DEMOCRATIC AND POPULAR REPUBLIC OF ALGERIA
الجمهورية الجزائرية الديمقراطية الشعبية

MINISTRY OF HIGHER EDUCATION
AND SCIENTIFIC RESEARCH

HIGHER SCHOOL IN APPLIED SCIENCES
--T L E M C E N--

وزارة التعليم العالي والبحث العلمي

المدرسة العليا في العلوم التطبيقية
-تلمسان-

Higher School of Applied Sciences of Tlemcen

Second-cycle department

Program: Industrial Engineering

Course Handout

Methods of Artificial Intelligence

Prepared by

Dr. Zoulikha KOUDAD

Academic Year: 2025/2026

Preface

Artificial Intelligence (AI) plays a strategic role today in many fields of engineering and applied sciences. Whether in process optimization, autonomous decision-making, or the modeling of complex systems, AI methods offer powerful and adaptive tools. For future engineers, mastering these techniques has become essential.

This coursebook is intended for **4$^{\text{th}}$-year students in Industrial Engineering** at the **Higher School of Applied Sciences**. It has been designed as a pedagogical and structured learning support, offering a gradual understanding of the main **artificial intelligence methods** currently used, both in the industrial sector and across other technical disciplines.

The document is organized into **five chapters**:

- **Chapter 1 – Introduction to Artificial Intelligence:** This introductory chapter presents the foundations, history, and application domains of AI, laying out the essential theoretical background.
- **Chapter 2 – Machine Learning and Neural Networks:** This chapter covers fundamental machine learning algorithms such as the *perceptron*, *logistic regression*, *gradient descent*, and the *backpropagation algorithm* used in neural networks. It includes **solved exercises** to help reinforce the key concepts.
- **Chapter 3 – Fuzzy Logic and Fuzzy Inference Systems:** This chapter introduces the basics of fuzzy logic, fuzzy sets, and the construction of *fuzzy inference systems*. It also contains **exercises with solutions** to illustrate the modeling process in uncertain or imprecise environments.
- **Chapter 4 – Intelligent Agents and Multi-Agent Systems:** This chapter explores the concept of autonomous agents, the characteristics of *intelligent agents*, and the principles behind *multi-agent systems*, which are increasingly applied in distributed intelligent environments.
- **Chapter 5 – Deep Learning. Convolutional Neural Networks:**This chapter introduces Convolutional Neural Networks (CNNs), highlighting their core operations and architectures for feature extraction and learning. This chapter provides students with a solid introduction to deep learning, which they will need for their final year projects in the coming academic year.

This course material does not aim to be exhaustive but provides a solid foundation to address real-world AI problems. It can also serve as a reference for practical assignments, projects, or further studies in the field.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction to Artificial Intelligence

1.1 Definition

1.1.1 Intelligence

Intelligence is the ability of an agent to perceive its environment and take actions that maximize its chances of successfully achieving its goals, adapting to new situations when necessary. By Stuart Russell and Peter Norvig, [RN10]

Intelligence measures an agent's ability to achieve goals in a wide range of environments, including the ability to adapt to changing conditions and learn from experience. By Legg and Hutter,[LH07]

1.1.2 Definitions of Artificial Intelligence

We see eight definitions of AI, arranged along two dimensions. These definitions can be about thought processes and reasoning, or they can address behavior. They can measure success in terms of fidelity to human performance, or they can be measured against an ideal performance measure, called rationality, [RN10] .

Thinking Humanly The exciting new effort to make computers think ... machines with minds, in the full and literal sense." (Haugeland, 1985).

[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning ..(Hellman, 1978).

Thinking Rationally The study of mental faculties through the use of computational models. (Charniak et McDermott, 1985).

The study of the computations that make it possible to perceive, reason, and act.(Winston, 1992)

Acting Humanly The art of creating machines that perform functions that require intelligence when performed by people."(Kurzweil, 1990)

The study of how to make computers do things at which, at the moment, people are better. (Rich et Knight, 1991)

Acting Rationally Computational Intelligence is the study of the design of intelligent agents.(Poole et at, 1998)

Al ... is concerned with intelligent behavior in artifacts (machines). (Nilsson, 1998)

1.2 The objective of AI

The goal assigned to AI was to build algorithms that automated certain cognitive tasks.

1.2.1 Receptive functions

- To see (computer vision, example in medicine, 3D modeling of a patient's real anatomy, in automobiles, detection of road signs, in photo, recognition of faces, etc.),
- Hearing (speech recognition, analysis of noises, signals, etc.),
- To perceive (recognition of situations, analysis of behaviors, security, etc.).

1.2.2 Memory functions

- in particular learning to recognize people, situations...

1.2.3 Reasoning functions

- Reasoning functions that are often considered to be specific to humans, but some of which have been modeled since antiquity (Aristotle's Logic).
- Decision support tools, particularly in emergency situations, are an example of support functions,
- In general, all expert activities where we have been able to model human reasoning. The power and speed of the machine make it possible to implement it more quickly and on more complex problems.

1.2.4 Functions of expression and action

- To speak: speech synthesis,
- To write: automated writing,
- To act: piloting automatons, the most famous of which are automatic cars.

1.3 Historical

Aristotle (384-322 BC) was the first to formulate a precise set of laws governing the rational part of the mind. He developed an informal system of syllogisms (Reasoning composed of three propositions: the major, the minor and the conclusion.)

In the 17th century: the work of Galileo which tends to show that any serious scientific discourse is formalizable.

In 1642 Blaise Pascal replaced the calculation with a simple operation achievable by a mechanical object.

In 1738 the Grenoble engineer Jacques de Vaucanson made a mechanical duck in gilded copper, capable of drinking, eating, digesting and even defecating. A programmable mechanism also allows it to quack and splash around like a real aquatic bird.

Around 1840 Charles Babbage invented the first programmable, mechanical computer.

Naturally, the 20th century was that of computing with the automation of intellectual tasks.

In 1896 Hermann Hollerith founded the Tabulating Machine Co which became IBM.

In 1923, an electromechanical encryption machine – Enigma – was marketed. It was capable of generating and transcoding secret messages that no human was capable of deciphering. It was not until 1942 that Alan Turing created a machine capable of breaking the code of the Enigma machine, much faster than a human.

In 1927, Vannevar Bush invented the "differential analyzer", an electromechanical pre-computer weighing a hundred tons, inspired by Babbage's machine, capable of solving differential equations using gears.

1.3.1 Birth of AI

The first recognized work of AI is the creation of the first mathematical and computer model of the biological neuron (artificial neuron), in 1943 by Warren McCulloch and Walter Pitts.

Donald Hebb (1949) demonstrated a simple updating rule for the connection strength between neurons (Hebbian learning).

Two students Marvin Minsky and Dean Edmonds built the first neural network computer in 1950.

In the early 1950s, John Von Neumann and Alan Turing founded the architectures of current computers based on binary logic.

And it was in 1956 that John McCarthy and Marvin Minsky introduced the term AI in the Dartmouth College conference and defined AI as: "the construction of computer programs that engage in tasks that are, for the time being, more satisfactorily performed by human beings because they require high-level mental processes such as: perceptual learning, memory organization and critical reasoning". During the summer of 1956, the researchers (McCarthy, Minsky, Claude Shannon, Nathaniel Rochester, Trenchard More, Arthur Samuel, Ray Solomonoff and Oliver Selfridge, Allen Newell and Herbert Simon) founded the main points of the discipline of AI.

The emergence of AI had great success and enthusiasm, promises of rapid development (intelligent robots and others see the promises given by science fiction) this enthusiasm quickly fell back in the early 1960s. The machines had very little memory, making it difficult to use a computer language. Researchers predicted that in 10 years the computer could beat man at chess and demonstrate mathematical theorems, which is achieved but after 40 years.

1.3.2 The first intelligent programs

- GPS General Problem Solver (Newell and Simon) and the "physical symbol system" hypothesis which states that any intelligent system must operate by manipulating data structures composed of symbols.
- Geometry theorem prover in 1959 by Herbert Gelernter.
- Checkers in 1952 by Arthur Samuel.
- The LISP programming language, time sharing and Advice taker in 1958 by John McCarthy.
- The SAINT program in 1963 by Slagl to solve integration problems.
- The ANALOGY program in 1968 by Tom Evan to solve geometric analogy problems.
- The STUDENT program in 1967 by Daniel Bobrow to solve algebra problems.

1.3.3 Back to reality (the problems)

- The first type of problem is produced in Russian-English machine translation efforts. Example (the mind is willing but the flesh is weak) is translated into (vodka is good but the meat is rotten).
- The second type of problem is the intractability of several problems following the discovery of computational complexity theorems (trying all combinations only works with a small number of objects).
- A third difficulty is due to fundamental limitations on the basic structures used to generate intelligent behavior (perceptron, multilayer neural network, etc.).

1.3.4 1980-1990 : Expert systems

With the advent of the first microprocessors at the end of 1970, AI took off again and entered the golden age of expert systems.

In 1965, DENDRAL (expert system specialized in molecular chemistry)

In 1972, MYCIN (system specialized in the diagnosis of blood diseases and the prescription of drugs)

In May 1997, Deep Blue (IBM expert system) won the chess game against Garry Kasparov.

1.3.5 The return of neural networks

In the mid-1980s the back-propagation learning algorithm developed in 1969 by Bryson et al was reinvented and attracted great interest.

1.3.6 From 1987

AI adopts a scientific method. Hidden Markov Models (HMM) have made a good tool for speech recognition. Machine translation has been very successful. Data mining technology has spawned a new dynamic industry. With the appearance of Bayesian networks (1985) probabilities enter the field of AI (uncertain reasoning) through the front door. Other revolutions have appeared, in robotics, computer vision, knowledge representation... The 1990s saw the appearance of intelligent agents. The 2000s marked the appearance of human-level AI HLAI and artificial general intelligence AGI, and the availability of large data sets.

1.3.7 Evolution of Neural Networks: From Early Models to Transformers

After the emergence of the formal neuron in 1943, following the work of McCulloch and Pitts, the first models—such as the Perceptron (a single-layer network developed in the 1950s and 1960s)—were simple binary classifiers. The introduction of the backpropagation algorithm in the 1980s was a major breakthrough, allowing for the effective training of deeper networks (with a few layers of neurons).

The 2010s marked the rise of deep learning, which refers to neural networks with a large number of hidden layers. This success was made possible by several factors: the increase in computing power through the emergence of GPUs (Graphics Processing Units), the explosion of digital data (Big Data), and new training techniques that brought significant improvements to the backpropagation algorithm. The main architectures of deep learning include Convolutional Neural Networks (CNNs), designed for image processing, and Recurrent Neural Networks

(RNNs), designed for processing sequential data such as text and speech. Among the most well-known RNN architectures are Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU).

The year 2017 saw the emergence of Transformers, introduced by Google in the paper "Attention Is All You Need"[VSP+17]. Transformers represented a true revolution in the field of artificial intelligence. They rely on two key mechanisms: attention and positional encoding. These mechanisms provide the power behind Transformers, which quickly came to dominate the field of natural language processing. This dominance was marked by the release of BERT (Bidirectional Encoder Representations from Transformers) by Google in 2018 and GPT (Generative Pre-trained Transformer) by OpenAI in the same year. GPT-4, launched in March 2023, demonstrated exceptional capabilities in text generation, summarization, translation, and even creative writing, ushering in the era of Large Language Models (LLMs).

Finally, at the end of 2024, a Chinese AI company launched DeepSeek [DBC+24], a large language model (LLM) with a focus on open-source availability and cost efficiency.

1.4 AI approaches

1.4.1 The symbolic approach

The environment is described as precisely as possible, as well as the laws that apply to it, and it is up to the program to choose the best option. This is the approach used in expert systems or fuzzy logic for example.

1.4.2 The connectionist approach

The software is given a way to evaluate whether what it is doing is good or not, and it is left to find solutions on its own, by emergence. This approach is that of neural networks or multi-agent systems.

These two approaches are not, however, totally contradictory, and can therefore be complementary to solve certain problems: for example, we can start from a symbolic base that can be supplemented by a connectionist approach.

1.5 Application Areas

Artificial intelligence techniques, particularly neural networks and deep learning, are applied across a wide range of domains. Below are some of the key application areas:

1. **Natural Language Processing (NLP)** Applications include machine translation, web search and information retrieval, and semantic error correction and grammar checking.

2. **Pattern Recognition (PR)** that include, computer-assisted vision (CAV), facial recognition, fingerprint analysis, badge identification, barcode decoding, satellite image interpretation, medical image analysis, and voice and speech recognition.

3. **Knowledge Representation and Extraction (Data Mining)**

4. **Decision Support Systems** AI can assist in: scheduling tasks and resources, transportation management, steering and navigation systems, project and strategic planning (e.g., sports teams), and timetable generation (for schools, hospitals, etc.).

5. **Problem Solving** include mathematical optimization problems, logical reasoning tasks, constraint satisfaction problems, and heuristic-based solutions. Problem-solving techniques are essential in domains such as operations research, logistics, cybersecurity, and intelligent tutoring systems.
6. **Diagnostic Support** Used in fields such as medicine, engineering, and IT.
7. **Games** Game AI for strategy, pathfinding, and behavior modeling.
8. **Optimization** Applied in operations research, finance, and engineering design.
9. **Simulation** For training, testing, and modeling complex systems.
10. **Virtual Reality (VR)** Enhancing immersive experiences and adaptive environments.
11. **Robotics** For perception, control, navigation, and autonomous decision-making.

Chapter 2

Machine learning and neural networks

2.1 Machine learning / machine learning

An agent learns whether it improves its performance on future tasks with experience.

2.1.1 Motivation

Why program learning programs?

- it is too difficult to anticipate all the inputs to be processed correctly (adapt to future data)
- it is possible that the relationship between input and output evolves over time.
- sometimes we have no idea how to program the desired function (face recognition)

2.1.2 Types of learning

Supervised learning

In this type of learning the desired output (target) is provided explicitly. Examples: in the case of a handwritten character recognition program, the data is a set of type pairs (character image, character).

Reinforcement learning

(or semi-supervised); The learning signal corresponds only to rewards and punishments. For example

- if the agent behaves well then we give +1;
- if the agent behaves badly we give -1.

Unsupervised learning

The output is not provided with the training data, the system must decide on the output based only on the input information. This type of learning is used mostly in clustering problems. Example: to classify and group documents by looking at the characteristics of each document (class names and their number are not provided as input).

2.1.3 Formulation of supervised learning

A supervised learning problem is formulated as follows: given a training set (or learning base) of n examples;

$$D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_m)\}$$

where y_i was generated by an unknown function $y = f(x)$, f is usually given by humans. The goal of supervised learning is to discover a function h, called a model or hypothesis, that will be a good approximation of f.

$$h(x) \approx f(x)$$

A learning algorithm can therefore be seen as a function A, which is given a training set D, and which returns this function h;

$$A(D) = h$$

2.2 K-nearest neighbor algorithm

Introduced in 1968 by Cover and Hart, the k-nearest neighbor (k-NN) is an instance-based learning method, the algorithm does not involve a real learning phase, and is also called lazy learning, memory-based classification since the training examples must be present in memory during each classification, case-based classification, and example-based classification. The main idea of the algorithm is as follows: the objects of the training set are simply recorded, when a new object to be classified arrives, it would be compared to the training objects using a similarity measure. We will choose the k objects that are the closest, and the majority class will be considered for the new object.

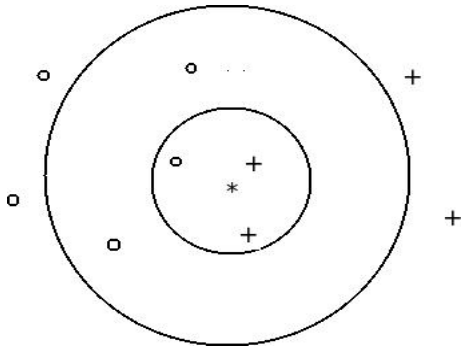


Figure 2.1: if k=3 then the element (*) is classified with the class of (+), if k=5 the element (*) is classified with the class of (∘)

2.2.1 K-PPV Algorithm

Algorithm 1 K-nearest neighbor algorithm

parameters: k:number of neighbors.
d: similarity measure (distance)
$D = (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_m)$ the training set
$Y = y_1, y_2, \ldots, y_m$ the set of classes
$\overline{x_i}$ the vector representation of $x_i, i = 1, \ldots, n$
$\overline{x}$ the vector representation of x
$N_k(x)$ the set of k nearest neighbors of x, initially empty.
for each $\overline{x_i}$ **do**
calculate $d(\overline{x_i}, \overline{x})$
endfor
$N_k(x) = argmax(d(\overline{x_i}, \overline{x}))_k$ % the k elements with maximum similarity
count the number of occurrences of each class in $N_k(x)$
assign the majority class to x

2.2.2 The training set

The training set $D = (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_m)$ with n training examples, each example is described by a set of features, and each example belongs to a class.

Example: classification of flowers The attributes taken for each flower are:

- petal length
- petal width
- petal color
- number of petals

2.2.3 Measure of similarity (or distance)

A distance (or similarity) metric must comply with the following four criteria:

- $d(x, x_i) \geq 0$ (non-negativity)
- $d(x, x_i) = 0$ iff $x = x_i$ (identity).
- $d(x_i, x) = d(x, x_i)$ (symmetry)
- $d(x, x_i) \leq d(x, z) + d(z, x_i)$ triangular inequality

One of the most used distance metrics is the Euclidean distance.

2.2.4 Euclidean distance:

Let two objects x and x_i be described respectively by the following attribute vectors $x = (x^1, x^2, ..., x^a)$, and $x_i = (x_i^1, x_i^2, ..., x_i^a)$, where a is the number of attributes or characteristics for each object.
The Euclidean distance between x and x_i is given by

$$d(x, x_i) = \sqrt{\sum_{j=1}^{a} (x^j - x_i^j)^2}$$

2.2.5 Choice of the value of k

A small value of k is useful for small learning bases and for fine structures.
A large value of k makes the classifier less sensitive to noise.
A large learning base allows a larger value of k.

2.2.6 Advantages

Fast learning.
The method is easy to understand.
High efficiency in areas where each class is represented by several prototypes with irregular boundaries (handwritten character recognition).

2.2.7 Disadvantages

Slow classification because all the examples must be reviewed for each new classification.
Memory-intensive method.
Sensitive to irrelevant attributes.
The larger the dimension (number of attributes), the larger the training set must be.

2.2.8 Improvement

Several improvements are made to the original algorithm;

- Reduce the training set based on the examples of the boundaries of each class (the class contours) by applying a cleaning algorithm.
- Partitioning the data by a tree structure according to the dimension of the problem.

2.2.9 example

handwritten character recognition.

2.2.10 Exercise

In a war game, we have: Two types of soldiers: infantrymen and knights.

Two characteristics: strength (between 0 and 20) and courage (between 0 and 20).

We have a collection of soldiers whose characteristics and type we know as shown in figure 2.2.

By applying the KNN algorithm (k nearest neighbors), determine the type of these new soldiers: Arthur (F:3,C:14), Roben(F:12,C:11), Leo(F:15, C:15), Chico(F:20, C:8).

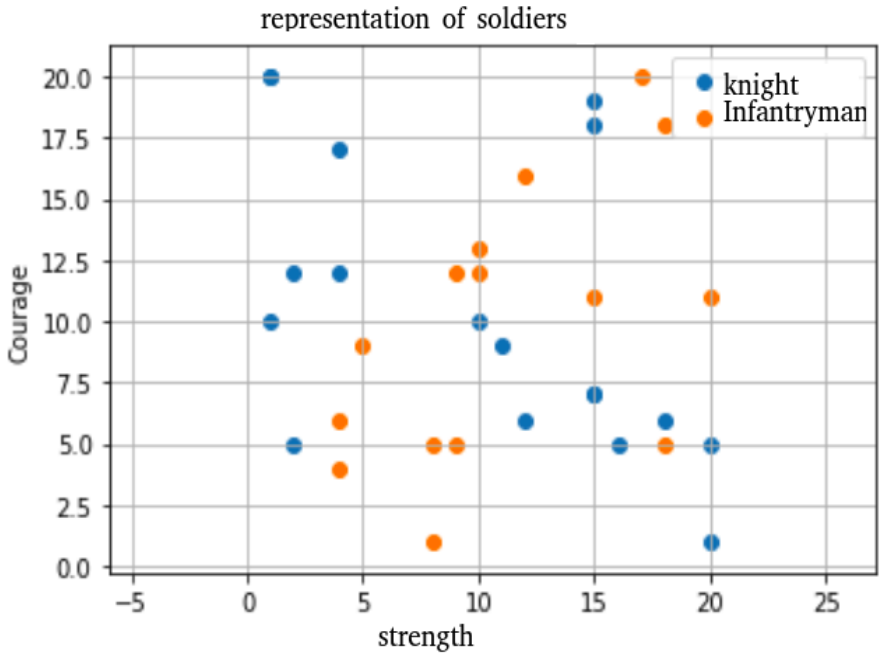


Figure 2.2: the characteristics and type of the known soldier set

2.3 Neural networks

2.3.1 Biological origin

AI researchers have been interested in the brain to be able to reproduce it and simulate human intelligence. The most important cells in the cerebral cortex are neurons (about a hundred billion in the human brain) as presented in figure 2.3. Neurons communicate with each other via impulses sent by sensors (eye, ear, skin, etc.). These signals are processed by neuronal cells that decide whether or not to send them to other neuronal cells, depending on their importance.

- The heart of the neuron is called the soma.
- Input signals pass through the dendrites.
- The output signal passes through the axon.
- The junction between two neurons is called the synapse

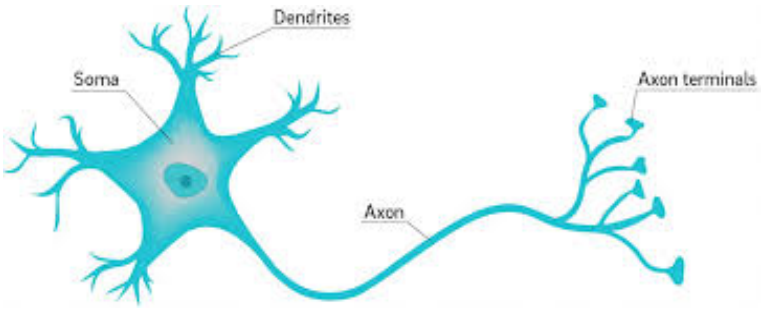


Figure 2.3: biological neuron

2.3.2 Artificial neuron

(By Mc cullogh and pitts 1943), by analogy with the biological neuron, the formal neuron, as presented in figure 2.4, is a processing unit that receives input data in the form of a vector and produces an output based on the inputs and the weights of the connections.
The aggregation function is used to calculate a single value from the corresponding inputs and weights, the threshold or bias is used to indicate when the neuron should act.
The activation function associates an output value with each aggregated value.

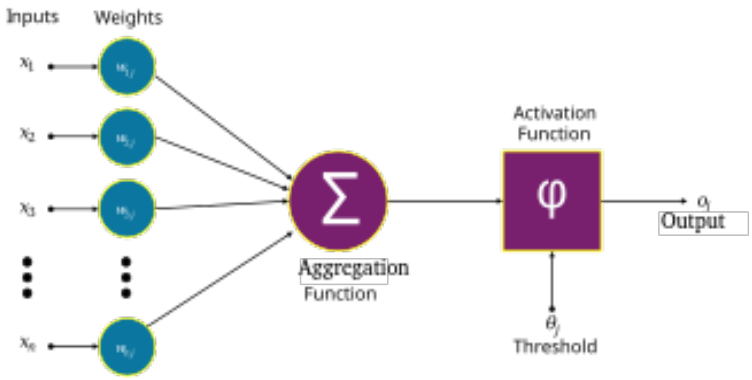


Figure 2.4: Artificial neuron

2.3.3 Aggregation function

The two most used functions are the weighted sum

$$\Sigma_{i=1}^{n} x_i w_i$$

or the distance

$$\sqrt{\Sigma_{i=1}^{n}(x_i - w_i)^2}$$

2.3.4 Activation function

Decides on the output of the neuron, this function can take several forms.
Example: sign or threshold or heavyside function

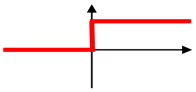
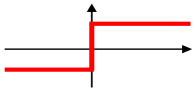
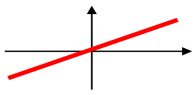
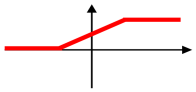
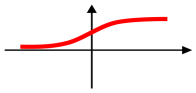
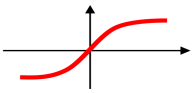
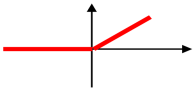
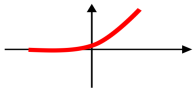
Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^{z})$	Multi-layer Neural Networks	

Figure 2.5: activation functions

2.3.5 The perceptron

Formulated in 1958 by Rosenblatt [Ros58], the perceptron is the simplest neural network . The perceptron is composed of a single layer of n neurons totally connected to a vector E of p inputs, the number of neurons corresponds to the number of outputs, the weight matrix W is of dimension $p * n$, the vector b of dimension $1 * n$ designates the n biases of the layer of neurons. Example with $n = 3$ neurons and $p = 5$ inputs is presented in figure 2.6

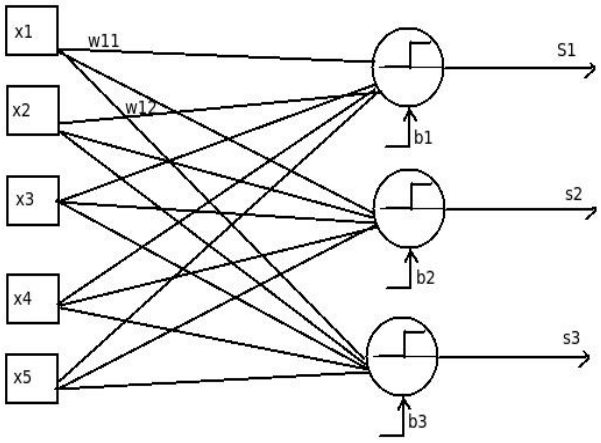


Figure 2.6: The network has 3*5 weights to adjust and 3 biases.

