

# **Part -1**

## **INTRODUCTION TO SOFT COMPUTING**

# SOFT COMPUTING (SC)

---

## 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

# CONTINUE...

---

- × Soft computing is a tolerance of following
- × Imprecise
- × Uncertainty
- × Partial truth
- × Approximation
- × Role model of soft computing is Human mind

# PROBLEM SOLVING TECHNIQUES

## HARD COMPUTING

**Precise Models**

Symbolic  
Logic  
Reasoning

Traditional  
Numerical  
Modeling and  
Search

## SOFT COMPUTING

**Approximate Models**

Approximate  
Reasoning

Functional  
Approximation  
and Randomized  
Search

# CONSTITUENTS OF SC

---

- ✘ Fuzzy System : Reasoning and imprecision
- ✘ Neural Network : Learning
- ✘ Evolutionary Computing (Genetic Algorithm)  
: Searching and Optimization

# APPLICATIONS OF SOFT COMPUTING

---

- Handwriting Recognition
- Image Processing and Data Compression
- Automotive Systems and Manufacturing
- Decision-support Systems
- Power Systems
- Fuzzy Logic Control
- Machine Learning Applications
- Speech and Vision Recognition Systems
- Process Control and So on ...

# Part -2

# GENETIC ALGORITHM

---

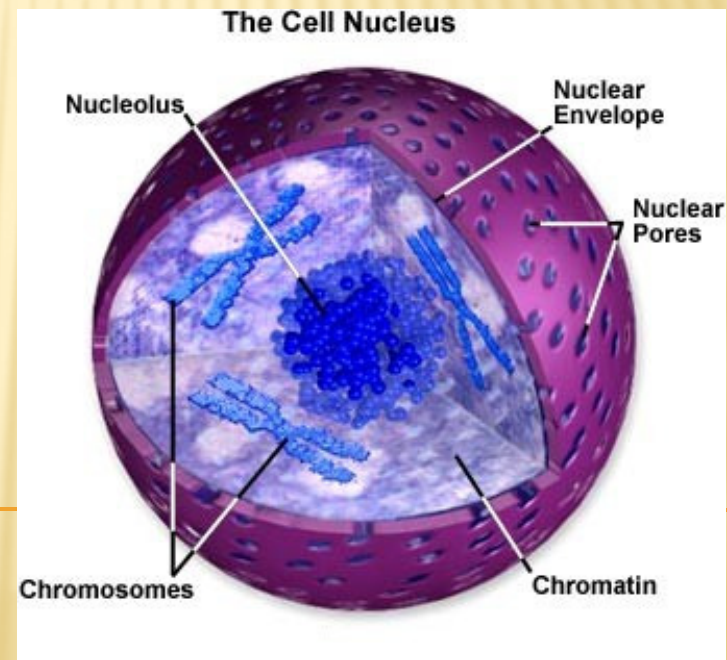
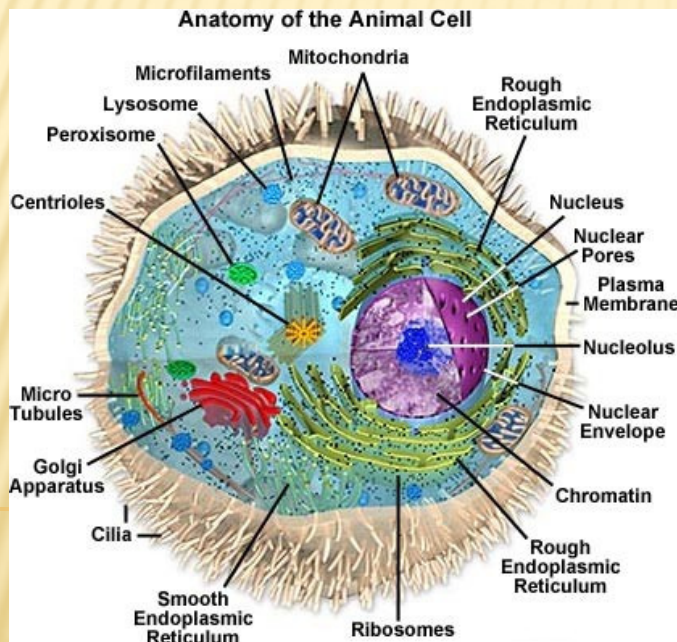
# General Introduction to GAs

- 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.
-



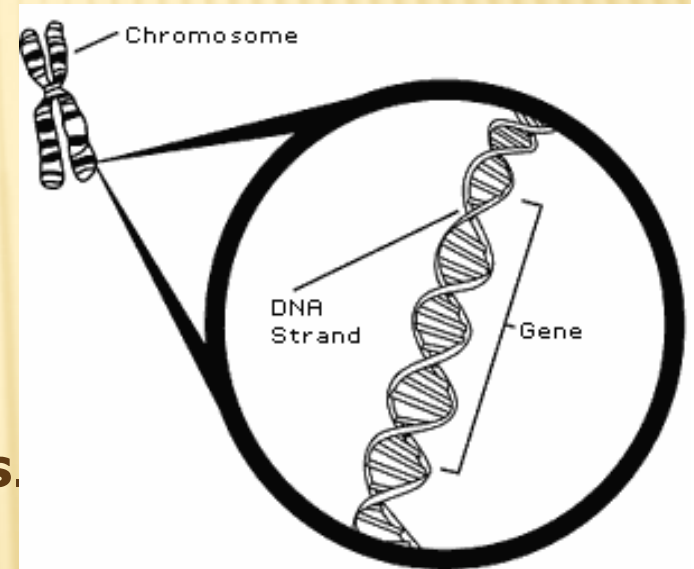
# Biological Background (1) – The Cell

- 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.



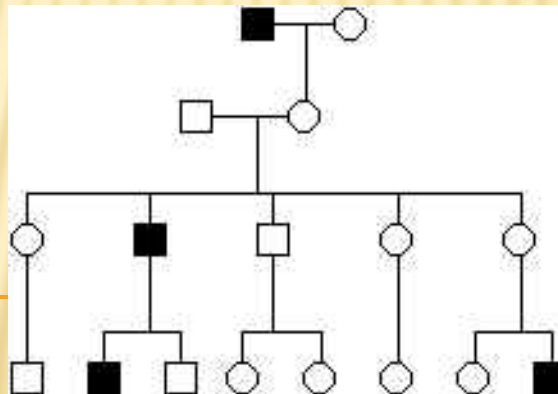
## Biological Background (2) – Chromosomes

- 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**.



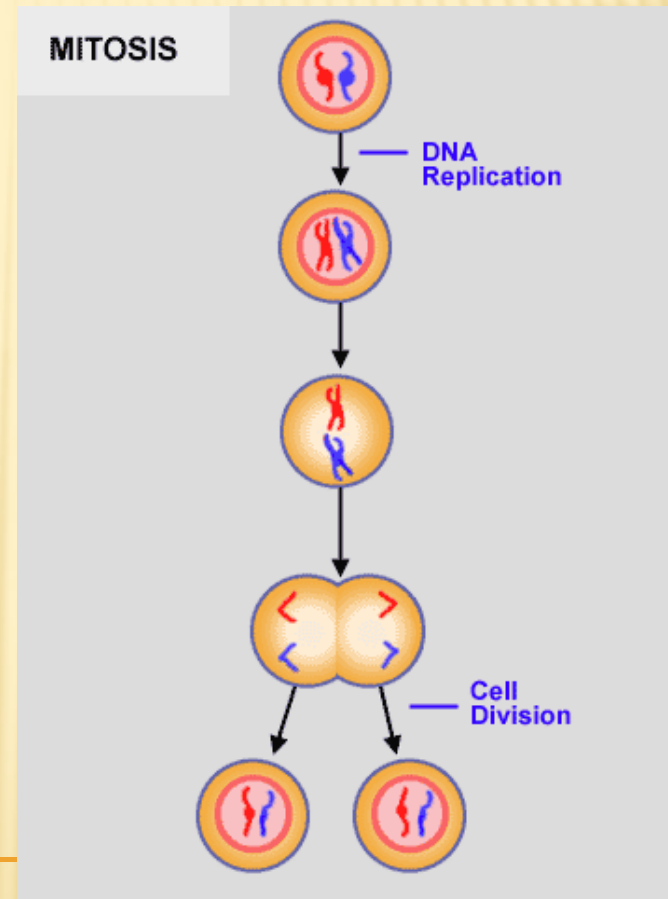
## Biological Background (3) – Genetics

- 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.



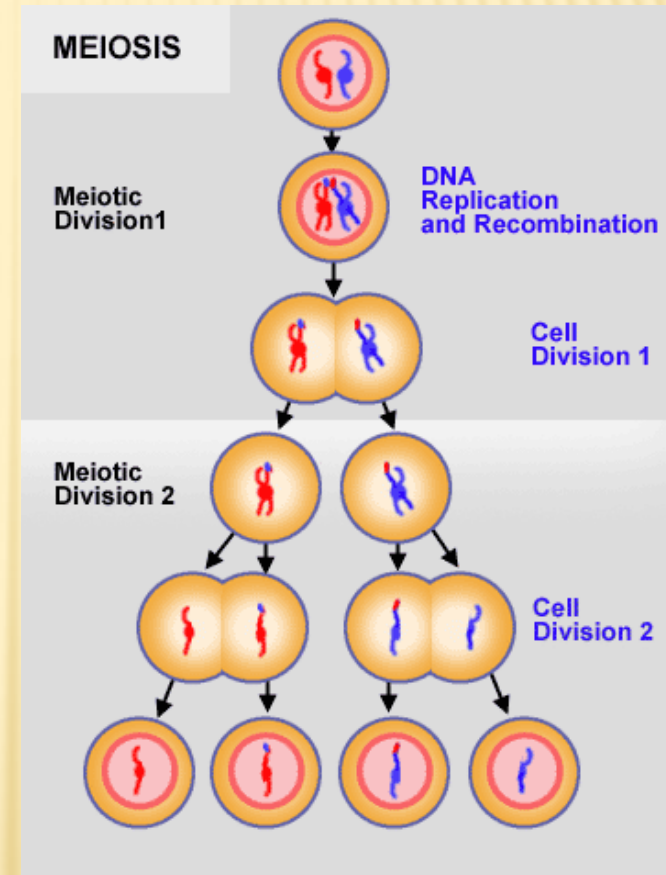
# Biological Background (4) – Reproduction

- 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 survive: **survival of the fittest**. For example, Giraffes with long necks.
  - Genetic variations due to crossover and mutation.
-

# Comparison of Natural and GA Terminology

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

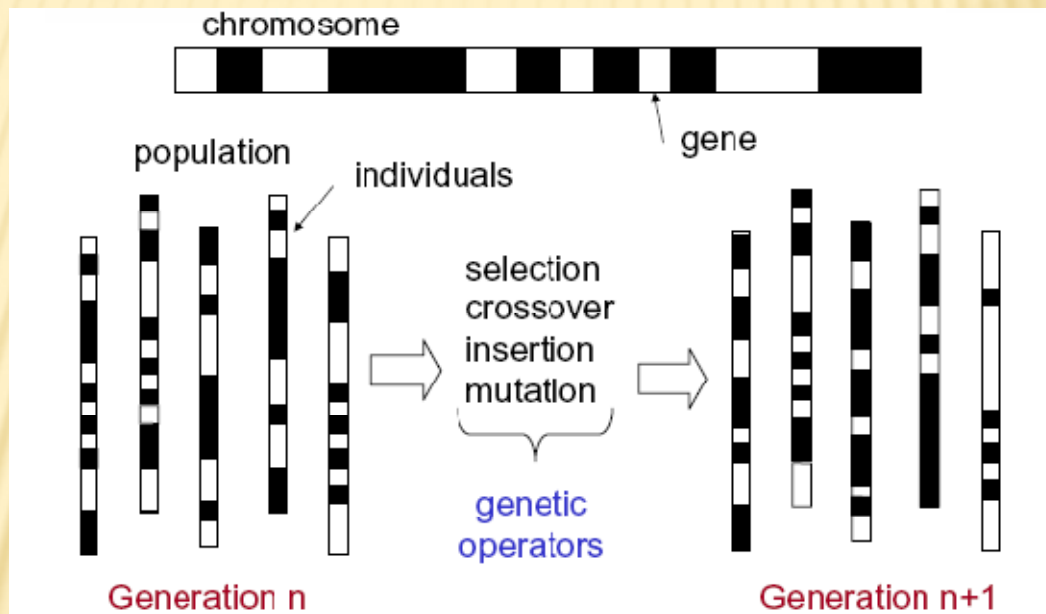
# PRINCIPLE OF NATURAL SELECTION

---

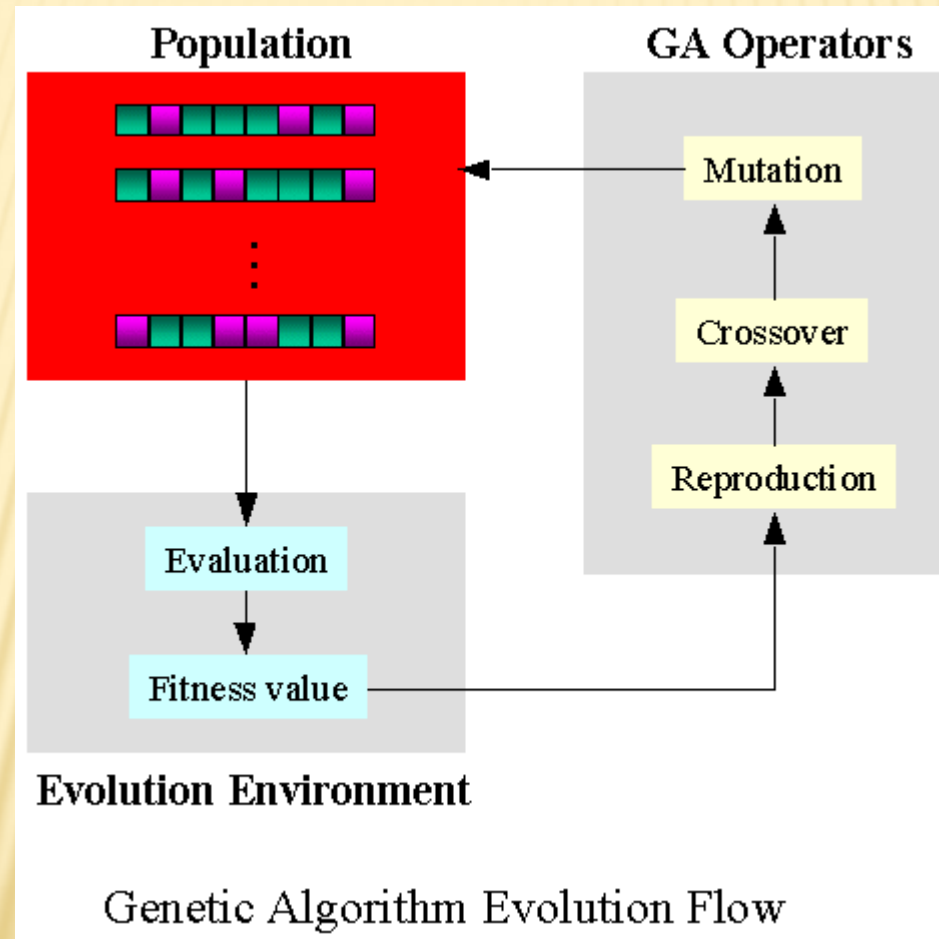
- ✘ *“Select The Best, Discard The Rest”*
- ✘ Two important elements required for any problem before a genetic algorithm can be used for a solution are:
  - ✘ Method for representing a solution (encoding)  
ex: string of bits, numbers, character
  - ✘ Method for measuring the quality of any proposed solution, using fitness function  
ex: Determining total weight



# GA ELEMENTS



# GA OPERATORS



# BASIC GA OPERATORS

---

**Selection** – To select set of chromosomes

**Crossover** - Looking for solutions near existing solutions

**Mutation** - Looking at completely new areas of search space

# FITNESS FUNCTION

---

- ✘ *quantifies the optimality of a solution (that is, a chromosome): that particular chromosome may be ranked against all the other chromosomes*
- ✘ A fitness value is assigned to each solution depending on how close it actually is to solving the problem.

# 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 concept

- 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).
-

# 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 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

# 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.
-



# ENCODING

---

- ✘ *The process of representing the solution in the form of a **string** that conveys the necessary information.*
- ✘ Just as in a chromosome, each gene controls a particular characteristic of the individual, similarly, each element in the string represents a characteristic of the solution.

# ENCODING METHODS

- ✘ **Binary Encoding** – Most common method of encoding. Chromosomes are strings of 1s and 0s and each position in the chromosome represents a particular characteristic of the problem.

Chromosome A	1011001011001110010
Chromosome B	1111111000000001111

1

- ✘ **Permutation Encoding** – Useful in ordering problems such as the Traveling Salesman Problem (TSP). Example. In TSP, every chromosome is a string of numbers, each of which represents a city to be visited.

Chromosome A	1	5	3	2	6	4	7	9	8
Chromosome B	8	5	6	7	2	3	1	4	9

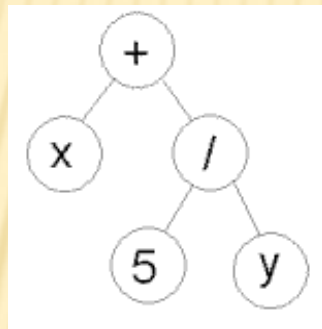
# ENCODING METHODS (CONTD.)

- ✘ **Real-Valued Chromosomes** Used in problems where complicated values, such as real numbers, are used and where binary encoding would not suffice.  
Good for some problems, but *often necessary to develop some specific crossover and mutation techniques for these chromosomes.*

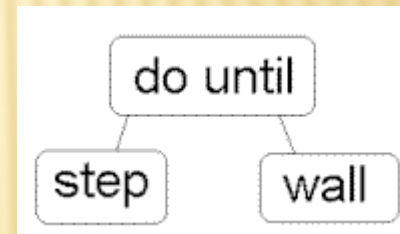
Chromosome A	1.235 5.323 0.454 2.321 2.454
Chromosome B	(left), (back), (left), (right), (forward)

# ENCODING METHODS (CONTD.)

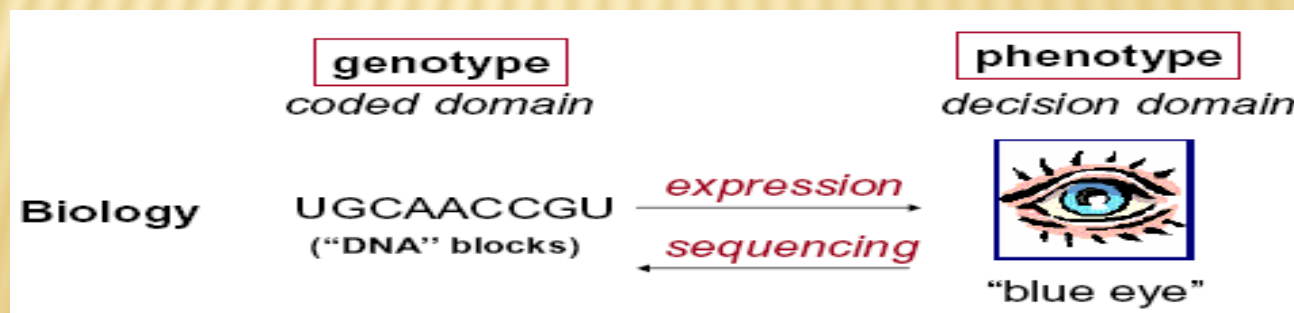
- ✘ **Tree structure** This encoding is used mainly for evolving programs or expressions, i.e. for Genetic programming.
- ✘ **Tree structure** every chromosome is a tree of some objects, such as values/arithmetic operators or commands in a programming language.



