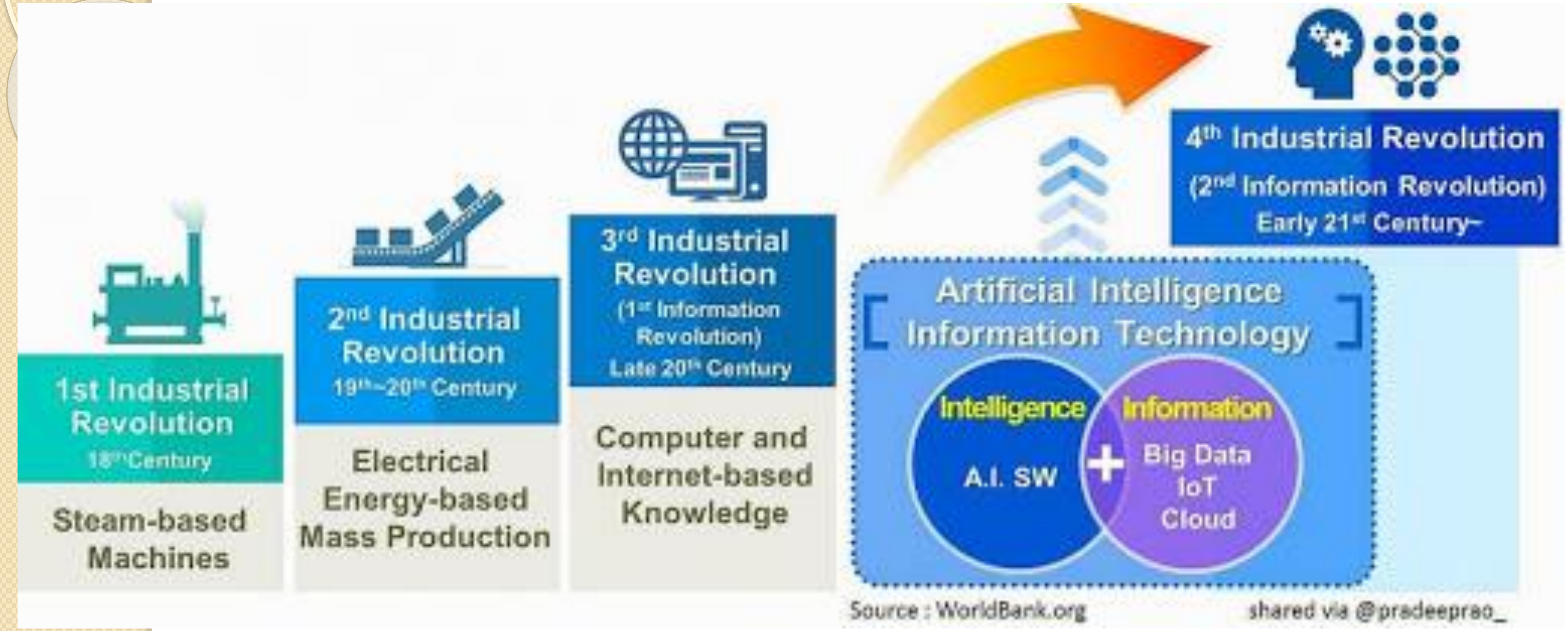
Introduction to Artificial Intelligence and its applications

Dr. Mohammed Abu Mallouh
Mechatronics Eng.
Hashemite University
Zarqa, Jordan

•4th Industrial revolution.


1st Industrial Revolution
18th Century
Steam-based Machines
2nd Industrial Revolution
19th~20th Century
Electrical Energy-based Mass Production
3rd Industrial Revolution
(1st Information Revolution)
Late 20th Century
Computer and Internet-based Knowledge
4th Industrial Revolution
(2nd Information Revolution)
Early 21st Century~
Artificial Intelligence
Information Technology
Intelligence
A.I. SW
Information
Big Data
IoT
Cloud
Source : WorldBank.org
shared via @pradeeprao_

https://www.weforum.org/about/the-fourth-industrial-revolution-by-klaus-schwab

•Unmanned security Boat

Play : AI boat

https://www.13newsnow.com/article/news/national/military-news/navy-turns-to-artificial-intelligence-for-security/291-14e879b3-49ce-4a08-a6d9-618e2bffc45b

What is Intelligence?

Intelligence is related with human ability to think, to understand, to learn, to store and recall fact, to make decisions and to solve a given problem based on known fact and relevant theorem.

What is Artificial Intelligence (AI)

Artificial Intelligence (AI) is the ability of an electronic device (computer) to accomplish any tasks that ordinary would have been handled by human (to learn and act like human).

What AI can add to Machines

- Reasoning (being logical/sensible): a way to infer facts from existing data. It is a general process of thinking rationally, to find valid conclusions.
- Learning: the ability to acquire knowledge from data and/or interactions with the environment.
- Problem solving: the ability to employ learning to solve a wide range of tasks.
- Perception: the ability to understand from learning.
- Decision making: the ability to make informed decision based on the learning.

When to call a machine… Intelligent?

- The term artificial intelligence was first coined by John McCarthy in 1956 when he held the first academic conference on the subject.

- Isn't there a solid definition of intelligence that doesn't depend on relating it to human intelligence? Not yet.

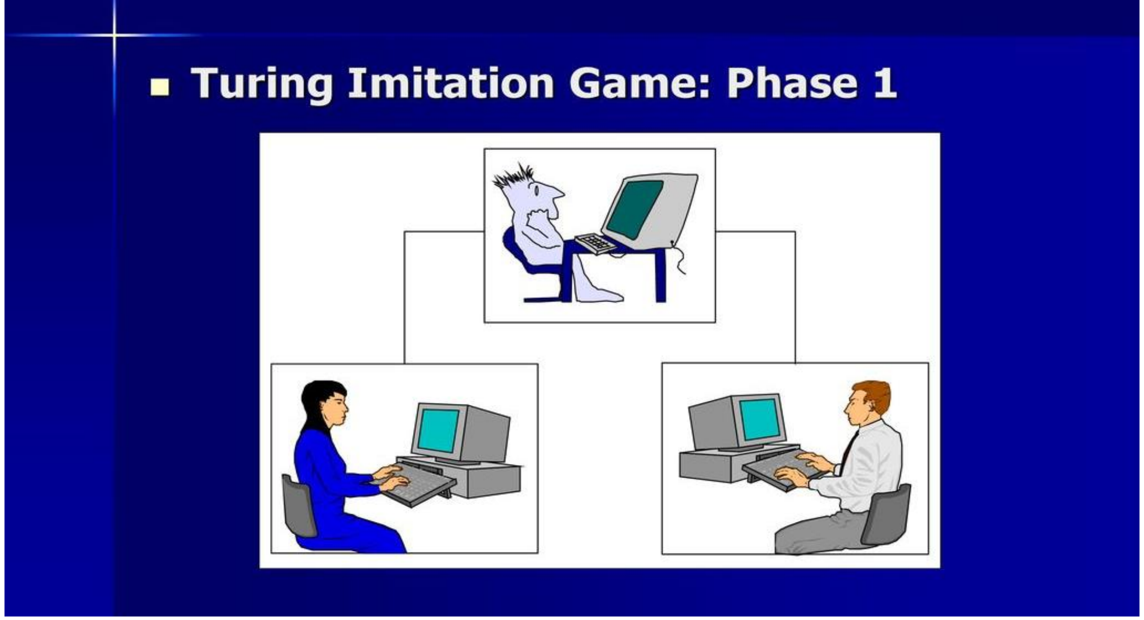
- The problem is that we cannot yet characterize in general what kinds of computational procedures we want to call intelligent.

When to call a machine ..Intelligent?

- One of the most significant papers on machine intelligence, “Computing Machinery and Intelligence’’, was written by the British mathematician Alan Turing over fifty years ago .

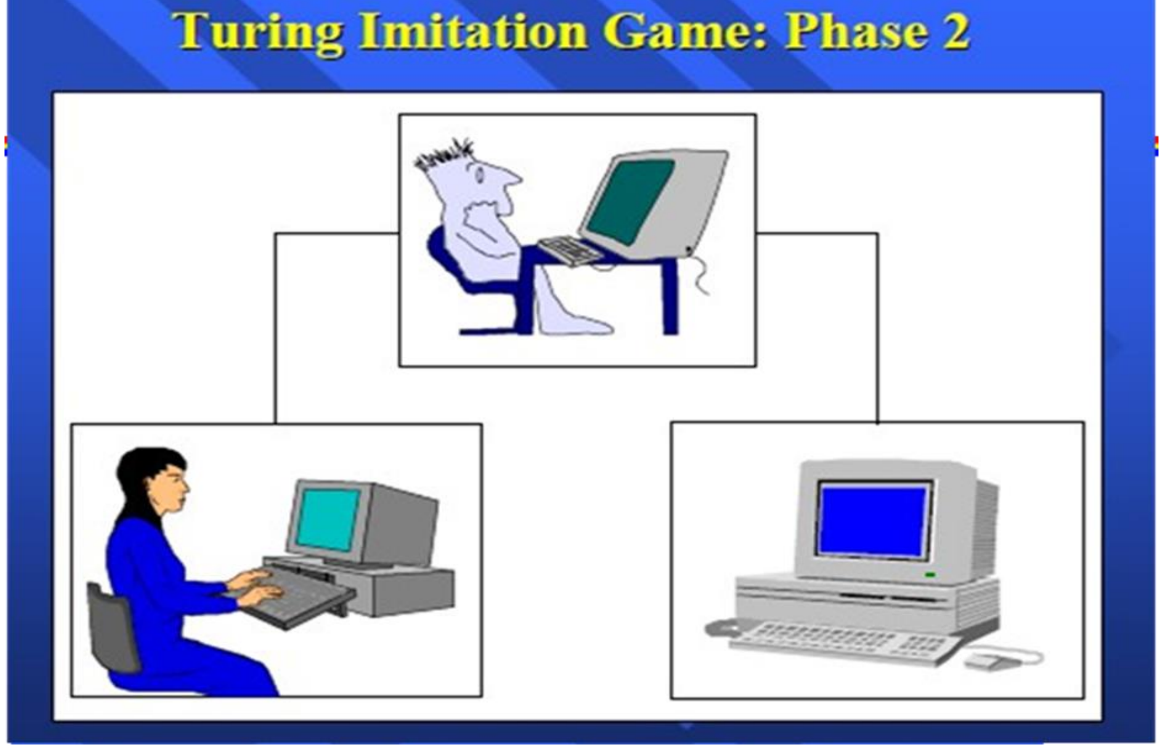
- Turing did not provide definitions of machines and AI, he just invented a game , the Turing Imitation Game.

- The machine (computer) will be called intelligent if it pass the Turing imitation game

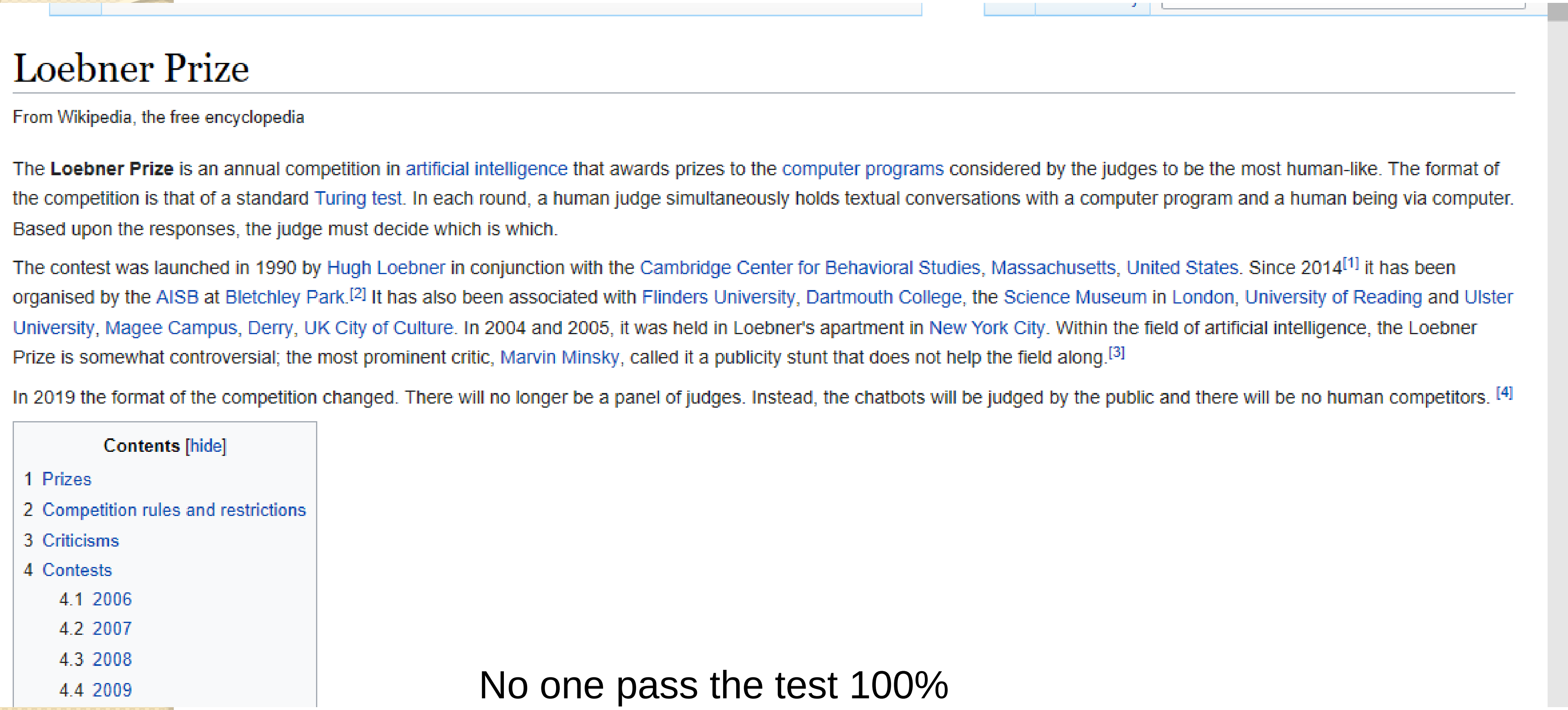


- The imitation game originally included two phases:

1. In the first phase, the interrogator, a man and a woman are each placed in separate rooms. The interrogator's objective is to work out who is the man and who is the woman by questioning them. The man should attempt to deceive the interrogator that he is the woman, while the woman has to convince the interrogator that she is the woman.



2. In the second phase of the game, the man is replaced by a computer programmed to deceive the interrogator as the man did. It would even be programmed to make mistakes and provide fuzzy answers in the way a human would . If the computer can fool the interrogator as often as the man did, we may say this computer has passed the intelligent behavior test.



Loebner Prize

From Wikipedia, the free encyclopedia

The **Loebner Prize** is an annual competition in artificial intelligence that awards prizes to the computer programs considered by the judges to be the most human-like. The format of the competition is that of a standard Turing test. In each round, a human judge simultaneously holds textual conversations with a computer program and a human being via computer. Based upon the responses, the judge must decide which is which.

The contest was launched in 1990 by Hugh Loebner in conjunction with the Cambridge Center for Behavioral Studies, Massachusetts, United States. Since 2014[1] it has been organised by the AISB at Bletchley Park.[2] It has also been associated with Flinders University, Dartmouth College, the Science Museum in London, University of Reading and Ulster University, Magee Campus, Derry, UK City of Culture. In 2004 and 2005, it was held in Loebner's apartment in New York City. Within the field of artificial intelligence, the Loebner Prize is somewhat controversial; the most prominent critic, Marvin Minsky, called it a publicity stunt that does not help the field along.[3]

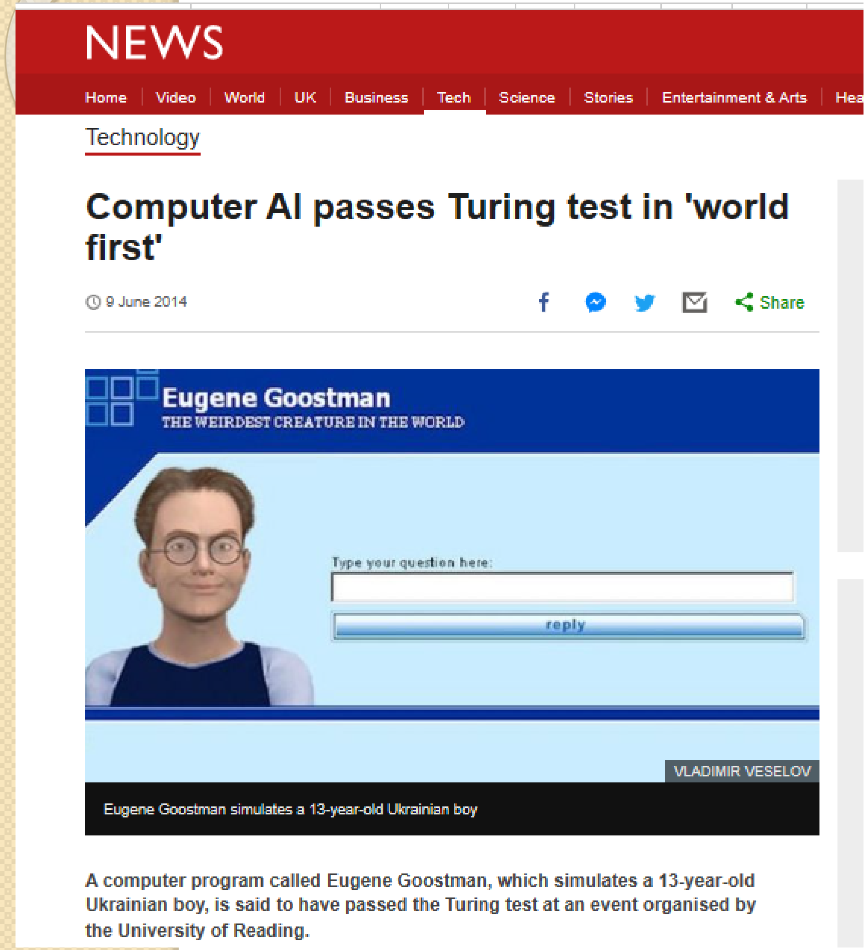
In 2019 the format of the competition changed. There will no longer be a panel of judges. Instead, the chatbots will be judged by the public and there will be no human competitors. [4]

Contents [hide]

No one pass the test 100%

https://en.wikipedia.org/wiki/Loebner_Prize#2014

A computer program called Eugene Goostman, which simulates a 13-year-old Ukrainian boy, is said to have passed the Turing test at an event organized by the University of Reading.


NEWS
Home | Video | World | UK | Business | Tech | Science | Stories | Entertainment & Arts | Hea
Share
Eugene Goostman
THE WEIRDEST CREATURE IN THE WORLD
Type your question here:
reply
VLADIMIR VESELOV

Technology

Computer AI passes Turing test in 'world first'

9 June 2014

Eugene Goostman simulates a 13-year-old Ukrainian boy

A computer program called Eugene Goostman, which simulates a 13-year-old Ukrainian boy, is said to have passed the Turing test at an event organised by the University of Reading.

On 7 June 2014, at a contest marking the 60th anniversary of Turing's death, 33% of the event's judges thought that Goostman was human;

https://www.bbc.com/news/technology-27762088

AI vs human

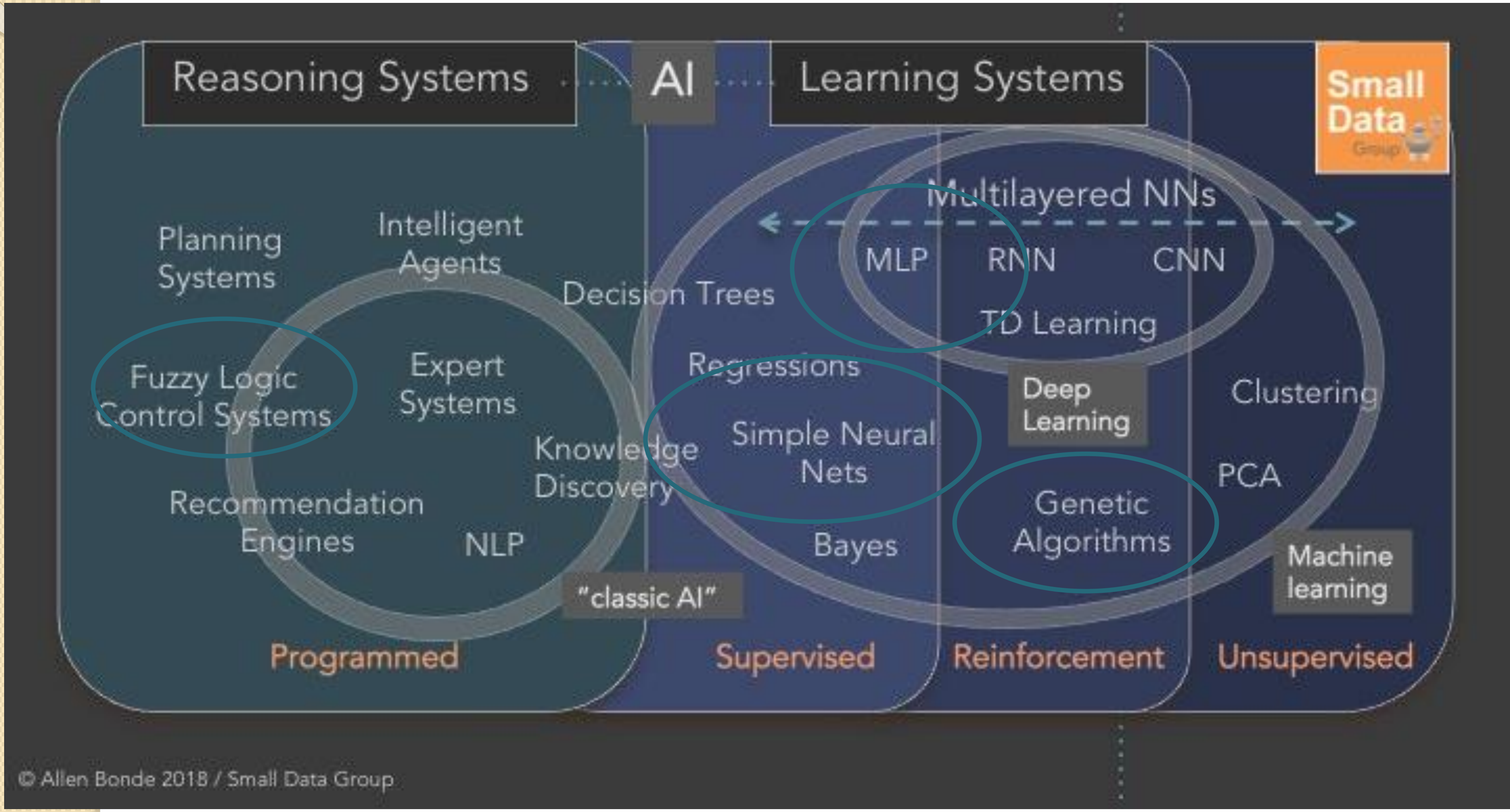
1. Computational models of human behavior?

   Algorithms/techniques (Programs) that behave (externally) like humans

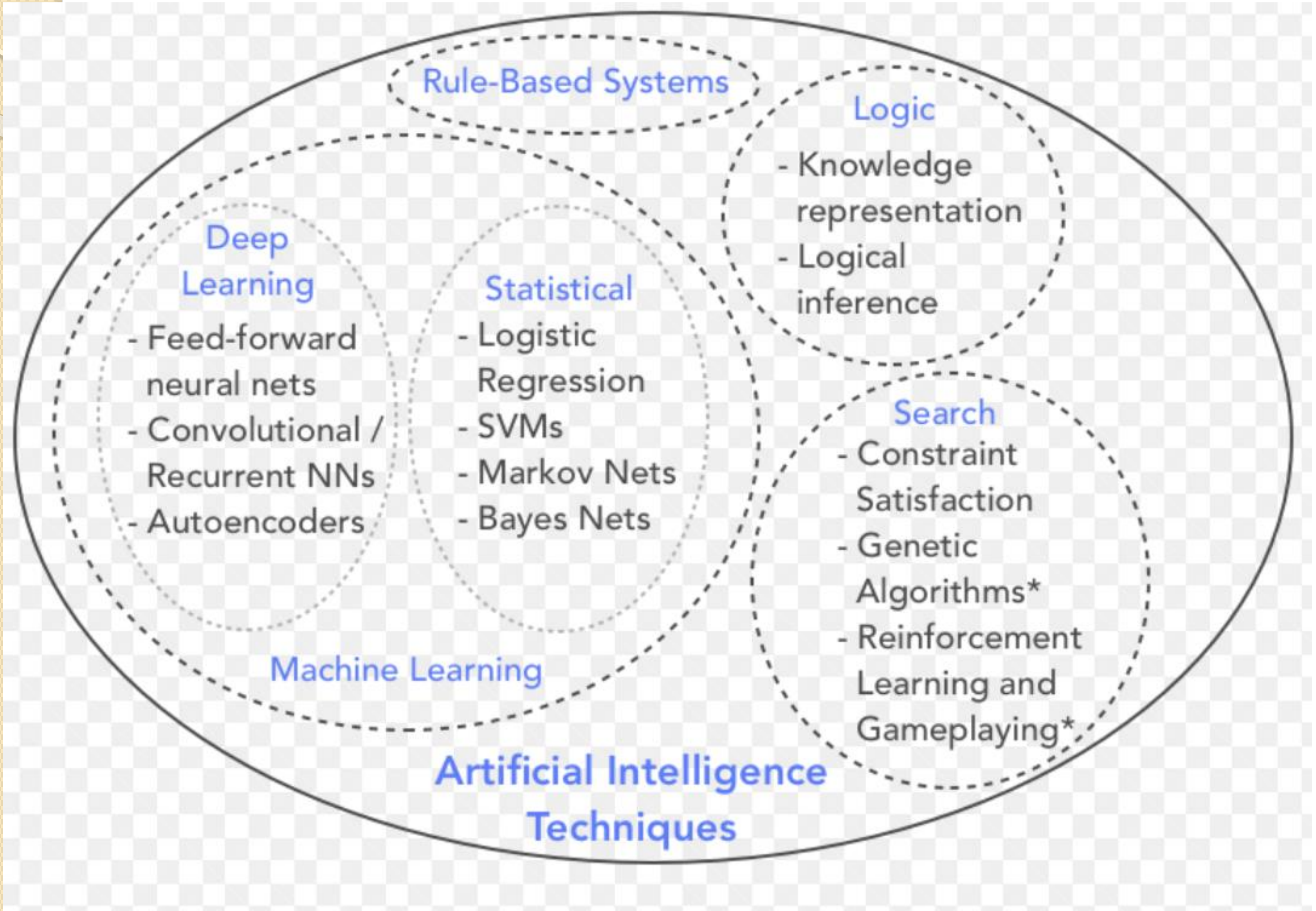
2. Computational models of human “thought”

   Algorithms/techniques Programs that operate (internally) the way humans do

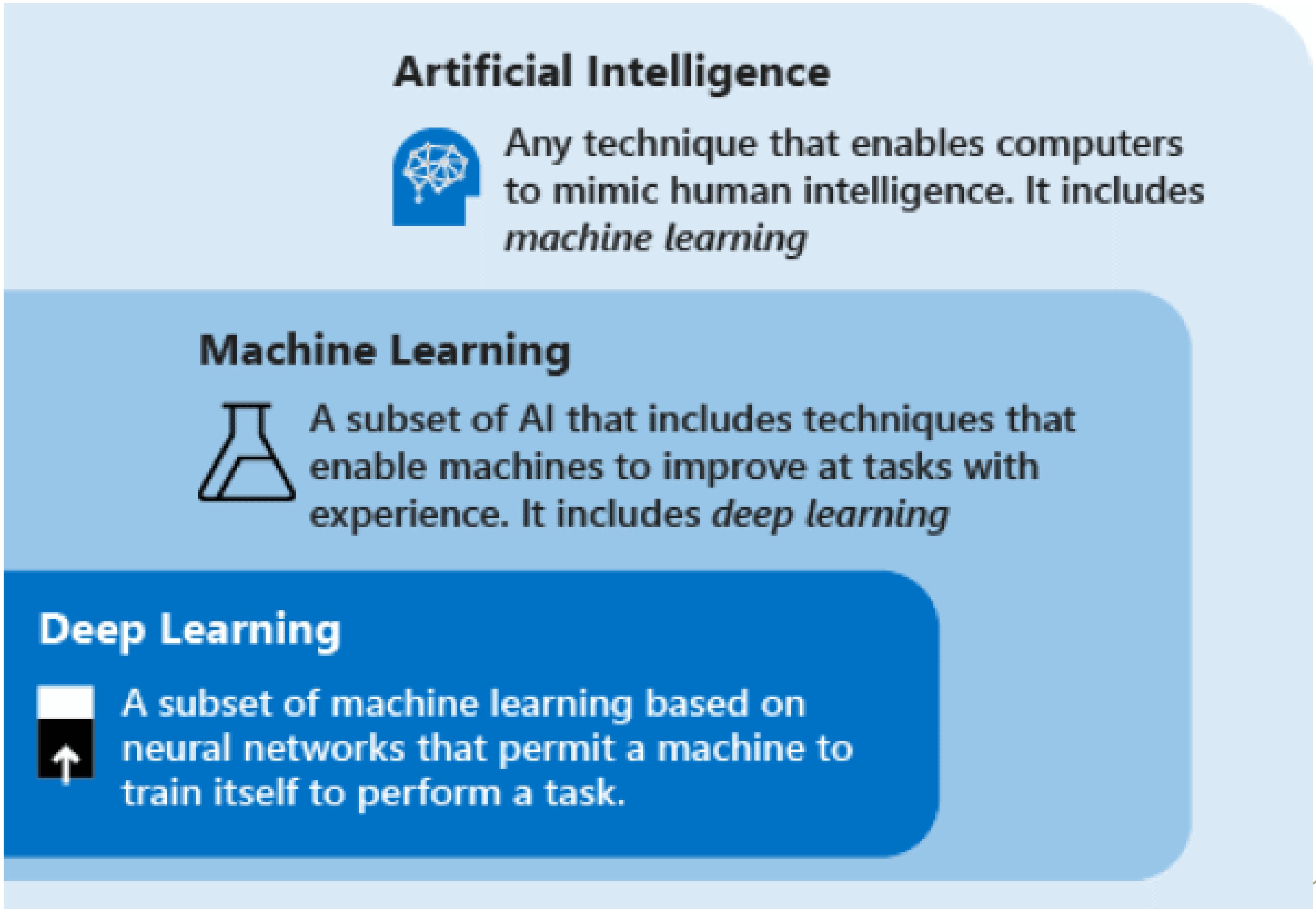
AI Taxonomy


Reasoning Systems
AI
Learning Systems
Small Data Group
Planning Systems
Intelligent Agents
Decision Trees
Multilayered NNs
MLP
RNN
CNN
TD Learning
Fuzzy Logic Control Systems
Expert Systems
Regressions
Deep Learning
Clustering
Knowledge Discovery
Simple Neural Nets
PCA
Recommendation Engines
NLP
Genetic Algorithms
Bayes
Machine learning
"classic AI"
Programmed
Supervised
Reinforcement
Unsupervised
© Allen Bonde 2018 / Small Data Group

AI Taxonomy


Rule-Based Systems
Logic
- Knowledge representation
- Logical inference
Deep Learning
- Feed-forward neural nets
- Convolutional / Recurrent NNs
- Autoencoders
Statistical
- Logistic Regression
- SVMs
- Markov Nets
- Bayes Nets
Search
- Constraint Satisfaction
- Genetic Algorithms*
- Reinforcement Learning and Gameplaying*
Machine Learning
Artificial Intelligence Techniques

Machine learning



Artificial Intelligence

Any technique that enables computers to mimic human intelligence. It includes *machine learning*

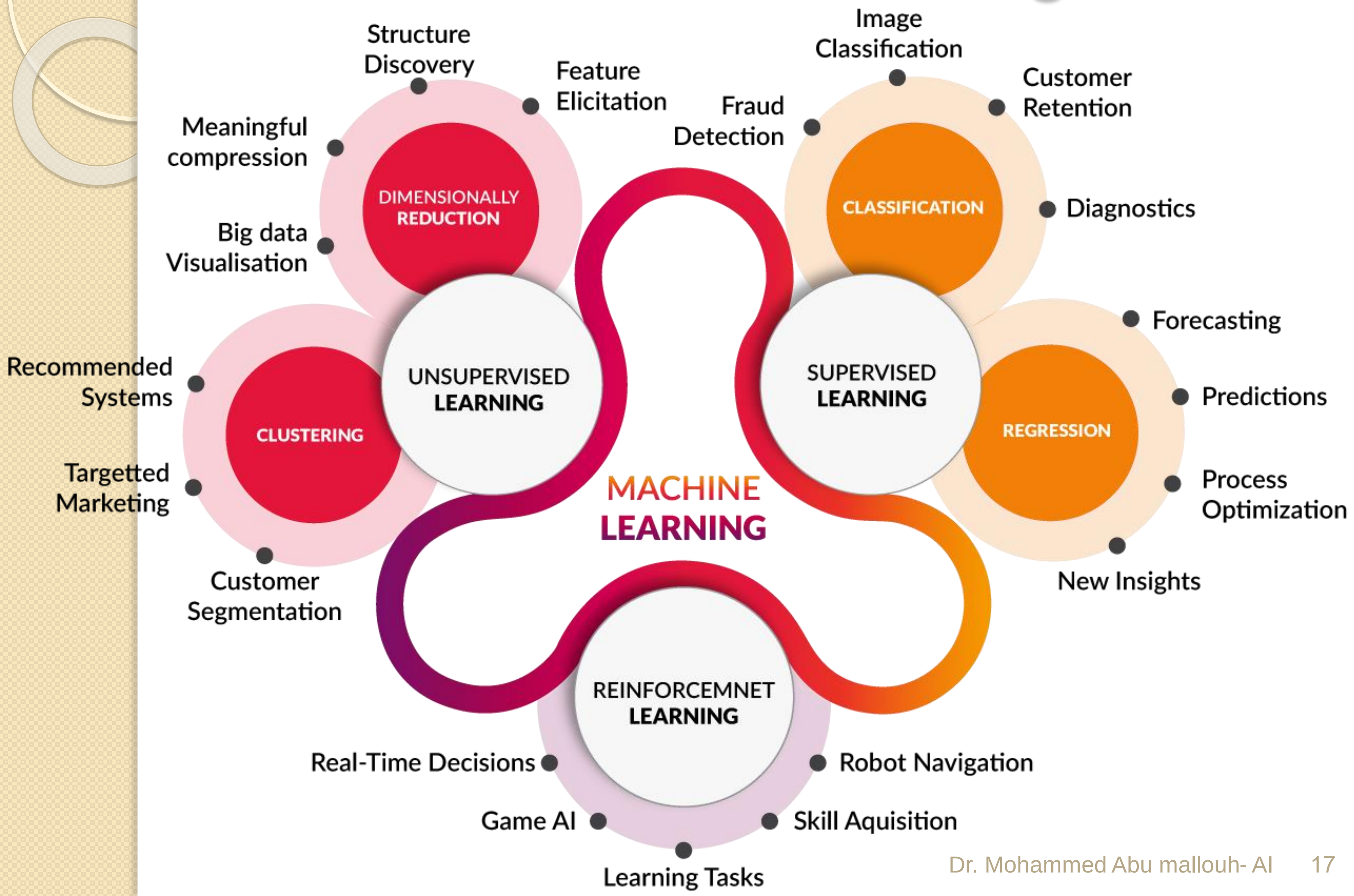
Machine Learning

A subset of AI that includes techniques that enable machines to improve at tasks with experience. It includes *deep learning*

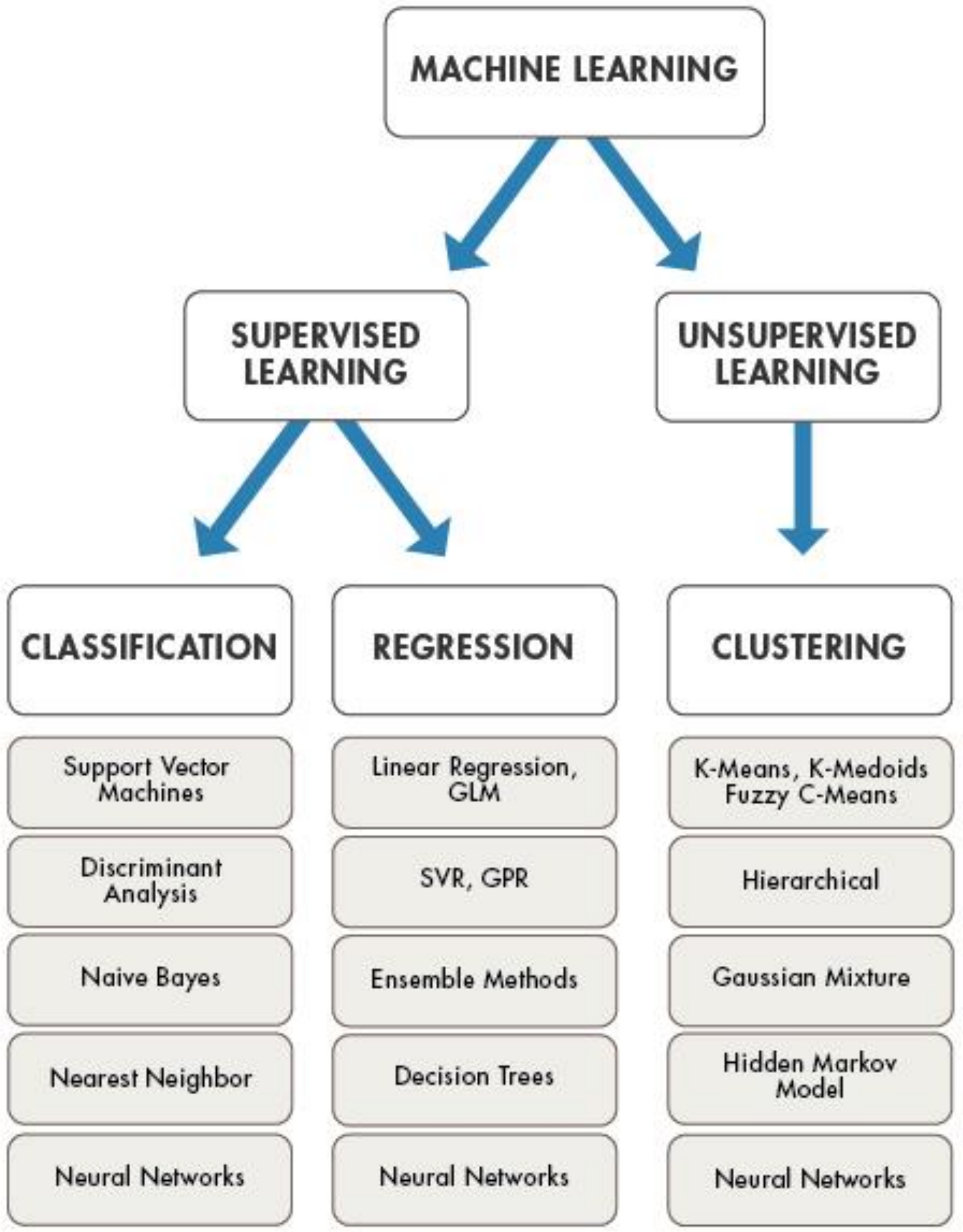
Deep Learning

A subset of machine learning based on neural networks that permit a machine to train itself to perform a task.

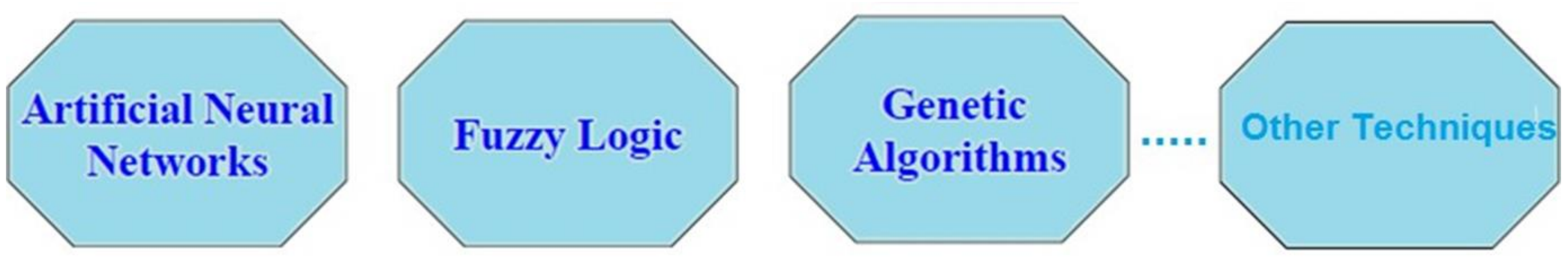
Machine learning



Machine learning



Artificial Intelligence Techniques



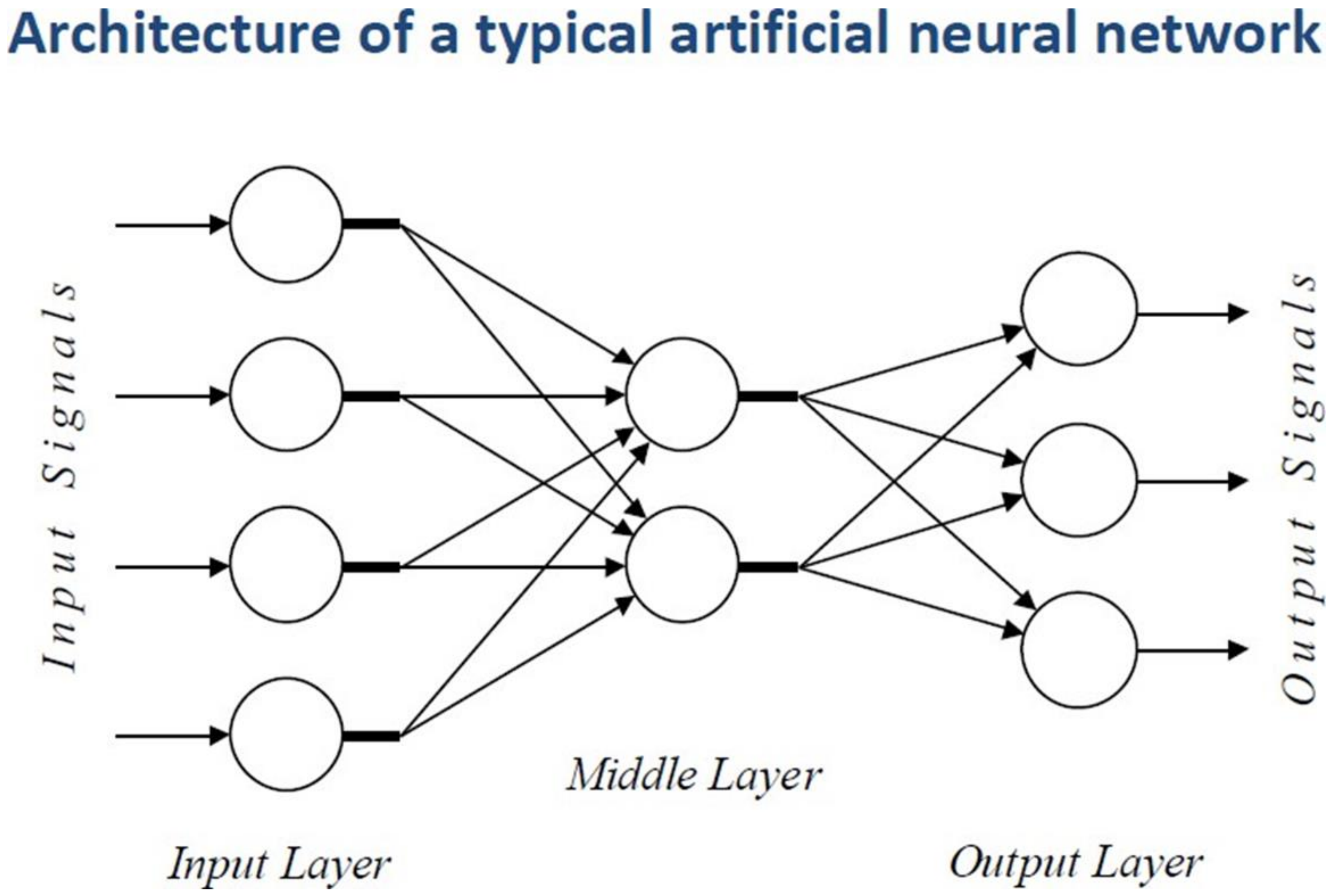
Applications

- Control
- Estimation
- System Identification
- Optimization

Artificial Neural network

- Computational models of human “thought”
- Programs that operate (internally) the way humans do

Artificial Neural network



Artificial Neural network

Advantages

- Learning capabilities
- Generalization
- No Mathematical model
- Fault tolerance
- Parallel processing

Drawbacks

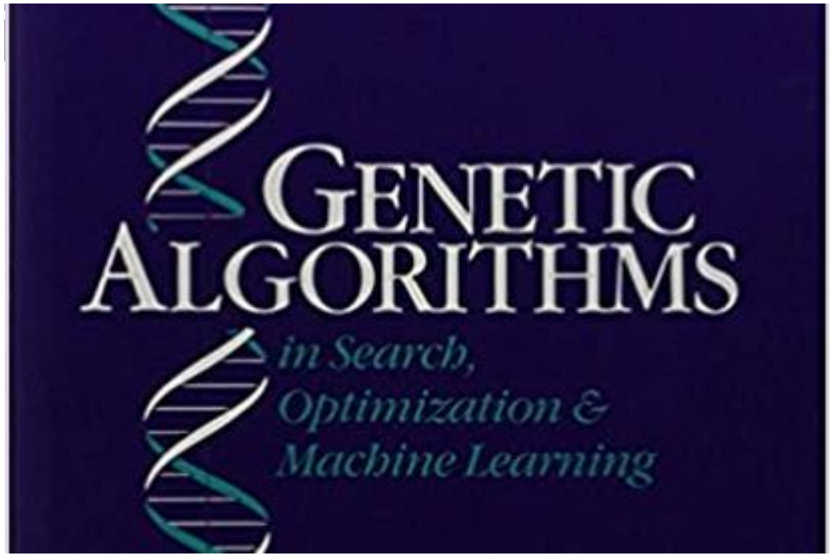
- Lack of design techniques
- Computational effort

Applications

- Control
- Estimation
- System Identification
- Optimization

Genetic algorithms

- Evolutionary computation, or learning by doing (early 1970s-onwards)
- Natural intelligence is a product of evolution, Therefore, by simulating biological evolution, we might expect to discover how living systems are propelled towards high-level



Genetic algorithms

- Computational models of human "thought"

- Programs that operate (internally) the way humans do

- The Genetic algorithms is based on the natural selection and genetics.
- Genetic algorithm works by simulating a population of individuals, evaluating their performance, generating a new population, and repeating this process a number of times.

-Advantages

- Derivative free
- Avoid local minimal

-Application

- Optimization
- Parameter tuning and estimation

Fuzzy logic

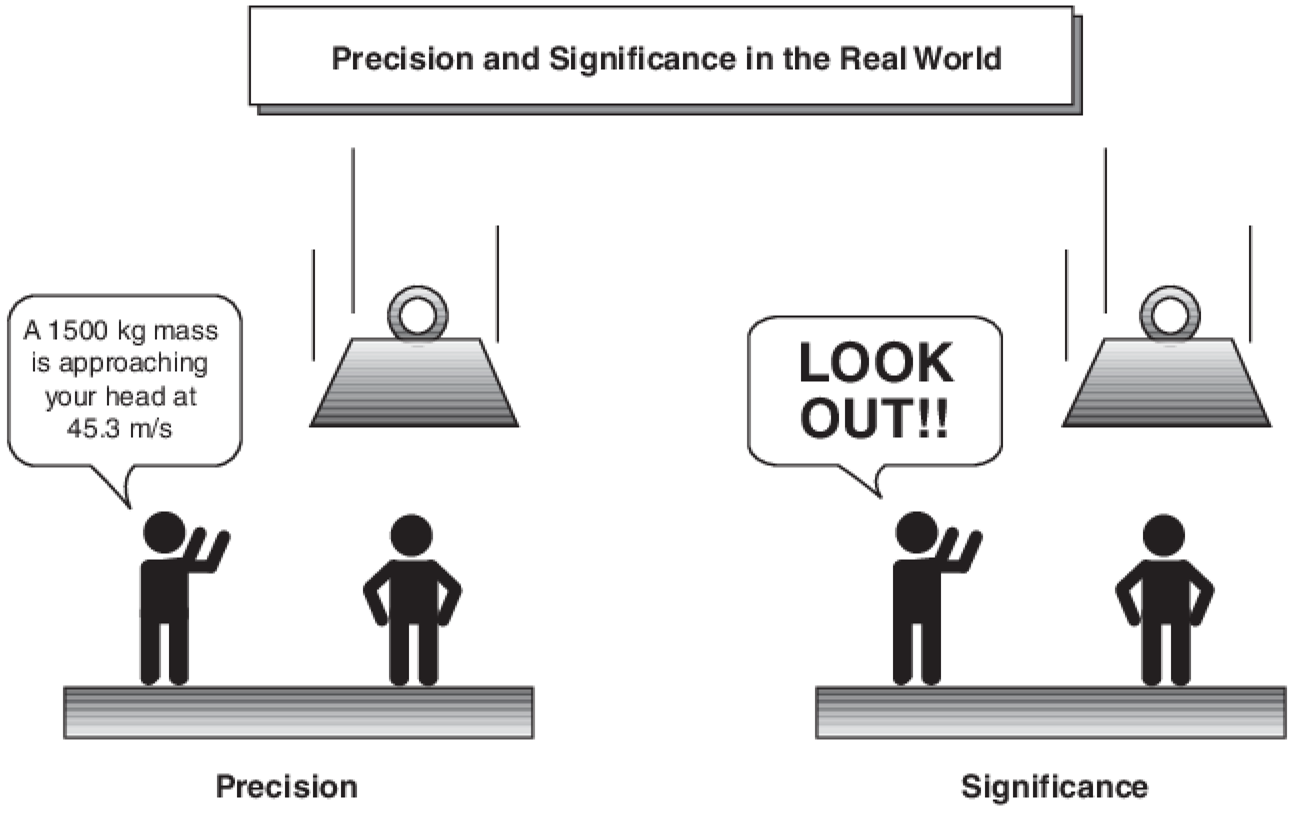
- Computational models of human behavior?
- Programs that behave (externally) like humans
- The new era of knowledge engineering, or computing with words (late 1980s-onwards)

- Human experts do not usually think in probability values, but in such terms as often, generally, sometimes, occasionally and rarely.

- Fuzzy logic is concerned with capturing the meaning of words, human reasoning and decision making.

Fuzzy logic

- At the heart of fuzzy logic lies the concept of a linguistic variable. The values of the linguistic variable are words rather than numbers.


Precision and Significance in the Real World
A 1500 kg mass is approaching your head at 45.3 m/s
LOOK OUT!!
Precision
Significance

Fuzzy logic

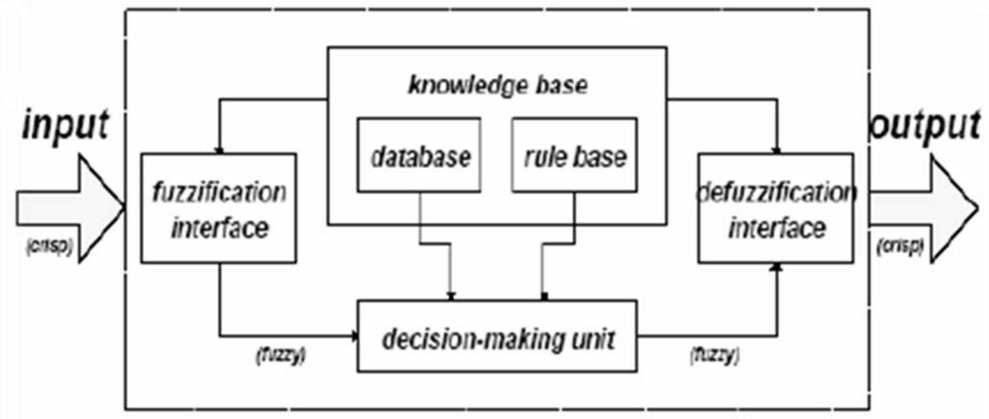
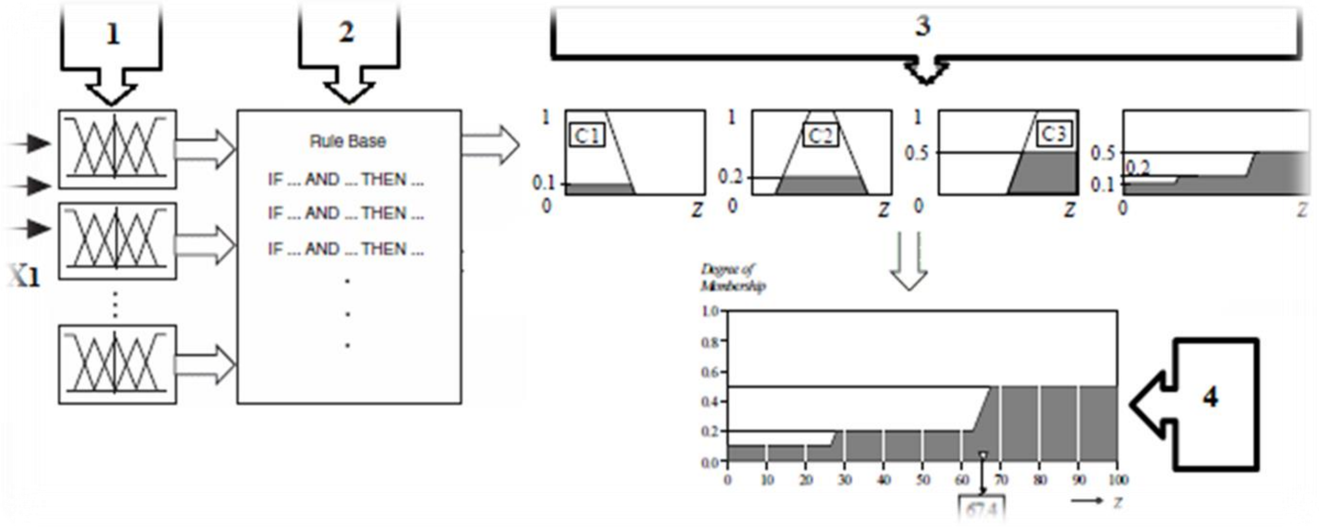


Fig. 4. Fuzzy Logic system structure.



How old is AI? The history of AI

- The birth of artificial intelligence(1943-1956)
- The rise of artificial intelligence, or the era of great expectations (1956-late 1960s)
- In the sixties, AI researchers attempted to simulate the thinking process by inventing general methods for solving broad classes of problems. They used the general-purpose search mechanism to find a solution to the problem. Such approaches, now referred to as weak methods, applied weak information about the problem domain.
- Unfulfilled promises, or the impact of reality (late 1960s-early 1970s)
- The main difficulties for AI in the late 1960s were :

Because AI researchers were developing general methods for broad classes of problems, early programs contained little or even no knowledge about a problem domain.

How old is AI? The history of AI

- The technology of expert systems, or the key to success(early 1970s -mid -1980s)

- Probably the most important development in the seventies was the realization that the domain for intelligent machines had to be sufficiently restricted.
- When weak methods failed, researchers finally realized that the only way to deliver practical results was to solve typical cases in narrow areas of expertise, making large reasoning steps.

- A 1986 survey reported a remarkable number successful expert system applications in different areas: chemistry, electronics, engineering, geology, management, medicine, process control and military science (Waterman,1986). Although Waterman found nearly 200 expert systems , most of the applications were in the field of medical diagnosis. Seven years later a similar survey reported over 2500 developed expert systems (Durkin,1994). The new growing area was business and manufacturing, which accounted for about 60% of the applications. Expert system technology had clearly matured.

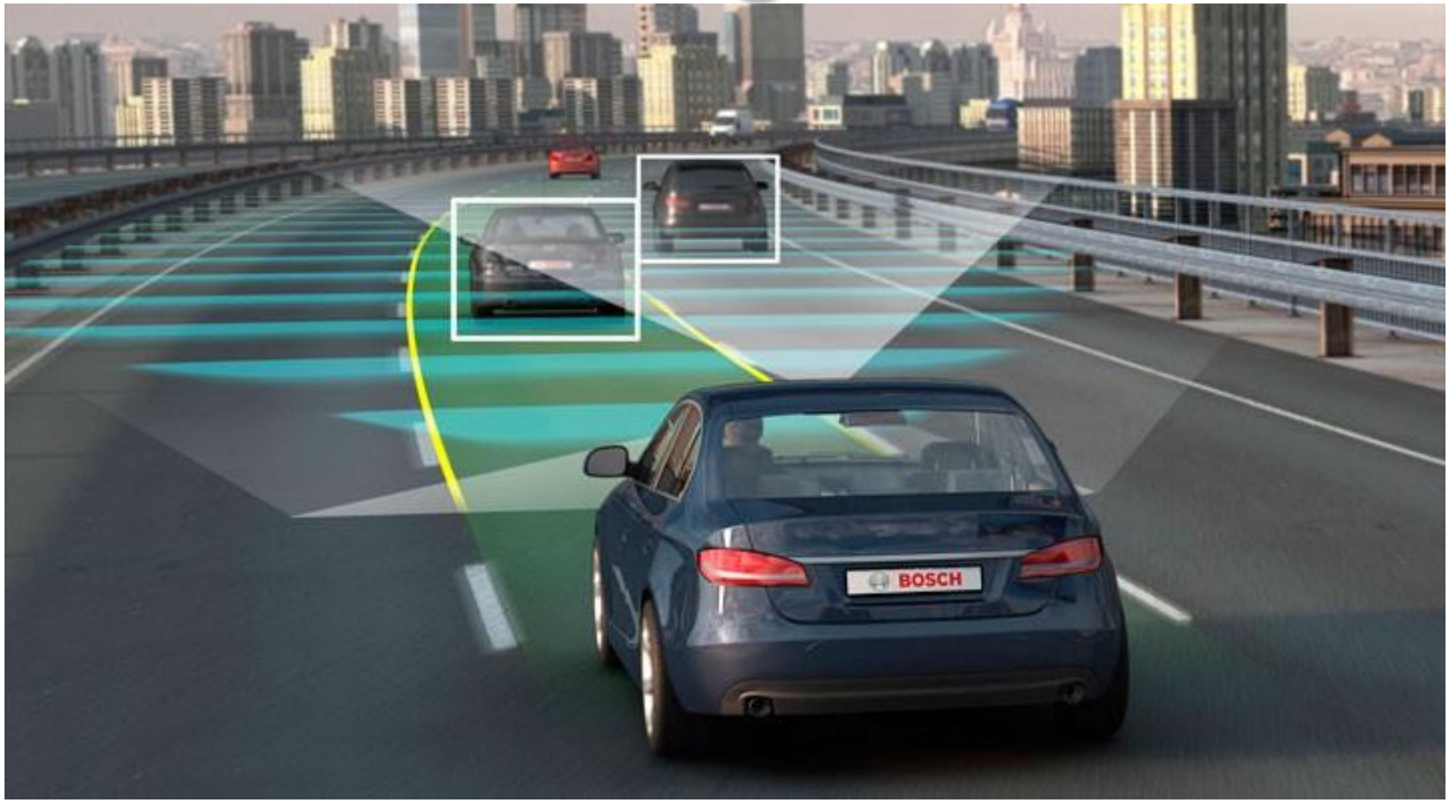
Applications and examples of AI

•Deep Blue defeated the world chess champion Garry Kasparov in 1997.

•In 1997, the Deep Blue chess program created by IBM, beat the current world chess champion, Gary Kasparov.

https://www.aaai.org/Papers/Workshops/1997/WS-97-04/WS97-04-001.pdf

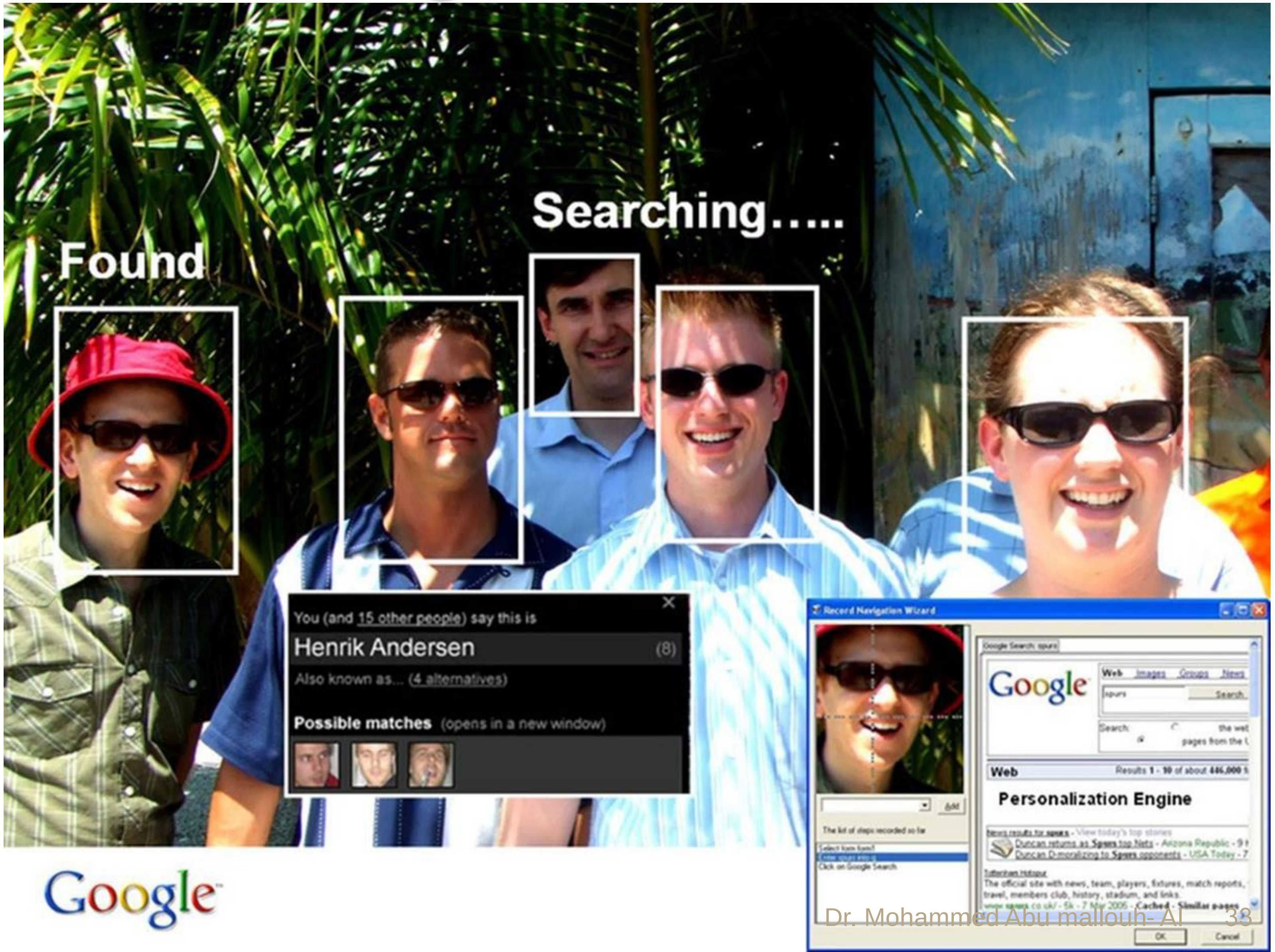
Autonomous ground vehicle



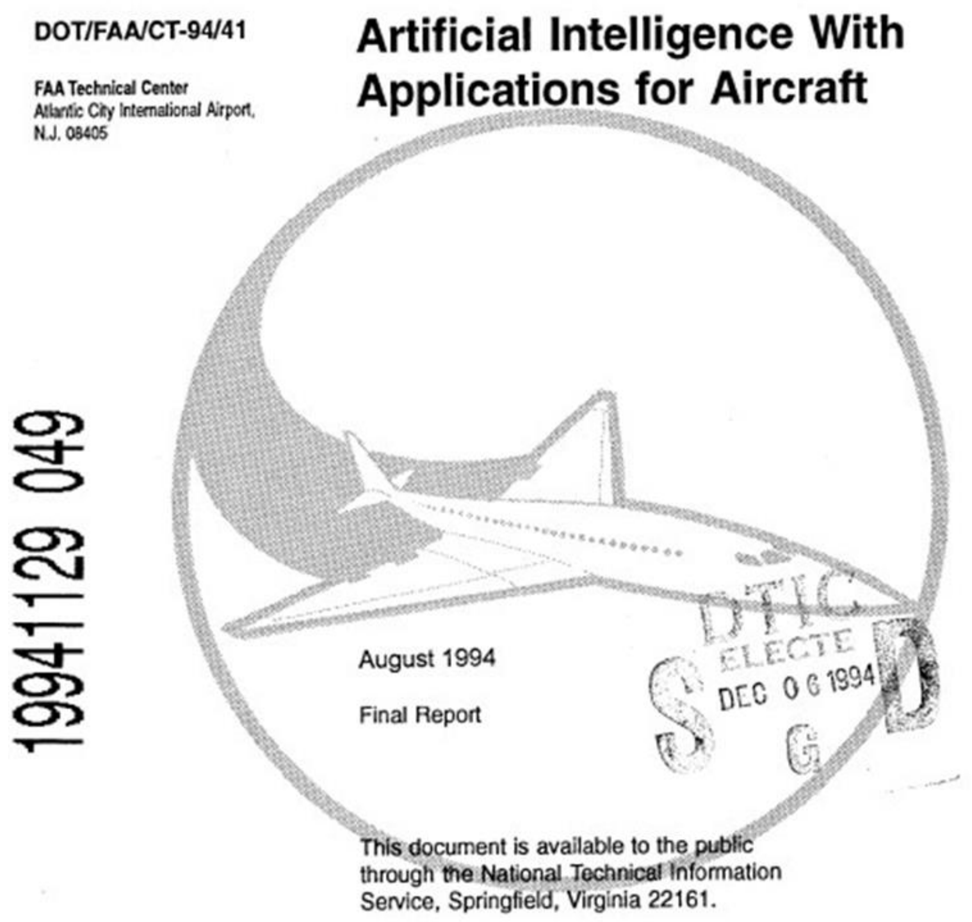
Self-driving Audis A8

https://youtu.be/aw2s-o-sC8c?t=6

Face recognition


Found
Searching…..
You (and 15 other people) say this is
Henrik Andersen (8)
Also known as... (4 alternatives)
Possible matches (opens in a new window)
Record Navigation Wizard
Google
Personalization Engine
Google

Aviation


DOT/FAA/CT-94/41

FAA Technical Center
Atlantic City International Airport,
N.J. 08405

Artificial Intelligence With Applications for Aircraft

19941129 049

August 1994

Final Report

DTIC ELECTE DEC 06 1994 S G D

This document is available to the public through the National Technical Information Service, Springfield, Virginia 22161.

