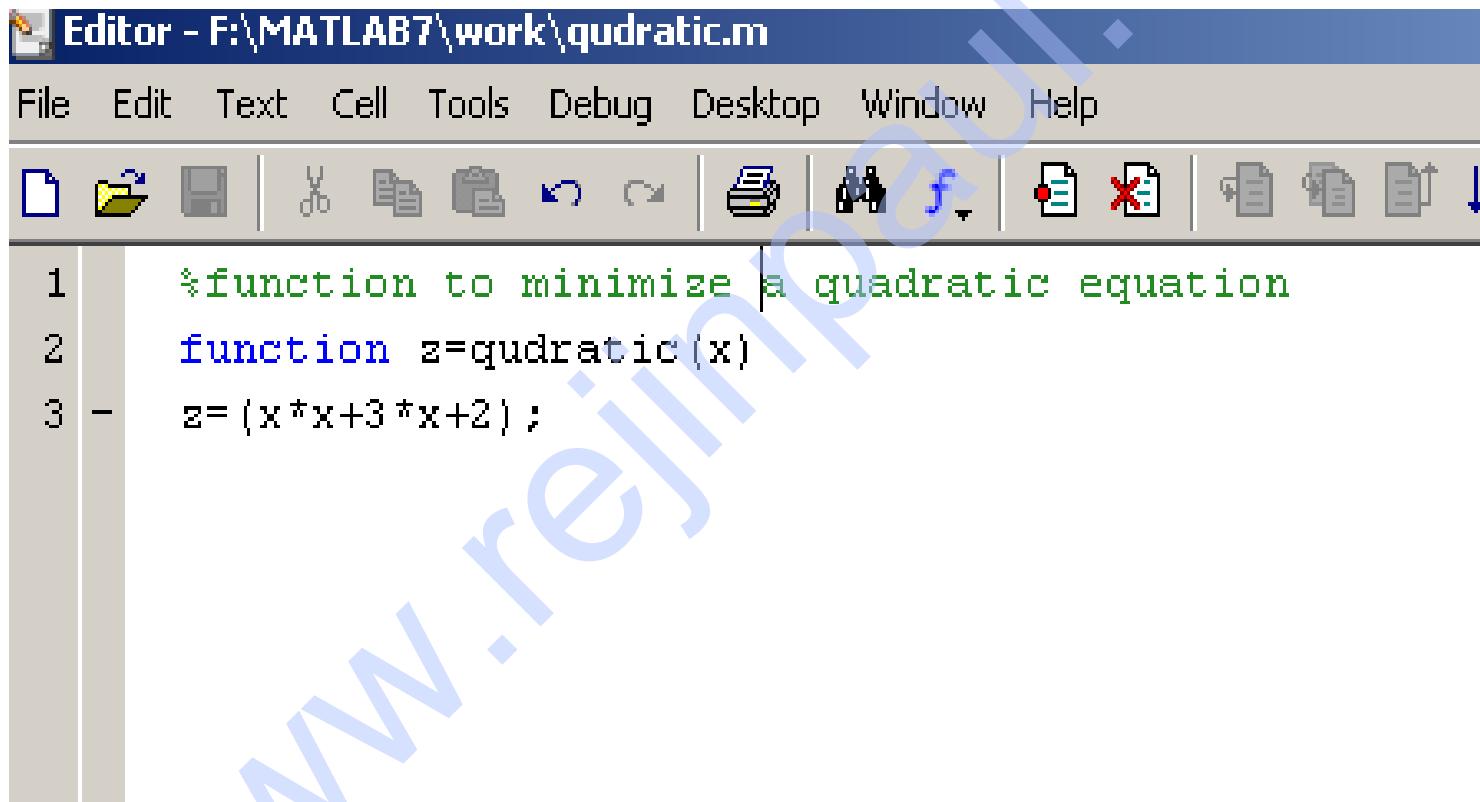
Stopping criteria

Generations:	100
Time limit:	Inf
Fitness limit:	-Inf
Stall generations:	50
Stall time limit:	20

## RUNNING AND SIMULATION

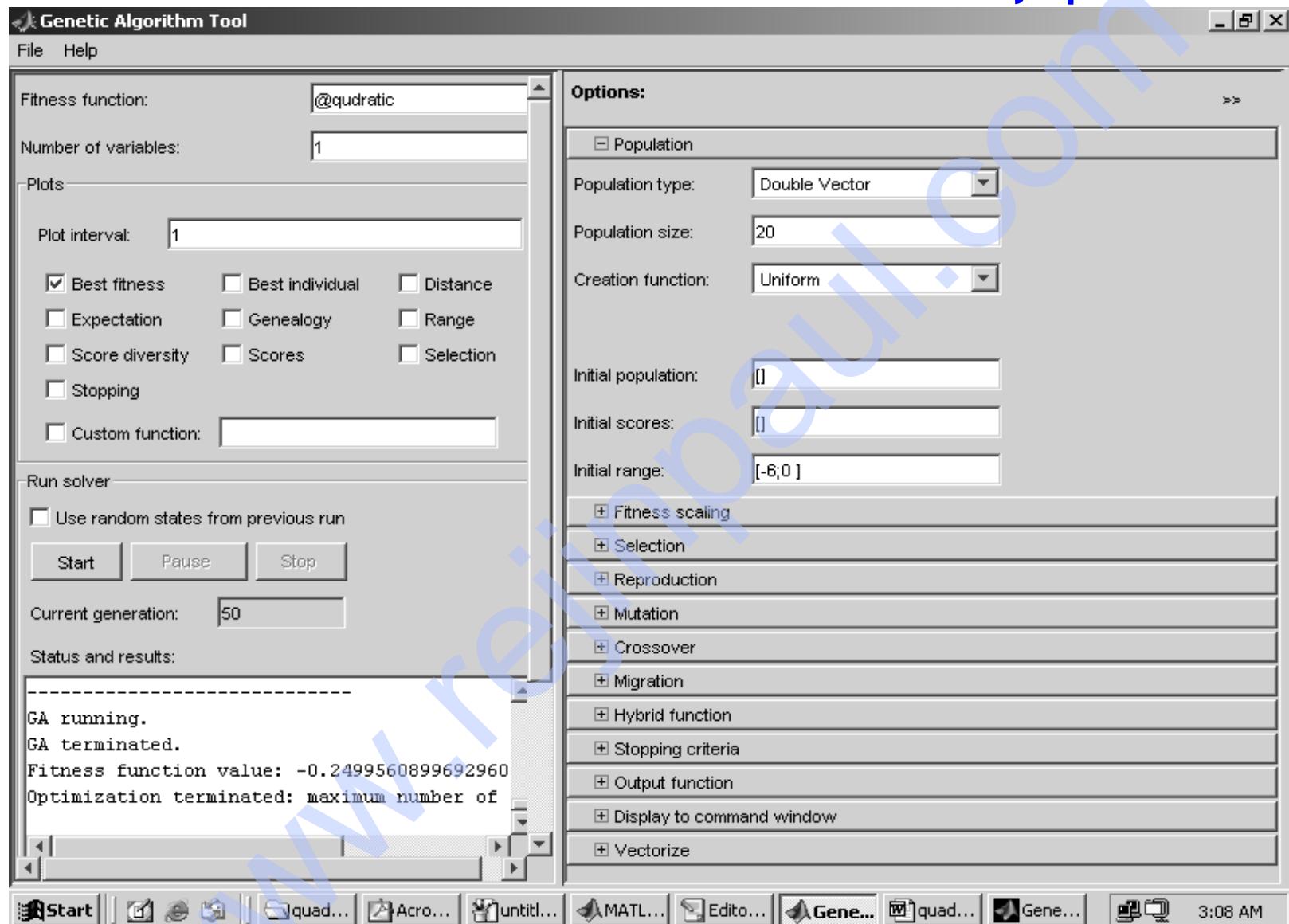
## EXAMPLE 1 – FUCNTION WITH SINGLE VARIABLE