2.3.6 The perceptron learning rule

Learning adjusts the weights w_i and the biases b_i The biases can be treated explicitly or implicitly as an additional weight as presented in figure 2.7.

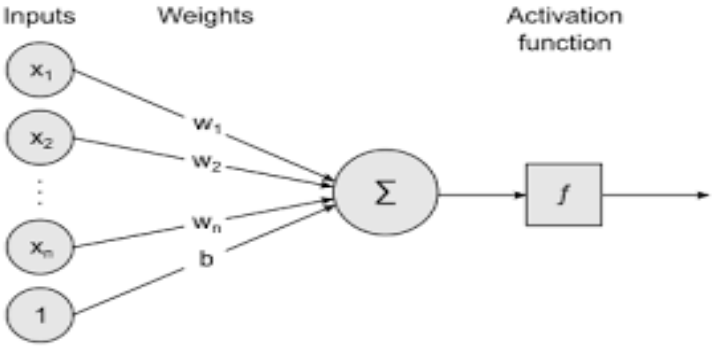


Figure 2.7: the bias treated implicitly.

The output of this perceptron is

$$s = h_w(x) = threshold(\Sigma_{i=1}^{p+1} w_i x_i) = threshold(\Sigma_{i=1}^{p} w_i x_i + b)$$

such that

$$threshold(z) = \begin{cases} 1 & if \quad z > 0 \\ 0 & otherwise \end{cases}$$

2.3.7 Perceptron algorithm

Let the learning base $D = (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_m)$. For each input vector X_i corresponds an output y_j. Each input vector X is made up of p elements according to the perceptron inputs

$$X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{pmatrix}$$

Algorithm 2 Perceptron Learning Algorithm

Input: $D = (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_m)$ % the training set.
α, W, B
repeat
  for each training sample $(x_t, y_t) \in D$ **do**
    compute $h_W(X_t) = Threshold(W.X_t)$
    if $y_t \neq h_W(X_t)$ **then**
      $W \leftarrow W + \alpha(y_t - h_W(X_t)).X_t$
    end if
  end for
until convergence (i.e., when all samples are correctly classified or a stopping criterion is met).

The algorithm 2 illustrates the different steps involved in training a perceptron. The multiplier $\alpha \in]0, 1[$ is called the learning rate, α is chosen according to the problem to be solved. If α is very small, it slows down the convergence enormously. If α is very large, it can prevent finding the optimal solution or make the learning diverge.

Note: we can increase α if the global error increases and decrease it otherwise.

2.3.8 Training set

After successful learning (error = 0) on all the learned examples, this does not provide any information on the generalization capacity of the perceptron (or other learning system). The data may have been learned by heart and the perceptron cannot decide on data outside the learning base. To solve this problem, the training set must be divided into two parts, a training part and a test part (this is called cross-validation).

2.3.9 The perceptron's "threshold" activation function

If we take the example of a single neuron with 2 inputs and 1 output, the output will be calculated as:

$$h_w(x) = \begin{cases} 1 & if \quad w_1x_1 + w_2x_2 + b > 0 \\ 0 & otherwise \end{cases}$$

The output of the network can take only two values 0 and 1, there exists in the input space a boundary delimiting the two regions corresponding to the two output classes, this boundary corresponds to the condition $w_1x_1 + w_2x_2 + b = 0$, which represents the equation of a straight line, in fact, the vector $W = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$ is perpendicular to this straight line.
If it is a 3-input problem (3D) the separating surface is a plan.
With n neurons, hence n classes, we will have n distinct boundaries.

2.3.10 Perceptron Properties

- If the examples are linearly separable, the perceptron is guaranteed to converge to a solution with zero error.

  Perceptron Convergence Theorem (Minsky and Papert 1969) If there exists a set of weights for the perceptron to provide the desired response to all input examples, then the perceptron learning rule will, in a finite number of iterations, result in a set of

weights that will provide the desired responses.

- The set of solution weights is not unique.
- The weights are only updated when an input stimulus is incorrectly classified.
- If the problem is not perfectly linearly separable, the perceptron can go to a solution with the smallest possible error

2.3.11 Example of problems

The XOR function is not linearly separable ,see figure 2.8, two neurons located on two levels (2 layers) are necessary to realize this function, it is not a perceptron. The "and" and the logical "or" are linearly separable.

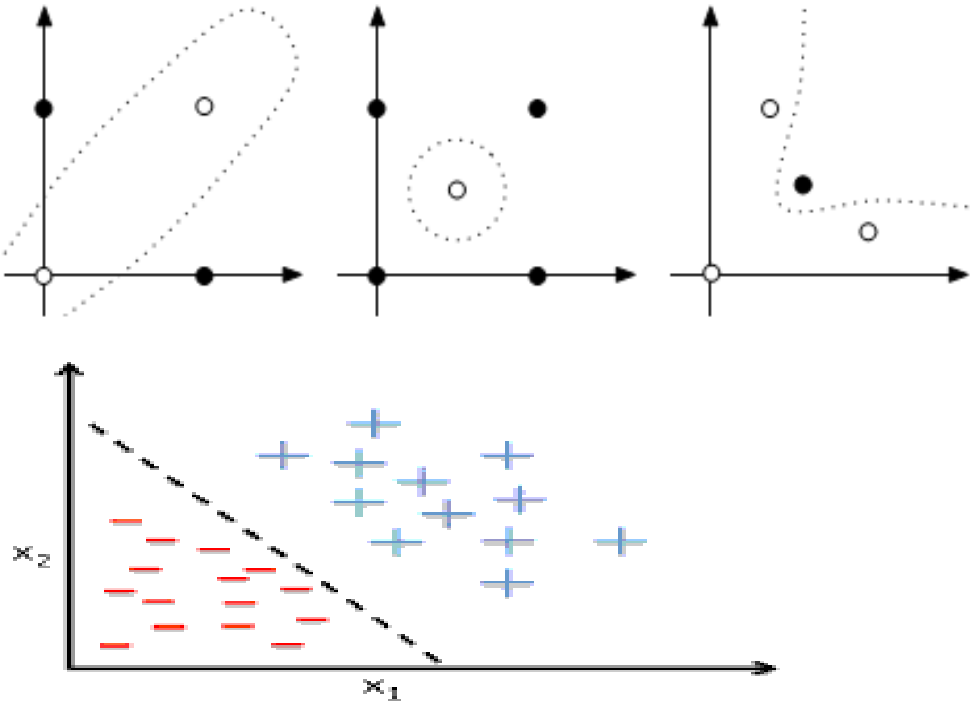


Figure 2.8: The first exemple illustrate XOR problem, the first three examples are not linearly separable, no straight line can separate the two classes,the last example illustrate a linear decision boundary for binary classification.

2.3.12 Exercise

We have a classification problem of two classes, class 1 and class 0.

Let the learning base contain 3 examples as follows:

$$D = \left\{ \left(X_1 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, y_1 = 1 \right), \left(X_2 = \begin{pmatrix} -1 \\ 2 \end{pmatrix}, y_2 = 0 \right), \left(X_3 = \begin{pmatrix} 0 \\ -2 \end{pmatrix}, y_3 = 0 \right) \right\}$$

We initialize $b = 0.5$, $\alpha = 0.1$ fixed, and $W = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$.

2.3.13 Solution

Number of inputs

The input vectors (X_i) contain 2 elements each, so the number of inputs of the perceptron=2.

Number of outputs/neurons

This is a 2 class classification, so only one neuron is enough and only one output.
If the output==1, it is the first class. If output==0, it is the second class.

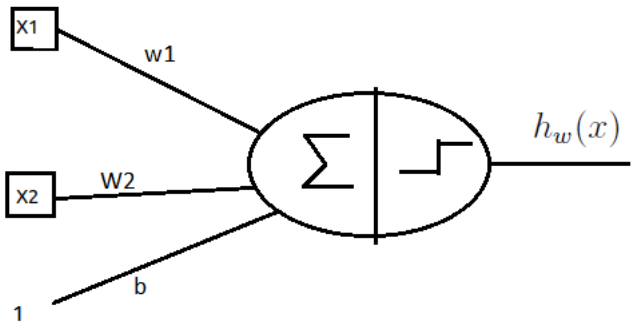


Figure 2.9: Perceptron structure

1^{st} Iteration : example 1 of the base

$$X_1 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, y_1 = 1,\ b = 0.5,\ W = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Calculating the output h

$$\begin{aligned} h_W(X_1) &= threshold\,(w_1.x_1 + w_2.x_2 + b) \\ &= threshold\,(0 * 1 + 0 * 2 + 0.5) \\ &= threshold\,(0.5) \\ &= 1 \end{aligned}$$

.

Note that $h_W(X_1) = 1$ and $y_1 = 1$. The output of the perceptron is the same as the desired output, so no learning error, and no weight updates.

1^{st} Iteration : example 2 of the base

$$X_2 = \begin{pmatrix} -1 \\ 2 \end{pmatrix}, y_2 = 0,\ b = 0.5,\ W = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Calculation of output h

$$\begin{aligned} h_W(X_2) &= threshold\,(w_1.x_1 + w_2.x_2 + b) \\ &= threshold\,(0 * -1 + 0 * 2 + 0.5) \\ &= threshold\,(0.5) \\ &= 1 \end{aligned}$$

We notice that $h_W(X_2) = 1$ and $y_2 = 0$. The output of the perceptron is not the same as the desired output, so there is a classification error which requires an update of the weights.

Weight updates :

Weights are updated using the perceptron learning rule which is as follows (in victorial format) :

$$W = W + \alpha (y - h(X_2)) X_2$$

We can write it in linear form, for i=1,2.

$$w_i = w_i + \alpha (y - h(X_2)) x_i$$

The same rule applies to the bias considering $x_i = 1$ as follows:

$$b = b + \alpha (y - h(X_2)) * 1.$$

We begin;

$$\begin{aligned} W &= W + \alpha (y - h(X_2)) X_2 \\ &= \begin{pmatrix} 0 \\ 0 \end{pmatrix} + 0.1 (-1) \begin{pmatrix} -1 \\ 2 \end{pmatrix} \\ &= \begin{pmatrix} 0.1 \\ -0.2 \end{pmatrix} \end{aligned}$$

We could have calculated each weight separately by the second formula:

$$\begin{aligned} w_1 &= w_1 + \alpha (y - h(X_2)) x_1 \\ &= 0 + 0.1 (-1) (-1) \\ &= 0.1 \end{aligned}$$

$$\begin{aligned} w_2 &= w_2 + \alpha (y - h(X_2)) x_2 \\ &= 0 + 0.1 (-1) (2) \\ &= -0.2 \end{aligned}$$

Not forgetting the bias update

$$\begin{aligned} b &= b + \alpha (y - h(X_2)) \\ &= 0.5 + 0.1 (-1) \\ &= 0.4 \end{aligned}$$

1^{st} Iteration : example 3 of the base

$X_3 = \begin{pmatrix} 0 \\ -2 \end{pmatrix}, y_3 = 0,\ b = 0.4,\ W = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 0.1 \\ -0.2 \end{pmatrix}$

Calculation of output h

$$\begin{aligned} h_W(X_3) &= threshold(w_1.x_1 + w_2.x_2 + b) \\ &= threshold(0.1 * 0 + (-0.2) * (-2) + 0.4) \\ &= threshold(0.8) \\ &= 1 \end{aligned}$$

We notice that $h_W(X_3) = 1$ et $y_3 = 0$. The output of the perceptron is not the same as the desired output, so there is a classification error which requires an update of the weights.

Update of the weights:

We will use the matrix form directly,

$$\begin{aligned} W &= W + \alpha(y - h(X_3))X_3 \\ &= \begin{pmatrix} 0.1 \\ -0.2 \end{pmatrix} + 0.1(-1)\begin{pmatrix} 0 \\ -2 \end{pmatrix} \\ &= \begin{pmatrix} 0.1 + 0 \\ -0.2 + 0.2 \end{pmatrix} \\ &= \begin{pmatrix} 0.1 \\ 0 \end{pmatrix} \end{aligned}$$

And for the bias:

$$\begin{aligned} b &= b + \alpha(y - h(X_3)) \\ &= 0.4 + 0.1(-1) \\ &= 0.3 \end{aligned}$$

As long as there are errors and weight updates, we continue the iterations, we stop when the desired output will be equal to the parceptron output for all the examples in the database.

****

2^{nd} Iteration : example 1 of the base

$X_1 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, y_1 = 1,\ b = 0.3,\ W = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 0.1 \\ 0 \end{pmatrix}$

In the following parts of the exercise, only the results of each step will be presented

Results

$X_1 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, y_1 = 1,\ b = 0.3,\ W = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 0.1 \\ 0 \end{pmatrix}$

2^{nd} Iteration : Results of the example 2 of the base

$X_2, w_1 = 0.2, w_2 = -0.2, b = 0.2$

2^{nd} Iteration : Results of the example 3 of the base

$X_3, w_1 = 0.2, w_2 = O, b = 0.1$

3^{th} Iteration : Results

- for X_3 : $w_1 = 0.2, w_2 = 0.2, b = 0$

4^{th} Iteration : Results

- for X_2 : $w_1 = 0.3, w_2 = 0, b = -0.1$

5^{th} Iteration : Results

No error; $w_1 = 0.3, w_2 = 0, b = -0.1$

Interpretation

The condition that separates the two which gives classes:

$$w_1.x_1 + w_2.x_2 + b > 0$$

for X_1, and

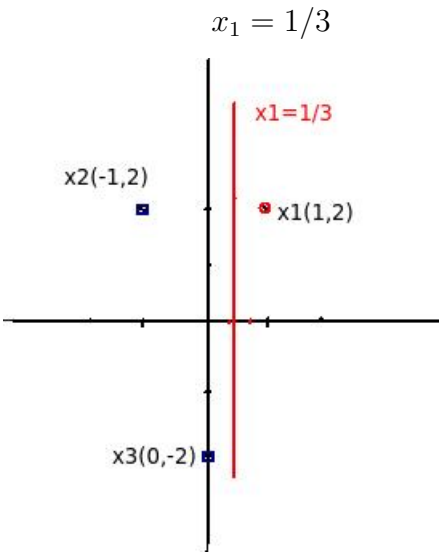
$$w_1.x_1 + w_2.x_2 + b < 0$$

for X_2 and X_3. The equation

$$w_1.x_1 + w_2.x_2 + b = 0$$

becomes:

$$O.3x_1 + 0.x_2 - 0.1 = 0$$



2.4 Partial derivative and gradient

2.4.1 Mathematical Reminder Derivatives

$$f^{'}(x) = \lim_{\Delta \to 0} \frac{f(x+\Delta) - f(x)}{\Delta}$$

if $f^{'}$ is positive f increases
if $f^{'}$ is negative f decreases
we note

$$\frac{\partial f}{\partial x}$$

$$\frac{\partial a}{\partial x} = 0$$

$$\frac{\partial x^n}{\partial x} = n x^{n-1}$$

$$\frac{\partial \ln(x)}{\partial x} = \frac{1}{x}$$

$$\frac{\partial \exp(x)}{\partial x} = \exp(x)$$

The derivative of a composite function

$$f \circ g(x)^{'} = f(g(x))^{'} = f^{'}(g(x)).g^{'}(x)$$

$$((f(x))^n)^{'} = n(f(x))^{n-1}.f^{'}(x)$$
$$(\exp(f(x)))^{'} = \exp(f(x)).f^{'}(x)$$
$$(\ln(f(x)))^{'} = \frac{1}{f(x)}.f^{'}(x)$$

****

$$(f(x) + g(x))^{'} = f^{'}(x) + g^{'}(x)$$

$$(f(x) * g(x))^{'} = f^{'}(x).g(x) + f(x).g^{'}(x)$$

$$(\frac{f(x)}{g(x)})^{'} = \frac{f^{'}(x)}{g(x)} - \frac{f(x)}{g(x)^2}.g^{'}(x)$$

examples $f(x) = \exp(\frac{x^3}{3})$
$f(x)^{'} = \exp(\frac{x^3}{3}).3.\frac{x^2}{3} = x^2.\exp(\frac{x^3}{3})$
$f(x) = x\exp(x)$

2.4.2 Partial derivative

Let $f(x, y)$ be a function of two variables x and y

The partial derivative of f with respect to x is the derivative of this function f with respect to x, while keeping y constant. It is denoted as $\frac{\partial f}{\partial x}$ and is given by:

$$\frac{\partial f(x,y)}{\partial x} = \lim_{\Delta \to 0} \frac{f(x+\Delta, y) - f(x,y)}{\Delta}$$

$$\frac{\partial f(x,y)}{\partial y} = \lim_{\Delta \to 0} \frac{f(x, y+\Delta) - f(x,y)}{\Delta}$$

Example 1 : In the perceptron, we are interested in the variations of the perceptron's outputs with respect to each weight w_{ij}, which are the variables.

Example 2 $: f(x,y) = x^2 \sin(y) - y$
$f(x,y,z) = \frac{\exp(y)}{\exp(x)+\exp(y)+\exp(z)}$

2.4.3 The Gradient

The gradient ∇f of a function f is the vector containing the partial derivatives of f with respect to all variables.

$$\nabla f(x,y,z) = \left[\frac{\partial f(x,y,z)}{\partial x}, \frac{\partial f(x,y,z)}{\partial y}, \frac{\partial f(x,y,z)}{\partial z}\right]$$

2.4.4 Chain differentiation (Chain rule)

The chain rule states that the derivative of a composite function is the derivative of the outer function evaluated at the inner function, multiplied by the derivative of the inner function.
If we can express a function $f(x)$ in terms of an intermediate result $g(x)$, then its derivative can be computed using the chain rule:

$$\frac{\partial f(x)}{\partial x} = \frac{\partial f(x)}{\partial g(x)} . \frac{\partial g(x)}{\partial x}$$

If we can express a function $f(x)$ in terms of intermediate results $g_i(x)$, then we can write its partial derivative using the chain rule:

$$\frac{\partial f(x)}{\partial x} = \sum_i \frac{\partial f(x)}{\partial g_i(x)} . \frac{\partial g_i(x)}{\partial x}$$

For example; $f(x) = \ln(x^2 + 2x + 1)$
In other cases, $f(x)$ can be written in terms of multiple intermediate functions, example:

$$f(x) = 4\exp(x) + 3(1+x)^3$$

We consider $g_1(x) = \exp(x)$, and $g_2(x) = 1 + x$.
So we can write $f(x) = 4g_1(x) + 3g_2(x)^3$
we can calculate the partial derivative as follows:

$$\frac{\partial f(x)}{\partial g_1(x)} = 4, \qquad \frac{\partial g_1(x)}{\partial x} = \exp(x).$$
$$\frac{\partial f(x)}{\partial g_2(x)} = 9g_2(x)^2, \qquad \frac{\partial g_2(x)}{\partial x} = 1$$

.
So the result is :

$$\begin{aligned}\frac{\partial f(x)}{\partial x} &= 4\exp(x) + 9g_2(x)^2 \\ &= 4\exp(x) + 9(1+x)^2.\end{aligned}$$

2.5 Gradient descent

Gradient descent is an optimization algorithm that updates parameters in the direction of the negative gradient to minimize a function.

Gradient Descent (GD) is applied when we seek the minimum of a function whose analytical expression is known, which is differentiable, but whose direct calculation of the minimum is difficult.
The problem is to find the value of x that minimizes $f(x)$. In this example, we know the analytical expression of the function f:

$$f(x) = 2x^2 \cos(x) - 5x$$

we can calculate the derivative

$$f^{'}(x) = 4x \cos(x) - 2x^2 \sin(x) - 5$$

The curve of f on the interval $[-5, 5]$ is represented in the figure 2.10.

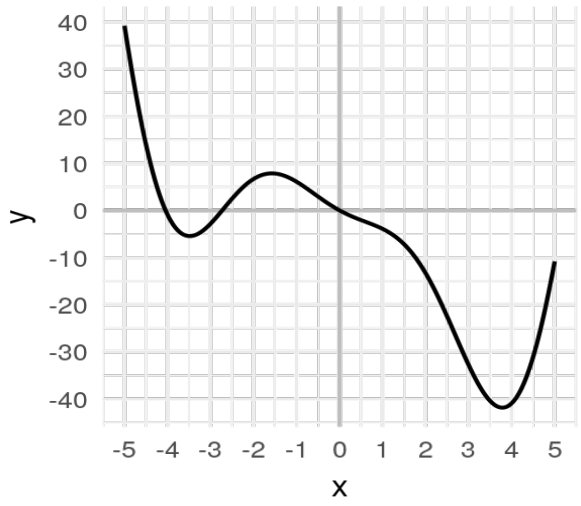


Figure 2.10: The curve of $f(x)$

To find the minimum of the function f analytically, we must find the roots of the equation $f^{'}(x) = 0$, which is "difficult". So we will use Gradient Descent, the principle of which is based on the value of the derivative that corresponds to the inclination of the slope at a given point.

- If the derivative is equal to 0, it is completely flat. This is the case at the minimum for example.
- If the derivative is negative, the curve goes down when we go to the right.
- If the derivative is high, it is because the slope is very steep.
- If the derivative is low, the slope is low.

The gradient descent algorithm consists of starting from a randomly chosen point, descending the slope step by step to the flat point ($f^{'}(x) = 0/min$) instead of finding the roots of $f^{'}(x) = 0$ directly.

- To descend, you must follow the sign of the derivative.
- The positive derivative indicates that the slope goes up to the right so we go down to the left.
- The negative derivative indicates that the slope goes down to the right so we go to the right.

The method can be summarized in 3 steps:

1. take a point x_0 at random.
2. repeat until $f' = 0$, or a maximum number of iterations is reached

   a calculate the value of the slope (the derivative) $f^{'}(x_i)$.

   b move in the opposite direction to the slope $x_{i+1} = x_i - \alpha.f^{'}(xi)$

α corresponds to the learning rate, and the minus allows to go in the opposite direction. With $\alpha = 0.05$. The descent of the gradient of the function f is illustrated in the figure 2.11.

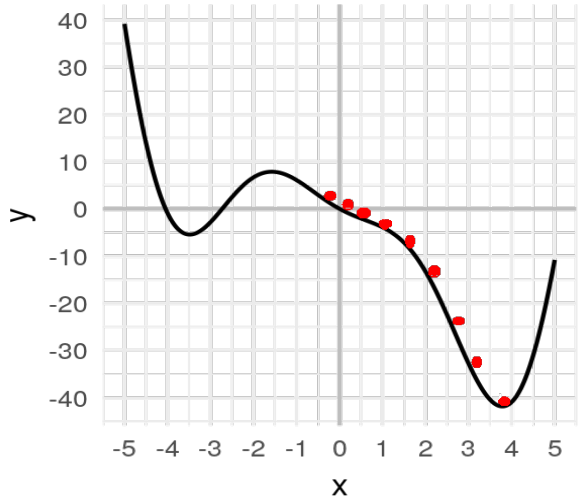


Figure 2.11: Gradient descent with a learning rate of 0.05

2.5.1 The local minima problem

Sometimes it will manage to find the global minimum, and other times, the algorithm will get stuck in a local minimum, in the figure 2.12, points A, B and C are local minima, only point D is a global minimum.
A technique to avoid this problem is to run the algorithm several times, changing the starting point at each execution, and to keep the smallest of the minima, but obviously it is more computationally intensive.