- Optimizing the use of airspace.
- Reducing the cost of flying.
- Meeting Air Traffic Control(ATC) requirements.
- Aiding the decision making process of the flight crew.
- Aiding maintenance activity.
- Assisting data management

Fuzzy Logic in Automotive Engineering

- Antilock Braking System (ABS) -Nissan and Mitsubishi.
- Engine Control-Nok and Nissan.
- Automatic transmission systems- Nissan, Honda, GM.
- Cruise control – Peugeot, Citroën.

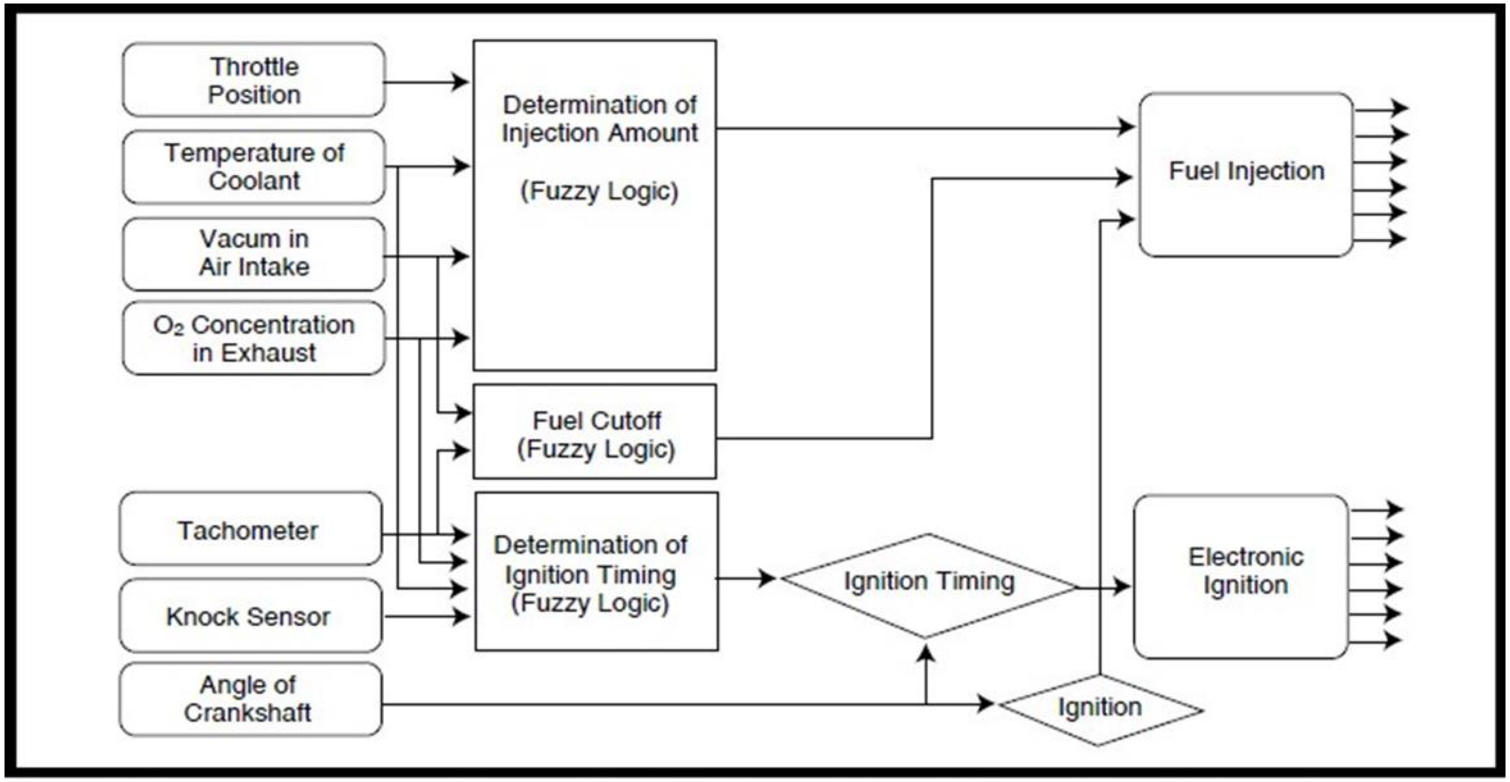
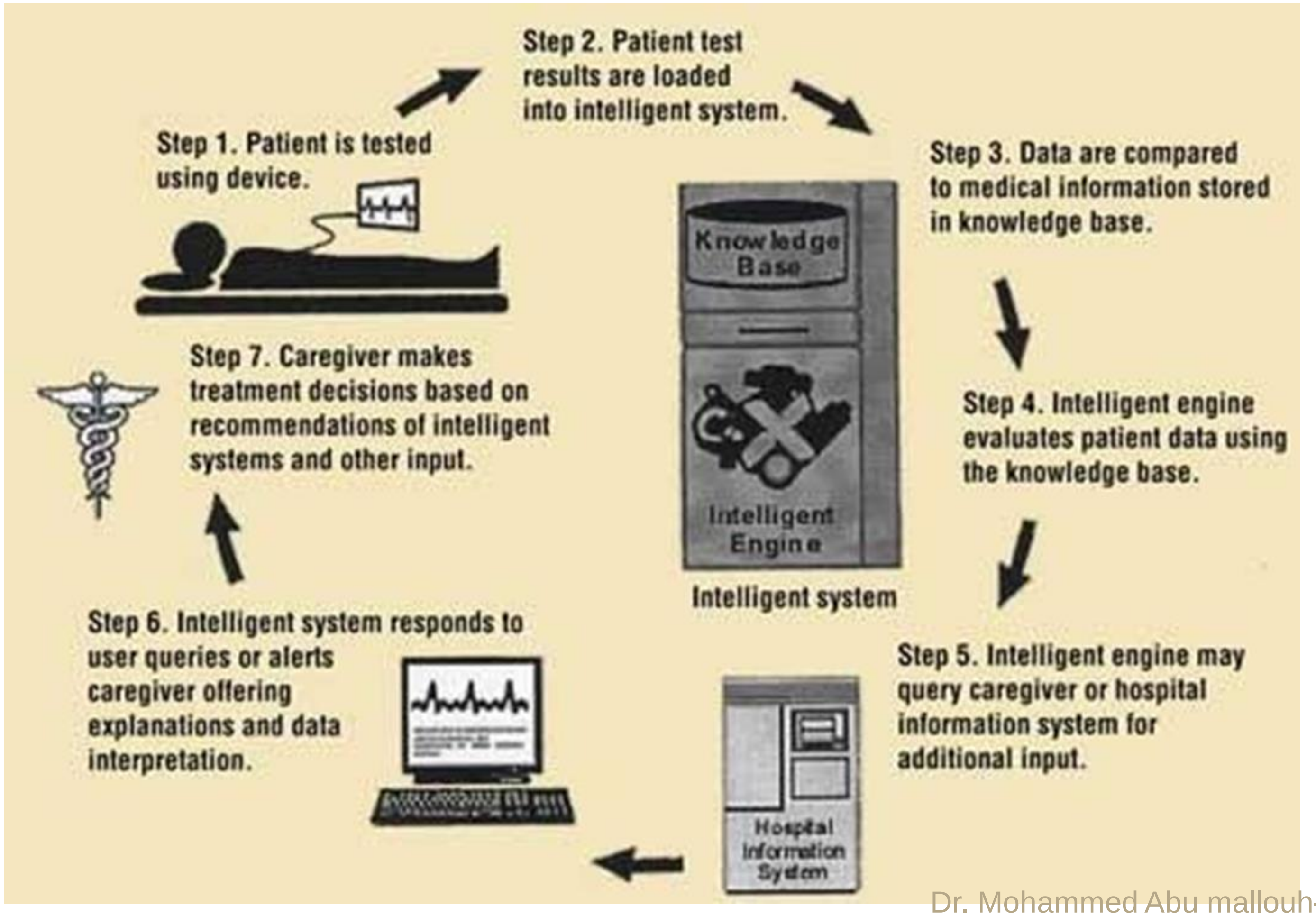


Figure 2—*As you can see, the engine controller of NOK Corporation contains three fuzzy-logic modules.*

AI In Medicine

•MYCIN: early expert system that used artificial intelligence to identify bacteria causing severe infections.


Step 1. Patient is tested using device.
Step 2. Patient test results are loaded into intelligent system.
Step 3. Data are compared to medical information stored in knowledge base.
Knowledge Base
Intelligent Engine
Intelligent system
Step 4. Intelligent engine evaluates patient data using the knowledge base.
Step 5. Intelligent engine may query caregiver or hospital information system for additional input.
Hospital Information System
Step 6. Intelligent system responds to user queries or alerts caregiver offering explanations and data interpretation.
Step 7. Caregiver makes treatment decisions based on recommendations of intelligent systems and other input.

AI In Manufacturing

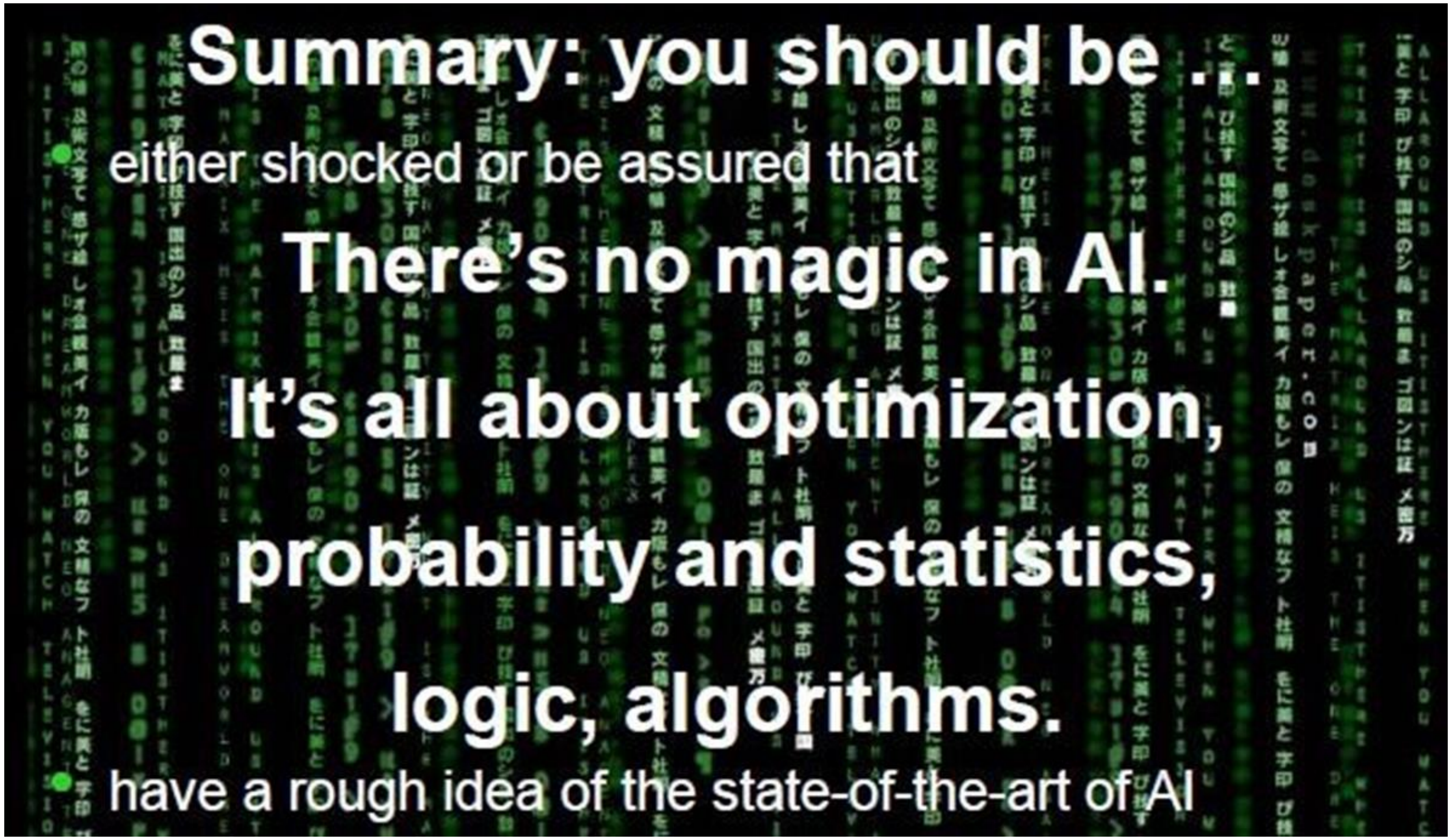
https://youtu.be/JFrk3DHotf8?t=5

AI in Robotics: Atlas robot

Atlas is a humanoid robot primarily developed by the American robotics company Boston Dynamics, with funding and oversight from the U.S. Defense Advanced Research Projects Agency (DARPA).

https://youtu.be/rVlhMGQgDkY?t=9

Contemporary Issue In AI



Summary: you should be ….

- either shocked or be assured that

There's no magic in AI.

It's all about optimization,

probability and statistics,

logic, algorithms.

- have a rough idea of the state-of-the-art of AI

References

1. Introduction to Artificial Intelligence, CS 271, Instructor: Professor Padhraic Smyth
2. http://www.ibm.com/smarterplanet/us/en/ibmwatson/what-is-watson.html
3. https://www.youtube.com/watch?v=qO1i7-Qx00k
4. Introduction to Artificial Intelligence, CS540-2, Bryan R. Gibson
5. DR. A. F. ADEKOYA, Department of Computer Science, University of Agriculture, Abeokuta, Nigeria. www.unaab.edu.ng
6. https://courses.csail.mit.edu/6.825/
7. Negnevitsky, Pearson Education, 2011
8. http://www.ncl.ac.uk/eece/staff/profile/shady.gadoue

Artificial neural networks: Supervised learning

- Machine learning involves adaptive mechanisms that enable computers to learn from experience, learn by example and learn by analogy.

- Learning capabilities can improve the performance of an intelligent system over time. The most popular approaches to machine learning are artificial neural networks and genetic algorithms.

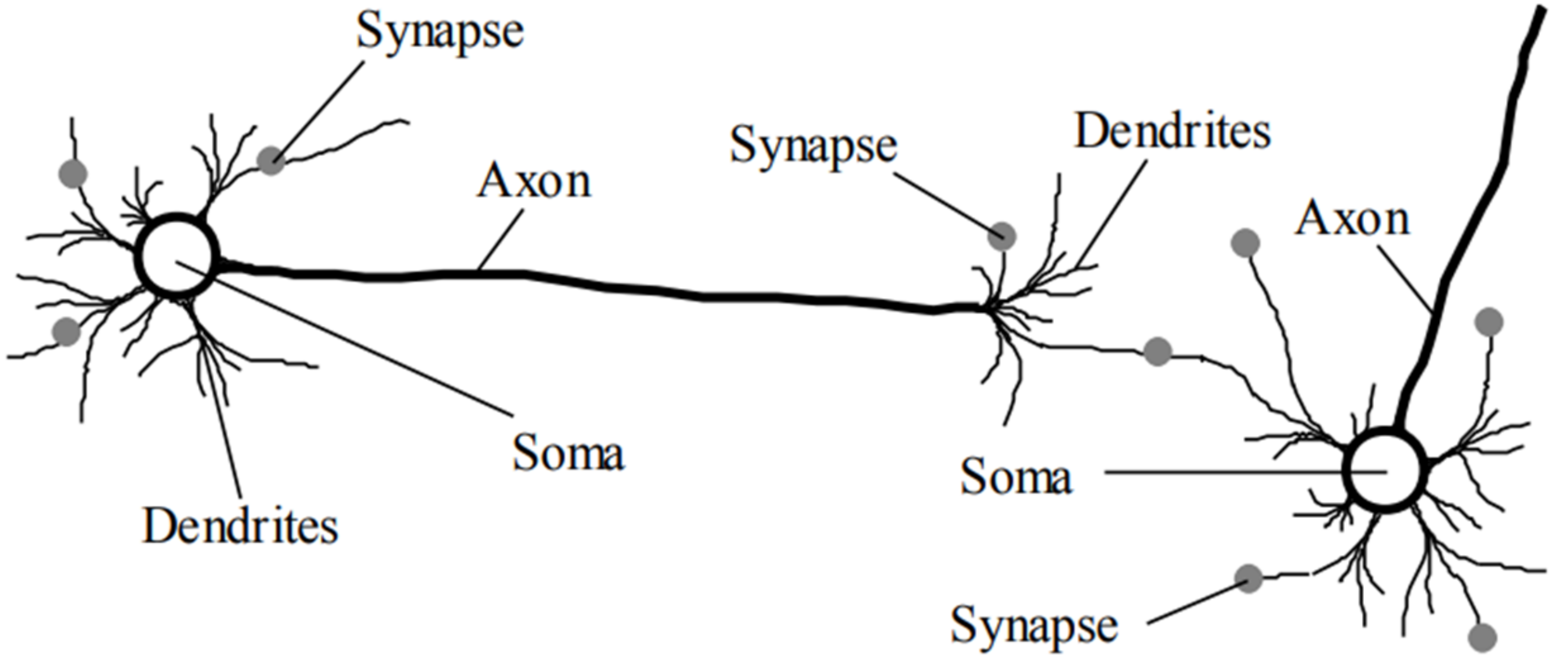
How the brain works

- A neural network can be defined as a model of reasoning based on the human brain.
- The brain consists of a densely interconnected set of nerve cells, or basic information-processing units, called neurons.
- The human brain incorporates nearly 10 billion neurons and 60 trillion connections, synapses, between them.

How the brain works

- By using multiple neurons simultaneously, the brain can perform its functions much faster than the fastest computers in existence.
- Each neuron has a very simple structure, but an army of such elements constitutes a tremendous processing power.
- A neuron consists of a cell body, soma, a number of fibers called dendrites, and a single long fiber called the axon.

Biological neural network



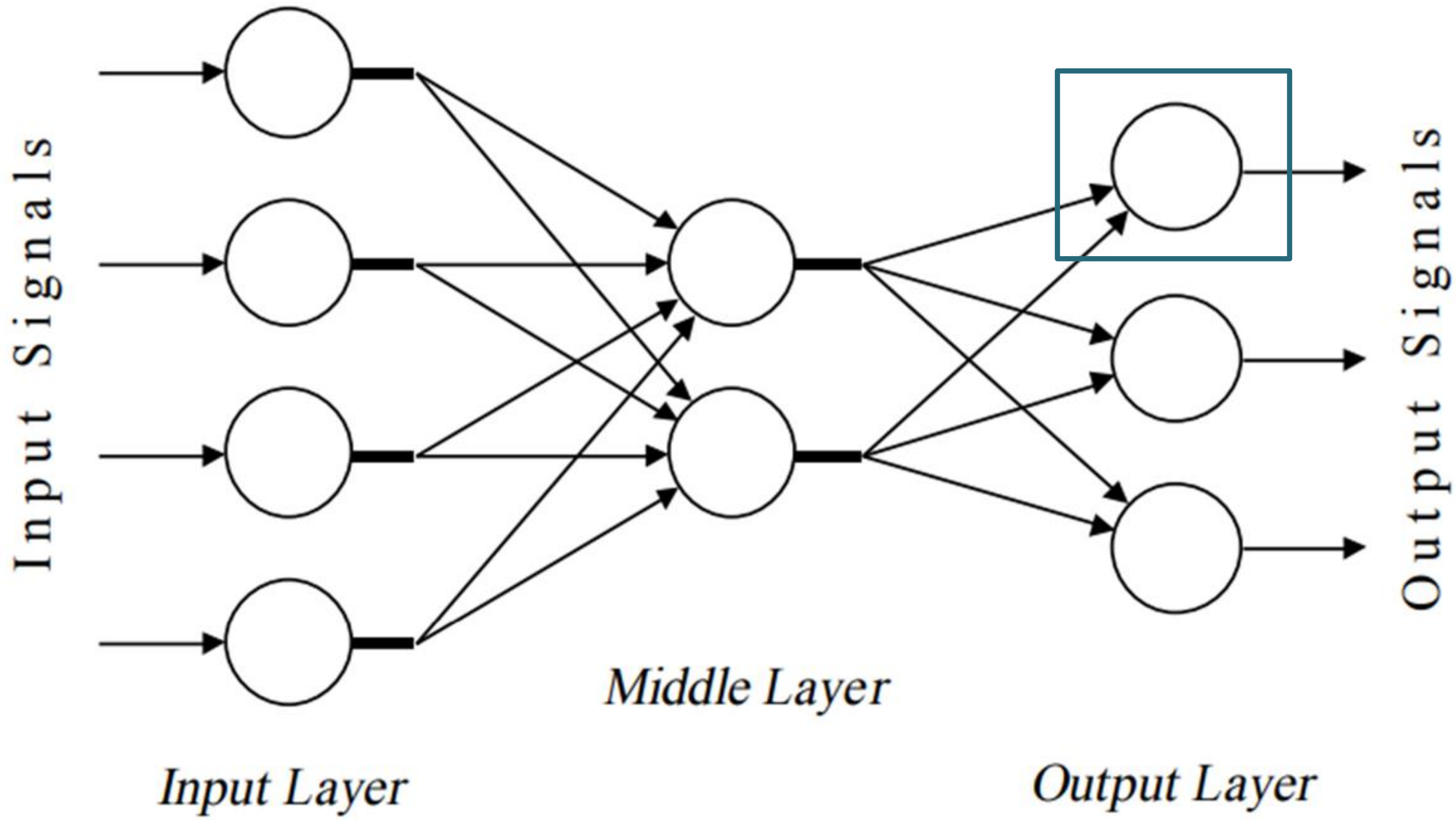
Biological neural network

- Our brain can be considered as a highly complex, non-linear and parallel information-processing system.
- Information is stored and processed in a neural network simultaneously throughout the whole network, rather than at specific locations. In other words, in neural networks, both data and its processing are global rather than local.
- Learning is a fundamental and essential characteristic of biological neural networks. The ease with which they can learn led to attempts to emulate a biological neural network in a computer.

Artificial neural network

- An artificial neural network consists of a number of very simple processors, also called neurons, which are analogous to the biological neurons in the brain.
- The neurons are connected by weighted links passing signals from one neuron to another.
- The output signal is transmitted through the neuron’s outgoing connection. The outgoing connection splits into a number of branches that transmit the same signal. The outgoing branches terminate at the incoming connections of other neurons in the network.

Architecture artificial neural network

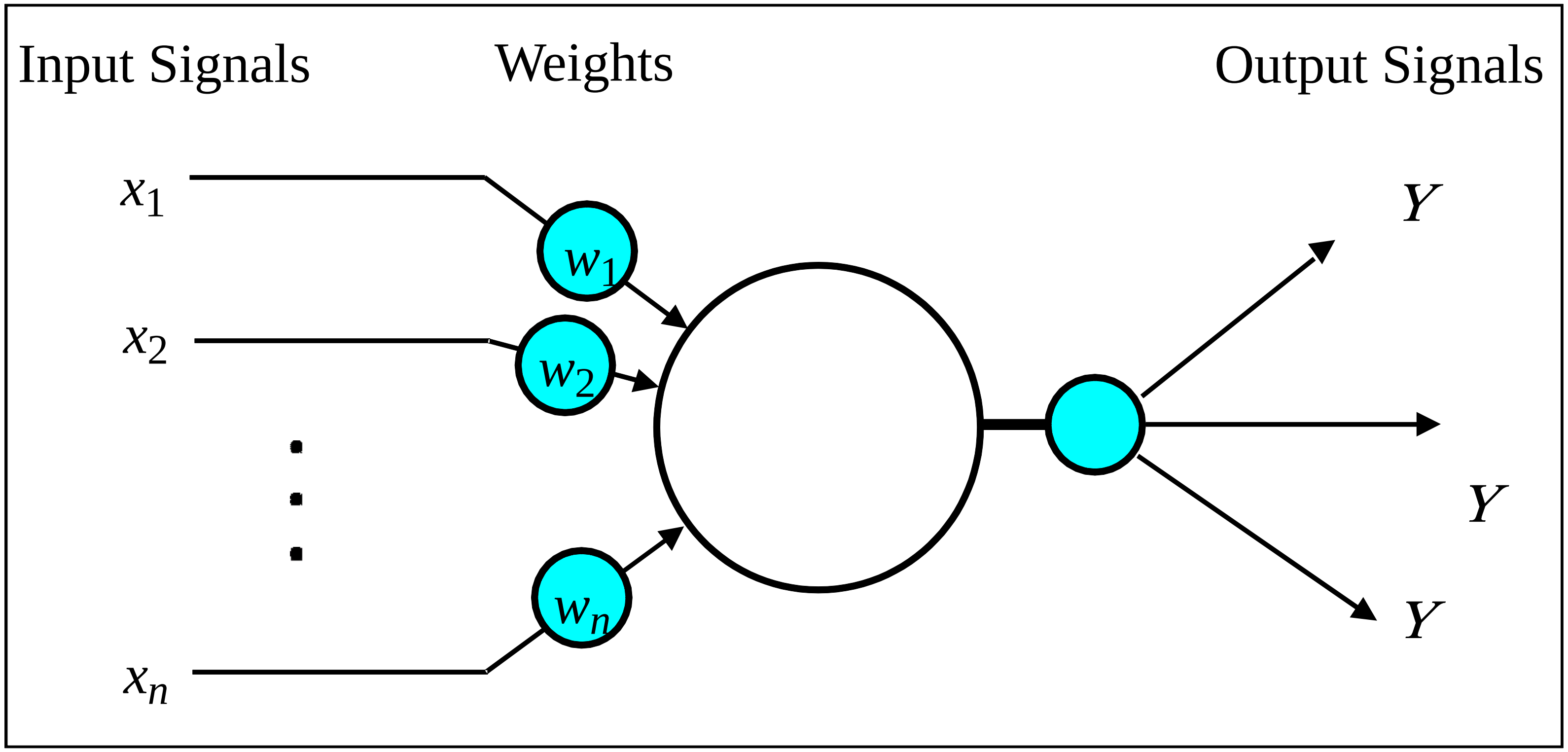


Analogy between biological and artificial neural networks

Biological Neural Network	***Artificial Neural Network***
• Soma	• Neuron
• Dendrite	• Input
• Axon	• Output
• Synapse	• Weight

The neuron as a simple computing element

Diagram of a neuron



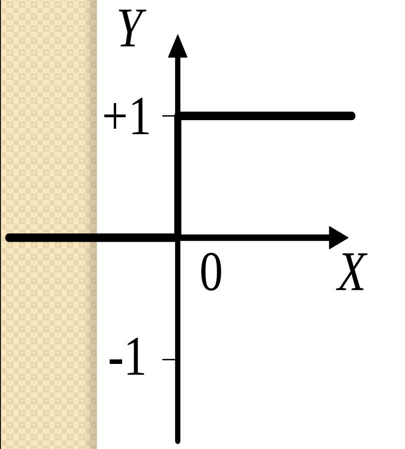
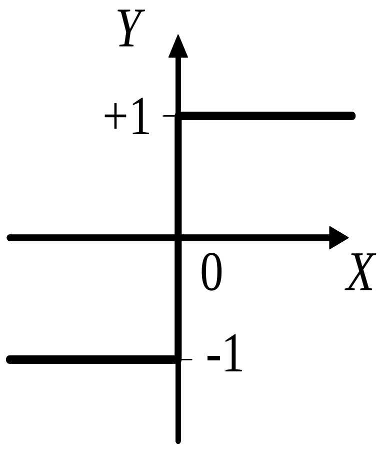
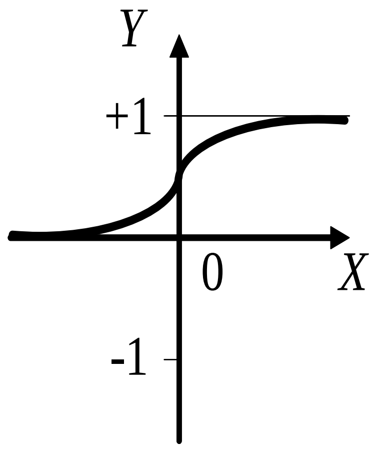
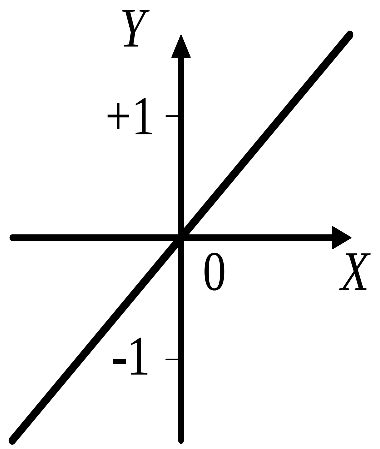
- The neuron computes the weighted sum of the input (a or X) signals and compares the result with a threshold value, θ.

- If the net input is less than the threshold, the neuron output is 0, But if the net input is greater than or equal to the threshold, the neuron becomes activated and its output attains a value +1.

- This type of activation function is called a step function.

$$a = X = \sum_{i=1}^{n} x_i w_i \qquad Y = \begin{cases} +1, & \text{if } X \geq \theta \\ 0, & \text{if } X < \theta \end{cases}$$

More activation functions

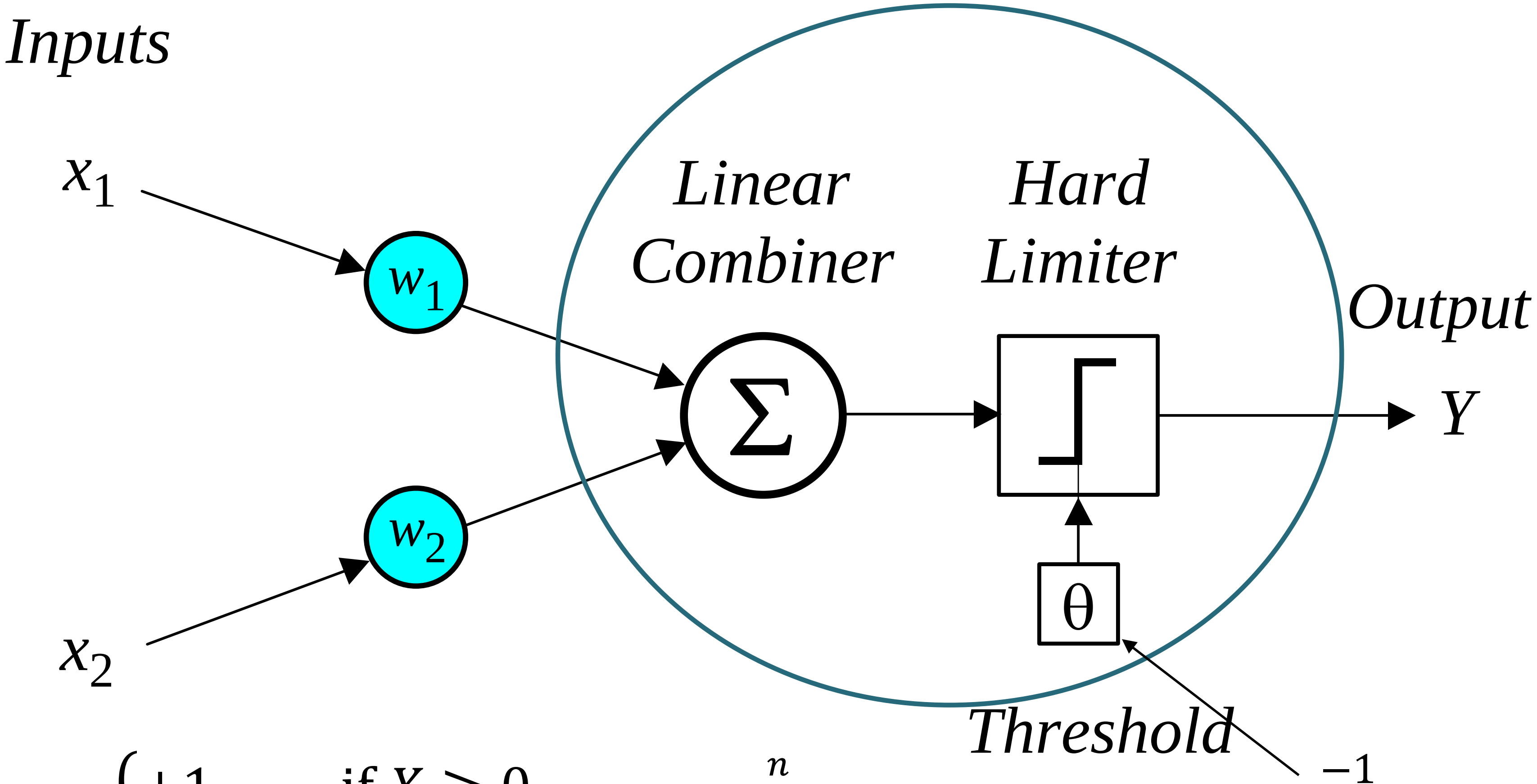
Step function	*Sign function*	*Sigmoid function*	*Linear function*
			
$Y^{step} = \begin{cases} 1, \text{ if } X \geq 0 \\ 0, \text{ if } X < 0 \end{cases}$	$Y^{sign} = \begin{cases} +1, \text{ if } X \geq 0 \\ -1, \text{ if } X < 0 \end{cases}$	$Y^{sigmoid} = \dfrac{1}{1+e^{-X}}$	$Y^{linear} = X$

Can a single neuron learn a task?

- In 1958, Frank Rosenblatt introduced a training algorithm that provided the first procedure for training a simple ANN: a perceptron.

- The perceptron is the simplest form of a neural network. It consists of a single neuron with adjustable synaptic weights and a hard limiter (sign function).

Single-layer two-input perceptron



$$Y = \begin{cases} +1, & \text{if } X \geq \theta \\ -1, & \text{if } X < \theta \end{cases} \qquad X = \sum_{i=1}^{n} x_i w_i$$

The Perceptron

- The weighted sum of the inputs is applied to the hard limiter, which produces an output equal to +1 if its input is positive and -1 if it is negative.

- The aim of the perceptron is to classify inputs, x1, x2, . . ., xn, into one of two classes, say A1 and A2.

- In the case of an elementary perceptron, the n- dimensional space is divided by a hyper plane into two decision regions. The hyper plane is defined by the linearly separable function:

$$\sum_{i=1}^{n} x_i w_i - \theta = 0 \quad Y = \begin{cases} +1, & \text{if } X \geq \theta \\ -1, & \text{if } X < \theta \end{cases} \quad X = \sum_{i=1}^{n} x_i w_i$$

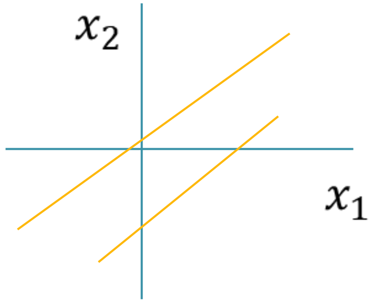
Linear Separability

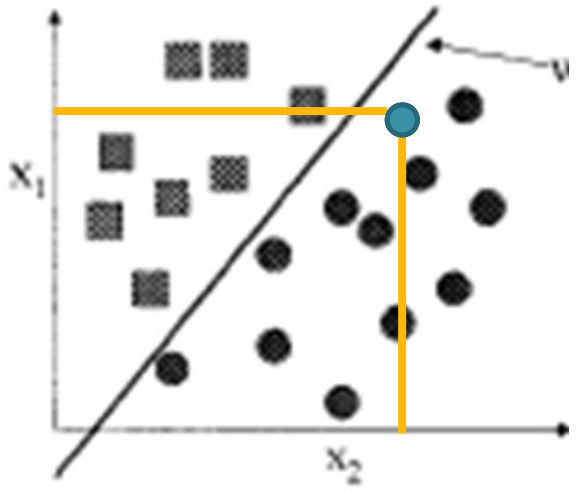
Consider a perceptron processor that only has 2 input . So (note: $\theta = w_0$)

When plotted in the space of the values of X1 and X2 this is just the equation of a straight line .

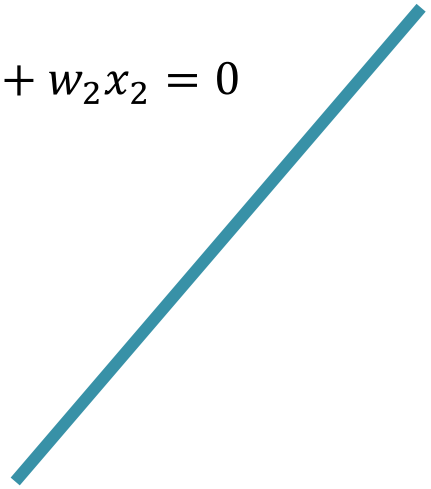
$$w_0(-1) + w_1 x_1 + w_2 x_2 = 0$$

$$(x_2 = c x_1 + b, where\ c = -w_1/w_2\ and\ b = w_0/w_2)$$



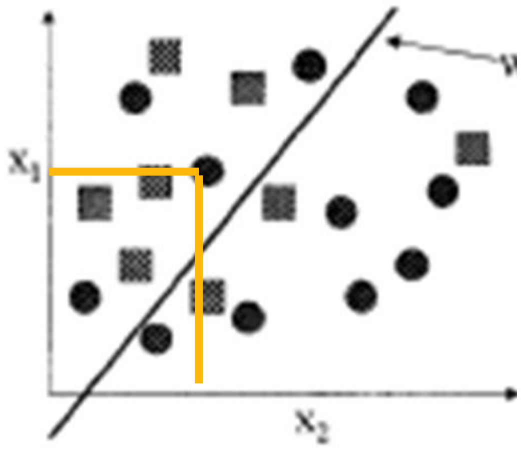


$$-w_0 + w_1 x_1 + w_2 x_2 = 0$$



- For inputs x1, X2 that fall on one side line , the activation (a)will be >0,and thus the output (y)will be 1. On the other side (and right on the line)the output will be -1.

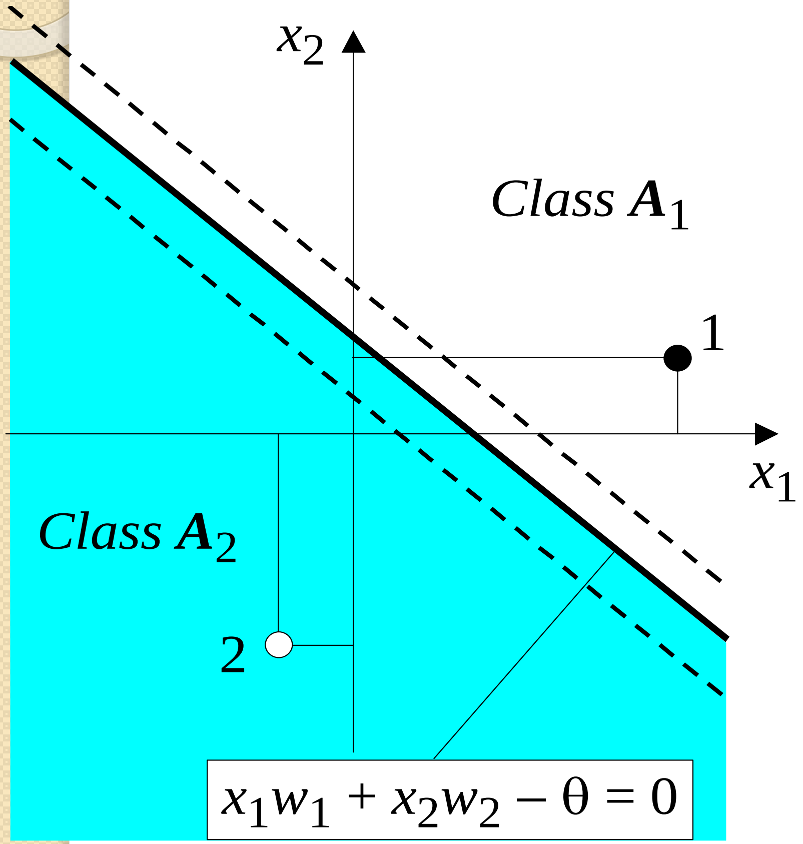
- So each set of values for w defines a straight line decision boundary and any possible line can be represented by some value of w .



$$-w_0 + w_1 x_1 + w_2 x_2 = 0$$

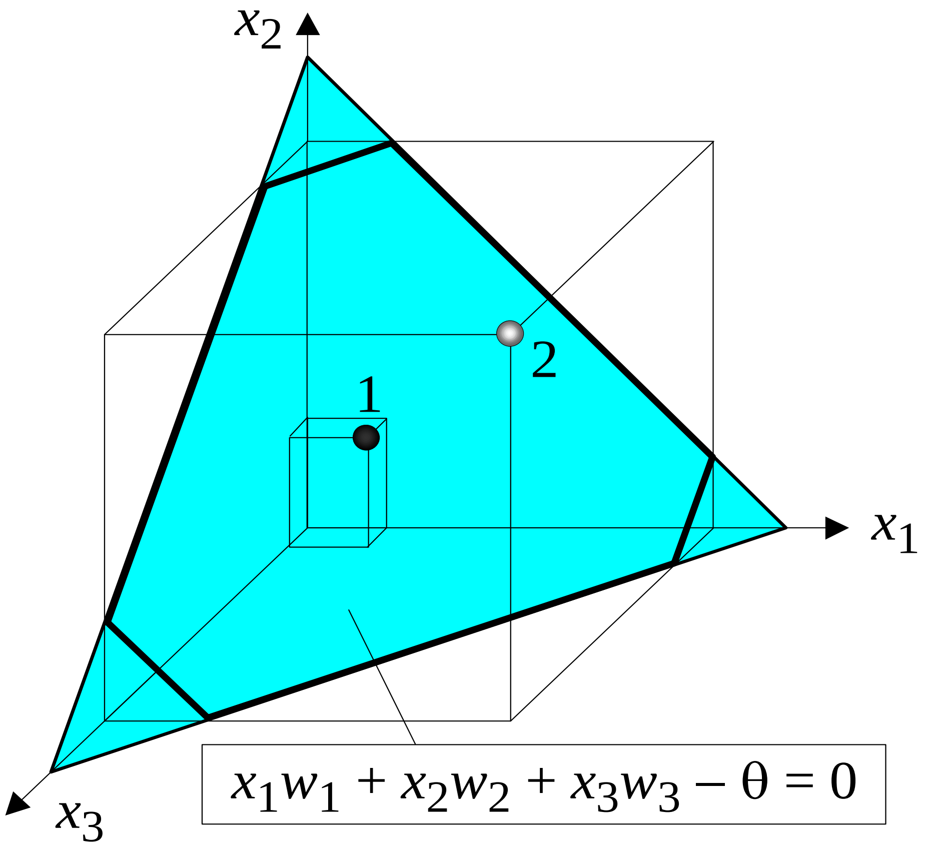
- So obviously , if we had a problem for which the examples did not separate nicely, we could not solve the problem completely using a perceptron processor.
- If the set of input/output pairs can be separated with a straight line , then it has the property of being linearly separable.
- (For input vectors of higher dimension, the boundary is a higher dimension hyper plane.Eg., a plane in a 3D input space)

Linear Separability in the perceptron's



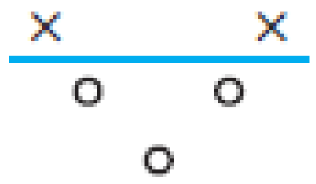
x_2
x_1
Class $\boldsymbol{A}_1$
1
Class $\boldsymbol{A}_2$
2
$x_1w_1 + x_2w_2 - \theta = 0$

(*a*) Two-input perceptron.

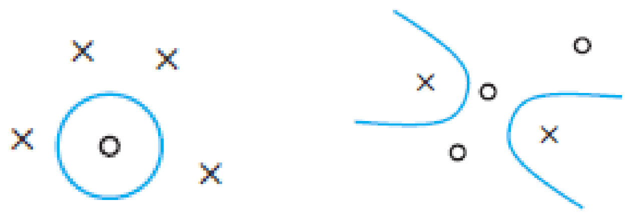


(*b*) Three-input perceptron.

Linear vs. nonlinear separation



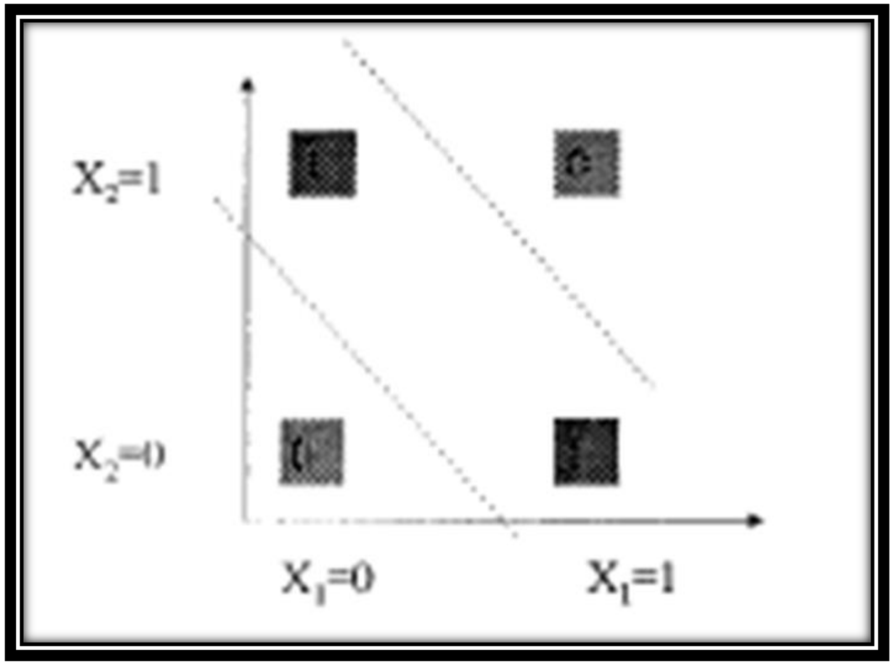
Linear Separation



Nonlinear Separation

XOR is linearly separable ?

•Consider the XOR Problem : To form a decision boundary for the XOR function.



•This task is not linearly separable. It would require 2 decision boundaries to separate the two classes.

Can perceptrons be used for tasks that are not linearly separable ?

With modifications, yes.

•If the task is not linearly separable, we accept that the result will not be perfect, but we wish to minimize the number of errors.

•The number of errors is defined by the values of the weights.

In general, networks of perceptron-like processors can solve most non-linearly separable tasks by using more than one layer of processors. Rosenblatt(and others)realized this in the 1960’s, but did not have a learning rule that would work effectively with more than one level(it wasn’t invented until the mid 1970’s).

How does the perceptron learn its classification tasks?

This is done by making small adjustments in the weights to reduce the difference between the actual and desired outputs of the perceptron. The initial weights are randomly assigned, usually in the range [-0.5, 0.5], and then updated to obtain the output consistent with the training examples.

- If at iteration p, the actual output is Y(p) and the desired output is Yd (p), then the error is given by:

$$e(p) = Y_d(p) - Y(p) \qquad \text{where } p = 1, 2, 3, \ldots$$

- Iteration p here refers to the pth training example presented to the perceptron.
- If the error, e(p), is positive, we need to increase perceptron output Y(p), but if it is negative, we need to decrease Y(p).

The perceptron learning rule

- A learning rule is a strategy by which example input/output pairs can be used to incrementally change the weights in away that gradually improves the performance of the network.

$$w_i(p+1) = w_i(p) + \alpha \cdot x_i(p) \cdot e(p)$$

where *p* = 1, 2, 3, . . .
α is the **learning rate**, a positive constant less than unity.

If the activation function was a step function. The perceptron learning rule involves the example input x, the computed output y, and the desired output d .

$$If\ y = 1, and\ d = 0\colon w_{i+1} = w_i - \alpha x_i (w_i = 1, \ldots., n)$$
$$If\ y = 0, and\ d = 1\colon w_{i+1} = w_i + \alpha x_i (w_i = 1, \ldots., n)$$

Where α is a small learning parameter

- Whenever the (y=1, and d=0)error occurs, it is because the activation a was too large, and decrementing the weights will reduce the activation for the same input. Similarly, the (y=0,and d=1)will result in appropriate increasing of activation for the same input .

Perceptron’s training algorithm

Step 1: Initialisation

Set initial weights w_1, w_2,…, w_n and threshold θ to random numbers in the range [–0.5, 0.5].

Perceptron's training algorithm (continued)

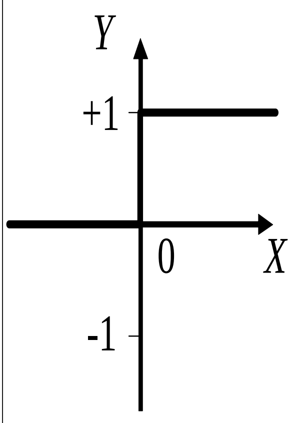
Step 2: Activation

Activate the perceptron by applying inputs $x_1(p)$, $x_2(p)$,…, $x_n(p)$ and desired output $Y_d(p)$. Calculate the actual output at iteration $p = 1$

$$Y(p) = step\left[\sum_{i=1}^{n} x_i(p)\, w_i(p) - \theta\right]$$

where *n* is the number of the perceptron inputs, and *step* is a step activation function.

Step function



$$Y^{step} = \begin{cases} 1, \text{ if } X \geq 0 \\ 0, \text{ if } X < 0 \end{cases}$$

Perceptron’s training algorithm

If the error, *e*(*p*), is positive, we need to increase perceptron output *Y*(*p*), but if it is negative, we need to decrease *Y*(*p*).

<u>Step 3</u>: Weight training

Update the weights of the perceptron

$$w_i(p+1) = w_i(p) + \Delta w_i(p)$$

where $\Delta w_i(p)$ is the weight correction at iteration *p*.

$$\Delta w_i(p) = \alpha \cdot x_i(p) \cdot e(p)$$

The weight correction is computed by the **delta**

<u>Step 4</u>: Iteration

Increase iteration *p* by one, go back to *Step 2* and repeat the process until convergence.

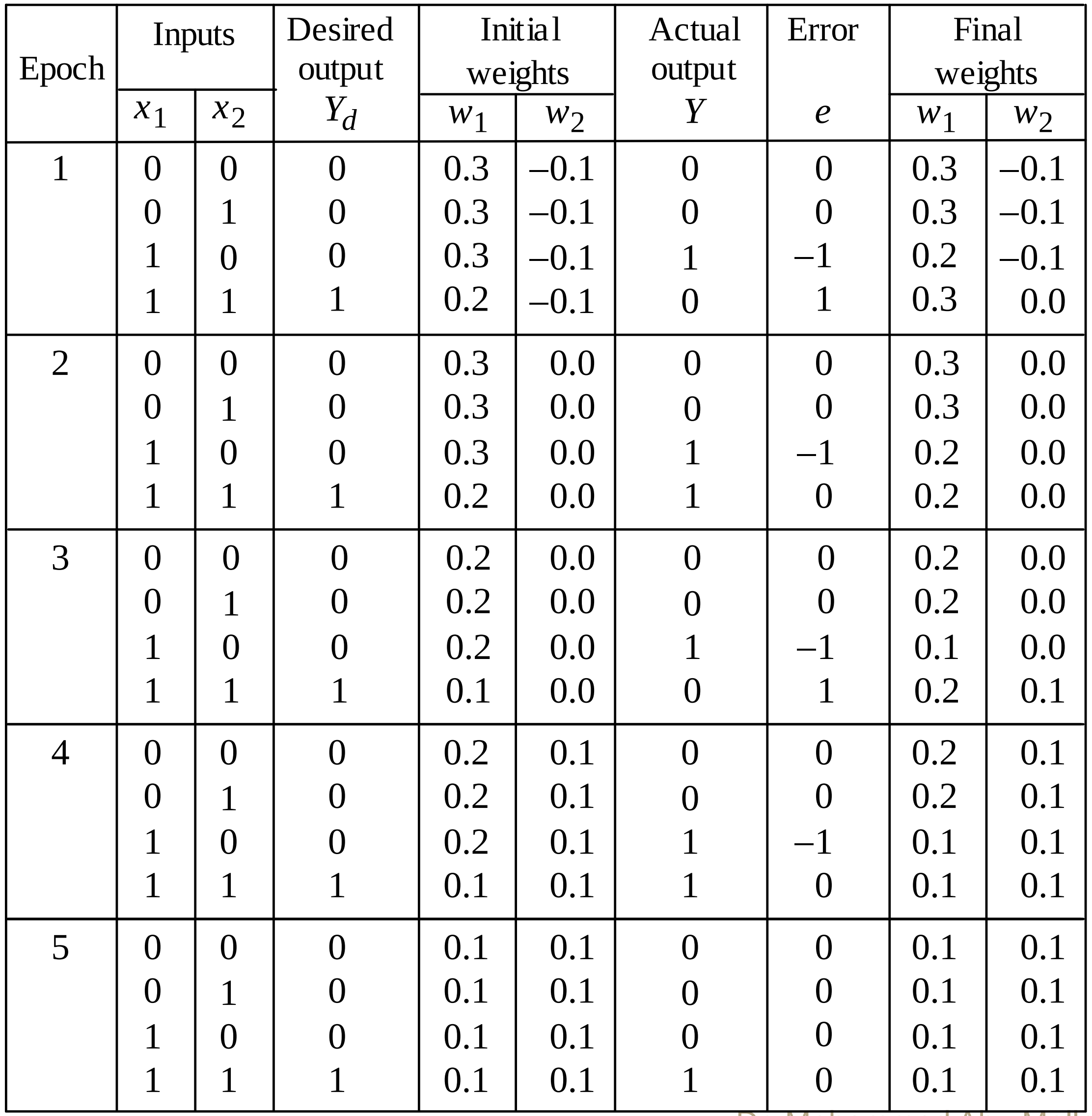
Example perceptron learning: logical operation *AND*

Epoch	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	–0.1				
	0	1	0						
	1	0	0						
	1	1	1						
2	0	0	0						
	0	1	0						
	1	0	0						
	1	1	1						
3	0	0	0						
	0	1	0						
	1	0	0						
	1	1	1						
4	0	0	0						
	0	1	0						
	1	0	0						
	1	1	1						
5	0	0	0						
	0	1	0						
	1	0	0						
	1	1	1						

$$Y = \begin{cases} +1, & \text{if } X \geq \theta \\ 0, & \text{if } X < \theta \end{cases}$$

Threshold: θ = 0.2; learning rate: α = 0.1

Example of perceptron learning: the logical operation *AND*



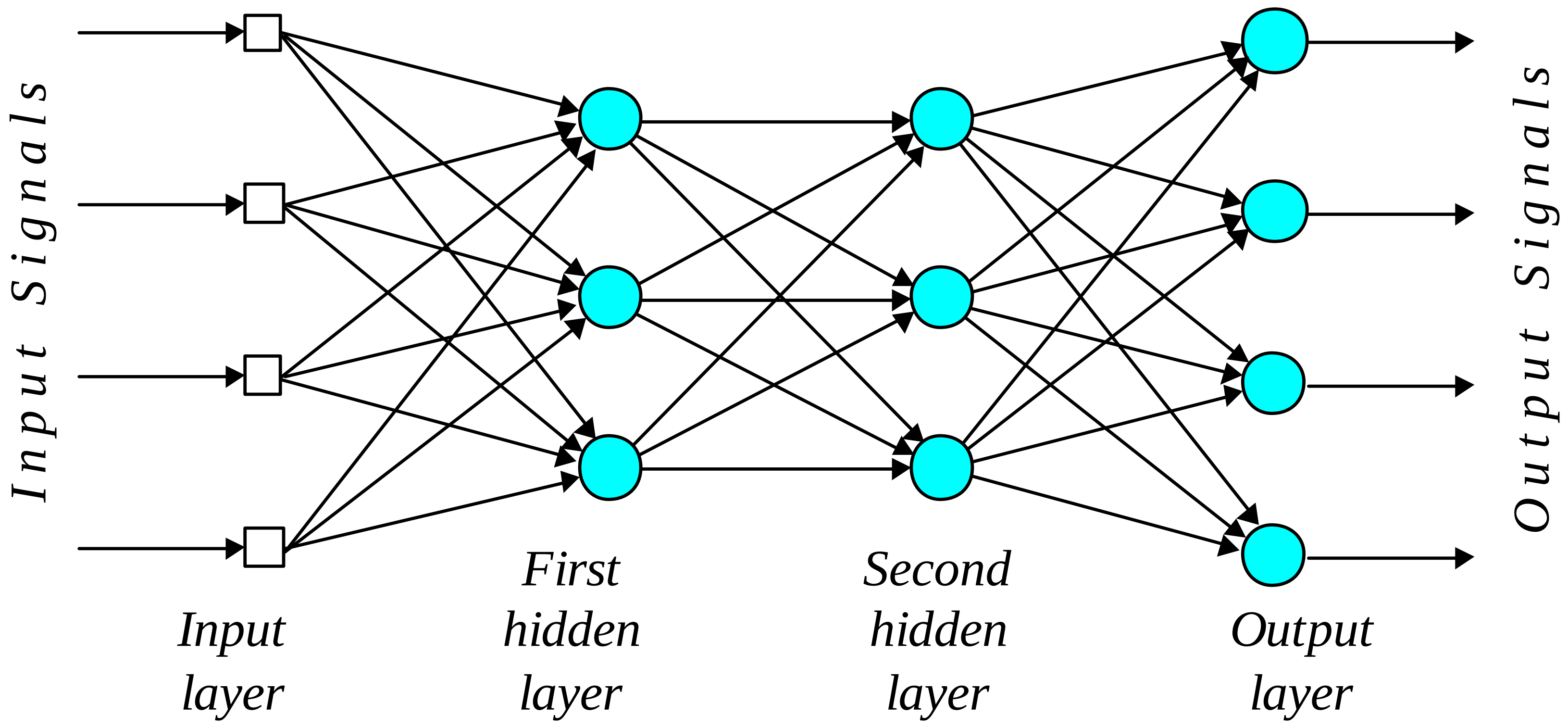
Epoch	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	–0.1	0	0	0.3	–0.1
	0	1	0	0.3	–0.1	0	0	0.3	–0.1
	1	0	0	0.3	–0.1	1	–1	0.2	–0.1
	1	1	1	0.2	–0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	–1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
3	0	0	0	0.2	0.0	0	0	0.2	0.0
	0	1	0	0.2	0.0	0	0	0.2	0.0
	1	0	0	0.2	0.0	1	–1	0.1	0.0
	1	1	1	0.1	0.0	0	1	0.2	0.1
4	0	0	0	0.2	0.1	0	0	0.2	0.1
	0	1	0	0.2	0.1	0	0	0.2	0.1
	1	0	0	0.2	0.1	1	–1	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1
5	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0	0	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1

Threshold: $\theta = 0.2$; learning rate: $\alpha = 0.1$

Multilayer neural networks

- A multilayer perceptron is a feed forward neural network with one or more hidden layers.
- The network consists of an **input layer** of source neurons, at least one middle or **hidden layer** of computational neurons, and an **output layer** of computational neurons.
- The input signals are propagated in a forward direction on a layer-by-layer basis.

Multilayer perceptron with two hidden layers



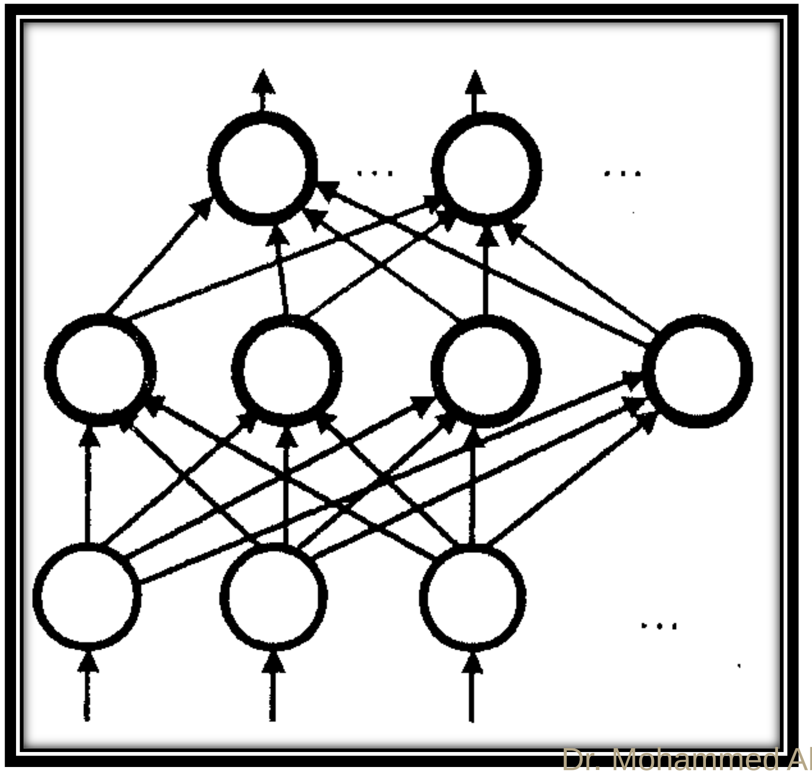
What does the middle layer hide?

- A hidden layer "hides" its desired output. Neurons in the hidden layer cannot be observed through the input/output behaviour of the network. There is no obvious way to know what the desired output of the hidden layer should be.

- Commercial ANNs incorporate three and sometimes four layers, including one or two hidden layers. Each layer can contain from 10 to 1000 neurons. Experimental neural networks may have five or even six layers, including three or four hidden layers, and utilise millions of neurons.

BACKPROPAGATION NETWORKS

To overcome the limitations of perceptrons, networks require more than one processing layer . If we try to extend the error-correction learning to more layers, we encounter the credit-assignment problem: How to determine which nodes are responsible for an outcome.
We consider a back propagation network with 2 processing layers, though they could have more. The layers are output, hidden, and input (which is just fan-in).