Define the given function  $f(x) = x^2+3x+2$  in a separate m-file



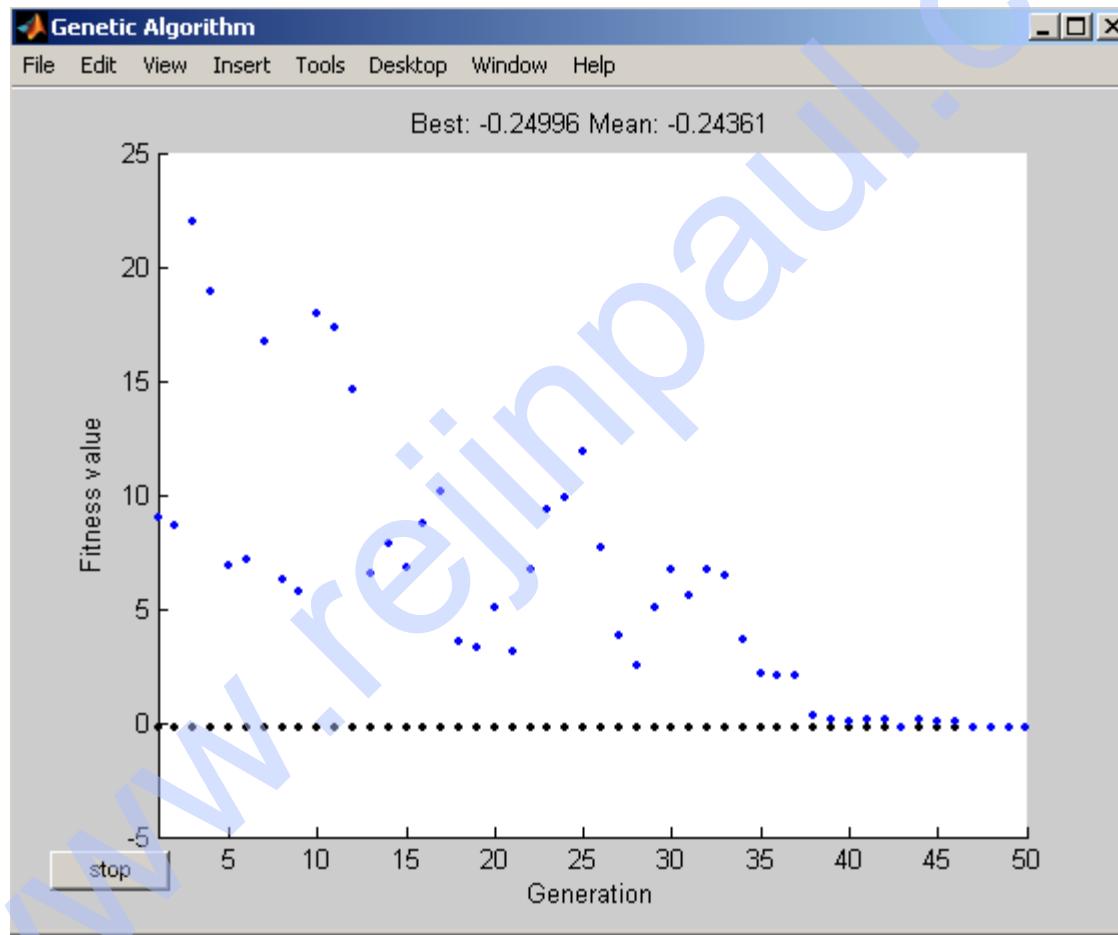
The screenshot shows the MATLAB Editor window with the title bar "Editor - F:\MATLAB7\work\quadratic.m". The menu bar includes File, Edit, Text, Cell, Tools, Debug, Desktop, Window, and Help. The toolbar below has various icons for file operations like Open, Save, Copy, Paste, and Undo. The code editor displays the following MATLAB script:

```
1 %function to minimize a quadratic equation
2 function z=quadratic(x)
3 - z=(x*x+3*x+2);
```

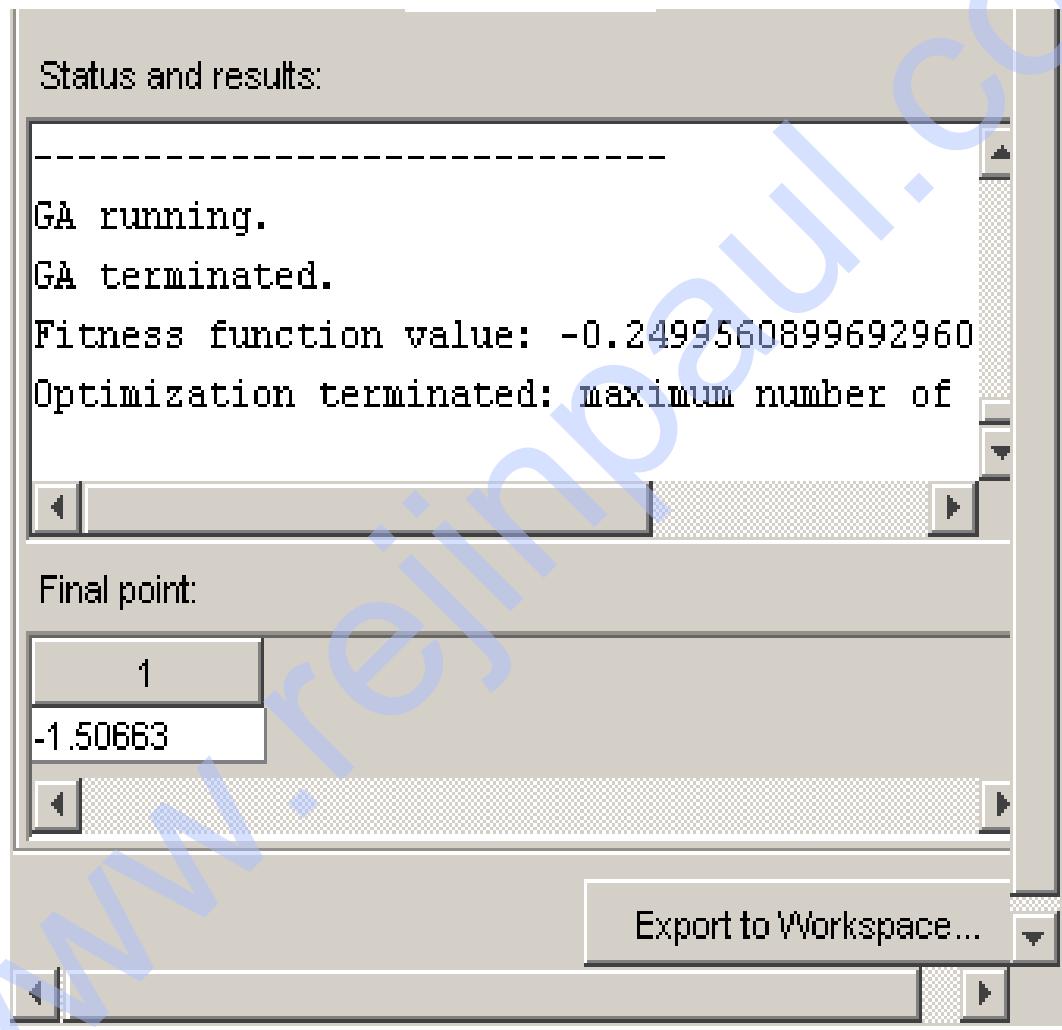


# OUTPUT

The output showing the best fitness for 50 generations.