2.5.2 Perceptron and gradient descent algorithm

The gradient descent method applies to single-layer perceptron networks. The weights are optimized by several passes on the training set.
We can derive the perceptron algorithm with the principle of minimizing the error, and formulate the learning problem as an optimization problem. Thus, for each training example , we

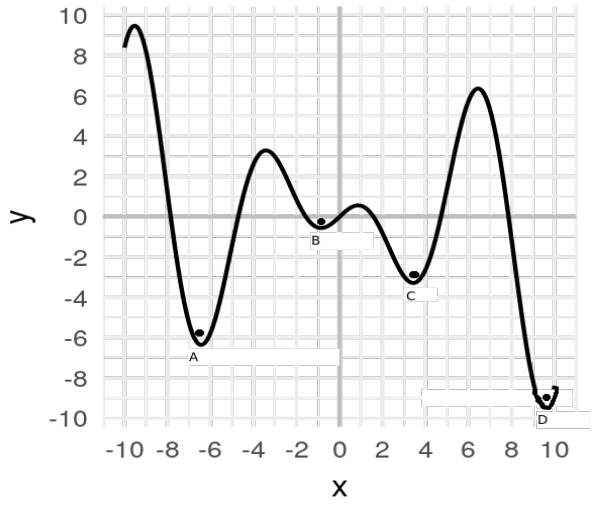


Figure 2.12: The local minima problem

try to minimize a certain distance (error) between the desired output y_t and the actual output produced by the perceptron $h_W(X_t)$.
We define the error by:

$$\mathbb{E}\left(y_t, h_W\left(X_t\right)\right) = -\left(y_t - h_W\left(X_t\right)\right) W.X_t$$

where

$$W.X_t = \sum_i w_i x_i$$

If the prediction is good then $\mathbb{E} = 0$. Otherwise, the error is the distance between $W.X_t$ and the threshold of the activation function of the perceptron.
For example if $y_t = 1$ but $h_W(X_t) = 0$, which means that $W.X_t < 0$, in this case
$\mathbb{E} = -(1-0).W.X_t = -W.X_t$.
Learning consists in minimizing the average error over the entire training set. the average error is calculated as follows:

$$\mathbb{E}_M = \frac{1}{|D|} \sum_{(x_t, y_t)_{\in D}} \mathbb{E}\left(y_t, h_W\left(X_t\right)\right)$$

the total error is given by

$$\mathbb{E}_T = \sum_{(x_t, y_t)_{\in D}} \mathbb{E}\left(y_t, h_W\left(X_t\right)\right)$$

The minimization of the error consists in improving the values of the weight vector W using the gradient descent method as follow:

1. initialize W randomly
2. repeat
   - **a** compute the partial derivative of the global error for all w_i. The gradient of the error is the vector of all partial derivatives.
   - **b** go in the opposite direction of the gradient.

$$\begin{aligned}\frac{\partial}{\partial w_i}\mathbb{E}_T &= \frac{\partial}{\partial w_i}\sum_{(x_t,y_t)\in D}\mathbb{E}\left(y_t, h_W\left(X_t\right)\right) \quad ,\forall i \\ &= \frac{\partial}{\partial w_i}\sum_{(x_t,y_t)\in D}\mathbb{E}\left(y_t, h_W\left(X_t\right)\right) \quad ,\forall i \\ &= \sum_{(x_t,y_t)\in D}\frac{\partial}{\partial w_i}\mathbb{E}\left(y_t, h_W\left(X_t\right)\right) \quad ,\forall i\end{aligned}$$

Perceptron Gradient Descent Algorithm

To perform gradient descent, one must calculate the sum of the derivatives on all training examples before updating the parameters (the w_i)

Algorithm 3 Perceptron Gradient Descent Algorithm

1: Initialize randomly w_i
2: While stopping criterion not met

a- For each example (X_t, y_t) of D

- $\triangle w_i = \triangle w_i + \frac{\partial}{\partial w_i}\mathbb{E}\left(y_t, h_W\left(X_t\right)\right), \qquad \forall i$
- End-for

b- $w_i = w_i - \alpha.\triangle w_i, \qquad \forall i$

3: End-while

Modifications are applied after each pass of the entire learning base (batch learning mode).

Widrow-Hoff algorithm (stochastic)

The gradient descent algorithm converges but it is not very fast. The Widrow-Hoff algorithm applies the same modification, but instead of doing it after seeing all the examples, it is applied after each test. This way, the algorithm converges faster on most problems. However, depending on the order and value of the examples, the algorithm can constantly ascillate between two values for a weight. When applying this algorithm, care must be taken to change the order of the training examples regularly.

Algorithm 4 Widrow-Hoff Alorithm

1: Initialize randomly w_i
2: While stopping criterion not met

a- for each example (X_t, y_t) of D

- $w_i = w_i - \alpha.\frac{\partial}{\partial w_i}\mathbb{E}\left(y_t, h_W\left(X_t\right)\right), \qquad \forall i$
- End-for

3: End-while

The gradient calculation (partial derivatives) for updating the weights of a perceptron with threshold function (heavyside) is as follows:

$$\begin{array}{rcl} \frac{\partial}{\partial w_i}\mathbb{E}\left(y_t, h_W\left(X_t\right)\right) & = & \frac{\partial}{\partial w_i}\left(-\left(y_t - h_W\left(X_t\right)\right)\right) W.X_t \\ & = & \frac{\partial}{\partial w_i}\left(-\left(y_t - h_W\left(X_t\right)\right)\right) \sum w_i x_{t,i} \\ & \cong & -\left(y_t - h_W\left(X_t\right)\right) x_{t,i} \end{array}$$

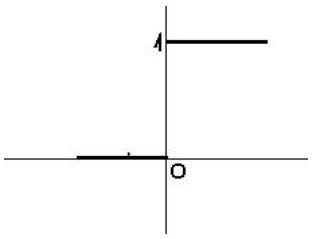
Then, the weights w_i are updated using the gradient descent algorithm, specifically the Widrow-Hoff learning rule (also known as the Least Mean Squares (LMS) algorithm), which adjusts the weights based on the error between the predicted and actual output.

$$w_i = w_i - \alpha.\frac{\partial}{\partial w_i}\mathbb{E}\left(y_t, h_W\left(X_t\right)\right), \qquad \forall i$$

This leads to the perceptron learning rule;

$$w_i = w_i + \alpha.\left(y_t - h_W\left(X_t\right)\right) x_{t,i}$$

Issue The gradient descent method is applicable to differentiable functions. However, in the case of a threshold perceptron, the function $h_W\left(X\right)$ is not defined ,and therefore not derivable, when $WX = 0$.



The threshold function can cause training instability, as it introduces discontinuities that prevent smooth weight updates.

2.5.3 Exercise

We are given a certain number of example vectors for 2 classes. We aim to perform a 2-class classification using any classifier.

1. Provide the different steps to follow, in the form of an algorithm, to perform the learning of these data, as well as a classification test with the calculation of correct classification rates, in stochastic learning mode.
2. Give an example of a stochastic learning algorithm.
3. Provide the different steps to follow, in the form of an algorithm, to perform the learning in batch learning mode.

2.5.4 Solution

From the words "stochastic" and "batch", we understand that it may involve a neural network with supervised learning. Before starting the learning process, for each learning algorithm, the following steps are required:

1. Construction of a training set and a test set.
2. Initialization of the classifier's parameters (for example : initializing the weights of a neural network with random values).

Stochastic Algorithm

For one learning epoch :

1. For each example in the training set
2. Begin
   (a) Present this example to the classifier
   (b) Compute the classifier's output
   (c) Compute the output error
   (d) Update the classifier's parameters based on the error

   End

Testing and Calculation of Correct Classification Rate

- Initialize a counter `cpt(i) = 0` for each class i
- For each example in the test set
  - Present this example to the classifier
  - Activate the classifier
  - If the example is correctly classified
    - Increment the counter of its class: `cpt(i)++`
  - EndFor
- For each class i
  - Correct classification rate for class i =
    `cpt(i)` / (number of test examples in class i)
  - EndFor

Formulation of the Global Algorithm

- while the number of epochs is not reached, or the error is not small enough
-
  - Perform learning for one epoch
  - Perform classification testing.
  - Calculation ofcorrect classification rate
  - EndWhile

Examples of a stochastic learning algorithm :

- Perceptron algorithm with threshold function.
- Widrow-Hoff algorithm

Batch Learning Algorithm

Error Accumulation

- For each example in the training set
- – Present the example to the classifier
  – Activate the classifier
  – Compute the output error and accumulate it

Global Formulation of the Algorithm

- For each learning cycle
- – Accumulate errors over one cycle
  – Update the classifier's parameters based on the accumulated error
  – Compute recognition rates

2.6 Logistic Regression

Definition Regression is the process of estimating the relationship between input variables and output variables.
In regression, it is assumed that the output variables depend on the input variables. Therefore, the input variables X_i are called independent variables, also known as predictors, while the output variables y_i are referred to as dependent variables.

2.6.1 Logistic regression

Logistic regression is one of the machine learning algorithms used for binary classification problems, involving predictions such as "yes or no" and "A or B".

2.6.2 Logistic Regression / Linear Regression

Logistic regression is used when the response variable is categorical, such as yes/no, true/false, or success/failure. Linear regression is used when the response variable is continuous, such as the number of hours, height, or weight.

Is logistic regression a non-linear regression ? Yes, indeed, it is a non-linear regression because the transfer function is non-linear, specifically the logistic function. However, to separate positive and negative cases, it constructs a linear decision boundary based on a linear combination of the variables. In this sense, it is referred to as a linear classifier. This perspective is commonly found in pattern recognition.

2.6.3 Linear and Logistic Regression Model

The objective is to model a response variable Y as a function of a set of predictors, assumed to be quantitative, and denoted as $x^1, ..., x^p$.

The Linear Regression Model

When the response Y is quantitative, the linear model assumes that the conditional expectation of Y, given the predictors $x^1, ..., x^p$, is a linear combination of the predictors: $\mathbb{E}(Y \mid x^1, \ldots, x^p) = \beta_0 + \sum_{j=1}^{p} \beta_j x^j$. The best approximation of Y is then given by:

$$Y = \beta_0 + \sum_{j=1}^{p} \beta_j x^j + \epsilon$$

where ϵ is a random variable with zero expectation and is independent of the joint distribution of the predictors.

The Logistic Regression Model

The logistic regression model (McCullagh and Nelder, 1983; Draper and Smith, 1966; Dobson, 1990) establishes a parametric relationship between a binary variable $Y \in \{0, 1\}$ and the vector of covariates (or explanatory variables); $X = (x^1, x^2, \ldots, x^a)^t$.
Suppose we observe n pairs $\{(X_1, Y_1), ..., (X_n, Y_n)\} \in R^a * \{0, 1\}$, It is no longer possible to directly model Y as a linear combination of the predictors. The challenge is then to model π, the posterior probability that the response is positive (equal to 1) given the predictors x_j. Thus,

we can write: $\pi = \mathcal{P}(Y = 1 \mid x^1, \ldots, x^p)$, We then deduce that: $P(Y = 0 \mid x^1, \ldots, x^p) = 1 - \pi$. Thus, we can deduce the value of Y:

$$\text{if} \quad \frac{\pi}{1-\pi} > 1 \quad \text{then} \quad Y = 1$$

$\frac{\pi}{1-\pi}$ is called the odds ratio. The logistic regression model assumes that the logit transformation of π can be modeled as a linear combination of the predictors:

$$\text{logit}(\pi) = \ln\left[\frac{\pi}{1-\pi}\right] = \beta_0 + \sum_{j=1}^{p} \beta_j x^j$$

The probability π can be calculated as follows:

$$\begin{array}{rcl} \ln\left[\frac{\pi}{1-\pi}\right] = BX & \iff & \frac{\pi}{1-\pi} = \exp(BX) \\ & \implies & \pi = \exp(BX)(1-\pi) \\ & \implies & \pi = \exp(BX) - \pi\exp(BX) \\ & \implies & \pi(1+\exp(BX)) = \exp(BX) \\ & \implies & \pi = \frac{\exp(BX)/\exp(BX)}{1+\exp(BX)/\exp(BX)} \\ & \implies & \pi = \frac{1}{1+\exp(-BX)} \end{array}$$

The logistic regression model predicts the probability of belonging to class 1.

$$h_W(X) = P(Y = 1|X) = \frac{1}{1+\exp(-WX)}$$

The logistic function (see figure 2.13) satisfies $0 \leqslant h_W(X) \leqslant 1$, and is differentiable everywhere. For classification, the most probable class is selected.

- If $h_W(X) \geqslant 0.5$ select class 1.
- Else select class 0.

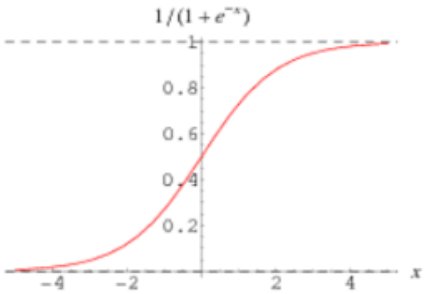


Figure 2.13: The ligistic function

2.6.4 Learning by Gradient Descent

The error of the logistic function is given by the formula :

$$\mathbb{E}(y_t, h_W(X_t)) = -y_t \log(h_W(X_t)) - (1-y_t)\log(1-h_W(X_t))$$

The error of the logistic function, also known as deviance, corresponds to the negative log-likelihood of the logistic model. Minimizing this error is equivalent to maximizing the likelihood function.

- If $y_t = 1$, we maximize the probability $P(Y = 1|X) = h_W(X_t)$
- If $y_t = 0$ we maximize the probability $P(Y = 0|X) = 1 - h_W(X_t)$

We compute the derivative of the error function to apply it in the gradient descent learning algorithm.

$$\begin{aligned}
\mathbb{E}(y_t, h_W(X_t)) &= -y_t \log(h_W(X_t)) - (1 - y_t) \log(1 - h_W(X_t)) \\
&= -y_t \log\left(\tfrac{1}{1+\exp(-WX)}\right) - (1 - y_t) \log\left(1 - \tfrac{1}{1+\exp(-WX)}\right) \\
&= -y_t \log(1) + y_t \log(1 + \exp(-WX)) - (1 - y_t) \log\left(\tfrac{1+\exp(-wx)-1}{1+\exp(-WX)}\right) \\
&= -y_t \log(1) + y_t \log(1 + \exp(-WX)) \\
&\quad - (1 - y_t)(\log(\exp(-WX)) - \log(1 + \exp(-WX))) \\
&= -y_t \log(1) + y_t \log(1 + \exp(-WX)) \\
&\quad - ((-WX) - \log(1 + \exp(-WX))) + y_t((-WX) - \log(1 + \exp(-WX))) \\
&= -y_t \log(1) + WX + \log(1 + \exp(-WX)) - y_t(WX)
\end{aligned}$$

The derivative of $\mathbb{E}$ with respect to w_i is then:

$$\begin{aligned}
\tfrac{\partial}{\partial w_i}\mathbb{E}(y_t, h_W(X_t)) &= \tfrac{\partial}{\partial w_i}(-y_t \log(1) + WX + \log(1 + \exp(-WX)) - y_t(WX)) \\
&= x_i - y_t x_i + \tfrac{1}{1+\exp(-WX)} . \exp(-WX) . (-x_i) \\
&= x_i\left(1 - y_t - \tfrac{\exp(-WX)}{1+\exp(-WX)}\right) \\
&= x_i\left(-y_t + \tfrac{1+\exp(-WX)-\exp(-WX)}{1+\exp(-WX)}\right) \\
&= -x_i\left(y_t - \tfrac{1+}{1+\exp(-WX)}\right) \\
&= -(y_t - h_W(X_t)) x_i
\end{aligned}$$

By applying the Gradient Descent Rule

$$w_i = w_i - \alpha . \frac{\partial}{\partial w_i}\mathbb{E}(y_t, h_W(X_t)), \qquad \forall i$$

The weight update becomes

$$w_i = w_i + \alpha(y_t - h_W(X_t)) x_{t,i}, \qquad \forall i$$

Which is the same as the perceptron learning rule, except that the definition of $h_W(X_t)$ is different.

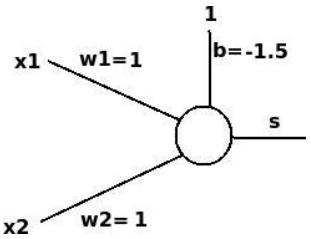
2.6.5 Exercise

1. Propose a threshold perceptron with its parameters to implement the logical AND function.
2. Propose a threshold perceptron with its parameters to implement the logical OR function.
3. Can a perceptron with a single neuron, whose output is a linear combination of the inputs, separate the output plane into two half-planes?

2.6.6 Solution

Logical AND Function

x_1	x_2	x_1 AND x_2
0	0	0
0	1	0
1	0	0
1	1	1



Perceptron for "AND" function

$s = h(x_1w_1 + x_2w_2 + b)$

To obtain $s = 1$, we must have: $x_1w_1 + x_2w_2 + b > 0$

Assume $w_1 = w_2 = 1$

We need to determine b

For $s = 1$, $x_1 + x_2 + b > 0$ only for $x_1 = 1$, $x_2 = 1$

Therefore:

$$2 + b > 0$$

And for $s = 0$ we can have $x_1 = 0$ or $x_2 = 0$ or $(x_1 = 0, and x_2 = 0)$.

We will then have

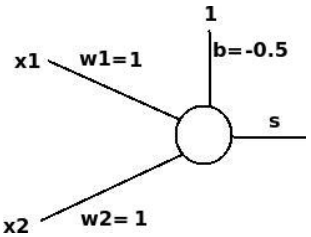
$$1 + b < 0 \quad and \quad 1 + b < 0$$

As a result, we will have

$$b = -1.5$$

The Logical OR Function

x_1	x_2	x_1 OR x_2
0	0	0
0	1	1
1	0	1
1	1	1



Perceptron for "OR" function

$s = h(x_1w_1 + x_2w_2 + b)$

To have $s = 1$, we must have: $x_1w_1 + x_2w_2 + b > 0$

Let us assume $w_1 = w_2 = 1$

We need to determine b

We want: $x_1 + x_2 + b < 0$ only for $x_1 = 0$, $x_2 = 0$

Therefore: $1 + b > 0 \wedge 0 + b < 0 \Longrightarrow b = -0.5$

Answer to Question 3

An output that is a linear combination of the inputs takes the form: $s = x_1w_1 + x_2w_2 + b$. For each input value (x_1, x_2), the output s can take different values, not only 0 or 1.

However, for the output plane to be divided into two half-planes, the output must have only two possible values (0 or 1), and the two classes must be linearly separable.

Therefore, the answer is **NO**, this perceptron cannot separate the output plane into two half-planes.

2.7 Multilayer Perceptron and Backpropagation

The multilayer perceptron (MLP) is a feedforward artificial neural network organized into layers, where information flows in only one direction, from

- the input layer to the output layer. The input layer does not contain neurons; it is a virtual layer associated with the system inputs $X(t)$.
- Neurons are connected to each other through weighted connections.
- There are no connections between neurons within the same layer.
- A layered network must contain at least one layer with a sigmoid (logistic) activation function.
- In the following, we will assume that all activation functions of the neurons in the network are of the sigmoid type. $h(x) = \frac{1}{1+\exp(-x)}$, see figure 2.14

An MLP enables nonlinear classification with decision boundaries that can be convex, concave, open, or closed, and it can approximate any function. Gradient backpropagation is an extension of gradient descent learning to multi-layer networks, developed to handle problems where classes can take arbitrary shapes. Backpropagation refers to how the gradient of the output error is used to adjust the network's synaptic weights, propagating from the output layer back to the preceding layers, with the goal of minimizing errors.

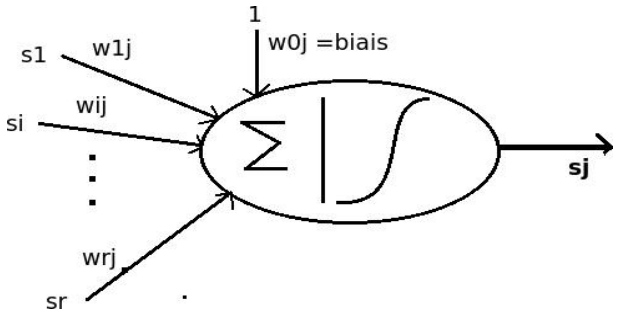


Figure 2.14: One neuron in MLP with sigmoid transfert function

The figure 2.15 shows an example of a layered network.

2.7.1 Network Learning

Let $(X(t), y(t))$ be the t^{th} training sample of the network.

$$X(t) = \{x_1(t), x_2(t), ..., x_p(t)\} \qquad y(t) = \{y_1(t), y_2(t), ..., y_q(t)\}$$

- p is the number of elements in the input layer or the number of features.
- q is the number of neurons in the output layer, e.g., the number of classes.
- The observed output vector is denoted as: $s(t) = \{s_1(t), s_2(t), ..., s_q(t)\}$.

Step 1: Forward Propagation

This step consists of presenting a stimulus ($X(t)$ at the network's input and allowing it to propagate to the output. This produces the observed output $s(t)$ from which we can compute the observed error $e(t)$.

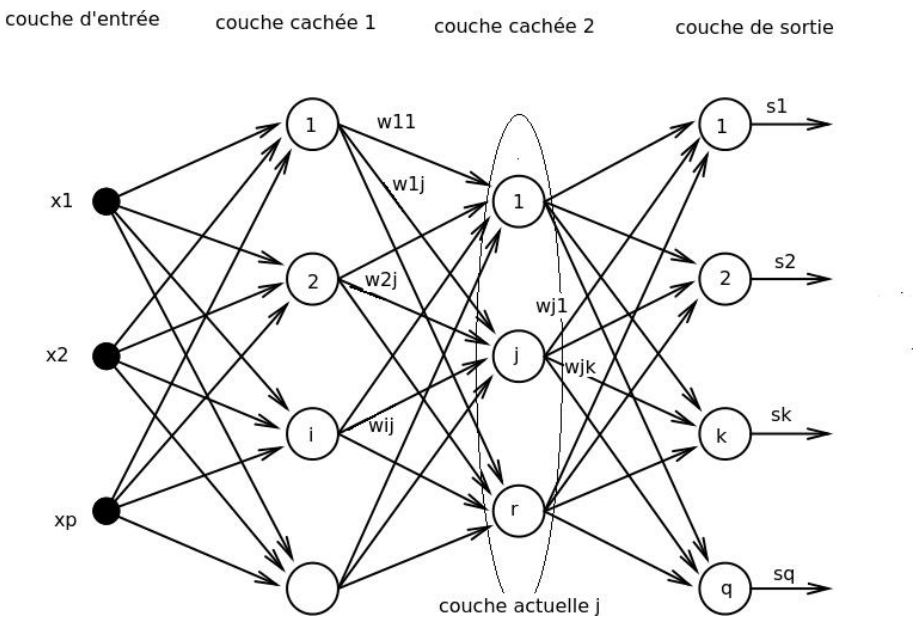


Figure 2.15: MLP or layered network.

Step 2: Backpropagation of the Gradient

The algorithm updates the weights neuron by neuron, starting from the output layer.
Let $e_j(t)$ be the observed error for the output neuron j at training step t, defined as:

$$e_j(t) = y_j(t) - s_j(t)$$

In the following

- The index j represents a neuron in the current layer.
- The index i represents a neuron in the previous layer.

The total error at the network's output is given by:

$$\mathbb{E}(t) = \frac{1}{2}\sum_{j=1}^{q} e_j^2(t)$$

The output $s_j(t)$ of neuron j is defined as:

$$s_j(t) = h\left(\sum_{i=0}^{r} w_{ij}(t)\, s_i(t)\right)$$

where h is the sigmoid/logistic activation function.
Let

$$v_j(t) = \sum_{i=0}^{r} w_{ij}(t)\, s_i(t)$$

The weighted sum of the inputs to neuron$j$.
w_{ij} represents the connection weight between neuron i in the previous layer and the current neuron j.
We assume that the previous layer contains r neurons.
To correct the observed error, the weights w_{ij} must be adjusted in the opposite direction of the error gradient $\frac{\partial \mathbb{E}(t)}{\partial w_{ij}(t)}$, Thus, the gradient descent algorithm is applied to minimize the network's overall error.