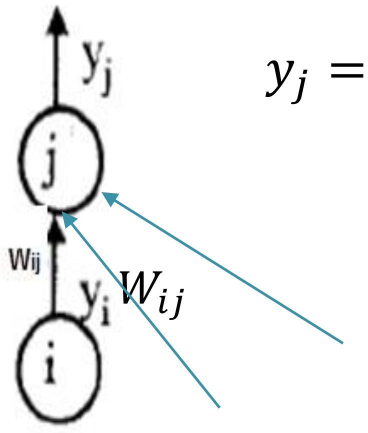


Each processing node uses

- The activation of node j is the sum of each of its inputs x_0, $x_1, \ldots, x_n$ times each of its weights or the corresponding input Wij

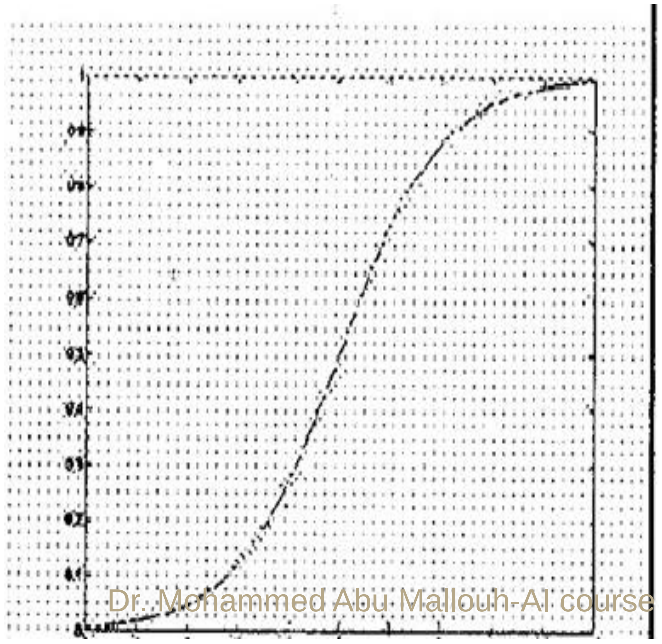
- Sometimes we refer to the inputs as $y_0, y_1, \ldots, y_n$ because they are often outputs of the nodes 1,…,n.
- Each processing node has a bias (for threshold) input x_0=-1.
- Each processing node creates an output signal,

Which is the sigmoidal function.



$$y_j = f(a_j) = \frac{1}{1 + e^{-a_j}}$$

$$a_j = \sum_{i=0}^{n} X_i \, W_{ij}$$



The training data consists of examples of inputs and desired outputs :{(x_1, d_1),…(x_p , d_p)}

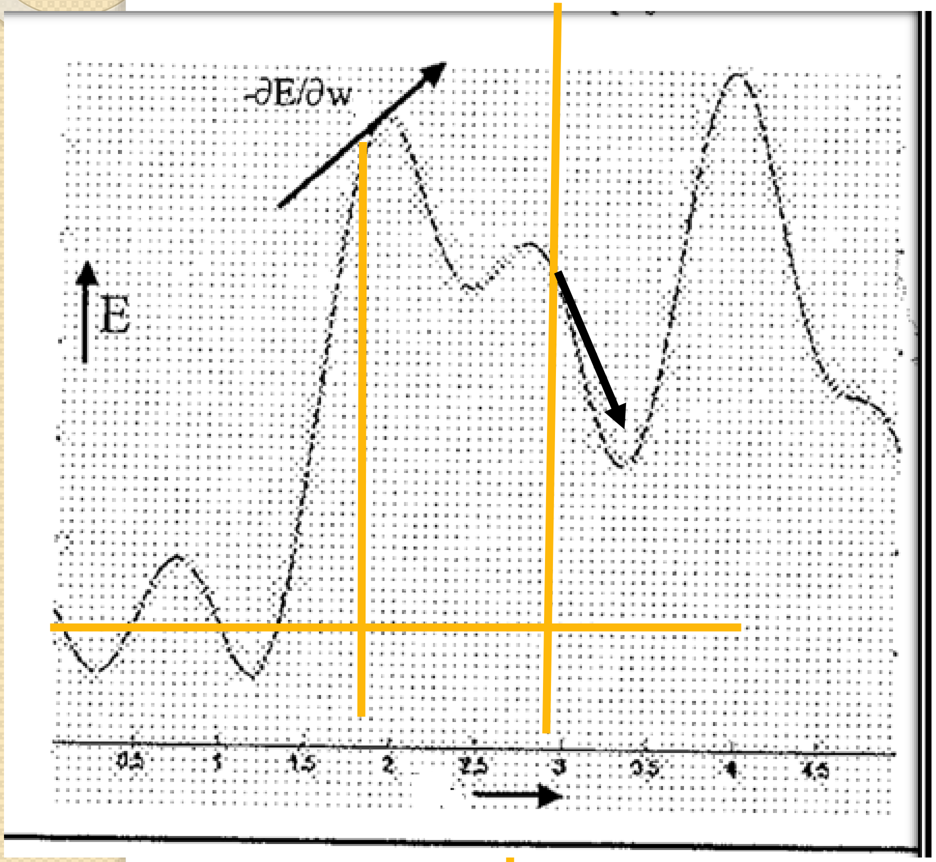
In general , we define a measure of error for any particular trial:

$$error = \sum_{i=1}^{m} e^2(y_k, d_k)$$

Where y_k are the outputs of each of the m output nodes, and d_k are the components of the desired result.

Usually the error function is the sum of square errors , or mean error.

$$e = d - \text{actual (y)}, \qquad E = \frac{\sum e^2}{2}.$$



W_{ij}

Consider the projection of a single weight ij in the error surface.

$\partial E/\partial Wij$ is the direction of steepest ascent, and so $-\partial E/\partial Wij$ is the direction of steepest descent.

We are interested in changing the weight to a value that decreases the error , so $-\partial E/\partial Wij$ tells us in what direction to move the weight value.

$$a = \sum_{i=0}^{n} X_i W_i$$

$$\triangle W_{ij=} - \alpha \frac{\partial E}{\partial w_{ij}}$$

we use gradient descant to determine the direction of change of the weights w

$$-\partial E / \partial W$$

We consider the adjustment to be made to individual weights w_{ij} ,which is the weight to node j from its i th input. Regardless of what layer it is in, we want to make the adjustment:

$$\triangle W_{ij} = -\alpha \frac{\partial E}{\partial w_{ij}}$$

We have:

$(1) E = (1/2)\sum e^2$	Total net work error
$(2) e_j = d_j - y_j$	Individual node error
$(3) y_j = f(a_j)$	Output is function of activation
$(4) a_j = \sum w_{ij} y_i$	Value of node's activation is sum of weight inputs

$$(1) E = \frac{1}{2}\sum e^2, \quad (2) e_j = d_j - y_j, \quad (3) y_j = f(a_j), \quad (4) a_j = \sum w_{ij} y_i$$

We calculate the change that we wish to make to each weight in the **output layer.**

$$\Delta w_{ij} = -\alpha \, \partial E / \partial w_{ij}$$

$$\Delta w_{ij} = -\alpha \, \partial E / \partial e_j . \partial e_j / \partial y_j . \partial y_j / \partial a_j . \partial a_j / \partial w_{ij}$$

$$\Delta w_{ij} = -\alpha \, e_j . (-1) . f'(a_j) . y_i$$

$$\Delta w_{ij} = \alpha \cdot \delta_j^0 \, y_i \quad \textit{where} \; \delta_j^0 = e_j f'(a_j)$$

This looks like the rule that was used for the Perceptron:
=Constant × error × input .

The difference is that δ_j^0, the error at output node j, is the usual error $(d_j - y_j)$

,scaled by a factor of $f'(a_j)$

$$f(x) = 1/(1+e^{-x})$$

$$f'(x) = e^{-x}/(1+e^{-x})^2$$

$$= 1/(1+e^{-x}) - 1/(1+e^{-x})^2$$

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

$$\text{so } \delta_j^0 = (d_j - y_j)y_j(1 - y_j)$$

•But bear in mind that this is only true if f(x) is the sigmoidal function. Other output functions can also be used , so $f'(a_j)$ is the more general form.

❖Note that the output function for Back propagation must be differentiable.

$(1) E = \frac{1}{2}\sum e^2$,   (2)$e_j$=$d_j$-$y_j$,   (3)y_j=f(a_j) ,   (4)a_j=$\sum w_{ij}\, y_i$

Now, let's consider how to calculate $\triangle w_{ij}$ for the **hidden nodes.**

$\triangle w_{ij}$ = - α $\partial E/\partial w_{ij}$

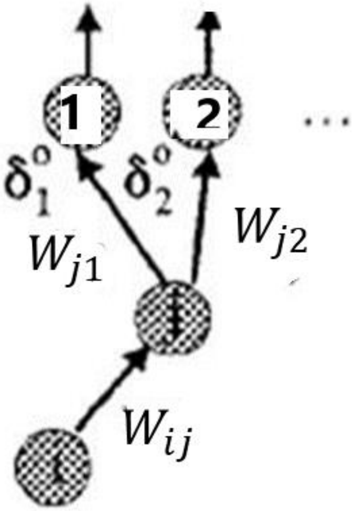
= -α $\partial E/\partial y_j$. $\partial y_j/\partial a_j$. $\partial a_j/\partial w_{ij}$ then using(3) and (4) as before

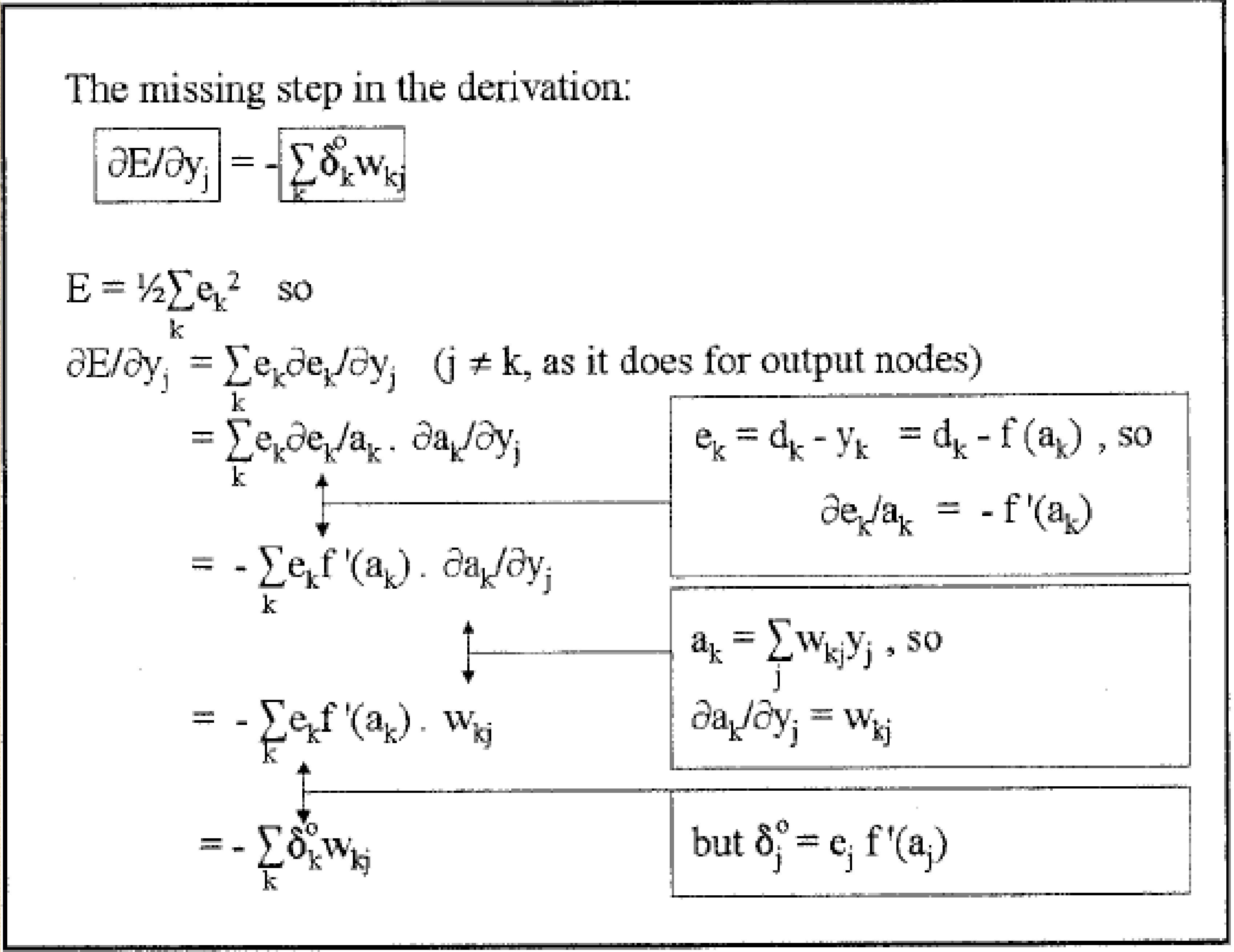
Later we will see why these
Outlined expressions are equal

= - α $\partial E/\partial y_j$. $f'(a_j)$. y_i

= α $\sum(\delta_k^0\, w_{jk})$. $f'(a_j)$. y_i

= α $\delta_j^h\, y_i$ where $\delta_j^h = \sum(\delta_k^0\, w_{jk})$. $f'(a_j)$





The missing step in the derivation:

$$\boxed{\partial E/\partial y_j} = -\boxed{\sum_k \delta_k^o w_{kj}}$$

$E = \frac{1}{2}\sum_k e_k^2$ so

$$\partial E/\partial y_j = \sum_k e_k \partial e_k/\partial y_j$$ (j ≠ k, as it does for output nodes)

$$= \sum_k e_k \partial e_k/a_k \,.\, \partial a_k/\partial y_j$$

$e_k = d_k - y_k = d_k - f(a_k)$, so

$\partial e_k/a_k = -f'(a_k)$

$$= -\sum_k e_k f'(a_k) \,.\, \partial a_k/\partial y_j$$

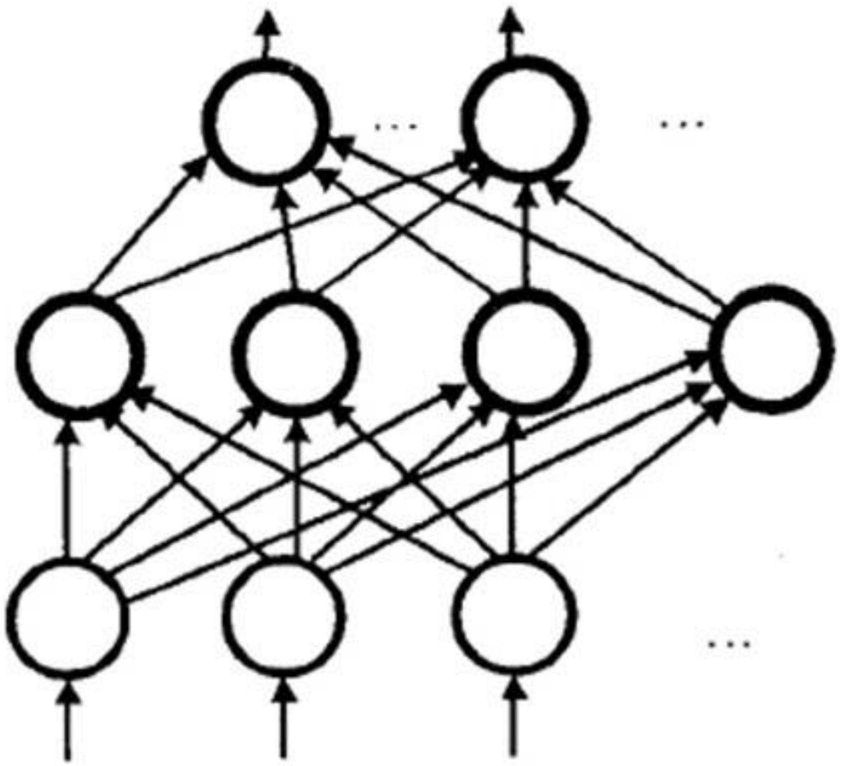
$a_k = \sum_j w_{kj} y_j$, so

$\partial a_k/\partial y_j = w_{kj}$

$$= -\sum_k e_k f'(a_k) \,.\, w_{kj}$$

but $\delta_j^o = e_j\, f'(a_j)$

$$= -\sum_k \delta_k^o w_{kj}$$



1. Provide an example input **x**, allow the network to compute in feedforward mode, and produce output **y**.

2. Calculate the error at the output nodes by copmparing the desired output **d** to the actual output **y**. $\delta_j^o = e_j\, f'(a_j)$

3. Calculate the error at the hidden nodes based on the output node errors. $\delta_j^h = \sum \delta_k^o w_{jk} \cdot f'(a_j)$ (error is propagated back to hidden layer)

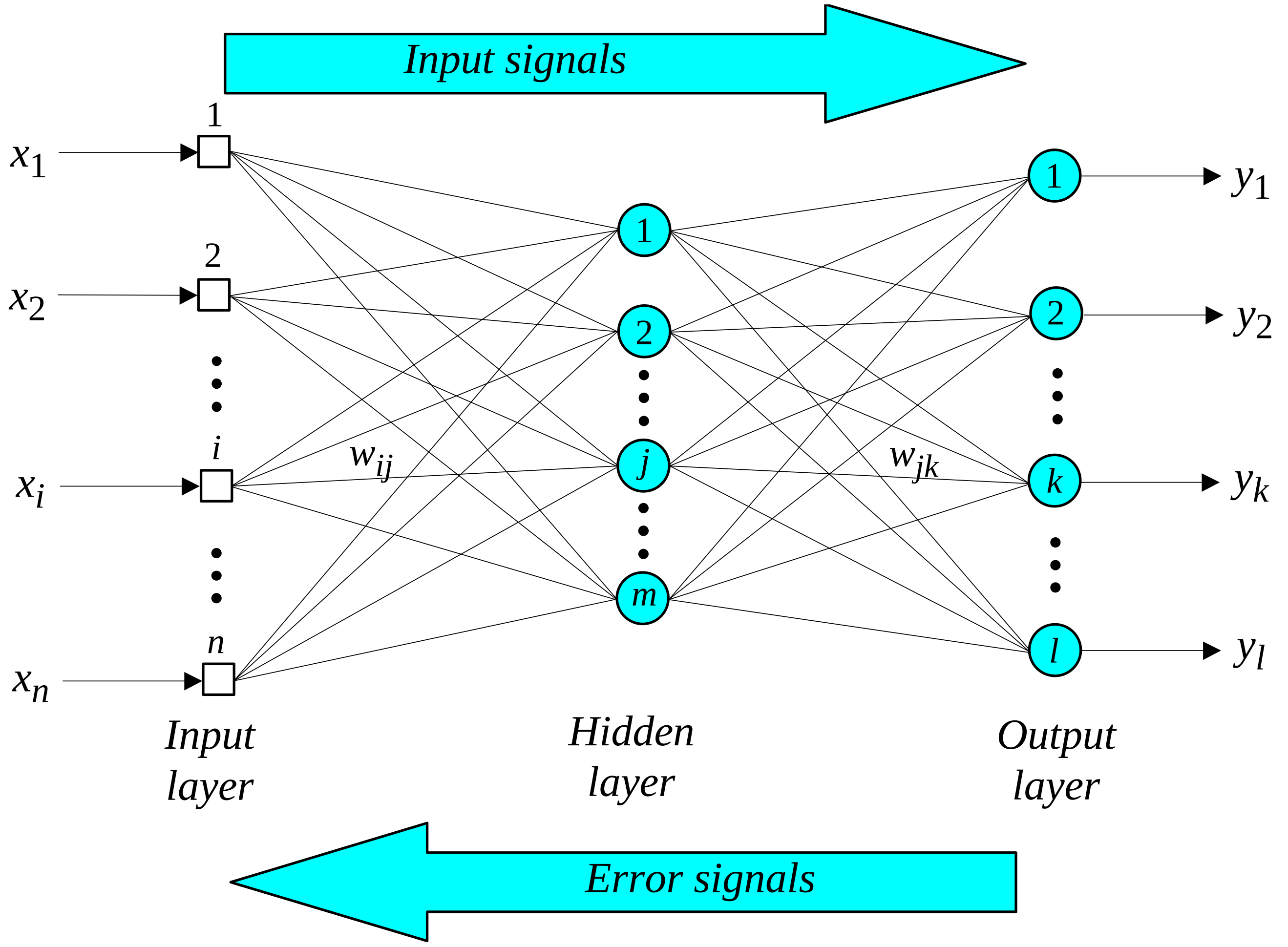
4. Adjust all weights (from i to j) at output and hidden nodes on the basis of their calculated error. $\Delta w_{ij} = \alpha\, \delta_j\, y_i$

Back-propagation neural network

- Learning in a multilayer network proceeds the same way as for a perceptron.
- A training set of input patterns is presented to the network.
- The network computes its output pattern, and if there is an error - or in other words a difference between actual and desired output patterns - the weights are adjusted to reduce this error.

- In a back-propagation neural network, the learning algorithm has two phases.
- First, a training input pattern is presented to the network input layer. The network propagates the input pattern from layer to layer until the output pattern is generated by the output layer.
- If this pattern is different from the desired output, an error is calculated and then propagated backwards through the network from the output layer to the input layer. The weights are modified as the error is propagated.

Three-layer back-propagation neural network



The back-propagation training algorithm

<u>Step 1</u>: Initialisation

Set all the weights and threshold levels of the network to random numbers uniformly distributed inside a small range:

$$\left(-\frac{2.4}{F_i}, \quad +\frac{2.4}{F_i}\right)$$

where *Fi* is the total number of inputs of neuron *i* in the network. The weight initialisation is done on a neuron-by-neuron basis.

Step 2: Activation

Activate the back-propagation neural network by applying inputs $x_1(p)$, $x_2(p)$,…, $x_n(p)$ and desired outputs $y_{d,1}(p)$, $y_{d,2}(p)$,…, $y_{d,n}(p)$.

(A)Calculate the actual outputs of the neurons in the hidden layer:

$$y_j(p) = sigmoid\left[\sum_{i=1}^{n} x_i(p) \cdot w_{ij}(p) - \theta_j\right]$$

where *n* is the number of inputs of neuron *j* in the hidden layer, and *sigmoid* is the *sigmoid* activation function.

Step 2: Activation (continued)

(B)Calculate the actual outputs of the neurons in the output layer:

$$y_k(p) = sigmoid\left[\sum_{j=1}^{m} x_{jk}(p) \cdot w_{jk}(p) - \theta_k\right]$$

where *m* is the number of inputs of neuron *k* in the output layer.

Step 3: Weight training

Update the weights in the back-propagation network propagating backward the errors associated with output neurons.

(a) Calculate the error gradient for the neurons in the output layer:

$$\delta_k(p) = y_k(p) \cdot [1 - y_k(p)] \cdot e_k(p)$$

Where $\boxed{e_k(p) = y_{d,k}(p) - y_k(p)}$

Calculate the weight corrections:

$$\Delta w_{jk}(p) = \alpha \cdot y_j(p) \cdot \delta_k(p)$$

Update the weights at the output neurons:

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

Step 3: Weight training (continued)

(*b*) Calculate the error gradient for the neurons in the hidden layer:

$$\delta_j(p) = y_j(p) \cdot [1 - y_j(p)] \cdot \sum_{k=1}^{l} \delta_k(p)\ \mathrm{w}_{jk}(p)$$

Calculate the weight corrections:

$$\Delta w_{ij}(p) = \alpha \cdot y_i(p) \cdot \delta_j(p)$$

Update the weights at the hidden neurons:

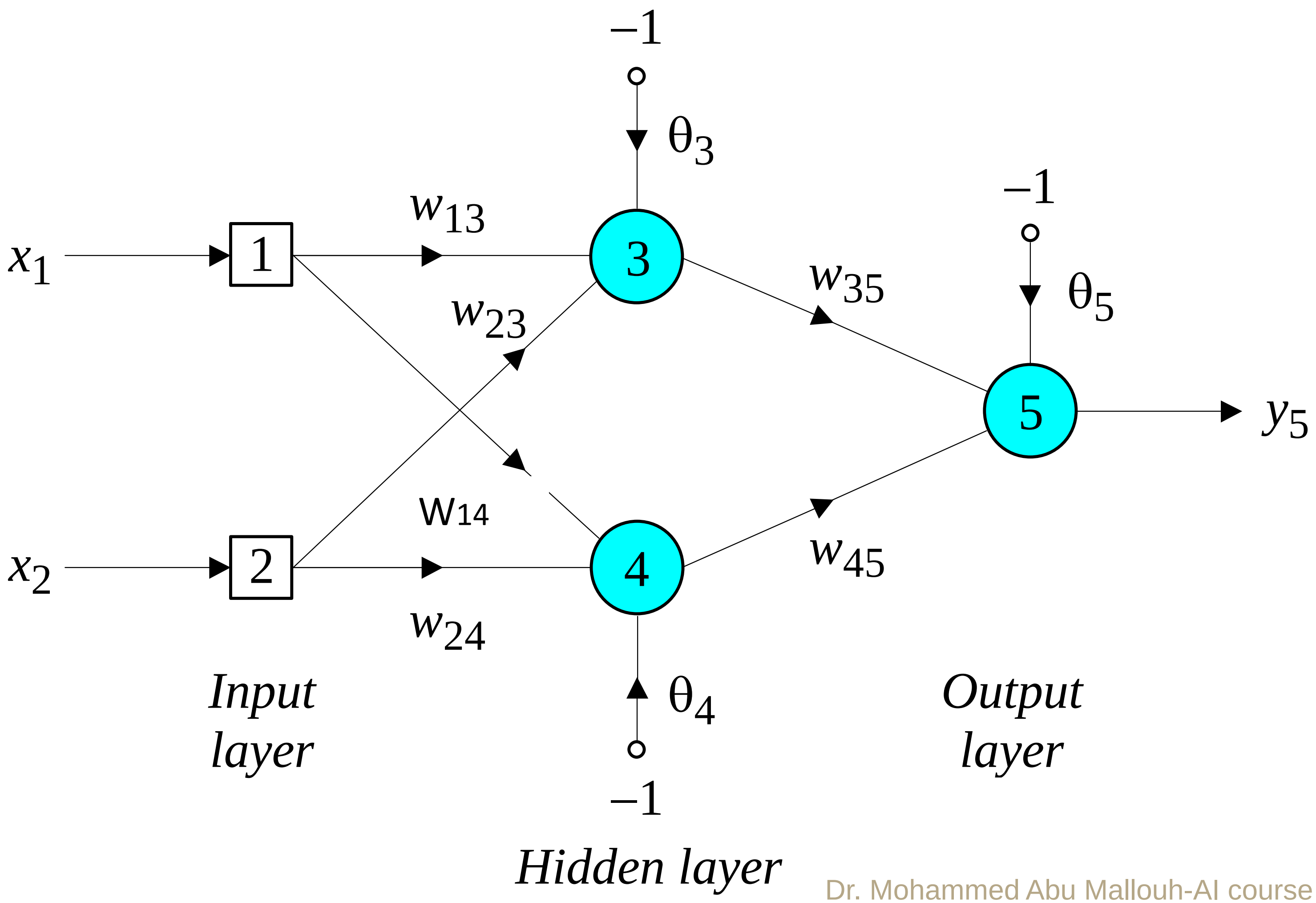
$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$

Step 4: Iteration

Increase iteration *p* by one, go back to *Step 2* and repeat the process until the selected error criterion is satisfied.

As an example, we may consider the three-layer back-propagation network. Suppose that the network is required to perform logical operation *Exclusive-OR*. Recall that a single-layer perceptron could not do this operation. Now we will apply the three-layer net.

Three-layer network for solving the Exclusive OR operation



- The effect of the threshold applied to a neuron inthe hidden or output layer is represented by its weight, θ, connected to a fixed input equal to -1.
- The initial weights and threshold levels are set randomly as follows:

$w_{13} = 0.5$, $w_{14} = 0.9$, $w_{23} = 0.4$, $w_{24} = 1.0$, $w_{35} = -1.2$,$w_{45} = 1.1$, $\theta_3 = 0.8$, $\theta_4 = -0.1$ and $\theta_5 = 0.3$.

➢We consider a training set where inputs *x*1 and *x*2 are equal to 1 and desired output $y_{d,5}$ is 0. The actual outputs of neurons 3 and 4 in the hidden layer are calculated as

$$y3 = sigmoid\ (x1w13 + x2w23 - \theta3) = 1/[1+ e^{-(1\times0.5+1\times0.4-1\times0.8)}\ ] = 0.5250$$
$$y4 = sigmoid\ (x1w14 + x2w24 - \theta4\ ) = 1/[1+ e^{-(1\times0.9+1\times1.0+1\times0.1)}\ ] = 0.8808$$

➢Now the actual output of neuron 5 in the output layer is determined as:

$$y5 = sigmoid(\ y3w35 + y4w45 - \theta5) = 1/[1+ e^{-(-0.5250\times1.2+0.8808\times1.1-1\times0.3)}\ ]= 0.5097$$

Thus, the following error is obtained:

$$\boxed{e = y_{d,5} - y_5 = 0 - 0.5097 = -0.5097}$$

- The next step is weight training. To update the weights and threshold levels in our network, we propagate the error, *e*, from the output layer backward to the input layer.

- First, we calculate the error gradient for neuron 5 in the output layer:

$$\delta_5 = y_5\,(1 - y_5)\,\mathrm{e} = 0.5097 \cdot (1 - 0.5097) \cdot (-0.5097) = -0.1274$$

- Then we determine the weight corrections assuming that the learning rate parameter, a, is equal to 0.1:

$$\Delta w_{35} = \alpha \cdot y_3 \cdot \delta_5 = 0.1 \cdot 0.5250 \cdot (-0.1274) = -0.0067$$

$$\Delta w_{45} = \alpha \cdot y_4 \cdot \delta_5 = 0.1 \cdot 0.8808 \cdot (-0.1274) = -0.0112$$

$$\Delta \theta_5 = \alpha \cdot (-1) \cdot \delta_5 = 0.1 \cdot (-1) \cdot (-0.1274) = -0.0127$$

- Next we calculate the error gradients for neurons 3 and 4 in the hidden layer:

$$\delta_3 = y_3(1 - y_3) \cdot \delta_5 \cdot \mathrm{w}_{35} = 0.5250 \cdot (1 - 0.5250) \cdot (-0.1274) \cdot (-1.2) = 0.0381$$

$$\delta_4 = y_4(1 - y_4) \cdot \delta_5 \cdot \mathrm{w}_{45} = 0.8808 \cdot (1 - 0.8808) \cdot (-0.127\,4) \cdot 1.1 = -0.0147$$

- We then determine the weight corrections:

$$\Delta w_{13} = \alpha \cdot x_1 \cdot \delta_3 = 0.1 \cdot 1 \cdot 0.0381 = 0.0038$$

$$\Delta w_{23} = \alpha \cdot x_2 \cdot \delta_3 = 0.1 \cdot 1 \cdot 0.0381 = 0.0038$$

$$\Delta \theta_3 = \alpha \cdot (-1) \cdot \delta_3 = 0.1 \cdot (-1) \cdot 0.0381 = -0.0038$$

$$\Delta w_{14} = \alpha \cdot x_1 \cdot \delta_4 = 0.1 \cdot 1 \cdot (-0.0147) = -0.0015$$

$$\Delta w_{24} = \alpha \cdot x_2 \cdot \delta_4 = 0.1 \cdot 1 \cdot (-0.0147) = -0.0015$$

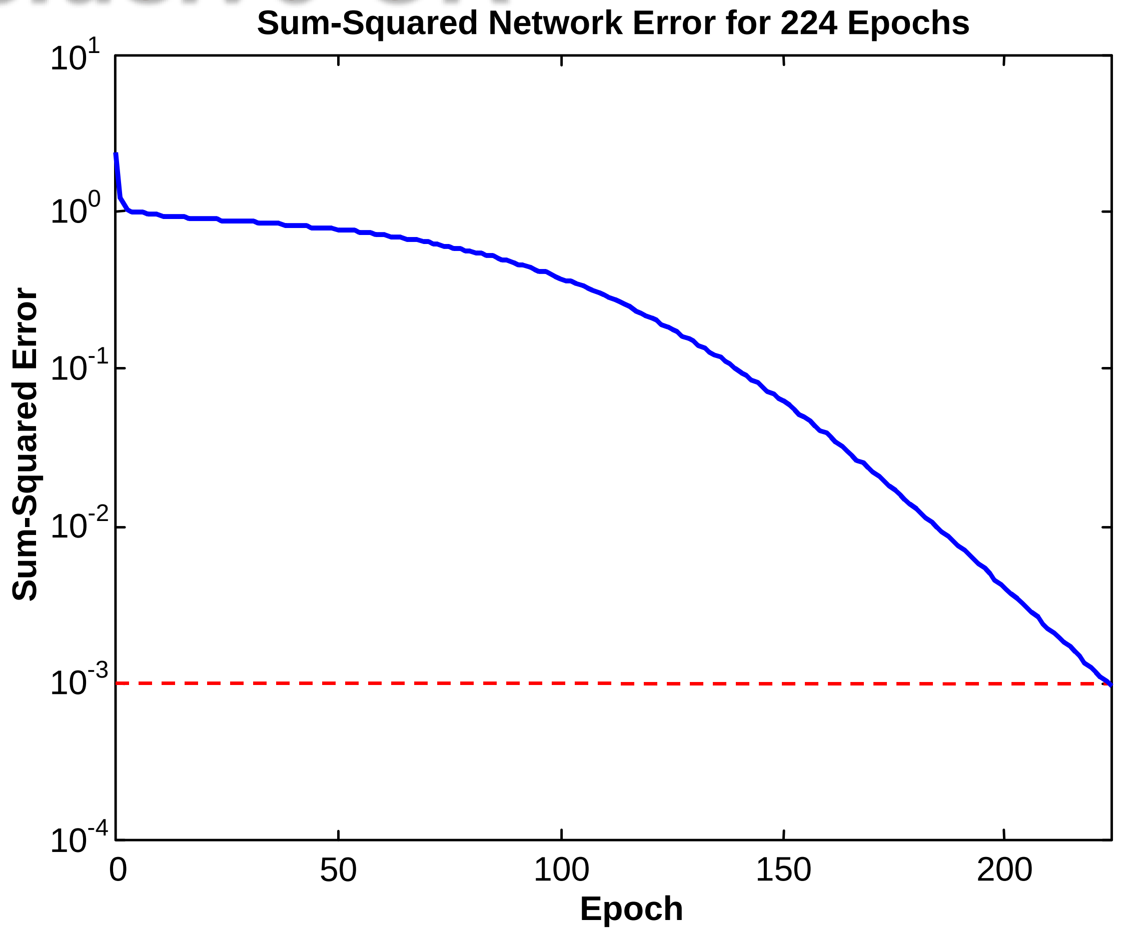
$$\Delta \theta_4 = \alpha \cdot (-1) \cdot \delta_4 = 0.1 \cdot (-1) \cdot (-0.0147) = 0.0015$$

- At last, we update all weights and threshold:

$$w_{13} = w_{13} + \Delta w_{13} = 0.5 + 0.0038 = 0.5038$$
$$w_{14} = w_{14} + \Delta w_{14} = 0.9 - 0.0015 = 0.8985$$
$$w_{23} = w_{23} + \Delta w_{23} = 0.4 + 0.0038 = 0.4038$$
$$w_{24} = w_{24} + \Delta w_{24} = 1.0 - 0.0015 = 0.9985$$
$$w_{35} = w_{35} + \Delta w_{35} = -1.2 - 0.0067 = -1.2067$$
$$w_{45} = w_{45} + \Delta w_{45} = 1.1 - 0.0112 = 1.0888$$
$$\theta_3 = \theta_3 + \Delta\theta_3 = 0.8 - 0.0038 = 0.7962$$
$$\theta_4 = \theta_4 + \Delta\theta_4 = -0.1 + 0.0015 = -0.0985$$
$$\theta_5 = \theta_5 + \Delta\theta_5 = 0.3 + 0.0127 = 0.3127$$

- The training process is repeated until the sum of squared errors is less than 0.001.

Learning curve for operation *Exclusive-OR*

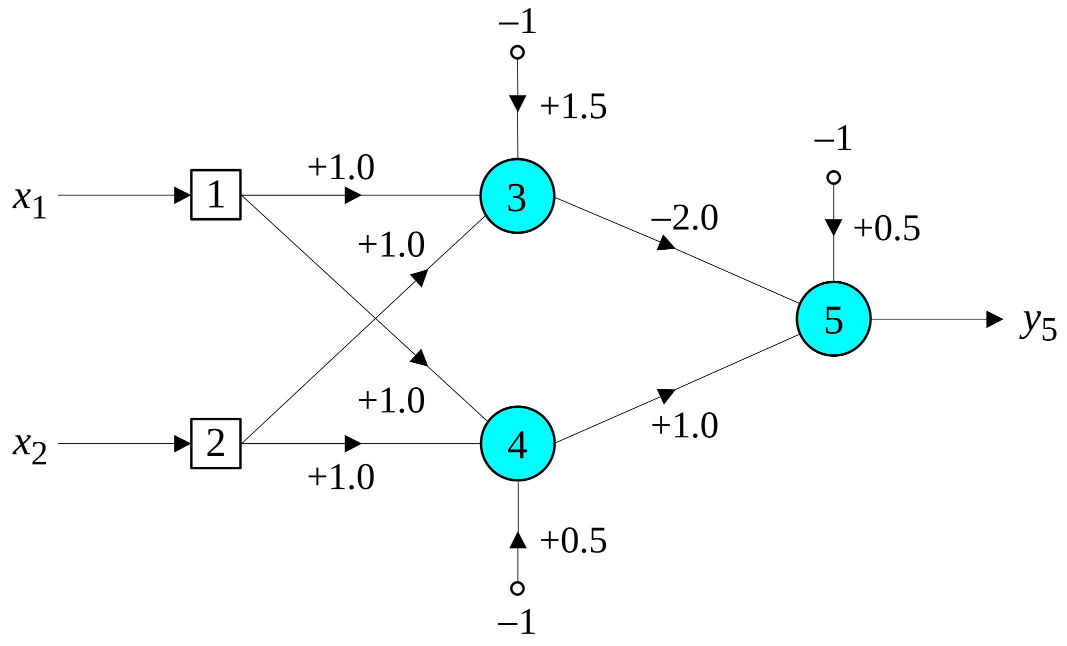


traingd Gradient Descent

Final results of three-layer network learning

Inputs		Desired output	Actual output	Error	Sum of squared errors
x_1	x_2	y_d	y_5	e	
1	1	0	0.0155	–0.0155	0.0010
0	1	1	0.9849	0.0151	
1	0	1	0.9849	0.0151	
0	0	0	0.0175	–0.0175	

Network represented by McCulloch-Pitts model for solving the *Exclusive-OR* operation



Accelerated learning in multilayer neural networks

- We can accelerate training by including a **momentum term** in the delta rule:

$$\Delta w_{jk}(p) = \beta \cdot \Delta w_{jk}(p-1) + \alpha \cdot y_j(p) \cdot \delta_k(p)$$

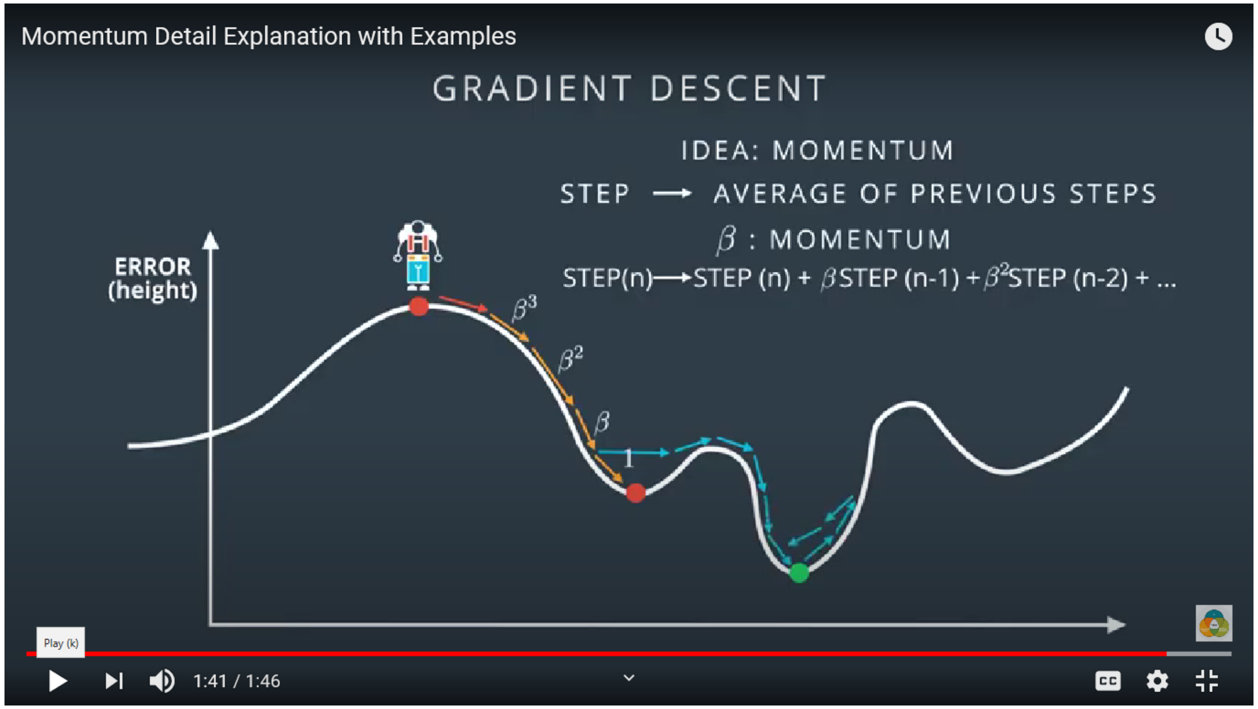
where β is a positive number ($0 \leq \beta < 1$) called the **momentum constant**. Typically, the momentum constant is set to 0.95.

https://youtu.be/6iwvtzXZ4Mo?t=14

Accelerated learning

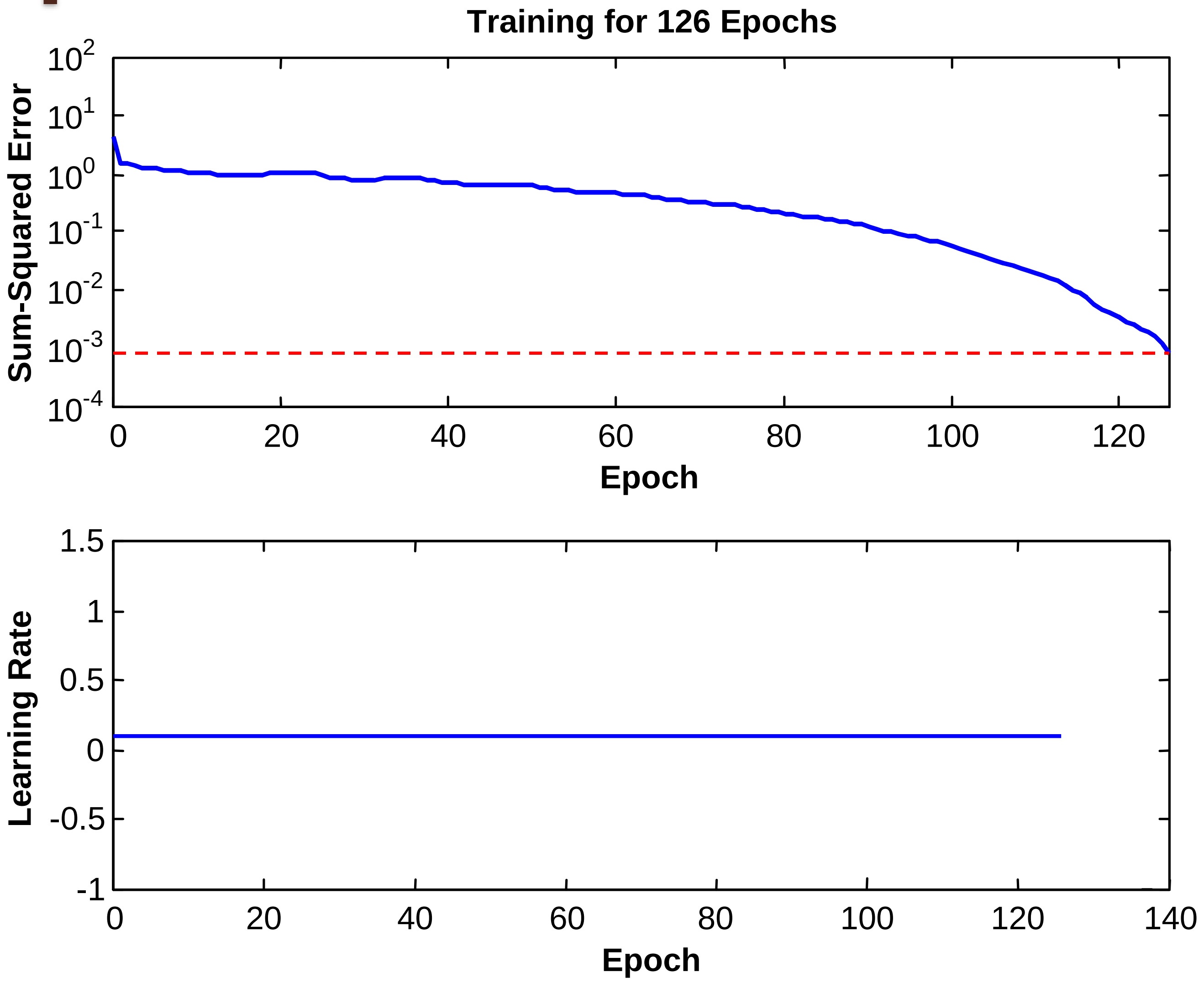
- We can accelerate training by including a **momentum term** in the delta rule:

$$\Delta w_{jk}(p) = \beta \cdot \Delta w_{jk}(p-1) + \alpha \cdot y_j(p) \cdot \delta_k(p)$$


Momentum Detail Explanation with Examples
GRADIENT DESCENT
IDEA: MOMENTUM
STEP → AVERAGE OF PREVIOUS STEPS
β : MOMENTUM
STEP(n)→STEP (n) + βSTEP (n-1) + β²STEP (n-2) + ...
ERROR (height)
β³
β²
β
1
Play (k)
1:41 / 1:46

https://youtu.be/6iwvtzXZ4Mo?t=14

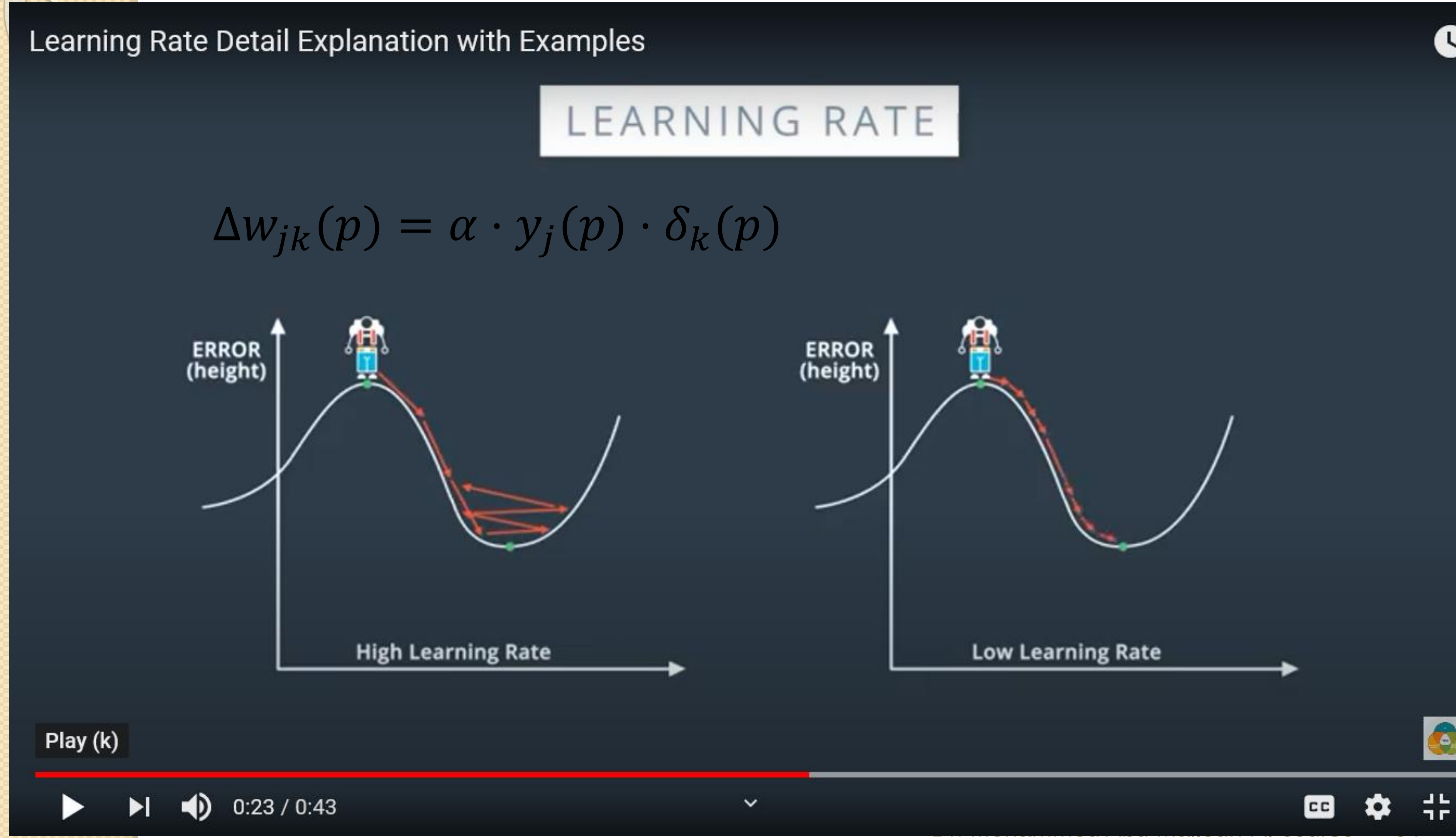
Learning with momentum for operation *Exclusive-OR*



traingdm	Gradient Descent with Momentum

Learning with adaptive learning rate

To accelerate the convergence and yet avoid the danger of instability, we can apply two heuristics:


Learning Rate Detail Explanation with Examples

LEARNING RATE

$$\Delta w_{jk}(p) = \alpha \cdot y_j(p) \cdot \delta_k(p)$$

Learning with adaptive learning rate

To accelerate the convergence and yet avoid the danger of instability, we can apply two heuristics:

Heuristic 1

If the change of the sum of squared errors has the same algebraic sign for several consequent epochs, then the learning rate parameter, α, should be increased.

Heuristic 2

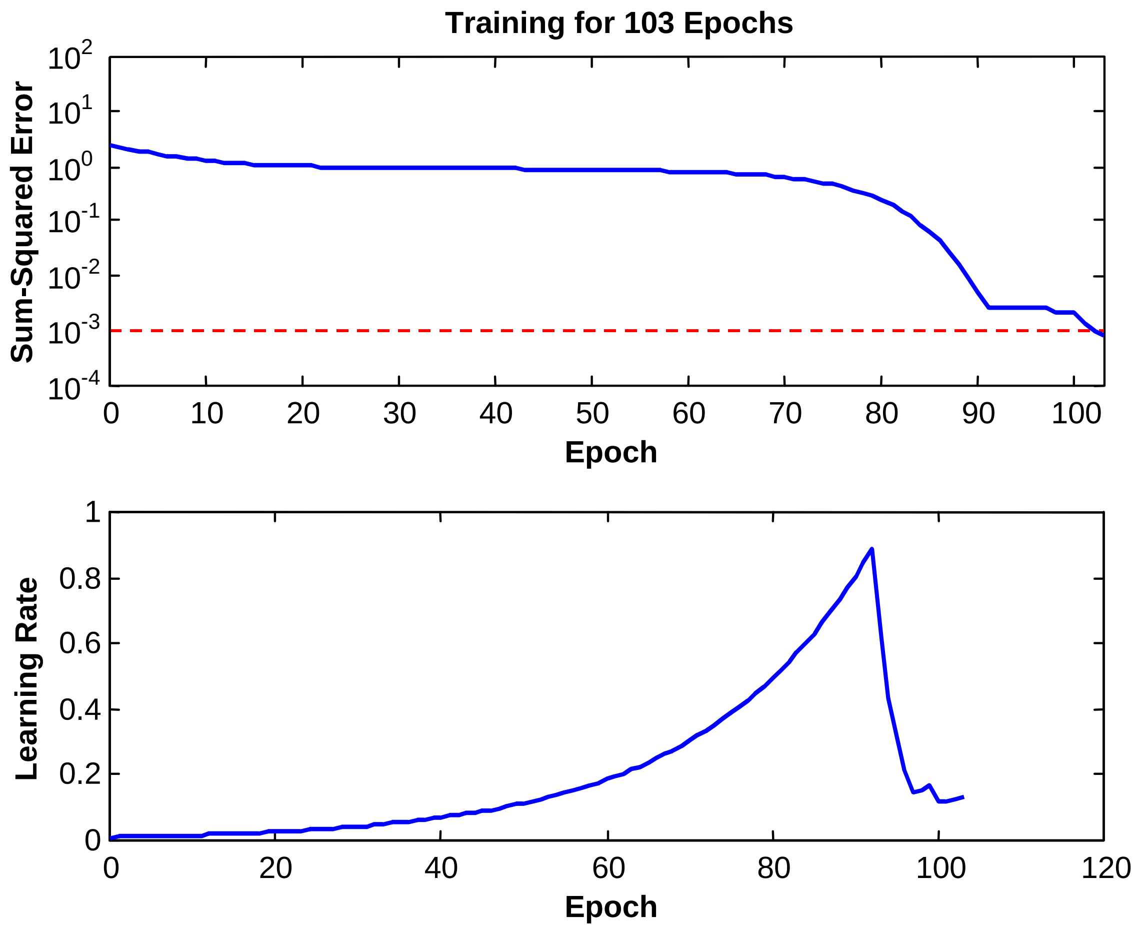
If the algebraic sign of the change of the sum of squared errors alternates for several consequent epochs, then the learning rate parameter, α, should be decreased.

https://youtu.be/TOtKVUtpz-s

- If the error is less than the previous one, the learning rate is increased (typically by multiplying by 1.05).

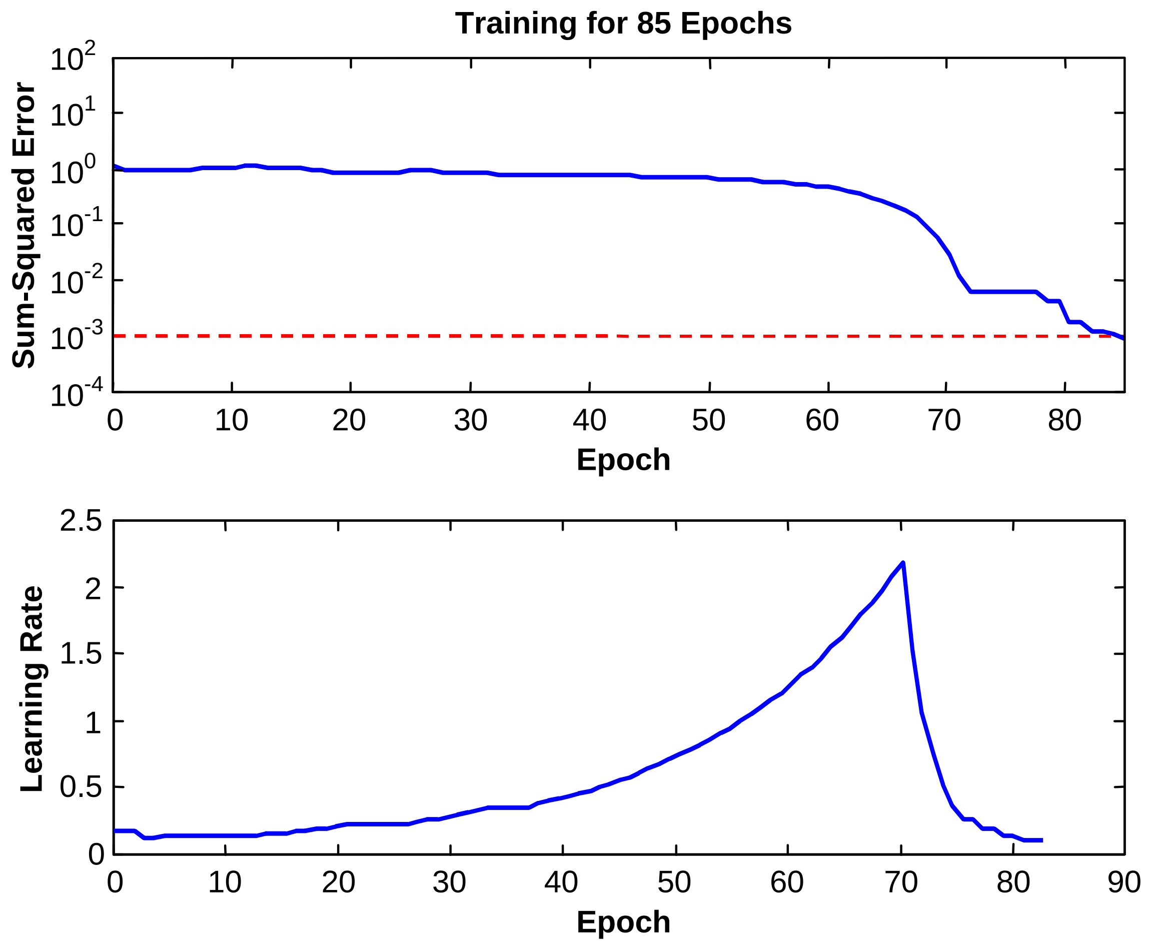
- If the sum of squared errors at the current epoch exceeds the previous value by more than a predefined ratio (typically 1.04), the learning rate parameter is decreased (typically by multiplying by 0.7) and new weights and thresholds are calculated.

Learning with adaptive learning rate



traingdA	Variable Learning Rate Gradient Descent

Learning with momentum and adaptive learning rate



traingdx

Divide Data for Optimal Neural Network Training

- One of the problems that occur during neural network training is called over fitting (overtraining). The error on the training set is driven to a very small value, but when new data is presented to the network the error is large. The network has memorized the training examples, but it has not learned to generalize to new situations.

- Note that if the number of parameters in the network is much smaller than the total number of points in the training set, then there is little or no chance of over fitting. If you can easily collect more data and increase the size of the training set, then there is no need to worry about over fitting.
- Minimum number of examples needed for training $\geq$ 2(number of NN parameters)

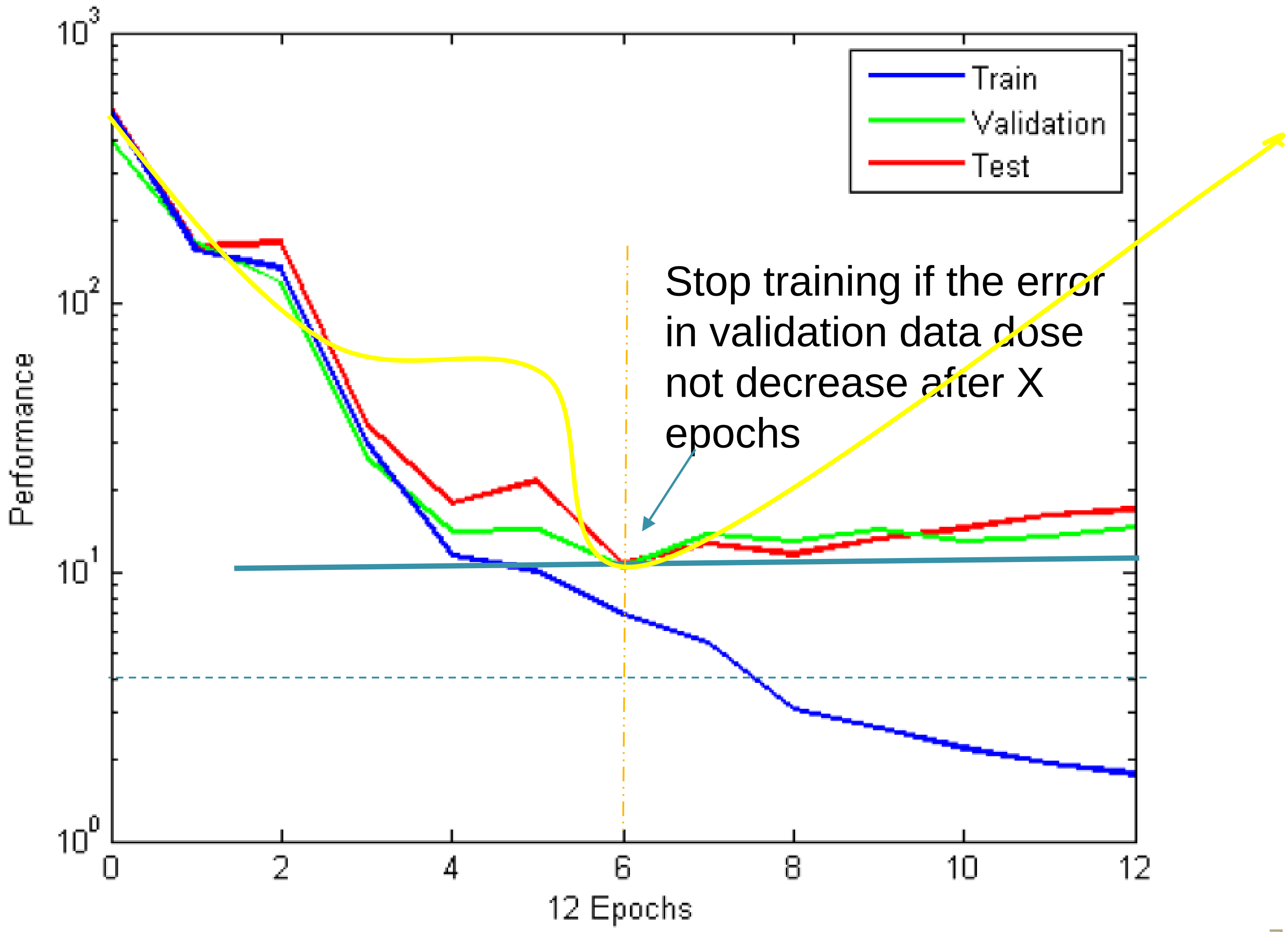
- One of the methods for improving generalization of a neural network is early stopping.

Divide Data for Optimal Neural Network Training

When training multilayer networks, the general practice is to first divide the data into three subsets.