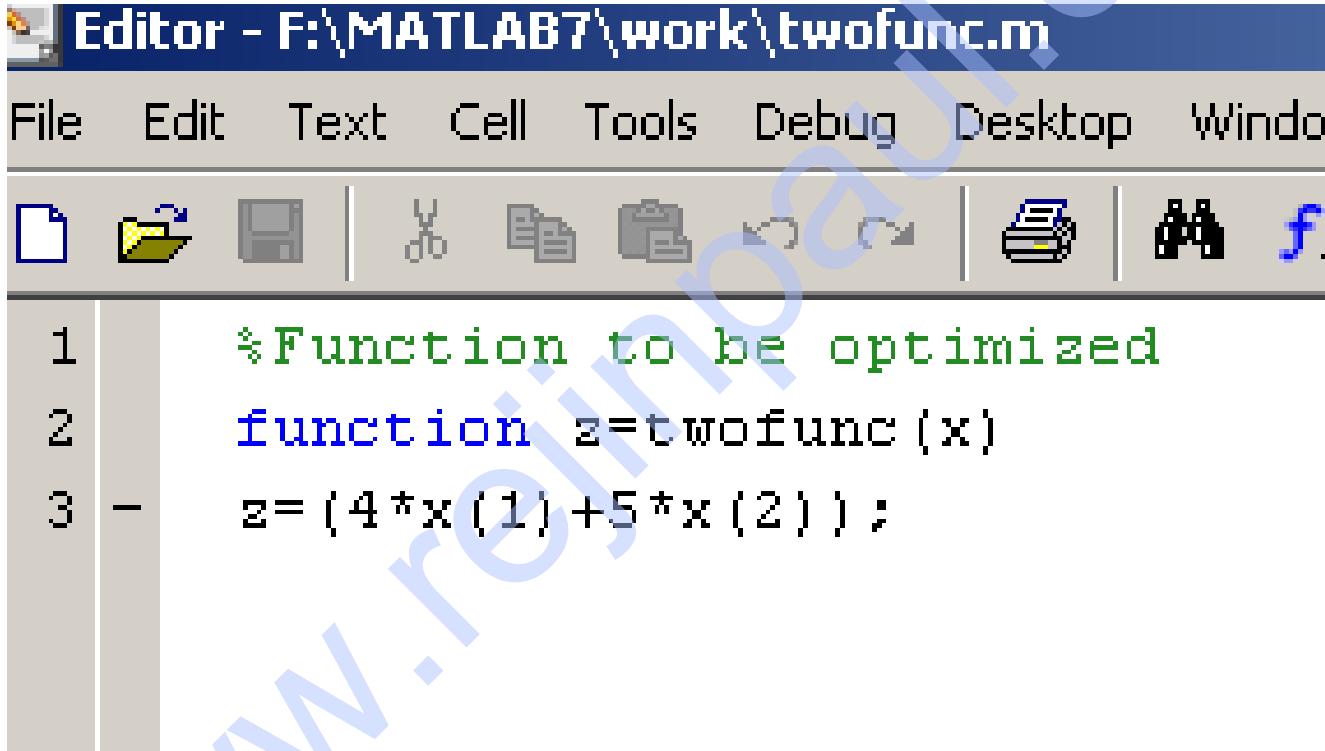


The status and results for this functions for 50 generations



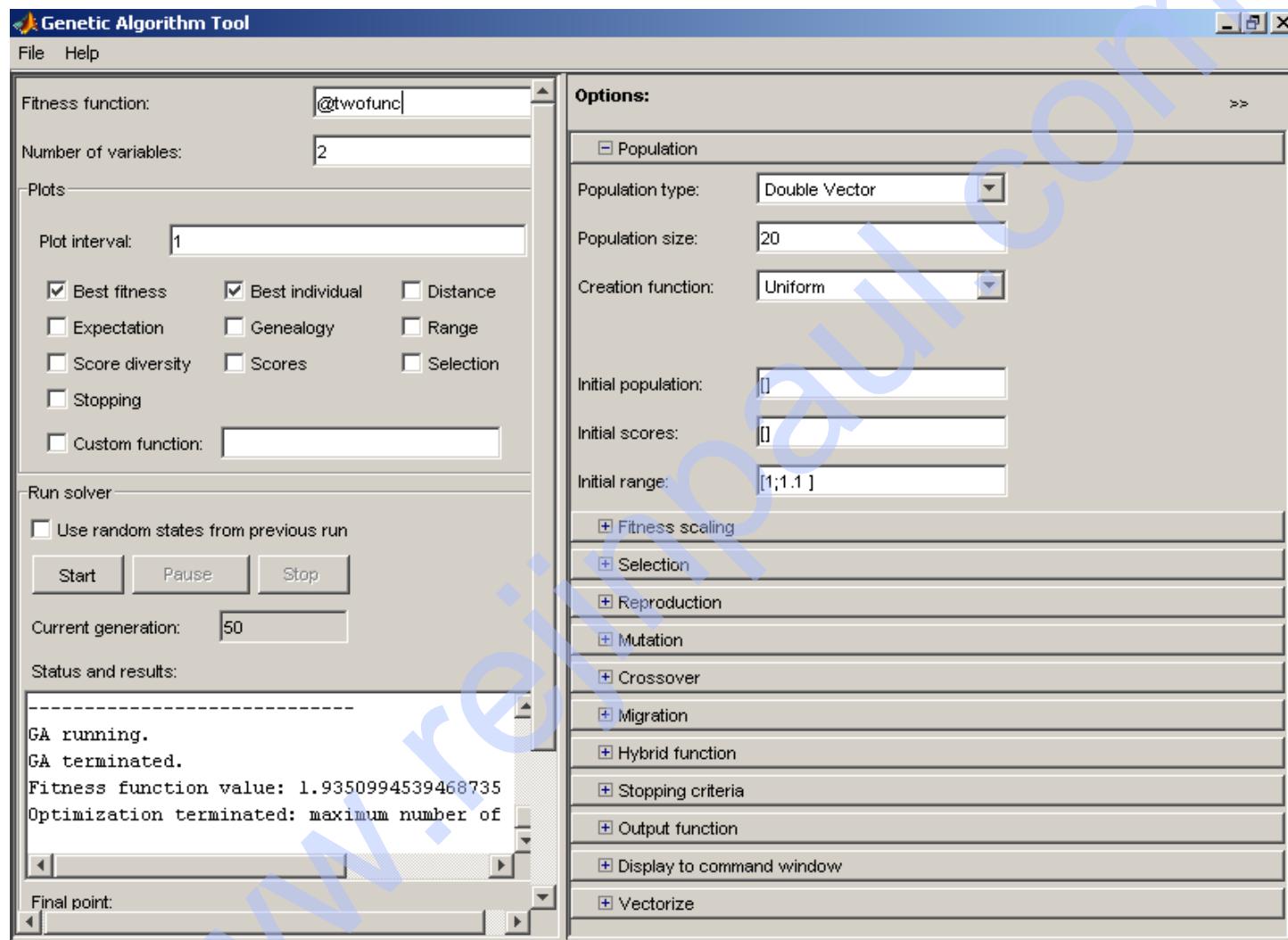
## EXAMPLE 2 – FUNCTION WITH TWO VARIABLES

Define the given function  $f(x_1, x_2) = 4x_1 + 5x_2$  in a separate m-file.