Gradient Computation for the Output Layer

The gradient descent rule is as follows: for each weight w_{ij} in the output layer:

$$\Delta w_{ij} = -\alpha . \frac{\partial \mathbb{E}}{\partial w_{ij}}$$

where $0 < \alpha < 1$ is the learning rate. Since

$$\mathbb{E}(t) = \frac{1}{2} \sum_{j=1}^{q} e_j^2(t)$$

And by applying the chain rule of differentiation for each neuron j in the output layer, we obtain:

$$\frac{\partial \mathbb{E}(t)}{\partial w_{ij}(t)} = \frac{\partial \mathbb{E}(t)}{\partial e_j(t)} . \frac{\partial e_j(t)}{\partial w_{ij}(t)}$$

Given that

$$e_j(t) = y_j(t) - s_j(t)$$

and

$$s_j(t) = h(v_j(t))$$

and

$$v_j(t) = \sum_{i=0}^{r} w_{ij}(t) s_i(t)$$

We can further apply the chain rule as follows:

$$\frac{\partial \mathbb{E}(t)}{\partial w_{ij}(t)} = \frac{\partial \mathbb{E}(t)}{\partial e_j(t)} . \frac{\partial e_j(t)}{\partial s_j(t)} . \frac{\partial s_j(t)}{\partial v_j(t)} . \frac{\partial v_j(t)}{\partial w_{ij}(t)} \tag{2.1}$$

By evaluating the terms of the gradient, we obtain:

1.

$$\begin{aligned} \frac{\partial \mathbb{E}(t)}{\partial e_j(t)} &= \frac{\partial \left(\frac{1}{2} \sum_{k=1}^{q} e_k^2(t) \right)}{\partial e_j(t)} \\ &= \frac{\partial \frac{1}{2} e_j^2(t)}{\partial e_j(t)} \\ &= \frac{2}{2} e_j(t) \\ &= e_j(t) \end{aligned} \tag{2.2}$$

2.

$$\begin{aligned} \frac{\partial e_j(t)}{\partial s_j(t)} &= \frac{\partial (y(t) - s_j(t))}{\partial s_j(t)} \\ &= -1 \end{aligned} \tag{2.3}$$

3.

$$\begin{aligned}
\frac{\partial s_j(t)}{\partial v_j(t)} &= \frac{\partial h(v_j(t))}{\partial v_j(t)} \\
&= \frac{\partial\left(\frac{1}{1+\exp(-v_j(t))}\right)}{\partial v_j(t)} \\
&= \frac{-\exp(-v_j(t))}{-(1+\exp(-v_j(t)))^2} \\
&= \frac{\exp(-v_j(t))}{(1+\exp(-v_j(t)))^2} \\
&= \frac{\exp(-v_j(t))}{1+\exp(-v_j(t))}.\frac{1}{1+\exp(-v_j(t))} \\
&= \frac{\exp(-v_j(t))+1-1}{1+\exp(-v_j(t))}.s_j(t) \\
&= \left(1-\frac{1}{1+\exp(-v_j(t))}\right).s_j(t) \\
&= s_j(t)(1-s_j(t)) \qquad (2.4)
\end{aligned}$$

4.

$$\begin{aligned}
\frac{\partial v_j(t)}{\partial w_{ij}(t)} &= \frac{\partial\left(\sum_{l=0}^{r}(w_{lj}(t).s_l(t))\right)}{\partial w_{ij}(t)} \\
&= \frac{\partial\left(w_{1j}(t).s_1(t)+w_{2j}(t).s_2(t)+\cdots+w_{ij}(t).s_i(t)+\cdots w_{rj}(t).s_r(t)\right)}{\partial w_{ij}(t)} \\
&= s_i(t) \qquad (2.5)
\end{aligned}$$

As a result, we obtain:

$$\begin{aligned}
\frac{\partial \mathbb{E}(t)}{\partial w_{ij}(t)} &= e_j(t)(-1)(s_j(t))(1-s_j(t))(s_i(t)) \\
&= -e_j(t)(s_j(t))(1-s_j(t))(s_i(t))
\end{aligned}$$

Consequently,

$$\begin{aligned}
\Delta w_{ij}(t) &= -\alpha\frac{\partial \mathbb{E}(t)}{\partial w_{ij}(t)} \\
&= -\alpha\left[-e_j(t)(s_j(t))(1-s_j(t))\right](s_i(t)) \\
&= \alpha.\delta_j(t).s_i(t)
\end{aligned}$$

Where $[\alpha.\delta_j(t).s_i(t)]$ is called the **delta rule** for the output layer.
And $[\delta_j(t) = e_j(t)(s_j(t))(1-s_j(t))]$ is called the **local gradient**.

Gradient Computation for Hidden Layer

The same procedure as for the output layer must be followed, but the problem is that we do not have desired outputs for the neurons in the hidden layers. Consequently, there is no observed error e.
We start with the last hidden layer, and the other hidden layers will follow the same process. Next, we consider:

- j is the index of a neuron in the current layer.
- i is the index of a neuron in the previous layer.
- k is the index of a neuron in the next layer.

We use the same formula for the gradient of the total error (2.1), but without using $e_j(t)$.

$$\frac{\partial \mathbb{E}(t)}{\partial w_{ij}(t)} = \frac{\partial \mathbb{E}(t)}{\partial s_j(t)} . \frac{\partial s_j(t)}{\partial v_j(t)} . \frac{\partial v_j(t)}{\partial w_{ij}(t)} \tag{2.6}$$

The last two terms yield the same result as (2.4) and (2.5). It remains to evaluate the first term $\frac{\partial \mathbb{E}(t)}{\partial s_j(t)}$.
$\mathbb{E}$ is still the total error of the output layer;

$$\mathbb{E}(t) = \frac{1}{2} \sum_{k=1}^{q} e_k^2(t)$$

Differentiating with respect to the hidden layer activation $s_j(t)$

$$\frac{\partial \mathbb{E}(t)}{\partial s_j(t)} = \frac{\partial \left(\frac{1}{2} \sum_{k=1}^{q} e_k^2(t) \right)}{\partial s_j(t)}$$

In the case of hidden layers, unlike output layer neurons, all the errors of the output neurons $e_k(t)$ depend on the output $s_j(t)$ of the hidden layer neuron j. This means that no constants will be eliminated during the differentiation process of the sum. We can therefore write:

$$\begin{aligned}
\frac{\partial \mathbb{E}(t)}{\partial s_j(t)} &= \sum_{k=1}^{q} \frac{\partial \left(\frac{1}{2} e_k^2(t) \right)}{\partial s_j(t)} \\
&= \sum_{k=1}^{q} e_k(t) . \frac{\partial (e_k(t))}{\partial s_j(t)} \\
&= \sum_{k=1}^{q} e_k(t) . \frac{\partial (e_k(t))}{\partial v_k(t)} . \frac{\partial v_k(t)}{\partial s_j(t)} \\
&= \sum_{k=1}^{q} \left[e_k(t) \frac{\partial (y_k(t) - h(v_k(t)))}{\partial v_k(t)} . \frac{\partial \left(\sum_{l=0}^{r} w_{lk}(t) s_l(t) \right)}{\partial s_j(t)} \right] \\
&= \sum_{k=1}^{q} \left[e_k(t) \underbrace{\frac{-\partial \left(\frac{1}{1+\exp(-v_k(t))} \right)}{\partial v_k(t)}}_{\text{déja calculé dans (2.4)}} . w_{jk}(t) \right] \\
&= \sum_{k=1}^{q} \left[e_k(t) . - \left[s_k(t) (1 - s_k(t)) \right] . w_{jk}(t) \right] \\
&= -\sum_{k=1}^{q} \left[\underbrace{e_k(t) \left[s_k(t) (1 - s_k(t)) \right]}_{\text{gradient local k}} . w_{jk}(t) \right]
\end{aligned} \tag{2.7}$$

As a result, we obtain:

$$\frac{\partial \mathbb{E}(t)}{\partial s_j(t)} = -\sum_{k=1}^{q} \delta_k(t) . w_{jk}(t) \tag{2.8}$$

Where

$$\delta_k = e_k(t)\left[s_k(t)\left(1 - s_k(t)\right)\right]$$

Substituting this into the gradient of the error with respect to the weights:

$$\begin{aligned}\frac{\partial \mathbb{E}(t)}{\partial w_{ij}(t)} &= -\left[\sum_{k=1}^{q} e_k(t)\left[s_k(t)\left(1 - s_k(t)\right)\right] w_{jk}(t)\right] s_j(t)\left(1 - s_j(t)\right) s_i(t) \\ &= -\left(s_j(t)\right)\left(1 - s_j(t)\right)\left(\sum_{k=1}^{q} \delta_k(t)\, w_{jk}(t)\right) s_i(t) \end{aligned} \tag{2.9}$$

Thus, the weight update equation follows the standard gradient descent formulation:

$$\Delta w_{ij}(t) = -\alpha \frac{\partial \mathbb{E}(t)}{\partial w_{ij}(t)} = +\alpha \delta_j(t)\, s_i(t) \tag{2.10}$$

Where the local gradient for hidden neurons is:

$$\delta_j(t) = s_j(t)\left(1 - s_j(t)\right) \sum_{k=1}^{q} \delta_k(t)\, .w_{jk}(t) \tag{2.11}$$

- Equations (2.10) and (2.11) hold for all hidden layers.
- In the first hidden layer, the inputs $s_i(t)$ are replaced by the raw input features $x_i(t)$.

2.7.2 The Gradient Backpropagation Algorithm

The pseudo-code of the gradient backpropagation method is presented in Algorithm 5.

The Generalized Delta Rule

The generalized delta rule is described as follows:

$$w_{ij}(t) = w_{ij}(t-1) + \alpha \delta_j(t)\, s_i(t) + \eta \Delta w_{ij}(t-1) \tag{2.12}$$

where η is the momentum coefficient.
Momentum is a technique used to accelerate gradient descent [Qia99], particularly in regions with high curvature, small but consistent gradients, or noisy gradients. It helps to smooth weight updates and prevents oscillations by incorporating the influence of past updates. The momentum-based weight update is given by:

$$\Delta w_{ij}(t) = \alpha \delta_j(t) s_i(t) + \eta \Delta w_{ij}(t-1) \tag{2.13}$$

This equation shows that the weight update at time t depends not only on the current gradient but also on the previous update, weighted by the momentum factor η. A higher momentum value ($\eta \approx 0.9$) helps maintain direction and prevents the optimization process from getting stuck in local minima, while a lower value allows for more responsiveness to recent gradients.

Algorithm 5 Backpropagation Algorithm

1. Initialize all weights with small random values in the range [-0.5, 0.5];
2. Prepare the training data;
3. Shuffle the training data randomly;
4. For each training sample t:

   a Compute the observed outputs by propagating the inputs forward;

   b Adjust the weights by backpropagating the observed error:

$$\begin{aligned} w_{ij}(t) &= w_{ij}(t-1) + \triangle w_{ij}(t) \\ &= w_{ij}(t-1) + \alpha \delta_j(t).s_i(t) \end{aligned}$$

   where the "local gradient" is defined as:

$$\delta_j(t) = \begin{cases} e_j(t).s_j(t).(1-s_j(t)) & if\ j \in output\ layer \\ s_j(t).(1-s_j(t)).(\sum_k \delta_k(t).w_{jk}(t)) & if\ j \in hidden\ layer. \end{cases}$$

   where $s_i(t)$ is the output of neuron i in the previous layer or $x_i(t)$ in the case of the first hidden layer.

5. Repeat steps 3 and 4 until a maximum number of iterations is reached or until the root mean square error (RMSE) falls below a certain threshold.

2.7.3 Exercise

We consider the set of training examples belonging to two sets A and B as follows:

$$A = \left\{ a_1 \begin{pmatrix} 0 \\ 4 \end{pmatrix}, a_2 \begin{pmatrix} 1 \\ 3 \end{pmatrix}, a_3 \begin{pmatrix} 1 \\ 4 \end{pmatrix}, a_4 \begin{pmatrix} 1 \\ 5 \end{pmatrix}, a_5 \begin{pmatrix} 2 \\ 4 \end{pmatrix}, a_6 \begin{pmatrix} 5 \\ 1 \end{pmatrix}, a_7 \begin{pmatrix} 5 \\ 2 \end{pmatrix}, a_8 \begin{pmatrix} 6 \\ 1 \end{pmatrix}, a_9 \begin{pmatrix} 6 \\ 2 \end{pmatrix} \right\}$$

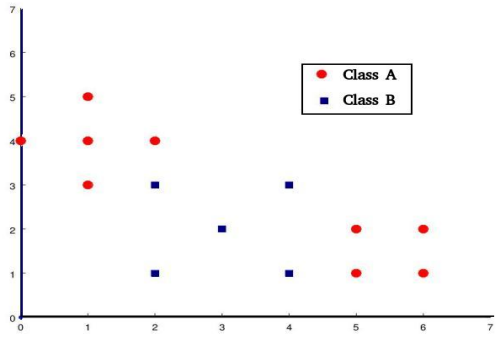
and

$$B = \left\{ b_1 \begin{pmatrix} 2 \\ 1 \end{pmatrix}, b_2 \begin{pmatrix} 3 \\ 2 \end{pmatrix}, b_3 \begin{pmatrix} 4 \\ 3 \end{pmatrix}, b_4 \begin{pmatrix} 2 \\ 3 \end{pmatrix}, b_5 \begin{pmatrix} 4 \\ 1 \end{pmatrix} \right\}$$

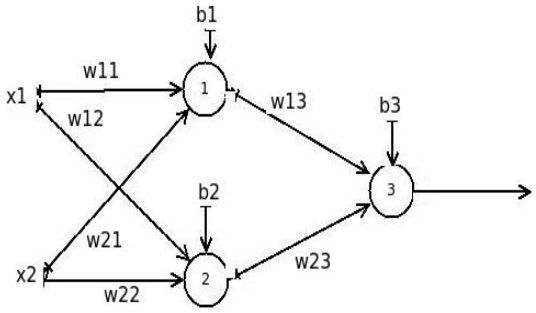
1. Plot the examples on a 2D plane.
2. Is there a perceptron that can perform this classification? What do you suggest?
3. Propose an architecture for your network.
4. Suggest weight values that would allow the classification to work.
5. Set two weights, $w_{1,1}$ and $w_{2,1}$, of the first layer to zero, and perform two learning iterations.

2.7.4 Solution

Data visualization

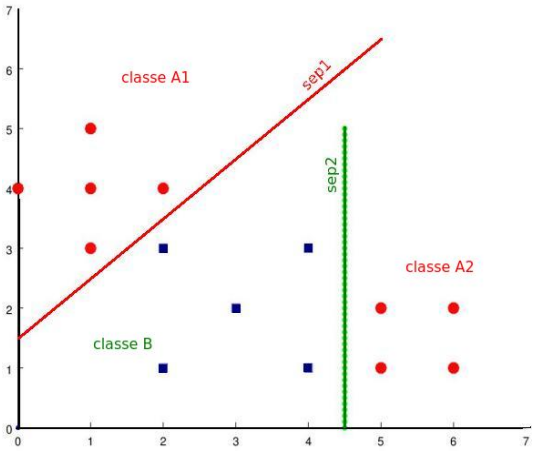


Proposed Architecture



Calculation of W_i

- Neuron 1 is the first separator.
- Neuron 2 is the second separator.
- For $h(v)$, the sigmoid function, to be equal to 1, we must have:
  $v_1 = w_{11}x_1 + w_{21}x_2 + b_1 > 0$
- The equation of the first separator is:
  $v_1 = w_{11}x_1 + w_{21}x_2 + b_1 = 0$



Graphical Solution

Reduced form of the line equation: $x_2 = Ax_1 + B$, and by comparing with

Separator 1:

$x_2 = (-w_{11}/w_{21})x_1 + (-b_1)$. This gives the slope $A = -w_{11}/w_{21}$ and $B = -b_1$.
From the graph, the equation of separator 1 is: $x_2 = x_1 + 1.5$, this implies: $b_1 = -1.5$ and $w_{11} = -1$, $w_{21} = +1$.

We still need to verify

With $a_5 = (2 \quad 4)^t$ We have: $v = -1 \cdot 2 + 4 \cdot 1 - 1.5 = 0.5 > 0$
With $a_2 = (1 \quad 3)^t$ We have: $v = -1 + 3 - 1.5 = 0.5 > 0$
With $b_4 = (2 \quad 3)^t$ We have: $v = -2 + 3 - 1.5 = -0.5 < 0$
Neuron 1 outputs 1 for class A_1 and 0 for class B and the rest.

For the second separator:

$x_2 = (-w_{12}/w_{22})x_1 + (-b_2)$
From the graph, the equation of $sep2$ is: $x_1 = 4.5$. This implies: $b_2 = -4.5$ and $w_{12} = +1$, $w_{22} = 0$

Verification of separator 2 outputs

With $a_7 = (5 \quad 2)^t$, We have: $v = 5 \cdot 1 - 4.5 = 0.5 > 0$
With $b_5 = (4 \quad 1)^t$, We have: $v = 4 - 4.5 = -0.5 < 0$

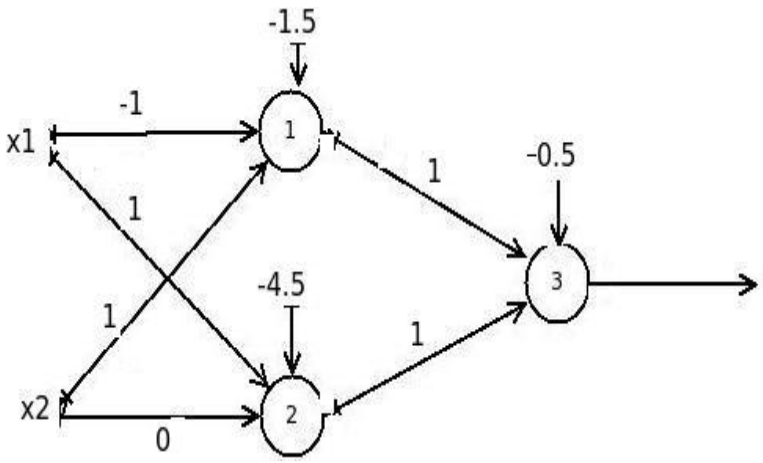
Neuron 2 outputs 1 for class A_2 and 0 for class B and the rest.

Output Layer

For the output neuron, if we assign 1 for class A and 0 for class B, then in the case where the outputs of the two hidden neurons (1 and 2) are:

- $(1 \quad 0) \Longrightarrow classA$; so output = 1,
- $(0 \quad 1) \Longrightarrow classA$; so output = 1,
- $(0 \quad 0) \Longrightarrow classB$; so output = 0.

This is equivalent to the logical OR case, where the case $(1 \quad 1)$ never occurs.
Therefore: $w_{13} = 1$, $w_{23} = 1$, $b_3 = -0.5$



Chapter 3

Fuzzy Logic and Fuzzy System

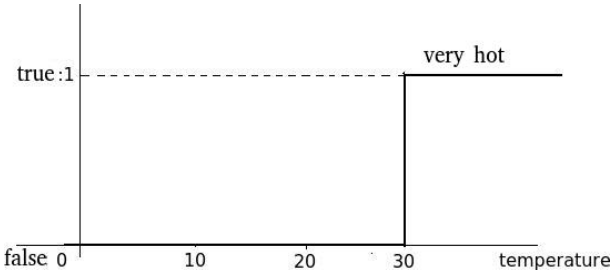
3.1 History and Introduction

Fuzzy logic officially began in 1965 with the publication of the article "Fuzzy Set" by Lotfi Zadeh, who was interested in understanding complex, nonlinear systems through methods that made their behavior clearer and more intelligible.
The main objective of fuzzy logic is to "formalize" all knowledge affected by imprecision (not to be confused with "uncertainty").
Imprecision manifests as a lack of precision in statements or instructions. For example, in the sentence "If it is very hot, then I will not wear a sweater," the imprecision lies in the notion of "very hot". If we consider that temperatures above 30° are very hot, what can we say when the temperature is 29.5°? "Very hot" is actually a fuzzy notion that depends on individuals, location, and context...
The decisions people make every day are often based on imprecise information, and this does not complicate life; on the contrary, it makes life even easier. The same applies to machines. For example, if a temperature of 30° triggers an entire ventilation system, but on a particular day, the temperature fluctuates between 30° and 29.5°, the ventilation system will oscillate between "on" and "off." This could be avoided if the machine could handle fuzzy statements.

3.2 Basic Concepts

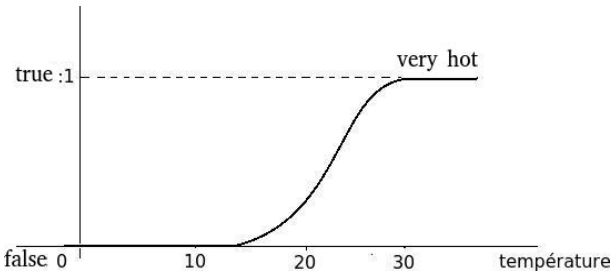
3.2.1 Boolean Logic and Fuzzy Logic

In Boolean logic (classical logic), a value can only be true or false. A precise threshold must be defined as a transition, and there are no intermediate values between '0' and '1'.



However, in fuzzy logic, boundaries are not strict, and there can be intermediate values between 'true' and 'false'.

Fuzzy logic attempts to associate the inherent imprecision of natural phenomena with the computational power of computers to create intelligent, robust, and simple reasoning systems. Fuzzy logic encompasses a set of disciplines, including:



- Approximate reasoning.
- Fuzzy logic.
- Fuzzy set theory.

3.2.2 The Fuzzy Set

The fundamental element of "Fuzzy Logic" is the fuzzy set, presented in figure 3.1. Fuzzy sets are named as such because their boundaries are imprecise. In contrast, classical sets have perfectly defined boundaries. They can be described by lists or expressions that indicate what belongs to the set and what does not.

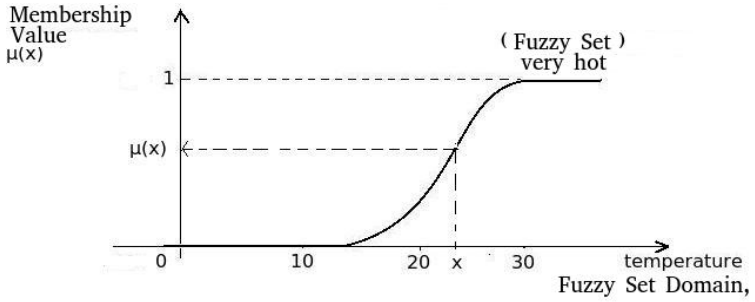


Figure 3.1: Structure of a fuzzy set

3.2.3 Membership Functions

In fuzzy logic, an element can belong to a fuzzy set with a membership value of 60% or 65%. The curve indicating the degrees of membership is called the membership function, denoted as $\mu(x)$. It allows defining a fuzzy set with boundaries that are not sharp but gradual.

Some of the most commonly used membership functions (see figure 3.2) include:

- The triangular function.
- The trapezoidal function.
- The gaussian function.
- The sigmoid function.

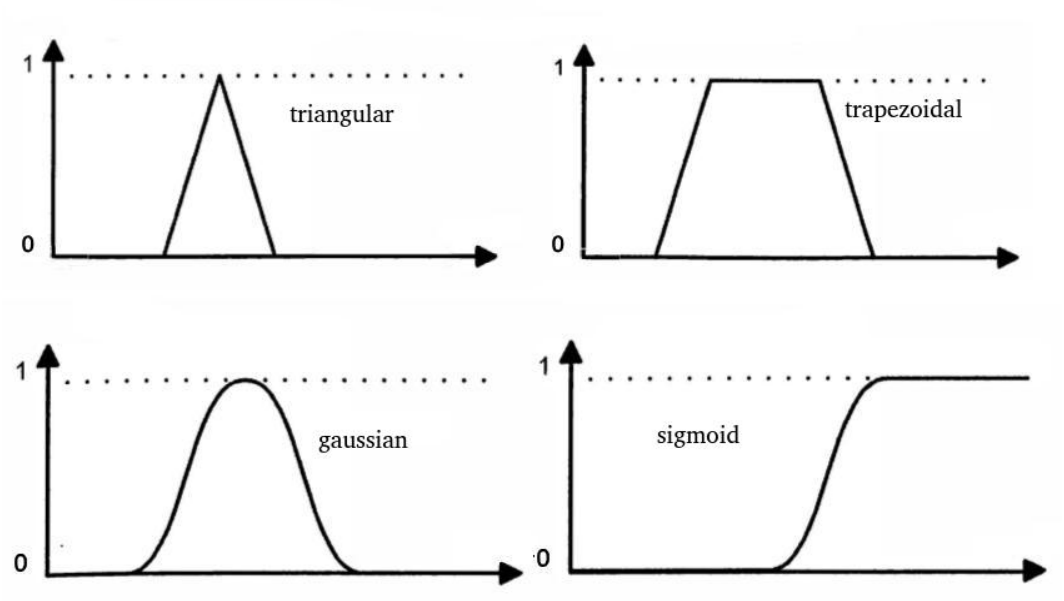


Figure 3.2: Membership functions

3.2.4 Characteristics of a Membership Function

Different membership functions are characterized by:

- **Height**: The maximum degree of membership that can be obtained.
- **Support**: The set of values for which the membership degree is different from 0.
- **core**: which represents the set of elements for which the membership degree is equal to 1

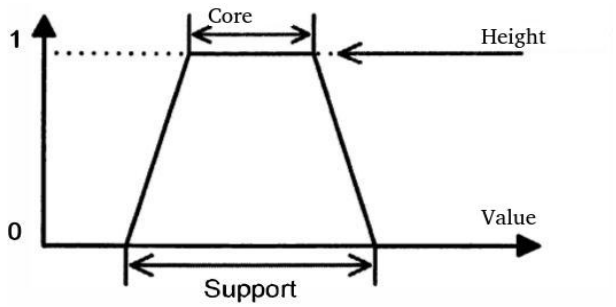


Figure 3.3: Caracteristics of a membership function

3.2.5 Linguistic Values and Variables

These are values and variables that represent terms from everyday language. For example, "temperature" is a linguistic variable, and "very hot" is a linguistic value. The different linguistic values for the linguistic variable "temperature" can be represented by the figure 3.4.