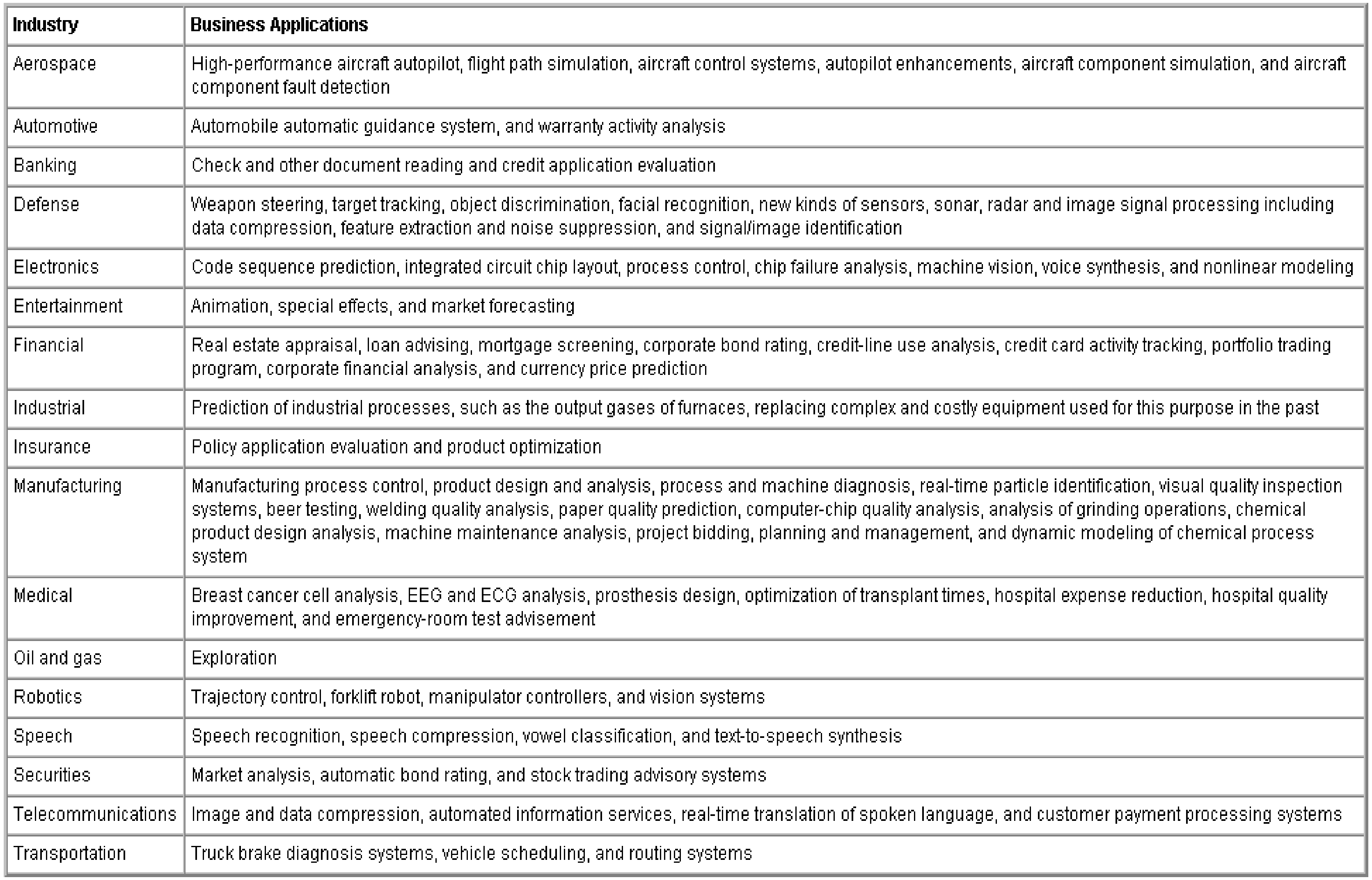
1) The first subset is the training set, which is used for computing the gradient and updating the network weights and biases (threshold). Usually 70% of total data
2) The second subset is the validation set. The error on the validation set is monitored during the training process. The validation error normally decreases during the initial phase of training, as does the training set error. However, when the network begins to overfit the data, the error on the validation set typically begins to rise. The network weights and biases are saved at the minimum of the validation set error. Usually 15% of total data
3) The test set represents a new data that were not used in training or validation to see how the system behaves for totally new data. It is also useful to plot the test set error during the training process. If the error on the test set reaches a minimum at a significantly different iteration number than the validation set error, this might indicate a poor division of the data set. Usually 15% of total data

Divide Data for Optimal Neural Network Training



Stop training if the error in validation data dose not decrease after X epochs

Applications



Industry	Business Applications
Aerospace	High-performance aircraft autopilot, flight path simulation, aircraft control systems, autopilot enhancements, aircraft component simulation, and aircraft component fault detection
Automotive	Automobile automatic guidance system, and warranty activity analysis
Banking	Check and other document reading and credit application evaluation
Defense	Weapon steering, target tracking, object discrimination, facial recognition, new kinds of sensors, sonar, radar and image signal processing including data compression, feature extraction and noise suppression, and signal/image identification
Electronics	Code sequence prediction, integrated circuit chip layout, process control, chip failure analysis, machine vision, voice synthesis, and nonlinear modeling
Entertainment	Animation, special effects, and market forecasting
Financial	Real estate appraisal, loan advising, mortgage screening, corporate bond rating, credit-line use analysis, credit card activity tracking, portfolio trading program, corporate financial analysis, and currency price prediction
Industrial	Prediction of industrial processes, such as the output gases of furnaces, replacing complex and costly equipment used for this purpose in the past
Insurance	Policy application evaluation and product optimization
Manufacturing	Manufacturing process control, product design and analysis, process and machine diagnosis, real-time particle identification, visual quality inspection systems, beer testing, welding quality analysis, paper quality prediction, computer-chip quality analysis, analysis of grinding operations, chemical product design analysis, machine maintenance analysis, project bidding, planning and management, and dynamic modeling of chemical process system
Medical	Breast cancer cell analysis, EEG and ECG analysis, prosthesis design, optimization of transplant times, hospital expense reduction, hospital quality improvement, and emergency-room test advisement
Oil and gas	Exploration
Robotics	Trajectory control, forklift robot, manipulator controllers, and vision systems
Speech	Speech recognition, speech compression, vowel classification, and text-to-speech synthesis
Securities	Market analysis, automatic bond rating, and stock trading advisory systems
Telecommunications	Image and data compression, automated information services, real-time translation of spoken language, and customer payment processing systems
Transportation	Truck brake diagnosis systems, vehicle scheduling, and routing systems

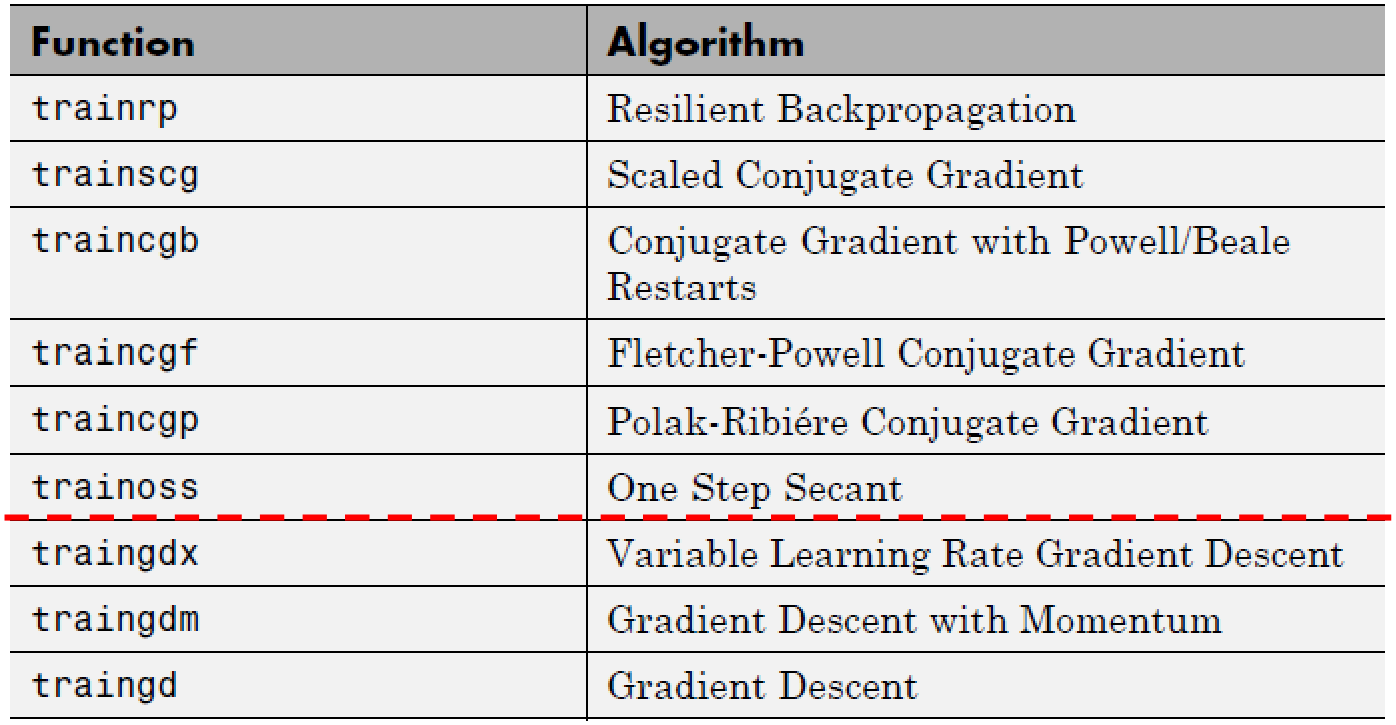
NN toolbox

As an example, the file housing. Mat contains a predefined set of input and target vectors. The input vectors define data regarding real-estate properties and the target values define relative values of the properties . Load the data using the following command:

Load house_dataset

Loading this file creates two variables. The input matrix house Inputs consists of 506 column vectors of 13 real estate variables for 506 different houses .The target matrix house Targets consists of the corresponding 506 relative valuations.

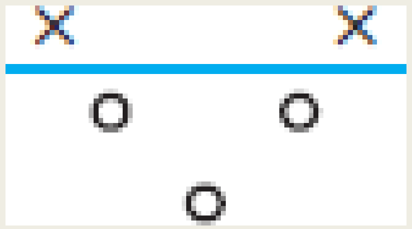
NN toolbox



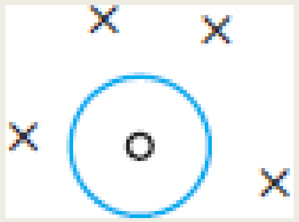
Function	Algorithm
`trainrp`	Resilient Backpropagation
`trainscg`	Scaled Conjugate Gradient
`traincgb`	Conjugate Gradient with Powell/Beale Restarts
`traincgf`	Fletcher-Powell Conjugate Gradient
`traincgp`	Polak-Ribiére Conjugate Gradient
`trainoss`	One Step Secant
`traingdx`	Variable Learning Rate Gradient Descent
`traingdm`	Gradient Descent with Momentum
`traingd`	Gradient Descent

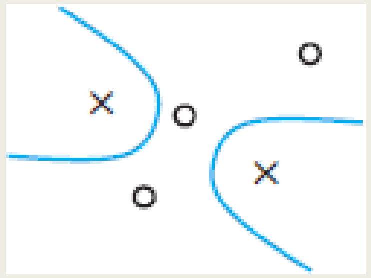
Radial Basis Functions NN

Linear vs. nonlinear separation



Linear Separation

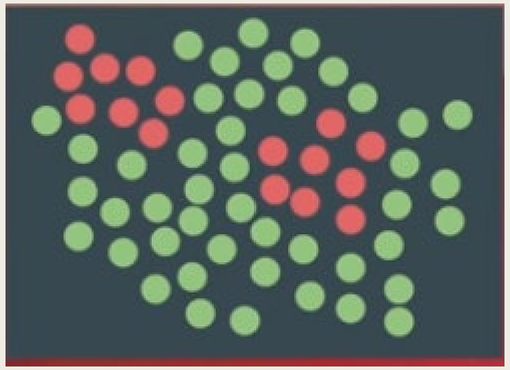
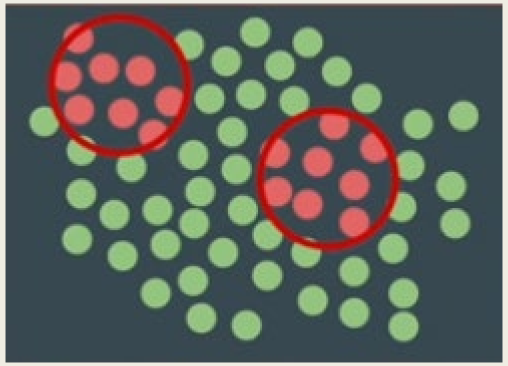
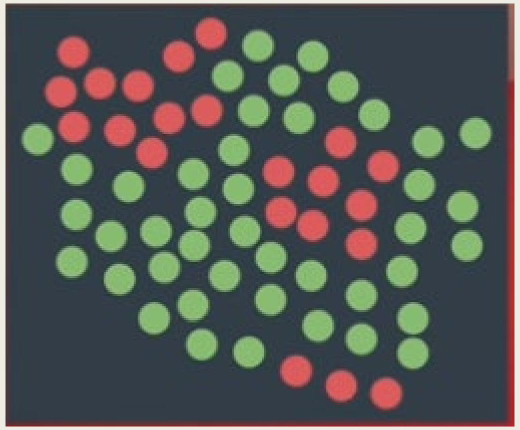
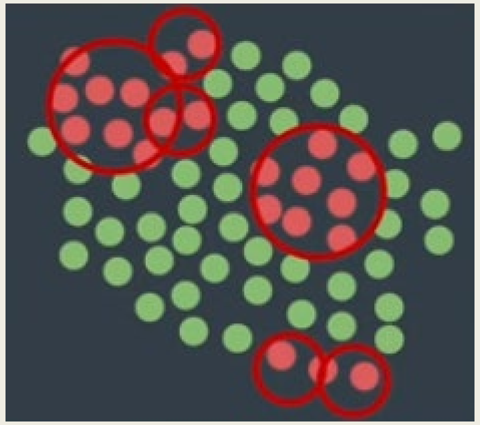




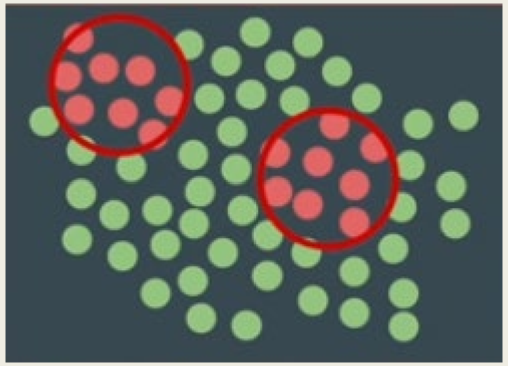
Nonlinear Separation

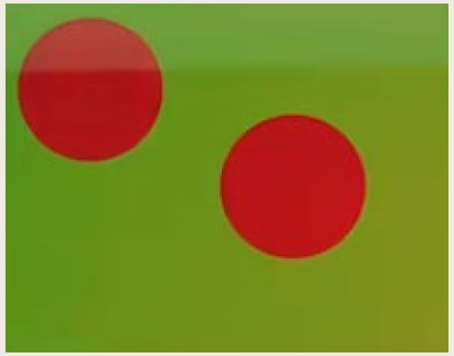
XOR is linearly separable ?

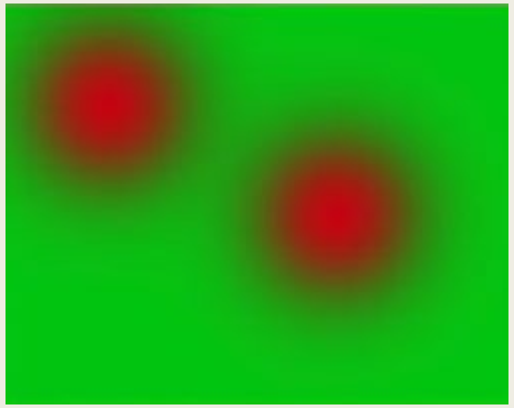
Radial Basis Functions NN

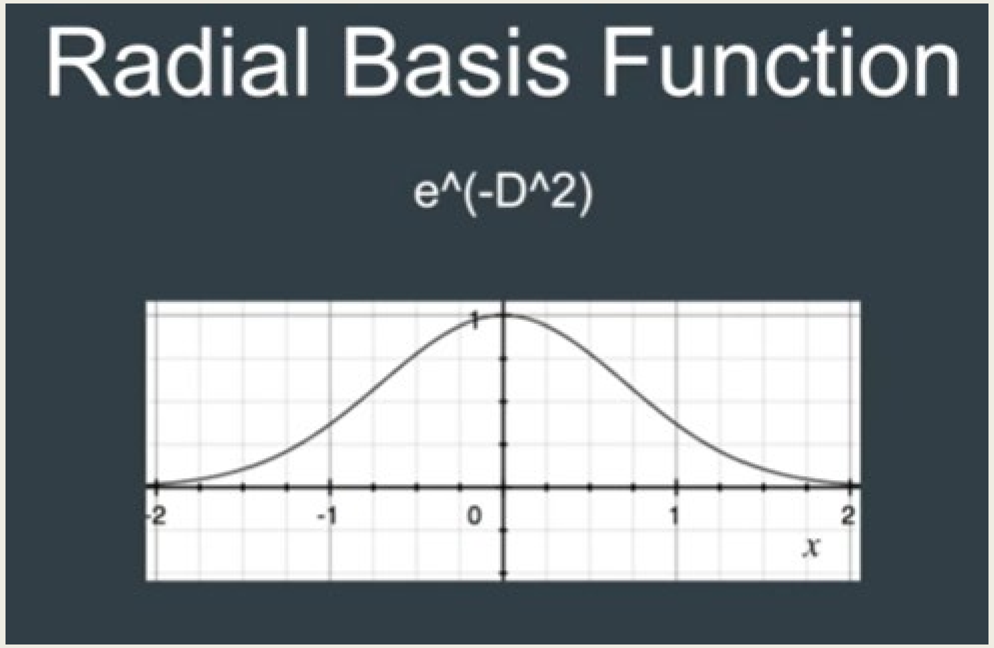





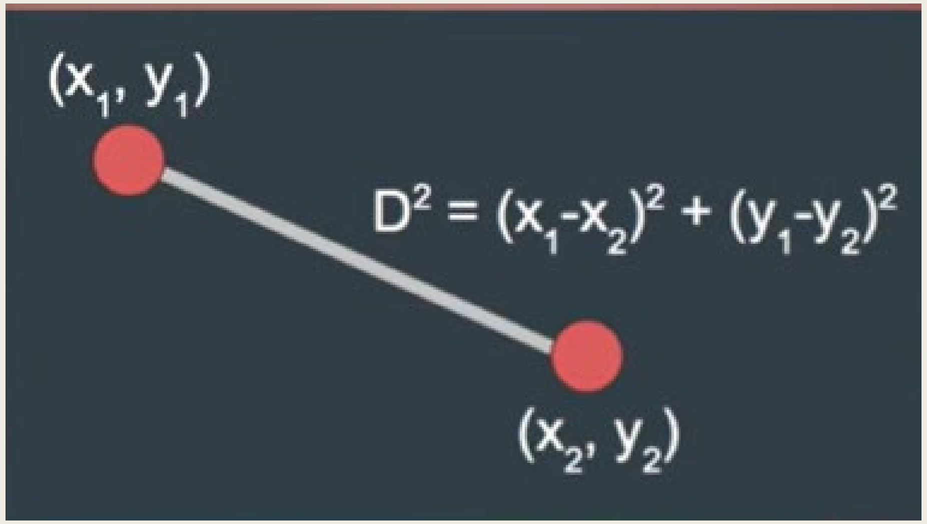
Linear vs. nonlinear separation

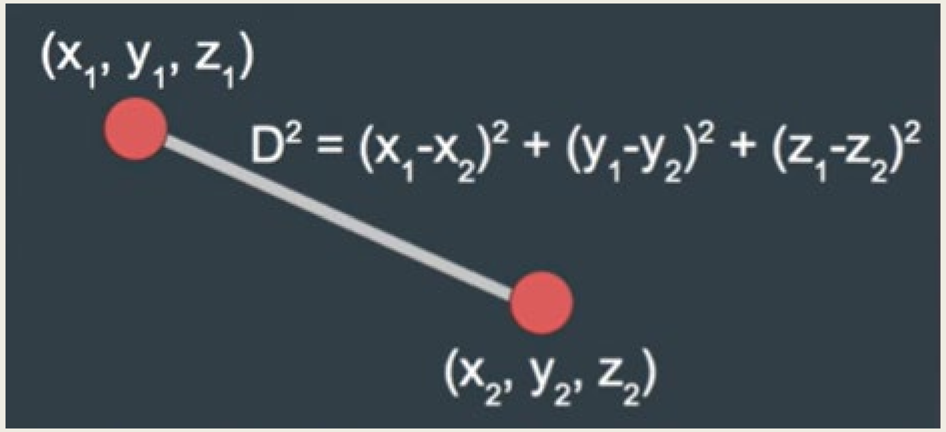






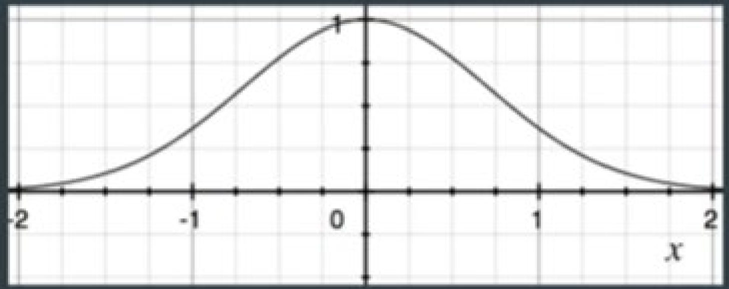


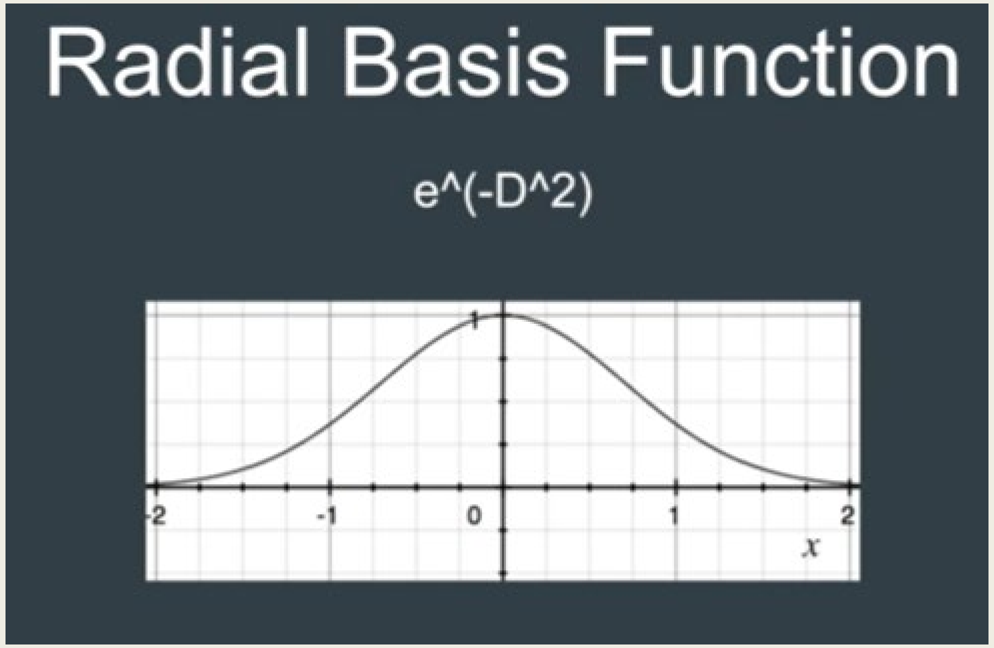




Radial Basis Function

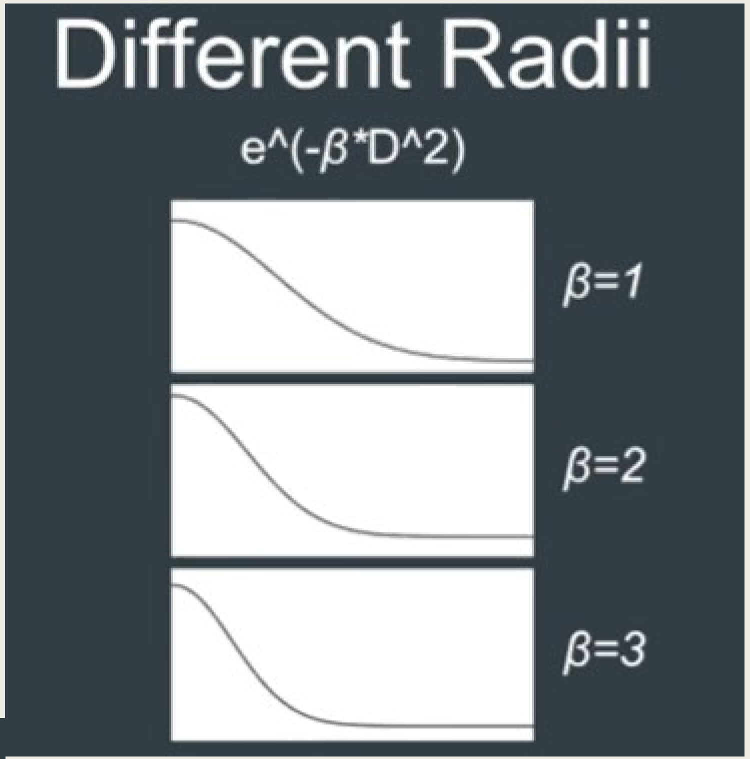
e^(-D^2)





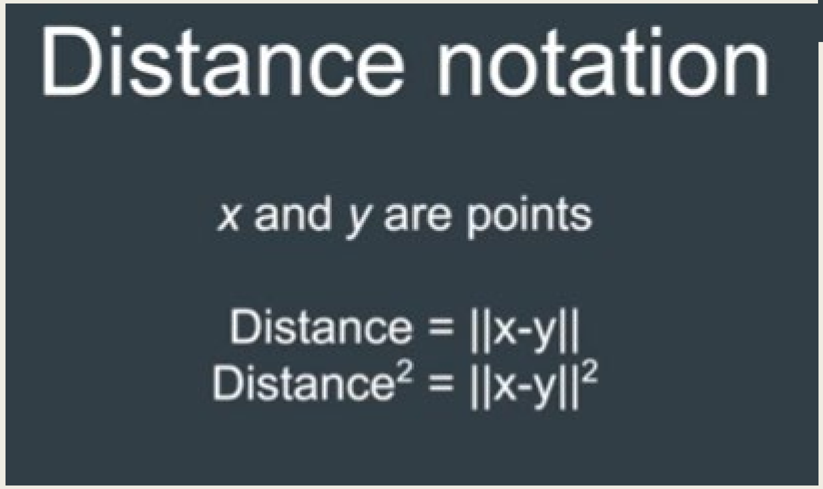
Radial Basis Function

e^(-D^2)



Different Radii

e^(-*β**D^2)



Distance notation

x and *y* are points

Distance = ||x-y||
Distance^2 = ||x-y||^2

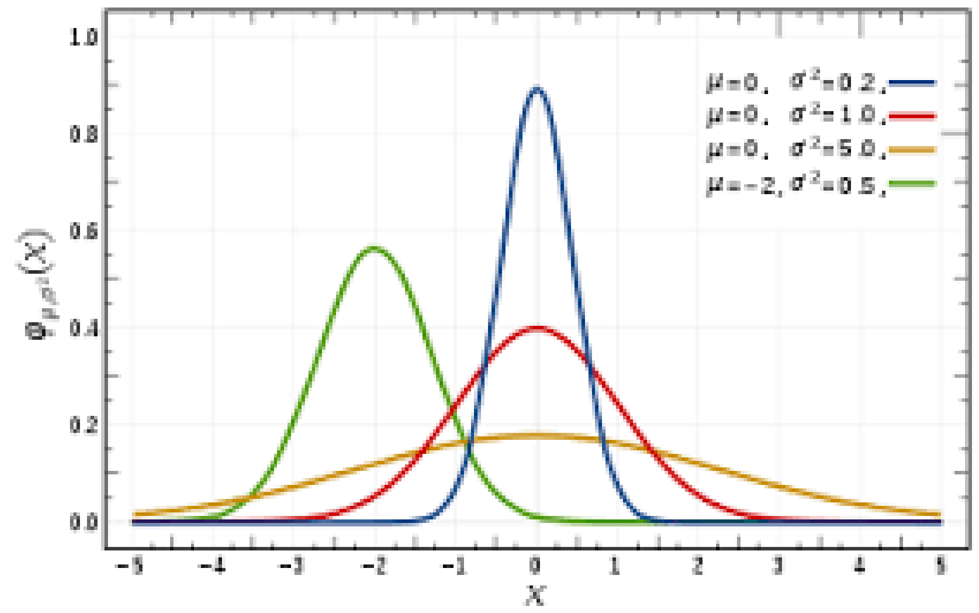
Radial Basis Functions

- RBF Functions use Gaussian-like functions
- RBF network can be used to perform complex pattern classification task and regression tasks also.
- Pattern Classification for nonlinear separation can be performed in two stages:
  - **Transform** nonlinear patterns into new linearly separated space
  - **Separate** the data using least-squares estimation

Gaussian Functions

In probability theory, a normal distribution is a continuous probability distribution for a real- valued random variable.

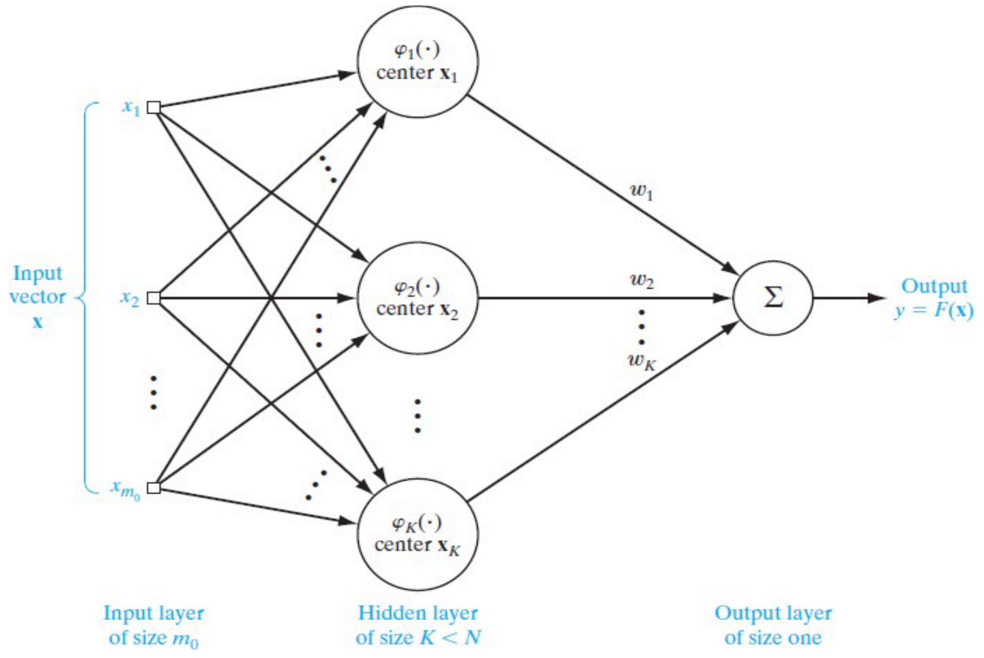
The general form of its probability density **function** is p(x). A random variable with a **Gaussian** distribution is said to be normally distributed and is called a normal deviate.



$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

RBF Network

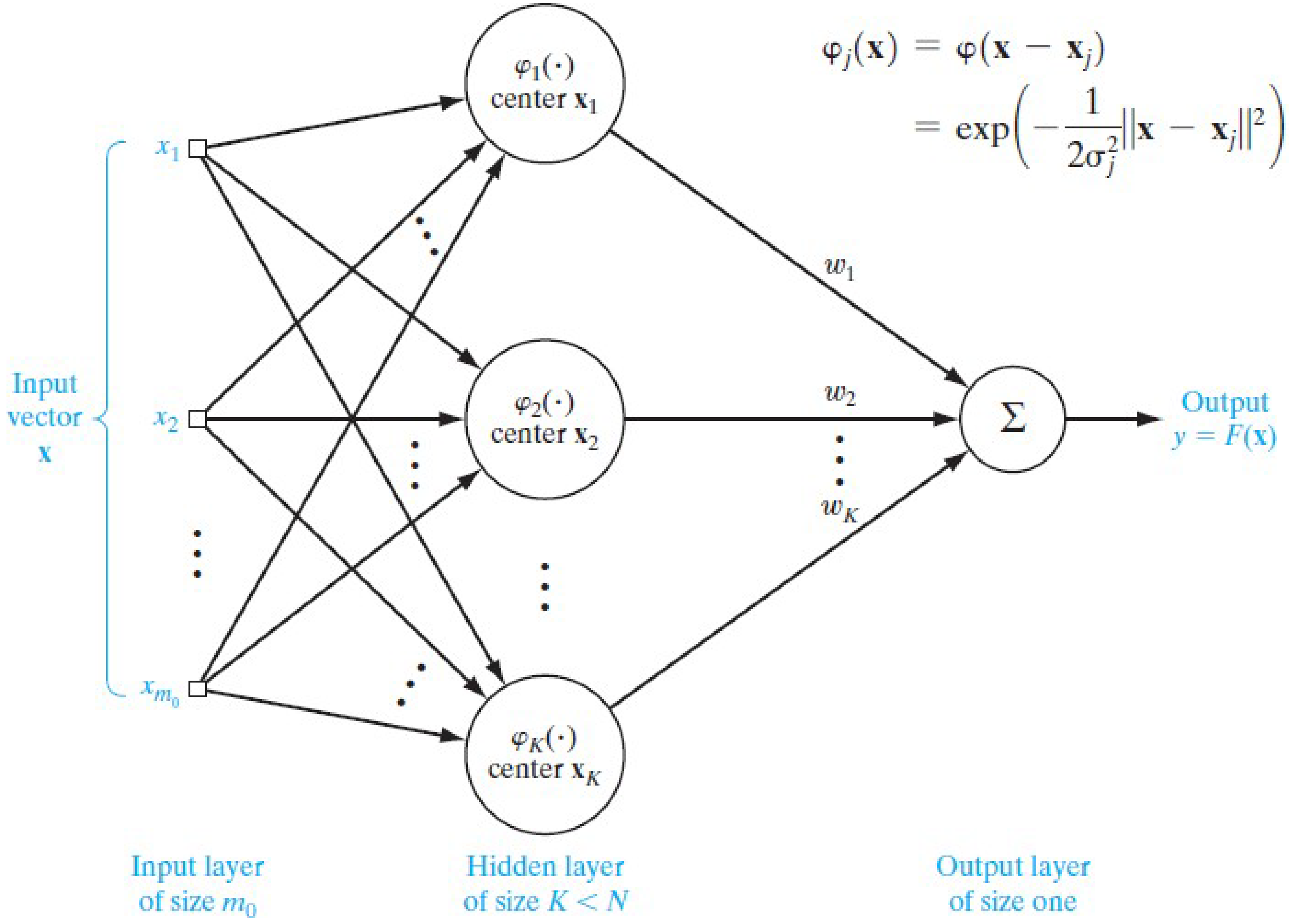
- RBF Network is composed of **three** layers
  - **Input layer** is made up of sensor inputs
  - **Hidden layer** applies nonlinear transformation from input space to hidden space
  - **Output layer** is linear



$\varphi_1(\cdot)$ center $\mathbf{x}_1$
$\varphi_2(\cdot)$ center $\mathbf{x}_2$
$\varphi_K(\cdot)$ center $\mathbf{x}_K$
x_1
x_2
x_{m_0}
Input vector $\mathbf{x}$
w_1
w_2
w_K
Σ
Output $y = F(\mathbf{x})$
Input layer of size m_0
Hidden layer of size $K < N$
Output layer of size one

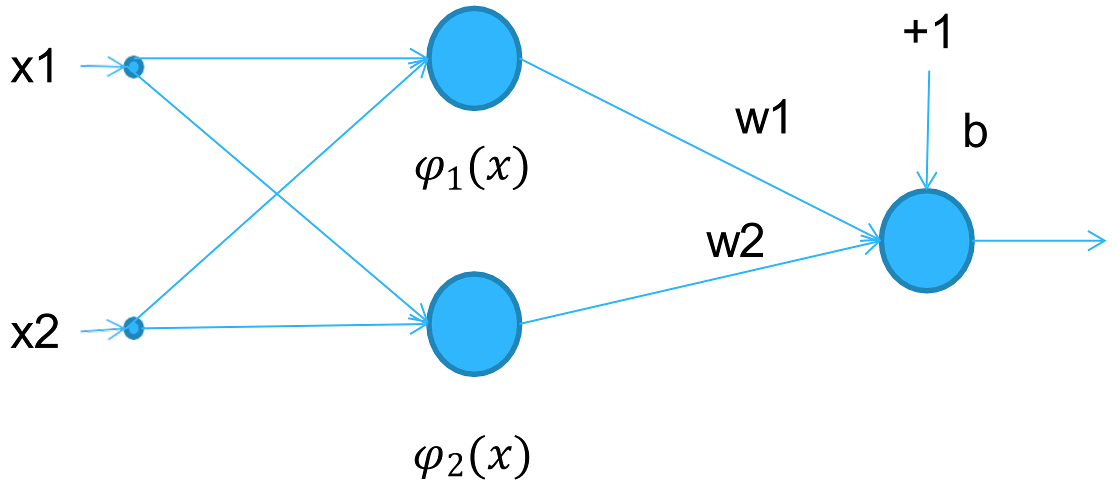
RBF Network Structure

$$\varphi_j(\mathbf{x}) = \varphi(\mathbf{x} - \mathbf{x}_j) = \exp\left(-\frac{1}{2\sigma_j^2}\|\mathbf{x} - \mathbf{x}_j\|^2\right), \qquad j = 1, 2, \ldots, N$$



Input vector $\mathbf{x}$
x_1
x_2
x_{m_0}
$\varphi_1(\cdot)$ center $\mathbf{x}_1$
$\varphi_2(\cdot)$ center $\mathbf{x}_2$
$\varphi_K(\cdot)$ center $\mathbf{x}_K$
w_1
w_2
w_K
Σ
Output $y = F(\mathbf{x})$
Input layer of size m_0
Hidden layer of size $K < N$
Output layer of size one

Simple RBF Network

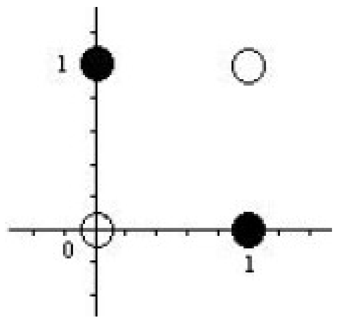


$$\varphi_i(x) = exp\left(-\frac{\|x - c_i\|^2}{2\sigma^2}\right)$$

$$y = w_1\varphi_1(x) + w_2\varphi_2(\mathrm{x}) + \mathrm{b}$$

XOR Problem Revisited

x2	x1	y
0	0	0
0	1	1
1	0	1
1	1	0



Linearly unseparable

$\varphi_1(x) = exp(-\|x - c_1\|^2)$ with centers *c1* = (1,1)

$\varphi_2(x) = exp(-\|x - c_2\|^2)$ with center *c2* = (0,0)

Pattern x	φ_1	φ_2
(0,0)	0.1353	1
(0,1)	0.3678	0.3678
(1,0)	0.3678	0.3678
(1,1)	1	0.1353

XOR Problem Revisited

Let w1 = w2 = -2.5, b = 2.84
c1=(1,1), c2=(0,0)

$$\varphi_i(x) = exp\left(-\frac{\|x - c_i\|^2}{2(0.5)}\right)$$

$$y = -2.5\ \varphi_1(x) - 2.5\ \varphi_2(\mathrm{x}) + 2.84$$

Input x=(0,1) or x=(1,0)

$$d_1 = (0-1)^2 + (1-1)^2 = 1$$
$$\varphi_1(x) = e^{-1} = 0.3678$$

$$d_2 = (0-0)^2 + (1-0)^2 = 1$$
$$\varphi_2(x) = e^{-1} = 0.3678$$

y = 2.84 - 2.5(0.3679) - 2.5(0.367) = 1

Input x=(0,0) or x=(1,1)

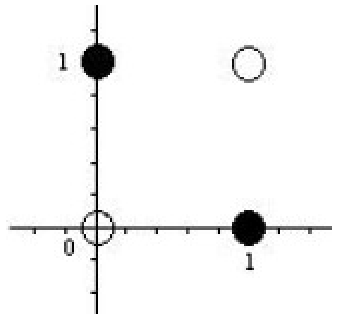
$$d_1 = (0-1)^2 + (0-1)^2 = 2$$
$$\varphi_1(x) = e^{-2} = 0.1353$$

$$d_2 = (0-0)^2 + (0-0)^2 = 0$$
$$\varphi_2(x) = e^0 = 1$$

y = 2.84 – 2.5(1) -2.5(0.1353) = 0

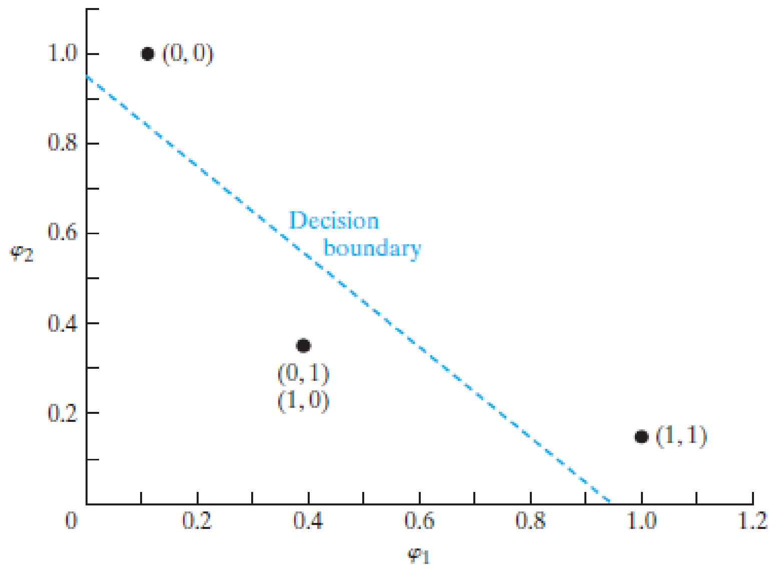
XOR Problem Re-visted

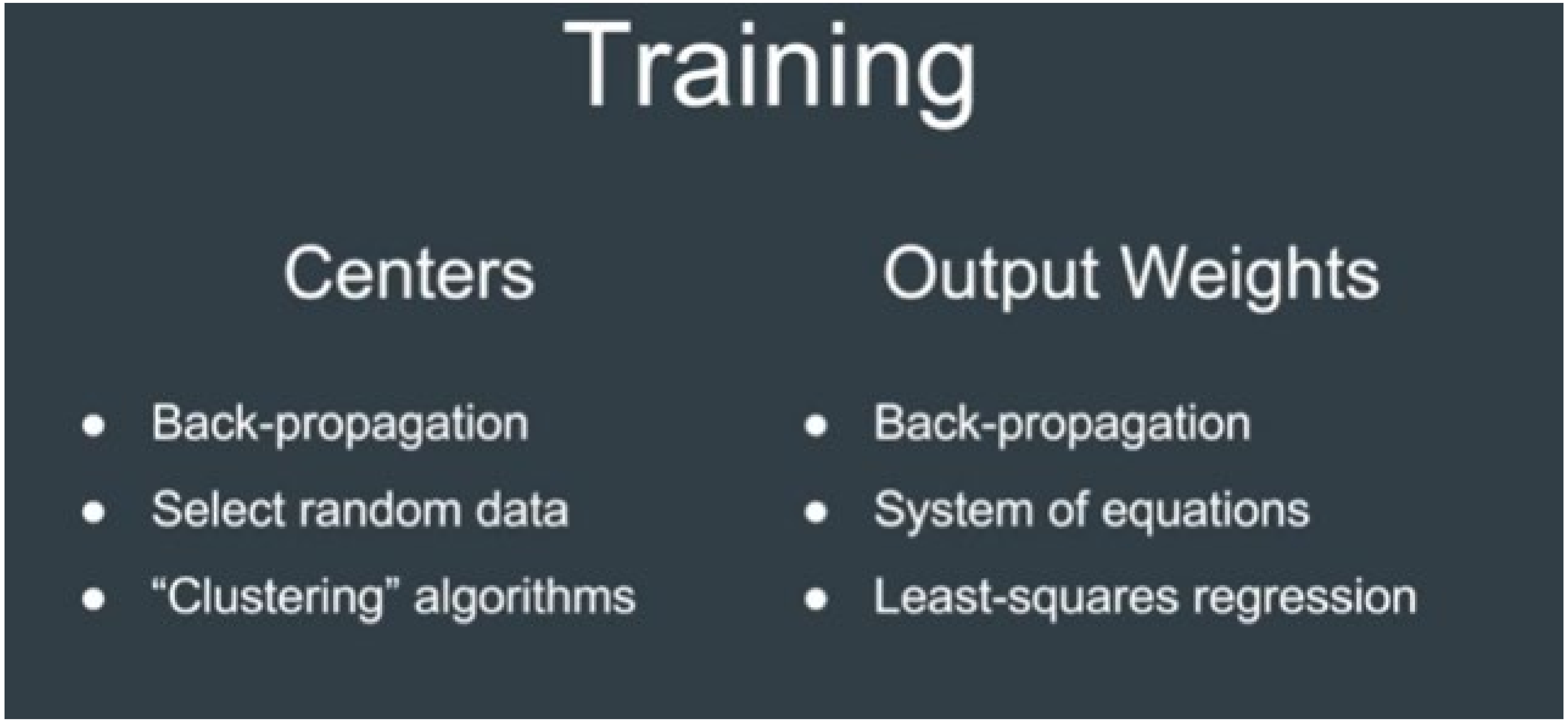
x2	x1	y
0	0	0
0	1	1
1	0	1
1	1	0



Transformation from input space to hidden space

Pattern x	φ_1	φ_2
(0,0)	0.1353	1
(0,1)	0.3678	0.3678
(1,0)	0.3678	0.3678
(1,1)	1	0.1353





Training

Centers

- Back-propagation
- Select random data
- “Clustering” algorithms

Output Weights

- Back-propagation
- System of equations
- Least-squares regression

Conclusions

- A Gaussian function is a normally distributed function that is used in probability theory

- Radial Basis Functions use Gaussian-like functions

- RBF Network is composed of three layers: input, hidden, and output
- The input layer is the input signals
- hidden layer transforms the input-space to a hidden-space
- The output layer uses a linear activation function

Lecture 8

Artificial neural networks:

Unsupervised learning

- Introduction
- Hebbian learning
- Generalised Hebbian learning algorithm
- Competitive learning
- Self-organising computational map: Kohonen network
- Summary

Introduction

The main property of a neural network is an ability to learn from its environment, and to improve its performance through learning. So far we have considered **supervised** or **active learning** – learning with an external "teacher" or a supervisor who presents a training set to the network. But another type of learning also exists: **unsupervised learning**.

- In contrast to supervised learning, unsupervised or **self-organised learning** does not require an external teacher. During the training session, the neural network receives a number of different input patterns, discovers significant features in these patterns and learns how to classify input data into appropriate categories. Unsupervised learning tends to follow the neuro-biological organisation of the brain.
- Unsupervised learning algorithms aim to learn rapidly and can be used in real-time.

Hebbian learning

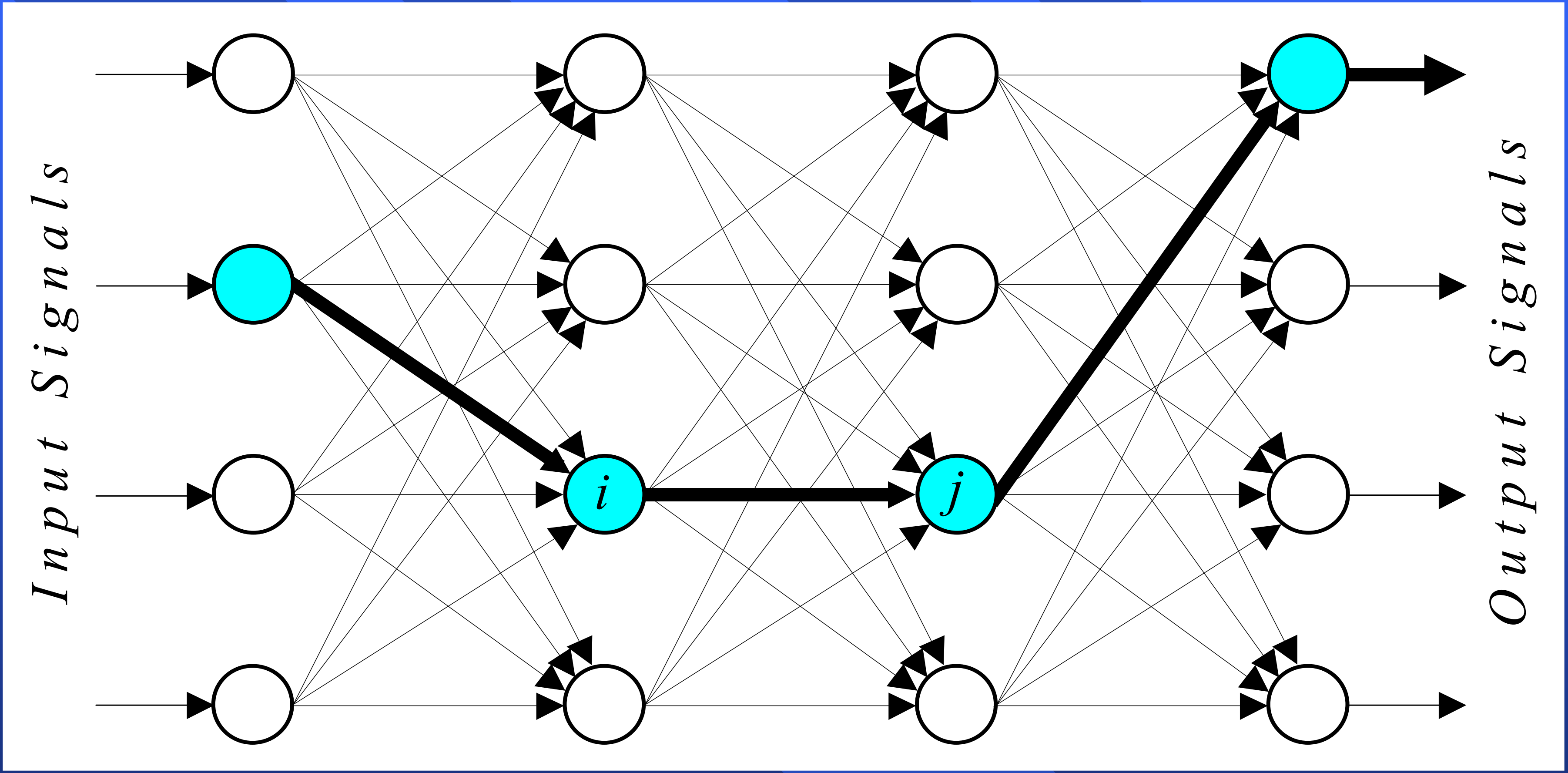
In 1949, Donald Hebb proposed one of the key ideas in biological learning, commonly known as **Hebb's Law**. Hebb's Law states that if neuron *i* is near enough to excite neuron *j* and repeatedly participates in its activation, the synaptic connection between these two neurons is strengthened and neuron *j* becomes more sensitive to stimuli from neuron *i*.

Hebb's Law can be represented in the form of two rules:

1. **If two neurons on either side of a connection are activated synchronously, then the weight of that connection is increased.**
2. **If two neurons on either side of a connection are activated asynchronously, then the weight of that connection is decreased.**

Hebb's Law provides the basis for learning without a teacher. Learning here is a **local phenomenon** occurring without feedback from the environment.

Hebbian learning in a neural network



- Using Hebb's Law we can express the adjustment applied to the weight w_{ij} at iteration p in the following form:

$$\Delta w_{ij}(p) = F[y_j(p), x_i(p)]$$

- As a special case, we can represent Hebb's Law as follows:

$$\Delta w_{ij}(p) = \alpha \; y_j(p) \; x_i(p)$$

where α is the *learning rate* parameter.

This equation is referred to as the **activity product rule**.

- Hebbian learning implies that weights can only increase. To resolve this problem, we might impose a limit on the growth of synaptic weights. It can be done by introducing a non-linear **forgetting factor** into Hebb's Law:

$$\Delta w_{ij}(p) = \alpha\ y_j(p)\ x_i(p) - \varphi\ y_j(p)\ w_{ij}(p)$$

where φ is the forgetting factor.

Forgetting factor usually falls in the interval between 0 and 1, typically between 0.01 and 0.1, to allow only a little "forgetting" while limiting the weight growth.

Hebbian learning algorithm

Step 1: Initialisation.

Set initial synaptic weights and thresholds to small random values, say in an interval [0, 1].

Step 2: Activation.

Compute the neuron output at iteration *p*

$$y_j(p) = \mathbf{Step}\left(\sum_{i=1}^{n} x_i(p)\, w_{ij}(p) - \theta_j\right)$$

where *n* is the number of neuron inputs, and θ_j is the threshold value of neuron *j*.

© Negnevitsky, Pearson Education, 2011

Step 3: Learning.

Update the weights in the network:

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

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

The weight correction is determined by the generalised activity product rule:

$$\Delta w_{ij}(p) = \varphi\, y_j(p)[\lambda\ x_i(p) - w_{ij}(p)]$$

Step 4: Iteration.

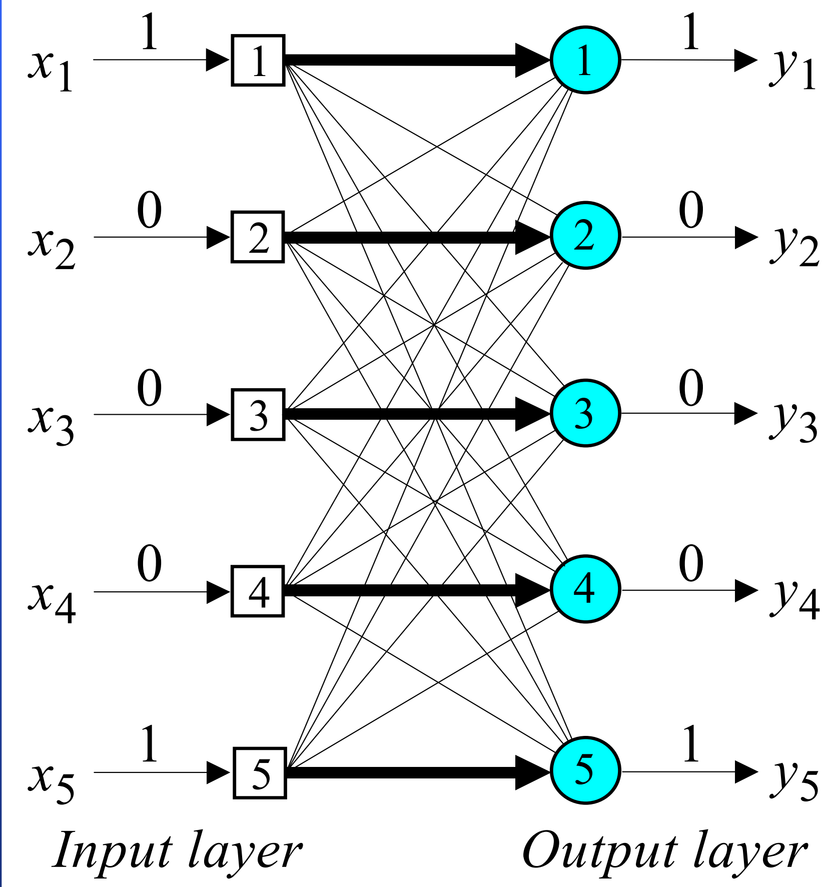
Increase iteration p by one, go back to Step 2.

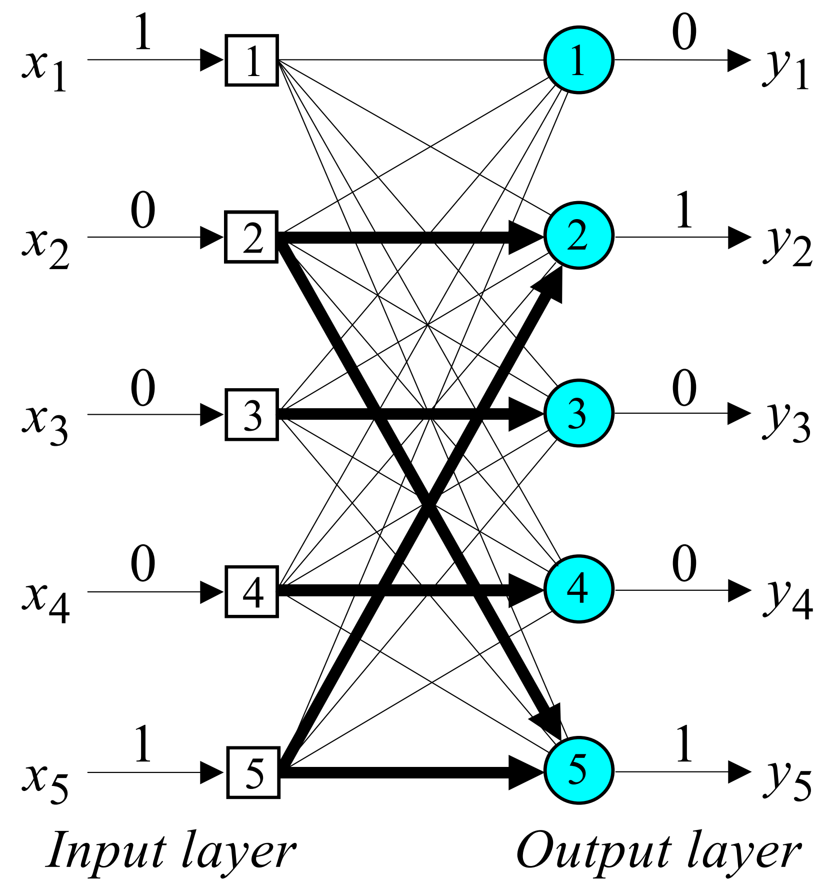
Hebbian learning example

To illustrate Hebbian learning, consider a fully connected feedforward network with a single layer of five computation neurons. Each neuron is represented by a McCulloch and Pitts model with the *step* activation function. The network is trained on the following set of input vectors:

$$\mathbf{X}_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{X}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad \mathbf{X}_3 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \mathbf{X}_4 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{X}_5 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Initial and final states of the network





Initial and final weight matrices

$$\begin{array}{cc|ccccc} & & \multicolumn{5}{c}{\textit{Output layer}} \\ & & 1 & 2 & 3 & 4 & 5 \\ \hline \textit{Input layer} & 1 & 1 & 0 & 0 & 0 & 0 \\ & 2 & 0 & 1 & 0 & 0 & 0 \\ & 3 & 0 & 0 & 1 & 0 & 0 \\ & 4 & 0 & 0 & 0 & 1 & 0 \\ & 5 & 0 & 0 & 0 & 0 & 1 \end{array}$$

$$\begin{array}{cc|ccccc} & & \multicolumn{5}{c}{\textit{Output layer}} \\ & & 1 & 2 & 3 & 4 & 5 \\ \hline \textit{Input layer} & 1 & 0 & 0 & 0 & 0 & 0 \\ & 2 & 0 & 2.0204 & 0 & 0 & 2.0204 \\ & 3 & 0 & 0 & 1.0200 & 0 & 0 \\ & 4 & 0 & 0 & 0 & 0.9996 & 0 \\ & 5 & 0 & 2.0204 & 0 & 0 & 2.0204 \end{array}$$

- A test input vector, or probe, is defined as

$$\mathbf{X} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

- When this probe is presented to the network, we obtain:

$$\mathbf{Y} = step\left\{ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 2.0204 & 0 & 0 & 2.0204 \\ 0 & 0 & 1.0200 & 0 & 0 \\ 0 & 0 & 0 & 0.9996 & 0 \\ 0 & 2.0204 & 0 & 0 & 2.0204 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 0.4940 \\ 0.2661 \\ 0.0907 \\ 0.9478 \\ 0.0737 \end{bmatrix} \right\} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Competitive learning

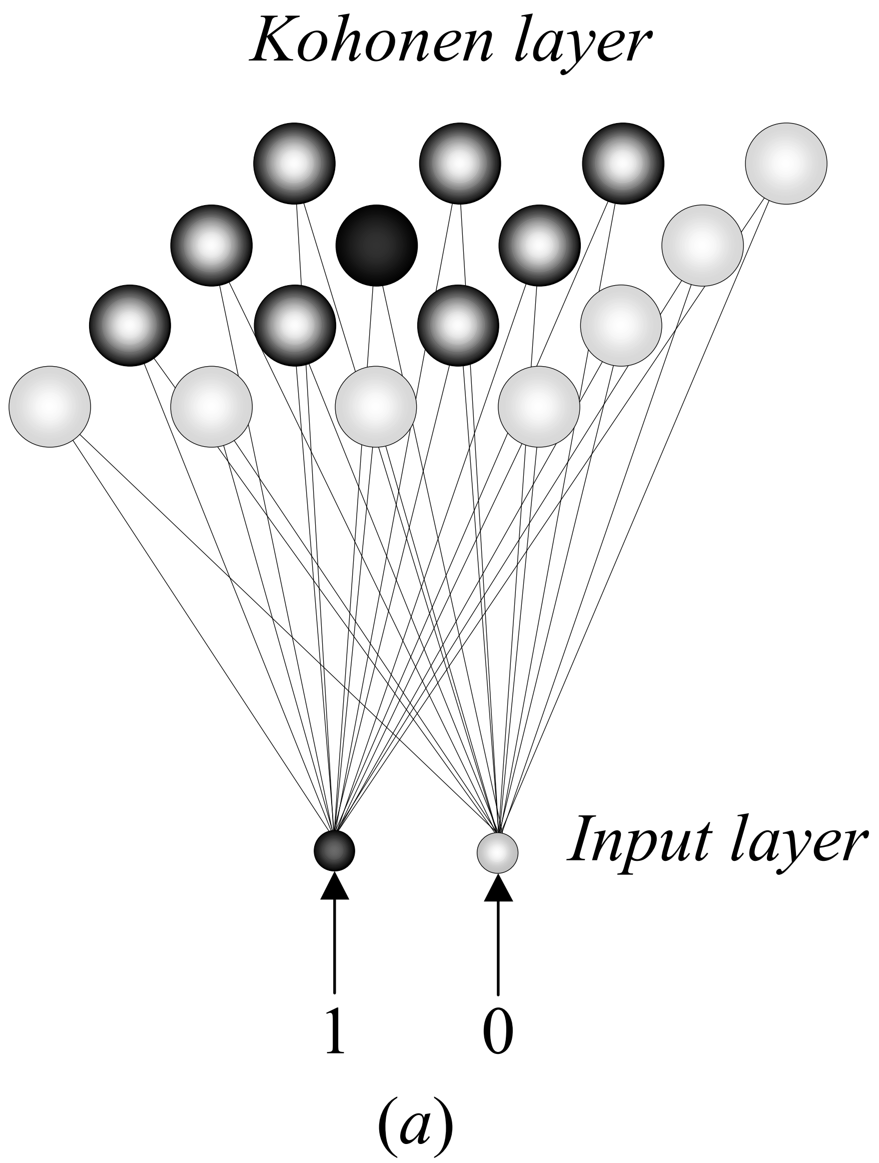
- In competitive learning, neurons compete among themselves to be activated.
- While in Hebbian learning, several output neurons can be activated simultaneously, in competitive learning, only a single output neuron is active at any time.
- The output neuron that wins the "competition" is called the ***winner-takes-all*** neuron.

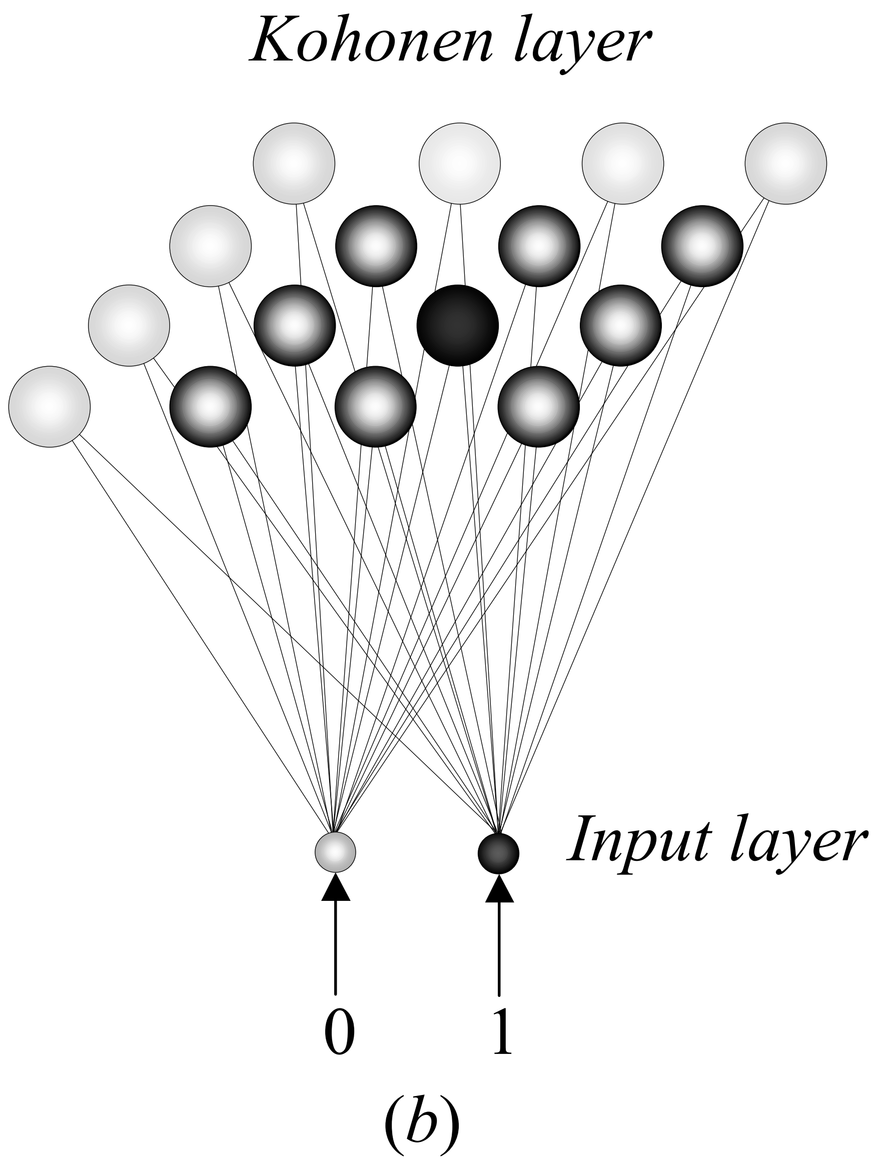
- The basic idea of competitive learning was introduced in the early 1970s.
- In the late 1980s, Teuvo Kohonen introduced a special class of artificial neural networks called **self-organising feature maps**. These maps are based on competitive learning.