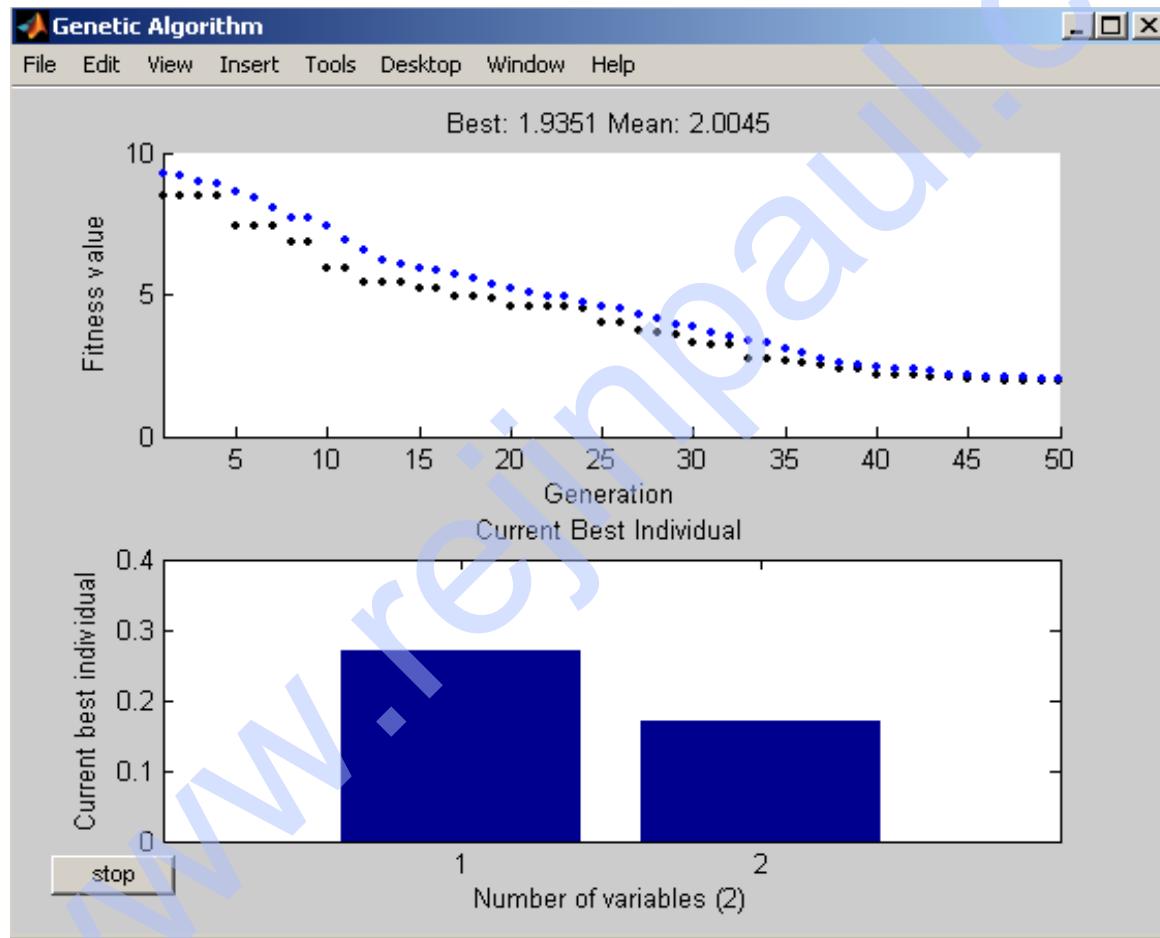
The screenshot shows the MATLAB Editor window with the title bar "Editor - F:\MATLAB7\work\twofunc.m". The menu bar includes File, Edit, Text, Cell, Tools, Debug, Desktop, and Windows. The toolbar below has icons for New, Open, Save, Copy, Paste, Find, and others. The code area contains three lines of MATLAB code:

```
1 %Function to be optimized
2 function z=twofunc(x)
3 - z=(4*x(1)+5*x(2));
```

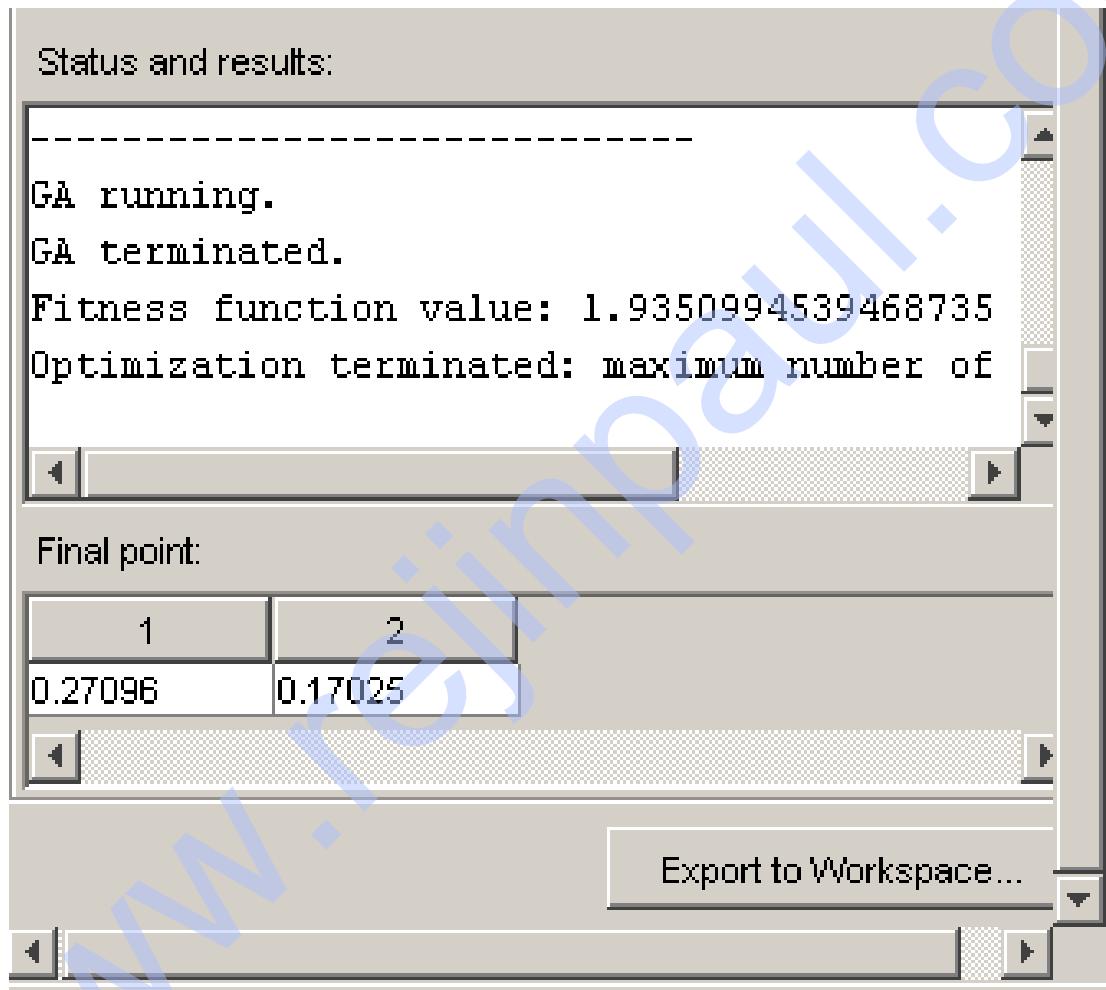


# OUTPUT

The output for 50 generations.