How can we read the membership degrees of a numerical value, for example, 16°? On the diagram, we see with dashed lines that for 16°, we intersect two curves: the one representing the value "cool" and the one representing the value "pleasant." At 16°, it is both cool and pleasant, but it is neither cold nor hot. The graph reading indicates that "cool" and "pleasant" are each true at 50%. Similarly, for 22° (in dotted lines), we see that it is neither "cold" nor "cool," but that it is "pleasant" at 60% (or 0.6) and "hot" at 40%.

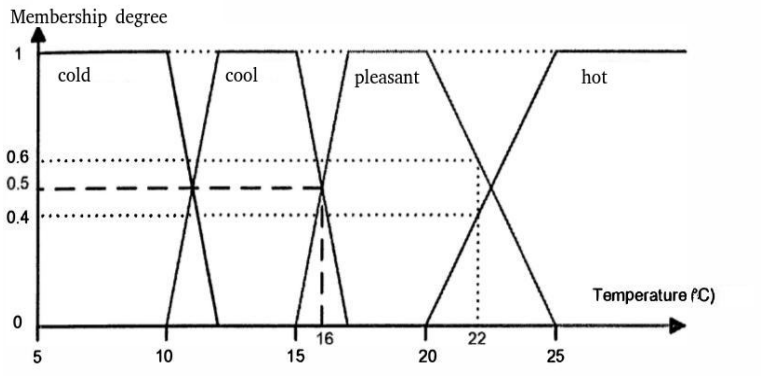


Figure 3.4: Example of linguistic values

3.2.6 The Fuzzy Rule

A fuzzy rule associates two or more fuzzy sets in the same sentence of the type "if... then...". The rules use fuzzy values instead of numerical values.
Fuzzy rules allow us to infer knowledge about the system's state based on linguistic qualifications provided by the fuzzification step. These knowledge elements are also linguistic qualifications. Usually, fuzzy rules are derived from the experience gained by operators or experts. This knowledge is translated into simple rules that can be used in a fuzzy inference process. However, it is possible to build a fuzzy rule base using learning methods without necessarily requiring a human expert.

3.2.7 The Fuzzy System

A fuzzy system is a set of assertions or rules that transform input data into results or outputs. The mathematical formalization of a fuzzy system is not fuzzy at all, but the user must understand it to use and adapt the fuzzy system. The user can program the system using words and phrases.

3.3 Implication Techniques

A fuzzy rule consists of two parts: the antecedent (or premise) and the consequent (or action). When executing a fuzzy rule, we must determine the truth value of the premise; if it is sufficient, the rule can be executed. In this case, we must take the fuzzy consequent set and modify the fuzzy solution variable. This process is called Implication. Implication ensures that the truth value transfer function follows a very simple law of approximate reasoning: "The truth value of the consequent cannot be higher than that of the antecedent."

To respect this law, the fuzzy consequent set must undergo some modification. There are two methods.

3.3.1 Minimum Implication

This method involves truncating the fuzzy consequent set at the level of the truth value of the premise. This operation removes the peak of the fuzzy consequent set. The truth value of the resulting fuzzy consequent set, μ_{sort}, as a function of the input μ_{ent} (the original consequent) and the premise μ_p, is given by the following equation:

$$\mu_{sort}(x) = \min(\mu_{ent}(x), \mu_p)$$

Let's consider the following rule:
If the price is high **Then** profit is close-to-manufacturing-cost.

We need to evaluate to what extent 'price' is an element of the "high" fuzzy set. The figures 3.5,3.6 shows how the membership value is calculated for a price of 24$.

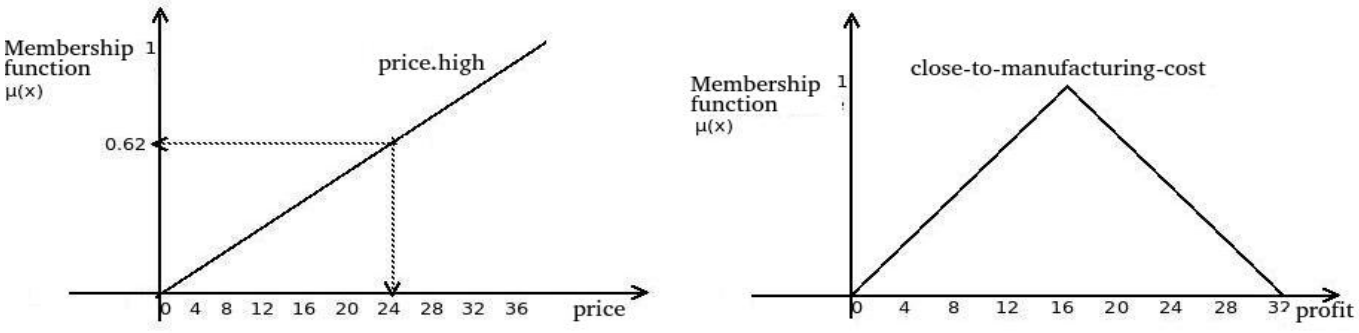


Figure 3.5: Inputs of minimum implcation

The result of the minimum implication is as follows:

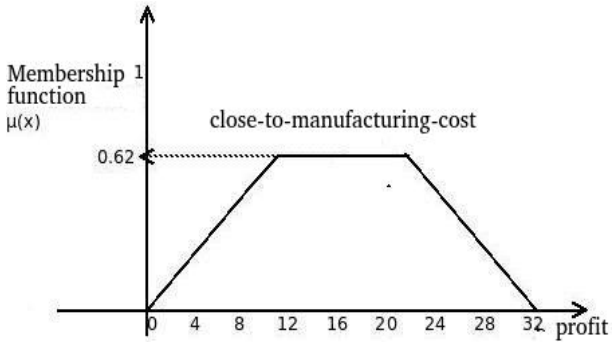


Figure 3.6: result of minimum implication

This operation removes the peak of the fuzzy consequent set. The fuzzy input set with membership function μ_{ent} is truncated using the truth value of the premise $\mu_p = 0.62$ to produce the "output" fuzzy consequent set μ_{sort}.

3.3.2 Product Implication

The product implication uses a different technique. The fuzzy consequent set is modified by multiplying each membership value by the truth value of the premise while preserving the overall shape of the fuzzy set. This is expressed by the following equation:

$$\mu_{sort}(x) = \mu_{ent}(x) . \mu_p$$

Using the same previous example, the result of the product implication is presented in figure 3.7:

The product implication preserves the overall shape of the input fuzzy set by contracting it so that its maximum is equal to the truth value of the premise.

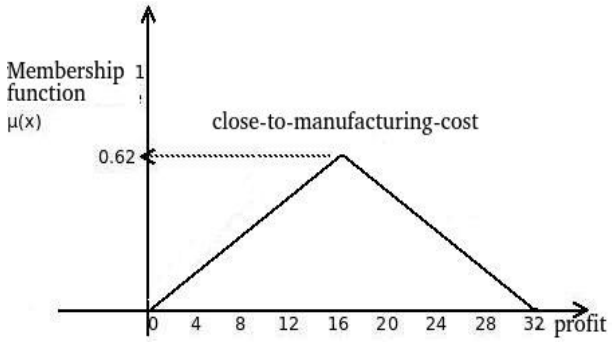


Figure 3.7: Result of product implication

3.4 Aggregation Techniques

Aggregation is a process of combining fuzzy sets to construct a single one that represents the solution variable. Just as with the implication process, there are also two ways to aggregate fuzzy consequent sets: the additive method and the Min/Max method. However, note that the implication method only deals with a single fuzzy set, whereas aggregation involves multiple sets.

3.4.1 Imperative Fuzzy Propositions

An imperative fuzzy proposition imposes a constraint on the solution and does not contain a premise; it is aggregated into the fuzzy solution without a prior implication process. These propositions are processed using the 'Minimum' operator (see figure 3.8). Let us consider the following rule base:

R1 The price must be high;

R2 The price must be low;

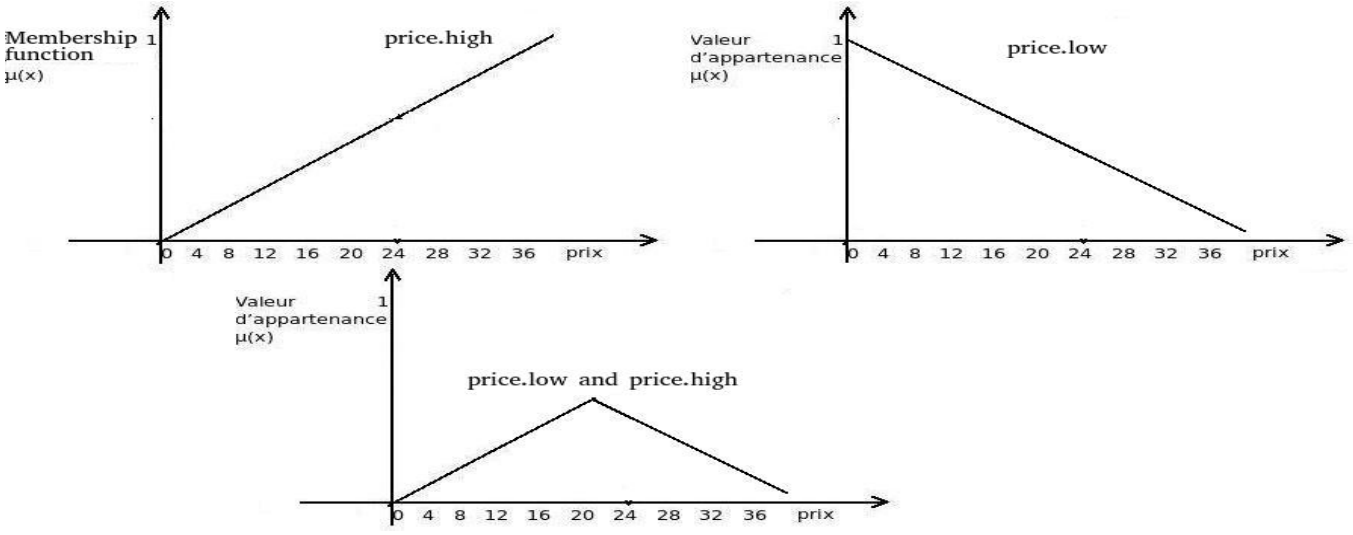


Figure 3.8: Aggregation of imperative proposition

The result of applying these two fuzzy rules is a set of values representing the smallest values from the two aggregated fuzzy sets, as indicated by the following equation:

$$\mu_{sol}(x) = \min_{i\in[1,\ldots,n]} (\mu_{sol}(x), \mu_{ent}(x)_i)$$

where μ_{sol} and μ_{ent} are the truth values of the input and output fuzzy solution sets. Imperative fuzzy propositions ensure that the model will at least provide acceptable default results.

3.4.2 Conditional Fuzzy Propositions

Such propositions follow the traditional 'if-then' syntax. There are two main techniques for aggregating conditional rules: the Min/Max method and the additive method. Let us consider the following rules:

R3 If manufacturing-cost is high Then price must be medium;

R4 If waste-rate is high Then price must be close to 1.5 * manufacturing-cost.

Let us consider the fuzzy descriptors ('fuzzy sets') of the fuzzy variables manufacturing-cost, in figure 3.9, price, and waste-rate in figure 3.10.

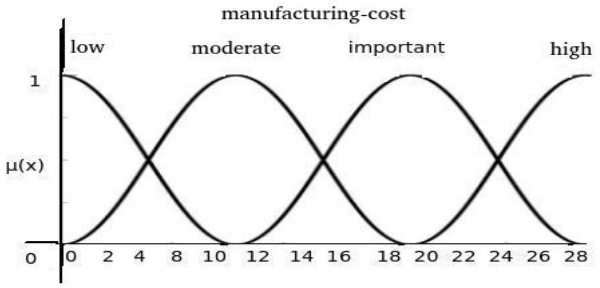


Figure 3.9: Fuzzy sets of "manufacturing-cost"

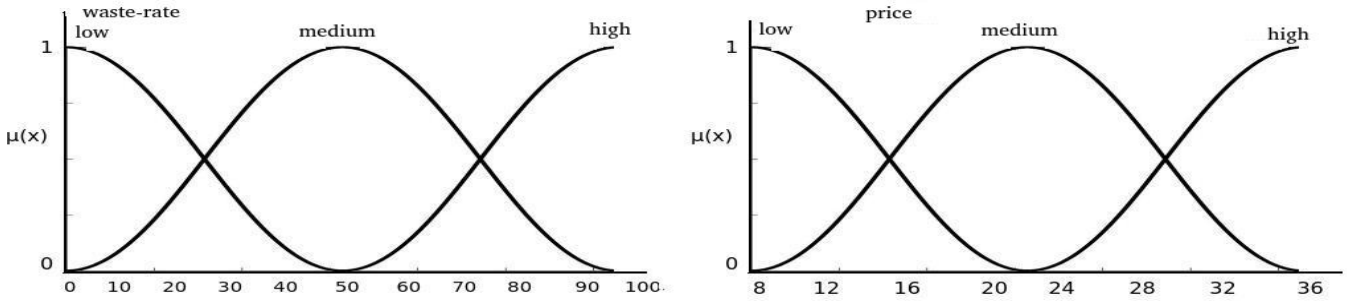


Figure 3.10: Fuzzy sets of "waste-rate"and "price".

Min/Max Aggregation

This technique performs the conjunction (using the 'OR' operator) of the fuzzy consequent sets with the fuzzy set of the solution variable. For each value of the variable, this process consists of taking as the membership value the highest among those given by the fuzzy sets.
By executing $R3$ in the previous example with a unit price of 16 dollars and $R4$ with a waste rate of 88%, we obtain the results presented in figures 3.11 and 3.12.

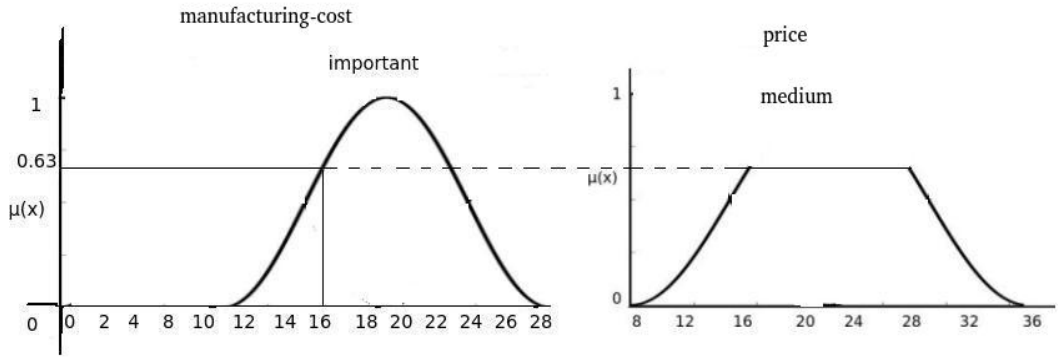


Figure 3.11: Execution of rule R3 (updating the fuzzy set 'price')

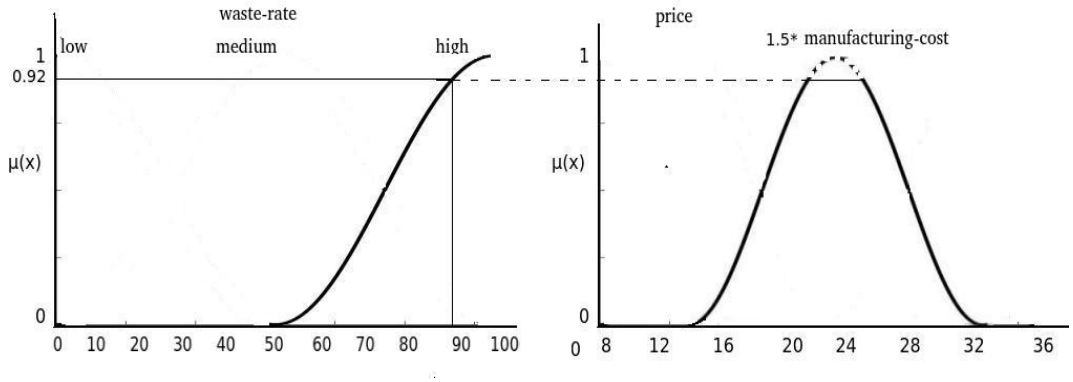


Figure 3.12: Execution of rule R4

The consequent of rule $R4$ specifies a value close to 1.5*manufacturing-cost. This results in a bell-shaped curve centered at '$16 * 1.5 = 24$', which is then truncated using the minimum implication process.

The Min/Max aggregation method uses the fuzzy union operator, taking the highest membership value between the fuzzy solution set being estimated and the truncated fuzzy number 24.00. The resulting 'price' variable, see figure 3.13, forms a double plateau. This operation is summarized as follows:

$$\mu_{sol}(x) = \max_{i \in [1,\ldots,n]} (\mu_{sol}(x), \mu_{co}(x)_i)$$

where μ_{co} is the membership value of the solution being estimated.

Figure 3.13: Aggregation of R4's consequent with R3's fuzzy solution set

Additive Aggregation

The additive technique actually adds the consequent fuzzy sets to the fuzzy solution set. For each value of the variable, a summation is performed, capped at '1'. Let's consider the same previous example, applying product implication.
The outcomes of applying rules R3 and R4 with the product implication are illustrated in Figures 3.14 and 3.15.

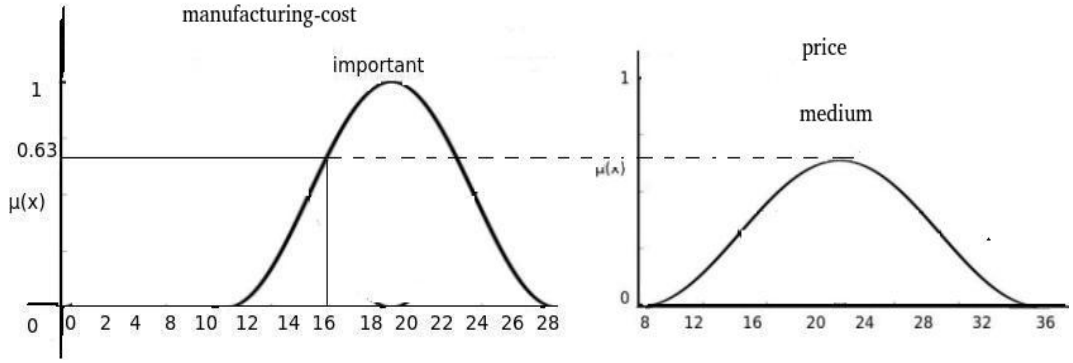


Figure 3.14: Execution of rule R3 with product implication

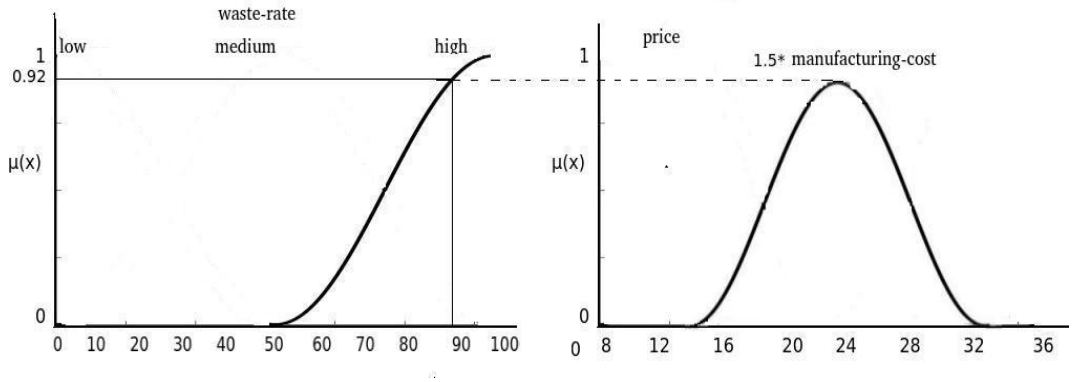


Figure 3.15: Execution of rule R4 with product implication

By applying a capped summation, we obtain a curve with a single plateau as presented in figure 3.16.

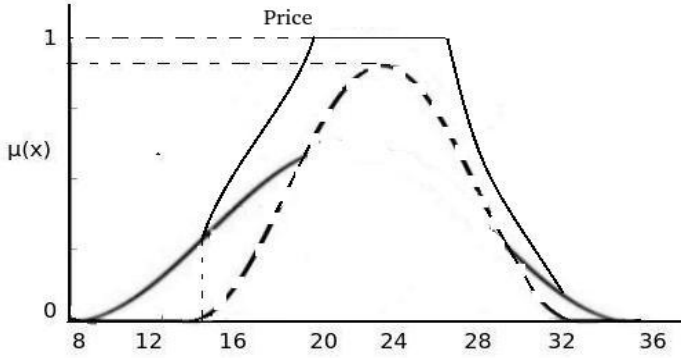


Figure 3.16: Result of additive aggregation

The additive aggregation is given by the following formula:

$$\mu_{sol}(x) = \min\left(\mu_{sol}(x) + \sum_{i=1}^{n} \mu_{co}(x)_i, 1\right)$$

Note An imperative rule should contribute to the solution only if all the truth values of the other rules are very small.

3.5 Defuzzification

'Defuzzification' is the process of finding a single value that best summarizes the information contained in the fuzzy solution set. This final value is called the 'representative value' of the solution variable. Many defuzzification methods exist in the literature, but the two most commonly used are the 'Center of Gravity' or 'Centroid' method and the 'Maximum' method.

Reasoning Method	Defuzzification Method
Minimum Implication and Min/Max Aggregation	Maximum
Product Implication and Additive Aggregation	Centroid

3.5.1 Maximum Method

Generally, for fuzzy solution sets with multiple plateaus, the representative value is determined by finding the plateau corresponding to the highest membership value. If the plateau has two boundaries, its midpoint is taken as presented in figure 3.17; if it has only one boundary, that single boundary is chosen as the representative value. If there are multiple plateaus with the same maximum level (meaning the solution resembles a sinusoidal shape), the average of the midpoints of the plateaus is taken.

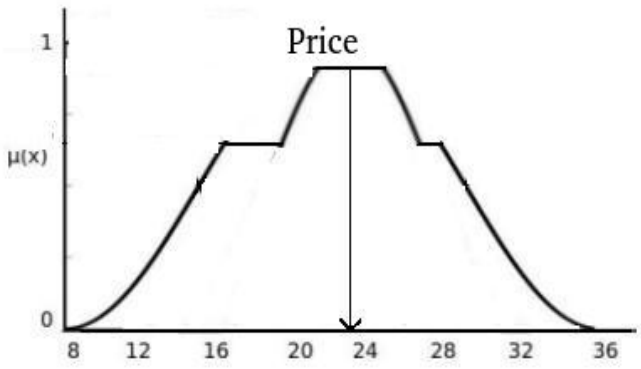


Figure 3.17: Maximum defuzzification

3.5.2 Centroid Method

The 'additive' aggregation combined with 'product' implication tends to create smooth regions with few or no plateaus. Applying the 'centroid' method divides the fuzzy set into two balanced

parts, as presented in figure 3.18. The center of gravity is given by the weighted average of the fuzzy set, as indicated by the following equation:

$$x = \frac{\sum_{i=1}^{n} d_i \mu_S(d_i)}{\sum_{i=1}^{n} \mu_S(d_i)}$$

where d_i represents the possible values of the fuzzy variable and $\mu_S(d_i)$ the corresponding membership values.

This way, the contributions of all triggered rules are added together, preventing the result from depending on a single rule.