What is a self-organising 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, somatosensory, etc.), and associated with different sensory inputs. We can say that each sensory input is mapped into a corresponding area of the cerebral cortex. **The cortex is a self-organising computational map in the human brain.**

Feature-mapping Kohonen model



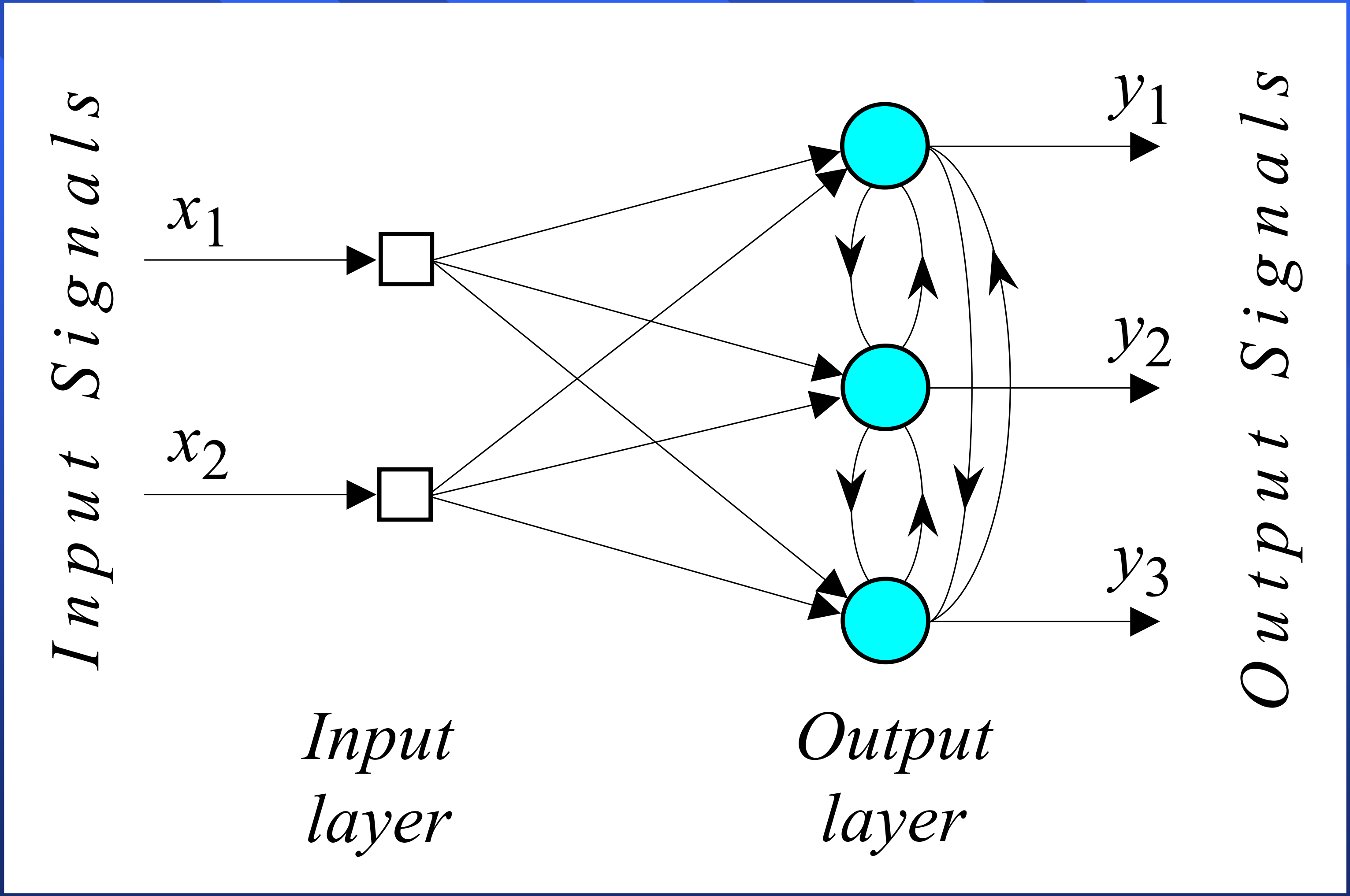


© Negnevitsky, Pearson Education, 2011

The Kohonen network

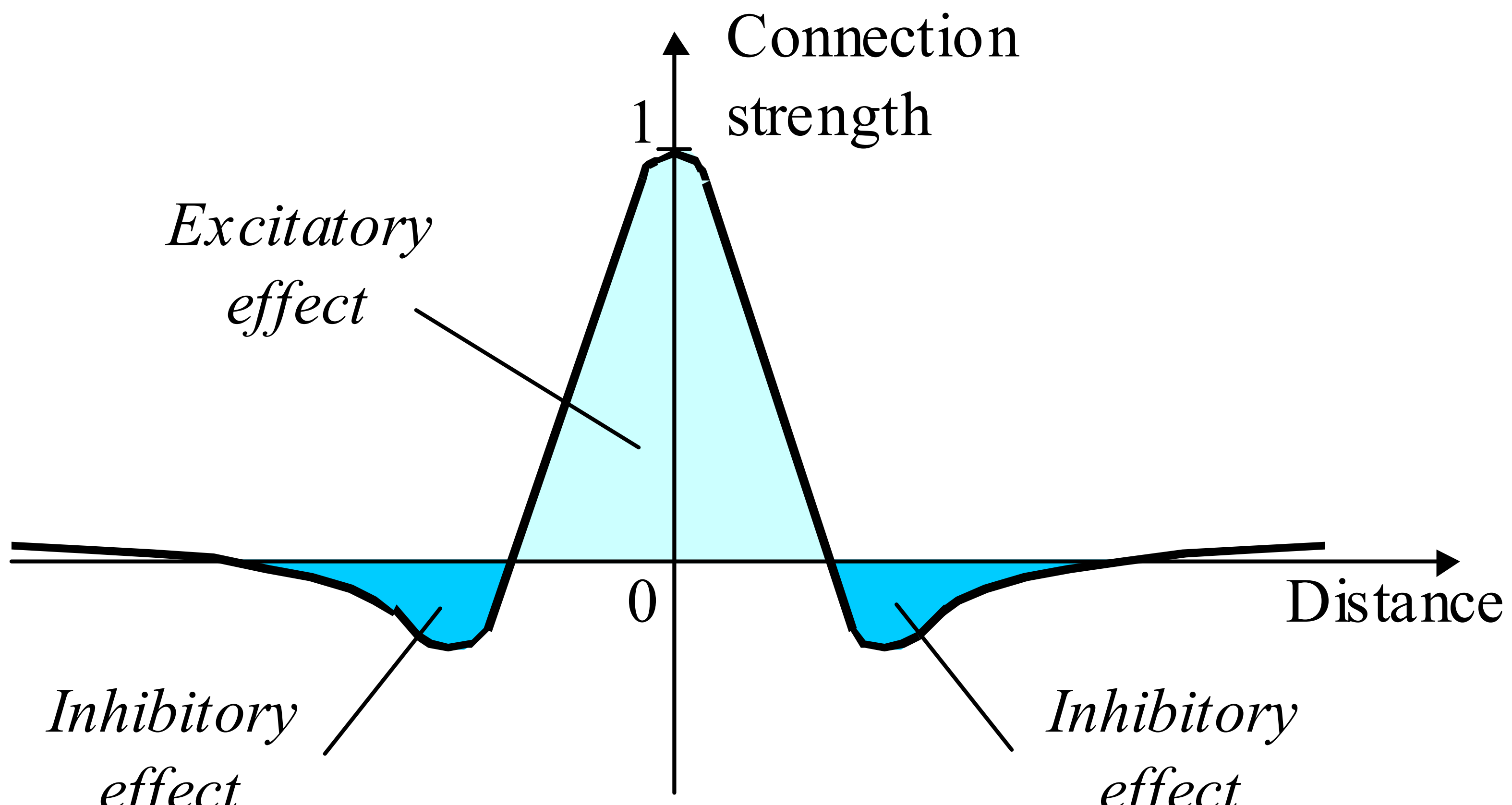
- The Kohonen model provides a topological mapping. It places a fixed number of input patterns from the input layer into a higher-dimensional output or Kohonen layer.
- Training in the Kohonen network begins with the winner's neighbourhood of a fairly large size. Then, as training proceeds, the neighbourhood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

Architecture of the Kohonen 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 Kohonen layer.

The Mexican hat function of lateral connection



- In the Kohonen network, a neuron learns by shifting its weights from inactive connections to active ones. Only the winning neuron and its neighbourhood are allowed to learn. If a neuron does not respond to a given input pattern, then learning cannot occur in that particular neuron.
- The **competitive learning rule** defines the change Δw_{ij} applied to synaptic weight w_{ij} as

$$\Delta w_{ij} = \begin{cases} \alpha\,(x_i - w_{ij}), & \text{if neuron } j \text{ wins the competitio n} \\ 0, & \text{if neuron } j \text{ loses the competitio n} \end{cases}$$

where x_i is the input signal and α is the *learning rate* parameter.

- The overall effect of the competitive learning rule resides in moving the synaptic weight vector $\mathbf{W}_j$ of the winning neuron j towards the input pattern $\mathbf{X}$. The matching criterion is equivalent to the minimum **Euclidean distance** between vectors.
- The Euclidean distance between a pair of n-by-1 vectors $\mathbf{X}$ and $\mathbf{W}_j$ is defined by

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

where x_i and w_{ij} are the ith elements of the vectors $\mathbf{X}$ and $\mathbf{W}_j$, respectively.

- To identify the winning neuron, $j_{\mathbf{X}}$, that best matches the input vector **X**, we may apply the following condition:

$$j_{\mathbf{X}} = \min_{j} \left\| \mathbf{X} - \mathbf{W}_j \right\|, \qquad j = 1, 2, \ldots, m$$

where *m* is the number of neurons in the Kohonen layer.

© Negnevitsky, Pearson Education, 2011

- Suppose, for instance, that the 2-dimensional input vector **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

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

- We find the winning (best-matching) neuron $j_{\mathbf{X}}$ using the minimum-distance Euclidean criterion:

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

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

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

- Neuron 3 is the winner and its weight vector $\mathbf{W}_3$ 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 $\mathbf{W}_3$ 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 $\mathbf{W}_3$ of the wining neuron 3 becomes closer to the input vector **X** with each iteration.

Competitive Learning Algorithm

Step 1: ***Initialisation*.**

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 α.

Step 2: *Activation and Similarity Matching.*

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

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

$j = 1, 2, \ldots, m$

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:

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

where α is the *learning rate* parameter, and $\Lambda_j(p)$ is the neighbourhood function centred around the winner-takes-all neuron $j_{\mathbf{X}}$ at iteration p.

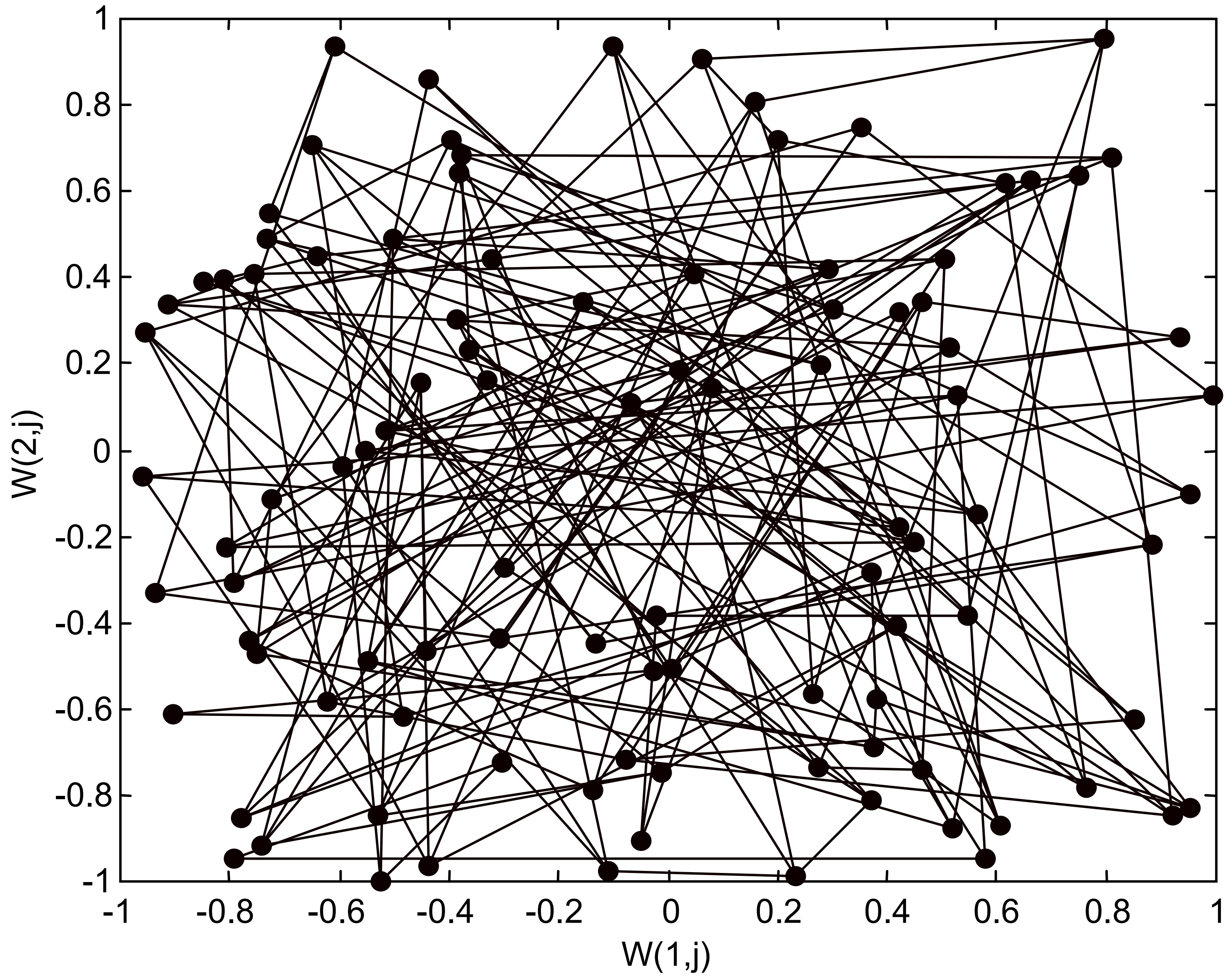
Step 4: *Iteration*.

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.

Competitive learning in the Kohonen network

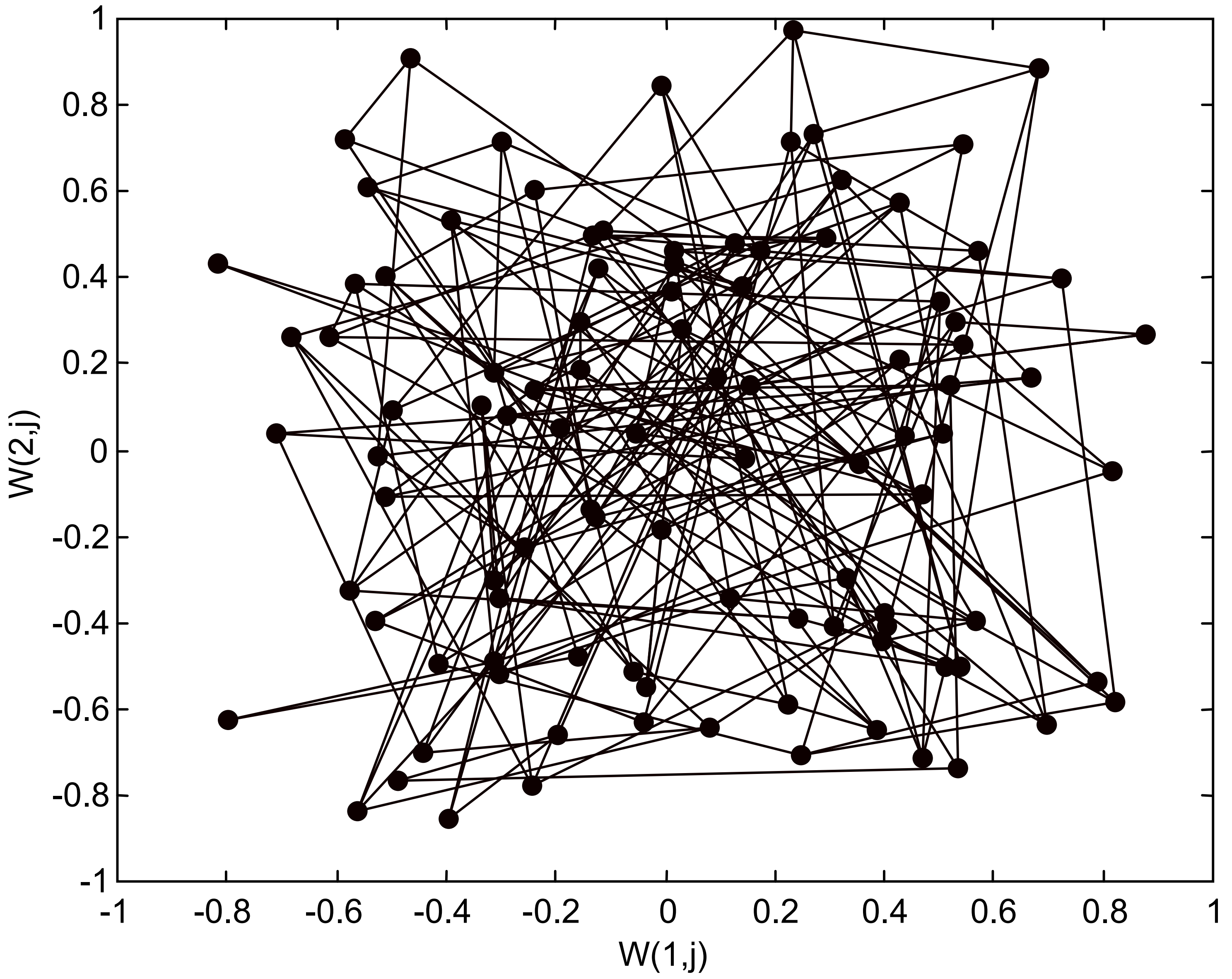
- To illustrate competitive learning, consider the Kohonen network with 100 neurons arranged in the form of a two-dimensional lattice with 10 rows and 10 columns. The network is required to classify two-dimensional input vectors – each neuron in the network should respond only to the input vectors occurring in its region.
- The network is trained with 1000 two-dimensional input vectors generated randomly in a square region in the interval between –1 and +1. The learning rate parameter α is equal to 0.1.

Initial random weights



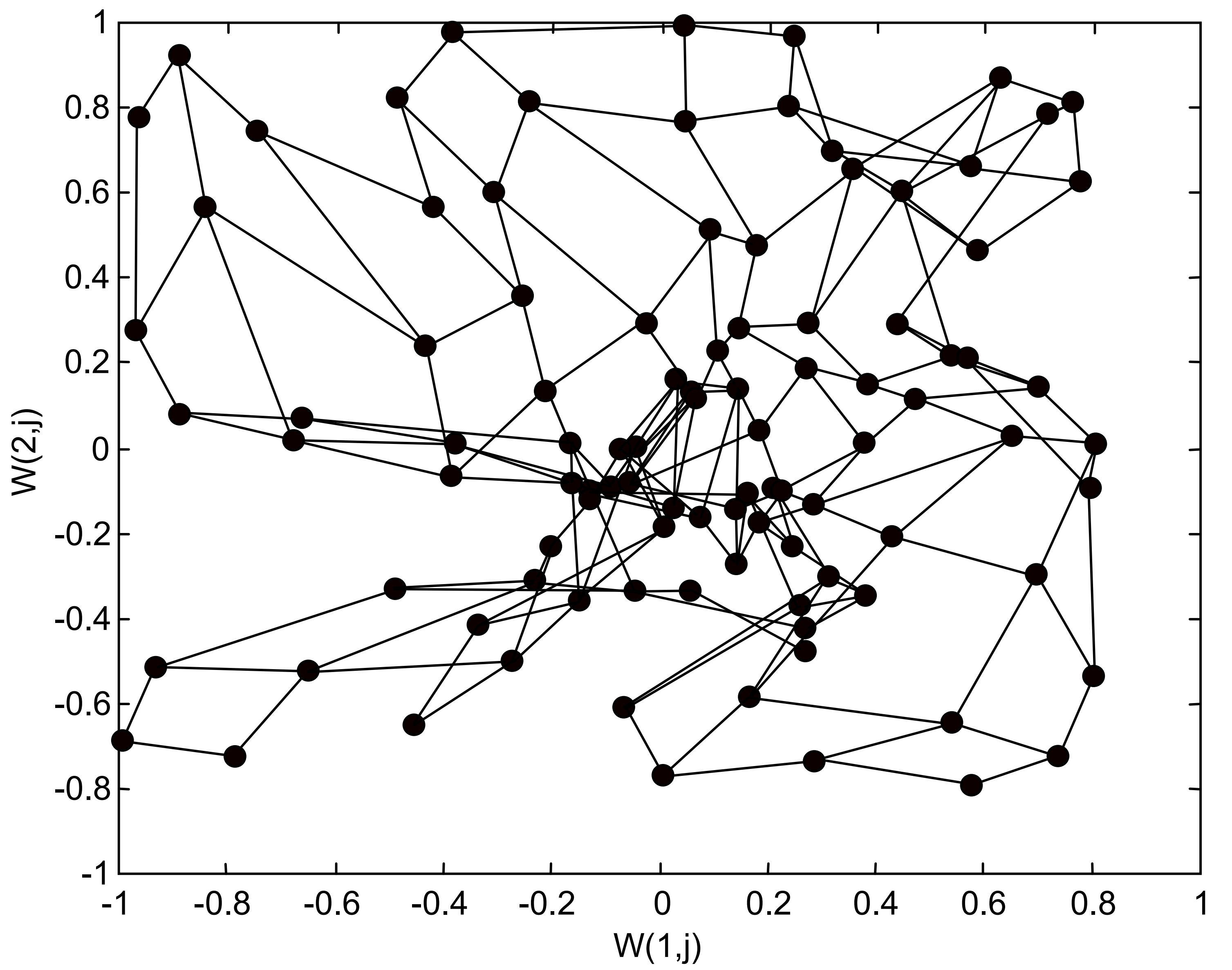
© Negnevitsky, Pearson Education, 2011

Network after 100 iterations



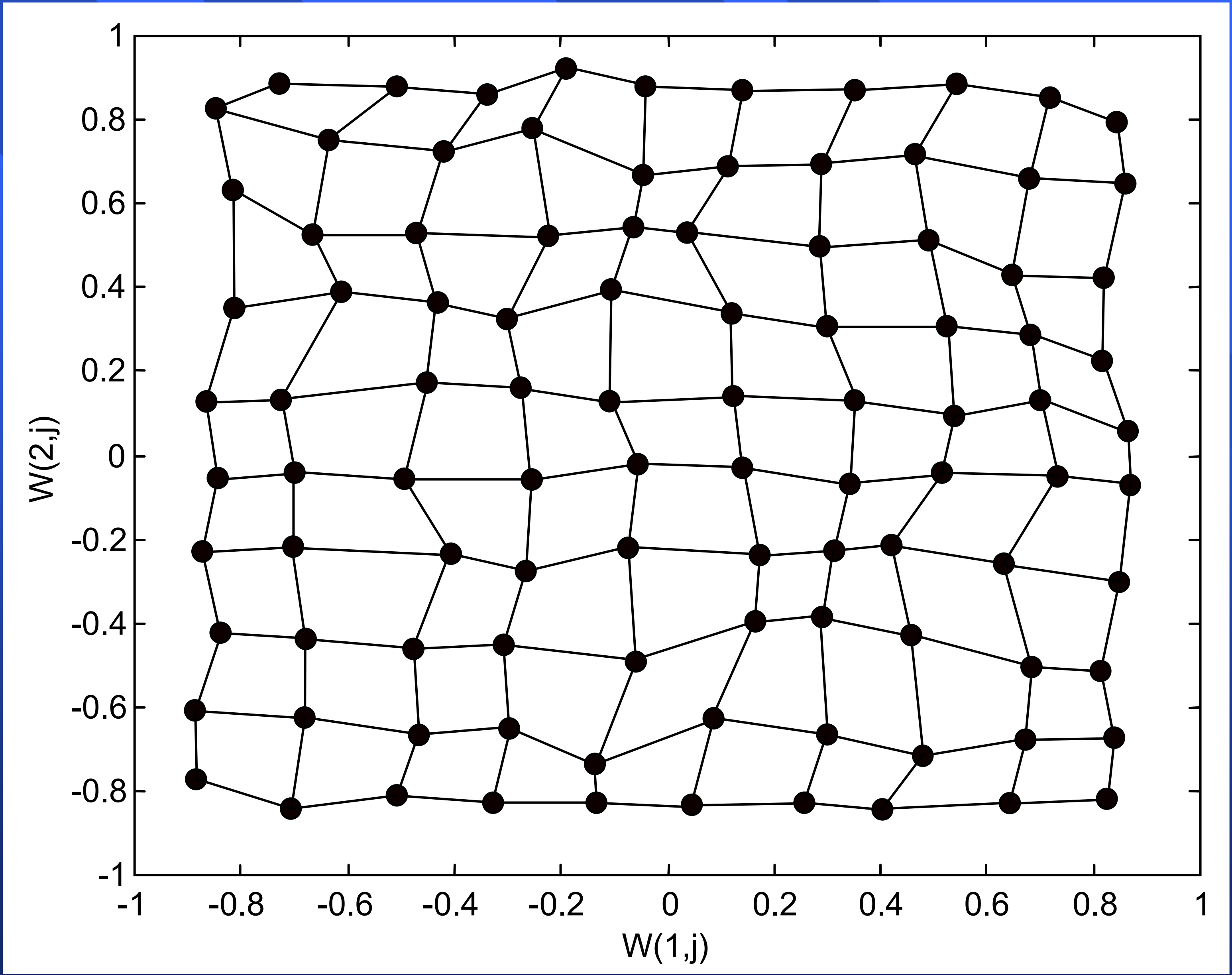
© Negnevitsky, Pearson Education, 2011

Network after 1000 iterations



© Negnevitsky, Pearson Education, 2011

Network after 10,000 iterations



Recurrent Neural networks

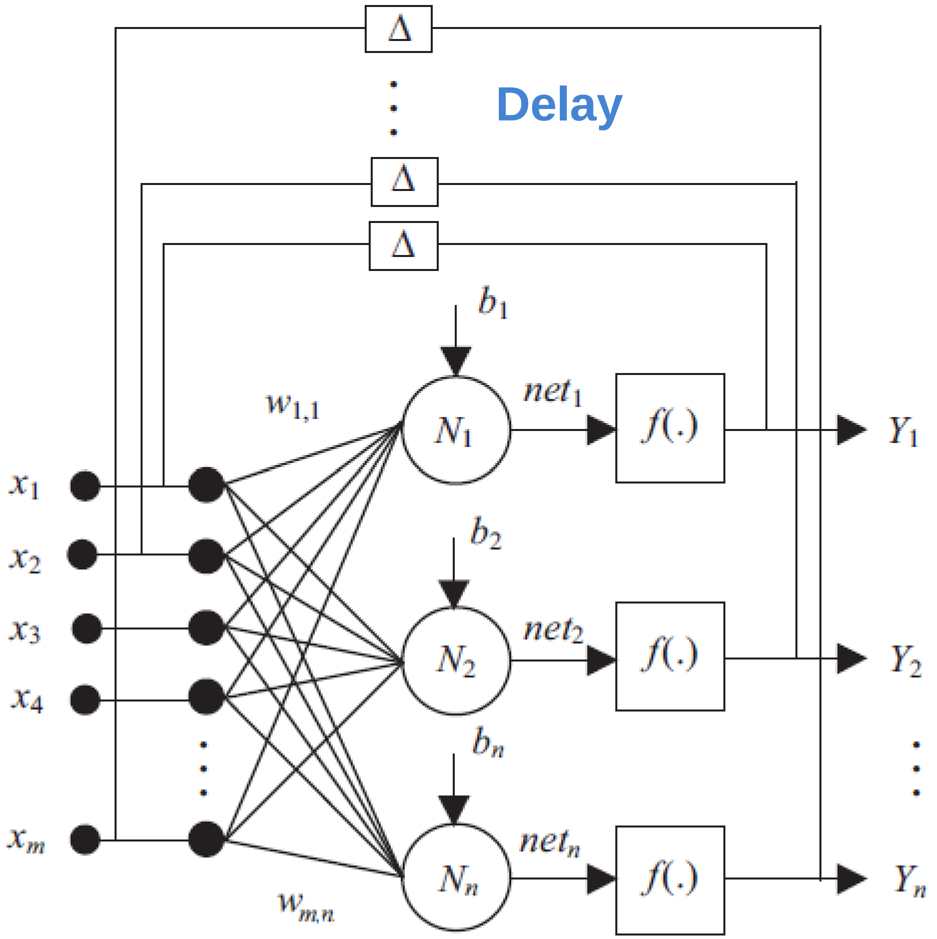
- Recurrent networks

A recurrent neural network is a type of artificial neural network commonly used in speech recognition and natural language processing. Recurrent neural networks recognize data's sequential characteristics and use patterns to predict the next likely scenario.

Recurrent Neural Networks

- A **recurrent network** is obtained from the feedforward network by **adding a connecting the neuron's output to their inputs**.

- Recurrent networks are also called **feedback networks**



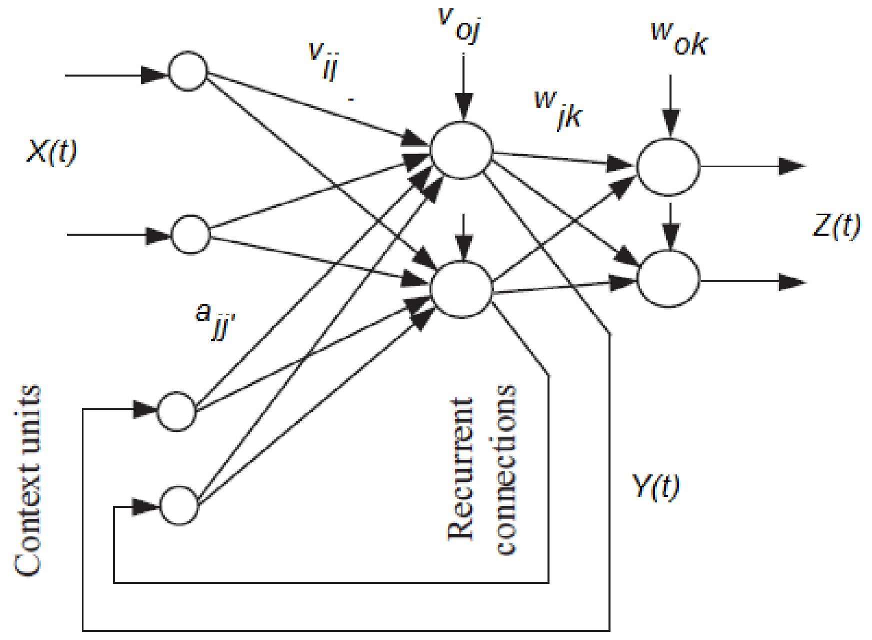
Elman Networks

- Elman networks are **three-layer backpropagation networks** with the addition of a **feedback** connection from the output of the **hidden** layer to its **input**.

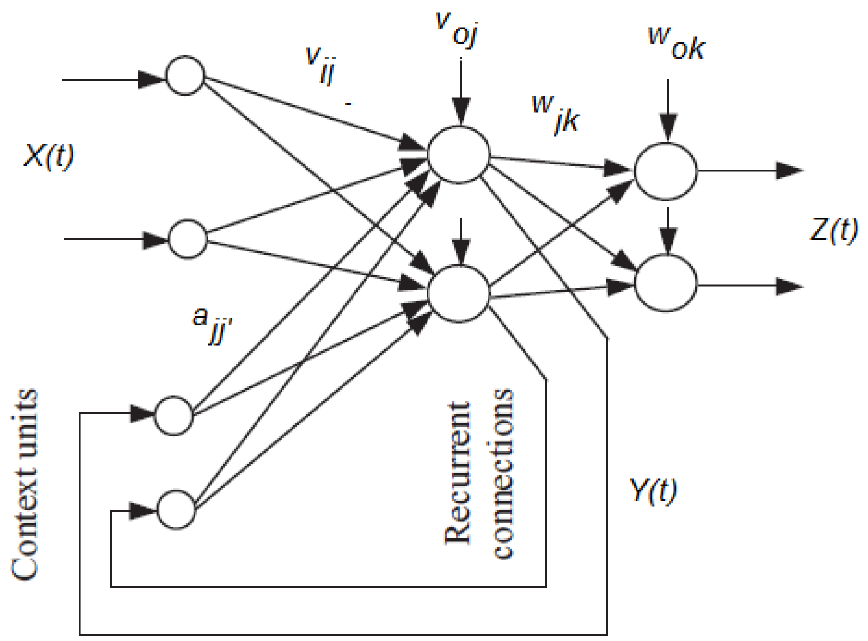
- The Elman architecture have an extra layers of neurons that copy the current activations in the hidden-layer neurons, and after **delaying** these values for **one time unit**, feed them back **as additional inputs** into the hidden layer neurons.

Elman Networks

- The Elman network will therefore have **three layers**:
  1. **Input layer** that consists of two different groups of neurons: **external inputs** and **internal inputs**
  2. **Hidden layer**
  3. **Output layer**



Elman Network Equations



$$Z = f(W^T Y + W_o)$$

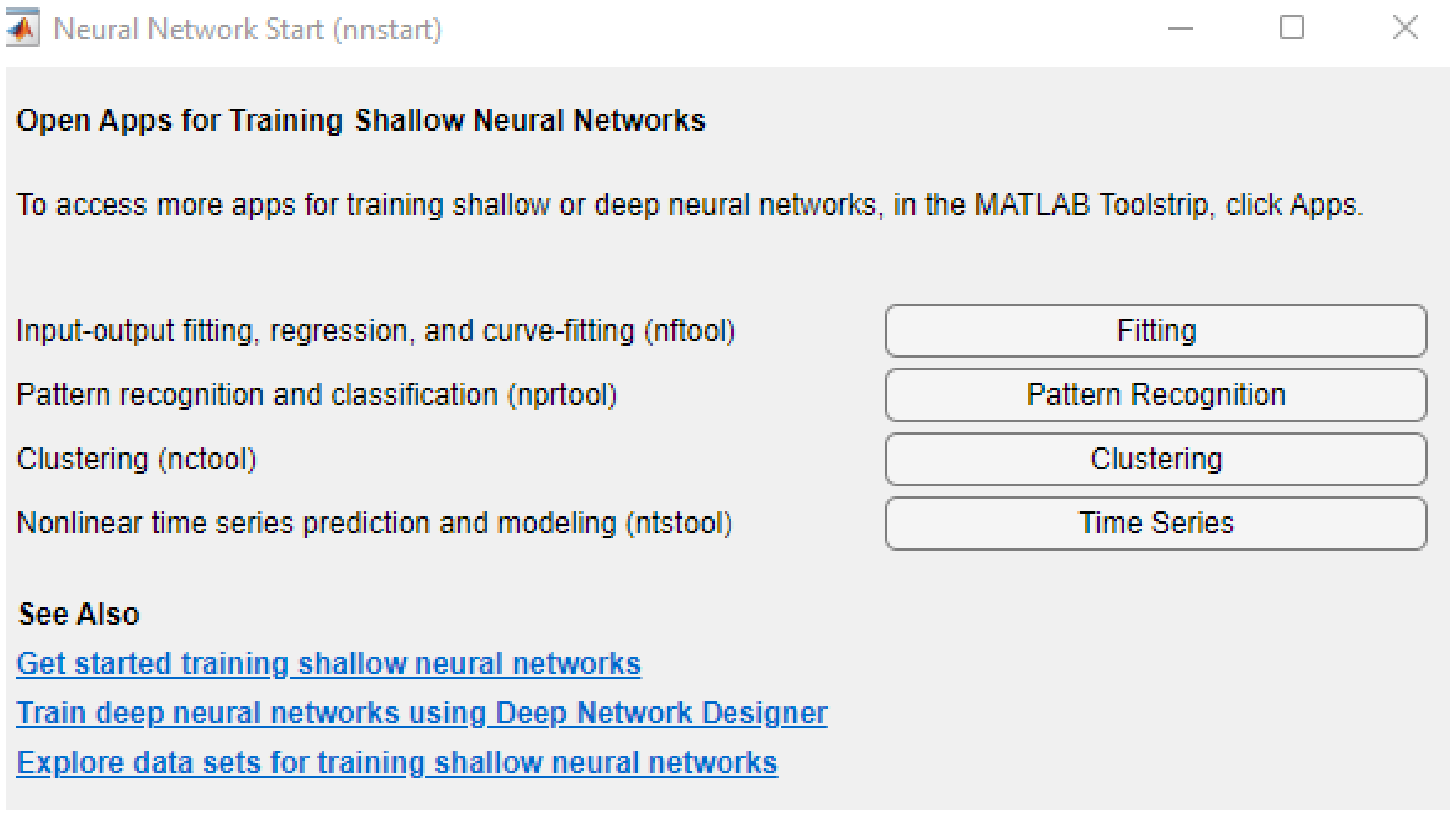
$$Y = f(V^T X + V_o + A^T Y)$$

Y is a function of Y ??

But there is a delay! And that is why we use '**t**' to represent the time sequence

$$Y(t) = f(V^T X + V_o + A^T Y(t-1))$$

nnstart : Time series



Neural Network Start (nnstart)

Open Apps for Training Shallow Neural Networks

To access more apps for training shallow or deep neural networks, in the MATLAB Toolstrip, click Apps.

Input-output fitting, regression, and curve-fitting (nftool)	Fitting
Pattern recognition and classification (nprtool)	Pattern Recognition
Clustering (nctool)	Clustering
Nonlinear time series prediction and modeling (ntstool)	Time Series

See Also

Get started training shallow neural networks

Train deep neural networks using Deep Network Designer

Explore data sets for training shallow neural networks

Time Series Networks

You can train a neural network to solve three types of time series problems.

NARX Network

In the first type of time series problem, you would like to predict future values of a time series $y(t)$ from past values of that time series and past values of a second time series $x(t)$. This form of prediction is called nonlinear autoregressive with exogenous (external) input, or NARX (see Design Time Series NARX Feedback Neural Networks), and can be writter as follows:

$$y(t) = f(y(t-1), ..., y(t-d), x(t-1), ..., (t-d))$$

Use this model to predict future values of a stock or bond, based on such economic variables as unemployment rates, GDP, etc. You can also use this model for system identification, in which models are developed to represent dynamic systems, such as chemical processes, manufacturing systems, robotics, aerospace vehicles, etc.

NAR Network

In the second type of time series problem, there is only one series involved. The future values of a time series $y(t)$ are predicted only from past values of that series. This form of prediction is called nonlinear autoregressive, or NAR, and can be written as follows:

$$y(t) = f(y(t-1), ..., y(t-d))$$

You can use this model to predict financial instruments, but without the use of a companion series.

Nonlinear Input-Output Network

The third time series problem is similar to the first type, in that two series are involved, an input series $x(t)$ and an output series $y(t)$. Here you want to predict values of $y(t)$ from previous values of $x(t)$, but without knowledge of previous values of $y(t)$. This input/output model can be written as follows:

$$y(t) = f(x(t-1), ..., x(t-d))$$

Time Series Networks

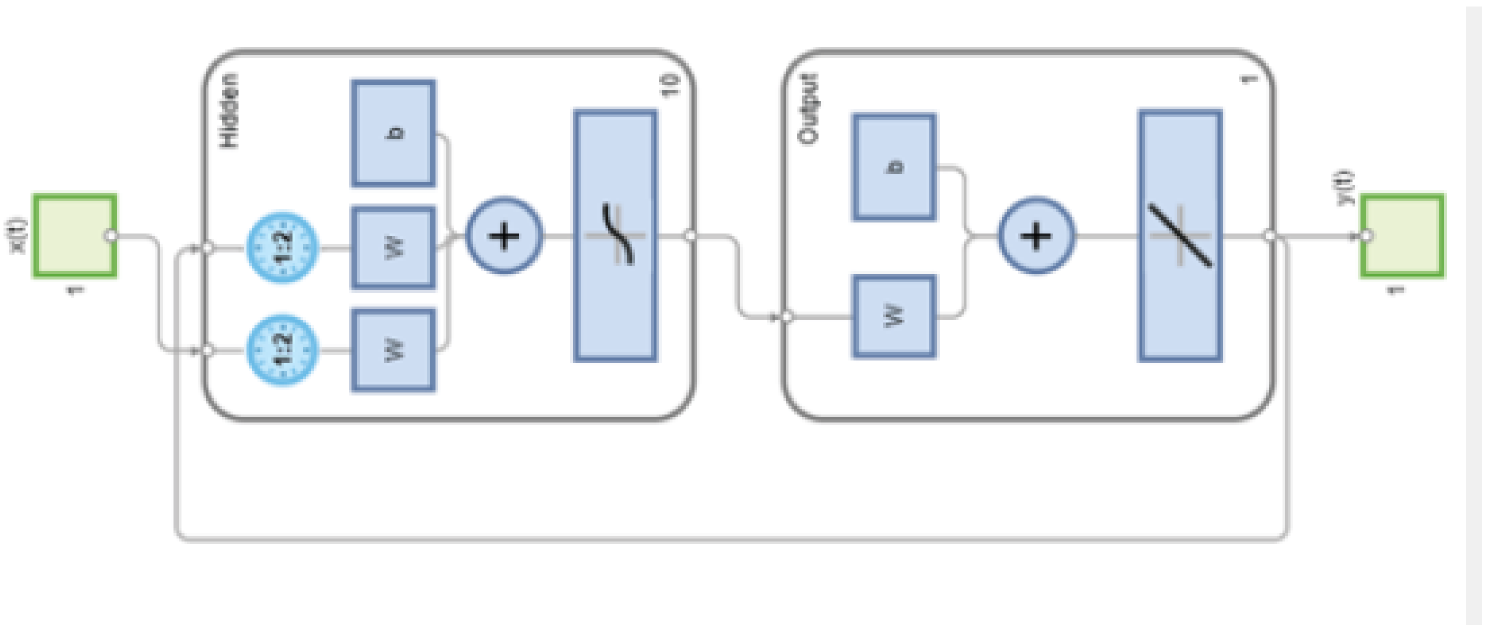
You can train a neural network to solve three types of time series problems.

NARX Network

In the first type of time series problem, you would like to predict future values of a time series $y(t)$ from past values of that time series and past values of a second time series $x(t)$. This form of prediction is called nonlinear autoregressive with exogenous (external) input, or NARX (see Design Time Series NARX Feedback Neural Networks), and can be writter as follows:

$$y(t) = f(y(t-1), \dots, y(t-d), x(t-1), \dots, (t-d))$$

Use this model to predict future values of a stock or bond, based on such economic variables as unemployment rates, GDP, etc. You can also use this model for system identification, in which models are developed to represent dynamic systems, such as chemical processes, manufacturing systems, robotics, aerospace vehicles, etc.

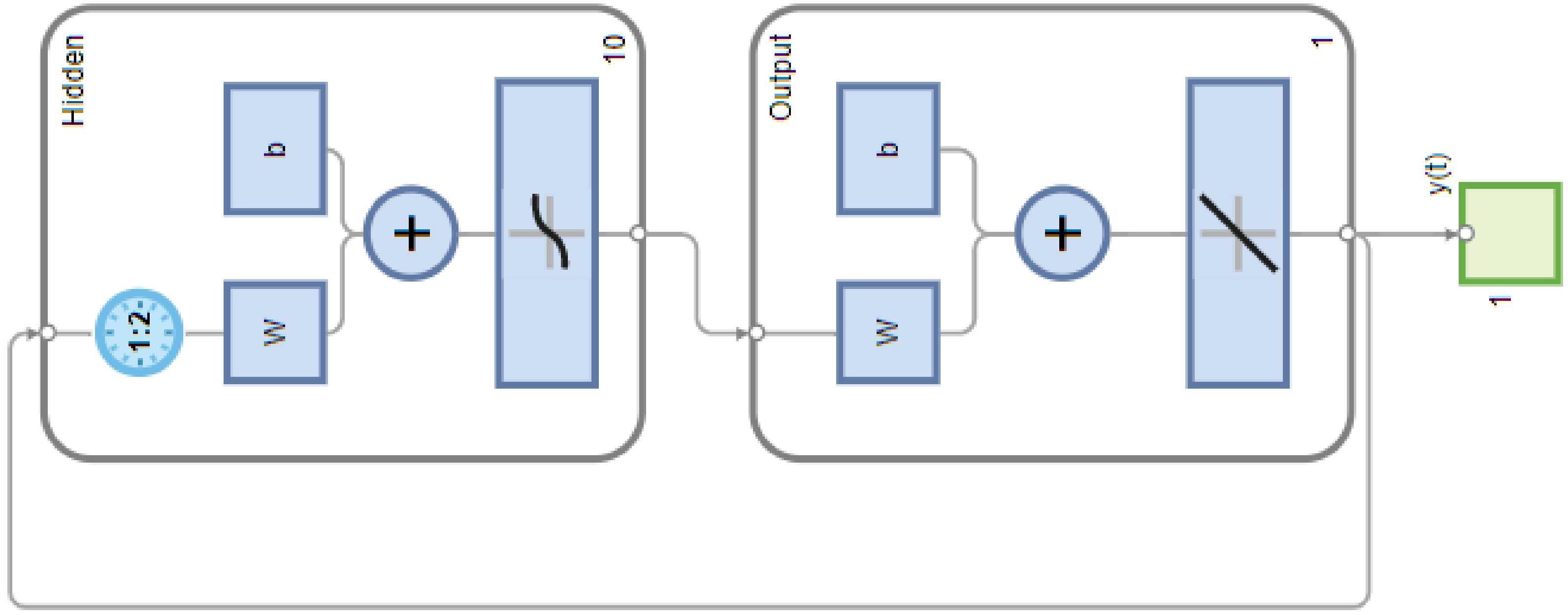


NAR Network

In the second type of time series problem, there is only one series involved. The future values of a time series $y(t)$ are predicted only from past values of that series. This form of prediction is called nonlinear autoregressive, or NAR, and can be written as follows:

$$y(t) = f(y(t-1), ..., y(t-d))$$

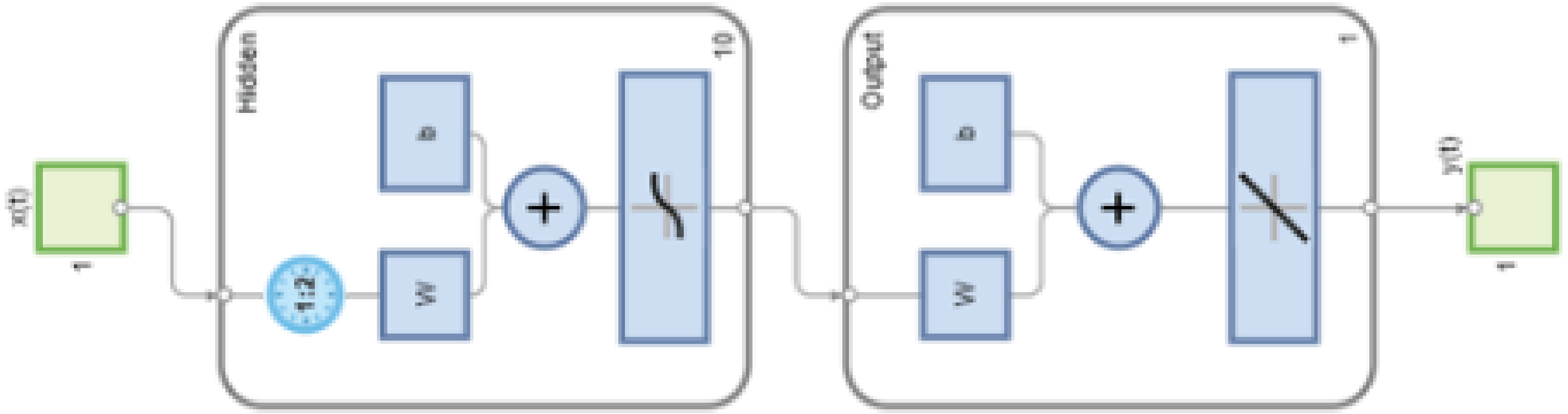
You can use this model to predict financial instruments, but without the use of a companion series.



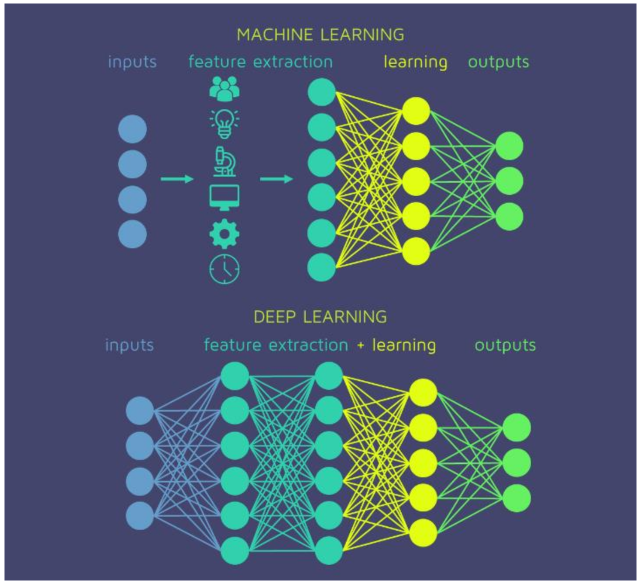
Nonlinear Input-Output Network

The third time series problem is similar to the first type, in that two series are involved, an input series $x(t)$ and an output series $y(t)$. Here you want to predict values of $y(t)$ from previous values of $x(t)$, but without knowledge of previous values of $y(t)$. This input/output model can be written as follows:

$$y(t) = f(x(t-1), \ldots, x(t-d))$$



Machine Intelligence
Introduction to Deep Learning



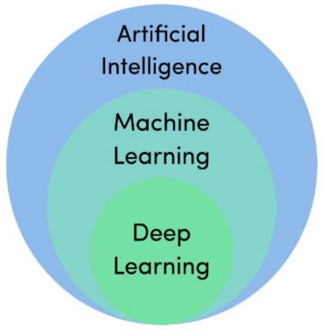
Slides from:

- Dr. Ahmad Mahasneh
- Introduction to Deep Learning. Professor Qiang Yang http://www.cs.ust.hk/~qyang/
- Deep Learning Tutorial Slides PPT - BT Tepper bt.tepper.cmu.edu

Outline

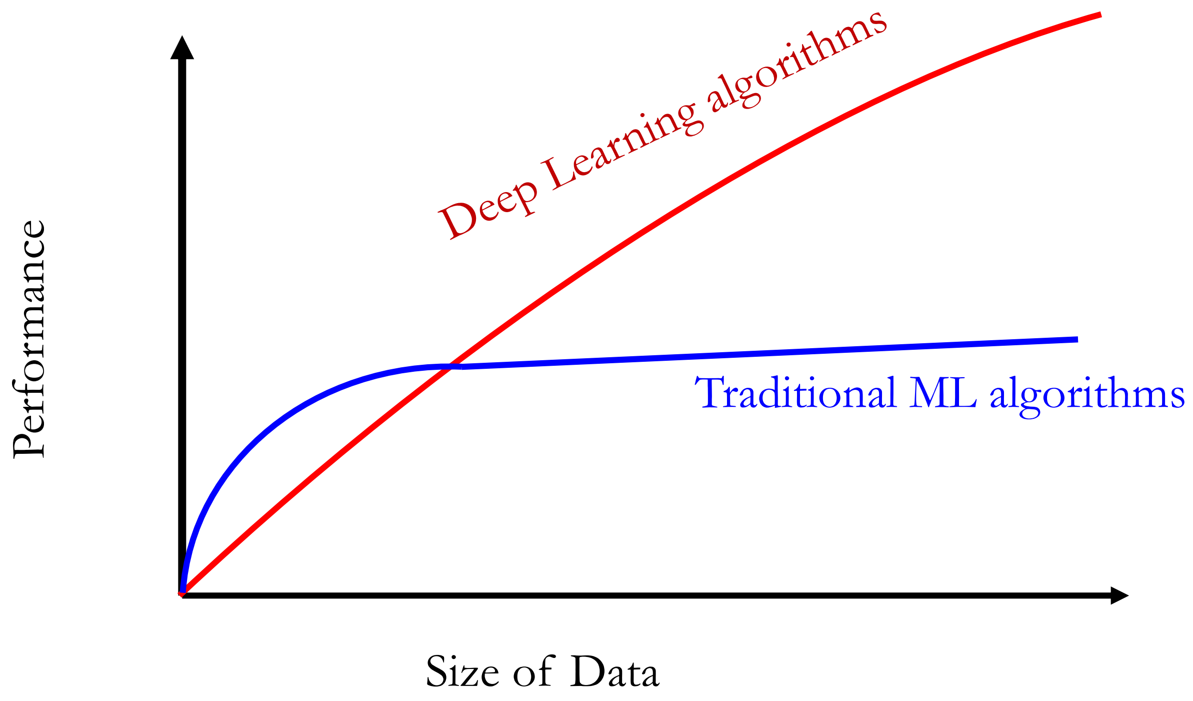
- Introduction
- Convolutional Neural Network
- Sequence Modelling: RNN and its extensions

What is Deep Learning?



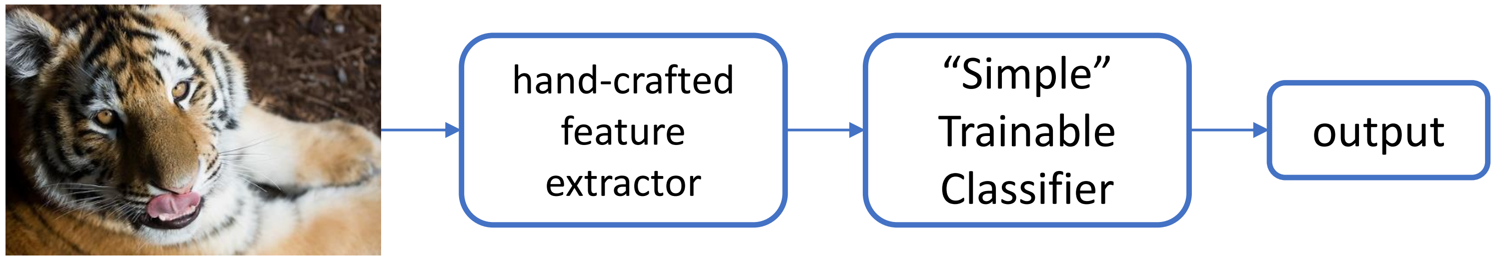
- Deep learning is a recent buzz word for neural networks.
- It describes **machine learning** framework that shows impressive performance on many **Artificial Intelligence** tasks.
- While increasing the hidden layers in the shallow networks does not improve its performance (sometimes it will degrade the performance), the increased number of layers in deep learning **allows for automatically learning the features in the input data (image).**
- Machine learning uses algorithms developed for specific tasks. Deep learning is more of a **data representation** based upon multiple layers of a matrix, where each layer uses output from the previous layer as input.

Deep learning Vs. traditional ML



Introduction

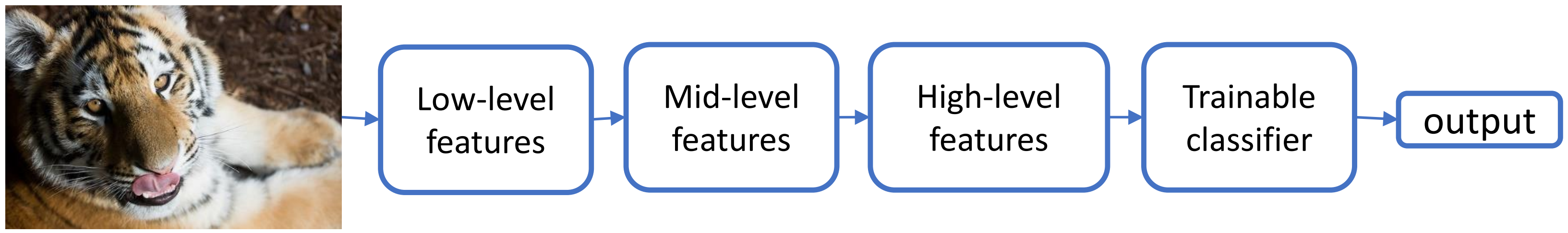
- Traditional pattern recognition models use hand-crafted features and relatively simple trainable classifier.



- This approach has the following limitations:
  - It is very tedious and costly to develop hand-crafted features
  - **The hand-crafted features are usually highly dependents on one application, and cannot be transferred easily to other applications**

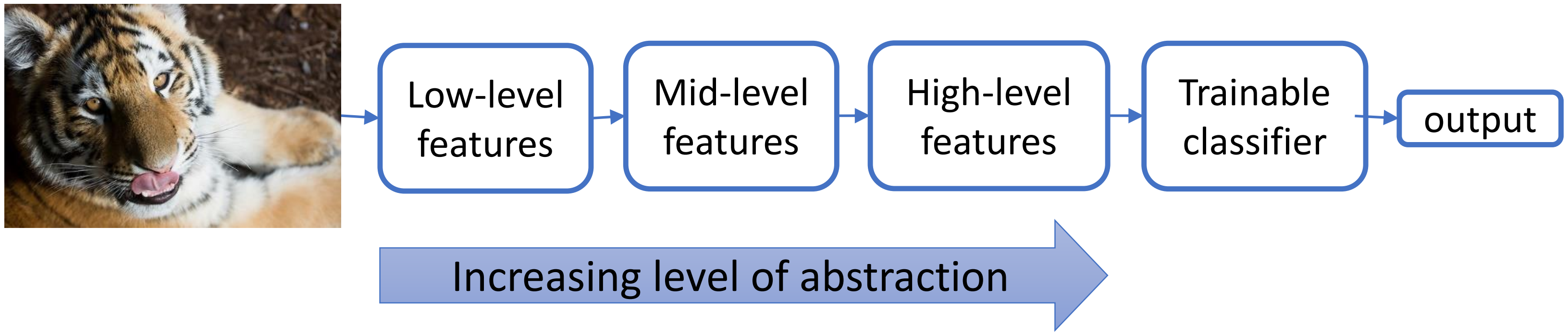
Deep Learning

- Deep learning (representation learning) seeks to learn rich hierarchical representations (i.e. features) automatically through multiple stage of feature learning process.