( + x ( / 5 y ) )



( do\_until step wall )



# **SELECTION TECHNIQUES**

---

**Roulette Selection**

**Rank Selection**

**Steady State Selection**

**Tournament Selection**

# ROULETTE WHEEL SELECTION

---

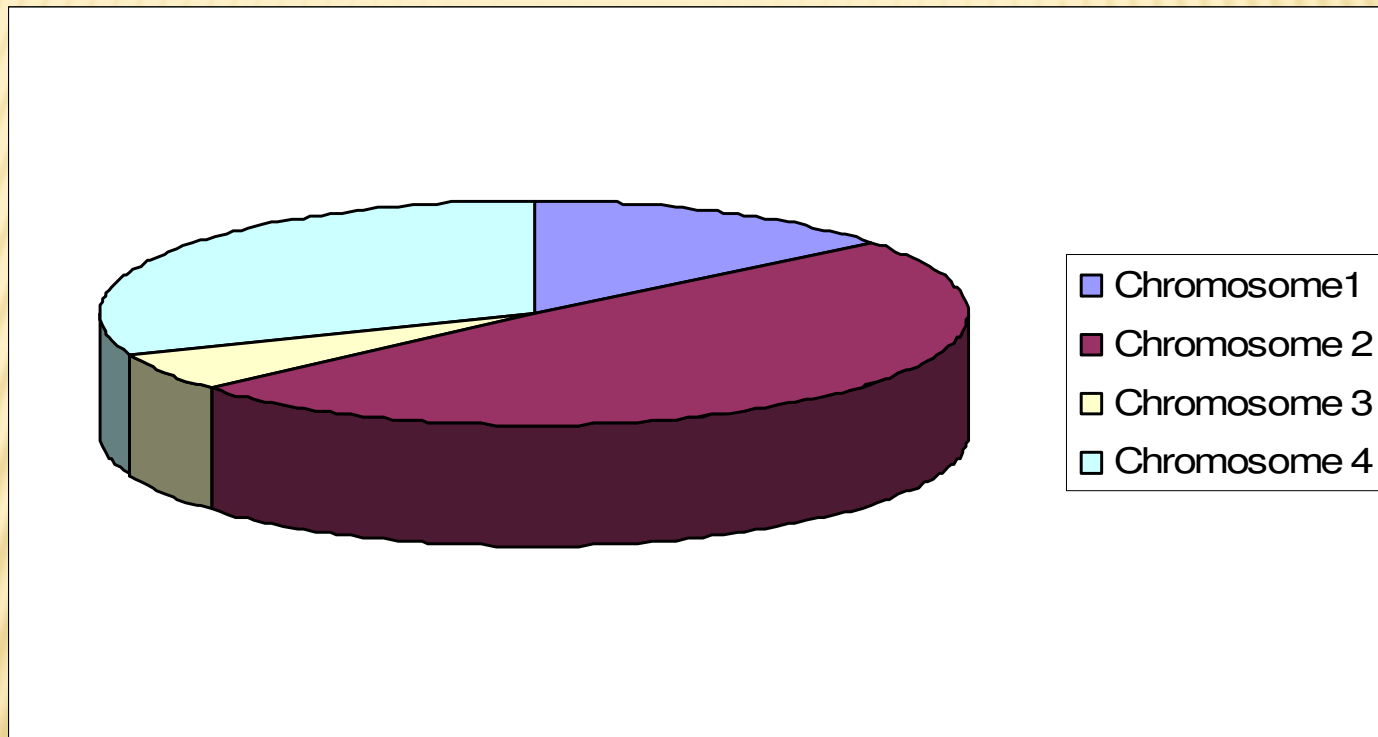
**Main idea**: the fitter is the solution with the most chances to be chosen

**HOW IT WORKS ?**

# EXAMPLE OF ROULETTE WHEEL SELECTION

<b>No.</b>	<b>String</b>	<b>Fitness</b>	<b>% Of Total</b>
1	01101	169	14.4
2	11000	576	49.2
3	01000	64	5.5
4	10011	361	30.9
<b>Total</b>		1170	100.0

# ROULETTE WHEEL SELECTION



**All you have to do is spin the ball and grab the chromosome at the point it stops**



# HOW TO SIMULATE ROULETTE WHEEL SELECTION SCHEME ?

---

- Suppose there are 10 chromosomes...
- Evaluate the fitness function of all the chromosomes
- Arrange the chromosomes in the descending order of their fitness values...(If u r going to maximize the objective function).
- Let say  $P_c = 0.8$ . Then select the best 8 chromosomes from the entire population.
- That means select the first 8 chromosomes.
- Now these chromosomes will be paired up randomly and then they will crossover.

# HOW TO SIMULATE TOURNAMENT SELECTION SCHEME ?

---

- ✘ Generate 2 random numbers between 1 to 10.
- ✘ Select the best chromosome between the two.
- ✘ If  $p_c = 0.8$  then there will be 8 crossovers. Two parents participate in one crossover, so we need 16 parents to crossover. Hence repeat the above mentioned procedure for 16 times.
- ✘ 1<sup>st</sup> and 2<sup>nd</sup> parent will crossover, then 3<sup>rd</sup> and 4<sup>th</sup> and so on.....

# CROSSOVER

---

**Main idea**: combine genetic material ( bits ) of 2 “parent” chromosomes ( solutions ) and produce a new “child” possessing characteristics of both “parents”.

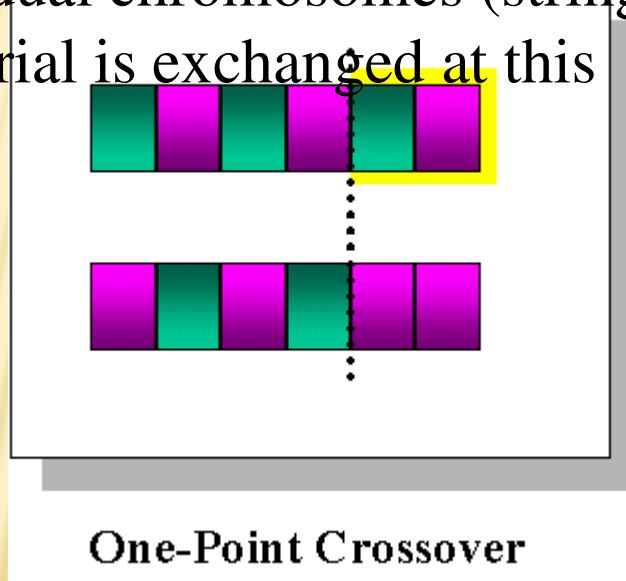
**How it works ?**

Several methods ....



# CONT INUE...

**Single Point Crossover-** A random point is chosen on the individual chromosomes (strings) and the genetic material is exchanged at this point.



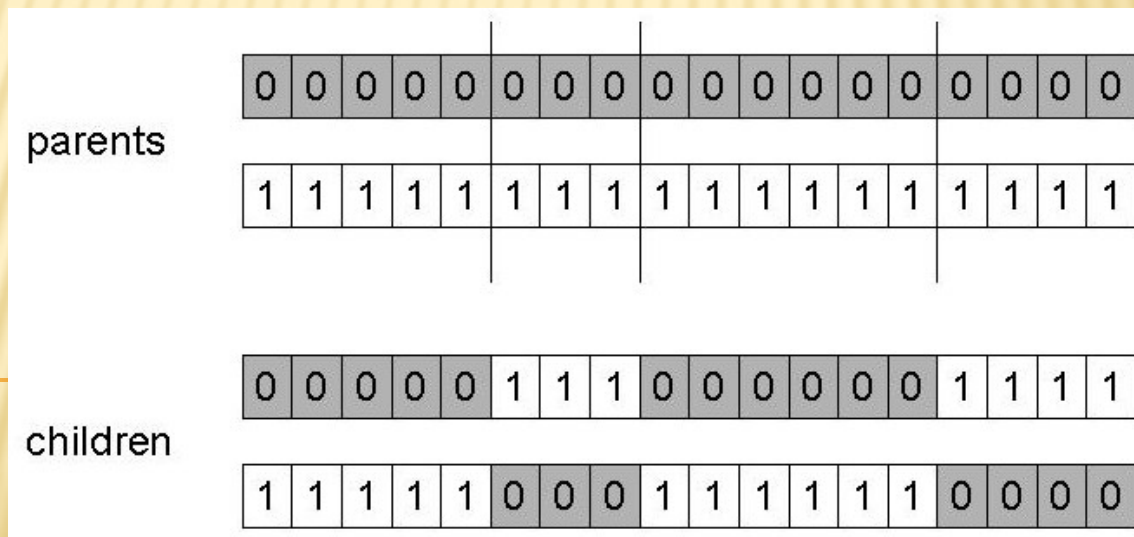
# TWO POINT CROSSOVER

Two-Point Crossover- Two random points are chosen on the individual chromosomes (strings) and the genetic material is exchanged at these points.

<b>Chromosome1</b>	<b>11011   00100  </b> <b>110110</b>
<b>Chromosome 2</b>	<b>10101   11000  </b> <b>011110</b>
<b>Offspring 1</b>	<b>10101   00100  </b> <b>011110</b>
<b>Offspring 2</b>	<b>11011   11000  </b> <b>110110</b>

# N- Point Crossover

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



# UNIFORM CROSSOVER METHODS

Each gene (bit) is selected randomly from one of the corresponding genes of the parent chromosomes

<b>Chromosome1</b>	<b>11011   00100   110110</b>
<b>Chromosome 2</b>	<b>10101   11000   011110</b>
<b>Offspring</b>	<b>10111   00000   110110</b>

**NOTE: Uniform Crossover yields ONLY 1 offspring.**



## CROSSOVER (CONTD.)

---

Crossover between 2 good solutions **MAY NOT ALWAYS** yield a better or as good a solution.

Since parents are good, probability of the child being good is high.

If offspring is not good (poor solution), it will be removed in the next iteration during “Selection”.

# ELITISM

---

**Main idea:** copy the best chromosomes (solutions) to new population *before applying crossover and mutation*

When creating a new population by crossover or mutation the best chromosome might be lost.

Forces GAs to retain some number of the best individuals at each generation.

Has been found that elitism significantly improves performance.

# MUTATION

---

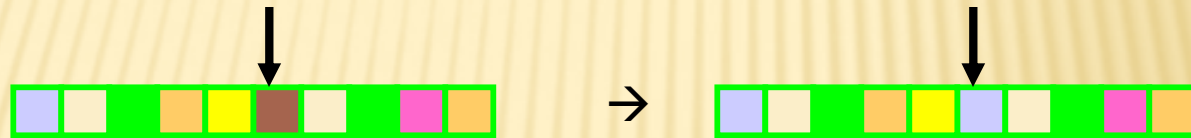
Main idea: *random inversion of bits in solution to maintain diversity in population set*



# CONTINUE...

- × Mutation

- + Generating new offspring from single parent



- + Maintaining the diversity of the individuals

- × Crossover can only explore the combinations of the current gene pool
- × Mutation can “generate” new genes

# MUTATION TECHNIQUES

---

- ✘ Flipping :Flipping of bit involves changing 0 to 1 and 1 to 0 based on mutation chromosome generated randomly
- ✘ Reversing: A random position is chosen and bits next to that position are reversed and child chromosome is produced
- ✘ Interchanging: Two random position of chromosome are chosen and the bits corresponding to that position are interchanged
- ✘

# GA AN EXAMPLE: THE MAXONE PROBLEM

---

Suppose we want to maximize the number of ones in a string of  $n$  binary digits

It may seem so because we know the answer in advance

However, we can think of it as maximizing the number of correct answers, each encoded by 1, to  $n$  'yes/no difficult questions'

## EXAMPLE (CONT)

- ✘ An individual is encoded (naturally) as a string of  $l$  binary digits
- ✘ The fitness  $f$  of a candidate solution to the MAXONE problem is the number of ones in its genetic code
- ✘ We start with a population of  $n$  random strings.  
Suppose that  $l = 10$  and  $n = 6$

# EXAMPLE (INITIALIZATION)

We toss a fair coin 60 times and get the following initial population:

$$s_1 = 1111010101 \quad f(s_1) = 7$$

$$s_2 = 0111000101 \quad f(s_2) = 5$$

$$s_3 = 1110110101 \quad f(s_3) = 7$$

$$s_4 = 0100010011 \quad f(s_4) = 4$$

$$s_5 = 1110111101 \quad f(s_5) = 8$$

$$s_6 = 0100110000 \quad f(s_6) = 3$$

Total fitness value is 34 ,Average fitness value will be  $34/6=5.66$



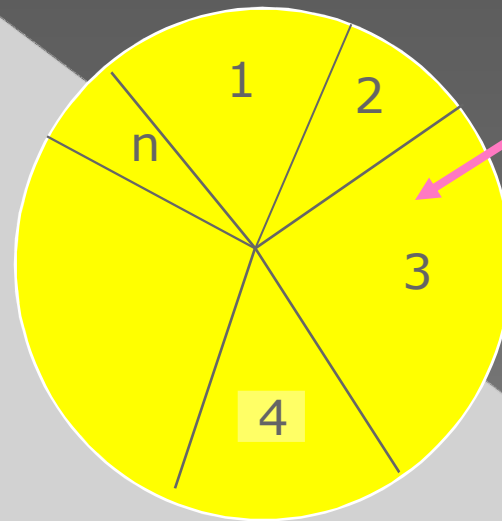
## Example (selection1)

Next we apply fitness proportionate selection with the roulette wheel method:

Individual  $i$  will have a probability to be chosen

$$\frac{f(i)}{\sum_i f(i)}$$

We repeat the extraction as many times as the number of individuals we need to have the same parent population size (6 in our case)



## EXAMPLE (SELECTION2)

Suppose that, after performing selection, we get the following population:

$$s_1' = 1111010101 \quad (s_1)$$

$$s_2' = 1110110101 \quad (s_3)$$

$$s_3' = 1110111101 \quad (s_5)$$

$$s_4' = 0111000101 \quad (s_2)$$

$$s_5' = 0100010011 \quad (s_4)$$

$$s_6' = 1110111101 \quad (s_5)$$

## EXAMPLE (CROSSOVER1)

Next we mate strings for crossover. For each couple we decide according to crossover probability (for instance 0.6) whether to actually perform crossover or not

Suppose that we decide to actually perform crossover only for couples  $(s_1, s_2)$  and  $(s_5, s_6)$ . For each couple, we randomly extract a crossover point, for instance 2 for the first and 5 for the second

## EXAMPLE (CROSSOVER2)

Before crossover:

$s_1' = 1111010101$   
 $s_2' = 1110110101$

$s_5' = 0100010011$   
 $s_6' = 1110111101$

After crossover:

$s_1'' = 1110110101$   
 $s_2'' = 1111010101$

$s_5'' = 0100011101$   
 $s_6'' = 1110110011$

# EXAMPLE (MUTATION1)

The final step is to apply random mutation: for each bit that we are to copy to the new population we allow a small probability of error (for instance 0.1)

Before applying mutation:

$$s_1 \text{ ' ' } = 11101\mathbf{1}0101$$

$$s_2 \text{ ' ' } = 1111\mathbf{0}1010\mathbf{1}$$

$$s_3 \text{ ' ' } = 11101\mathbf{1}11\mathbf{0}1$$

$$s_4 \text{ ' ' } = 0111000101$$

$$s_5 \text{ ' ' } = 0100011101$$

$$s_6 \text{ ' ' } = 11101100\mathbf{1}1$$

# EXAMPLE (MUTATION2)

After applying mutation:

$$s_1''' = 1110100101 \quad f(s_1''') = 6$$

$$s_2''' = 1111110100 \quad f(s_2''') = 7$$

$$s_3''' = 1110101111 \quad f(s_3''') = 8$$

$$s_4''' = 0111000101 \quad f(s_4''') = 5$$

$$s_5''' = 0100011101 \quad f(s_5''') = 5$$

$$s_6''' = 1110110001 \quad f(s_6''') = 6$$

---

## EXAMPLE (END)

In one generation, the total population fitness changed from 34 to 37, thus improved by ~9%

At this point, we go through the same process all over again, until a stopping criterion is met

# APPLICATIONS OF GA

---

1. Function Optimization
2. System Identification
3. Channel Equalization



# **Part- 3**

## **ARTIFICIAL NEURAL NETWORKS: AN INTRODUCTION**

# 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)**



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

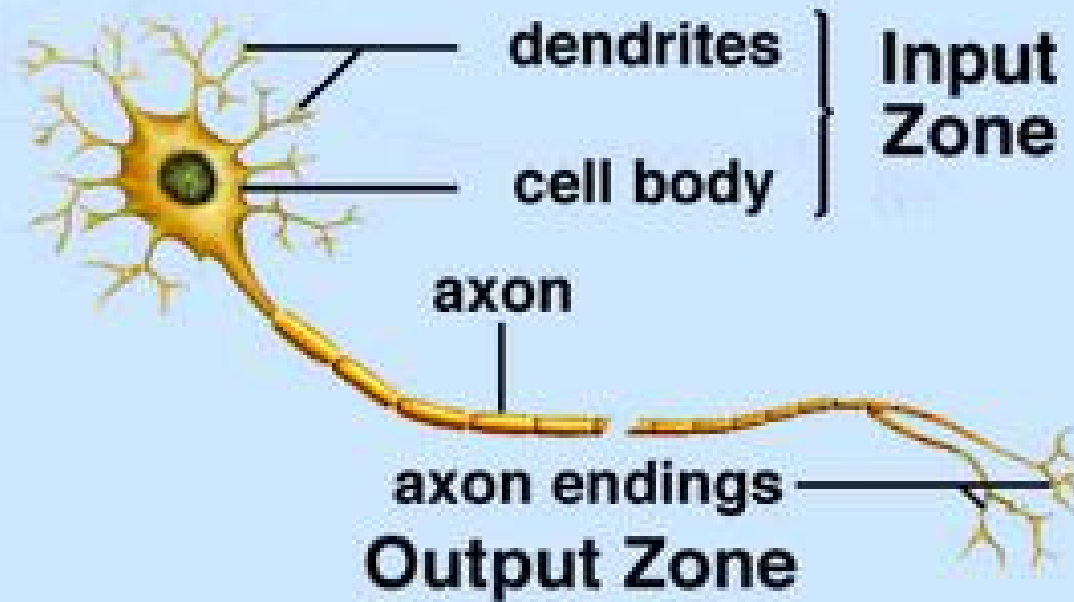


The **human brain** contains about 10 billion nerve cells, or neurons. On average, each neuron is connected to other neurons through approximately 10,000 synapses.