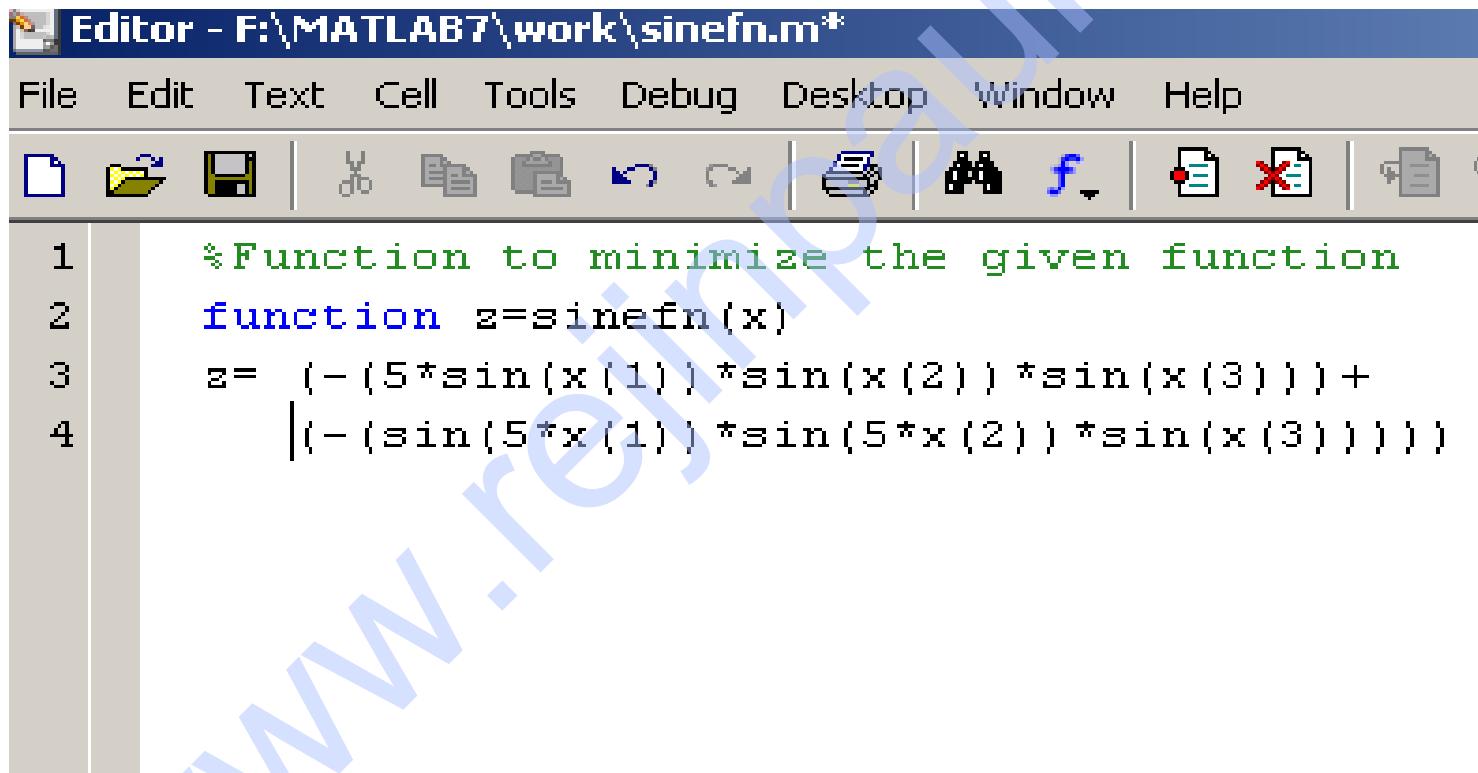


The status and result for this function.