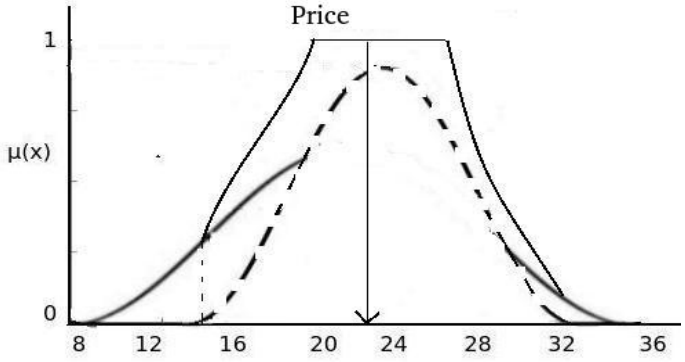


Figure 3.18: Centroid defuzzification

3.6 Fuzzy Reasoning System

The input data (scalars, vectors, and other fuzzy sets such as fuzzy numbers) are read by the system. Each rule is potentially executable. However, some rules are not triggered if the truth value of the premise falls below a certain threshold. Each executed rule provides an assumption contributing to modifying the value of the solution variable. This assumption, represented as a fuzzy set, is assigned a truth value based on the premise and then aggregated into the fuzzy region of the solution variable. Once all the rules (that can be triggered) have been executed, the composite solution variable is defuzzified, yielding a single numerical value as the crisp solution.

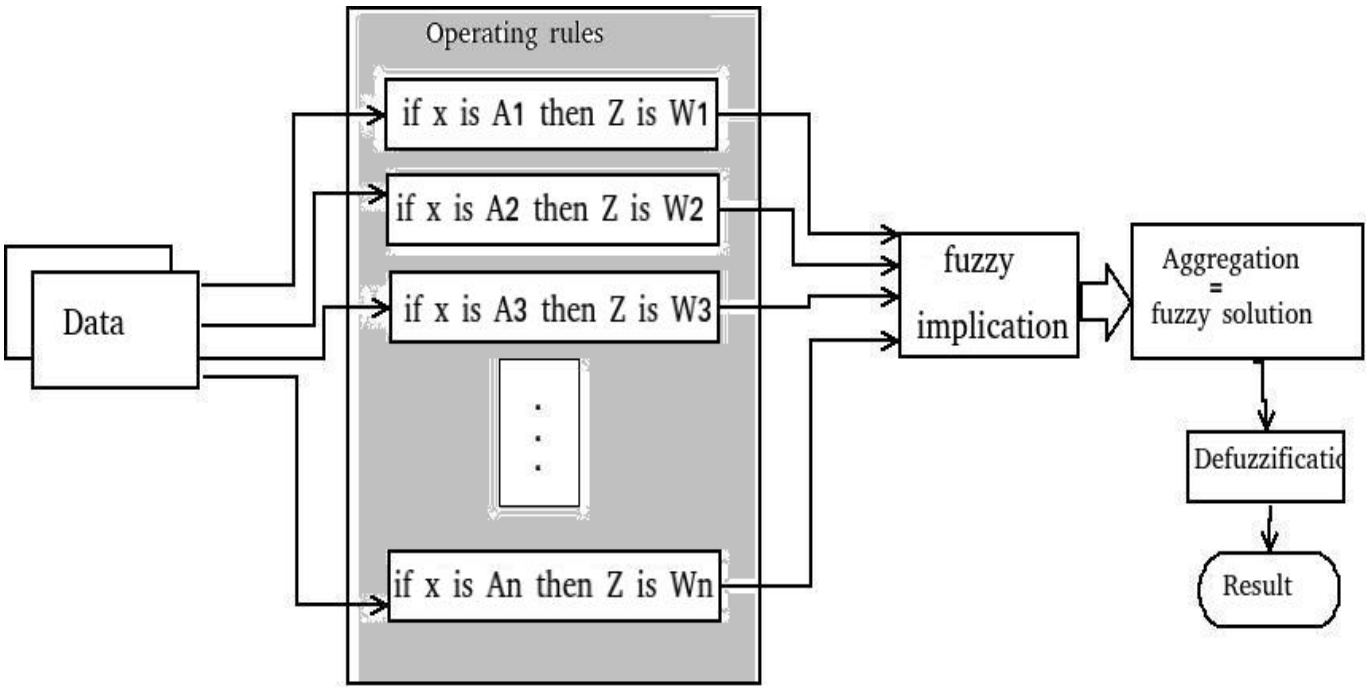
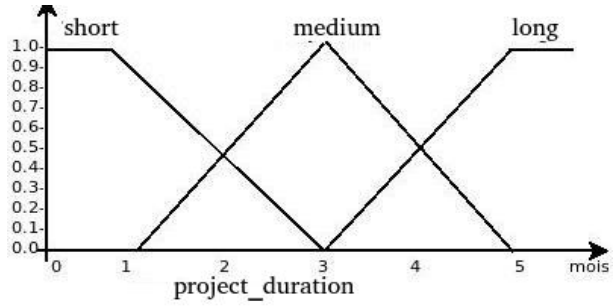


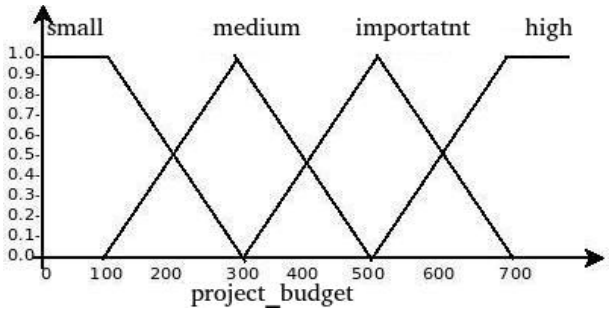
Figure 3.19: Fuzzy Reasoning System

3.7 exercice

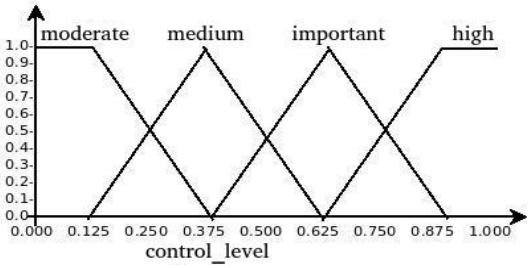
Let the fuzzy system be dedicated to determining the level of control of projects according to their characteristics.
We assume that the fuzzy sets for the fuzzy variables are defined as follows:

The Premises





The Consequent



Fuzzy Rules

- If project-duration is long, then control-level is important.
- If project-budget is medium, then control-level is medium.

Assigned Task

Perform implication, aggregation, and defuzzification by executing the two rules with the following data:
project-duration = 4.5 months.
project-budget = 320.

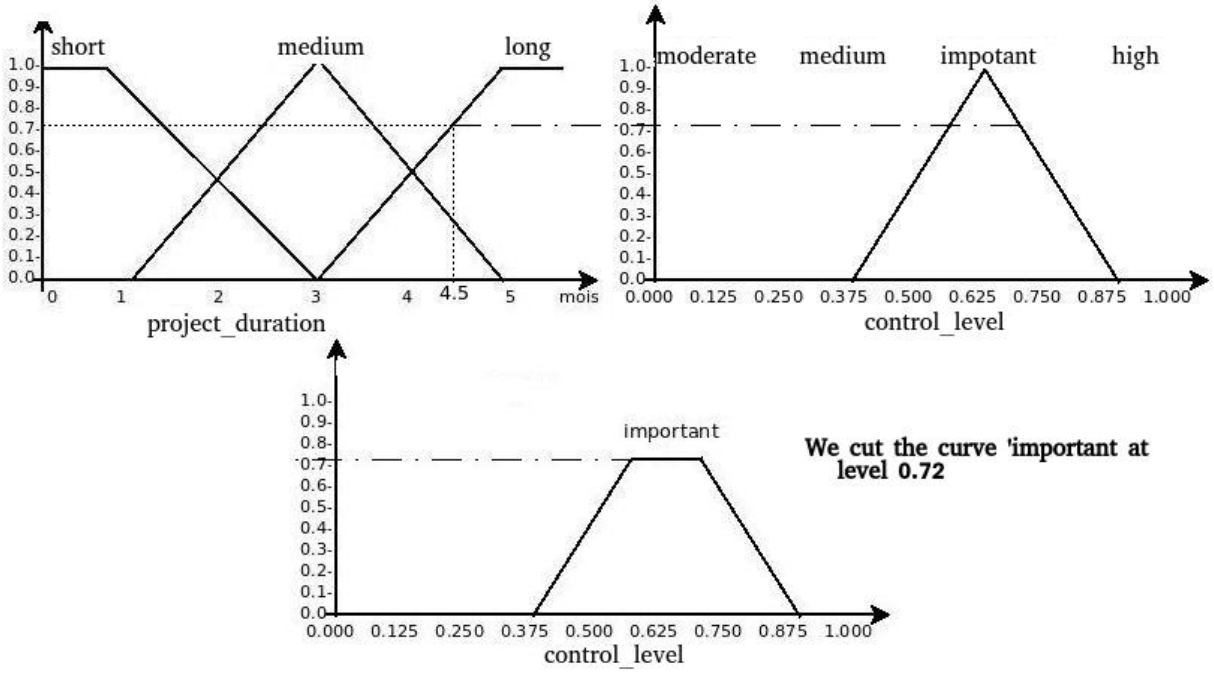
Note: You may choose the methods for implication, aggregation, and defuzzification as long as they are consistent.

3.8 Solution1

Minimum Implication

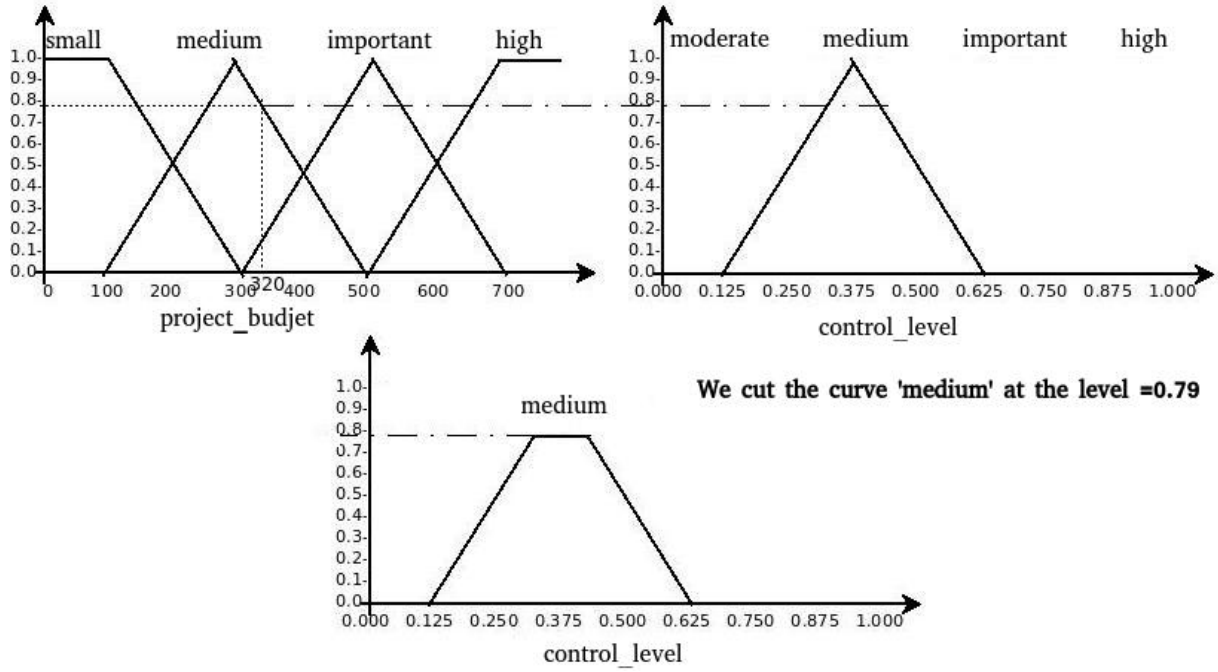
- **R1:** If project-duration is long, then control-level is important.

with project-duration = 4.5 months.

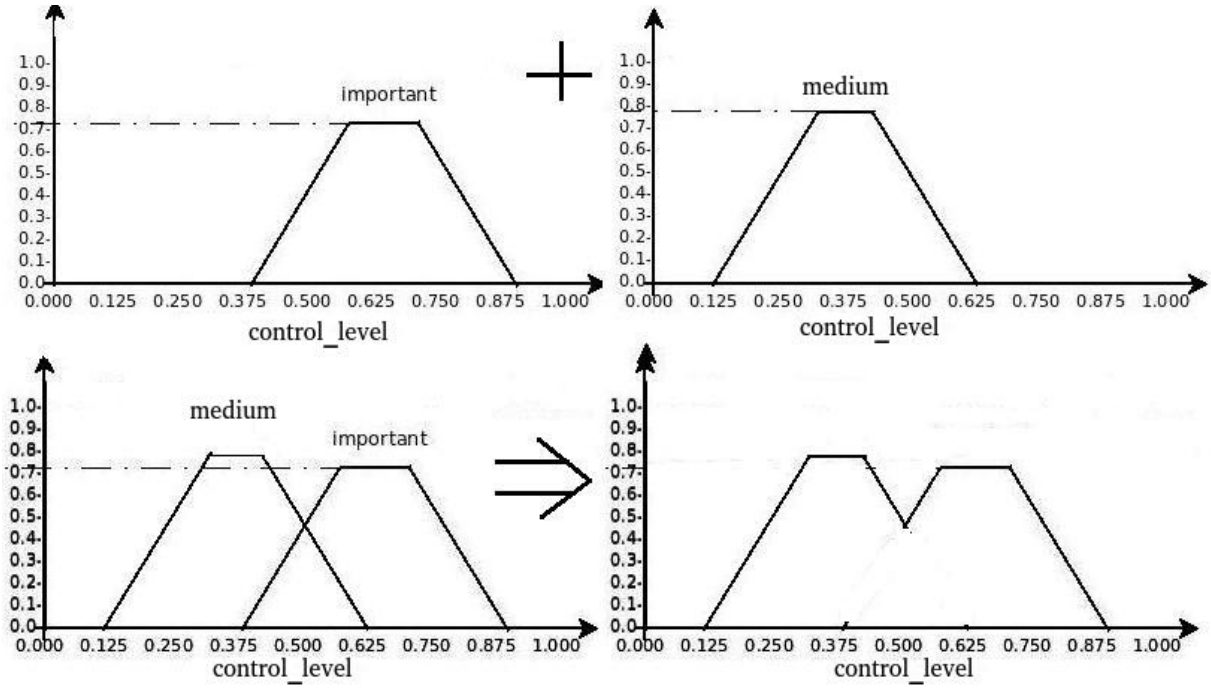


- **R2:** If project-budget is medium, then control-level is medium.

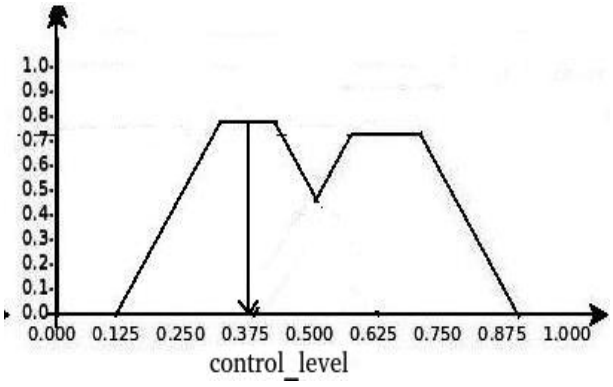
with budget-projet= 320.



Min/Max Agrégation



Maximum Defuzzification



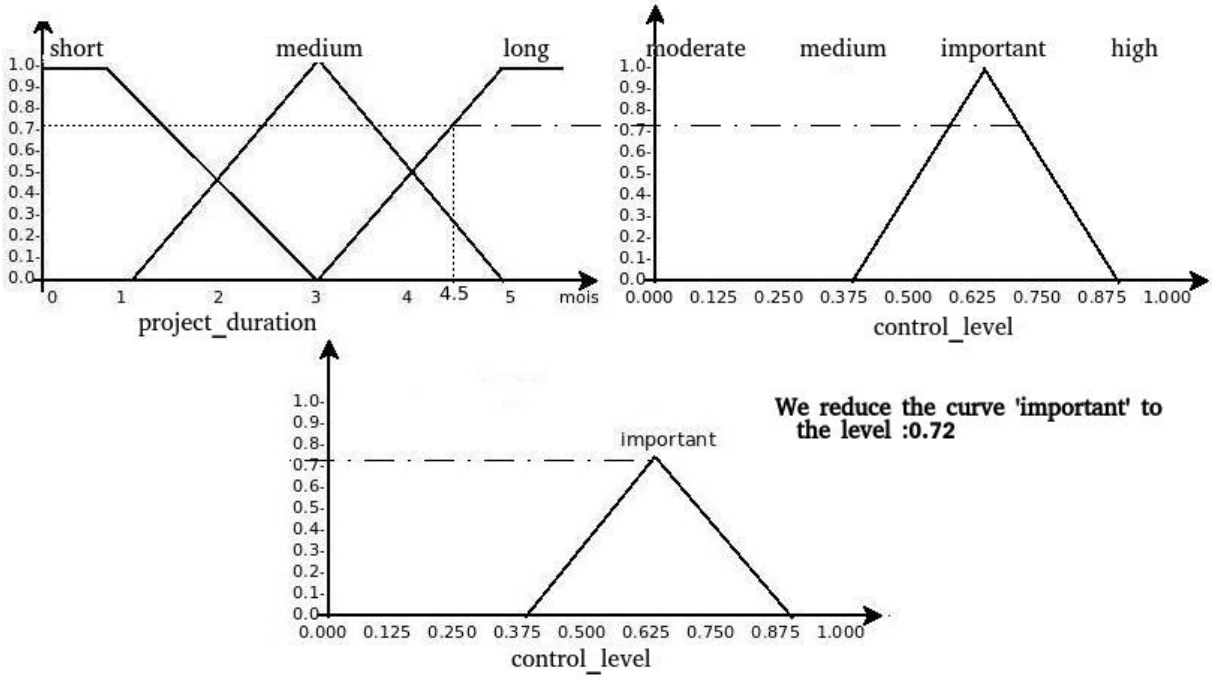
The resulting control level = 0.375.

3.9 Solution2

Product Implication

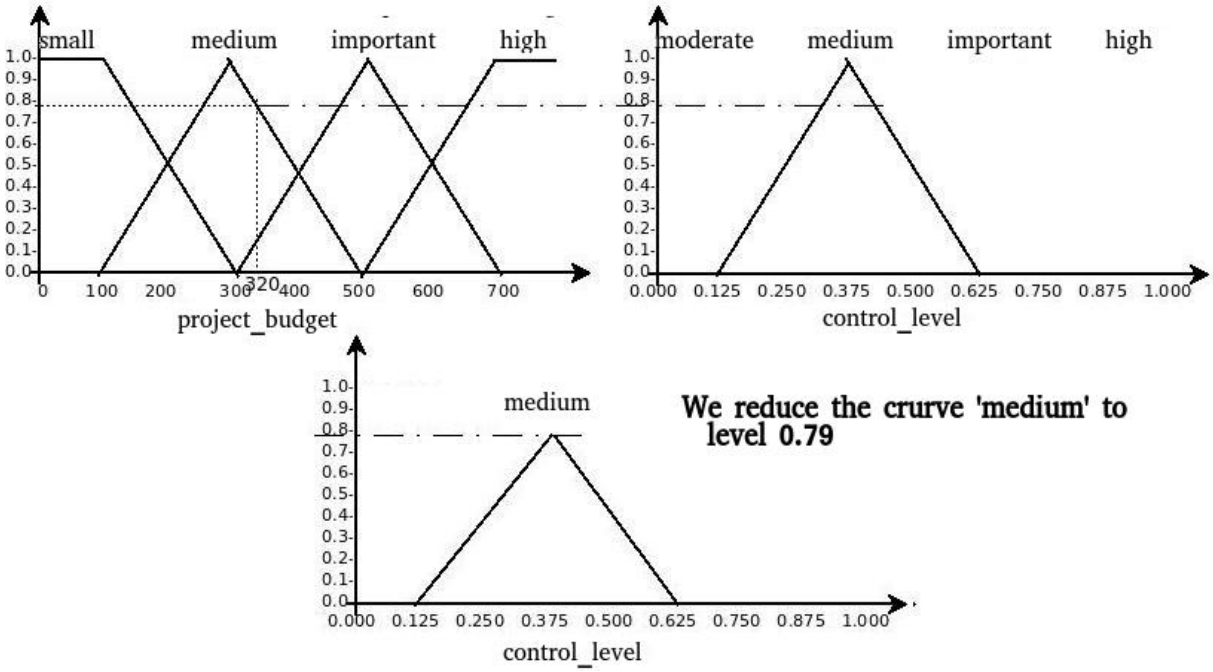
- **R1:** If project-duration is long, then control-level is important.

with project-duration = 4.5 months.

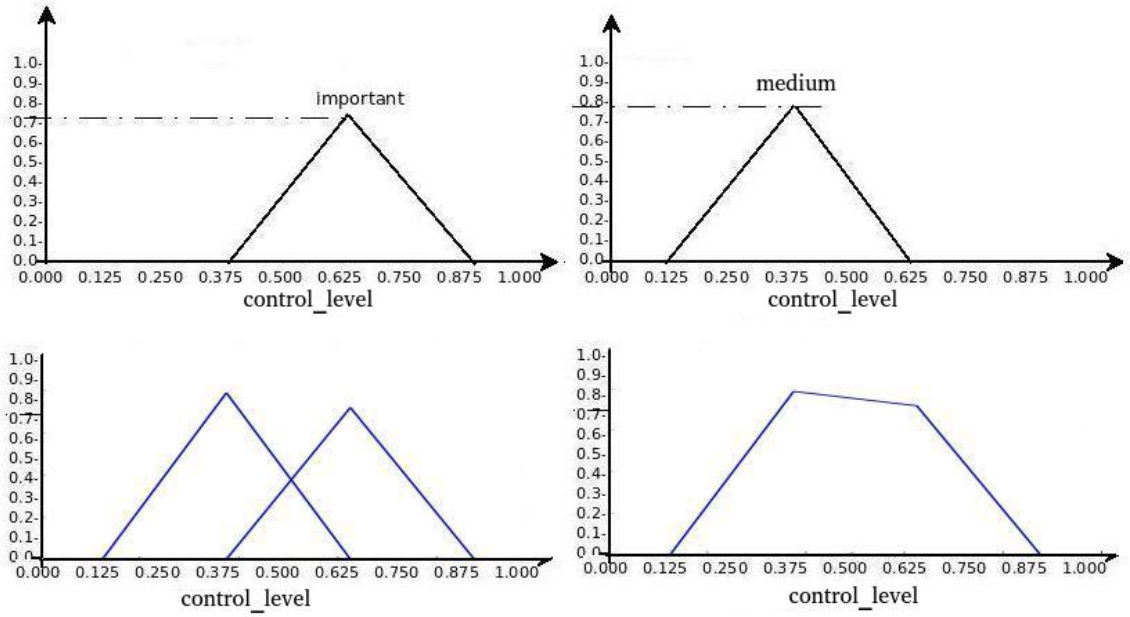


- **R2:**If project-budget is medium, then control-level is medium.

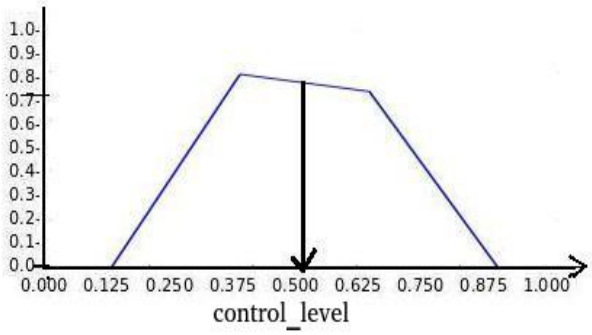
with project-budget = 320.



Additive Aggregation



Centroid Defuzzification



The resulting control level is given by: $= \frac{\Sigma x_i \mu_i}{\Sigma \mu_i} = 0.49421$

Chapter 4

Multi-Agent Systems

4.1 Introduction

Since the early days of artificial intelligence, designing an intelligent machine meant creating a system capable of performing complex tasks as well as a human. However, a human cannot develop properly without being surrounded by others. In other words, social interaction is essential for cognitive development. Similarly, in the field of computing, systems are becoming increasingly complex, requiring decomposition into independent modules with limited and well-controlled interactions (Ferber, 1995).
Sometimes, problems are naturally distributed, such as in the case of explorer robots or air traffic control systems. This necessity leads to the development of distributed intelligence, more specifically, multi-agent systems. A multi-agent system can be defined as a set of entities interacting with each other and with their environment, collectively producing a spatiotemporal organization.

4.1.1 Biological Origin

The algorithms used in multi-agent systems are inspired by observations in biology, particularly ethology. Insect colonies are capable of solving complex problems (such as building a termite mound) despite the fact that each individual insect has limited abilities.
There are social insects, ranging from simple to highly complex and well-organized societies. The most highly organized ones are referred to as eusocial insects. They exhibit the following characteristics:

- The population is divided into castes, with each caste having a specific role.
- Reproduction is limited to a particular caste.
- Larvae and juveniles are raised together within the colony.

The three best-known eusocial species are bees, termites, and ants (Mathivet, 2014).