	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

## A Motor Neuron



# 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.

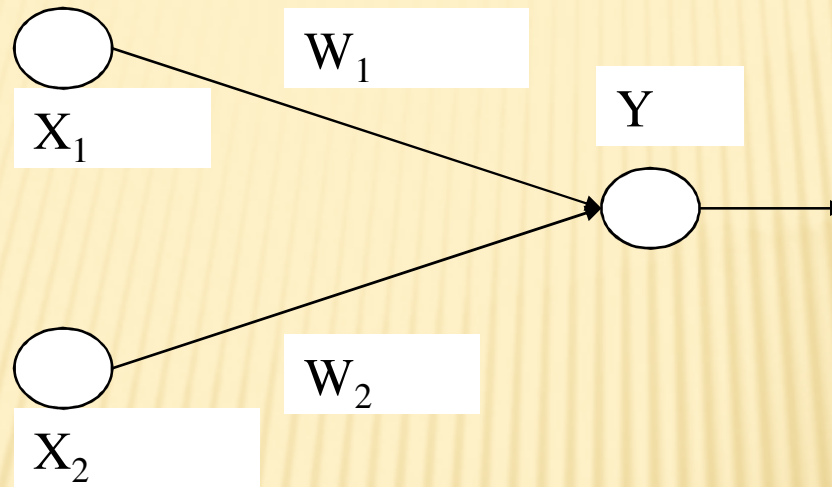
# MOTIVATION FOR NEURAL NET

- 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
- Adaptively
- Inherent contextual information processing
- Fault tolerance
- Low energy consumption.

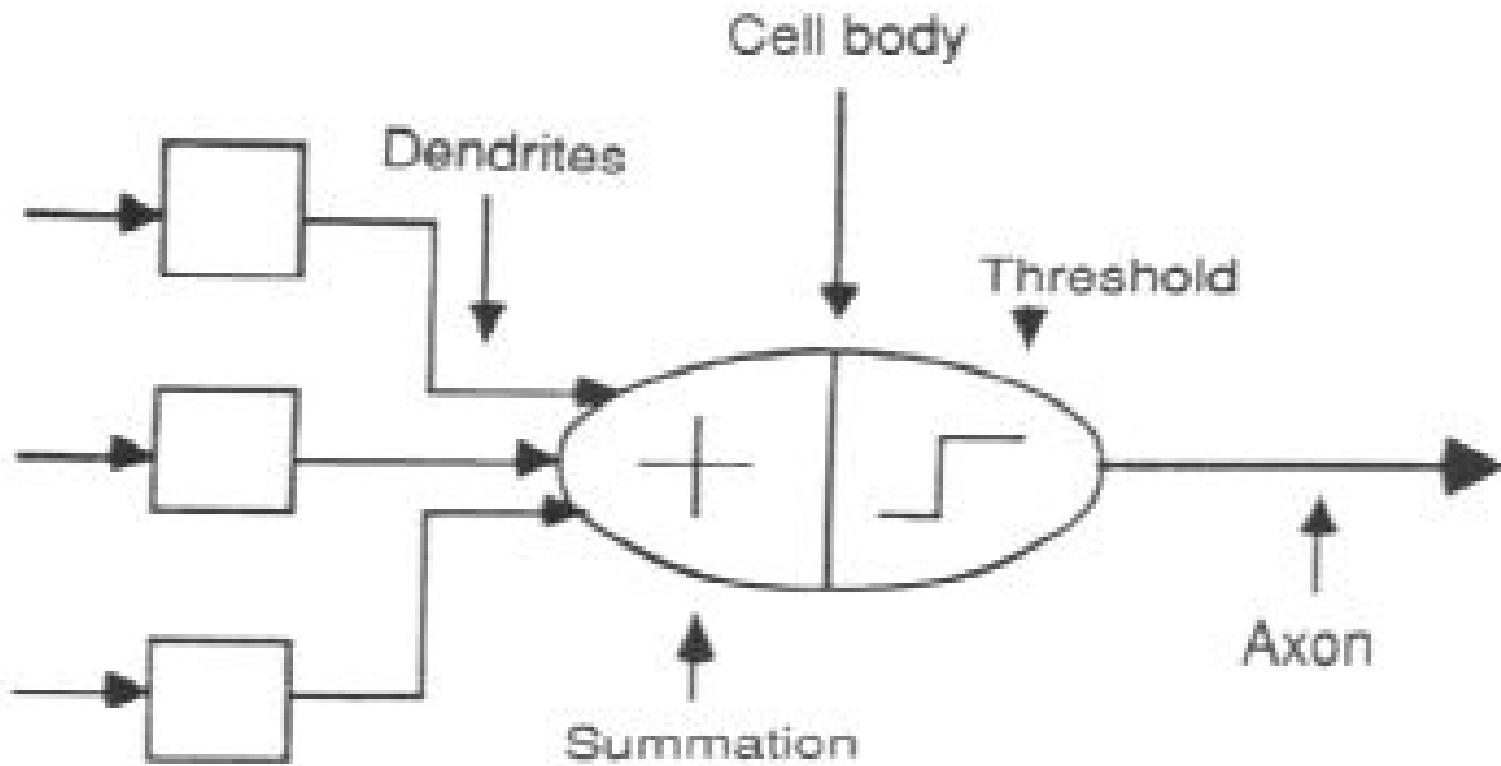
# ARTIFICIAL NEURAL NET



The figure shows a simple artificial neural net with two input neurons ( $X_1, X_2$ ) and one output neuron ( $Y$ ). The inter connected 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):

$$u = \sum_{j=1}^m W_j X_j$$

3. Activation function for limiting the amplitude of the neuron output.

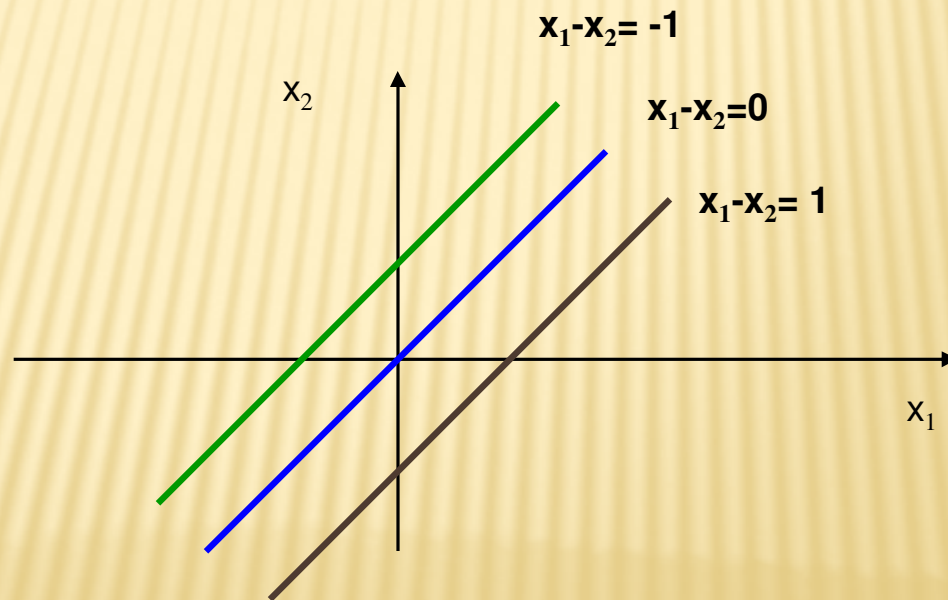
$$y = \varphi(u + b)$$

# 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.

$$Y_{in} = \sum w_i x_i + b, \text{ where } b \text{ is the bias}$$



# MULTI LAYER ARTIFICIAL NEURAL NET

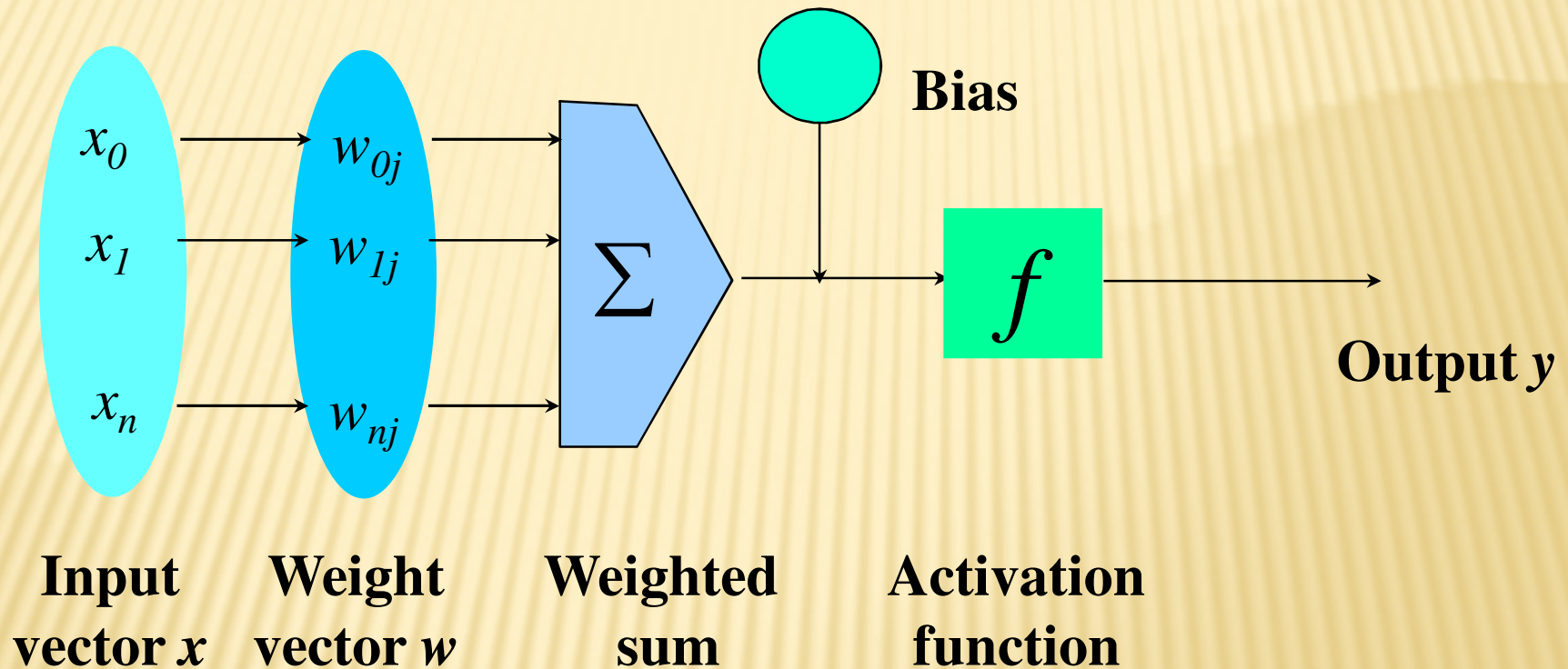
**INPUT:** records without class attribute with normalized attributes values.

**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.

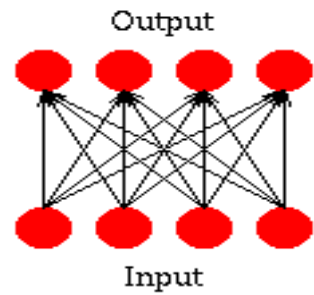
# NEURAL NETWORKS

- **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.

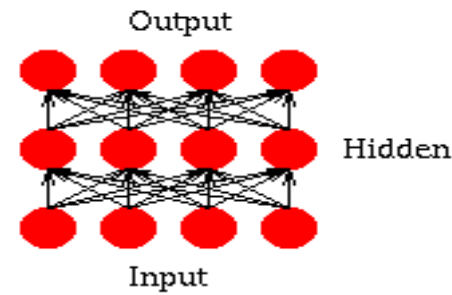


# **BUILDING BLOCKS OF ARTIFICIAL NEURAL NET**

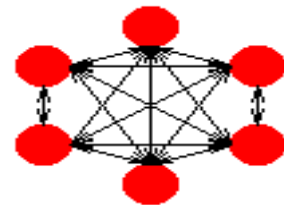
- Network Architecture (Connection between Neurons)
- Setting the Weights (Training)
- Activation Function



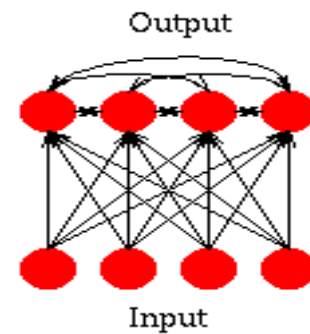
Single Layer Feedforward



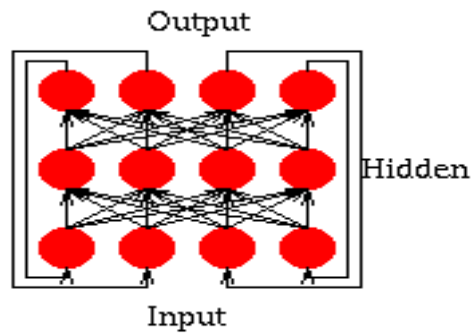
Multi Layer Feedforward



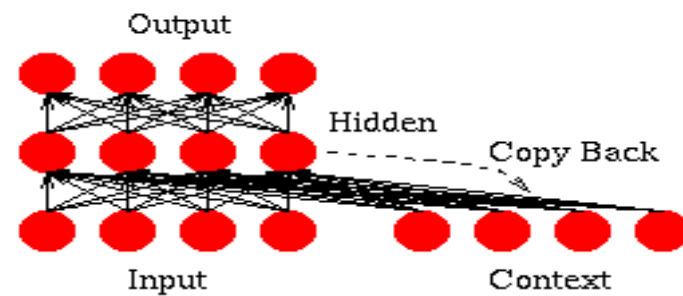
Fully Recurrent Network



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 METHODS

- **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

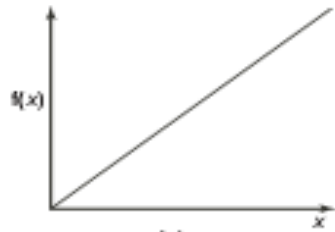
(B) Binary step

(C) Bipolar step

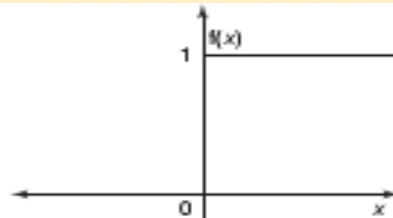
(D) Binary sigmoidal

(E) Bipolar sigmoidal

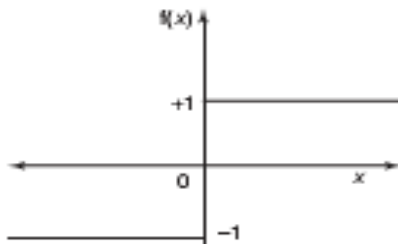
(F) Ramp



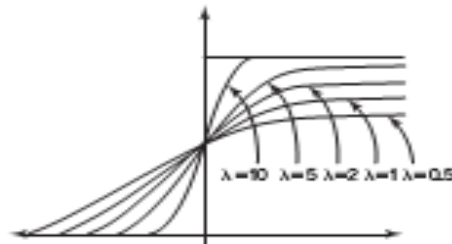
(a)



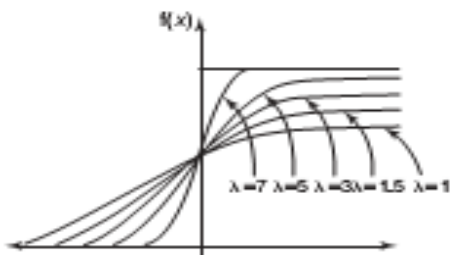
(b)



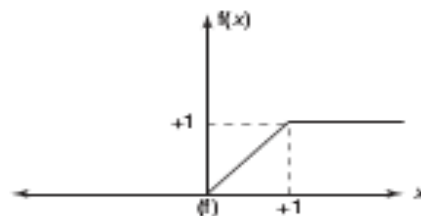
(c)



(d)



(e)



(f)

# 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.



# **SALIENT FEATURES OF ANN**

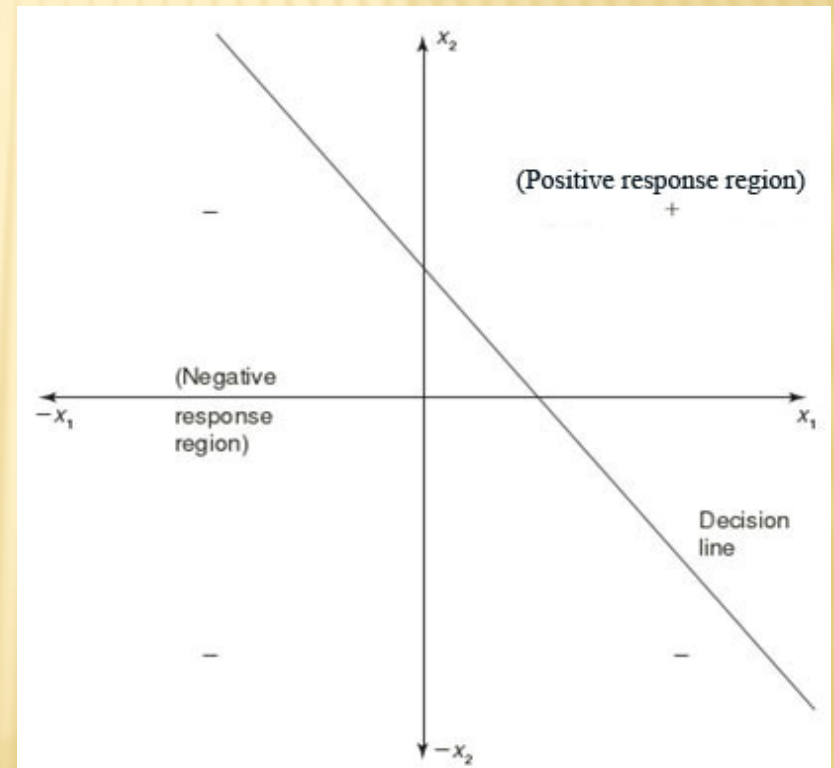
- Adaptive learning
- Self-organization
- Real-time operation
- Fault tolerance via redundant information coding
- Massive parallelism
- Learning and generalizing ability
- Distributed representation

# 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:

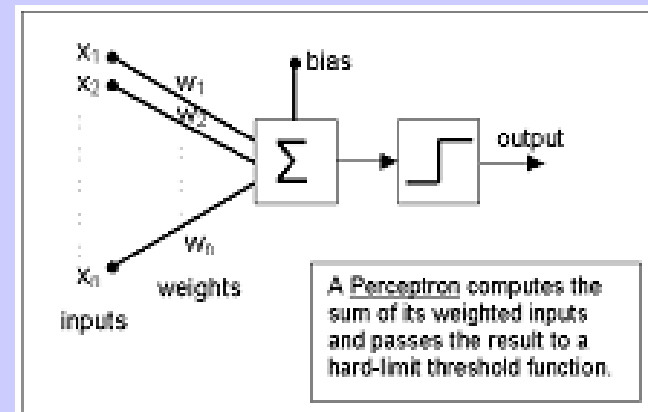
$$\frac{dw_{ij}}{dt} \sim \text{correlation}(x_i, x_j)$$

- 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.