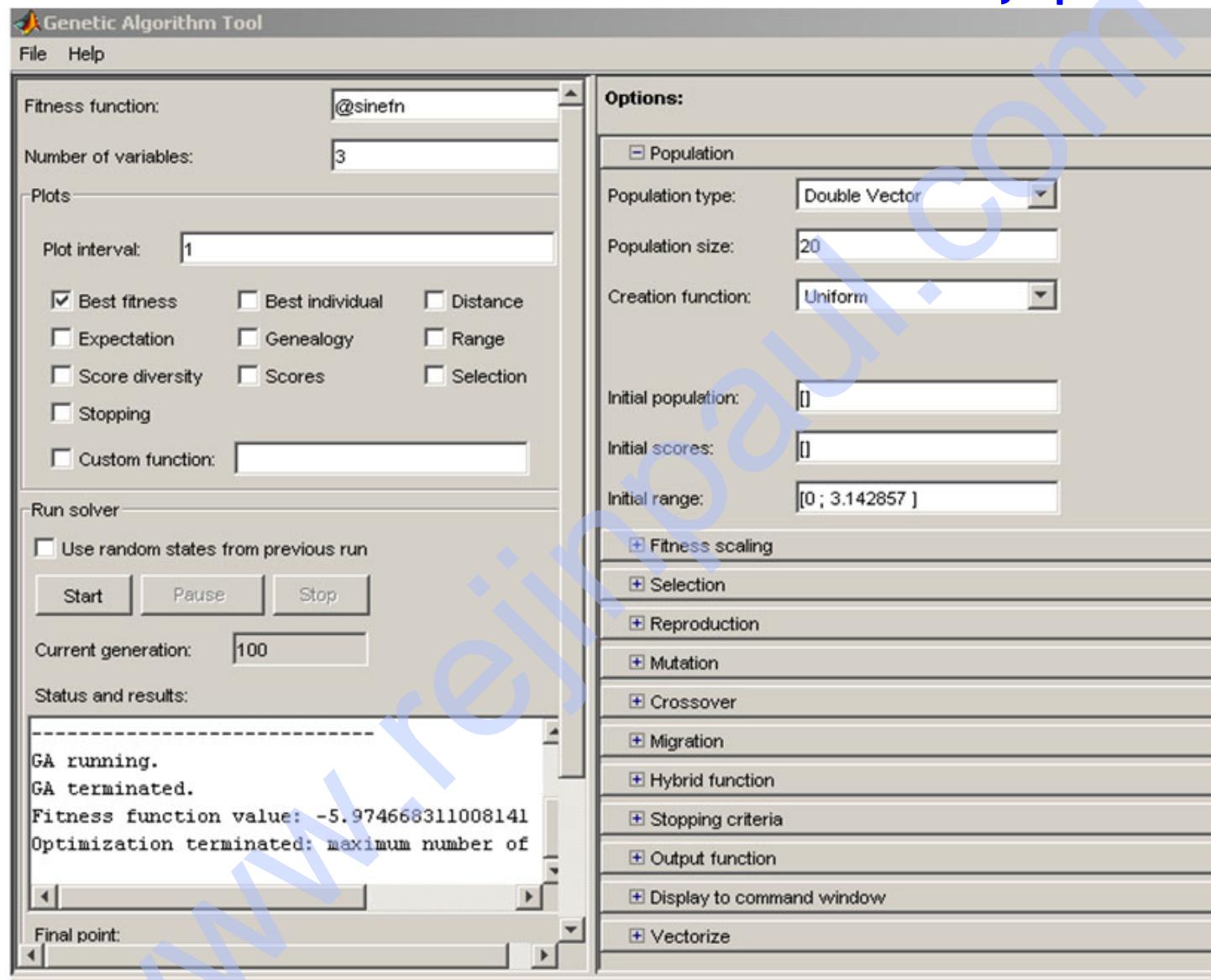
## EXAMPLE 3 – SINE FUNCTION

$f(x_1, x_2, x_3) = -5\sin(x_1)\sin(x_2)\sin(x_3) + -\sin(5x_1)\sin(5x_2)\sin(x_3)$   
where  $0 \leq x_i \leq \pi$ , for  $1 \leq i \leq 3$ .



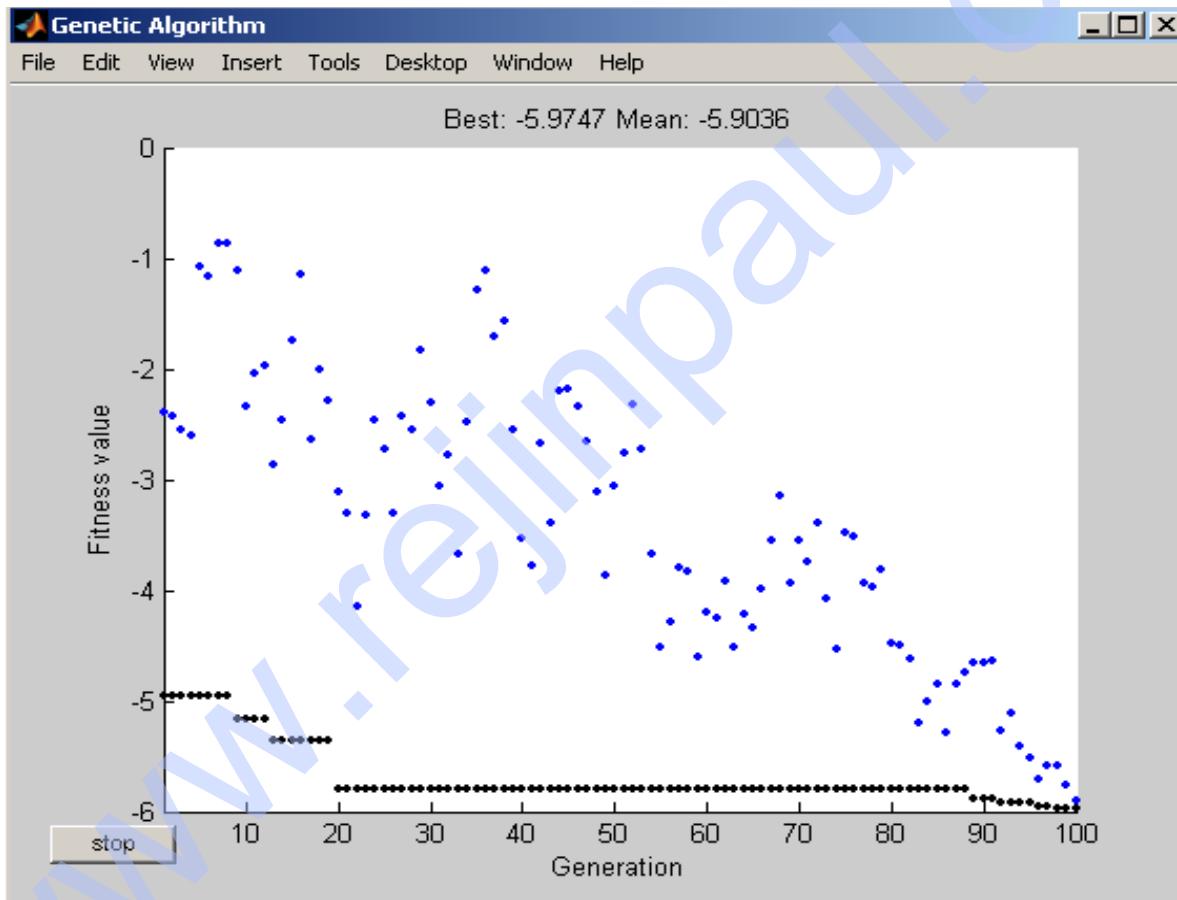
The screenshot shows the MATLAB Editor window with the file `sinefn.m*` open. The code defines a function to minimize the given function  $f(x_1, x_2, x_3)$ . The code is as follows:

```
1 %Function to minimize the given function
2 function z=sinefn(x)
3 z= (- (5*sin(x(1))*sin(x(2))*sin(x(3))) +
4      |(- (sin(5*x(1))*sin(5*x(2))*sin(x(3)))) )
```