Bees and the Waggle Dance

The species *Apis Mellifera* (the honeybee) is a eusocial species. Each hive functions as a complete society, containing a queen responsible for laying eggs, worker bees (sterile females), and a few males.
Worker bees have various roles, including caring for the brood (where the larvae are), maintaining the hive, searching for food, collecting food, and defending the hive.

Scout bees are responsible for finding new sources of food (pollen), water sources, resin collection sites, and even new locations for creating or relocating the hive. When they return to the nest, they perform a dance to indicate to the follower bees where to go and what they can expect to find as illustrated in figure 4.1.
If the source is nearby, they perform a round dance, turning in one direction and then the other. The follower bees touch the dancer to determine the type of food found through taste and smell. If the source is farther away—ranging from about ten meters to several kilometers—the scout bee performs a waggle dance in the shape of a figure-eight: the central axis of the figure-eight, in relation to the vertical, indicates the direction of the source relative to the sun. The size and frequency of the waggling indicate the distance and importance of the source. Additionally, the tactile interaction allows the follower bees to assess the quality and nature of the discovered resource. This dance is highly precise, allowing bees to locate food with an error margin of less

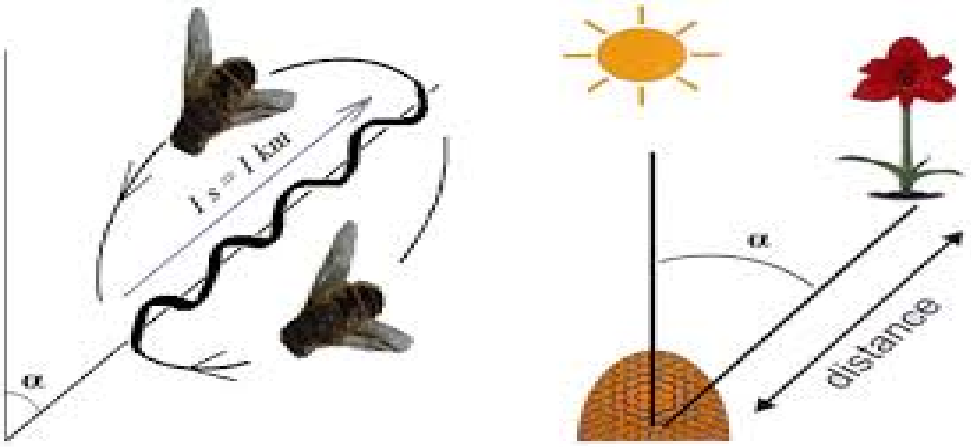


Figure 4.1: the Waggle Dance

than three degrees in direction. Moreover, they adapt to the movement of the sun across the sky and account for wind conditions that may accelerate or delay their flight times.
Thus, each bee has limited capabilities and a simple role, but the combination of different roles and the highly developed communication between individuals enable the hive to survive and thrive.

Termites and Civil Engineering

Termites are also eusocial animals. They live in massive colonies of thousands of individuals. At the center of the colony live the king and the queen, or even secondary queens. They are surrounded by workers, soldiers, juveniles, larvae, and more.
Workers are responsible for food gathering, larval care, and termite mound construction. The termite mound (see figure 4.2)has impressive characteristics: its interior maintains constant temperature and humidity despite extreme and fluctuating external temperatures, particularly in Africa. This internal climate control is due to its very particular structure, which includes wells, a chimney, pillars, and an elevated central nest. This results in passive ventilation. Additionally, termite mounds can reach up to 8 meters in height, with a base circumference of 30 meters.
Studies have investigated how termites can build such structures, known as cathedral mounds. In reality, termites have no awareness of the overall structure or construction plans. Each termite picks up a small ball of soil and deposits it elsewhere with a probability proportional to the number of soil balls already present.
Thus, the complete structure is emergent. Multiple simple agents can collectively solve complex

Figure 4.2: Termites civil Engineering

problems.

Ants and Path Optimization

Ants are also eusocial insects. Their colonies can contain up to a million individuals. One or more queens lay eggs, while the males and workers perform various tasks. Workers have multiple roles, including larval care, nest maintenance, foraging, harvesting, and colony defense.
These colonies thrive due to communication among their members, which is primarily achieved through pheromones detected by their antennae. The primary form of communication is for food source identification. Scout ants move randomly, and if one finds food, it returns to the nest with it, depositing pheromones along the way as presented in figure 4.3. The intensity of these pheromones depends on the food source and the path length. Other scouts encountering a pheromone trail are likely to follow it, with a probability proportional to the amount of pheromone present. Moreover, pheromones naturally evaporate over time.
Thanks to these simple rules, ants can determine the shortest paths between the nest and a

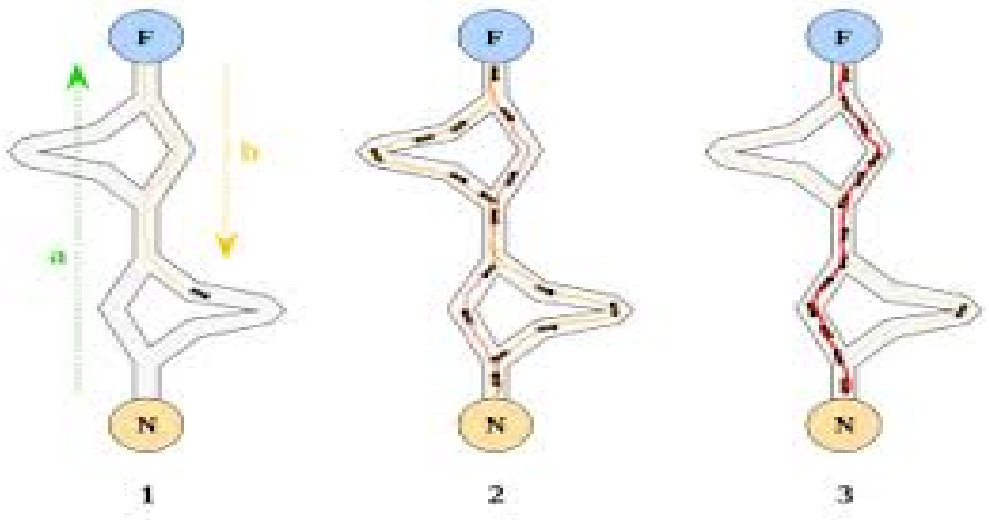


Figure 4.3: Ants path optimization

food source. Longer paths are used less frequently than shorter ones, reinforcing the latter. This form of communication via environmental modifications (pheromone trails) is called stigmergy.

4.2 Some Definitions

4.2.1 Multi-Agent Systems

All techniques classified as multi-agent systems aim to implement this social intelligence, known in computing as distributed intelligence. To achieve this, the following components are required:

- An environment.
- Fixed or movable objects, acting as obstacles or points of interest.
- Agents with simple behaviors.

The algorithm's objective is never explicitly coded; rather, the solution emerges from the interaction of all these elements.

4.2.2 The Environment

Objects and agents exist within an environment, which primarily corresponds to the problem being addressed. The environment can vary in complexity: it can be a bounded space (such as a warehouse or a forest), a graph, or even a purely virtual space. This environment must be able to evolve over time: agents can move within it, and objects can be modified.

4.2.3 Objects

The environment contains objects that agents can interact with. These objects may represent food sources, building blocks, cities to visit, obstacles, and more.
In some systems, there are no objects, only agents. Thus, their presence is optional.

4.2.4 The Agent

Definition by [Fer95]

An agent is a physical or virtual entity that:

a Can act within an environment.

b Can directly communicate with other agents.

c Is driven by a set of tendencies (such as individual goals or an optimization function related to satisfaction or survival).

d Possesses its own resources.

e Can perceive its environment, albeit in a limited manner.

f Has only a partial representation of its environment.

g Possesses skills and offers services.

h May potentially reproduce.

i Exhibits behavior aimed at achieving its objectives while considering its available resources, skills, perception, representations, and received communications.

Definition by Jennings, Sycara, and Wooldridge [JSW98]

An agent is a computing system situated in an environment that acts autonomously and flexibly to achieve its designed objectives. Three key concepts emerge from this definition:

Situated An agent is said to be situated if it can act upon its environment based on sensory inputs received from it.

Autonomous An agent can operate without direct human or other agent intervention, controlling its own actions and internal state.

Flexible An agent is considered flexible if it can act reactively, proactively, and socially.

Reactive The ability to perceive its environment and respond in real time.

Proactive Taking initiative and seizing opportunities at the right moment.

Social The ability to interact with other agents when necessary.

4.2.5 Multi-Agent System

Definition by (Ferber 95): A multi-agent system (MAS) consists of the following elements:

1. An environment E, which is generally a space with a defined metric.
2. A set of objects O. These objects are situated, meaning that at any given moment, a position in E can be associated with each object. These objects are passive, meaning they can be perceived, created, destroyed, and modified by agents.
3. A set of agents A, which are special objects ($A \subseteq O$) representing the system's active entities.
4. A set of relationships R linking objects (and therefore agents) to one another.
5. A set of operations Op enabling agents in A to perceive, produce, consume, transform, and manipulate objects in O.
6. Operators representing the application of these operations and the world's reaction to these modifications, known as the laws of the universe.

4.2.6 Agent Architectures

Two major categories of agents exist: reactive agents and cognitive agents. This distinction primarily concerns the agent's decision-making process and its representation of the environment.

Reactive Agent

A reactive agent merely responds to changes in the environment. In other words, such an agent neither deliberates nor plans; it simply perceives stimuli and reacts accordingly, allowing for rapid action and reaction. Reactive agents can be divided into two subcategories.

Simple Reflex Agent This type of agent, as presented in figure 4.4 acts solely based on its current perceptions. It employs a predefined set of rules, structured as **If** condition **then** action, to determine its behavior. For example, an agent responsible for the defense control of a frigate might follow this rule:
If missile-heading-toward-frigate **then** launch-interception-missile.

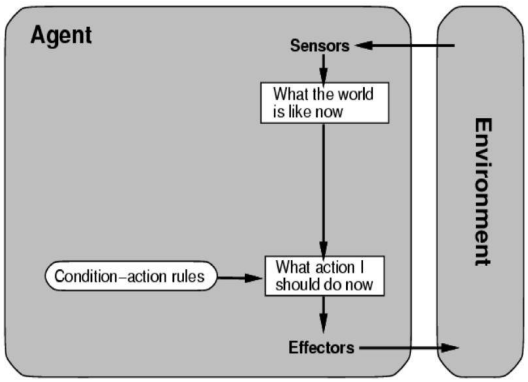


Figure 4.4: Simple Reflex Agent

Agents maintaining a world trace An agent's sensors do not provide a complete view of the world. To address this issue, the agent must maintain internal information about the state of the world to distinguish between two situations that have identical perceptions but are actually different. The agent must have information about how the world evolves and how its own actions affect the surrounding environment (see figure 4.5).
Example: If a missile moves at a speed of 300 m/s, then 5 seconds later, it will have traveled 1500 meters. If the frigate turns, the agent must understand that everything around it also rotates.

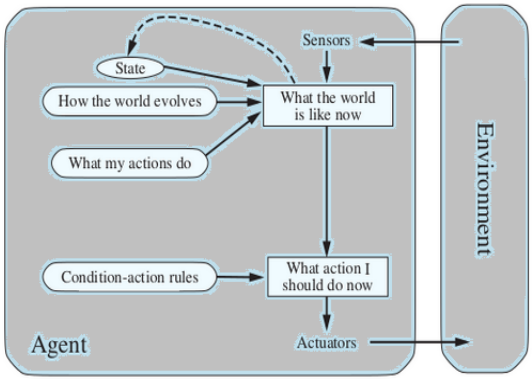


Figure 4.5: Agents maintaining a world trace

Deliberative Agents

Deliberative agents perform some deliberation to choose their actions. Such deliberation can be based on the agent's goals or on a specific utility function.

Goal-Based Agents An agent needs, in addition to the description of the current state of its environment, some information describing its goals as shown in figure 4.6, such as arriving at port X. The agent can then combine goal information with information about the outcomes

of its actions to choose the actions that will enable it to achieve its goals. It must use planning techniques to predict the actions leading to its goal.

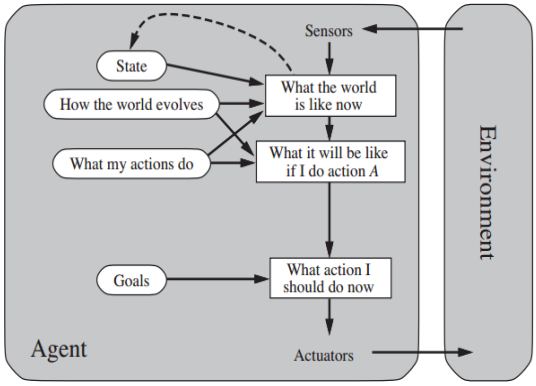


Figure 4.6: Goal-Based Agents

Utility-Based Agents In many situations, goals alone are not sufficient to generate high-quality behavior. For example, if multiple routes lead to a port, some may be faster while others are more dangerous. In this situation, the agent must rely on a finer evaluation method to determine the satisfaction level of each state. Thus, an agent will prefer one state over another if its utility is greater in the first state than in the second. The architecture of this type of agent is presented in figure 4.7

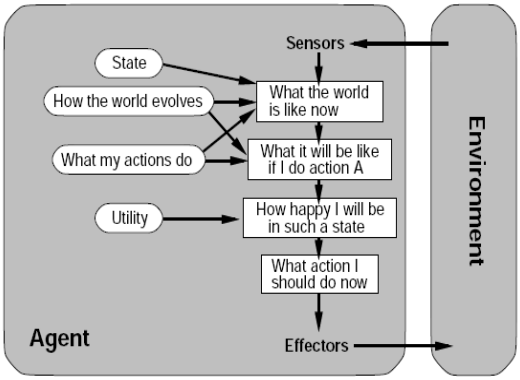


Figure 4.7: Utility-Based Agents

BDI Agents The BDI architecture (see figure 4.8) is another approach used in designing deliberative agents. BDI stands for Belief, Desire, and Intentions. Agents use these three aspects to choose their actions. Wooldridge (99) proposes an architecture with seven components:

- A set of current beliefs, representing the information the agent has about its current environment.
- A belief revision function that updates the agent's current beliefs.
- An option generation function, which determines available options based on the agent's current beliefs and intentions.
- A set of desires, representing the available options for the agent.

- A filtering function that represents the agent's deliberation process and determines its intentions based on its beliefs, desires, and current intentions.
- A set of current intentions, representing the agent's focus or goals it has committed to.
- An action selection function, which determines the action to execute based on the agent's current intentions.

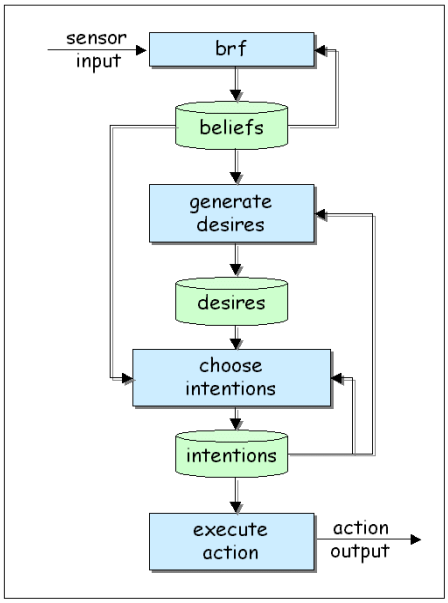


Figure 4.8: BDI Agents

Hybrid Agents

A hybrid architecture typically consists of multiple software layers as presented in figure 4.9. The layers can be arranged vertically (where only one layer has access to sensors and effectors) or horizontally (where all layers have access to inputs and outputs). Most architectures consider three layers to be sufficient. At the lowest level, a purely reactive layer makes decisions based on raw sensor data. The intermediate layer abstracts raw data and works with a knowledge-based representation of the environment. Finally, the top layer handles the social aspects of the environment.

4.3 Typologies of Multi-Agent Systems (MAS)

4.3.1 Reactive MAS

A reactive multi-agent system consists of reactive agents. The study of reactive MAS aims to understand the system's functioning as a whole, focusing on collective aspects such as interactions and emerging dynamics. However, designing such systems faces challenges in predicting global behavior that is not explicitly represented and in controlling individual behaviors relative to an unrepresented objective.

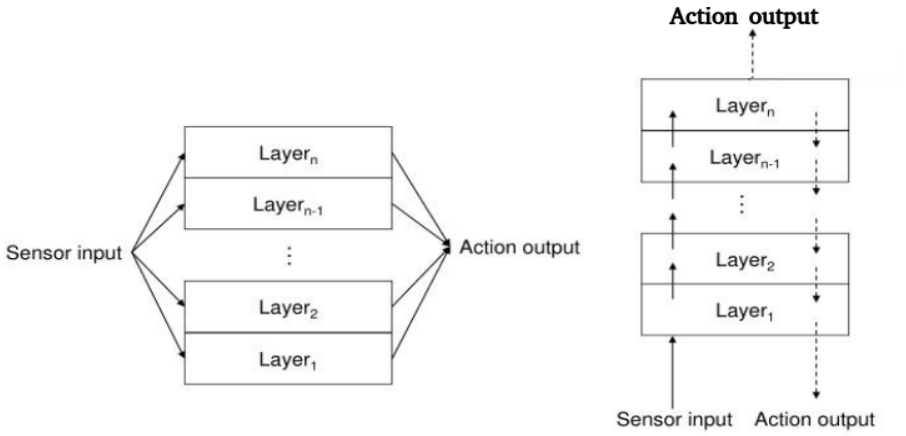


Figure 4.9: Hybrid Agents

4.3.2 Cognitive MAS

A cognitive multi-agent system consists of cognitive agents. The study of these systems aims to improve individual agent behavior by focusing on their intelligence, cognitive models, and communication. This type of system emphasizes the agent and its capabilities. However, it presents some drawbacks, such as the difficulty of representing knowledge in complex problems, communication complexity between agents, low performance for real-time actions, long task execution times, and lack of adaptability in dynamic environments.

4.4 Interactions and Cooperation

An interaction situation is defined as a set of behaviors resulting from the grouping of agents who must act to achieve their objectives while considering constraints such as limited resources and individual competencies.

4.4.1 Simple Collaboration

Compatible goals, sufficient resources, insufficient skills. Simple collaboration consists of a basic combination of skills without requiring additional coordination actions between participants.

4.4.2 Congestion

Compatible goals, insufficient resources, sufficient skills. Congestion characterizes situations where agents hinder each other's task execution despite not needing each other. A typical example is road traffic or air traffic regulation.

4.4.3 Coordinated Collaboration

Compatible goals, insufficient resources, insufficient skills. Complex collaboration requires agents to coordinate their actions to leverage the synergy of their collective skills. Many industrial activities requiring a distributed approach, such as network control, product design and manufacturing, distributed regulation, or autonomous robot societies, fall into this category.

4.4.4 Pure Individual Competition

Incompatible goals, sufficient resources, sufficient skills. When goals are incompatible, agents must compete or negotiate to achieve their goals. In "sportive" competition, all agents have the same resources and start in identical initial situations. An example of pure competition is a foot race.

4.4.5 Pure Collective Competition

Incompatible goals, sufficient resources, insufficient skills. When agents lack sufficient skills, they must form coalitions or associations to achieve their objectives. This grouping follows a dual movement: uniting individuals into collaborative groups and opposing these groups against each other, such as in relay races.

4.4.6 Individual Conflict Over Resources

Incompatible goals, insufficient resources, sufficient skills. When resources cannot be shared, a conflict arises, where each agent seeks to acquire resources exclusively for itself.

4.4.7 Collective Conflicts for Resources

Incompatible goals, insufficient resources, insufficient skills. This type of situation combines collective competition with individual conflicts over resources. Coalitions struggle against each other to obtain a monopoly on a good, a territory, or a position.

4.5 Design of an MAS

Several MAS design methods exist. They can be distinguished into high-level methods and design methods.

4.5.1 GAIA

Gaia considers two levels: a macro-level (which models a society of agents) and a micro-level (which focuses on the agent). Thanks to the richness of its models, Gaia makes it easy to manipulate a large number of multi-agent concepts. The modeled agents, which can be heterogeneous, are computational systems that try to maximize a global measure. The organization is static over time. The autonomy property of agents is expressed by the fact that a role encapsulates its functionality, which is internal and not affected by the environment. Reactivity and proactivity are poorly expressed.

4.5.2 MaSE

MaSE (MultiAgent Software Engineering) manipulates models very similar to those used in object-oriented design (hierarchy, sequence diagrams, agent classes, deployment diagrams). The autonomy of agents, as in Gaia, is ensured by encapsulating functionalities within roles. Likewise, proactivity is specified through automata. However, reactivity and social properties (group, organization) are not clearly defined.

4.5.3 INGENIAS

The general approach to specifying an MAS consists of dividing the problem into several more concrete aspects that form the different views of the system. It is a project that extends the UML notation. INGENIAS is based on the definition of meta-models describing the different aspects of a multi-agent system and their relationships. As in Gaia, the autonomy of agents is ensured by goal encapsulation. The reactivity of agents is achieved through the specification of events and actions in agent/role and interaction models. Similarly, proactivity is linked to goals. Social concepts specific to MAS, such as organization or collaboration, are expressed.

4.5.4 PASSI

PASSI (Process for Agent Societies Specification and Implementation) is a step-by-step method integrating design models and concepts from object-oriented engineering and artificial intelligence using UML. PASSI is generic, applicable to any domain, and has the particularity of taking into account the modeling of mobile agents. It covers almost all phases of the development cycle. Its use is easy since it is based on an extension of the UML language.

4.5.5 PROMETHEUS

This is a complete method (process, notations, and tools) for cognitive (BDI) agents. The autonomy of agents is ensured by encapsulating goals and plans. Although the notion of groups is advanced, the notions of roles are nonexistent, and their management is not addressed.

4.5.6 ADELFE

The ADELFE (Atelier de DEveloppement de Logiciels à Fonctionnalité Emergente) method aims to help any developer design software with emergent functionalities. ADELFE is an interesting approach as it quite comprehensively describes the characteristics of the agents used. However, it remains very specific to systems based on agent cooperation.

4.5.7 VOYELLES Approach

Voyelles is a high-level approach, difficult to classify as a method, but very often cited as it is based on purely multi-agent principles. It is based on the decomposition of a system view into four dimensions (or vowels): Agent, Environment, Interaction, Organization. This approach allows for an associative writing scheme to express the type of approach chosen for system development.
Concepts such as autonomy, reactivity, proactivity, or social aspects are not clearly expressed but assumed.

4.6 Other MAS Design Platforms

4.6.1 Jade

Jade (Java Agent Development Framework) is a platform developed by Telecom Italia under the LGPL license for developing multi-agent systems using the Java language. This platform (presented in figure **??**)is developed around a set of concepts. First, the Runtime Environment represents the environment where agents are located (the containers). There are two types of containers. The main container contains two specific agents: the AMS (Agent Management System) agent, which manages the platform (for example, ensuring the uniqueness of agent

names), and the DF (Directory Facilitator) agent, which provides a list of agents and their services (yellow pages service). This container must be activated when creating agents or creating other normal containers. The set of active containers (main container and other normal containers) forms a platform [BCG07].

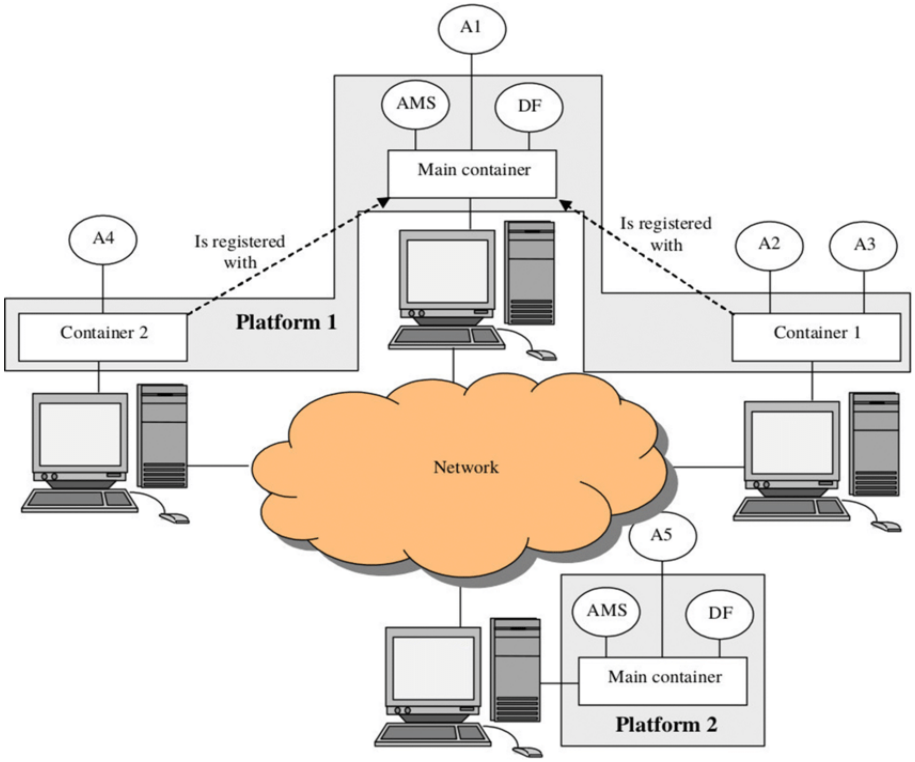


Figure 4.10: Jade platform

4.6.2 PADE

PADE is a framework for the development, execution, and management of multi-agent systems, written in Python and using the Twisted libraries for implementing communication between network nodes. PADE is open source under the terms of the MIT license. Twisted is an event-driven networking engine written in Python and licensed as open source. PADE uses it to implement custom protocols, both for exchanging internal control messages and for providing an interface to the protocols described by the FIPA standard **??**.