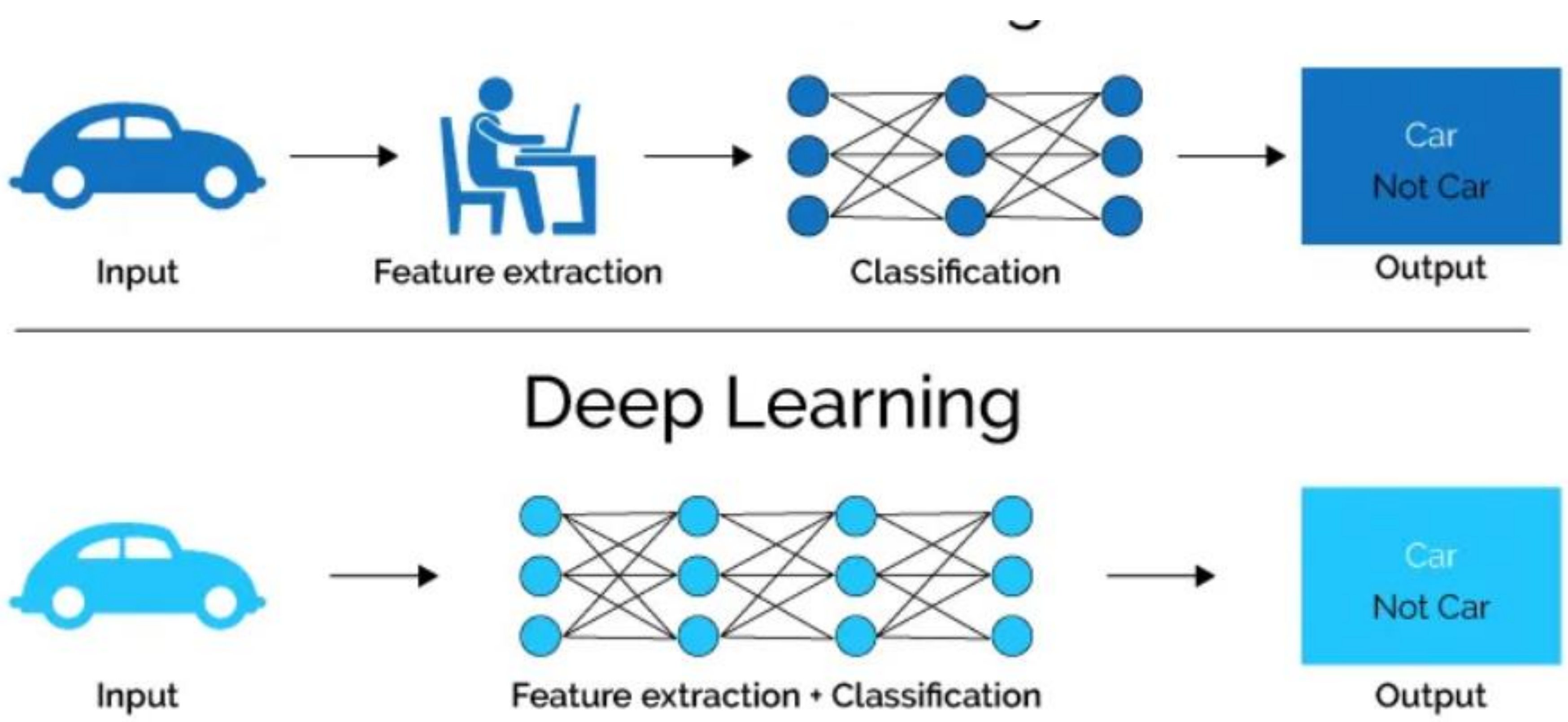
Feature visualization of convolutional net trained on ImageNet
(Zeiler and Fergus, 2013)

Learning Hierarchical Representations

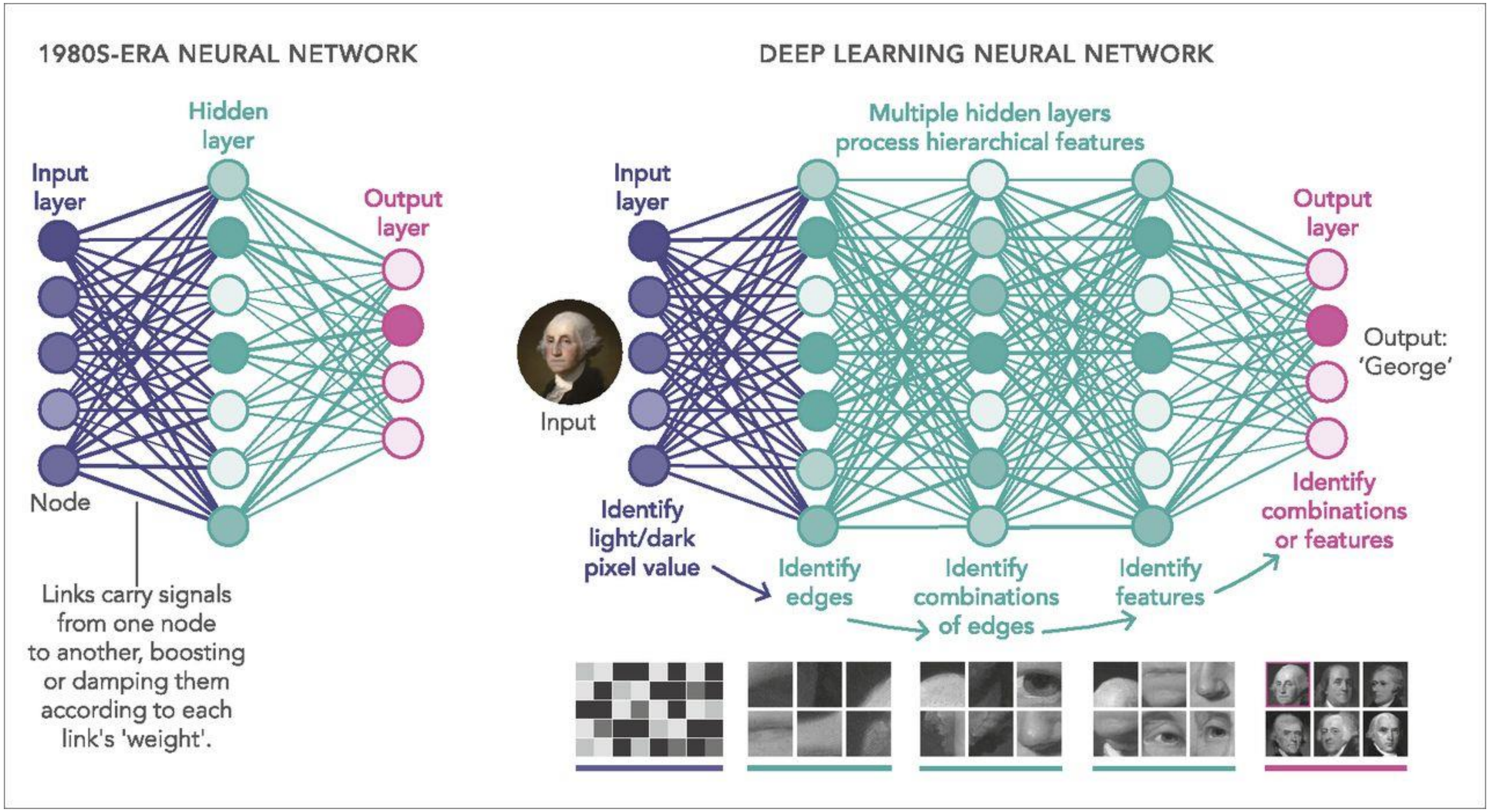


- Hierarchy of representations with increasing level of abstraction. Each stage is a kind of trainable nonlinear feature transform
- Image recognition
  - Pixel → edge → motif → part → object
- Text
  - Character → word → word group → clause → sentence → story

Shallow learning



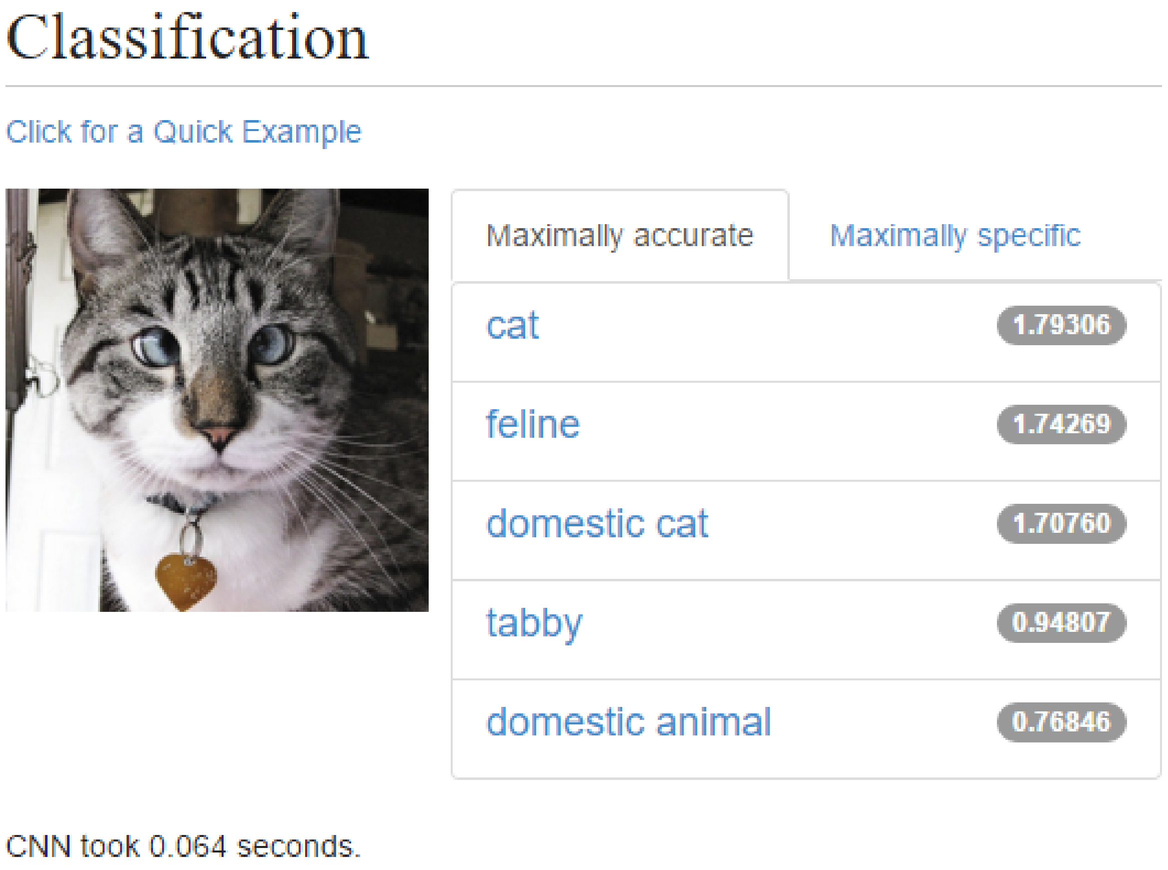
Machine learning Vs. Deep learning


1980S-ERA NEURAL NETWORK
Hidden layer
Input layer
Output layer
Node
Links carry signals from one node to another, boosting or damping them according to each link's 'weight'.
DEEP LEARNING NEURAL NETWORK
Multiple hidden layers process hierarchical features
Input layer
Output layer
Input
Output: 'George'
Identify light/dark pixel value
Identify edges
Identify combinations of edges
Identify features
Identify combinations or features

Supervised Learning

- Convolutional Neural Network
- Sequence Modelling
  - Why do we need RNN?
  - What are RNNs?
  - RNN Extensions
  - What can RNNs can do?

Deep CNN for Image Classification



Classification

Click for a Quick Example

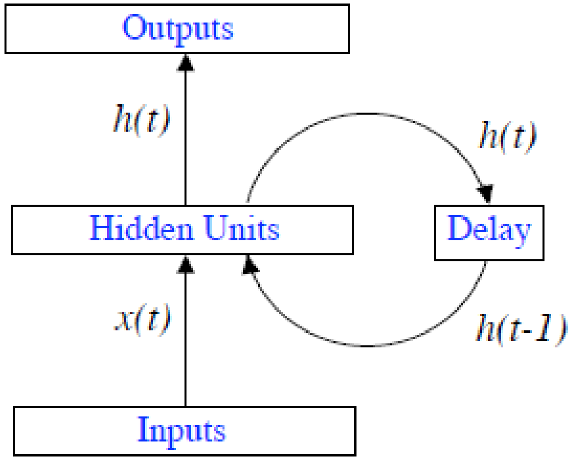
Maximally accurate	Maximally specific
cat	1.79306
feline	1.74269
domestic cat	1.70760
tabby	0.94807
domestic animal	0.76846

CNN took 0.064 seconds.

Try out a live demo at
http://demo.caffe.berkeleyvision.org/

What are RNNs?

Recurrent neural networks (RNNs) are connectionist models with the ability to selectively pass information across sequence steps, while processing sequential data one element at a time.



The simplest form of ***fully recurrent neural network*** is an MLP with the previous set of hidden unit activations feeding back into the network along with the inputs

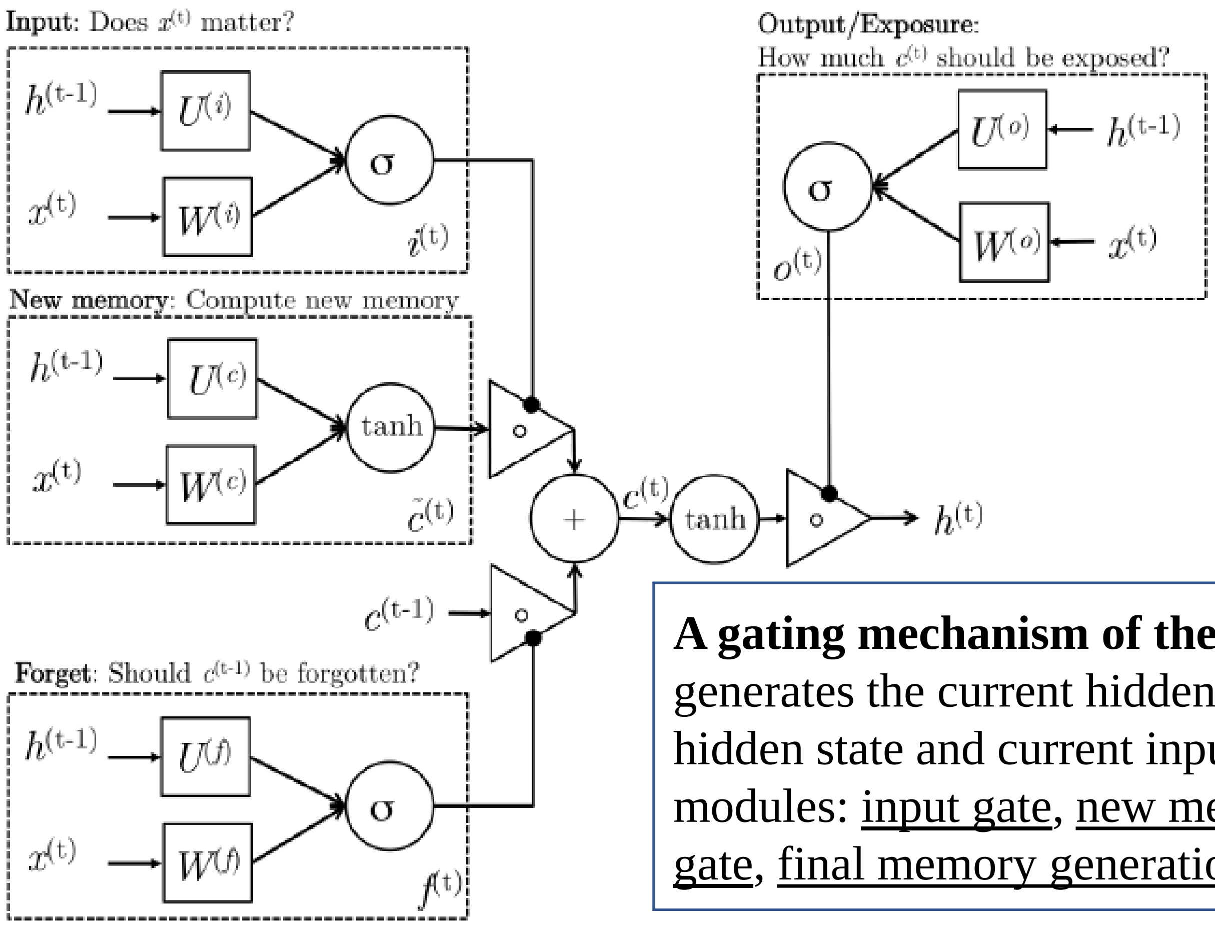
Allow a 'memory' of previous inputs to persist in the network's internal state, and thereby influence the network output

$$h(t) = f_H(W_{IH}x(t) + W_{HH}h(t-1))$$
$$y(t) = f_O(W_{HO}h(t))$$

f_H and f_O are the activation function for hidden and output unit; W_{IH}, W_{HH}, and W_{HO} are connection weight matrices which are learnt by training

RNN Extensions: Long Short-term Memory

LSTMs (Hochreiter and Schmidhuber, 1997) were designed to combat vanishing gradients through a *gating* mechanism.



A gating mechanism of the LSTM , which generates the current hidden state by the past hidden state and current input **..**It contains five modules: <u>input gate</u>, <u>new memory cell</u>, <u>forget gate</u>, <u>final memory generation</u>, and <u>output gate</u>.

What can RNNs can do?

Machine Translation

Visual Question Answering

Ch2 Fuzzy expert systems:

Fuzzy logic

■Introduction, or what is fuzzy thinking?

■Fuzzy sets and Linguistic variables

■ fuzzy sets and Fuzzy rules

■Fuzzy inference systems:
 1-Mamdani Fuzzy Logic
 2-Sugeno Fuzzy Logic

Introduction, or what is fuzzy thinking?

- ❑ Fuzzy logic reflects how people think. It attempts to model our sense of words, our decision making and our common sense. As a result, it is leading to new, more human, intelligent systems.

- ❑ How can we represent expert knowledge that uses vague and ambiguous terms in a computer?

- Boolean logic uses sharp distinctions. It forces us to draw lines between members of a class and non- members. For instance, we may say, Tom is tall because his height is 181 cm.If we drew a line at 180 cm, we would find that David, who is 179 cm, is small. Is David really a small man or we have just drawn an arbitrary line in the sand?

- Fuzzy logic is based on the idea that all things admit of degrees. Temperature, height, speed, distance- all come on a sliding scale.

- In 1965 Lotfi Zadeh, published his famous paper "Fuzzy set Zadeh extended the work on possibility theory into a formal system of mathematical logic, and introduced a new concept for applying natural language terms.

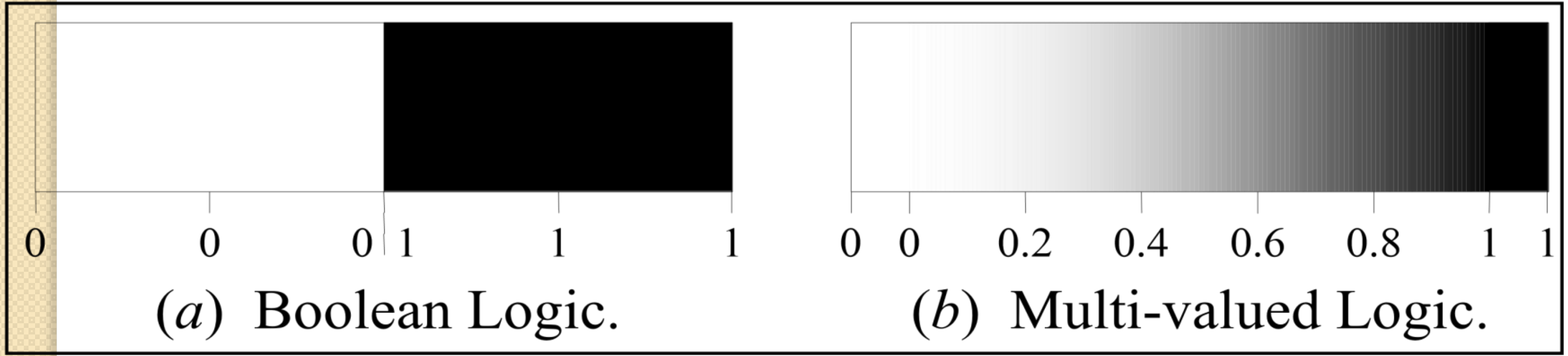
This new logic for representing and manipulating fuzzy terms was called fuzzy logic, and Zadeh became the Master of fuzzy logic.

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

- Unlike two-valued Boolean logic, fuzzy logic is multi-valued. It deals with degrees of membership and degrees of truth.

- Fuzzy logic uses the continuum of logical values between 0 (completely false) and 1 (completely true).Instead of just black and white, it employs the spectrum of colors, accepting that things can be partly true and partly false at the same time.

Range of logical values in Boolean and fuzzy logic

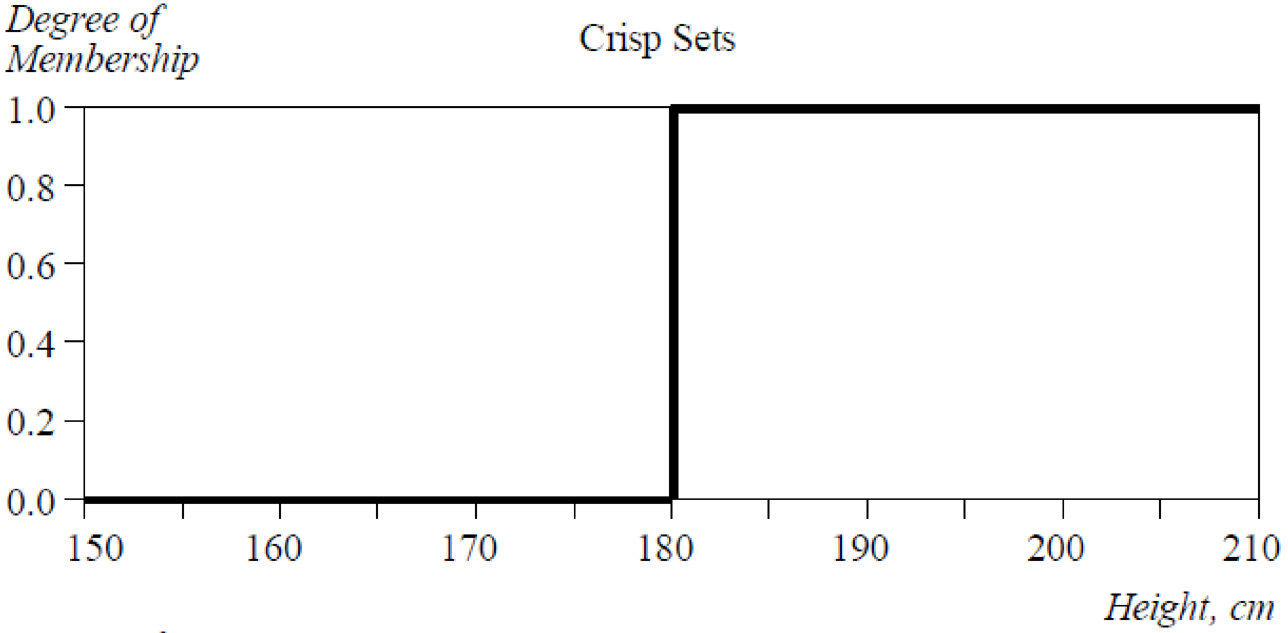


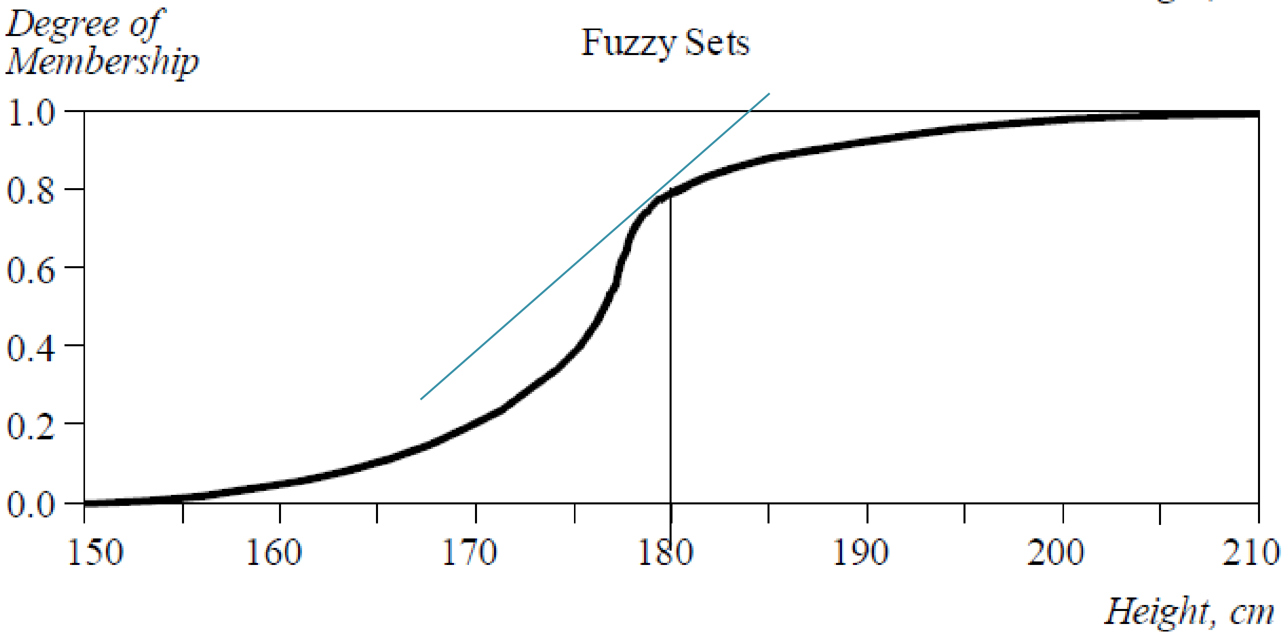
Negnevitsky, Pearson Education, 2011

Fuzzy sets

- The classical example in fuzzy sets is tall men. The elements of the fuzzy set "tall men" are all men, but their degrees of membership depend on their height.

Name	Height, cm	Degree of Membership	
		Crisp	*Fuzzy*
Chris	208	1	1.00
Mark	205	1	1.00
John	198	1	0.98
Tom	181	1	0.82
David	179	0	0.78
Mike	172	0	0.24
Bob	167	0	0.15
Steven	158	0	0.06
Bill	155	0	0.01
Peter	152	0	0.00





Crisp and fuzzy sets of “tall men”

- The x-axis represents the universe of discourse - the range of all possible values applicable to a chosen variable. In our case, the variable is the man height.

- The y-axis represents the membership value of the fuzzy set. In our case, the fuzzy set of "tall men" maps height values into corresponding membership values.

A fuzzy set is a set with fuzzy boundaries.

■ Let X be the universe of discourse and its elements be denoted as x. In the classical set theory, crisp set A of X is defined as function $F_A(x)$ called the characteristic function of A

$$F_A(x): X \rightarrow \{0,1\}, \text{where } F_X(x) = \begin{cases} 1, \text{if } x \in A \\ 0, \text{if } x \notin A \end{cases}$$

This set maps universe X to a set of two elements. For any element x of universe X, characteristic function $F_A(x)$ is equal to 1 if x is an element of set A, and is equal to 0 if x is not an element of A.

In the fuzzy theory, fuzzy set A of universe X is defined by function $\mu_A(x)$: called the membership function of set A .

$\mu_A(x) : X \rightarrow [0, 1]$, where $\mu_A(x) = 1$ if x is totally in A;
$\mu_A(x) = 0$ if x is not in A;
$0 < \mu_A(x) < 1$ if x is partly in A.

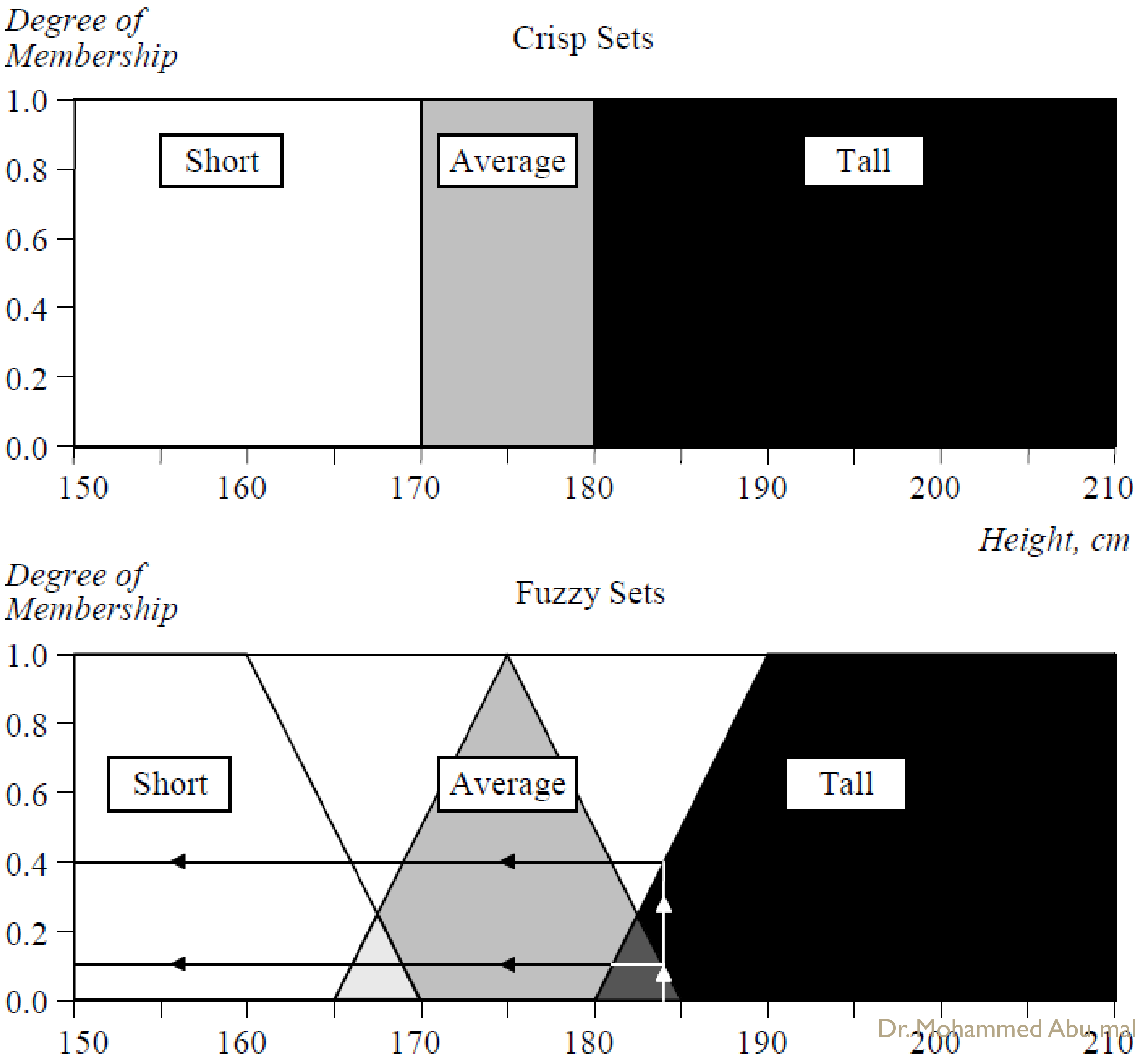
This set allows a continuum of possible choices. For any element x of universe X, membership function $\mu A(x)$ equals the degree to which x is an element of fuzzy set A. This degree, a value between 0 and 1, represents the degree of membership, also called membership value, of element x in set A.

How to represent a fuzzy set in a computer ?

■ First, we determine the membership functions. In our "tall men" example, we can obtain fuzzy sets of tall, short and average men.

■ The universe of discourse - the men's heights - consists of three sets: short, average and tall men. As you will see, a man who is 184 cm tall is a member of the average men set with a degree of membership of 0.1, and at the same time, he is also a member of the tall men set with a degree of 0.4.

Crisp and fuzzy sets of short, average and tall men



Linguistic variables

- At the root of fuzzy set theory lies the idea of linguistic variables.

- A linguistic variable is a fuzzy variable. For example, the statement "John is tall" implies that the linguistic variable John takes the linguistic value tall.

In fuzzy expert systems, linguistic variables are used in fuzzy rules. For example:

IF	wind is strong
THEN	sailing is good
IF	project duration is long
THEN	completion risk is high
IF	speed is slow
THEN	stopping distance is short

What is a fuzzy rule?

- A fuzzy rule can be defined as a conditional statement in 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 universe of discourses X and Y, respectively.

What is the difference between classical and fuzzy rules?

A classical IF-THEN rule uses binary logic, for example,

Rule: 1
IF speed > 100
THEN stopping distance is long

Rule: 2
IF speed < 40
THEN stopping distance is short

The variable speed can have any numerical value between 0 and 220 km/h, but the linguistic variable stopping_distance can take either value long or short. In other words, classical rules are expressed in the black-and-white language of Boolean logic.

We can also represent the stopping distance rules in a fuzzy form:

Rule: 1
IF speed is fast

THEN stopping distance is long

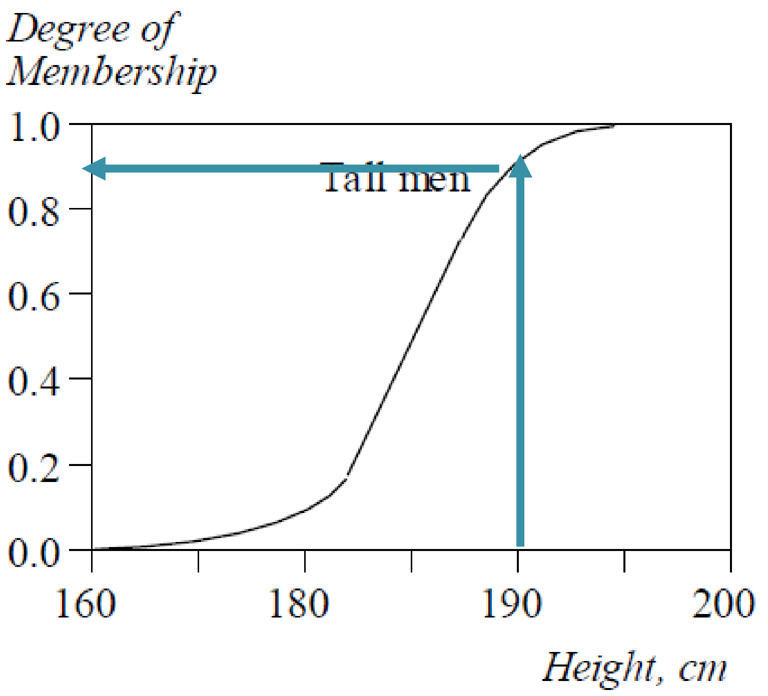
Rule: 2
IF speed is slow

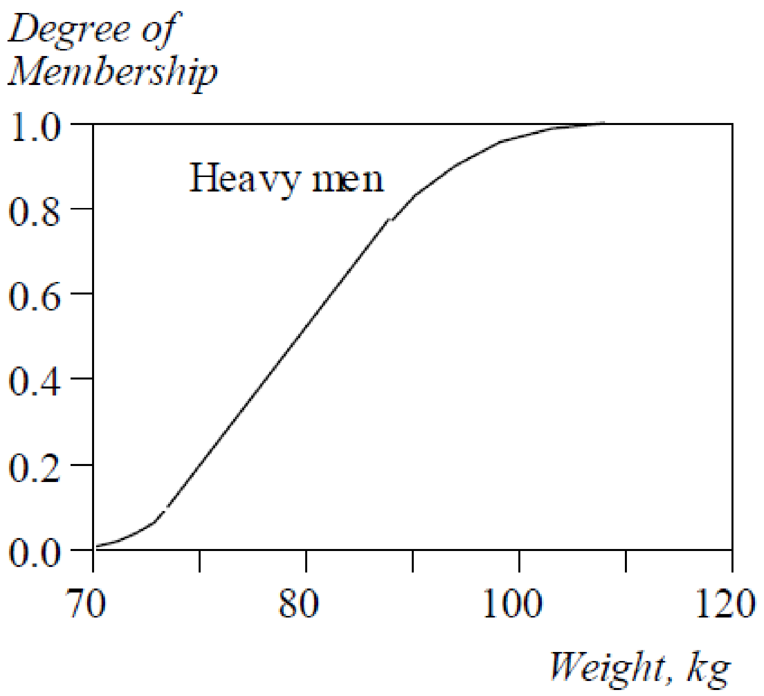
THEN stopping distance is short

- In fuzzy rules, the linguistic variable speed also has the range (the universe of discourse) between 0 and 220 km/h, but this range includes fuzzy sets, such as slow, medium and fast. The universe of discourse of the linguistic variable stopping_distance can be between 0 and 300 m and may include such fuzzy sets as short, medium and long.

- Fuzzy rules relate fuzzy sets.
- In a fuzzy system, all rules fire to some extent, or in other words they fire partially. If the antecedent is true to some degree of membership, then the consequent is also true to that same degree.

Fuzzy sets of tall and heavy men

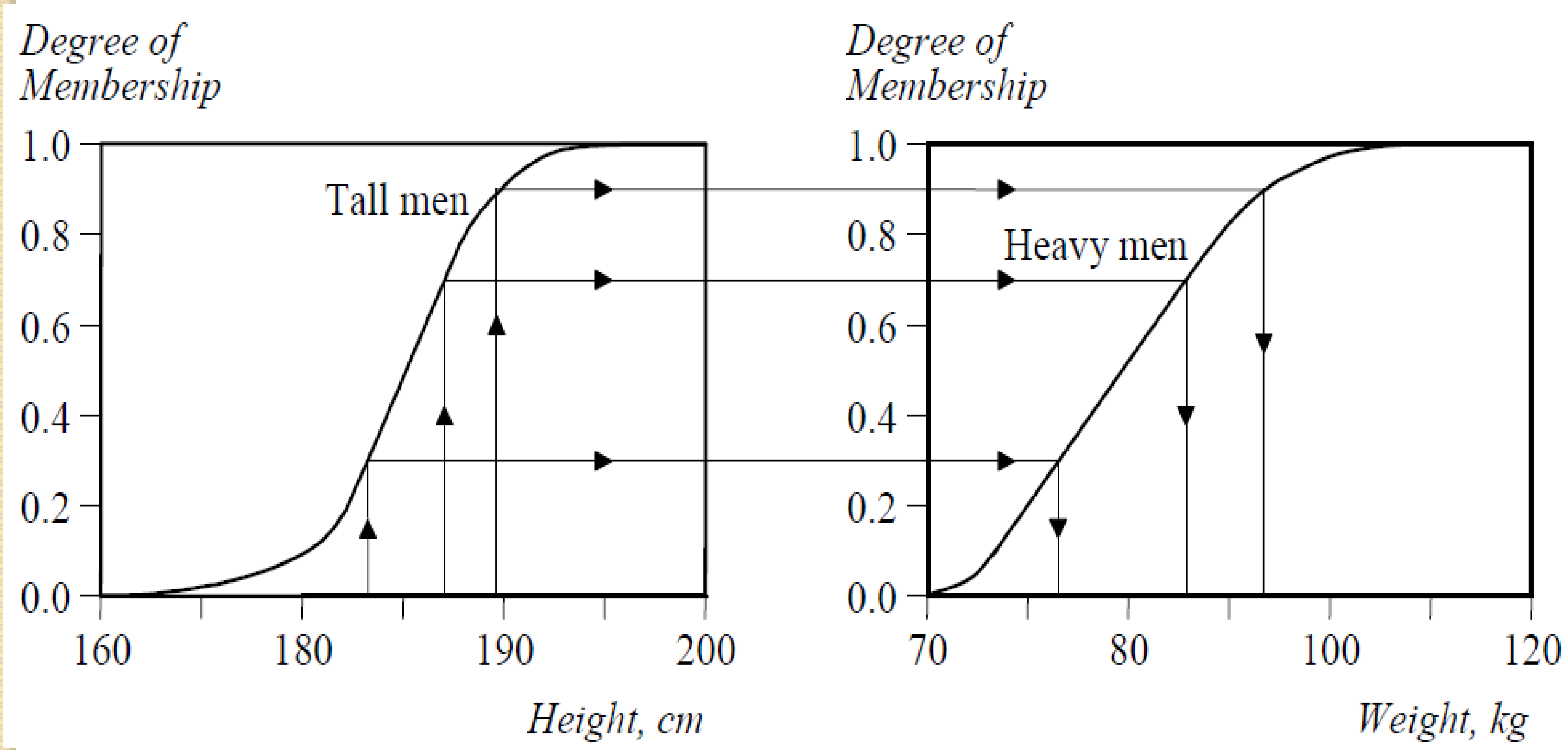




These fuzzy sets provide the basis for a weight estimation model. The model is based on a relationship between a man's height and his weight:

IF height is tall
THEN weight is heavy

- The value of the output or a truth membership grade of the rule consequent can be estimated directly from a corresponding truth membership grade in the antecedent. This form of fuzzy inference uses a method called monotonic selection.



A fuzzy rule can have multiple antecedents, for example:

IF	project_duration is long
AND	project_staffing is large
AND	project_funding is inadequate
THEN	risk is high

IF	service is excellent
OR	food is delicious
THEN	tip is generous

The consequent of a fuzzy rule can also include multiple parts, for instance:

IF	temperature is hot
THEN	hot water is reduced;
	cold-water is increased

Fuzzy inference systems:

1-Mamdani Fuzzy Logic

2-Sugeno Fuzzy Logic

1-Mamdani Fuzzy Logic

The most commonly used fuzzy inference technique is the so-called Mamdani method. In 1975, Professor Ebrahim Mamdani of London University built one of the first fuzzy systems to control a steam engine and boiler combination. He applied a set of fuzzy rules supplied by experienced human operators.

Mamdani fuzzy Logic

➢ The Mamdani-style fuzzy inference process is performed in four steps:

- fuzzification of the input variables,
- rule evaluation;
- aggregation of the rule outputs, and finally
- defuzzification.

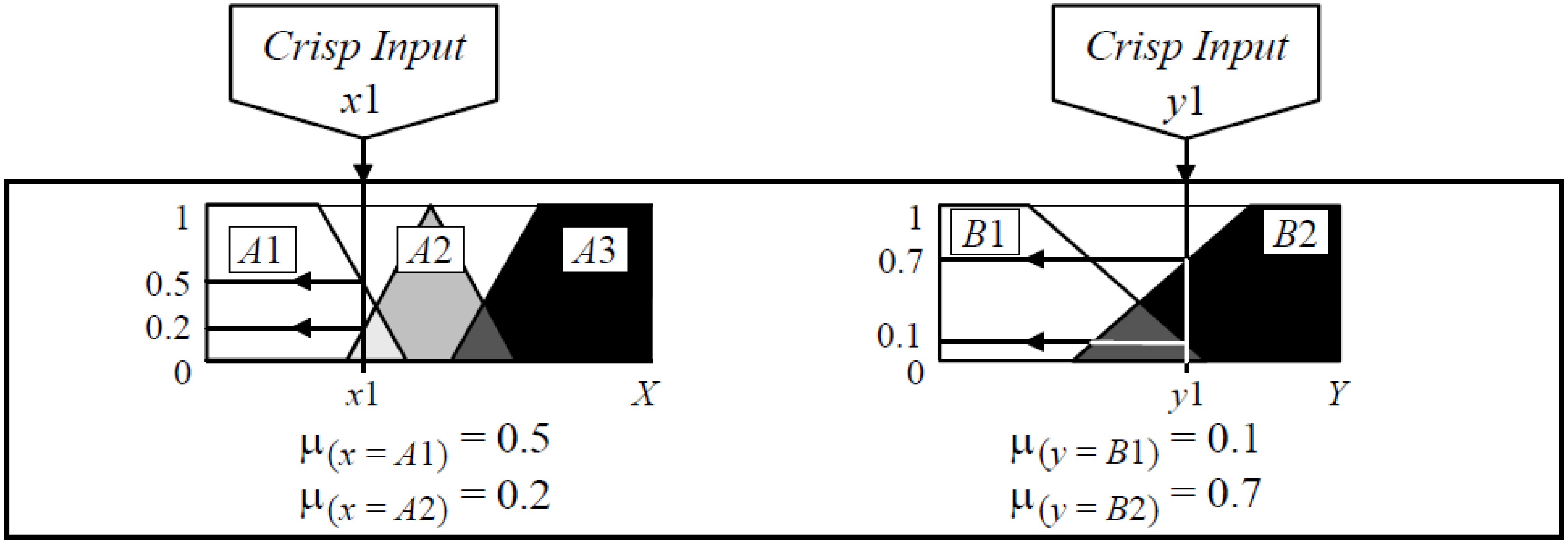
We examine a simple two-input one-output problem that includes three rules:

Rule: 1
IF x is $A3$
OR y is $B1$
THEN z is $C1$
Rule: 2
IF x is $A2$
AND y is $B2$
THEN z is $C2$
Rule: 3
IF x is $A1$
THEN z is $C3$

Rule: 1
IF *project_funding* is *adequate*
OR *project_staffing* is *small*
THEN *risk* is *low*
Rule: 2
IF *project_funding* is *marginal*
AND *project_staffing* is *large*
THEN *risk* is *normal*
Rule: 3
IF *project_funding* is *inadequate*
THEN *risk* is *high*

Step 1: Fuzzification

The first step is to take the crisp inputs, x1 and y1 (project funding and project staffing), and determine the degree to which these inputs belong to each of the appropriate fuzzy sets.



$\mu_{(x=A1)} = 0.5$

$\mu_{(x=A2)} = 0.2$

$\mu_{(y=B1)} = 0.1$

$\mu_{(y=B2)} = 0.7$

Step 2: Rule Evaluation

- The second step is to take the fuzzified inputs, $\mu(x=A1) = 0.5$, $\mu(x=A2) = 0.2$, $\mu(y=B1) = 0.1$ and $\mu(y=B2) = 0.7$, and apply them to the antecedents of the fuzzy rules.

- If a given fuzzy rule has multiple antecedents, the fuzzy operator (AND or OR) is used to obtain a single number that represents the result of the antecedent evaluation. This number (the truth value) is then applied to the consequent membership function.

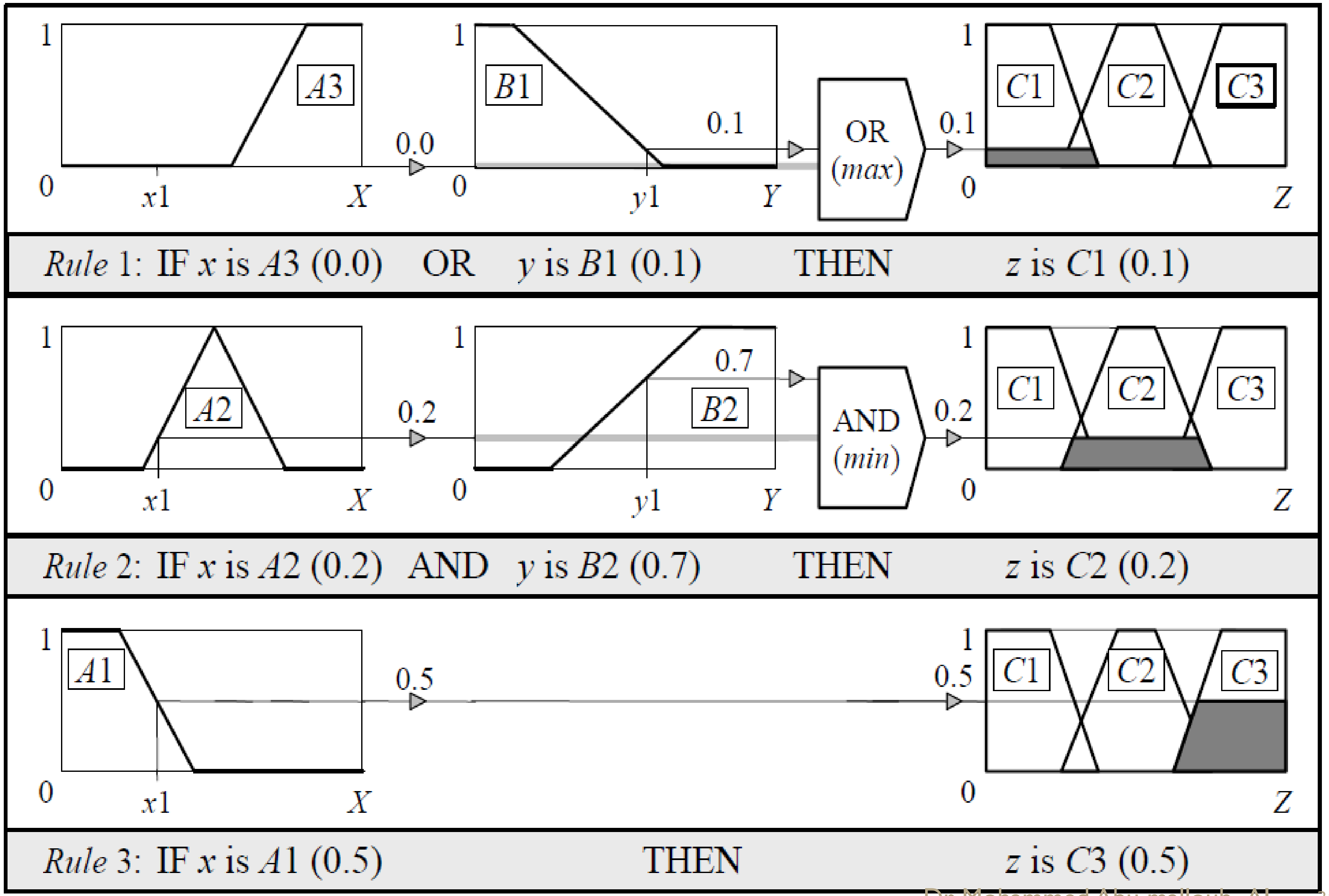
- To evaluate the disjunction of the rule antecedents, we use the OR fuzzy operation. Typically, fuzzy expert systems make use of the classical fuzzy operation union:

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

- Similarly, in order to evaluate the conjunction of the rule antecedents, we apply the AND fuzzy operation intersection:

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

Mamdani-style rule evaluation



Rule 1: IF *x* is *A*3 (0.0) OR *y* is *B*1 (0.1) THEN *z* is *C*1 (0.1)

Rule 2: IF *x* is *A*2 (0.2) AND *y* is *B*2 (0.7) THEN *z* is *C*2 (0.2)

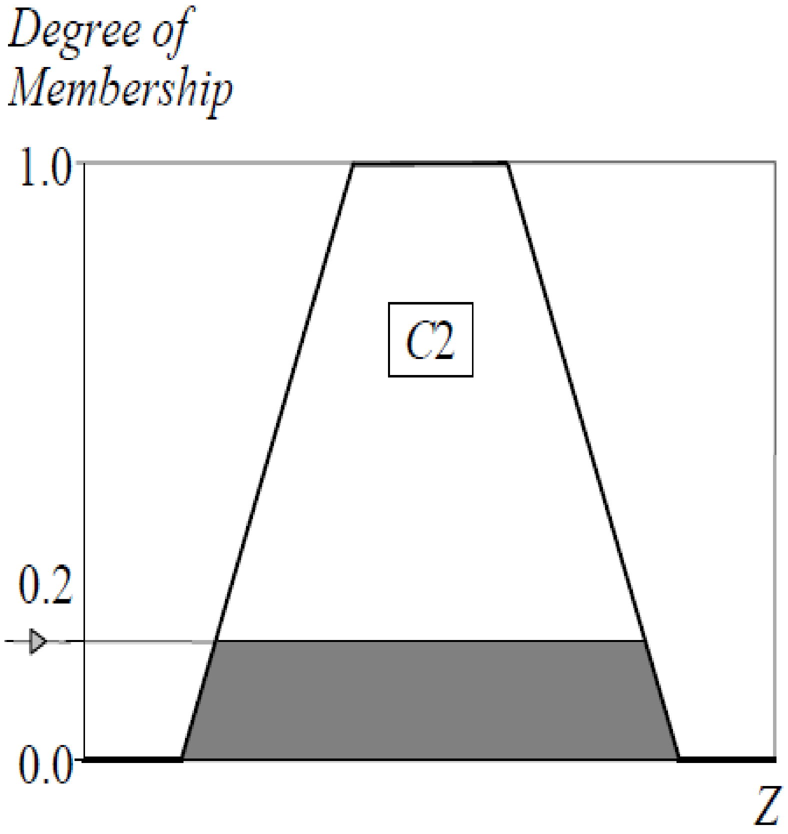
Rule 3: IF *x* is *A*1 (0.5) THEN *z* is *C*3 (0.5)

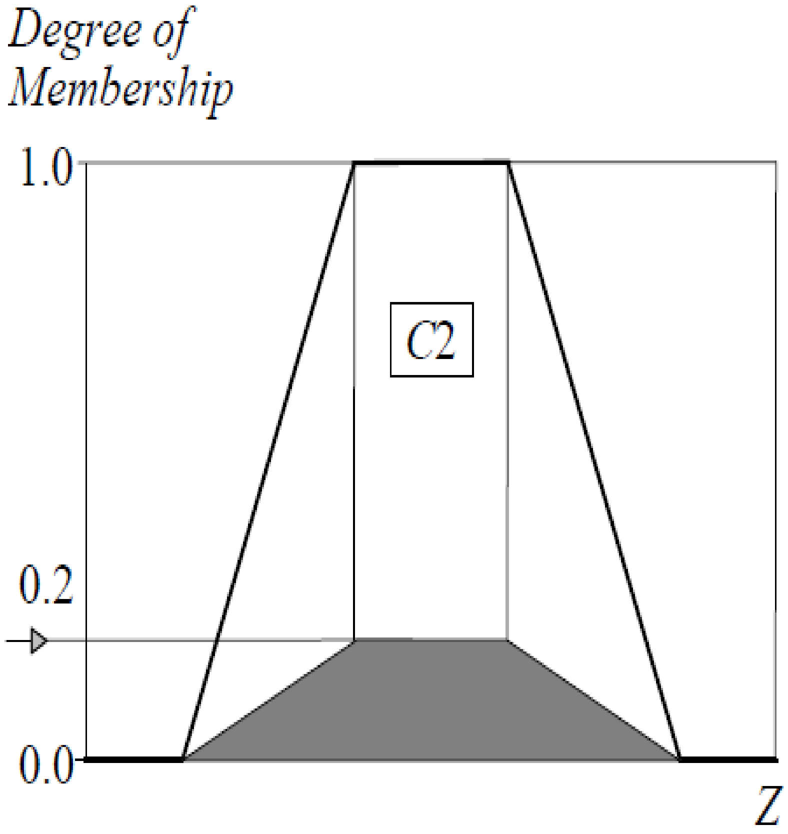
Now the result of the antecedent evaluation can be applied to the membership function of the consequent.

■ The most common method of correlating the rule consequent with the truth value of the rule antecedent is to cut the consequent membership function at the level of the antecedent truth. This method is called clipping. Since the top of the membership function is sliced, the clipped fuzzy set loses some information. However, clipping is still often preferred because it involves less complex and faster mathematics, and generates an aggregated output surface that is easier to defuzzify.

While clipping is a frequently used method, scaling offers a better approach for preserving the original shape of the fuzzy set. The original membership function of the rule consequent is adjusted by multiplying all its membership degrees by the truth value of the rule antecedent. This method, which generally loses less information, can be very useful in fuzzy expert systems.

Clipped and scaled membership functions



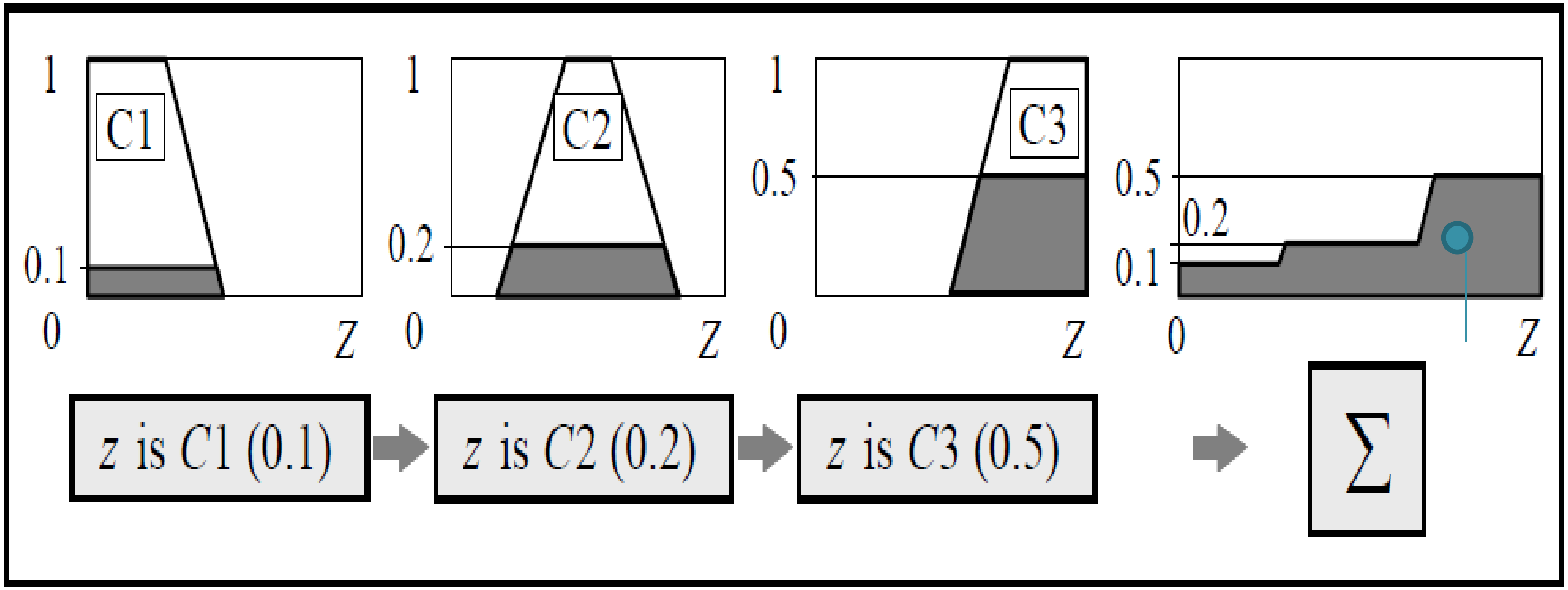


Step 3: Aggregation of the rule outputs

Aggregation is the process of unification of the outputs of all rules. We take the membership functions of all rule consequents previously clipped or scaled and combine them into a single fuzzy set.

The input of the aggregation process is the list of clipped or scaled consequent membership functions, and the output is one fuzzy set for each output variable.

Aggregation of the rule outputs



Step 4: Defuzzification

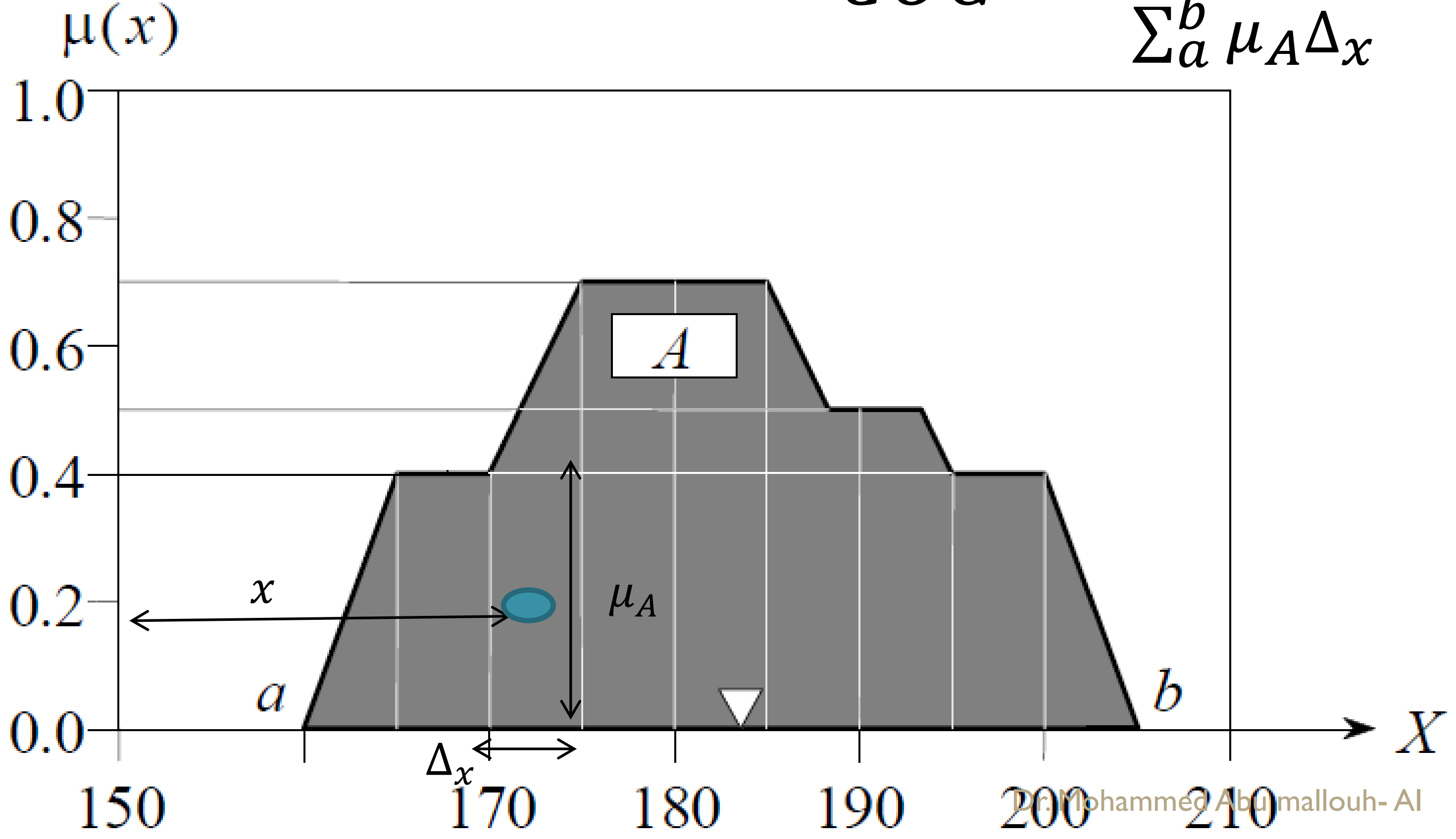
The last step in the fuzzy inference process is defuzzification. Fuzziness helps us to evaluate the rules, but the final output of a fuzzy system has to be a crisp number. The input for the defuzzification process is the aggregate output fuzzy set and the output is a single number.

- There are several defuzzification methods, but probably the most popular one is the centroid technique. It finds the point where a vertical line would slice the aggregate set into two equal masses. Mathematically this center of gravity (COG) can be expressed as:

$$COG = \frac{\int_a^b \mu_A(x)\,x\,dx}{\int_a^b \mu_A(x)\,dx}$$

- Centroid defuzzification method finds a point representing the center of gravity of the fuzzy set, A, on the interval, ab.
- A reasonable estimate can be obtained by calculating it over a sample of points.

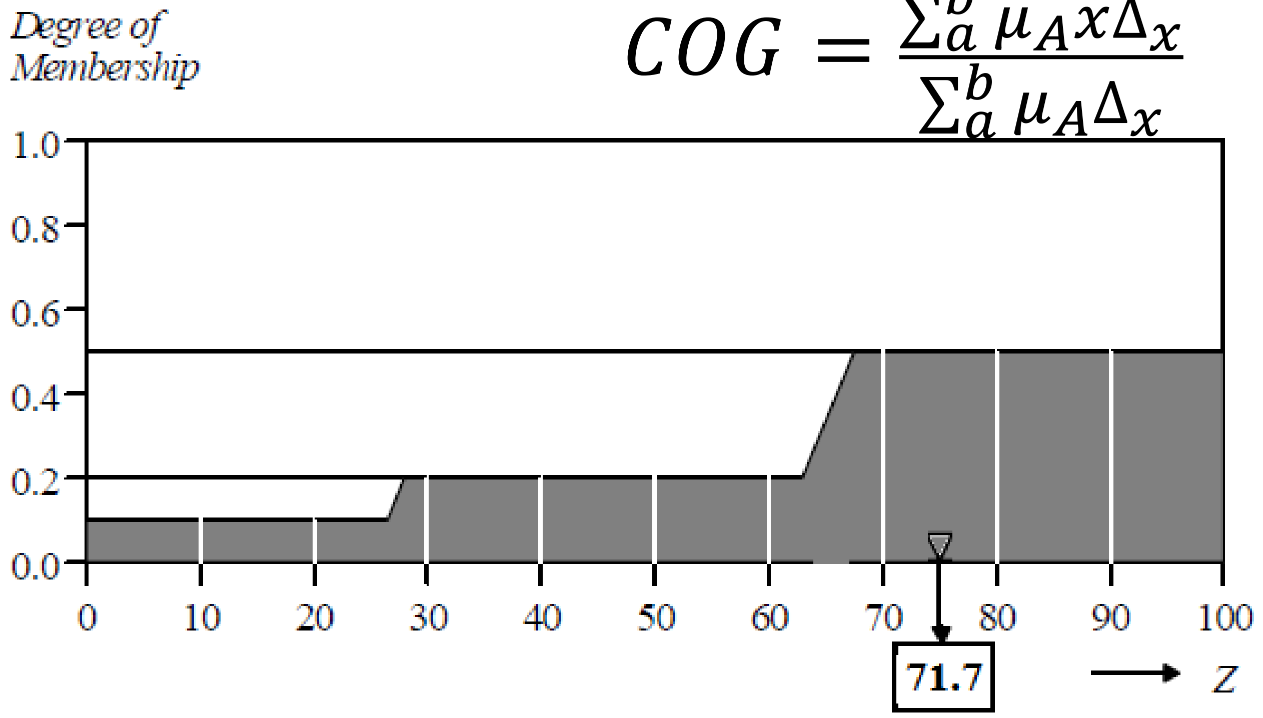
$$COG = \frac{\sum_a^b \mu_A x \Delta_x}{\sum_a^b \mu_A \Delta_x}$$



Centre of gravity (COG):

$$COG = \frac{[(5+15+25)\times 0.1 + (35+45+55+65)\times 0.2 + (75+85+95)\times 0.5]\ (10)}{[0.1+0.1+0.1+0.2+0.2+0.2+0.2+0.5+0.5+0.5]\ (10)} = 71.7$$

$$COG = \frac{\sum_a^b \mu_A x \Delta_x}{\sum_a^b \mu_A \Delta_x}$$



Sugeno fuzzy inference

■ Mamdani-style inference, as we have just seen, requires us to find the centroid of a two-dimensional shape by integrating across a continuously varying function. In general, this process is not computationally efficient.

■ Michio Sugeno suggested to use a single spike, a singleton, as the membership function of the rule consequent. A singleton, or more precisely a fuzzy singleton, is a fuzzy set with a membership function that is unity at a single particular point on the universe of discourse and zero everywhere else.

Sugeno-style fuzzy inference is very similar to the Mamdani method. Sugeno changed only a rule consequent. Instead of a fuzzy set, he used a mathematical function of the input variable. The format of the Sugeno-style fuzzy rule is :

IF x is A

AND y is B

THEN z is f (x, y) =$[k_{10} + k_{11}X + k_{12}.Y]$

where x, y and z are linguistic variables; A and B are fuzzy sets on universe of discourses X and Y, respectively; and f (x, y) is a mathematical function.