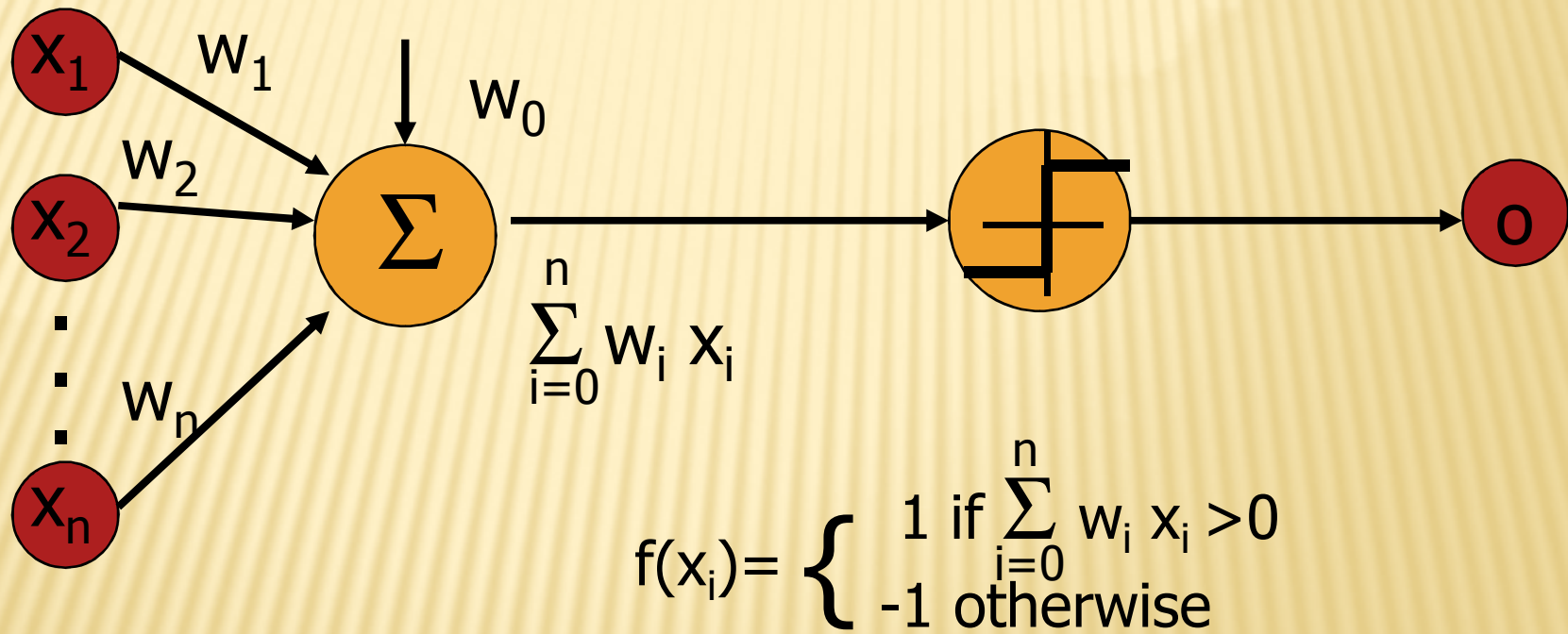


# PERCEPTRON NETWORKS

- 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  $\text{Error} = -1$ .

- **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.
- **Output, 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<sub>j</sub>) 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.

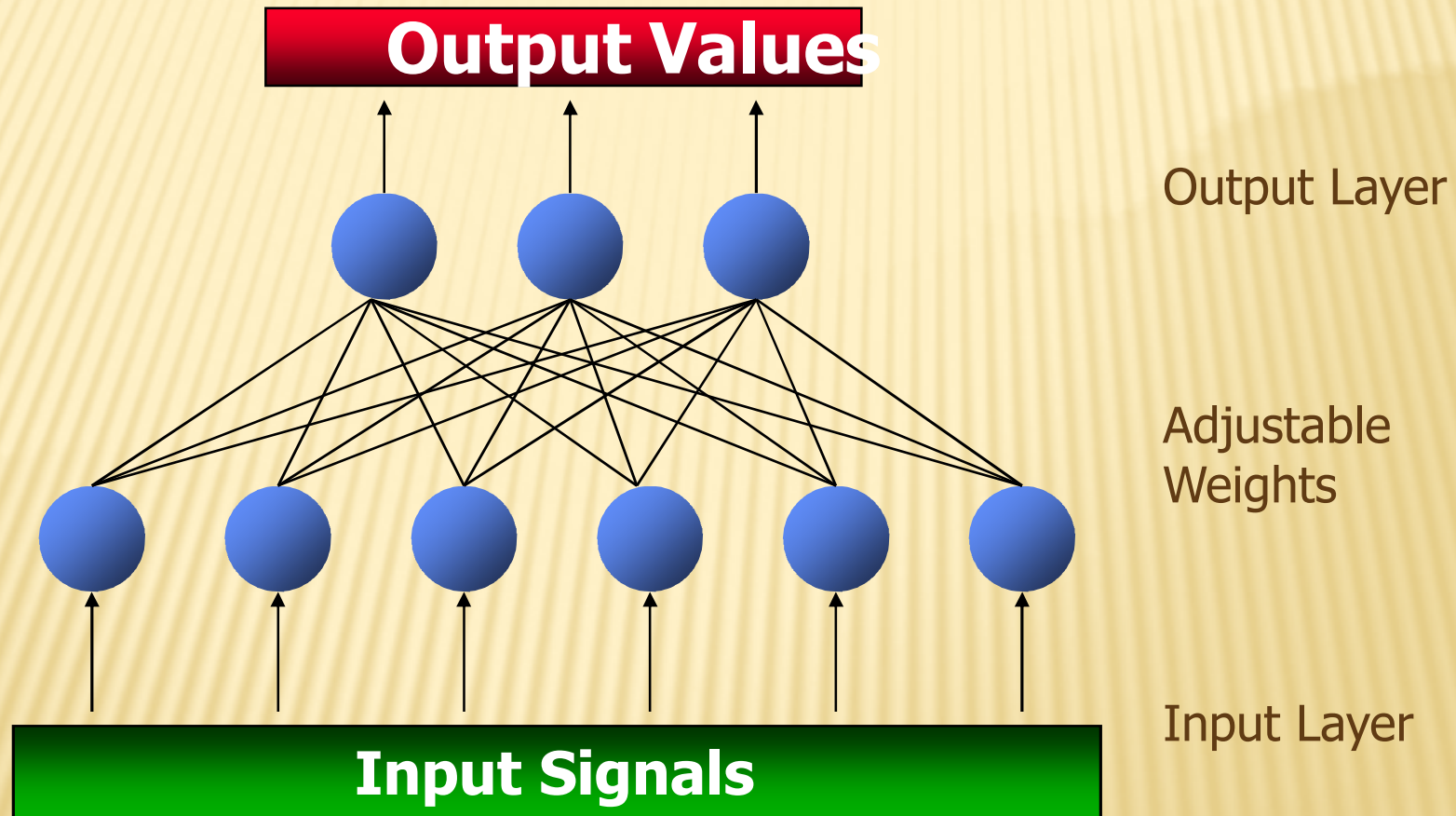
# TRAINING ALGORITHM

---

- 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):**

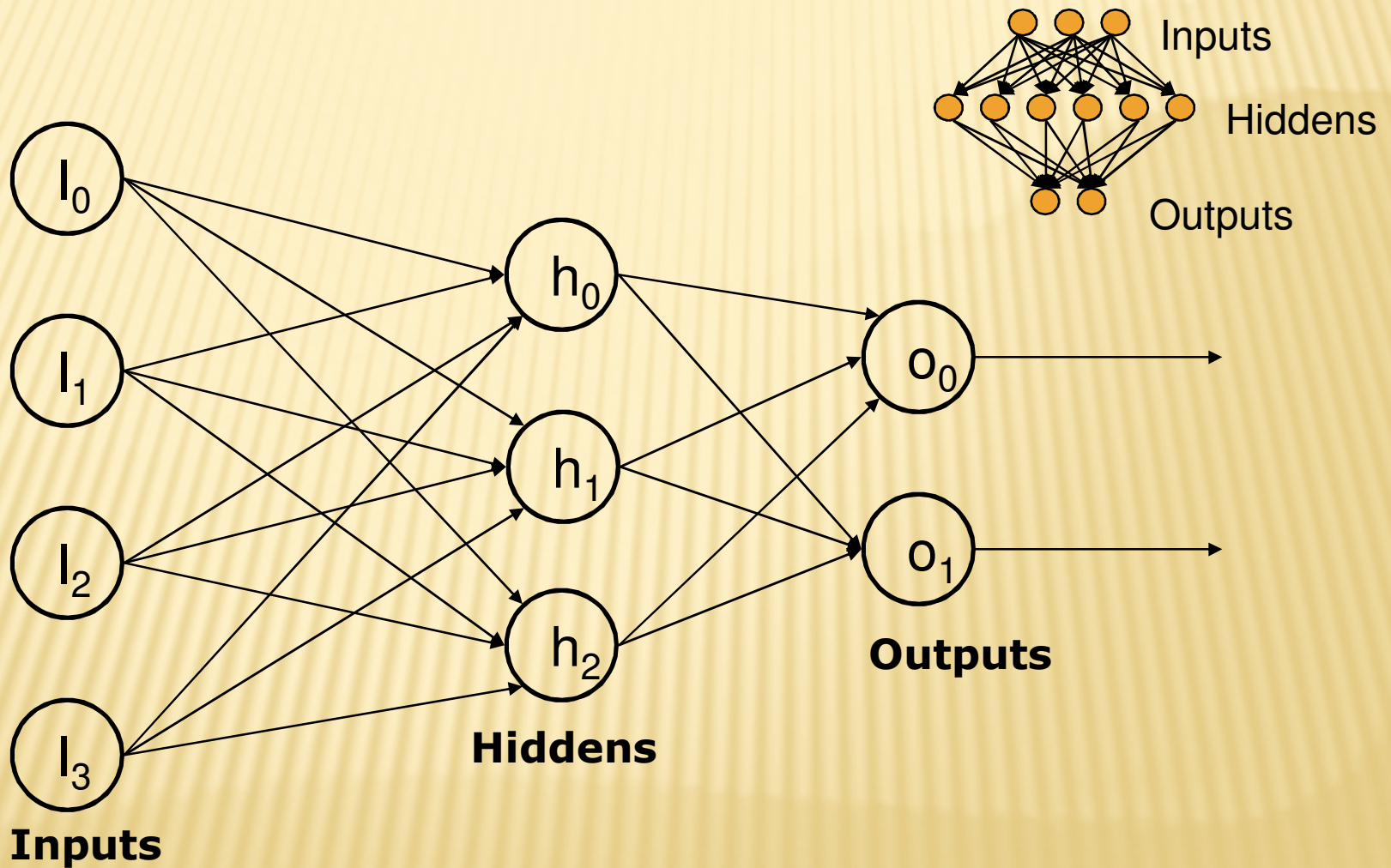
- Performs classification of features.
- Two hidden layers are sufficient to solve any problem.
- Features imply more layers may be better.

## ➤ **The output layer:**

- Functionally is just like the hidden layers.
- Outputs are passed on to the world outside the neural network.

- 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.

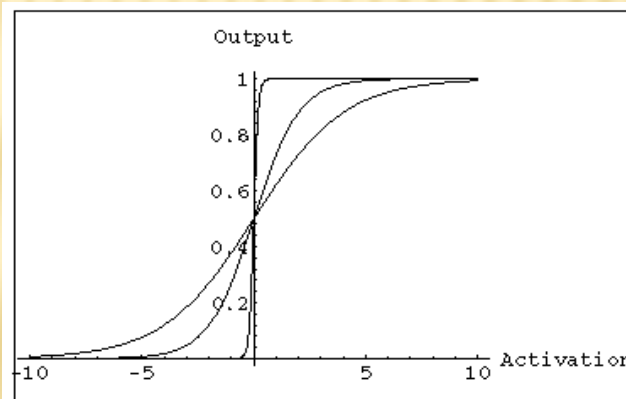
# MULTILAYER FEEDFORWARD NETWORK





# 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

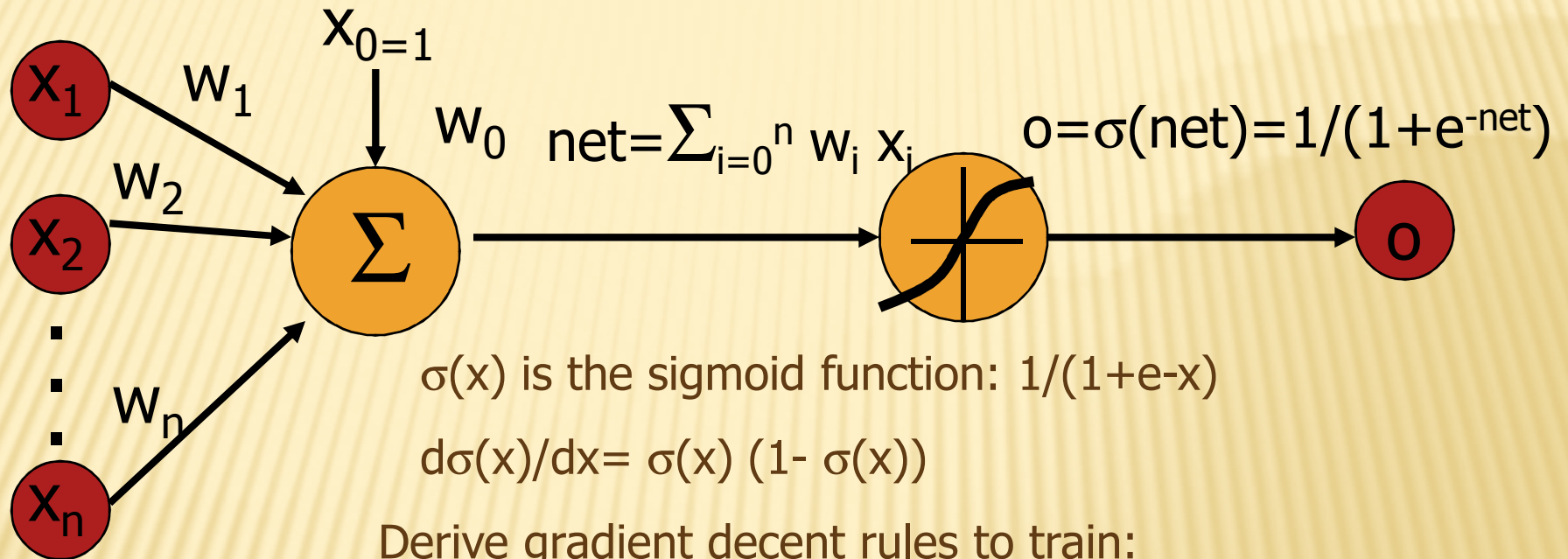
- $\Delta w_i = \Delta w_i + \eta (t - o) x_i$
- For each linear unit weight  $w_i$  Do
- $w_i = w_i + \Delta w_i$

# MODES OF GRADIENT DESCENT

---

- Batch mode : gradient descent  
 $w = w - \eta \nabla E_D[w]$  over the entire data  $D$   
 $E_D[w] = 1/2 \sum d (t_d - o_d)^2$
- Incremental mode: gradient descent  
 $w = w - \eta \nabla E_d[w]$  over individual training examples  $d$   
 $E_d[w] = 1/2 (t_d - o_d)^2$
- Incremental Gradient Descent can approximate Batch Gradient Descent arbitrarily closely if  $\eta$  is small enough.

# SIGMOID ACTIVATION FUNCTION



$\sigma(x)$  is the sigmoid function:  $1/(1+e^{-x})$

$$d\sigma(x)/dx = \sigma(x) (1 - \sigma(x))$$

Derive gradient decent rules to train:

- one sigmoid function  
 $\partial E/\partial w_i = -\sum d(td - od) od (1 - od) x_i$
- 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
      - $\Delta w_{i,j} = \eta \delta_j x_{i,j}$

# BACKPROPAGATION

---

- 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 OF BACKPROPAGATION NETWORK

---

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

# **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

---

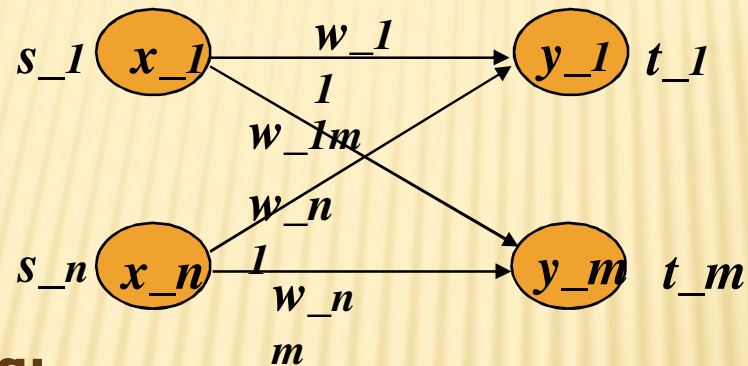
## ➤ WORKING PROCESS

- 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  $w_{ij}$  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  $s:t$ ,  $t_j = f(sT_wj)$  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  $\mathbf{s}$  and  $\mathbf{t}$  be row vectors.

Then for a particular training pair  $\mathbf{s}:\mathbf{t}$

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

and

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



# 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 
$$y_j = \begin{cases} 1 & \text{if } y_{in_j} > 0 \\ 0 & \text{if } y_{in_j} \leq 0 \end{cases}$$
- 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

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

$$s^T(3) \cdot t(3) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} (0 \ 1) = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \quad s^T(4) \cdot t(4) = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} (0 \ 1) = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}$$

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

# TEST/ RECALL THE NETWORK

$$x = [1000]$$

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

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

$x = [0100]$  similar to  $s(1)$  and  $s(2)$

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

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

$$x = [0110]$$

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

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

$(1 \ 0 \ 0 \ 0), (1 \ 1 \ 0 \ 0)$  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 a 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	$(111-1) \cdot W = (444-4) \rightarrow (111-1)$
noisy pattern	$(-111-1) \cdot W = (222-2) \rightarrow (111-1)$
missing info	$(001-1) \cdot W = (222-2) \rightarrow (111-1)$
more noisy	$(-1-11-1) \cdot W = (0000)$ 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.