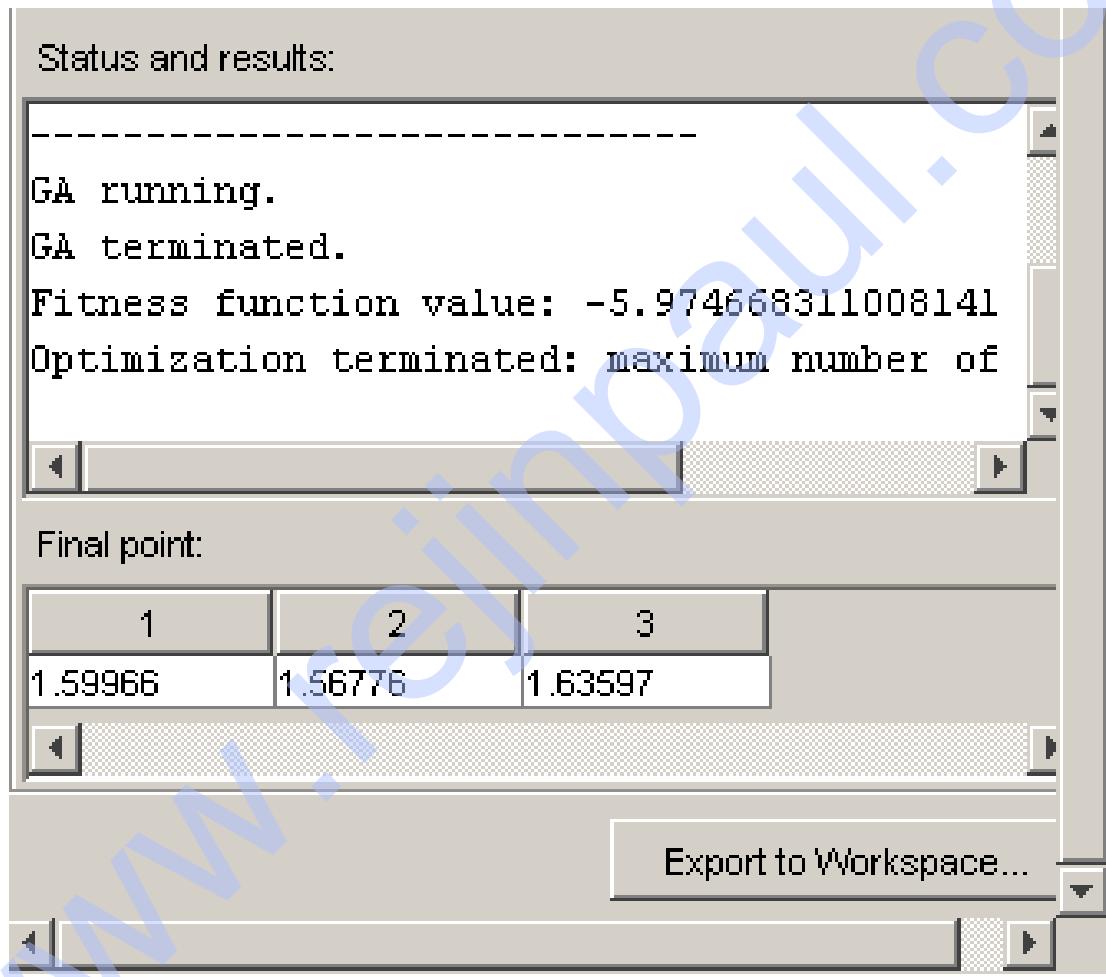


# OUTPUT

The output for 100 generations.



The status and result for this function



# UNIT –V

# HYBRID SOFT COMPUTING TECHNIQUES

## HYBRID SYSTEMS

### Neural Network Systems

Neural networks are the simplified models of the human nervous systems mimicking our ability to adapt to certain situations and to learn from the past experiences.

### Fuzzy Logic

Fuzzy logic or fuzzy systems deal with uncertainty or vagueness existing in a system and formulating fuzzy rules to find a solution to problems.

### Genetic Algorithm

Genetic algorithms inspired by the natural evolution process are adaptive search and optimization algorithms.

The main aim of the concept of hybridization is to overcome the weakness in one technique while applying it and bringing out the strength of the other technique to find solution by combining them.

Neural networks are good at recognizing patterns but they are not good at explaining how they reach their decisions.

On the contrary, fuzzy logic is good at explaining the decisions but cannot automatically acquire the rules used for making the decisions. Also, the tuning of membership functions becomes an important issue in fuzzy modeling. Genetic algorithms offer a possibility to solve this problem.

These limitations act as a central driving force for the creation of hybrid soft computing systems where two or more techniques are combined in a suitable manner that overcomes the limitations of individual techniques.

The use of hybrid systems is growing rapidly with successful applications in areas such as

- engineering design
- stock market analysis and prediction
- medical diagnosis
- process control
- credit card analysis and
- few other cognitive simulations.

## VARIOUS HYBRID SYSTEMS

The following three different hybrid systems are :

- Neuro fuzzy hybrid system;
- Neuron genetic hybrid system;
- Fuzzy genetic hybrid systems.

# NEURO FUZZY HYBRID SYSTEMS

## Definition:

A neuro-fuzzy hybrid system (also called fuzzy neural hybrid) is a learning mechanism that utilizes the training and learning algorithms from neural networks to find parameters of a fuzzy system.

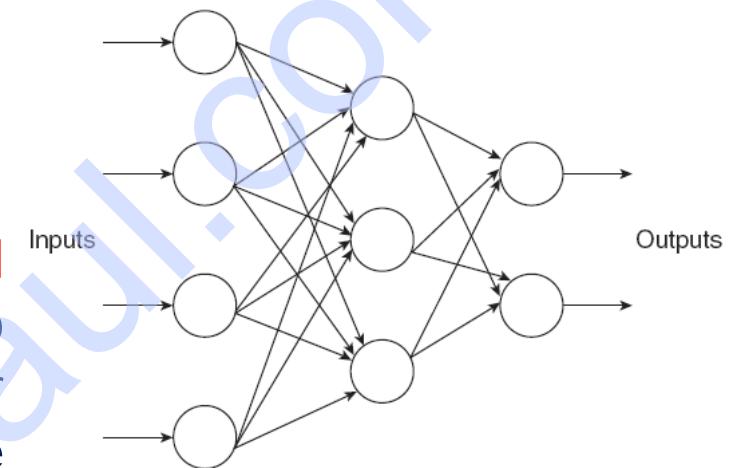
## Advantages of neuro fuzzy hybrid systems:

- It can handle any kind of information (numeric, linguistic, logical, etc.).
- It can manage imprecise, partial, vague or imperfect information.
- It can resolve conflicts by collaboration and aggregation.
- It has self-learning, self-organizing and self-tuning capabilities.
- It doesn't need prior knowledge of relationships of data.

# ARCHITECTURE OF NEURO FUZZY HYBRID SYSTEMS

The general architecture of neuro-fuzzy hybrid system

The architecture is a three-layer feed forward neural network model. It can also be observed that the first layer corresponds to the input variables, and the second and third layers correspond to the fuzzy rules and output variables, respectively. The fuzzy sets are converted to (fuzzy) connection weights.



## TYPES OF NEURO FUZZY HYBRID SYSTEMS

NFSs can be classified into the following two systems:

1. Cooperative NFSs.
2. General neuro-fuzzy hybrid systems.

# CO-OPERATIVE NEURO FUZZY SYSTEMS

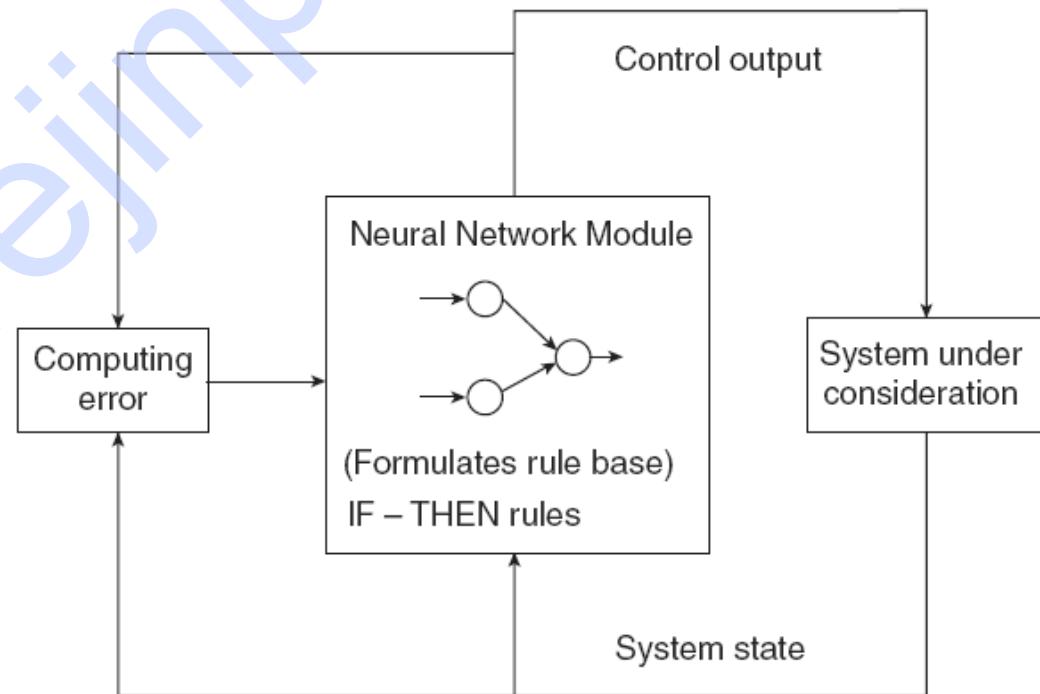
www.rejinpaul.com

In this type of system, both **artificial neural network (ANN)** and **fuzzy system** work independently from each other. The ANN attempts to learn the parameters from the fuzzy system.