- The most commonly used zero-order Sugeno fuzzy model applies fuzzy rules in the following form:

  IF x is A

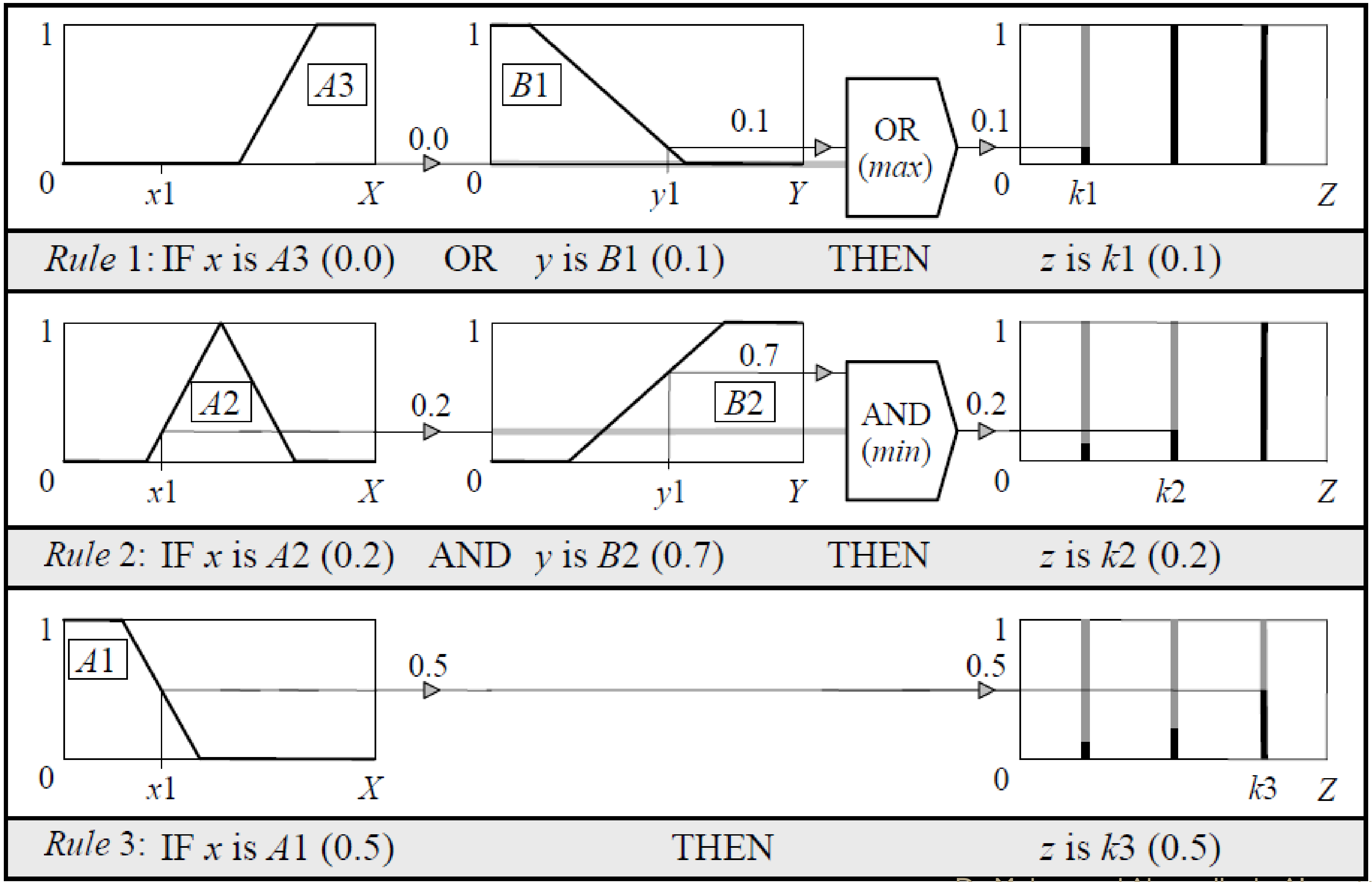
  AND y is B

  THEN z is k = $[k_{10} + k_{11} + k_{12}]$

where k is a constant.

- In this case, the output of each fuzzy rule is constant. All consequent membership functions are represented by singleton spikes.

Sugeno-style rule evaluation

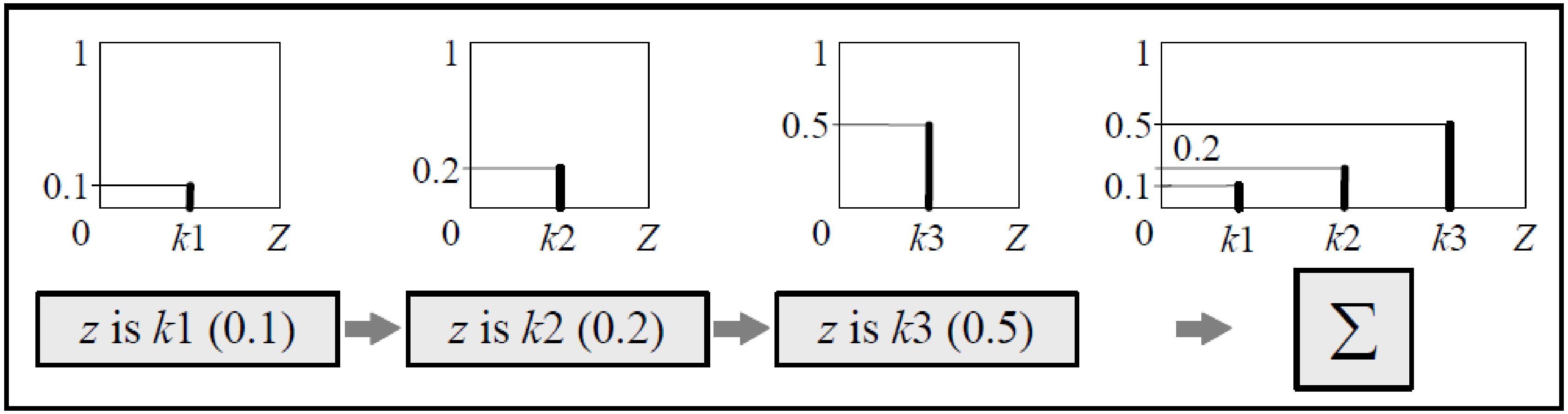


Rule 1: IF *x* is *A*3 (0.0) OR *y* is *B*1 (0.1) THEN *z* is *k*1 (0.1)

Rule 2: IF *x* is *A*2 (0.2) AND *y* is *B*2 (0.7) THEN *z* is *k*2 (0.2)

Rule 3: IF *x* is *A*1 (0.5) THEN *z* is *k*3 (0.5)

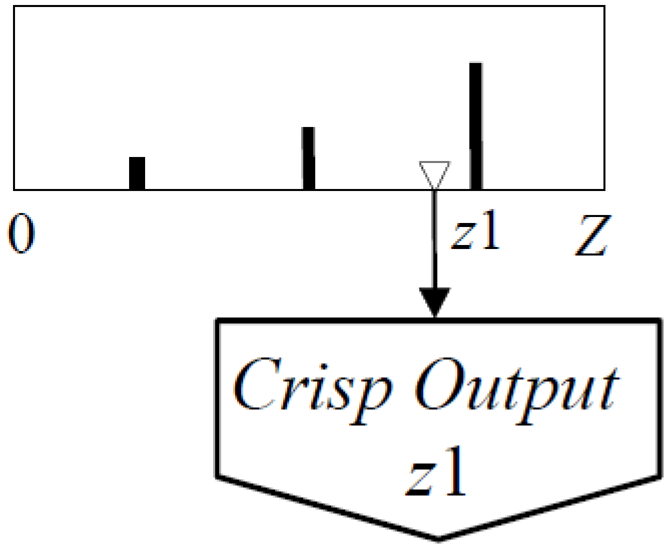
Sugeno-style aggregation of the rule outputs



Weighted average (WA) zero order:

$$WA = \frac{\mu_1[k_1]+\mu_2[k_2]+\mu_3[k_3]}{\mu_1+\mu_2+\mu_3} = \frac{0.1[20]+0.2[50]+0.5[80]}{0.1+0.2+0.5} = 65$$

Sugeno-style defuzzification



Weighted average (WA) first order:

$$WA = \frac{\mu_1[k_{10} + k_{11}X + k_{12}.Y] + \mu_2[k_{20} + k_{21}X + k_{22}.Y] + \mu_3[k_{30} + k_{31}X + k_{32}.Y]}{\mu_1 + \mu_2 + \mu_3}$$

How to make a decision on which method to apply - Mamdani or Sugeno?

■ Mamdani method is widely accepted for capturing expert knowledge. It allows us to describe the expertise in more intuitive, more human-like manner. However, Mamdani-type fuzzy inference entails a substantial computational burden.

■ On the other hand, Sugeno method is computationally effective and works well with optimization and adaptive techniques, which makes it very attractive in control problems, particularly for dynamic nonlinear systems.

Process of developing a fuzzy expert system

1. Specify the problem and define linguistic variables.

2. Determine fuzzy sets.

3. construct fuzzy rules.

4. Encode the fuzzy sets, fuzzy rules and procedures to perform fuzzy inference into the expert system.

5. Evaluate and tune the system.

Tuning fuzzy systems

1. Review model and output variables, and if required redefine ranges..

2.Review the fuzzy sets, and if required define additional sets on the universe of discourse.

3. The use of wide fuzzy sets may cause the fuzzy system to perform roughly..

Tuning fuzzy systems

4. Provide sufficient overlap between neighboring sets.

5. It is suggested that triangle to triangle and trapezoid to triangle fuzzy setts should overlap between 25% to 50% of their bases.

6. Review the existing rules, and if required add new rules to the rule base..

7. Revise shapes of the fuzzy sets.. In most cases, fuzzy systems are highly tolerant of a shape approximation..

Ch 4
Hybrid intelligent systems:
Neural expert systems and neuro-fuzzy systems

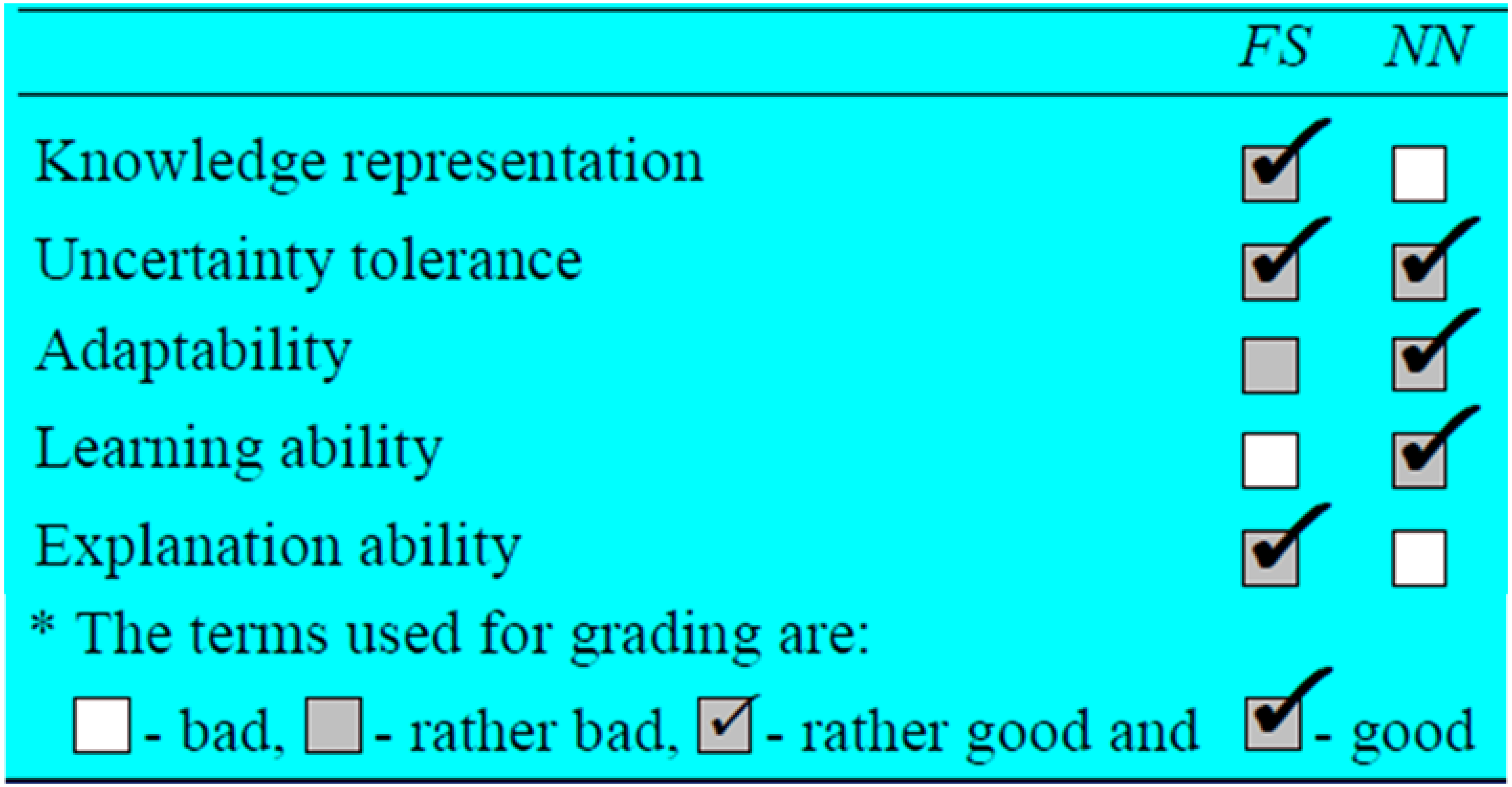
- Introduction
- Neuro-fuzzy systems
- ANFIS: Adaptive Neuro-Fuzzy Inference system
- Summary

Introduction

- A hybrid intelligent system is one that combines at least two intelligent technologies .

- For example, combining a neural network with a fuzzy system results in a hybrid neuro-fuzzy system .

Comparison of Fuzzy Systems, Neural Networks



	FS	*NN*
Knowledge representation	✔	☐
Uncertainty tolerance	✔	✔
Adaptability	▨	✔
Learning ability	☐	✔
Explanation ability	✔	☐

* The terms used for grading are:

☐ - bad, ▨ - rather bad, ☑ - rather good and ✔ - good

Neuro-fuzzy systems

- Fuzzy logic and neural networks are important tools in building intelligent systems.
- However, fuzzy systems lack the ability to learn and cannot adjust themselves to a new environment
- On the other hand, although neural networks can learn, they are opaque to the user.

Neuro-fuzzy systems

- The merger of a neural network with a fuzzy system into one integrated system therefore offers a promising approach to building intelligent systems.
- Integrated neuro-fuzzy systems can combine the parallel computation and learning abilities of neural networks with the humanlike knowledge representation and explanation abilities of fuzzy systems.
- As a result, neural networks become more transparent, while fuzzy systems become capable of learning.

Neuro-fuzzy systems

- A neuro-fuzzy system is, in fact, a neural network that is functionally equivalent to a fuzzy inference model.
- It can be trained to develop IF-THEN fuzzy rules and determine membership functions for input and output variables of the system.
- Expert knowledge can be easily incorporated into the structure of the neuro-fuzzy system.
- The structure of a neuro-fuzzy system is similar to a multi-layer neural network.
- In general, a neuro-fuzzy system has input and output layers, and three hidden layers that represent membership functions and fuzzy rules.

ANFIS: Adaptive Neuro-Fuzzy Inference System

- The Sugeno fuzzy model was proposed for a systematic approach to generating fuzzy rules from a given input-output data set. A typical Sugeno fuzzy rule can be expressed in the following form:

IF x_1 is A_1

AND x_2 is A_2

....

AND x_m is A_m

THEN $y = f(x_1, x_2, \ldots\ldots, x_m)$

Where $x_1, x_2, \ldots\ldots, x_m$ are input variables; $A_1, A_2, \ldots\ldots, A_m$ are fuzzy sets; and y is

near function of the input variables. when y is a constant, we obtain a zero-order Sugeno fuzzy model in which the consequent of a rule is specified by a singlton . When y is a first-order polynomial, i.e

$y = k_0 + k_1 x_1 + k_2 x_2 + \ldots + k_m x_m$

We obtain a first-order Sugeno fuzzy model .

ANFIS: Adaptive Neuro-Fuzzy Inference System

•Jang’s ANFIS is normally represented by a six-layer feed forward neural network.

•ANFIS architecture that corresponds to the first order Sugeno fuzzy model.

•For simplicity, we assume that the ANFIS has two inputs: x1 and x2 ,
and one output: y. Each input is represented by two fuzzy

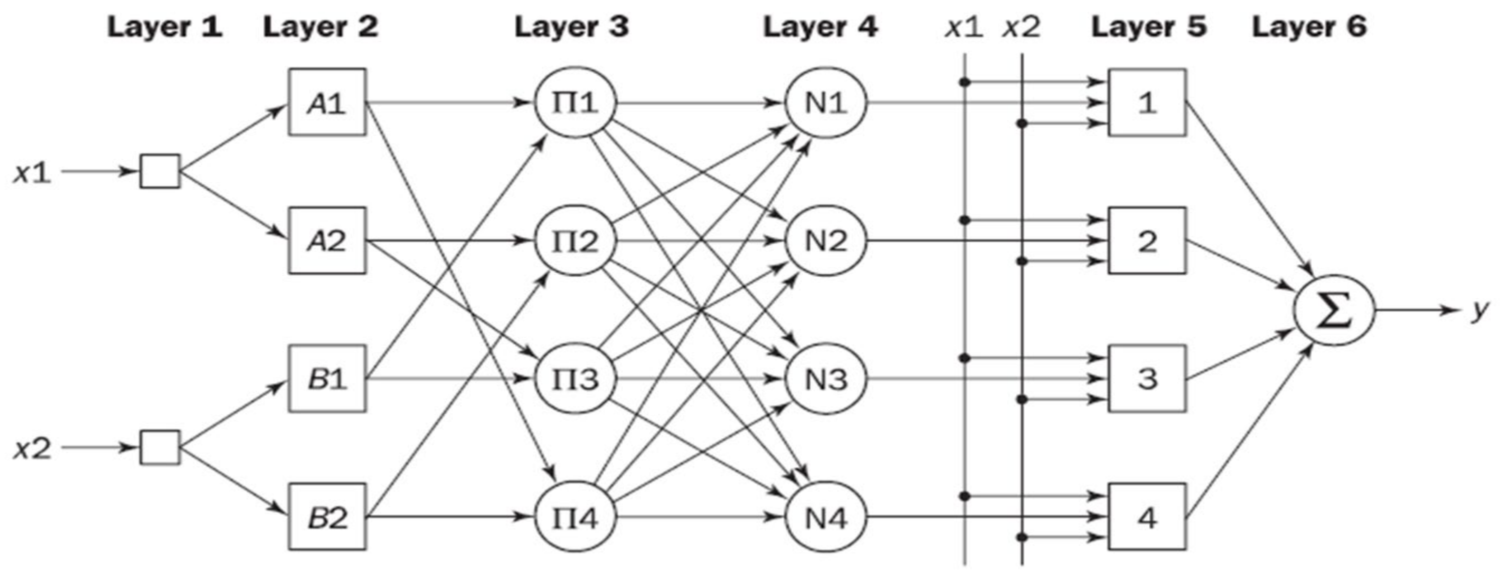


Figure 8.10 Adaptive Neuro-Fuzzy Inference System (ANFIS)

ANFIS: Adaptive Neuro-Fuzzy Inference System

Rule 1 :
IF X1 is A1
AND X2 is B1
THEN y=$f_1 = k_{10} + k_{11}x1 + k_{12}x2$

Rule 2 :
IF X1 is A2
AND X2 is B2
THEN y=$f_2 = k_{20} + k_{21}x1 + k_{22}x2$

Rule 3 :
IF X1 is A2
AND X2 is B1
THEN y=$f_3 = k_{30} + k_{31}x1 + k_{32}x2$

Rule 4 :
IF X1 is A1
AND X2 is B2
THEN y=$f_4 = k_{40} + k_{41}x1 + k_{42}x2$

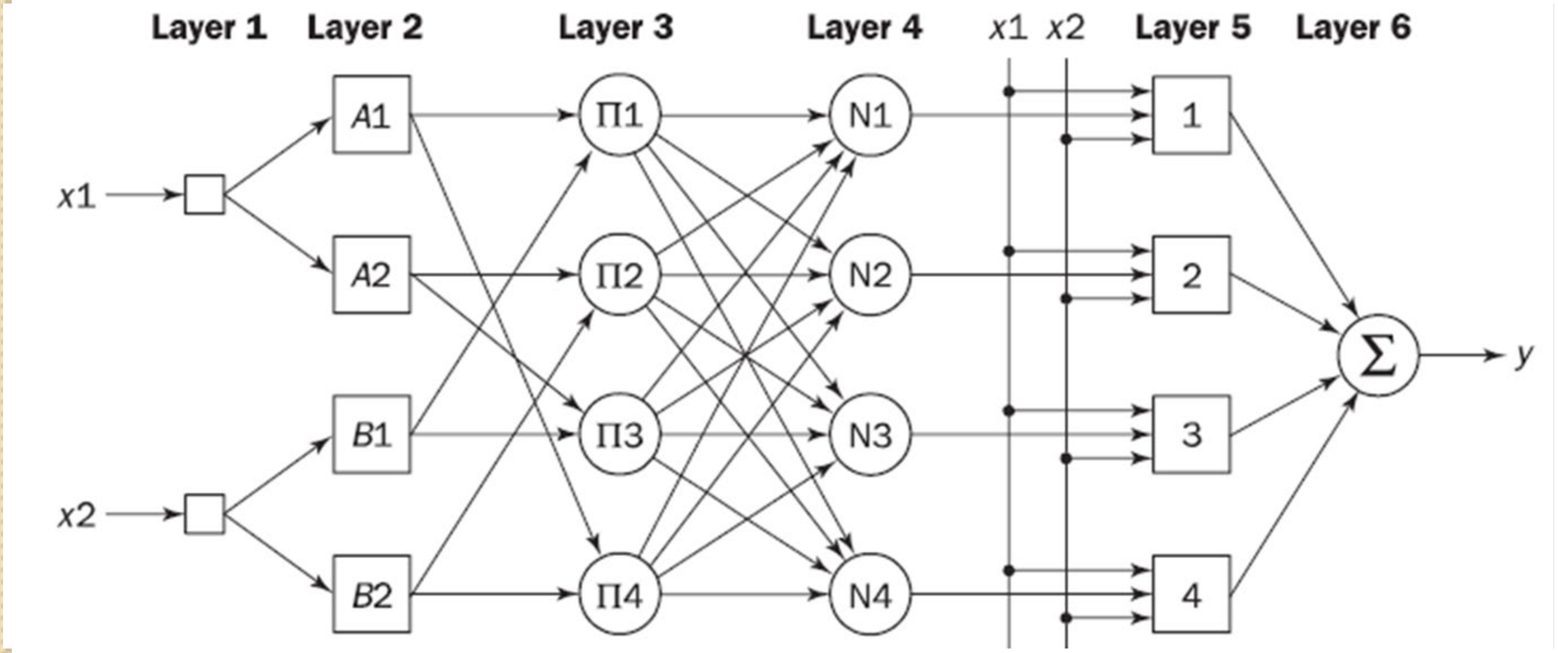
Where x1,x2 are input variables;A1 and A2 are fuzzy sets on the universe of discourse X1; B1 and B2 are fuzzy sets on the universe of discourse X2;andk_{i0},$k_{i1}$and$k_{i2}$is a set of parameters specified for rule i .

ANFIS: Adaptive Neuro-Fuzzy Inference System

- Let us now discuss the purpose of each layer in Jang's ANFIS.
- Layer 1 is the input layer. Neurons in this layer simply pass external crisp signals to Layer 2.

$$y_i^{(1)} = x_i^{(1)},$$

Where $x_i^{(1)}$ is the input and $y_i^{(1)}$ is the output of input neuron i in Layer 1 .



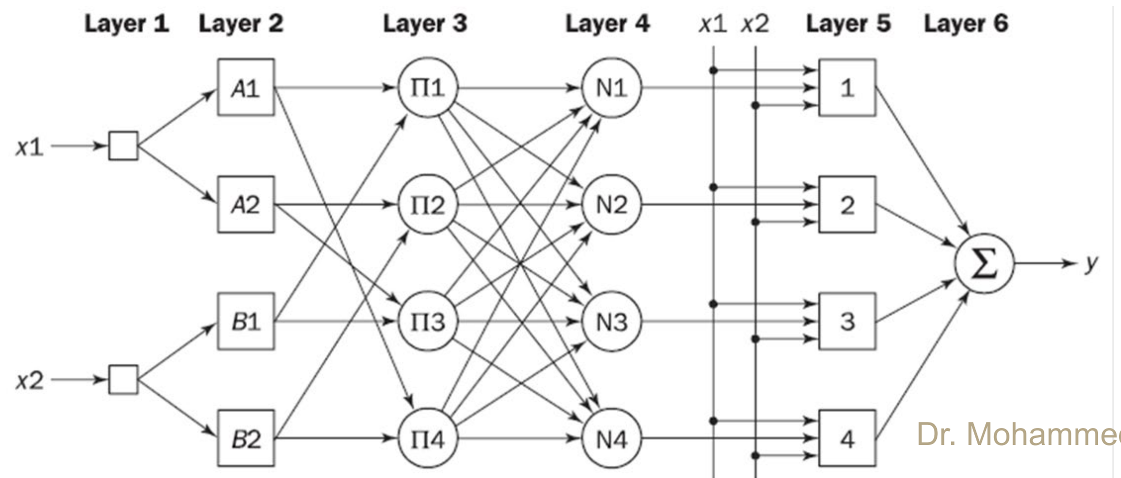
ANFIS: Adaptive Neuro-Fuzzy Inference System

Layer 2 is the fuzzification layer . Neuros in this layer perform fuzzification .

In Jang's model, fuzz bell activation function ,which has a regular bell shape, is specified as:

$$y_i^{(2)} = \frac{1}{1 + \left(\frac{x_i^{(2)} - a_i}{c_i}\right)^{2b_i}}$$

Where $x_i^{(2)}$ is the input and $y_i^{(2)}$ is the output of neuron i in Layer 2 ;$a_i c_i$, b_i and and Layer 2 is the fuzzy are parameters that control, respectively, the centre,slope and width the In Jang's model, fuzz bell activation function of neuron i.



ANFIS: Adaptive Neuro-Fuzzy Inference System

Layer 3 is the rule layer. Each neuron in this layer corresponds to a single Sugeno-type fuzzy rule. A rule neuron receives inputs from the respective fuzzification neurons and calculates the firing strength of the rule it represents. In an ANFIS ,the conjunction of the rule antecedents is evaluated by the operator product. Thus, the output of neuron I in layer 3 is obtained as ,

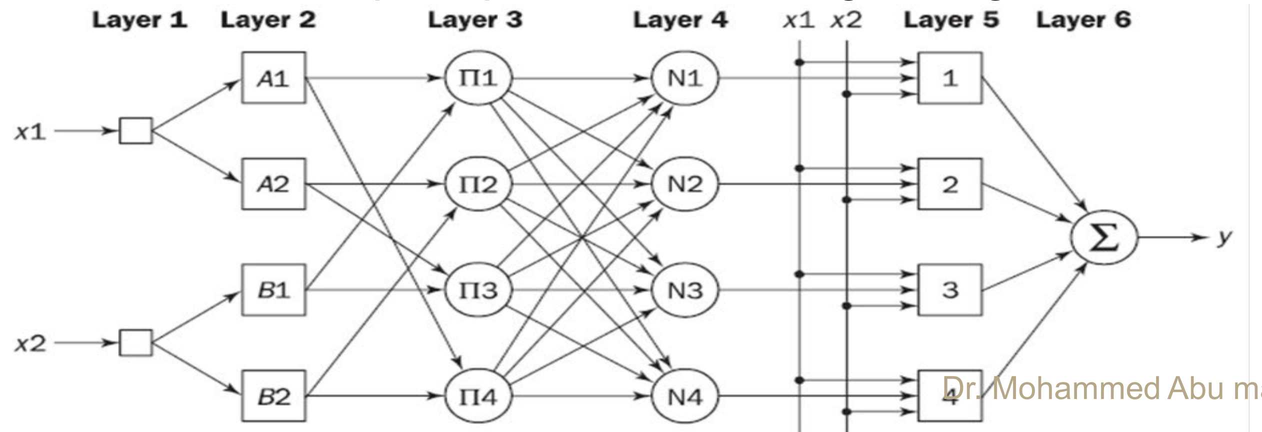
$$y_i^{(3)} = \prod_{i=1}^{k} x_{ij}^{(3)}$$

where $x_{ij}^{(3)}$ are the inputs and $y_i^{(3)}$ is the outputs of rule neuron i in layer 3

For example,

$$\mathrm{y}_1^{(3)} = \mu_{A1} \times \mu_{B1} = \mu_1,$$

Where the value of μ1 represents the firing strength ,or the truth value, of rule ,



ANFIS: Adaptive Neuro-Fuzzy Inference System

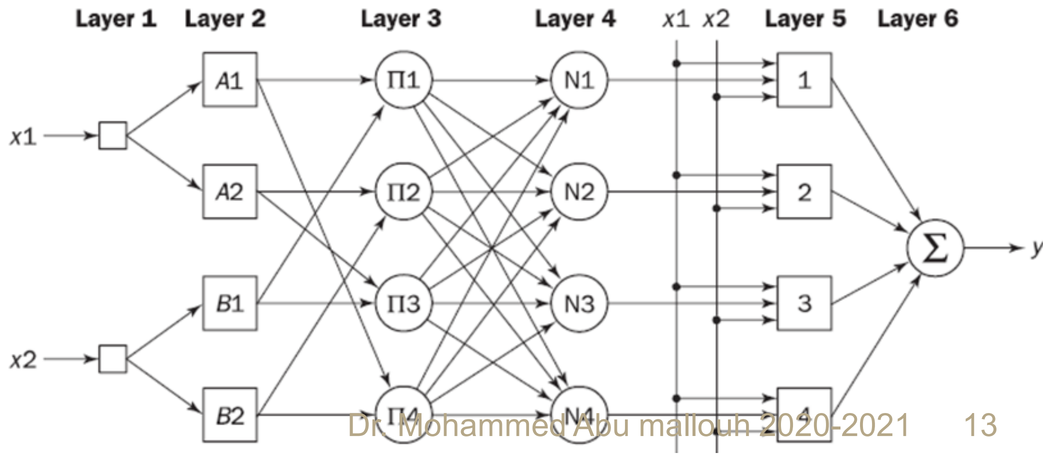
Layer 4 is the normalization layer .Each neuron in this layer receives inputs from all neurons in the rule layer, and calculates the normalized firing strength of a given rule.
The normalized firing strength is the ratio of the firing strength of a given rule to the sum of firing strength of all rules. It represents the contribution of a given rule to the final result .
Thus, the output of neuron I in layer 4 is determined as ,

$$y_i^{(4)} = \frac{x_i^{(4)}}{\sum_{j=1}^{n} x_{ij}^{(4)}} = \frac{\mu_i}{\sum_{j=1}^{n} \mu_j} = \overline{\mu_i},$$

where $x_{ij}^{(4)}$ is the input from neuron j located in layer 3 to neuron i in layer 4, and n is the total number of rule neurons.

For example,

$$\mathrm{y}_{\mathrm{N1}}^{(4)} = \frac{\mu_1}{\mu_1 + \mu_2 + \mu_3 + \mu_4} = \overline{\mu_1}$$



ANFIS: Adaptive Neuro-Fuzzy Inference System

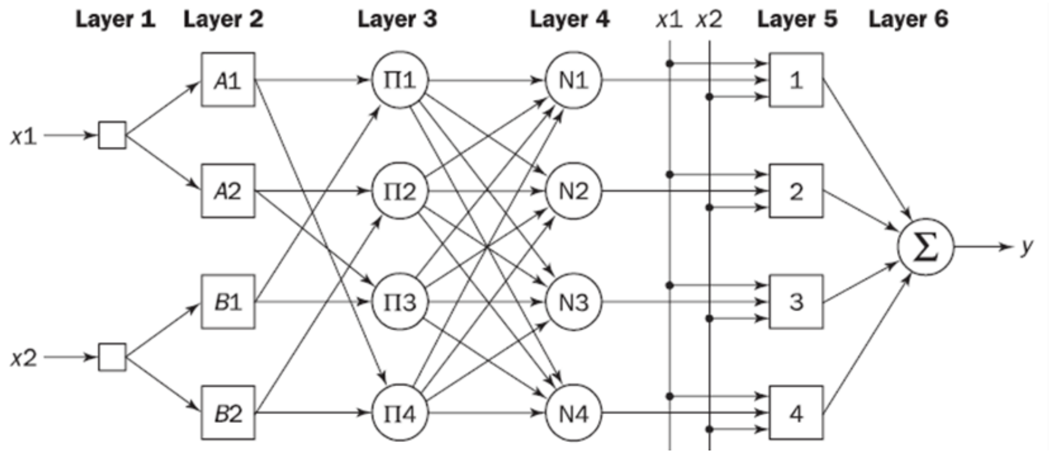
Layer 5 is the deffuzzification layer. Each neuron in this layer is connected to the respective normalization neuron , and also receives initial inputs,x1 and x2 .
A defuzzification neuron calculates the weighted consequent value of a given rule as:

$$y_i^{(5)} = x_i^{(5)}\left[k_{i0} + k_{i1}x_1 + k_{i2}x_2\right] = \overline{\mu_i}\left[k_{i0} + k_{i1}x_1 + k_{i2}x_2\right],$$

where $x_i^{(5)}$ is the input and $y_i^{(5)}$ is the output of defuzzification neuron i in layer 5, and k_{i0}, k_{i1} and k_{i2} is a set of consequent parameters of rule i.

Layer 6 is represented by a single summation neuron. This neuron calculates the sum of outputs of all defuzzification neurons and produces the overall ANFIS output, y,

$$\mathrm{y} = \sum_{i=1}^{n} x_i^{(6)} = \sum_{i=1}^{n} \overline{\mu_i}\left[k_{i0} + k_{i1}x_1 + k_{i2}x_2\right]$$



How does an ANFIS learn?

- An ANFIS uses a hybrid learning algorithm that combines the least-squares estimator and the gradient descent method

- First, initial activation functions are assigned to each membership neuron.

- In the ANFIS training algorithm, each epoch is composed from a forward pass and a backward pass.

- In the forward pass, a training set of input patterns (an input vector) is presented to the ANFIS, neuron outputs are calculated on the layer-by-layer basis, and rule consequent parameters are identified by the least squares estimator

How does an ANFIS learn?

- In the Sugeno-style fuzzy inference, an output, y, is a linear function. Thus, given the values of the membership parameters and a training set of P input-output patterns, we can form P linear equations in terms of the consequent parameters as:

$$\left\{\begin{bmatrix} y_d(1) = \overline{\mu_1}(1) f_1(1) + \overline{\mu_2}(1) f_2(1) + \ldots + \overline{\mu_n}(1) f_n(1) \\ y_d(2) = \overline{\mu_1}(2) f_1(2) + \overline{\mu_2}(2) f_2(2) + \ldots + \overline{\mu_n}(2) f_n(2) \\ . \\ . \\ . \\ y_d(p) = \overline{\mu_1}(p) f_1(p) + \overline{\mu_2}(p) f_2(p) + \ldots + \overline{\mu_n}(p) f_n(p) \end{bmatrix}\right.$$

How does an ANFIS learn?

$$\begin{cases} y_d(1) = \bar{\mu}_1(1)\left[k_{10} + k_{11}x_1(1) + k_{12}x_2(1) + \ldots + k_{1m}x_m(1)\right] \\ \qquad + \bar{\mu}_2(1)\left[k_{20} + k_{21}x_1(1) + k_{22}x_2(1) + \ldots + k_{2m}x_m(1)\right] + \ldots \\ \qquad + \bar{\mu}_n(1)\left[k_{n0} + k_{n1}x_1(1) + k_{n2}x_2(1) + \ldots + k_{nm}x_m(1)\right] \\ y_d(2) = \bar{\mu}_1(2)\left[k_{10} + k_{11}x_1(2) + k_{12}x_2(2) + \ldots + k_{1m}x_m(2)\right] \\ \qquad + \bar{\mu}_2(2)\left[k_{20} + k_{21}x_1(2) + k_{22}x_2(2) + \ldots + k_{2m}x_m(2)\right] + \ldots \\ \qquad + \bar{\mu}_n(2)\left[k_{n0} + k_{n1}x_1(2) + k_{n2}x_2(2) + \ldots + k_{nm}x_m(2)\right] \\ . \\ . \\ . \\ y_d(p) = \bar{\mu}_1(p)\left[k_{10} + k_{11}x_1(p) + k_{12}x_2(p) + \ldots + k_{1m}x_m(p)\right] \\ \qquad + \bar{\mu}_2(p)\left[k_{20} + k_{21}x_1(p) + k_{22}x_2(p) + \ldots + k_{2m}x_m(p)\right] + \ldots \\ \qquad + \bar{\mu}_n(p)\left[k_{n0} + k_{n1}x_1(p) + k_{n2}x_2(p) + \ldots + k_{nm}x_m(p)\right] \end{cases}$$

Where m is the number of input variables, n is the number of neurons in the rule layer, and $y_d(p)$ is the desired overall outpt of the ANFIS when inputs $x_1(p)$, $x_2(p)$,….., $x_m(p)$ are presented to it .

How does an ANFIS learn?

y_d=AK

K=A^-1y_d

where $\mathrm{y_d}$ is a $\mathrm{p} \times 1$ desired output vector,

$$\mathrm{y_d} = \begin{cases} \mathrm{y_d}(1) \\ y_d(2) \\ . \\ . \\ . \\ y_d(p) \end{cases}$$

A is a $\mathrm{P} \times \mathrm{n}(1+\mathrm{m})$ matrix,

$$\begin{bmatrix} \overline{\mu_1}(1) & \overline{\mu_1}(1)x_1(1) \ldots \overline{\mu_1}(1)x_m(1) \ldots \overline{\mu_\mathrm{n}}(1) & \overline{\mu_\mathrm{n}}(1)x_1(1) \ldots \overline{\mu_\mathrm{n}}(1)x_{m,}(1) \\ \overline{\mu_1}(2) & \overline{\mu_1}(2)x_1(2) \ldots \overline{\mu_1}(2)x_m(2) \ldots \overline{\mu_\mathrm{n}}(2) & \overline{\mu_\mathrm{n}}(2)x_1(12) \ldots \overline{\mu_\mathrm{n}}(2)x_{m,}(2) \\ . & . \quad\quad . \quad\quad . & . \quad\quad . \\ . & . \quad \ldots \quad . \quad \ldots \quad . & . \quad \ldots \quad . \\ . & . \quad\quad . \quad\quad . & . \quad\quad . \\ \overline{\mu_1}(p) & \overline{\mu_1}(p)x_1(p) \ldots \overline{\mu_1}(p)x_m(p) \ldots \overline{\mu_\mathrm{n}}(p) & \overline{\mu_\mathrm{n}}(p)x_1(p) \ldots \overline{\mu_\mathrm{n}}(p)x_{m,}(p) \end{bmatrix}$$

and k is an $\mathrm{n}(1+\mathrm{m}) \times 1$ vector of unknown consequent parameters,

$$\mathrm{k} = \left[\mathrm{k_{10}} k_{11} k_{12} \ldots \mathrm{k_{1m}} k_{20} k_{21} k_{22} \ldots \mathrm{k_{2m}} \ldots \mathrm{k_{n0}} k_{n1} k_{n2} \ldots \mathrm{k_{nm}}\right]^T$$

How does an ANFIS learn?

- Usually the number of input-output patterns P used in training is greater than the number of consequent parameters.
- It means that we are dealing here with an over determined problem, and thus exact solution may not even exist.
- Instead, we solve for K numerically.
- In the ANFIS training algorithm suggested by Jang, both antecedent parameters and consequent parameters are optimized.
- In the forward pass, the consequent parameters are adjusted while the antecedent parameters remain fixed.
- In the backward pass, the antecedent parameters are tuned while the consequent parameters are kept fixed.

Example (matlab command line)

Function approximation using the ANFIS model

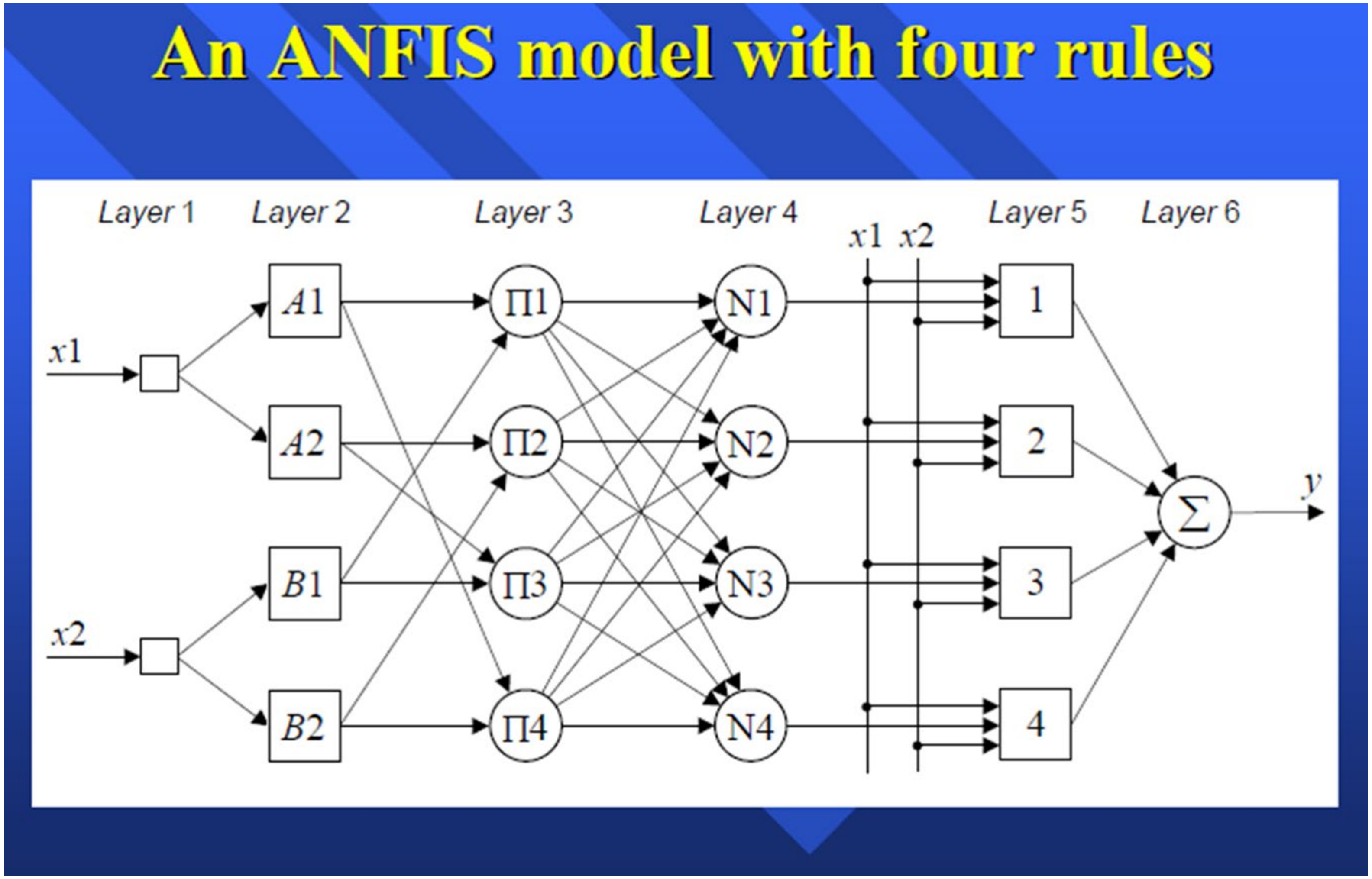
• In this example, an ANFIS is used to follow a trajectory of the non-linear function defined by the equation

$$y = \frac{\cos(2x_1)}{e^{x_2}}$$

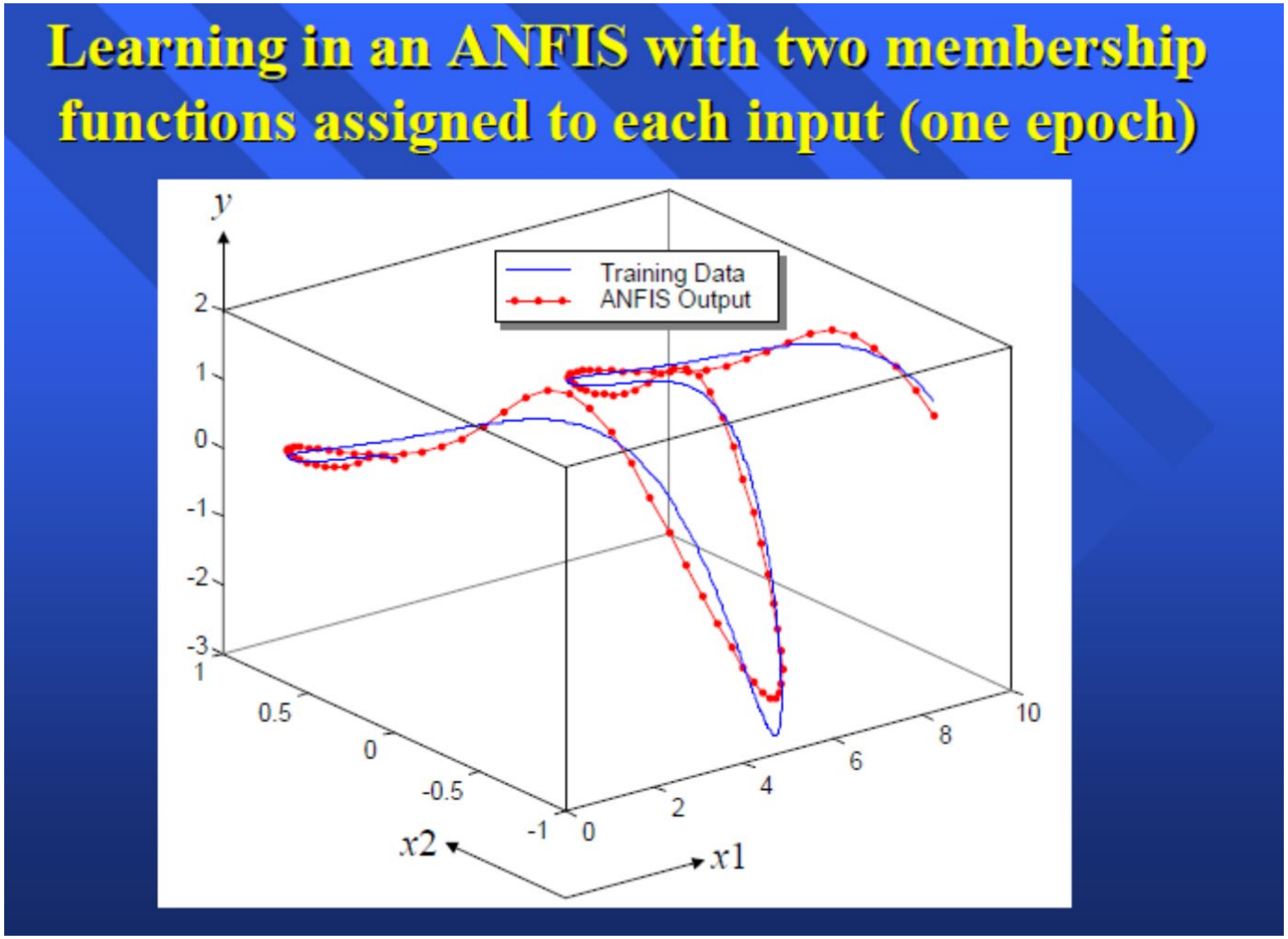
First, we choose an appropriate architecture for the ANFIS. An ANFIS must have two input-x1 and x2- and one output – y.

•Thus, in our example , the ANFIS is defined by four rules, and has the structure shown below.

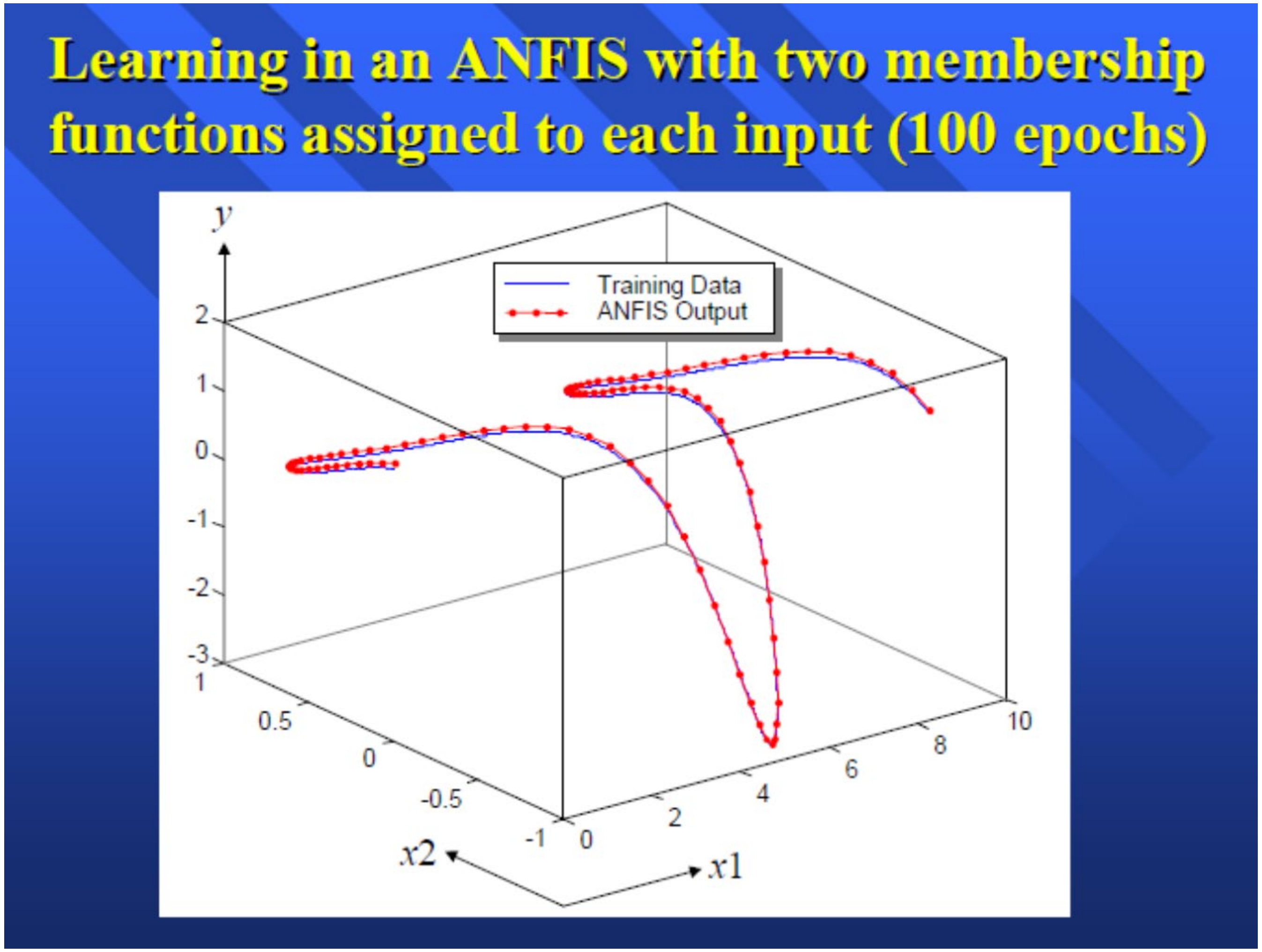
Example (matlab command line)



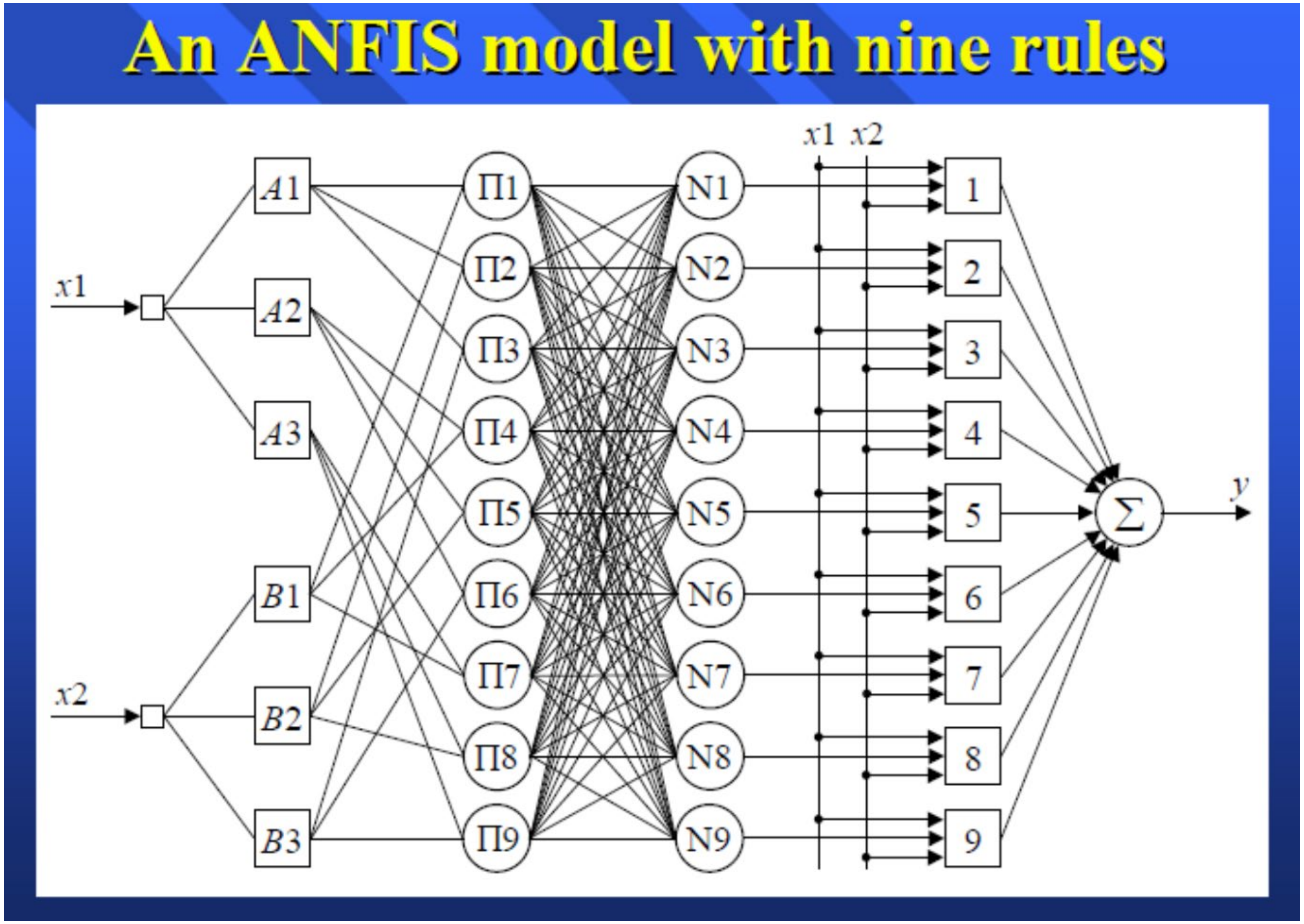
Example (matlab command line


Learning in an ANFIS with two membership functions assigned to each input (one epoch)

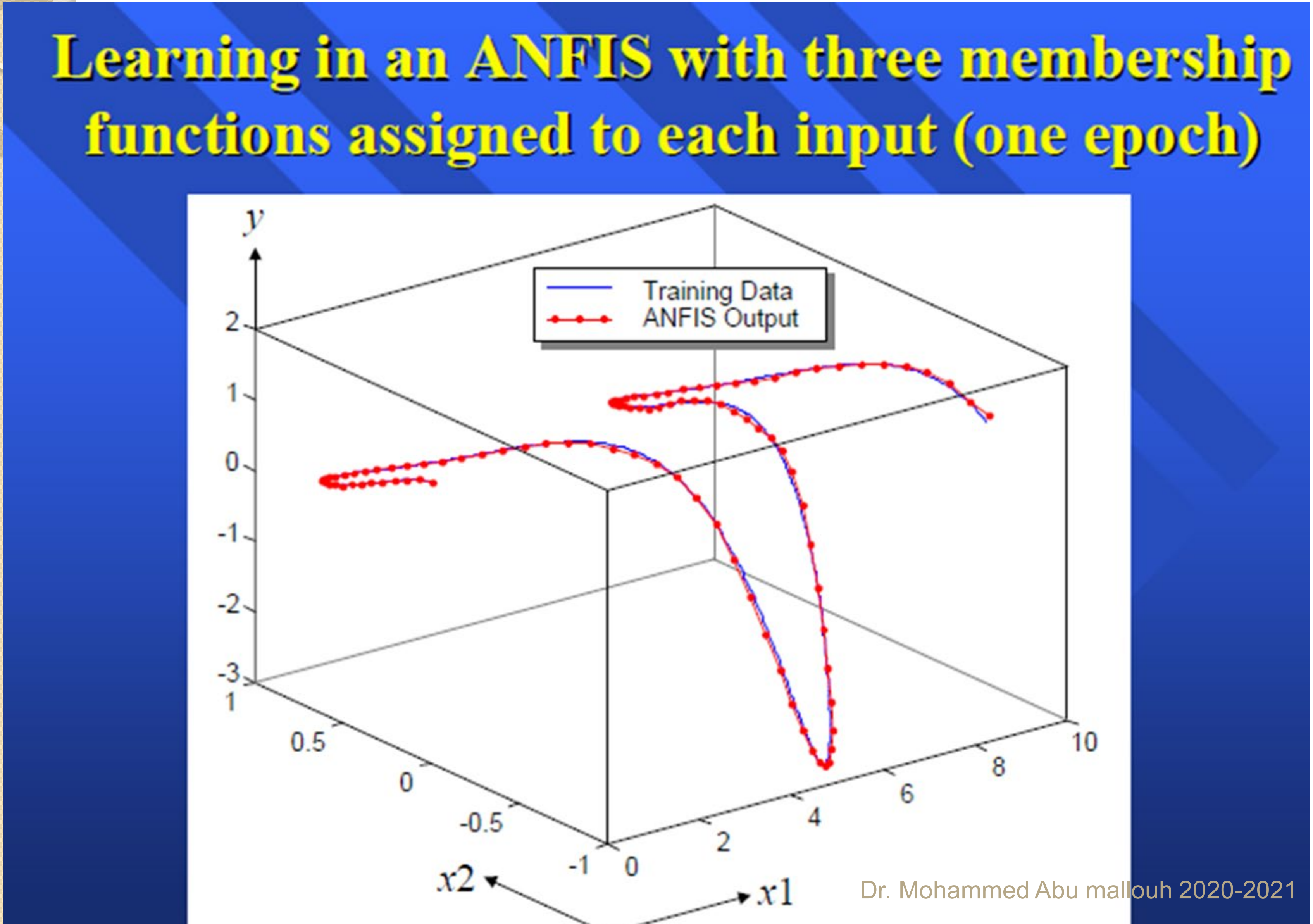
Example (matlab command line)


Learning in an ANFIS with two membership functions assigned to each input (100 epochs)

Example (matlab command line)

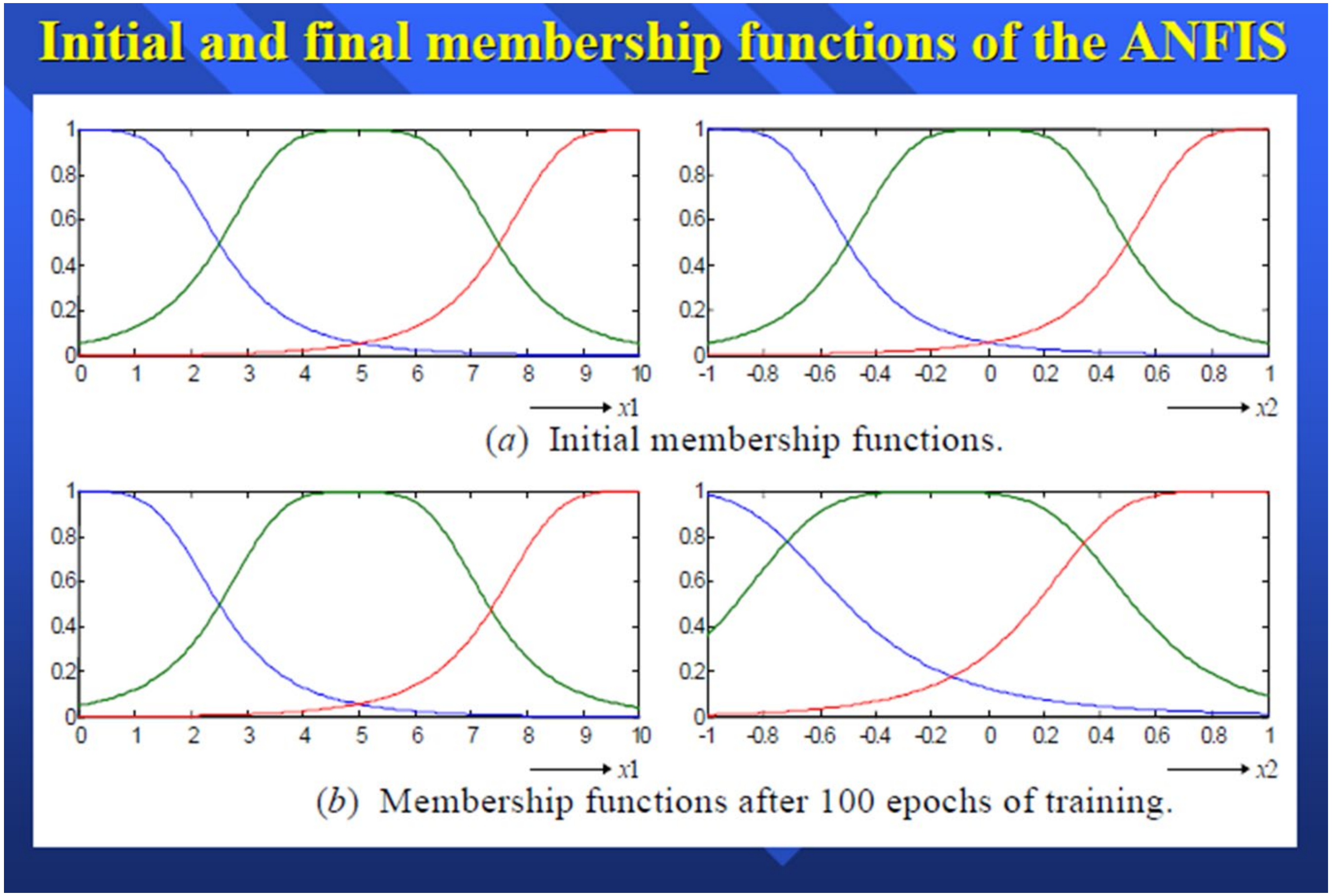


Example (matlab command line)



Learning in an ANFIS with three membership functions assigned to each input (one epoch)

Example (matlab command line)



(a) Initial membership functions.

(b) Membership functions after 100 epochs of training.

Example (ANFIS Editor GUI MatLab in Fuzzy Logic Toolbox™ help)
command: anfisedit

Chapter 5:Evolutionary and Genetic algorithms

- Intelligence can be defined as the capability of a system to adapt its behavior to ever-changing environment.

- Optimization iteratively improves the quality of solutions until an optimal, or at least feasible, solution is found.

- If, over successive generations, the organism survives, we can say that this organism is capable of learning to predict changes in its environment.

Simulation of natural evolution

- Evolution can be seen as a process leading to the maintenance of a population's ability to survive and reproduce in a specific environment. This ability is called fitness.

- The fitness, or the quantitative measure of the ability to predict environmental changes and respond adequately, can be considered as the quality that is optimized in natural life.

How is a population with increasing fitness generated?

- Let us consider a population of rabbits. Some rabbits are faster than others, and we may say that these rabbits possess superior fitness, because they have a greater chance of avoiding foxes, surviving and then breeding.

- If two parents have superior fitness, there is a good chance that a combination of their genes will produce an offspring with even higher fitness. Over time the entire population of rabbits becomes faster to meet their environmental challenges in the face of foxes.

Simulation of natural evolution

- All methods of evolutionary computation simulate natural evolution by creating a population of individuals, evaluating their fitness, generating a new population through genetic operations, and repeating this process a number of times.
- We will start with Genetic Algorithms (GAs) as most of the other evolutionary algorithms can be viewed as variations of genetic algorithms .

Genetic Algorithms

- In the early 1970s, John Holland introduced the concept of genetic algorithms.
- His aim was to make computers do what nature does.
- Holland was concerned with algorithms that manipulate strings of binary digits.
- Each artificial “chromosomes” consists of a number of “genes”, and each gene is represented by 0 or 1:

1	0	1	1	0	0	1	0	0	1	0	1	0	1	0	1

Genetic Algorithms

- Nature has an ability to adapt and learn without being told what to do.
- In other words, nature finds good chromosomes blindly. GAs do the same.
- Two mechanisms link a GA to the problem it is solving: encoding and evaluation.
- The GA uses a measure of fitness of individual chromosomes to carry out reproduction.
- As reproduction takes place, the crossover operator exchanges parts of two single chromosomes, and the mutation operator changes the gene value in some randomly chosen location of the chromosome.

Basic genetic algorithms

Step 1: Represent the problem variable domain as a chromosome of a fixed length, choose the size of a chromosome population N, the crossover probability pc around 0.7 (70%) and the mutation probability pm around 0.01(1%).