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

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

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

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

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

# 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

$$\begin{array}{l} (1 \ -1 \ -1 \ 1) \\ (1 \ 1 \ -1 \ 1) \end{array} \rightarrow \mathbf{W}_0 = \begin{bmatrix} 0 & 0 & -2 & 2 \\ 0 & 0 & 0 & 0 \\ -2 & 0 & 0 & -2 \\ 2 & 0 & -2 & 0 \end{bmatrix} \quad \begin{array}{l} (1-1-11) \cdot \mathbf{W}_0 = (1 \ 0 \ -1 \ 1) \\ \text{It is not stored correctly} \end{array}$$

- Orthogonal

$$\begin{array}{l} (1 \ 1 \ -1 \ -1) \\ (-1 \ 1 \ 1 \ -1) \\ (-1 \ 1 \ -1 \ 1) \end{array} \rightarrow \mathbf{W}_0 = \begin{bmatrix} 0 & -1 & -1 & -1 \\ -1 & 0 & -1 & -1 \\ -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix} \quad \begin{array}{l} \text{All three patterns can be} \\ \text{correctly recalled} \end{array}$$

# BIDIRECTIONAL ASSOCIATIVE MEMORY (BAM) NETWORK

## Architecture:

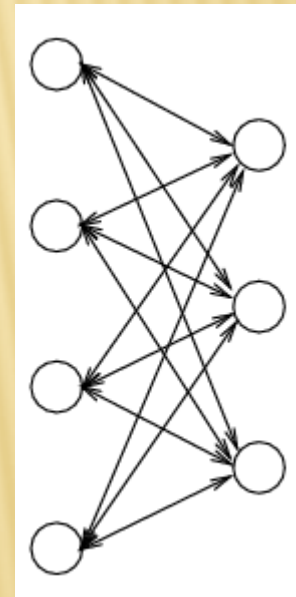
- Two layers of non-linear units: X-layer, Y-layer.
- Units: discrete threshold, continuing sigmoid (can be either binary or bipolar).

## 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.

<b>Activation function for the Y-layer</b>	<b>Activation function for the X-layer</b>
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.

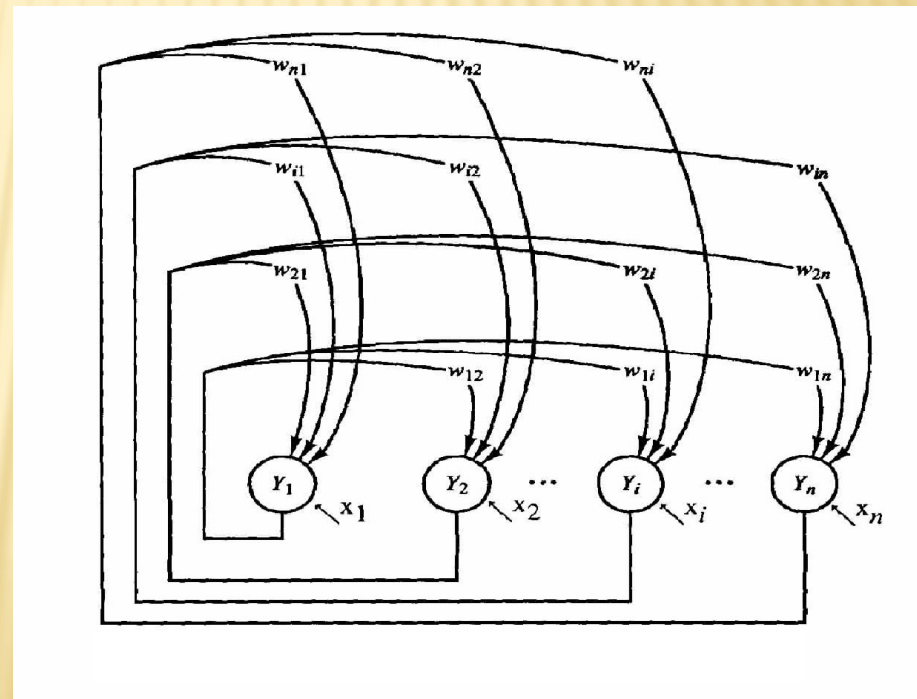
# ARCHITECTURE OF DHN

## ➤ Architecture

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

$$w_{ij} = w_{ji}$$
$$w_{ii} = 0$$

$x_i$  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  $\mathbf{x}$  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**

# UNSUPERVISED LEARNING

- 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.

# FEW UNSUPERVISED LEARNING NETWORKS

---

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.

# 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

---

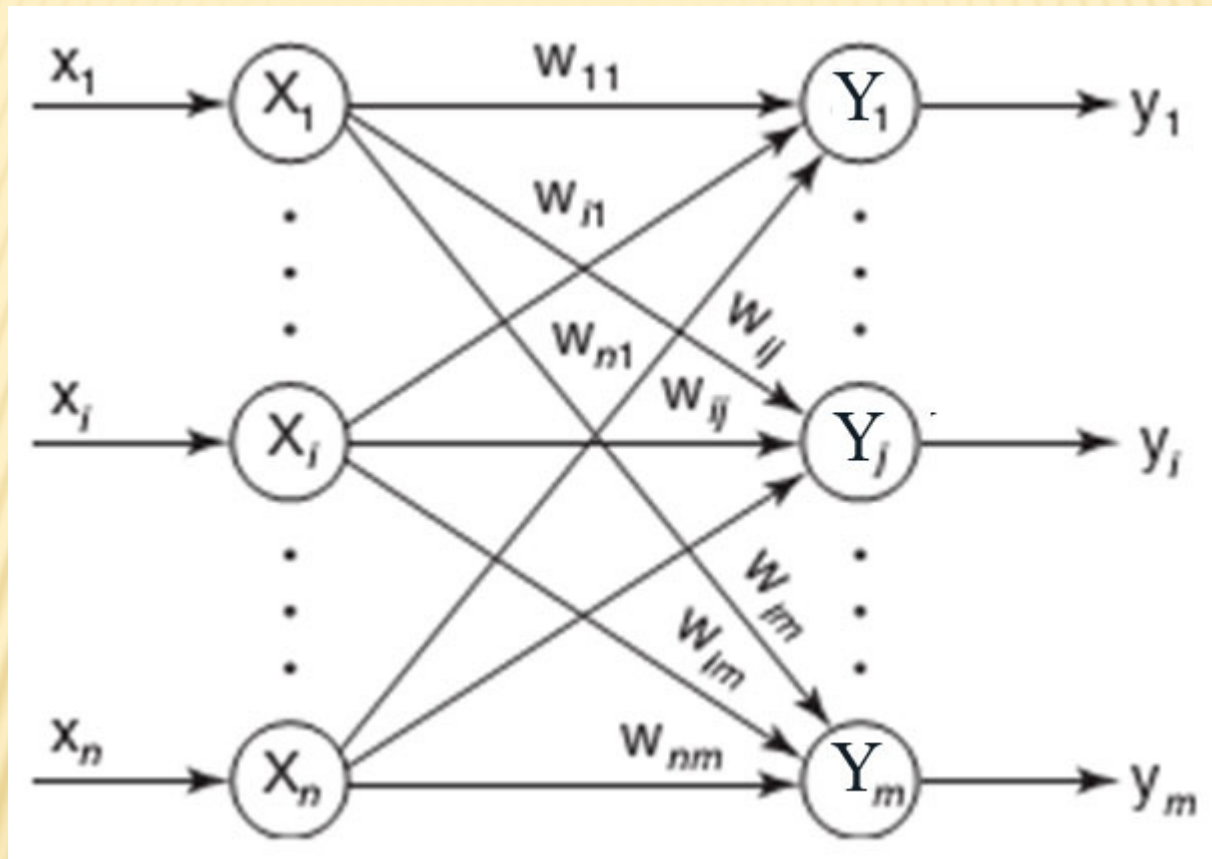
- Discover significant patterns or features in the input data.
- Discovery is done without a teacher.
- Synaptic weights are changed according to local rules.
- The changes affect a neuron's immediate environment until a final configuration develops.

# KOHONEN SELF-ORGANIZING FEATURE MAP (KSOFM)

---

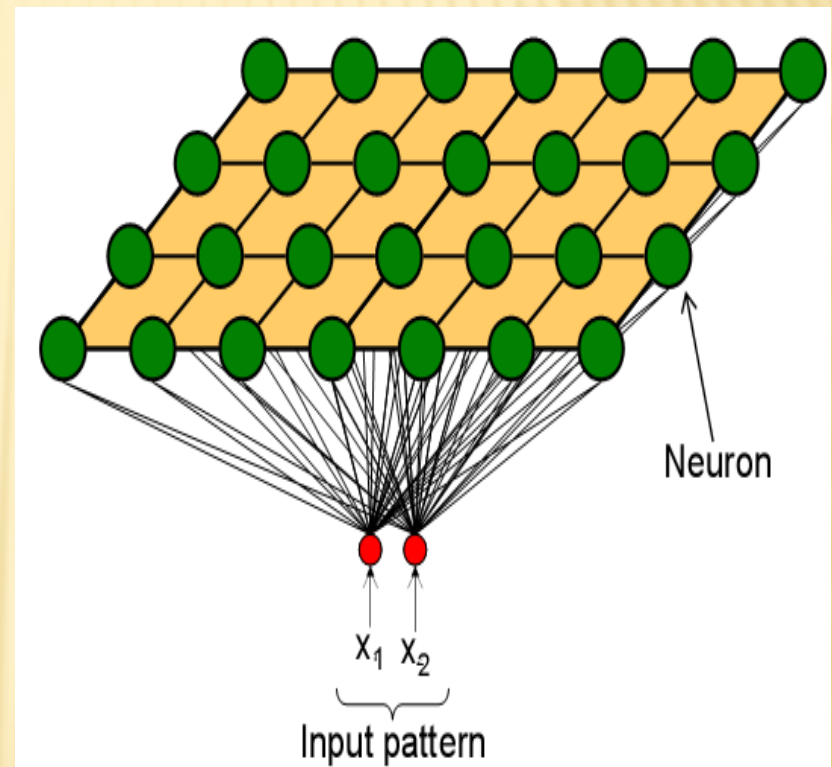
- The Kohonen model provides a topological mapping.
- It places a fixed number of input patterns from the input layer into a higher dimensional output or Kohonen layer.
- Training in the Kohonen network begins with the winner's neighborhood of a fairly large size. Then, as training proceeds, the neighborhood 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

---

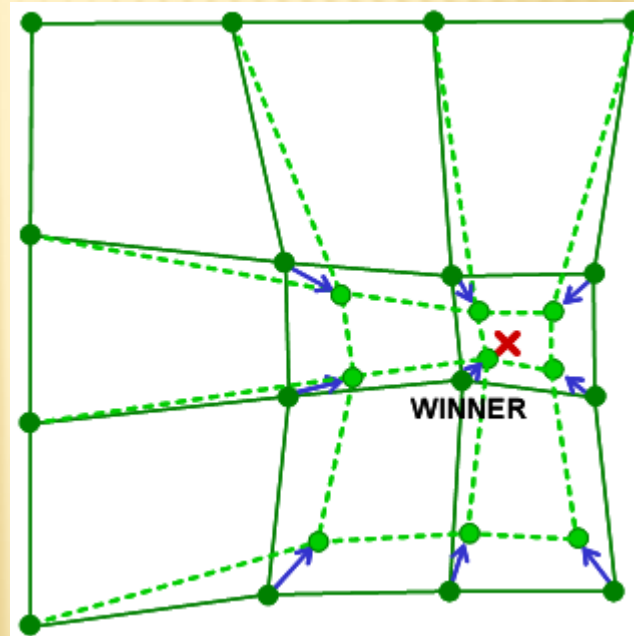
- The activation of the winning neuron is spread to neurons in its immediate neighborhood.
  - This allows topologically close neurons to become sensitive to similar patterns.
- The winner's neighborhood is determined on the lattice topology.
  - Distance in the lattice is a function of the number of 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

$$\mathbf{W}_1 = \begin{bmatrix} 0.27 \\ 0.81 \end{bmatrix} \quad \mathbf{W}_2 = \begin{bmatrix} 0.42 \\ 0.70 \end{bmatrix} \quad \mathbf{W}_3 = \begin{bmatrix} 0.43 \\ 0.21 \end{bmatrix}$$

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:

$$\mathbf{W}_3(p+1) = \mathbf{W}_3(p) + \Delta \mathbf{W}_3(p) = \begin{bmatrix} 0.43 \\ 0.21 \end{bmatrix} + \begin{bmatrix} 0.01 \\ -0.01 \end{bmatrix} = \begin{bmatrix} 0.44 \\ 0.20 \end{bmatrix}$$

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

# 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

# **Part 4**

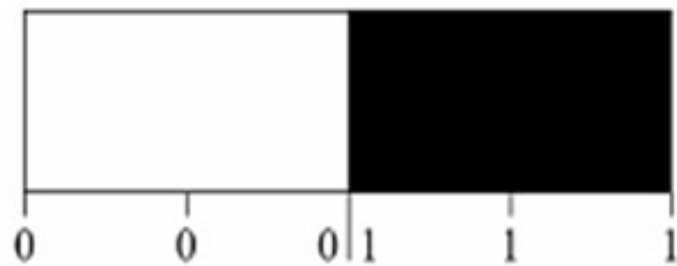
## **INTRODUCTION TO FUZZY LOGIC, CLASSICAL SETS AND FUZZY SETS**

# FUZZY LOGIC

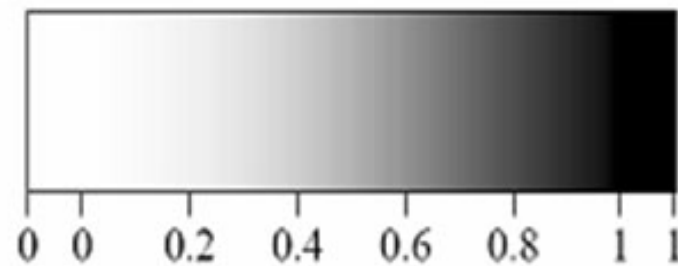
---

- 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 multivalued 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 multivalued system,
  - **Range over the fuzzy subsets of T** in fuzzy logic.

- Boolean logic uses sharp distinctions.
- Fuzzy logic reflects how people think.



(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.

# 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)

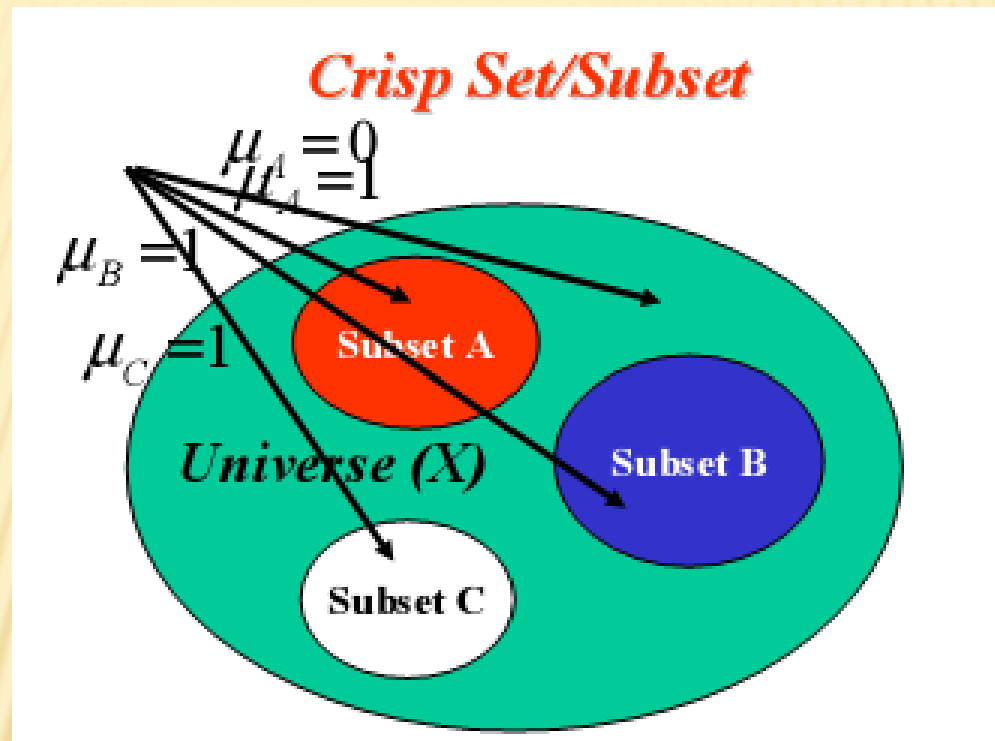
---

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

**{True, False}**

**{1, 0}**

# CRISP SETS



# 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

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

6. Identity

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

7. Involution (double negation)

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

8. Law of excluded middle two point crossover

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

9. Law of contradiction

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

10. DeMorgan's law

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

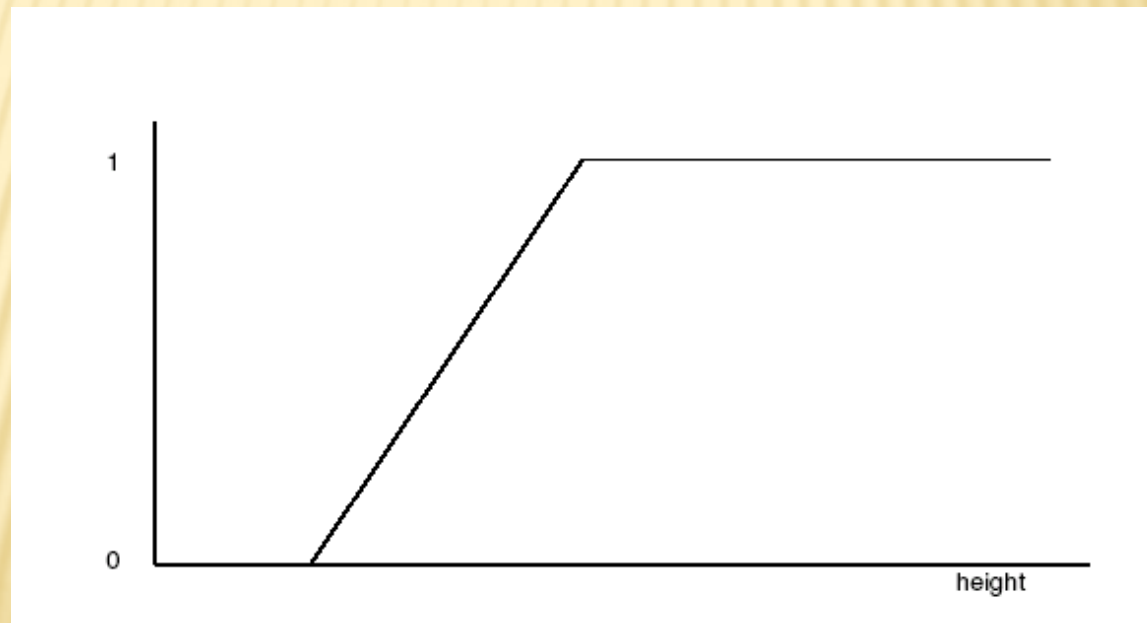
# FUZZY LOGIC THEN . . .