## GENERAL NEURO FUZZY HYBRID SYSTEMS

General **neuro-fuzzy hybrid systems** (NFHS) resemble neural networks where a fuzzy system is interpreted as a neural network of special kind.

The rule base of a fuzzy system is assumed to be a neural network; the fuzzy sets are regarded as weights and the rules and the input and output variables as neurons.



# GENERAL NEURO HYBRID SYSTEMS

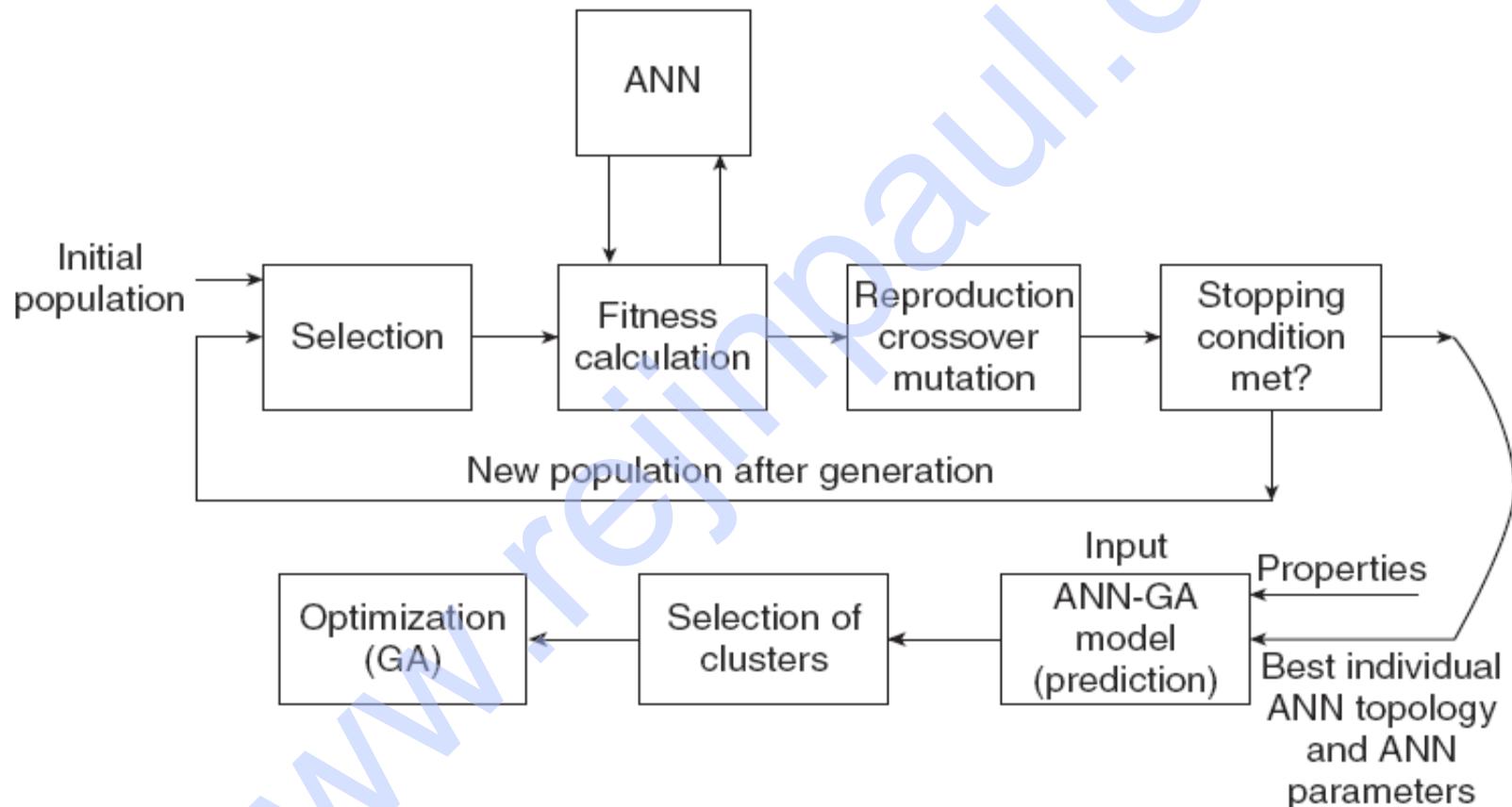
## Definition:

neuro-genetic hybrid or a **genetic\_neuro-hybrid system** is one in which a neural network employs a genetic algorithm to optimize its structural parameters that define its architecture.

## Properties of genetic neuro-hybrid systems:

1. The parameters of neural networks are encoded by genetic algorithms as a string of properties of the network, that is, chromosomes. A large population of chromosomes is generated, which represent the many possible parameter sets for the given neural network.
2. Genetic Algorithm \_Neural Network, or GANN, has the ability to locate the neighborhood of the optimal solution quickly, compared to other conventional search strategies.

# BLOCK DIAGRAM OF GENETIC NEURO HYBRID SYSTEMS



# ADVANTAGES OF GENETIC NEURO HYBRID SYSTEMS

The various **advantages** of neuro-genetic hybrid are as follows:

- GA performs optimization of neural network parameters with simplicity, ease of operation, minimal requirements and global perspective.
- GA helps to find out complex structure of ANN for given input and the output data set by using its learning rule as a fitness function.
- Hybrid approach ensembles a powerful model that could significantly improve the predictability of the system under construction.

# **GENETIC FUZZY HYBRID SYSTEMS**

[www.rejinpaul.com](http://www.rejinpaul.com)

The hybridization of genetic algorithm and fuzzy logic can be performed in the following two ways:

1. By the use of fuzzy logic based techniques for improving genetic algorithm behavior and modeling GA components. This is called fuzzy genetic algorithms (FGAs).
2. By the application of genetic algorithms in various optimization and search problems involving fuzzy systems.

## **ADVANTAGES OF GENETIC FUZZY HYBRID SYSTEMS**

- GAs allow us to represent different kinds of structures, such as weights, features together with rule parameters, etc., allowing us to code multiple models of knowledge representation. This provides a wide variety of approaches where it is necessary to design specific genetic components for evolving a specific representation.
- Genetic algorithm efficiently optimizes the rules, membership functions, DB and KB of fuzzy systems. The methodology adopted is simple and the fittest individual is identified during the process.

# SOFT COMPUTING APPLICATIONS

- A Fusion Approach of Multispectral Images with SAR (Synthetic Aperture Radar) Image for Flood Area Analysis.
- Optimization of Traveling Salesman Problem using Genetic Algorithm Approach.
- Genetic Algorithm based Internet Search Technique.
- Soft Computing Based Hybrid Fuzzy Controllers.
- Soft Computing Based Rocket Engine Control.

## Soft Computing ...

- Soft computing employs NN, SVM, FL etc, in a complementary rather than a competitive way.
- The real world problems are pervasively imprecise and uncertain
- Precision and certainty carry a cost

Exploit the tolerance for imprecision, uncertainty, partial truth, and approximation to achieve tractability, robustness and low solution cost.