Step 2: Define a fitness function to measure the performance, or fitness, of an individual chromosome in the problem domain. The fitness function establishes the basis for selecting chromosomes that will be mated during reproduction

Step 3: Randomly generate an initial population of chromosomes of size N:

$$x_1, x_2, \ldots, x_n$$

Step 4: Calculate the fitness of each individual chromosome:

$$f(x_1), f(x_2), \ldots, f(x_n)$$

Step 5: Select a pair of chromosomes for mating from the current population. Parent chromosomes are selected with a probability related to their fitness

Step 6: Create a pair of offspring chromosomes by applying the genetic operators − crossover and mutation.

Step 7: Place the created offspring chromosomes in the new population

Step 8: Repeat Step 4 until the size of the new chromosome population becomes equal to the size of the initial population, N.

Step 9: Replace the initial (parent) chromosome population with the new (offspring) population.

Step 10: Go to Step 4, and repeat the process until the termination criterion is satisfied

Genetic algorithms

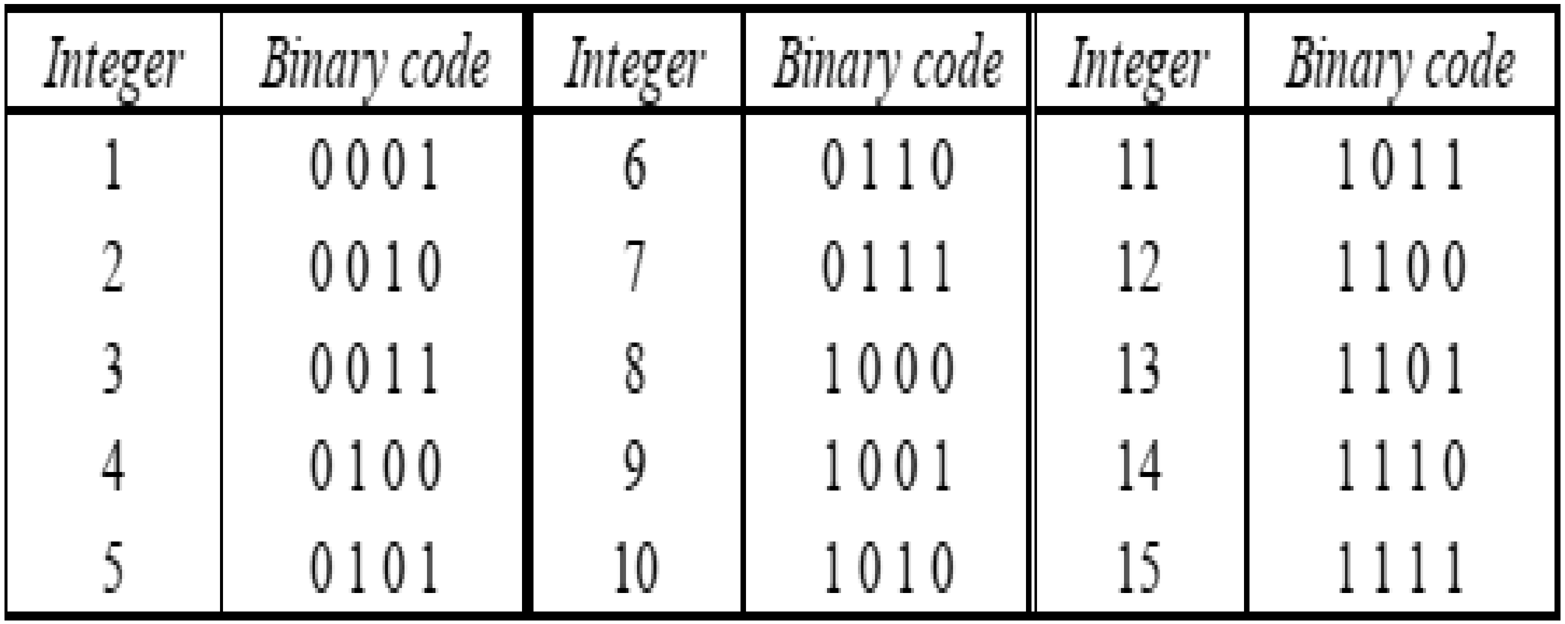
- GA represents an iterative process. Each iteration is called a generation. A typical number of generations for a simple GA can range from 50 to over 500. The entire set of generations is called a run.

- Because GAs use a stochastic search method, the fitness of a population may remain stable for a number of generations before a superior chromosome appears.

- A common practice is to terminate a GA after a specified number of generations and then examine the best chromosomes in the population. If no satisfactory solution is found, the GA is restarted

Genetic algorithms: case study

- A simple example will help us to understand how a GA works.
- Let us find the maximum value of the function $(15x - x^2)$ where parameter x is integer varies between 1 and 15.
- For simplicity, we may assume that x takes only integer values. Thus, chromosomes can be built with only four genes:



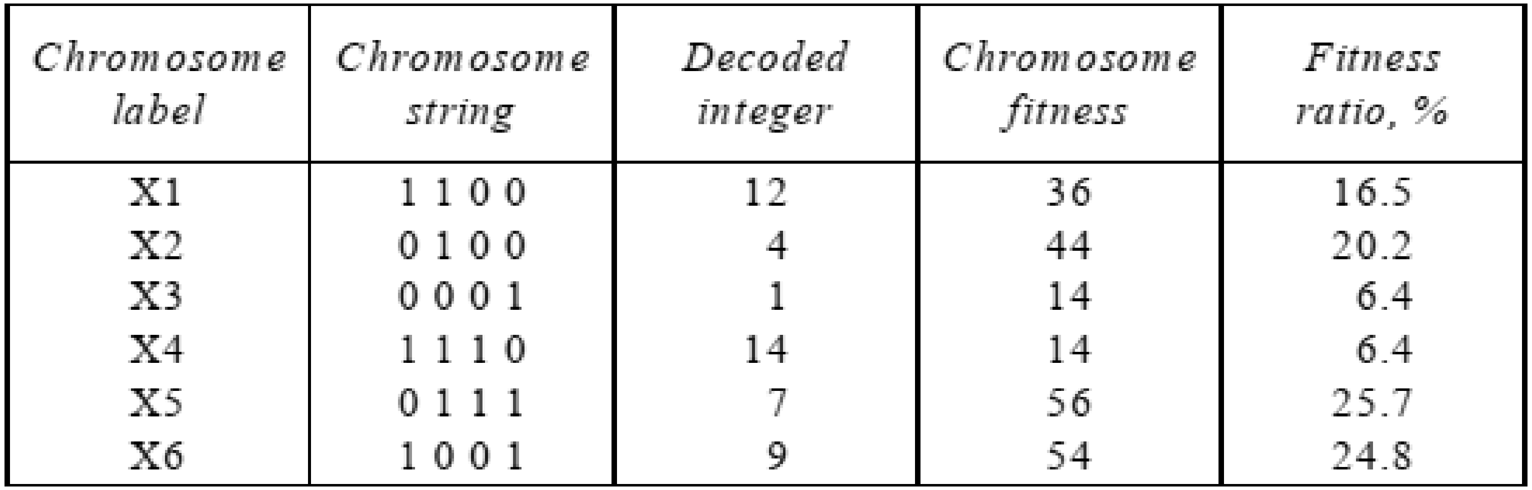
Integer	Binary code	Integer	Binary code	Integer	Binary code
1	0001	6	0110	11	1011
2	0010	7	0111	12	1100
3	0011	8	1000	13	1101
4	0100	9	1001	14	1110
5	0101	10	1010	15	1111

Genetic algorithms: case study

- Suppose that the size of the chromosome population N is 6
- The crossover probability p_c equals 0.7
- The mutation probability p_m equals 0.001
- The fitness function in our example is defined by:

$$f(x) = 15x - x^2$$

The fitness function and chromosome locations



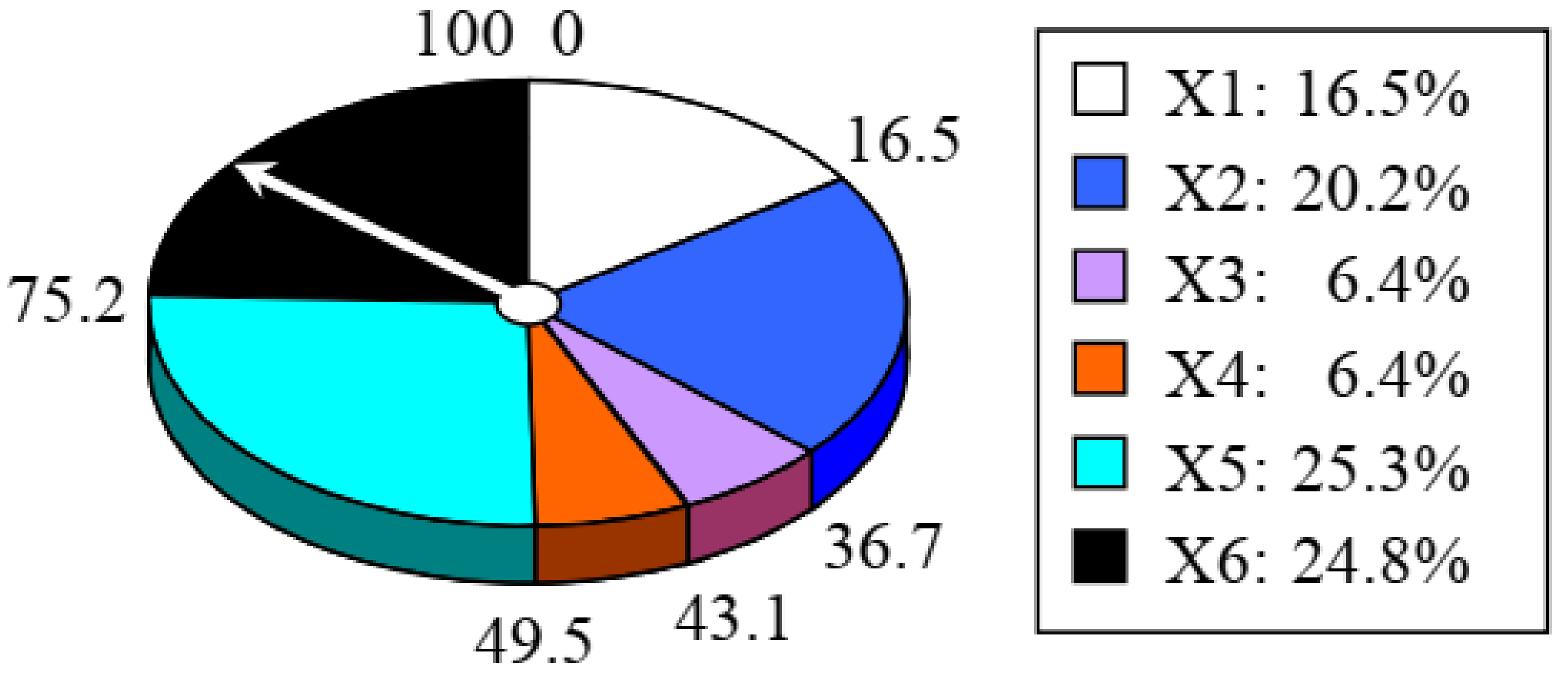
Chromosome label	*Chromosome string*	*Decoded integer*	*Chromosome fitness*	*Fitness ratio, %*
X1	1 1 0 0	12	36	16.5
X2	0 1 0 0	4	44	20.2
X3	0 0 0 1	1	14	6.4
X4	1 1 1 0	14	14	6.4
X5	0 1 1 1	7	56	25.7
X6	1 0 0 1	9	54	24.8

$$= \frac{\text{Chromosome fitness}}{\text{Sum of Chromosome fitness}}$$

- In natural selection, only the fittest species can survive, breed, and thereby pass their genes on to the next generation.
- GAs use a similar approach, but unlike nature, the size of the chromosome population remains unchanged from one generation to the next.
- The last column in Table shows the ratio of the individual chromosome’s fitness to the population’s total fitness.
- This ratio determines the chromosome’s chance of being selected for mating. The chromosome’s average fitness improves from one generation to the next.

Roulette wheel selection

The most commonly used chromosome selection techniques is the roulette wheel selection.



Crossover operator

- In our example, we have an initial population of 6 chromosomes. Thus, to establish the same population in the next generation, the roulette wheel would be spun six times.

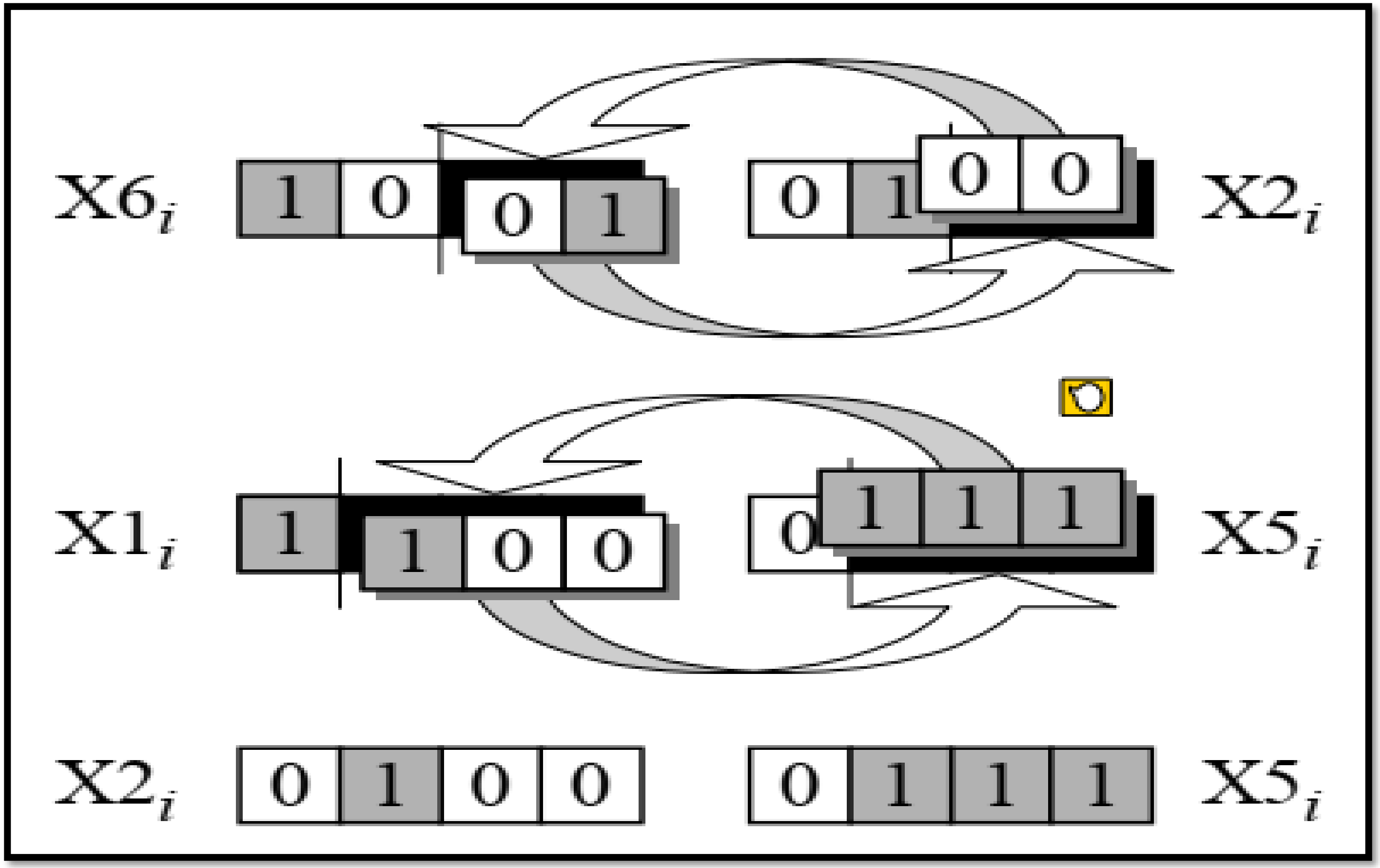
- Once a pair of parent chromosomes is selected, the crossover operator is applied.

Crossover operator

- First, the crossover operator randomly chooses a crossover point where two parent chromosomes “break”, and then exchanges the chromosome parts after that point. As a result, two new offspring are created.

- If a pair of chromosomes does not cross over, then the chromosome cloning takes place, and the offspring are created as exact copies of each parent.

Crossover



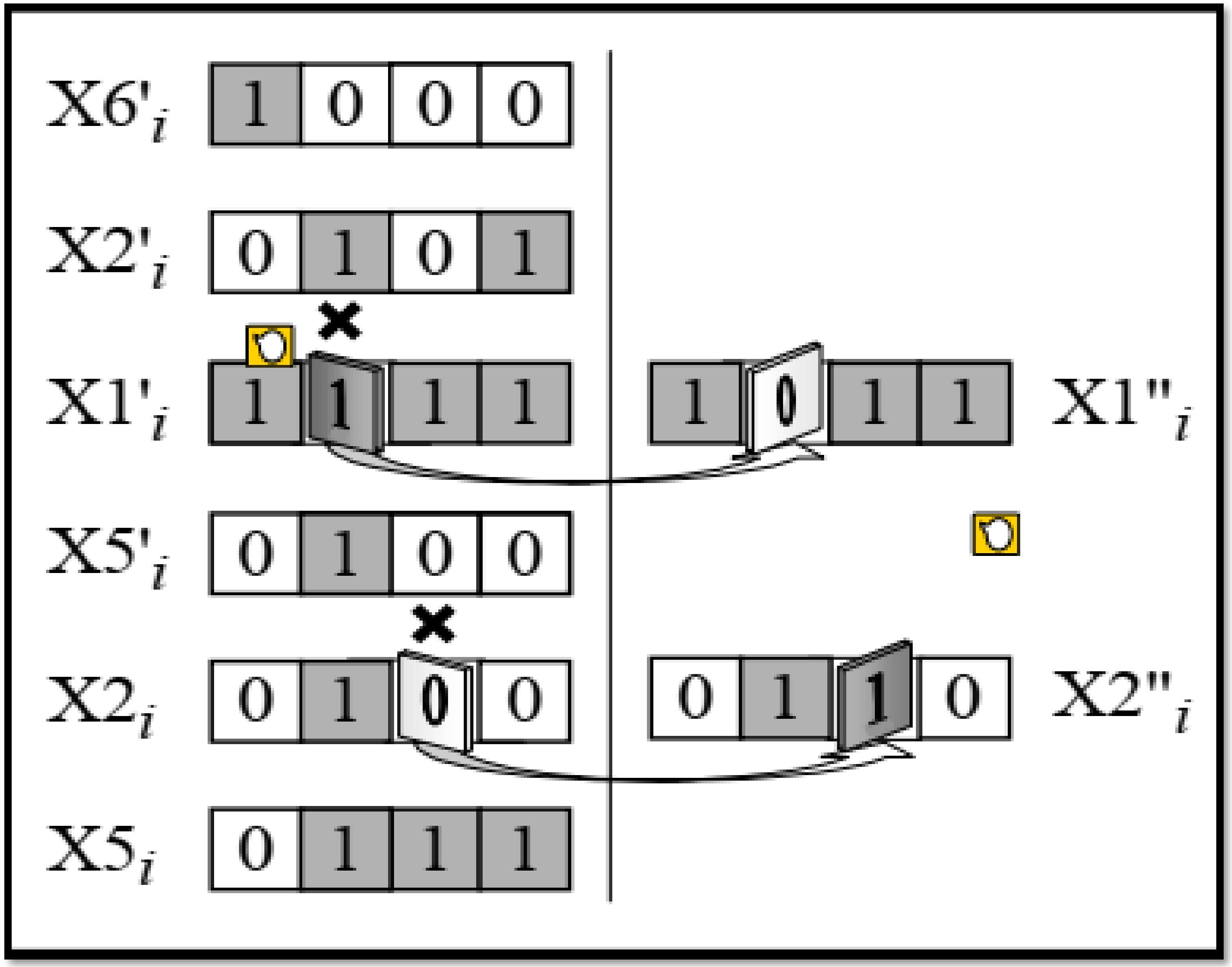
- **generate Random Number (RN1) [0-1]. if RN1 is less than Pc (0.7) then do crossover, otherwise no crossover**

- **crossover point is randomly selected (RN2) [1-3]**

Mutation operator

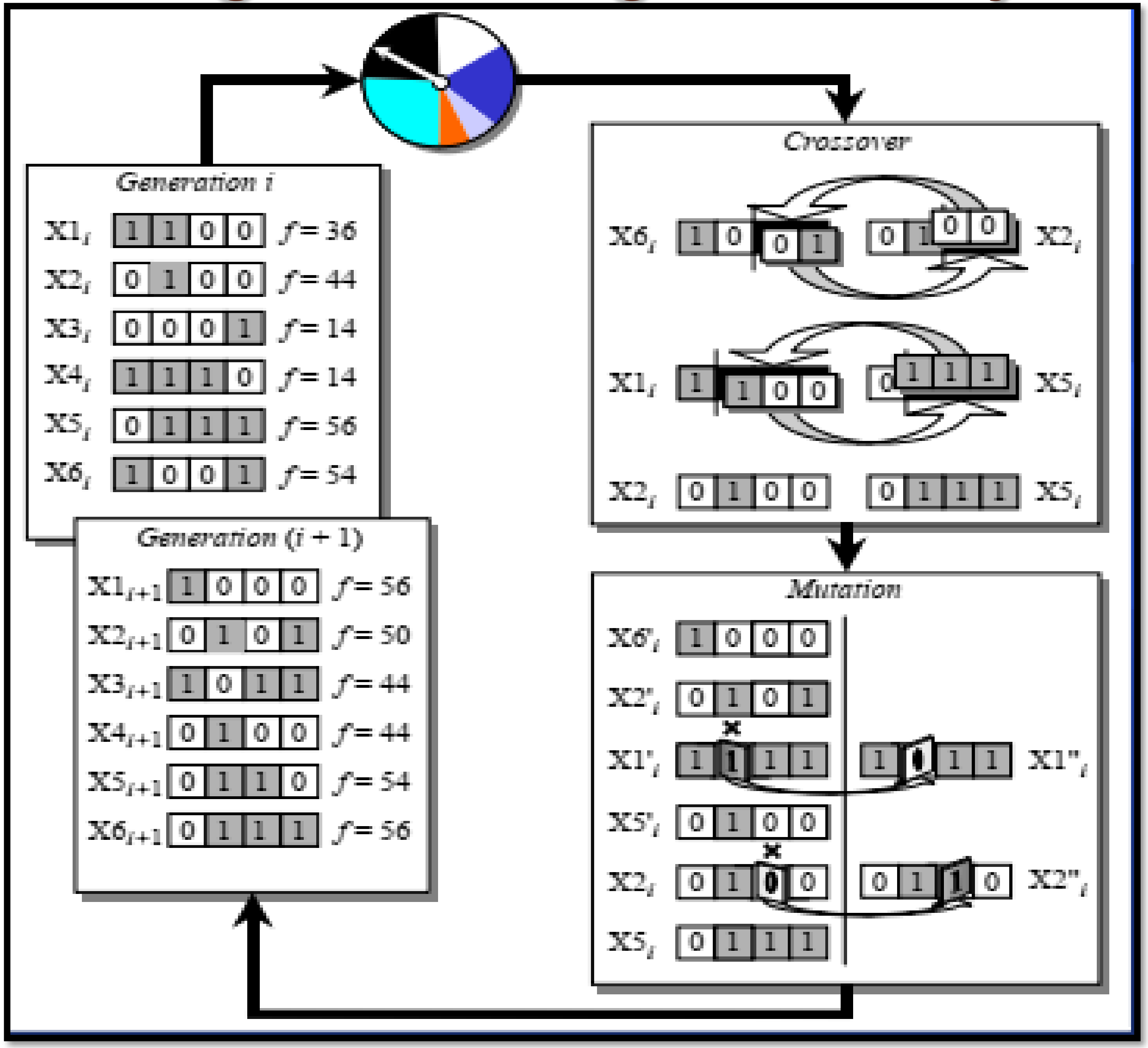
- Mutation represents a change in the gene.
- Mutation is a background operator. Its role is to provide a guarantee that the search algorithm is not trapped on a local optimum.
- The mutation operator flips a randomly selected gene in a chromosome.
- The mutation probability is quite small in nature, and is kept low for GAs, typically in the range between 0.001 and 0.01.

Mutation



- **generate Random Number (RN1) [0-1]. if RN1 is less than Pm (0.001) then do mutation, otherwise no mutation.**
- **mutation point is randomly selected** (RN2) [1-4]

The genetic algorithm cycle



Steps in the GA development

1. Specify the problem, define constraints and optimum criteria;

2. Represent the problem domain as a chromosome;

3. Define a fitness function to evaluate the chromosome performance;

4. Construct the genetic operators;

5. Run the GA and tune its parameters.

Case study: maintenance scheduling

- Maintenance scheduling problems are usually solved using a combination of search techniques and heuristics.

- These problems are complex and difficult to solve.

Case study

Scheduling of 7 units in 4 equal intervals

The problem constraints:

- The maximum loads expected during four intervals are 80, 90, 65 and 70 MW;

- Maintenance of any unit starts at the beginning of an interval and finishes at the end of the same or adjacent interval.

Case study

Scheduling of 7 units in 4 equal intervals

The problem constraints:

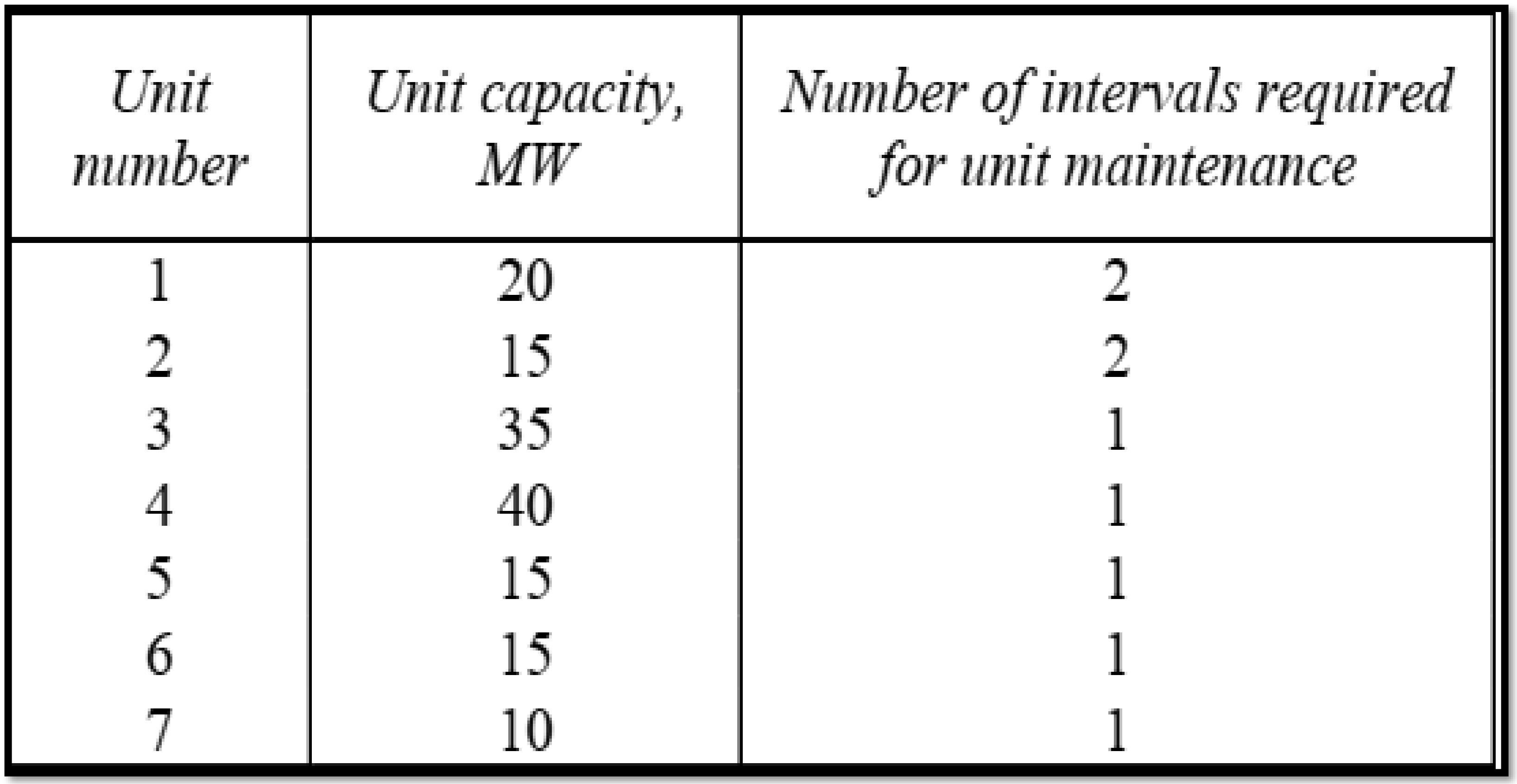
- The maintenance cannot be aborted or finished earlier than scheduled;

- The net reserve of the power system must be greater or equal to zero at any interval.

- $Netreserve = \sum capacity - \sum load$

The optimum criterion is the maximum of the net reserve at any maintenance period.

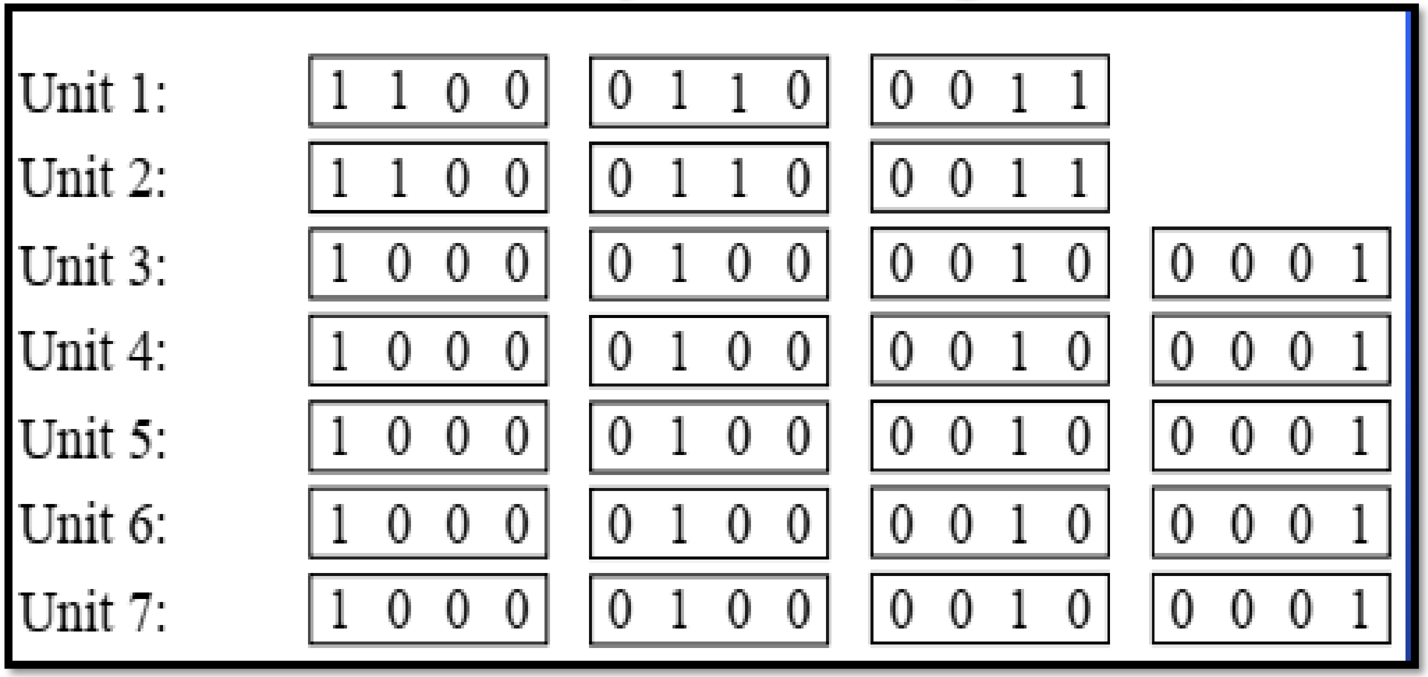
Case study
Unit data and maintenance requirements



Unit number	*Unit capacity, MW*	*Number of intervals required for unit maintenance*
1	20	2
2	15	2
3	35	1
4	40	1
5	15	1
6	15	1
7	10	1

Max capacity=150 MW

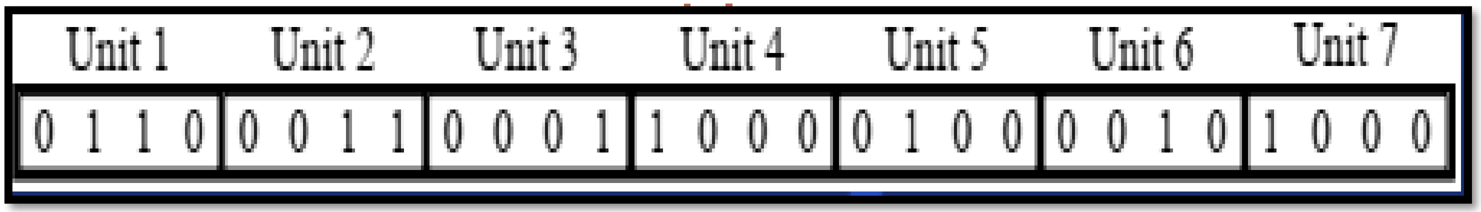
Case study Unit gene pools



Unit 1:	1 1 0 0	0 1 1 0	0 0 1 1	
Unit 2:	1 1 0 0	0 1 1 0	0 0 1 1	
Unit 3:	1 0 0 0	0 1 0 0	0 0 1 0	0 0 0 1
Unit 4:	1 0 0 0	0 1 0 0	0 0 1 0	0 0 0 1
Unit 5:	1 0 0 0	0 1 0 0	0 0 1 0	0 0 0 1
Unit 6:	1 0 0 0	0 1 0 0	0 0 1 0	0 0 0 1
Unit 7:	1 0 0 0	0 1 0 0	0 0 1 0	0 0 0 1

1: off
0: on

Chromosome for the scheduling

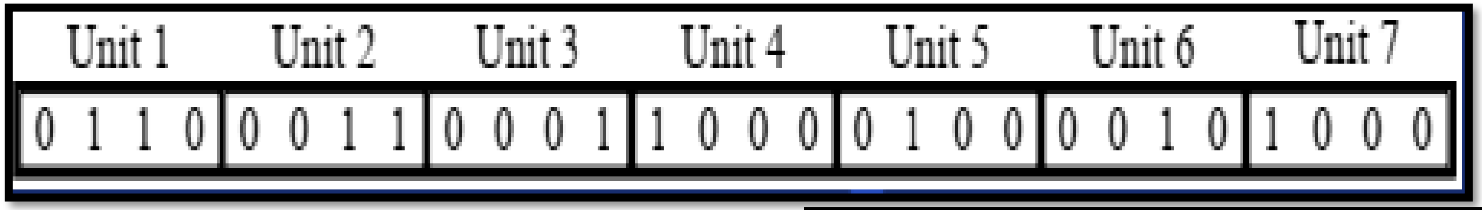


Unit 1	Unit 2	Unit 3	Unit 4	Unit 5	Unit 6	Unit 7
0 1 1 0	0 0 1 1	0 0 0 1	1 0 0 0	0 1 0 0	0 0 1 0	1 0 0 0

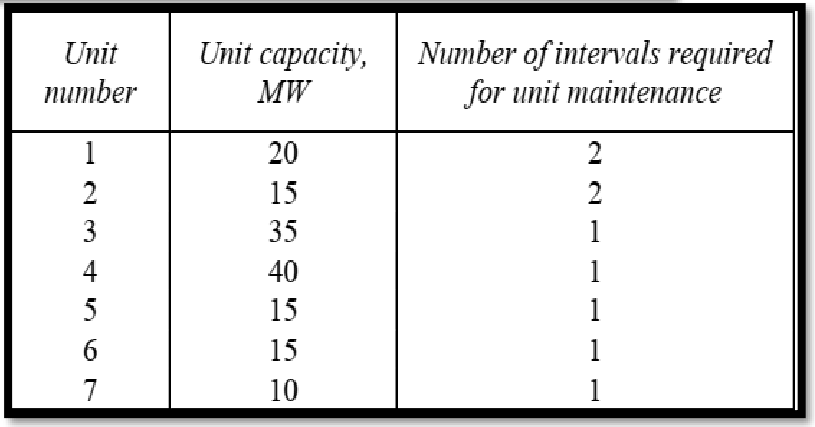
$$fitness = Net\ reserve = \sum capacity - \sum load$$

$fitness$ 1=

Calculate Fitness



Unit 1	Unit 2	Unit 3	Unit 4	Unit 5	Unit 6	Unit 7
0 1 1 0	0 0 1 1	0 0 0 1	1 0 0 0	0 1 0 0	0 0 1 0	1 0 0 0



Unit number	*Unit capacity, MW*	*Number of intervals required for unit maintenance*
1	20	2
2	15	2
3	35	1
4	40	1
5	15	1
6	15	1
7	10	1

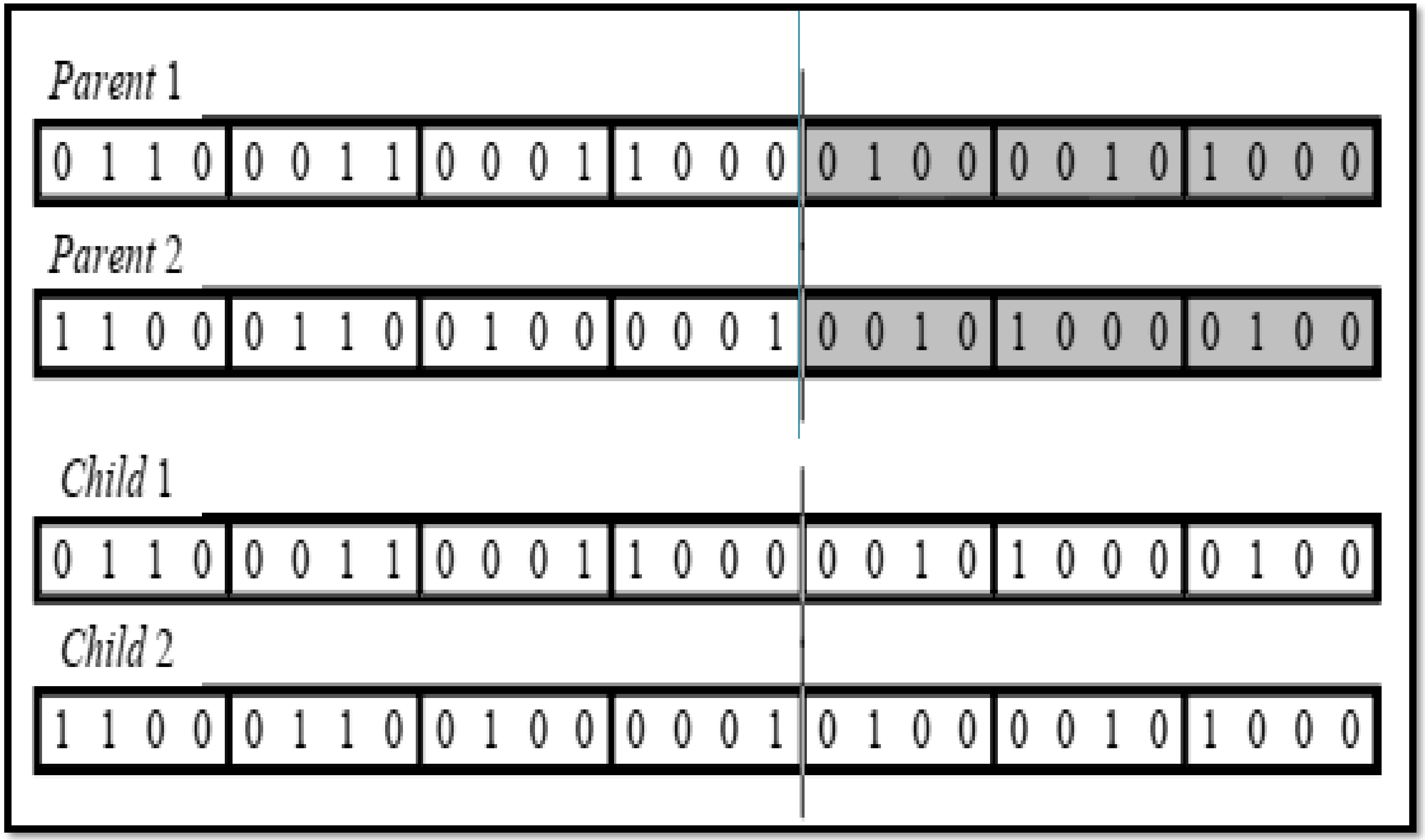
- The maximum loads expected during four intervals are 80, 90, 65 and 70 MW;

$$fitness = Net\ reserve = \sum capacity - \sum load$$

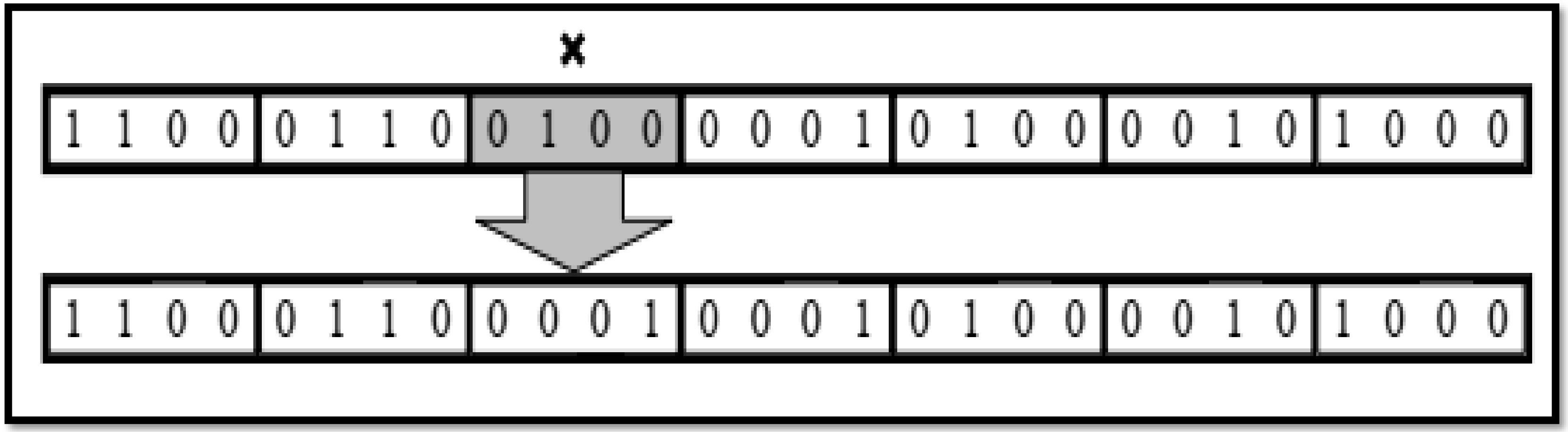
$fitness$ 1= 100-80=20
$fitness$ 2= 115-90=25
$fitness$ 3= 100-65=35
$fitness$ 4= 100-70=30

$fitness$ = 20 (minimum)

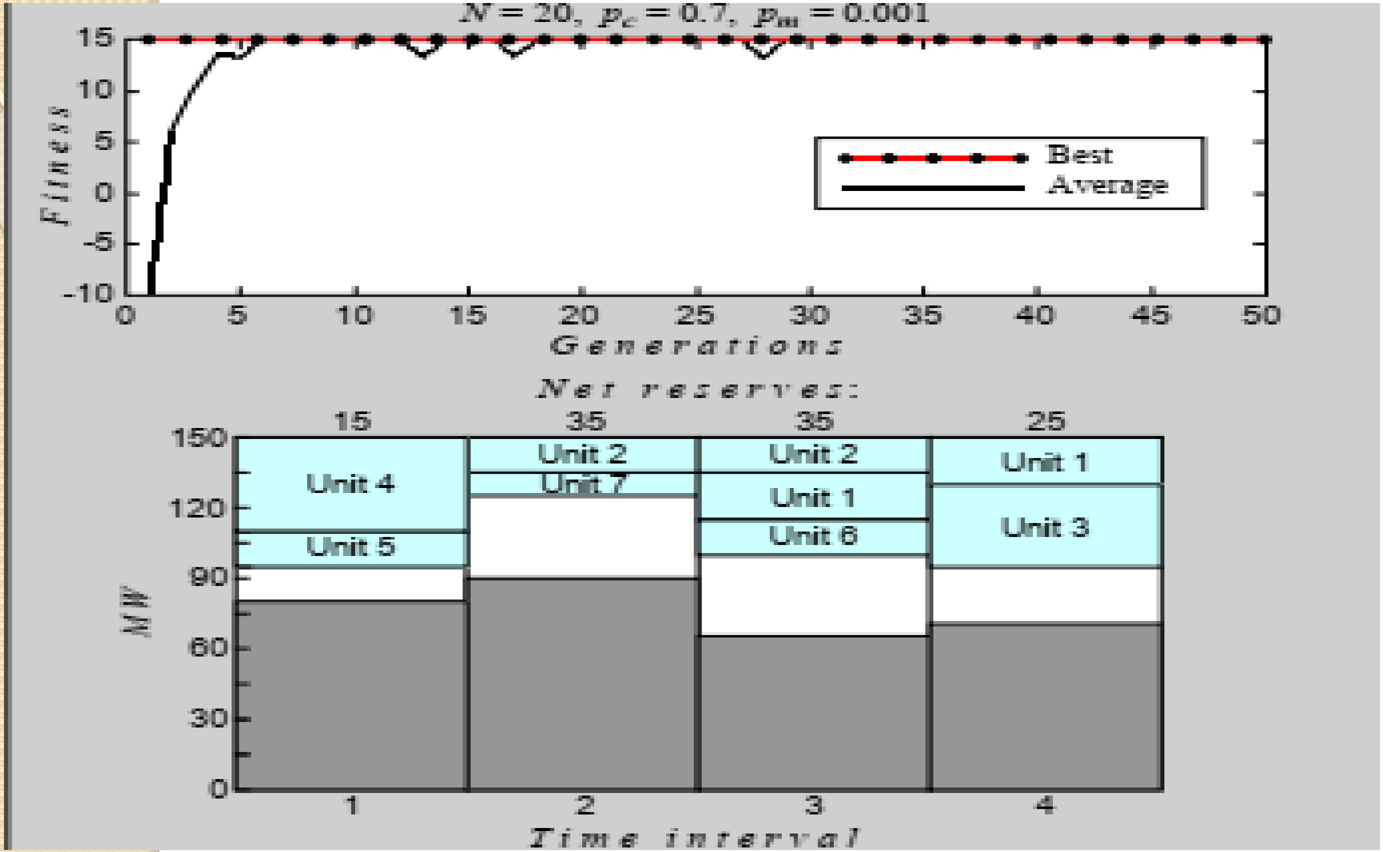
Case study
The crossover operator

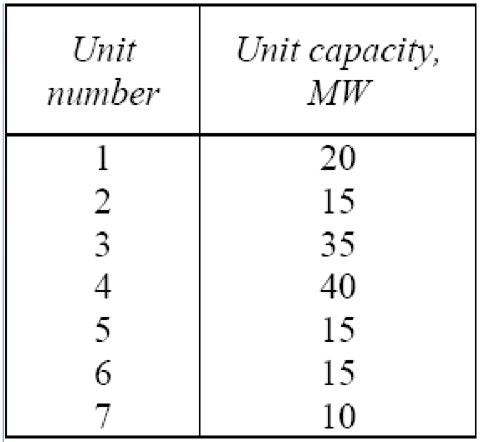


Case study
The mutation operator



Performance graphs and the best maintenance schedules created in a population of 20 chromosomes



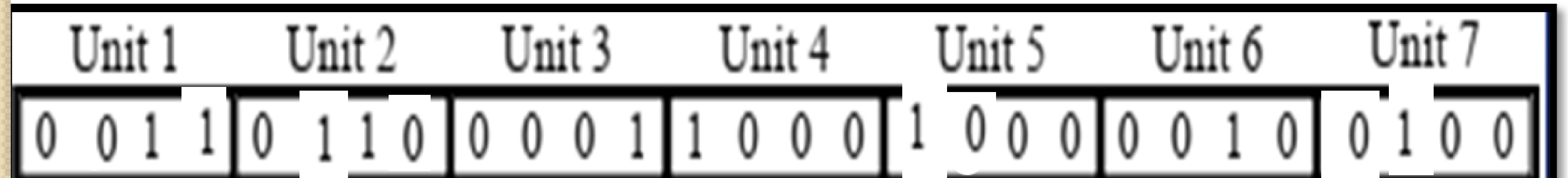


Unit number	*Unit capacity, MW*
1	20
2	15
3	35
4	40
5	15
6	15
7	10

(a) 50 generations

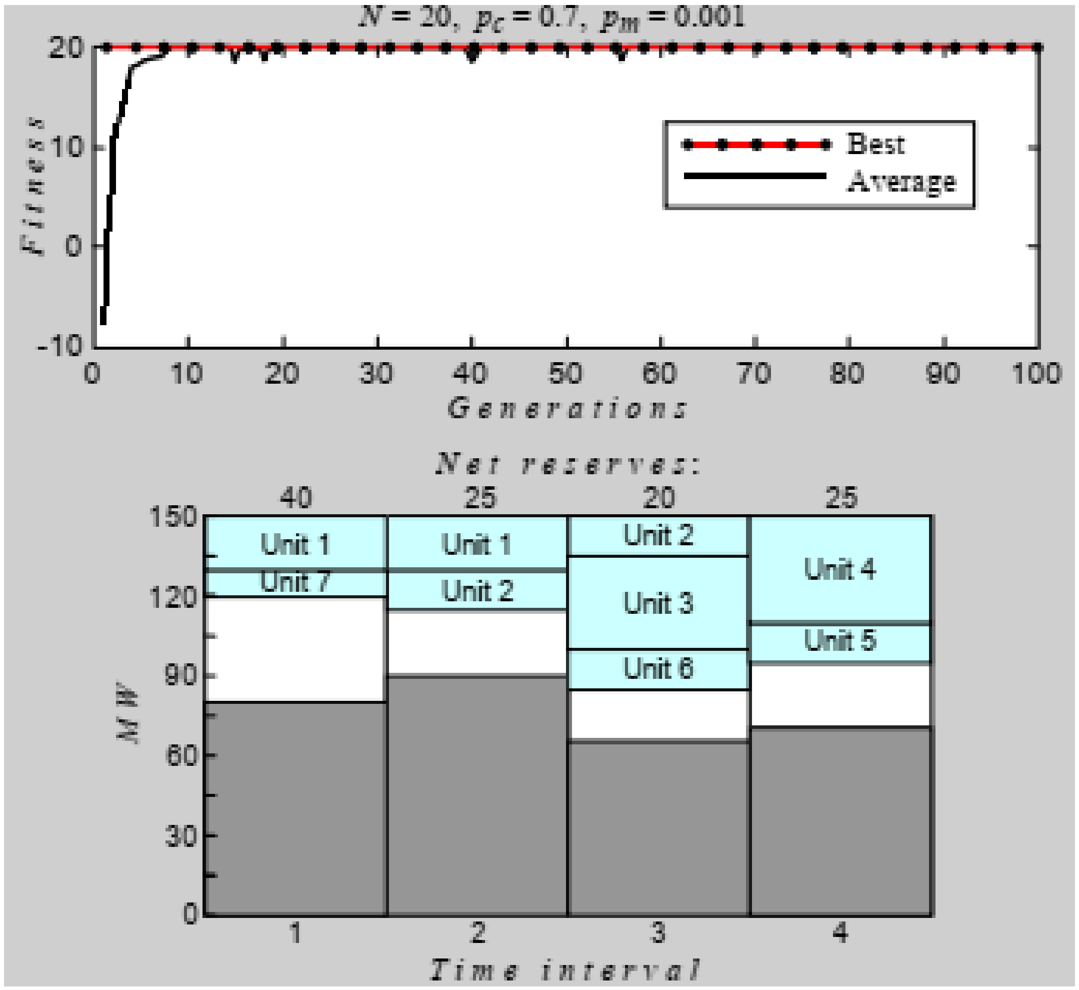
- The maximum loads during four intervals are 80, 90, 65 and 70 MW

$$fitness = Net\ reserve = \sum capacity - \sum load$$



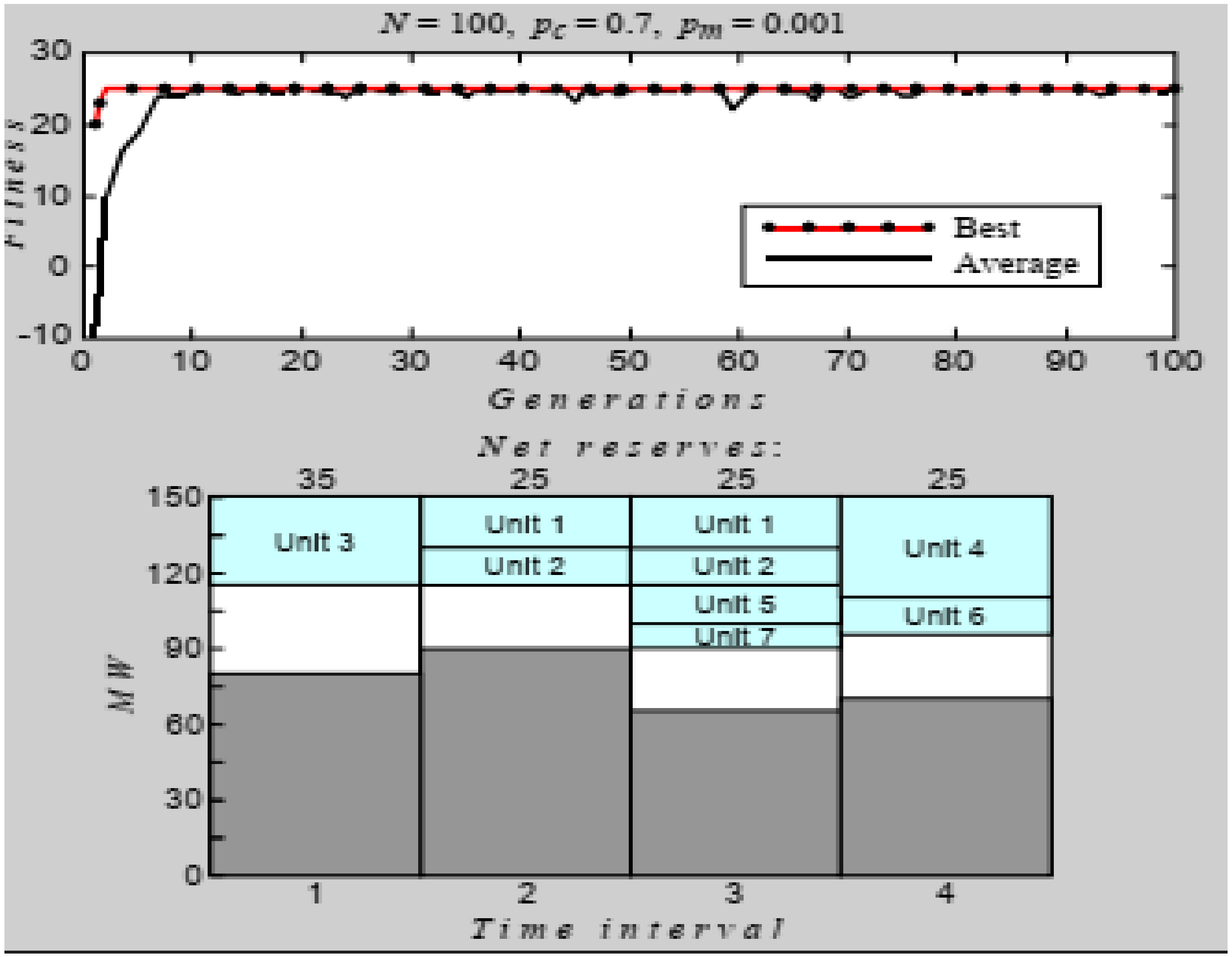
Unit 1	Unit 2	Unit 3	Unit 4	Unit 5	Unit 6	Unit 7
0 0 1 1	0 1 1 0	0 0 0 1	1 0 0 0	1 0 0 0	0 0 1 0	0 1 0 0

Performance graphs and the best maintenance schedules created in a population of 20 chromosomes



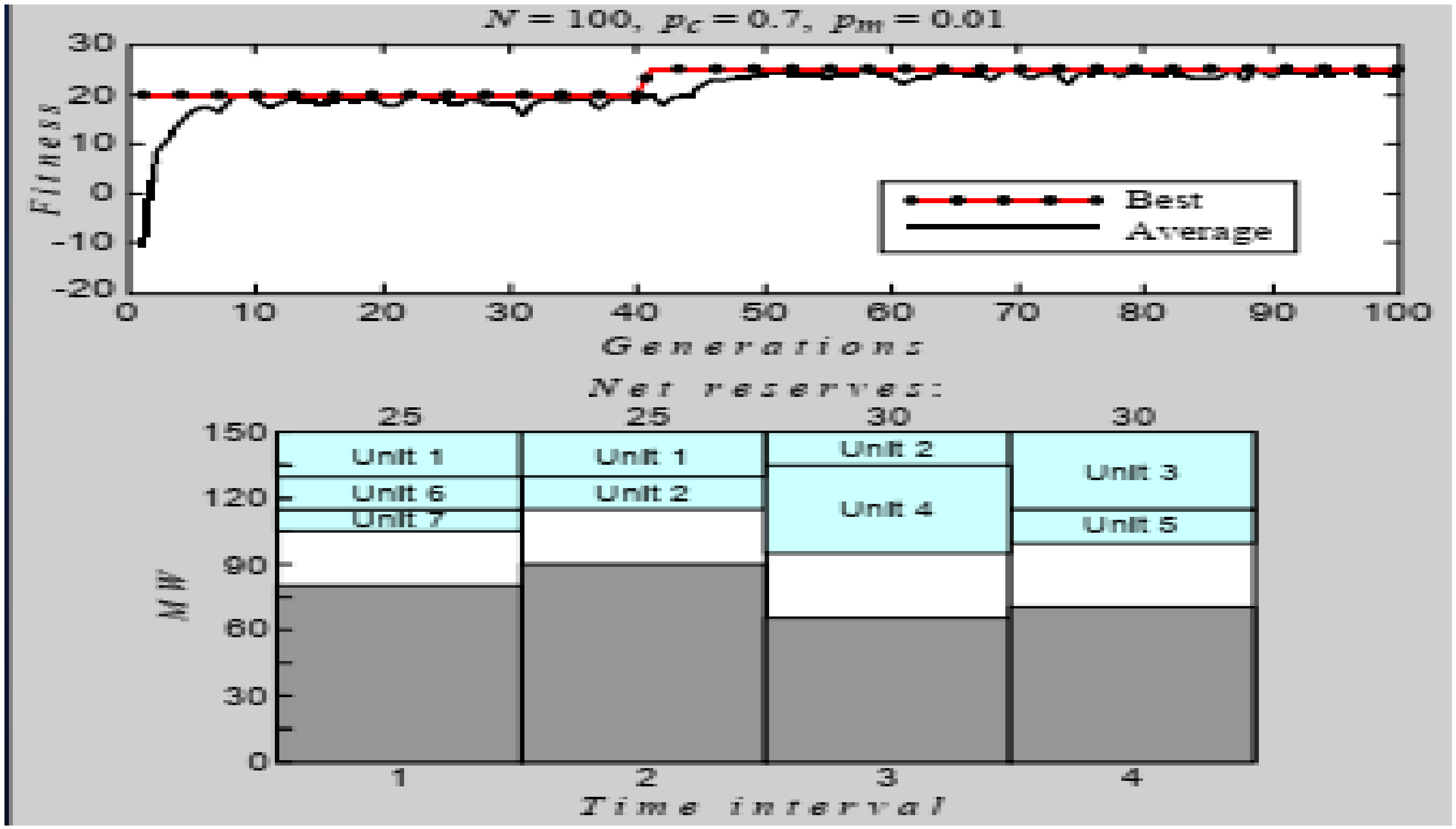
(b) 100 generations

Performance graphs and the best maintenance schedules created in a population of 100 chromosomes



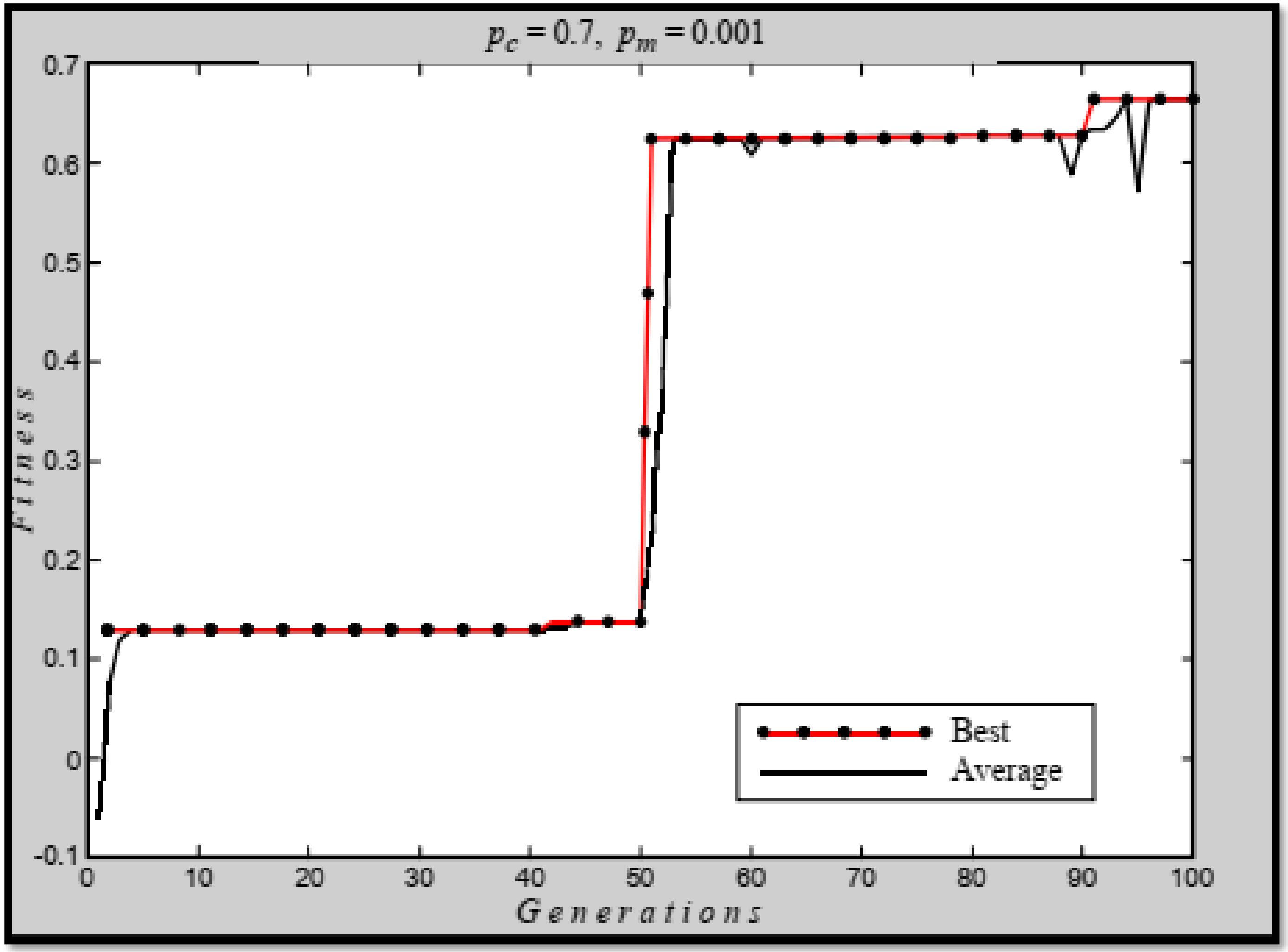
(a) Mutation rate is 0.001

Performance graphs and the best maintenance schedules created in a population of 100 chromosomes



(b) Mutation rate is 0.01

Performance graphs for 100 generations of 6 chromosomes: local maximum


$p_c = 0.7,\ p_m = 0.001$
Fitness
Generations
Best
Average