---

- ✘ is particularly good at handling uncertainty, vagueness and imprecision.
- ✘ especially useful where a problem can be described linguistically (using words).
- ✘ Applications include:
  - robotics
  - washing machine control
  - nuclear reactors
  - focusing a camcorder
  - information retrieval
  - train scheduling

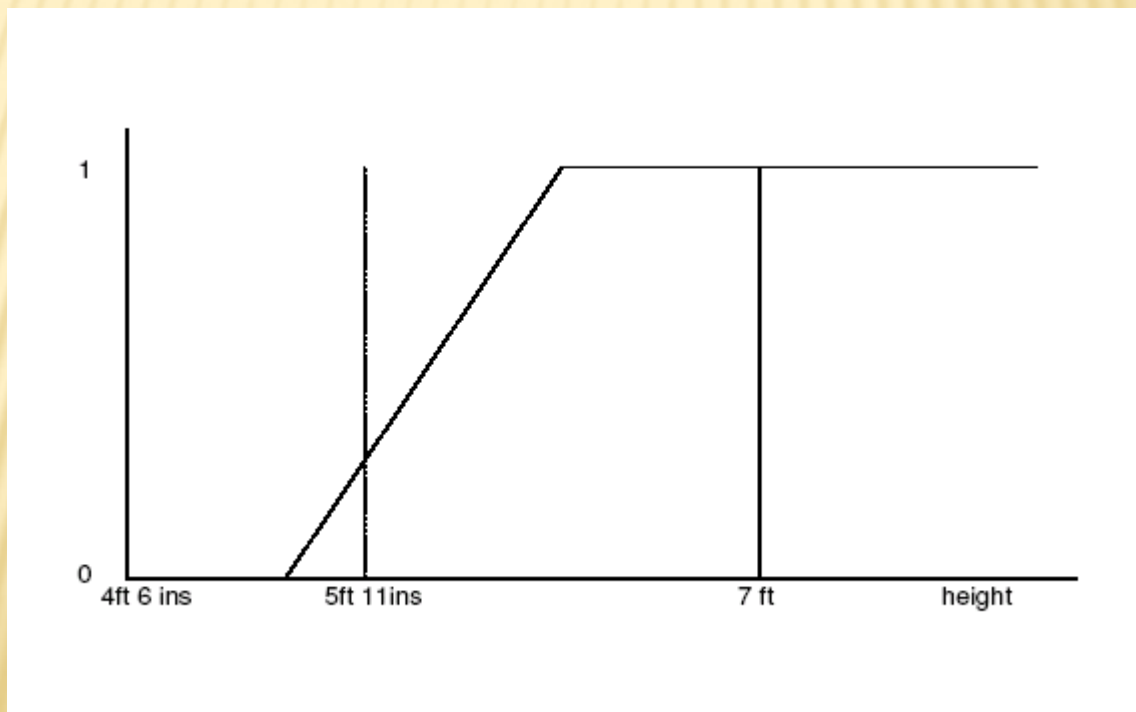
# FUZZY SETS

- ✦ The shape you see is known as the membership function



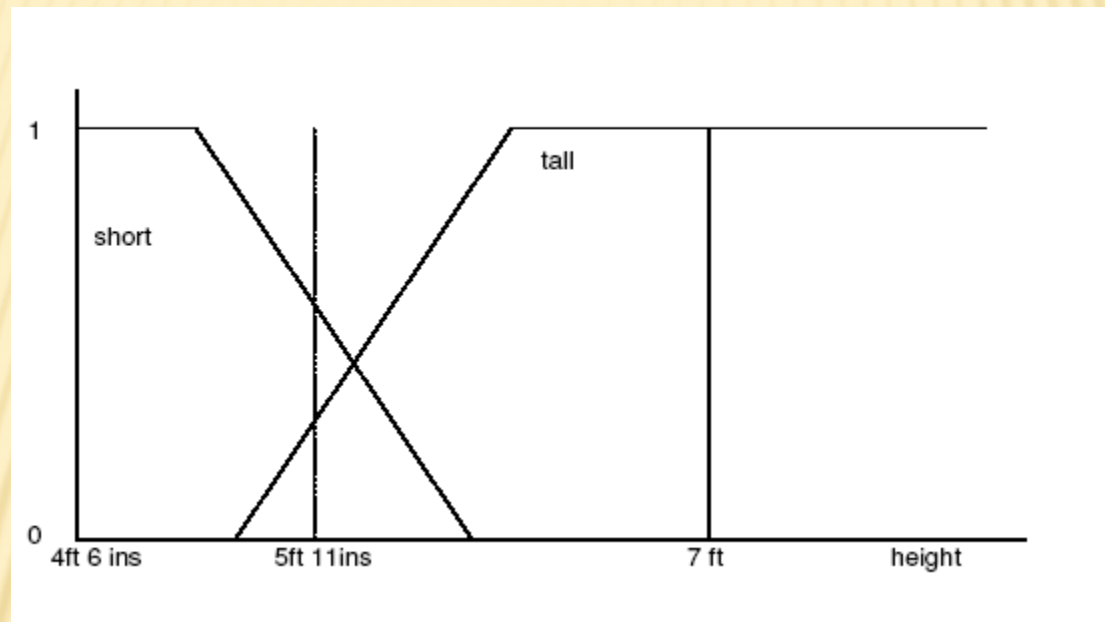
# FUZZY SETS

- ✦ Now we have added some possible values on the height - axis





# FUZZY SETS



Shows two membership functions:  
'tall' and 'short'

# NOTATION

---

- ✦ For any fuzzy set,  $A$ , the function  $\mu_A$  represents the membership function for which  $\mu_A(x)$  indicates the degree of membership of  $x$  (of the universal set  $X$ ) in set  $A$ . It is usually expressed as a number between 0 and 1:

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

# NOTATION

For the member,  $x$ , of a discrete set with membership  $\mu$  we use the notation  $\mu/x$ . In other words,  $x$  is a member of the set to degree  $\mu$ . Discrete sets are written as:

$$A = \mu_1/x_1 + \mu_2/x_2 + \dots + \mu_n/x_n$$

**Or**

$$A = \sum_{i=1, n} \mu_i/x_i$$

where  $x_1, x_2, \dots, x_n$  are members of the set  $A$  and  $\mu_1, \mu_2, \dots, \mu_n$  are their degrees of membership. A continuous fuzzy set  $A$  is written as:

$$A = \int_X \mu(x)/x.$$

# FUZZY SETS

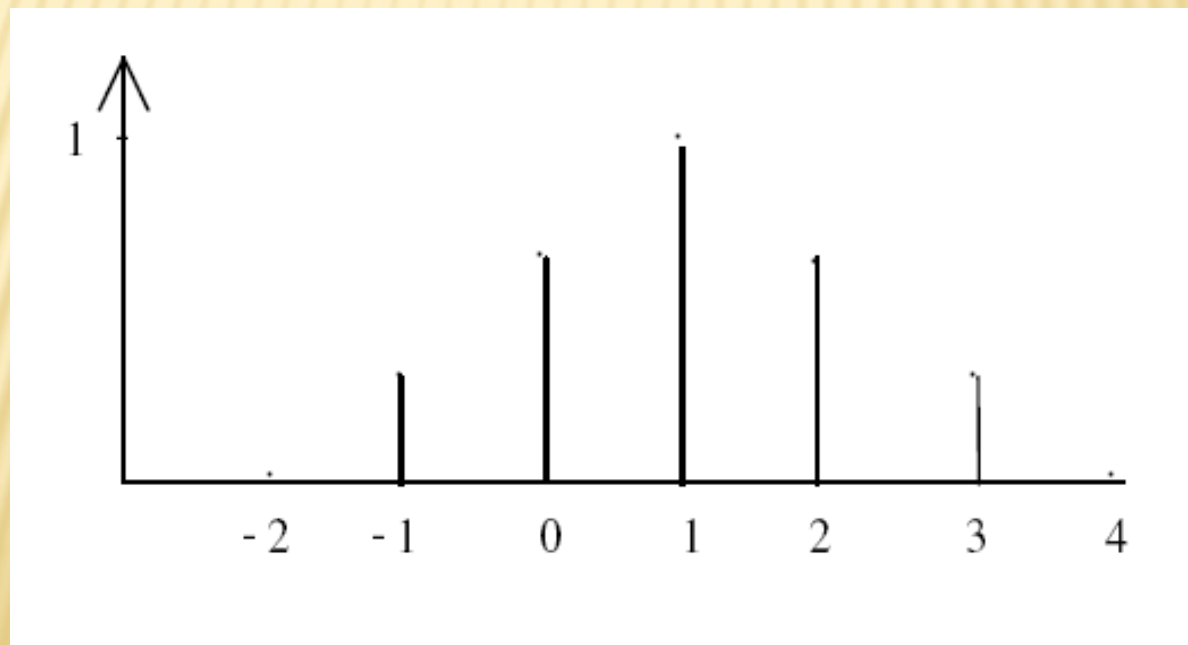
---

- ✘ The members of a fuzzy set are members to some degree, known as a membership grade or degree of membership.
- ✘ The membership grade is the degree of belonging to the fuzzy set. The larger the number (in  $[0,1]$ ) the more the degree of belonging. (N.B. This is not a probability)
- ✘ The translation from  $x$  to  $\mu_A(x)$  is known as fuzzification.
- ✘ A fuzzy set is either continuous or discrete.
- ✘ Graphical representation of membership functions is very useful.

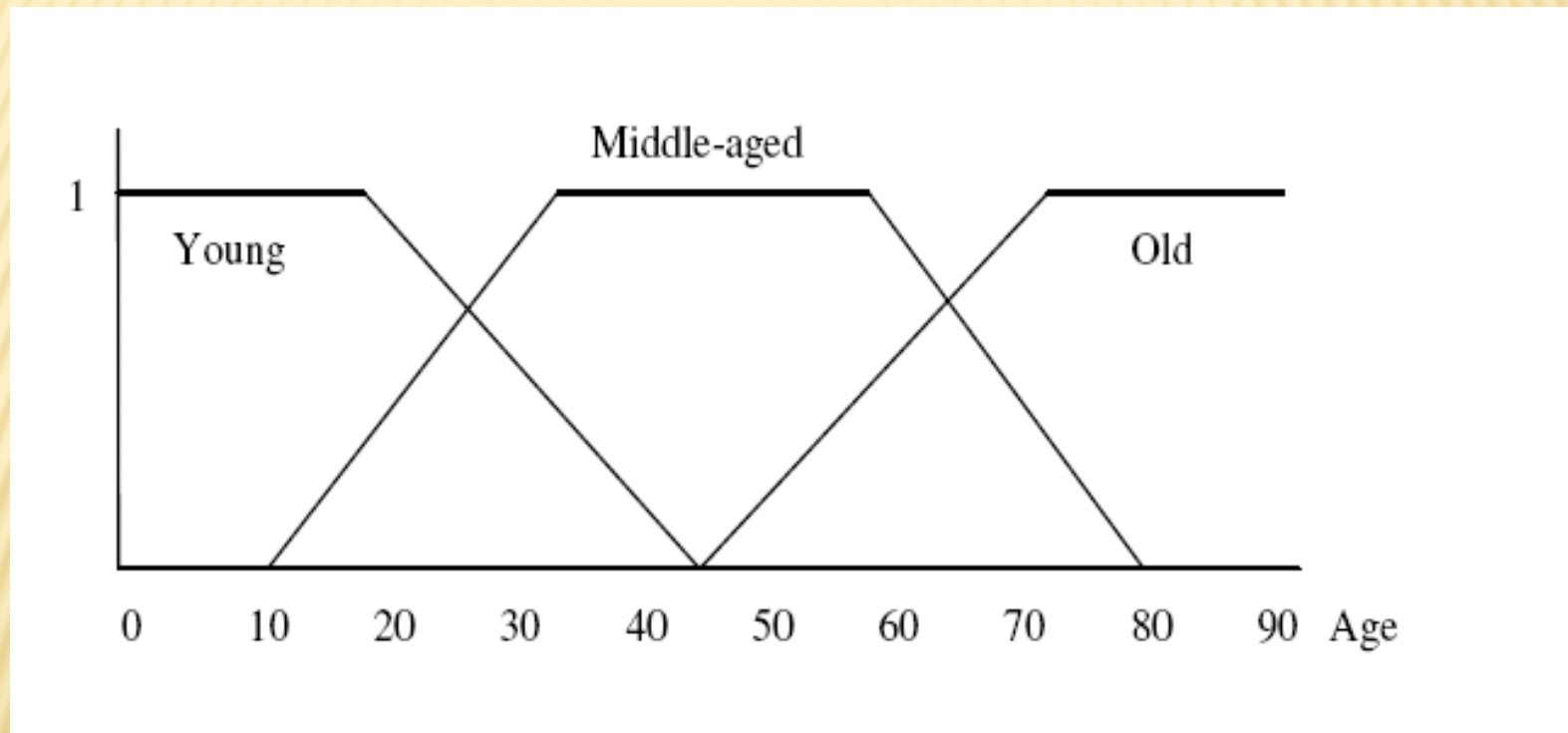
# FUZZY SETS - EXAMPLE

“numbers close to 1”

$$A = 0.0/-2 + 0.3/-1 + 0.6/0 + 1.0/1 + 0.6/2 + 0.3/3 + 0.0/4$$

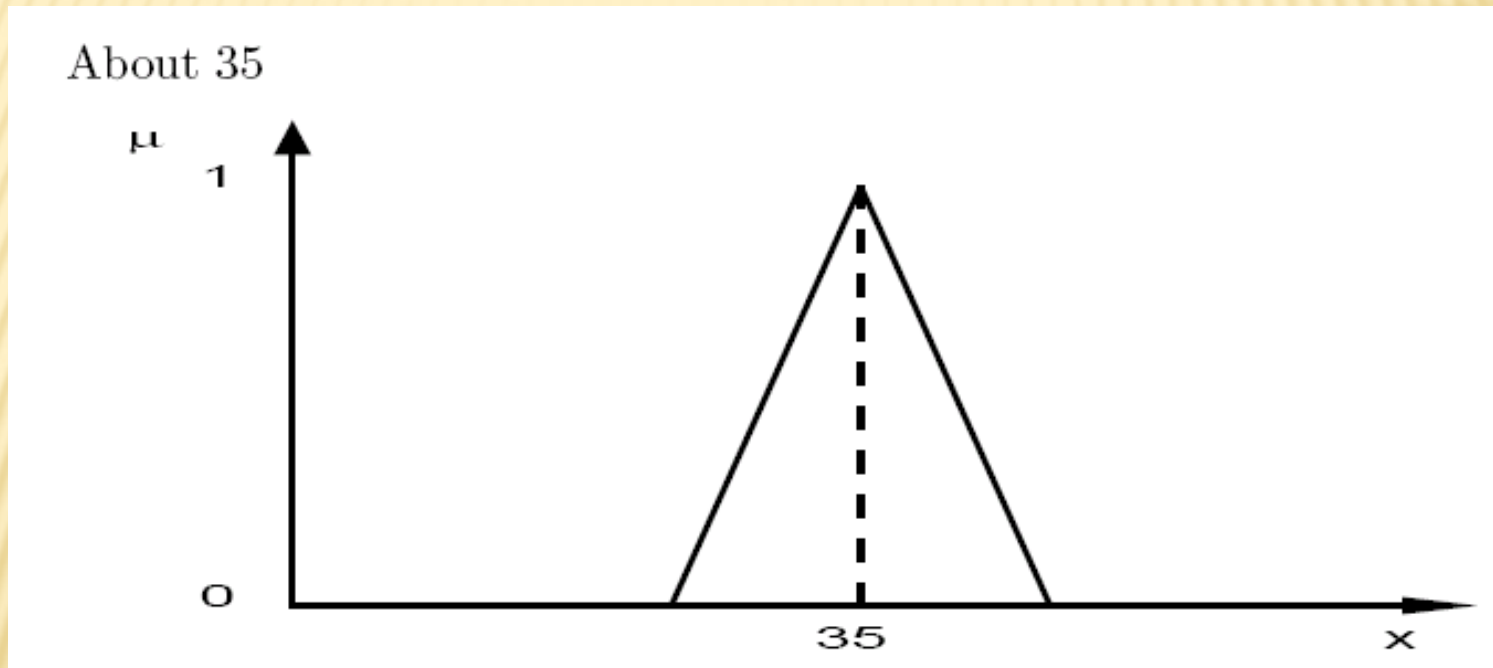


# FUZZY SETS - EXAMPLE



Again, notice the overlapping of the sets reflecting the real world more accurately than if we were using a traditional approach.

# IMPRECISION



Words are used to capture imprecise notions, loose concepts or perceptions.

# OPERATIONS ON FUZZY SETS

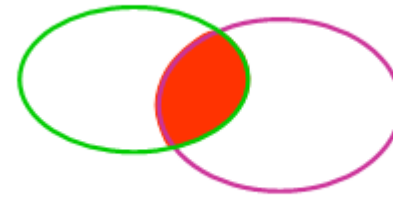
- Union:  $\mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x))$
- Intersection:  $\mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x))$
- Complement:  $\mu_{\neg A}(x) = 1 - \mu_A(x)$

**Fuzzy union operation or fuzzy *OR***



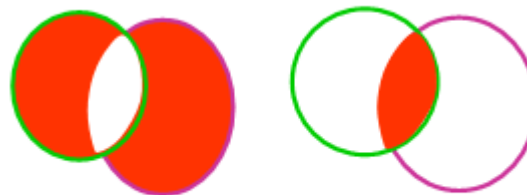
$$\mu_{A+B}(x) = \max[\mu_A(x), \mu_B(x)]$$

**Fuzzy intersection operation or fuzzy *AND***



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

**Complement operation**



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



## PROPERTIES OF FUZZY SETS

The same as for crisp sets

Commutativity

Associativity

Distributivity

Idempotency

Identity

De Morgan's Laws

...

### 1. Commutativity

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

### 2. Associativity

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

### 3. Distributivity

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

### 4. Idempotency

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

### 5. Identity

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

6. Involution (double negation)

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

7. Transitivity

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

8. Demorgan's law

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

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

# 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

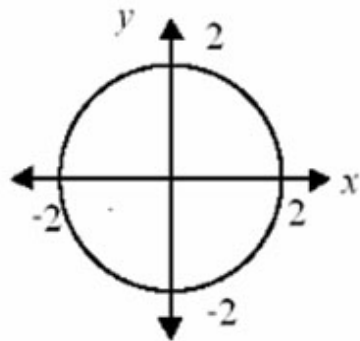
If  $A$  and  $B$  are two sets and there is a specific property between elements  $x$  of  $A$  and  $y$  of  $B$ , this property can be described using the ordered pair  $(x, y)$ . A set of such  $(x, y)$  pairs,  $x \in A$  and  $y \in B$ , is called a relation  $R$ .

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

# CRISP BINARY RELATIONS

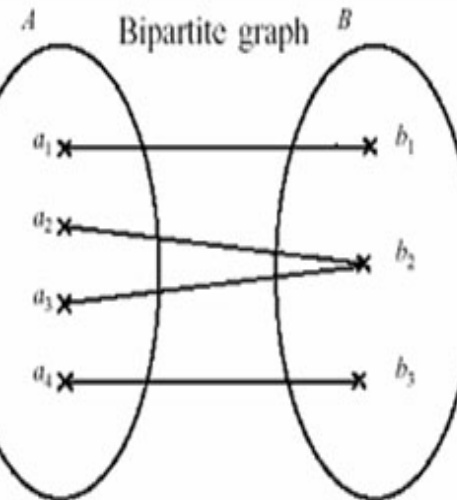
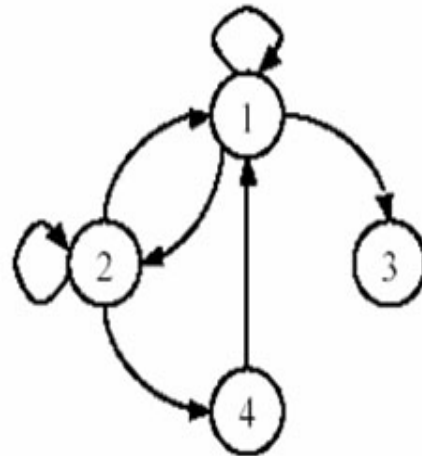
## Examples of binary relations

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

---

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

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

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

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

# FUZZY Relation

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

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

## 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 fuzzy relation  $R \times S$  is given by

$$\begin{array}{c} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \end{array} \begin{bmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} \\ \mathbf{1} & \mathbf{0.5} & \mathbf{0.9} \\ \mathbf{0.8} & \mathbf{0.5} & \mathbf{0.8} \\ \mathbf{0.2} & \mathbf{0.2} & \mathbf{0.2} \end{bmatrix}$$

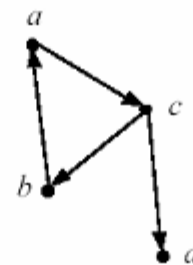
# FUZZY RELATION

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  $\mu_R(x,y)$

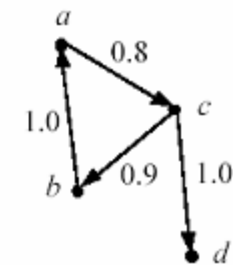
$$\mu_R : A \times B \rightarrow [0, 1]$$

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

## Crisp relation vs. Fuzzy relation



(a) Crisp relation



(b) Fuzzy relation

Corresponding fuzzy  
relation matrix

$A \backslash A$	$a$	$b$	$c$	$d$
$a$	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:

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

2. Intersection:

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

3. Complement:

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

4. Containment:

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

## 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

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

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

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

$M_{S \bullet R} = M_R \bullet M_S$  (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:

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

Using max-min composition,

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

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

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

$$= \vee_y [\mu_R(x, y) \bullet \mu_S(y, z)]$$

$$M_{S \bullet R} = M_R \bullet M_S \text{ (matrix notation)}$$

(max-product composition)

Max-product example:

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

Consider the following fuzzy relations:

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

Using max-product composition,

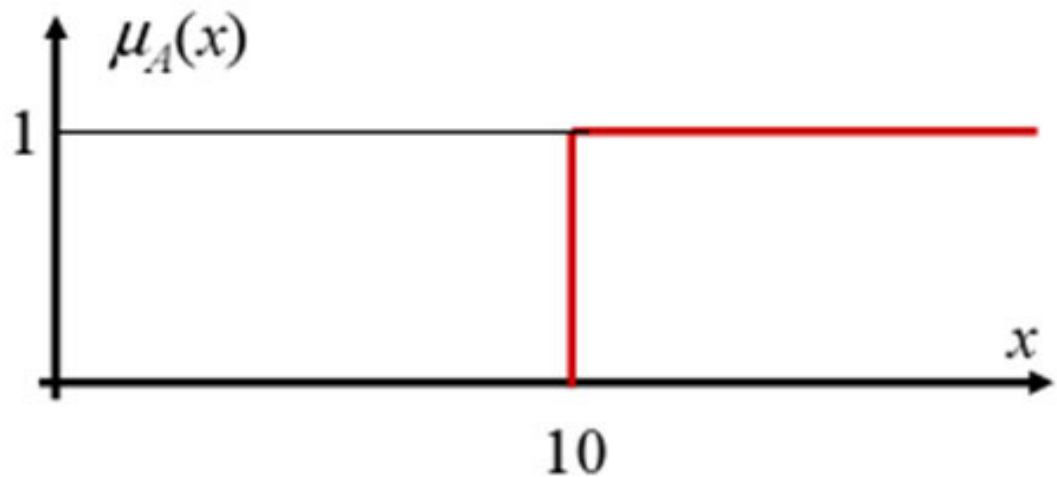
$$\left. \begin{aligned} \mu_{\tilde{T}}(x_2, z_2) &= \bigvee_{y \in Y} (\mu_{\tilde{R}}(x_2, y) \bullet \mu_{\tilde{S}}(y, z_2)) \\ &= \max[(0.8, 0.6), (0.4, 0.7)] \\ &= 0.48 \end{aligned} \right\} \tilde{T} = \begin{array}{c} z_1 \quad z_2 \quad z_3 \\ x_1 \begin{bmatrix} .63 & .42 & .25 \end{bmatrix}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

# **MEMBERSHIP FUNCTIONS**

# CRISP MEMBERSHIP FUNCTIONS

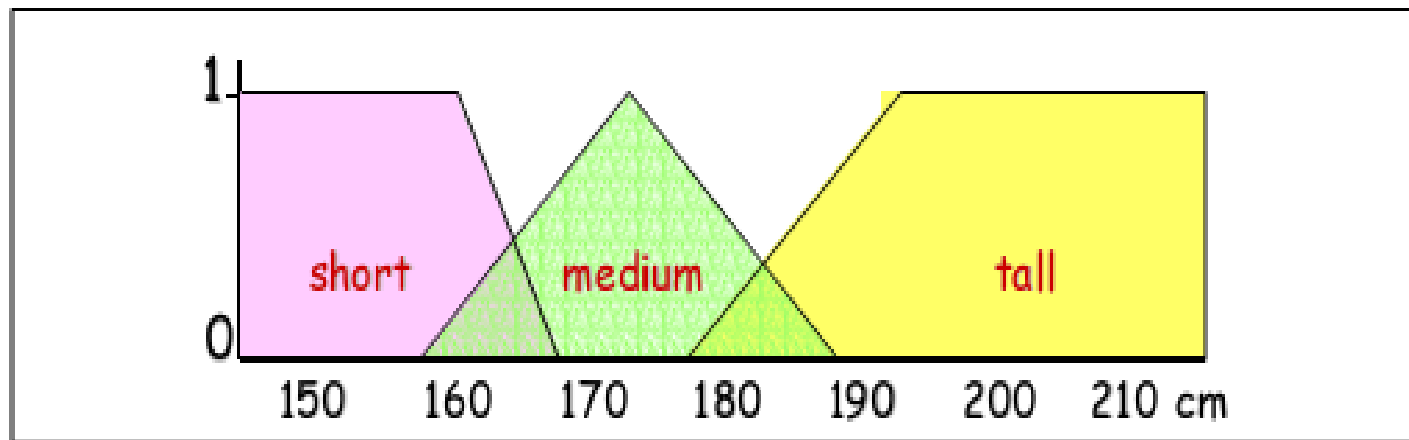
- Crisp membership functions ( $\mu$ ) are either one or zero.
- Consider the example: Numbers greater than 10. The membership curve for the set A is given by

$$A = \{x \mid x > 10\}$$



# 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:

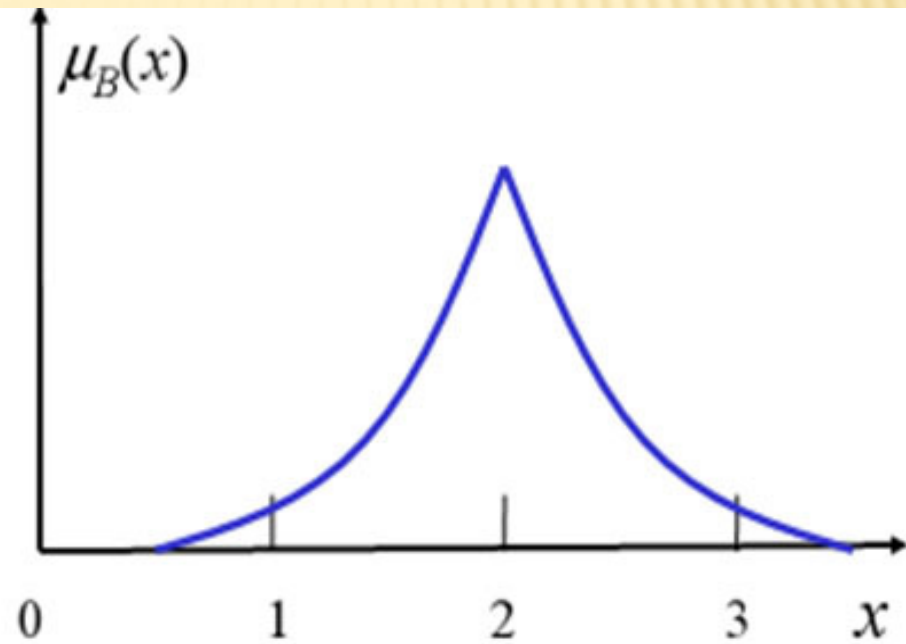
$$A = \{\mu_A(x_1)/x_1, \mu_A(x_2)/x_2, \dots, \mu_A(x_n)/x_n\}$$

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



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

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

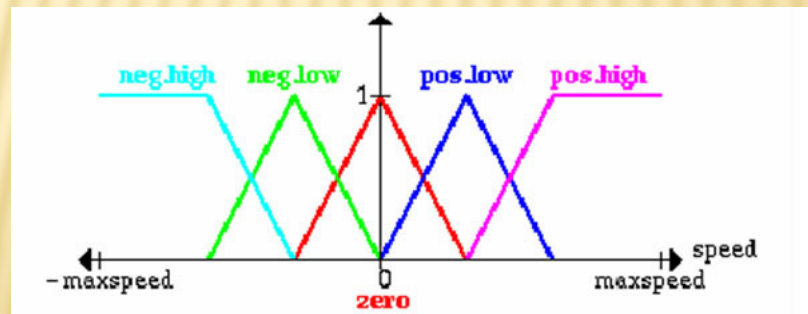


# LINGUISTIC VARIABLE

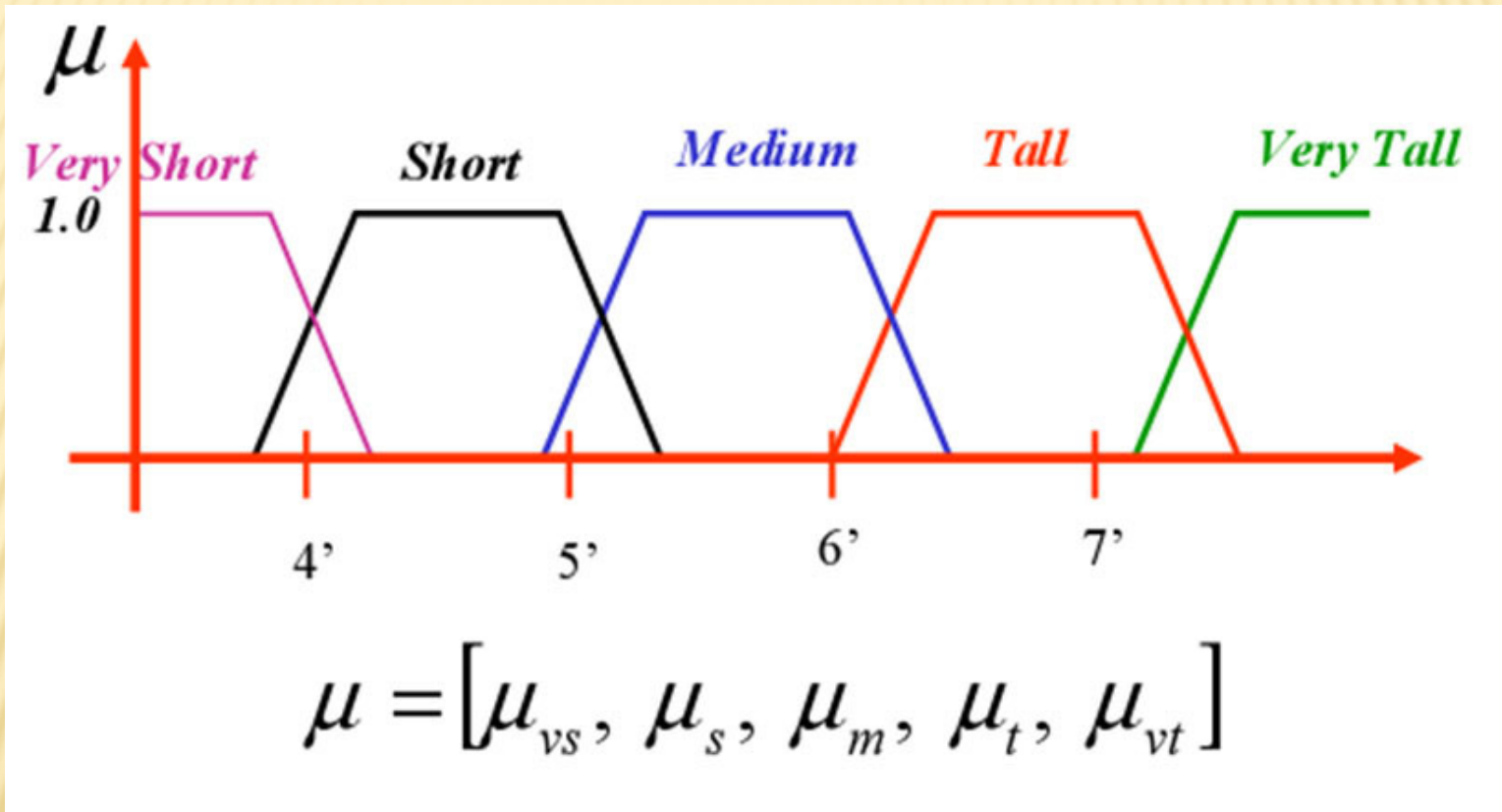
- Let  $x$  be a linguistic variable with the label "speed".
- Terms of  $x$ , which are fuzzy sets, could be "positive low", "negative high" from the term set  $T$ :

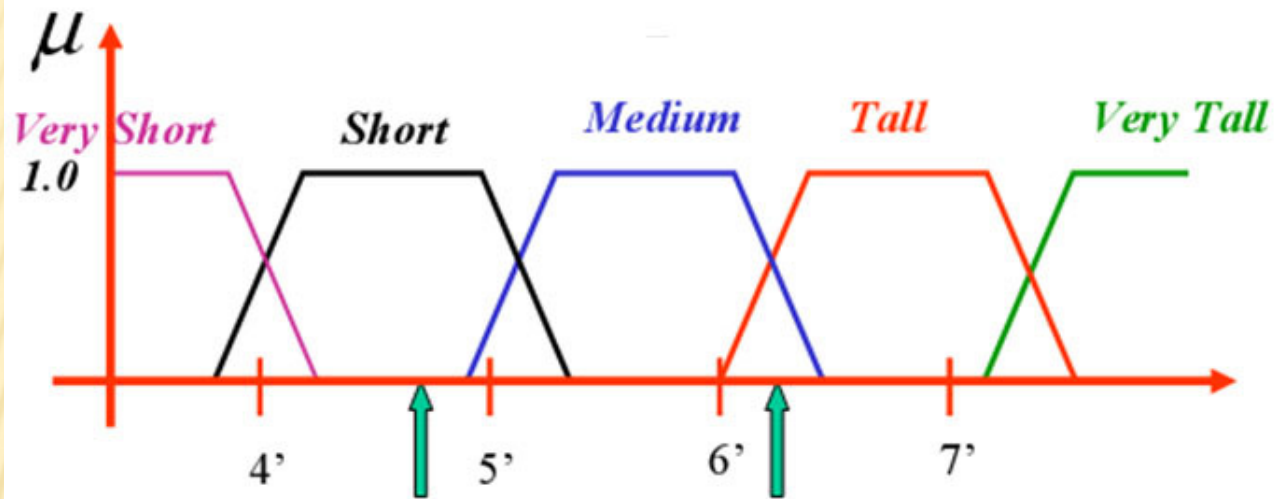
$$T = \{PositiveHigh, PositiveLow, NegativeLow, NegativeHigh, Zero\}$$

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



# MEMBERSHIP FUNCTIONS





Short

Medium Tall

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

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

# FEATURES OF MEMBERSHIP FUNCTIONS

➤ CORE:

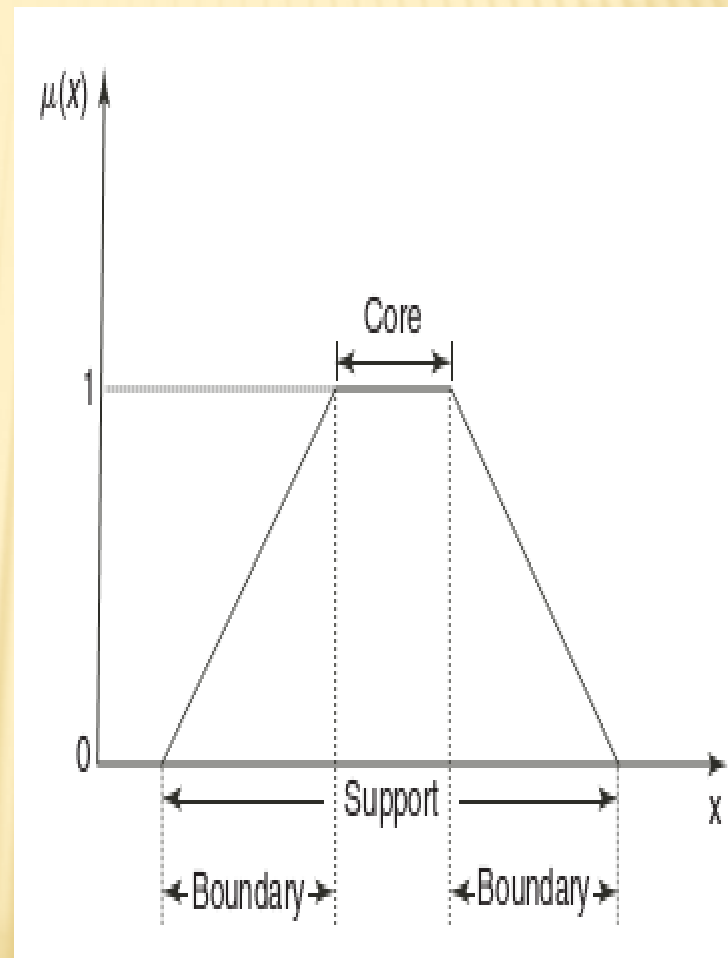
$$\mu_A(x) = 1$$

➤ SUPPORT:

➤ BOUNDARY:

$$\mu_A(x) > 0$$

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

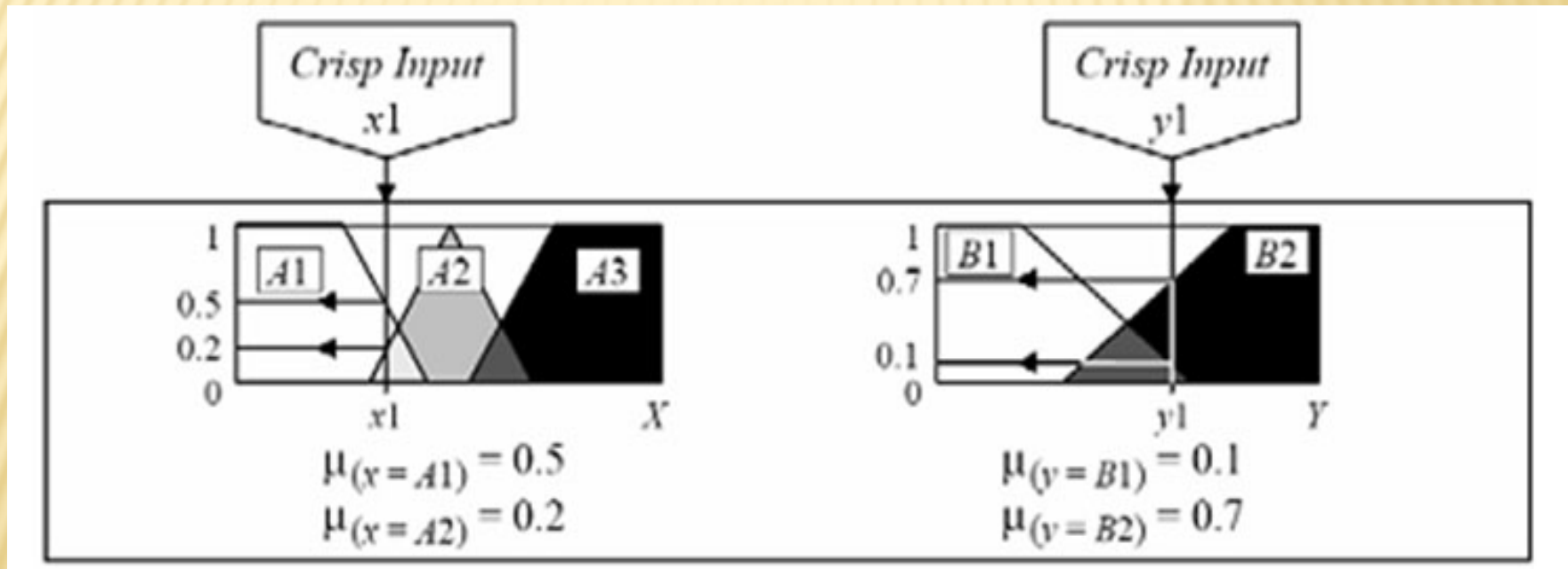


# **FUZZIFICATION**

---

- **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).

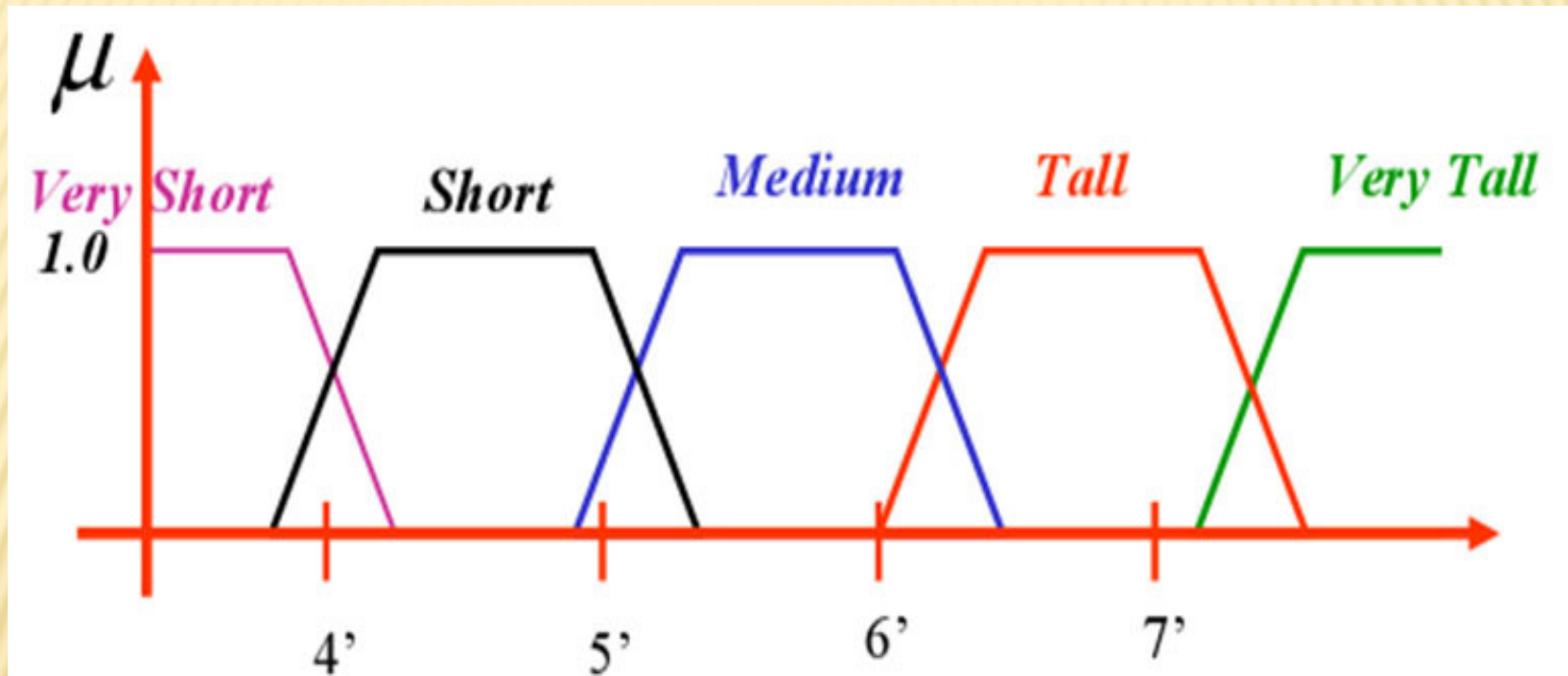


## **EXAMPLE - FUZZIFICATION**

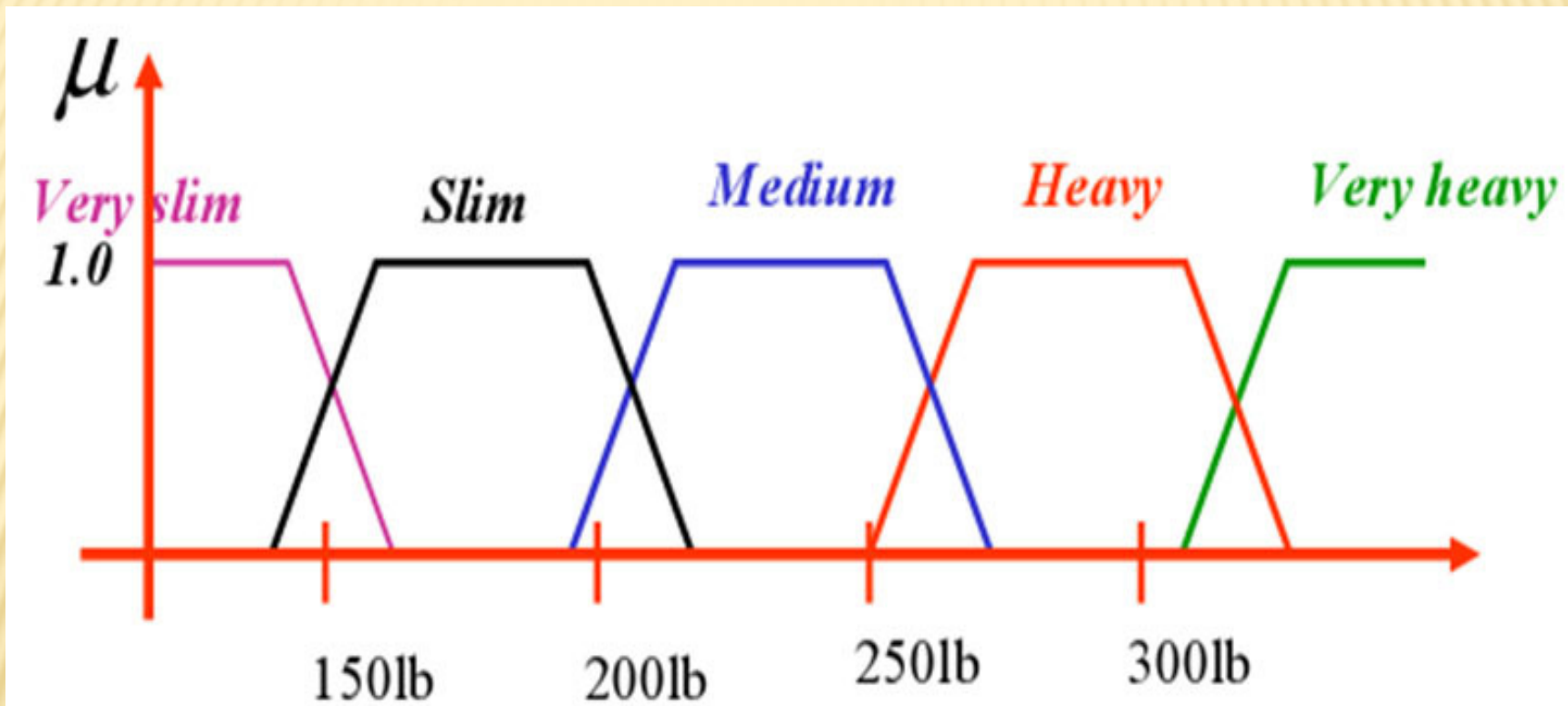
- **Assume we want to evaluate the health of a person based on his height and weight.**
- **The input variables are the crisp numbers of the person's height and weight.**
- **Fuzzification is a process by which the numbers are changes into linguistic words**



# FUZZIFICATION OF HEIGHT



# FUZZIFICATION OF WEIGHT



## 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:

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

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  $A_0 = X$ .

## LAMBDA CUT FOR FUZZY RELATIONS

Let  $\underline{R}$  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  $\underline{R}$  denotes a discrete membership function for a fuzzy set  $\underline{R}_j$ . A fuzzy relation can be converted into a crisp relation in the following manner:

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

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

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

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

# DEFUZZIFICATION

# 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.

# 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 INFERENCE SYSTEM**

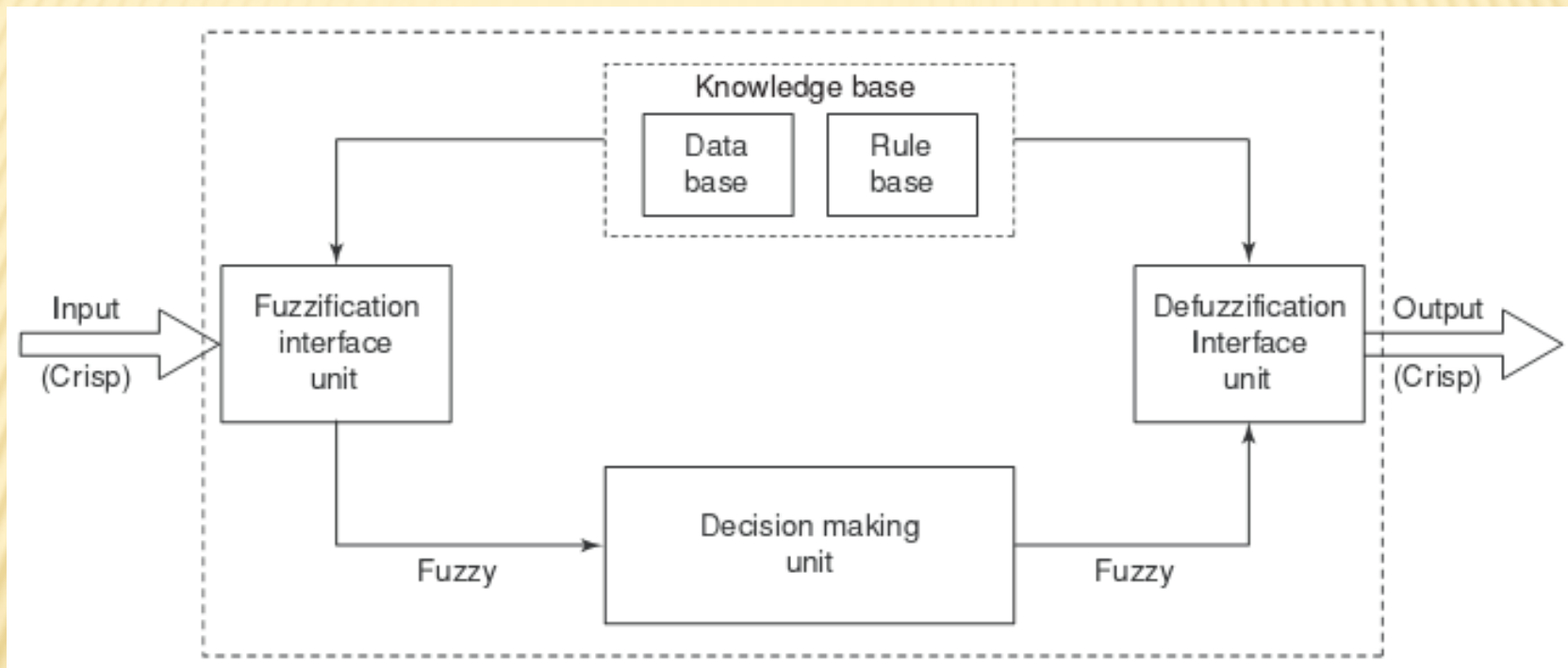


# FUZZY INFERENCE SYSTEMS (FIS)

---

- 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)

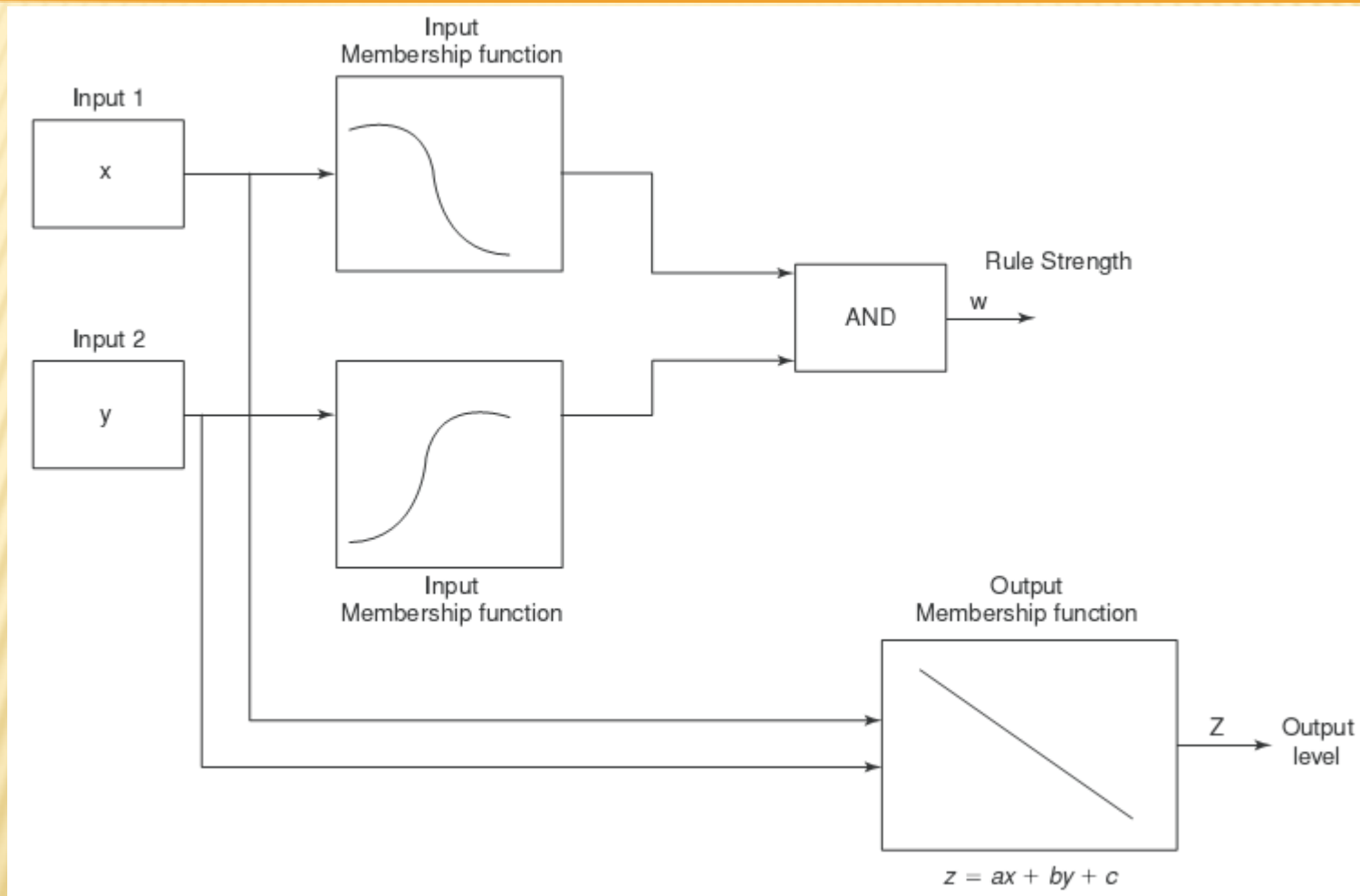
---

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.

## References

- Principles of soft computing by S.N.Deepa and S.N. Shivanandan
- Multi objective optimization using evolutionary algorithms by Kalyanmoy deo
- Genetic algorithms in search ,optimization and machine learning by David E.goldberg
- Neural Networks and Fuzzy Systems by Bart Kosko
- Neural Network ,Fuzzy Logic and Genetic Algorithm by Rahshekhran and Pai
- **Note :1. Students are advised to refer online resources and above mentioned books to get more information related to soft computing**  
**2. Figures, formulae and many slides are taken as it is from the CD of book of soft computing techniques by shivanandan and Deepa**