4.6.3 Mesa

Mesa is an agent-based modeling framework licensed under Apache2 in Python. It allows users to quickly create agent-based models using built-in core components, visualize them using a browser-based interface, and analyze their results. Mesa is modular, meaning that its modeling, analysis, and visualization components are separate but designed to work together. The modules are grouped into three categories:

- Modeling: Modules used to construct the models themselves: a model and agent classes, a scheduler to determine the sequence in which agents act, and the space for them to move.
- Analysis: Tools for collecting data generated from your model or for running it multiple times with different parameter values.

- Visualization: Classes for creating and launching an interactive model visualization using a server with a JavaScript interface [dpM25].

4.7 Mobile Agents

It was in 1994 that James E. White, affiliated with General Magic Inc. at that time, published a white paper that initiated dedicated research on what we call mobile agents today. A mobile agent [BR05], is a program that can migrate from a starting host to many other hosts in a network of heterogeneous computer systems. It works autonomously and communicates with other agents and host systems. During the self-initiated migration, the agent carries all its code and data, and in some systems it also carries some kind of execution state. The key advantages of mobile agents include:

- **Reduced Network Load**: By processing data locally rather than transferring large datasets, mobile agents help reduce network traffic.
- **Asynchronous Execution**: Mobile agents can operate autonomously, even when disconnected from the source system.
- **Dynamic Adaptability**: They can react to changes in the environment and optimize their behavior accordingly.
- **Fault Tolerance**: Mobile agents can be designed to replicate and recover from failures.

Mobile agents are widely used in various applications, including network management, e-commerce, distributed information retrieval, and collaborative computing.

4.7.1 The Migration Framework

The typical behavior of a mobile agent is to migrate from one agency to another. During the process of migration, the current agency (i.e., the one the agent currently resides on) is called the sender agency and the other agency (to which the agent wants to migrate) is called the receiver agency. During the migration process the sender and the receiver must communicate over the network and exchange data about the agent that wants to migrate. Thus, we can say that some kind of communication protocol is driven, and we call this the migration protocol. Some systems simplify this task to an asynchronous communication, comparable to sending an email, whereas other systems develop rather complicated network protocols in addition to TCP/IP.

4.8 New ISO Architectures for Multi-Agent Systems

The International Organization for Standardization (ISO) has introduced new architectures to ensure the interoperability, scalability, and robustness of multi-agent systems (MAS). These architectures focus on:

ISO standards are crucial for MAS development because they:

- Ensure interoperability between different agent-based platforms.
- Provide guidelines for designing secure and reliable MAS.
- Facilitate compliance with global industry regulations.
- Improve trust in autonomous agent-based decision-making.

4.8.1 ISO/IEC 25010: Software Quality Model

This standard defines the key quality attributes for software systems [Int25, MMSB14], including:

- **Reliability**: Ensuring that MAS can operate under expected conditions without failure.
- **Maintainability**: Allowing modifications and updates to be applied efficiently.
- **Performance Efficiency**: Optimizing response time and resource utilization.

4.8.2 ISO/IEC 19510: BPMN for MAS

The ISO/IEC 19510 standard defines Business Process Model and Notation (BPMN), which is used for modeling interactions between agents in an MAS. This standard helps in:

- Defining workflows for autonomous agents.
- Ensuring clear communication between heterogeneous agents.
- Facilitating integration with external business processes.

4.8.3 ISO/IEC 23894: Artificial Intelligence Risk Management

This standard provides a framework for assessing and mitigating risks in AI-based MAS. It covers :

- Ethical concerns in decision-making by autonomous agents.
- Bias detection and transparency in multi-agent interactions.
- Security risks related to malicious agents.

By adopting ISO-compliant architectures, developers can create MAS that are more robust, scalable, and adaptable to real-world applications.

Chapter 5

Deep Learning. Convolutional Neural Networks

5.1 Introduction

Despite the early success of neural networks and the backpropagation algorithm in the 1980s, a major limitation soon became apparent: as the number of layers increased, training became less and less effective. In other words, networks could not be made very deep. Researchers at the time proposed normalization and regularization methods, but these improvements were not sufficient to overcome the problem.
Deep learning algorithms, convolutional networks, and backpropagation already existed as early as 1989. The LSTM algorithm for time series was introduced in 1997. However, it was not until 2012 that the first truly deep networks managed to train successfully. This delay illustrates a key principle: progress in AI is driven by experimental achievements; theoretical advances alone are not enough. Breakthroughs in hardware, the availability of large-scale datasets, and the introduction of new ideas were all necessary for deep learning to emerge [Cho17].
Computer vision in particular requires high computational power and speed. The introduction of GPUs (Graphics Processing Units) in the 2000s by NVIDIA and AMD enabled massive and parallel processing of 2D and 3D imaging data. At the same time, the growth of the internet made it possible to collect huge datasets of images, videos, speech, and text, fueling progress in both computer vision and natural language processing. Public competitions also contributed by creating large benchmark datasets and driving the development of increasingly effective deep learning algorithms.
Finally, deep learning was further accelerated by improvements in the core algorithms: the adoption of new activation functions, better weight initialization strategies, and more advanced optimization methods such as RMSProp and Adam.

5.2 Convolutional Neural Networks (CNN)

Convolutional Neural Networks are neural networks that, instead of relying solely on general matrix multiplications, make use of the convolution operation in at least one of their layers.

5.3 The Convolution Operation

Convolution, in its general form, is an operation on two continuous functions, denoted with an asterisk, and defined as follows:

$$s\left(t\right) = \left(x * w\right)\left(t\right) = \int_{-\infty}^{+\infty} x\left(a\right)\ w\left(t-a\right)\ da$$

Such that:

- x: the input signal (e.g., a sound, a continuous function).
- w: the kernel (also called filter or the impulse response of a system)
- t: the shift at which the result is computed

In neural networks, the output of the convolution operation is called a feature map.

When working with data on a computer, the input is fed step by step into a neural network. Time is therefore discretized, which requires a discrete version of the convolution operation, defined as follows:

$$s\left(t\right) = \left(x * w\right)\left(t\right) = \sum_{a=-\infty}^{+\infty} x\left(a\right)\ w\left(t-a\right)$$

For two-dimensional input data, such as an image I, a two-dimensional kernel K is required, and thus we obtain a 2D convolution:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)\, K(i-m, j-n)$$

We can equivalently write:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)\, K(m,n)$$

Note that the input values and the kernel values are considered to be zero outside their respective dimensions. The convolution operation can also be written in the form of a cross-correlation, as follows:

$$S(i,j) = (I \star K)(i,j) = \sum_m \sum_n I(i+m, j+n)\, K(m,n)$$

This latter formula is widely used in machine learning algorithms. The learning algorithm will learn the appropriate values of the kernel. Figure 5.1 illustrates an example of a convolution operation applied to a 2D image. The kernel size must be much smaller than the size of the image, which makes the network architecture independent of the image size and significantly reduces the number of parameters that the learning algorithm has to learn. It also enables parameter sharing, since a kernel element is used multiple times in a convolutional network, whereas in a simple fully connected network, each weight is used only once [GBC16]. In practice, the kernel window slides over the input image, and at each step the result of the convolution corresponds to one cell (or pixel) in the resulting feature map. As shown in Figure 5.1, starting from an input image (feature map) of size 3x4 and using a 2x2 kernel, the dimensions of the output feature map are reduced to 2x3.

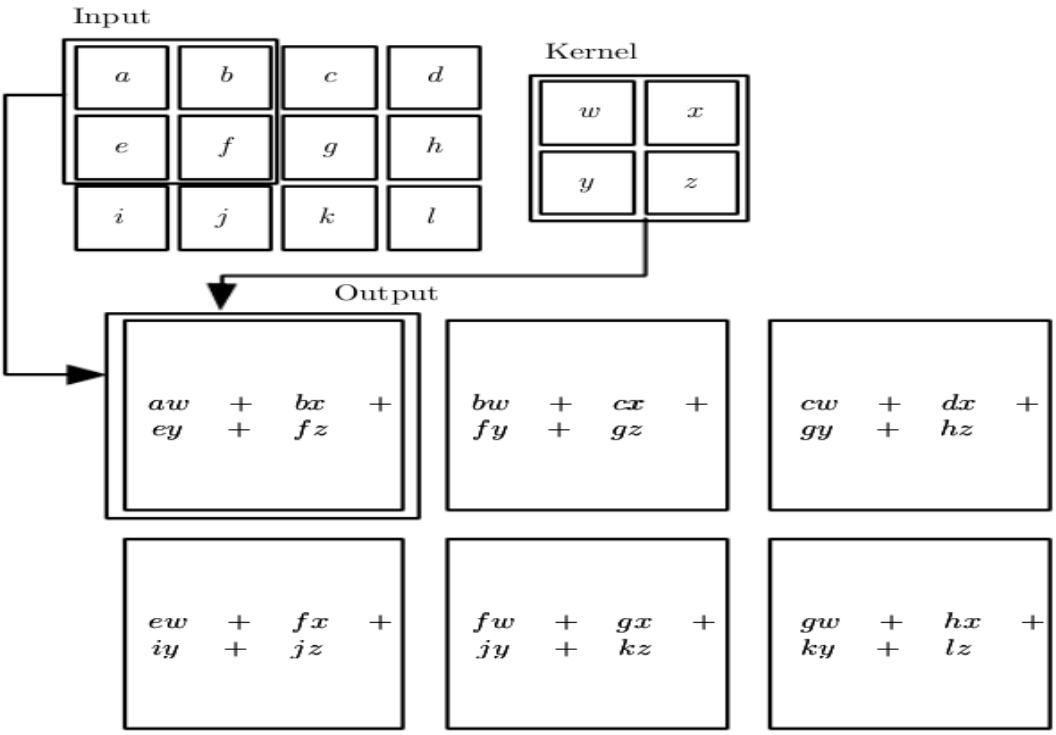


Figure 5.1: An example of 2-D convolution (*from [GBC16]*).

5.3.1 PADDING

If we have an input feature map of size 5x5 and a kernel of size 3x3, the output feature map will be of size 3x3, as shown in Figure 5.2.

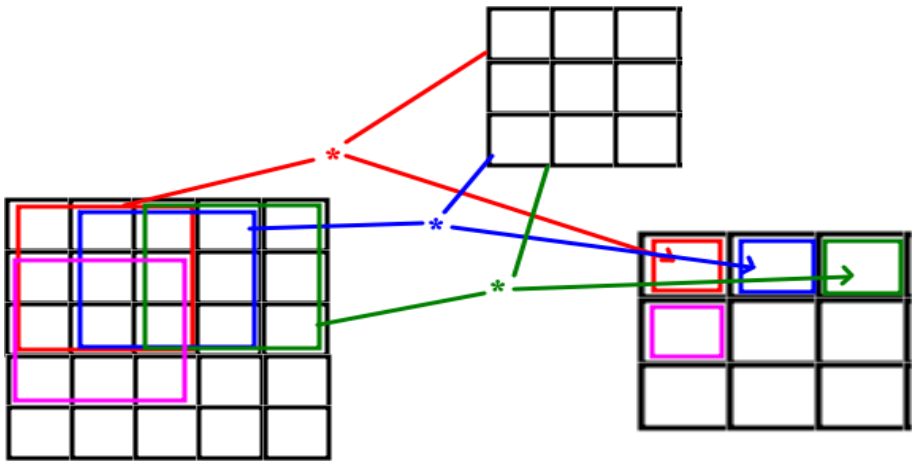


Figure 5.2: An example of 2-D convolution without padding.

If we want the output feature map to have the same size as the input, which is not always necessary, we use padding, i.e., we add a certain number of rows and columns around the input map, thereby increasing its initial size, as illustrated in Figure 5.3.

In programming, the padding of a layer is configurable through the padding property. The value valid indicates no increase in the input map, meaning no padding, while the value same applies adequate padding so that the output map has the same size as the input map [Cho17].

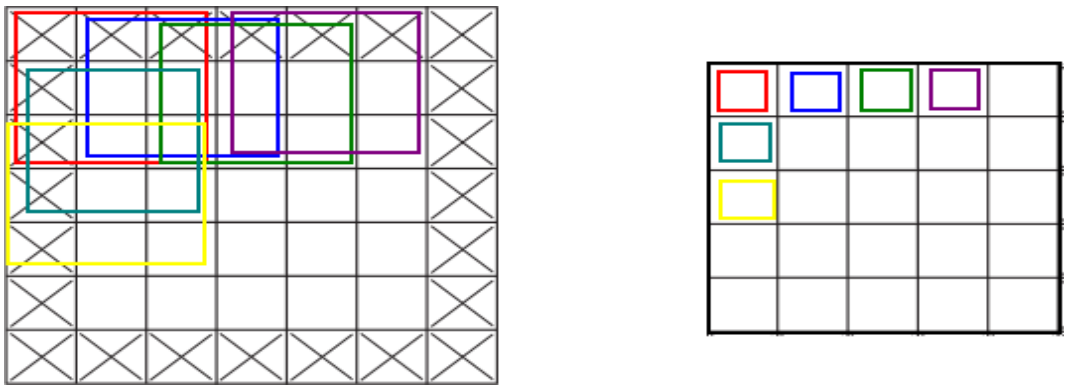


Figure 5.3: An example of 2-D convolution with padding.

5.3.2 Strides

Another factor that affects the size of the output feature map is the step by which the kernel window moves over the input map, which is set to 1 by default, as shown in the previous examples. This step is called the stride. However, it is possible to have a convolution stride greater than 1. The example in Figure 5.4 shows a convolution with a 5x5 input feature map, a 3x3 kernel, and a stride of 2.

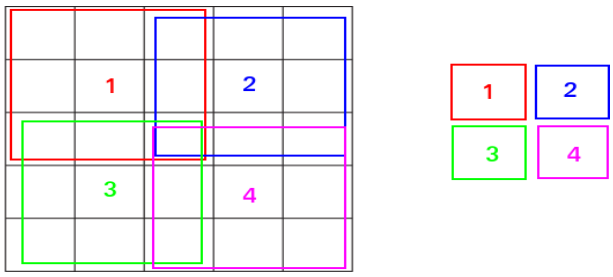


Figure 5.4: An example of 2-D convolution with a 5x5 input feature map, a 3x3 kernel, and a stride of 2, the resulte feature map is 2x2.

In practice, a convolutional layer involves three steps. In the first stage, the layer performs several convolutions in parallel to produce a set of linear activations. In the second stage, each linear activation is passed through a nonlinear activation function, such as the rectified linear unit (ReLU), implemented by a neuron.

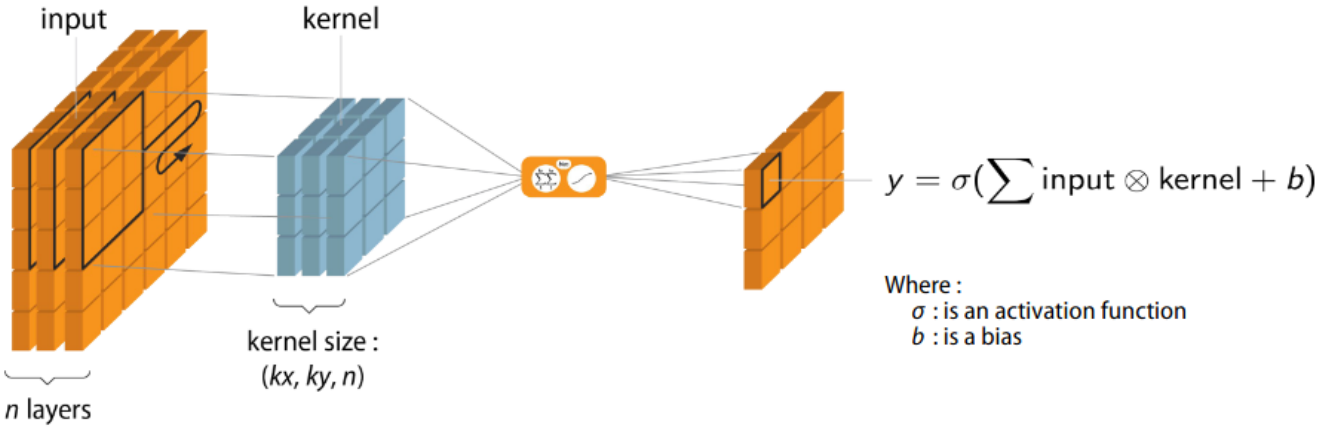


Figure 5.5: An example of convolution layer with an ectivation function (from [GA22]).

The number of parameters to learn corresponds to the number of elements in the kernels plus one bias for each neuron, as illustrated in Figure 5.5. This stage is sometimes referred to as the detector stage. In the third stage, a pooling function is applied to further modify the output of the layer.

5.4 Pooling

The pooling function summarizes the content of the resulted feature map from the previous step by reducing their size. Several pooling functions can be used: max pooling, average pooling, weighted average, and so on. Max pooling consists of scanning the feature map with a small window, usually of size 2×2, and taking only the maximum value. Thus, if the feature map is, for example, of size 26×16 before the pooling operation, it becomes 13×8 after pooling, as shown in Figure 5.6.

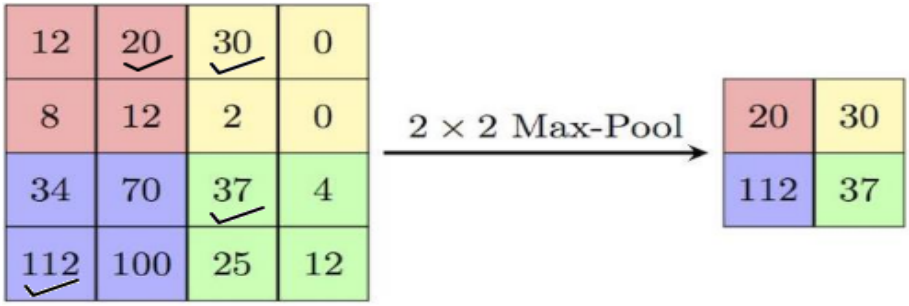


Figure 5.6: An example of pooling operation with a featuremap 4x4 the result is a map of 2x2.

5.5 The Architecture of a Convolutional Network

In a convolutional network, each kernel operates as a filter to extract a specific type of feature from the input image. Since the goal is to extract multiple (if not all) types of features, several kernels (i.e., several convolutions) are applied in parallel in each convolutional layer. The number of neurons corresponds to the number of convolutions (kernels).

Obviously, multiple convolutional layers are needed to extract information at different levels. At each stage, the feature map resulting from one convolutional layer becomes the input for another convolutional layer with different kernels in parallel, followed by a pooling operation, as shown in Figure 5.7.

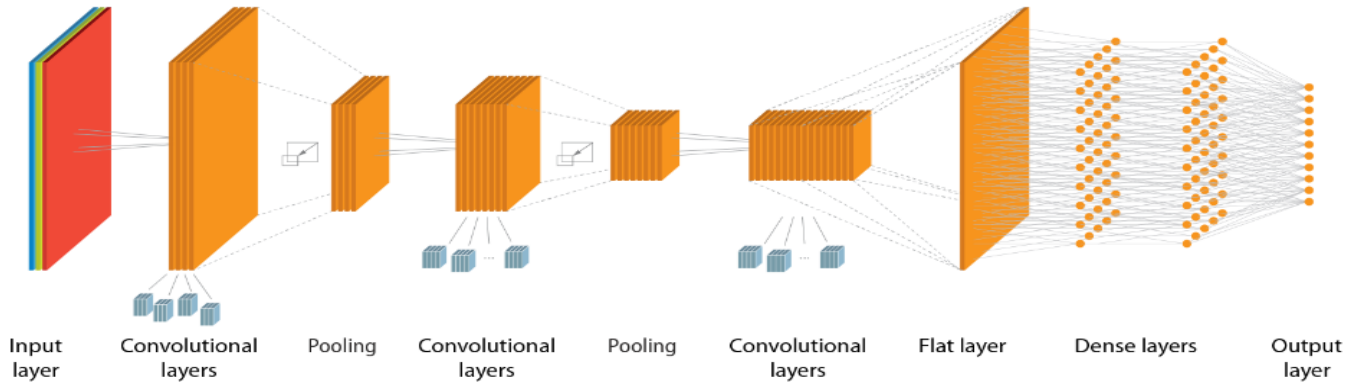


Figure 5.7: An example of convolutional network architecture (from [GA22]).

After a certain number of convolutional layers, we obtain small separated patches in the feature map, and applying further convolutions is no longer useful, since all the important information is assumed to have been extracted. At this stage, all these small maps are combined into one large flattened map, as illustrated in Figure 5.7. This is the flattening layer. The resulting flattened map is then fed into a traditional fully connected network to meet the network's objective (classification, recognition, regression, etc.).

5.6 Pre-trained Convolutional Networks

In deep learning, especially in computer vision, pre-trained convolutional neural networks (CNNs) are often used. These networks have been trained on large datasets (such as ImageNet) and are employed either for feature extraction (to obtain image representations) or fine-tuning (to adapt the network to a new task with fewer data).
Among the most well-known pre-trained CNNs are the classical architectures trained on ImageNet. **AlexNet** (2012) was the first breakthrough CNN, which revolutionized computer vision. **VGG16** and **VGG19** (2014) proposed a simple yet effective architecture based on stacks of 3×3 convolutions. **GoogLeNet**, also known as Inception v1 to v4, introduced deeper networks with inception modules. **ResNet** (2015) brought the concept of skip connections, also called residual connections, which made it possible to train very deep networks efficiently. **DenseNet** (2017) further improved connectivity by introducing dense connections between layers.

More modern CNNs have been designed to achieve better efficiency. **EfficientNet** (2019) optimized the balance between network depth, width, and input resolution, providing state-of-the-art results with fewer parameters. Other architectures such as **RegNet** and **ConvNeXt** were inspired by the Transformer family while keeping the CNN paradigm.

In addition, lightweight architectures were developed for specific use cases, particularly in mobile and embedded systems, where computational resources are limited. Examples include **MobileNet** and **ShuffleNet**, both designed for efficiency, as well as **SqueezeNet**, which is lightweight and fast.

Pre-trained versions of these models are widely available in popular deep learning frameworks such as **PyTorch** (via torchvision.models), **TensorFlow/Keras** (via tf.keras.applications), and even through **HuggingFace Transformers**, which now provide vision models in addition to natural language processing.

Bibliography

[BCG07] Fabio L. Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing Multi-Agent Systems with JADE*. Wiley Series in Agent Technology. John Wiley & Sons, 2007.

[BR05] Peter Braun and Wilhelm Rossak. *Mobile Agents: Basic Concepts, Mobility Models, and the Tracy Toolkit*. 01 2005.

[Cho17] François Chollet. *Deep Learning with Python*. Manning Publications, 2017.

[DBC+24] DeepSeek-AI, Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Yiyuan Liu, Wenfeng Liang, and et al. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*, Jan 2024. Open-source release of 7B/67B LLMs outperforming LLaMA-2 70B :contentReference[oaicite:1]index=1.

[dpM25] Equipe du projet Mesa. Mesa: Agent-based modeling in python — documentation officielle, 2025.

[Fer95] Jacques Ferber. *Les systèmes multi-agents : vers une intelligence collective*. InterÉditions, Paris, 1995.

[GA22] Université Grenoble-Alpes. Formation fidle (saison 2022-2023). https://cloud.univ-grenoble-alpes.fr/s/?dir=/Saison2022. Consulted: 2025-09-13.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[Int25] International Organization for Standardization. ISO 25010: Software Product Quality Models, 2025. Accessed: 2025-02-16.

[JSW98] Nicholas R. Jennings, Katia Sycara, and Michael Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

[LH07] Shane Legg and Marcus Hutter. Universal intelligence: A definition of machine intelligence. *Minds and Machines*, 17(4):391–444, 2007.

[MMSB14] Toufik Marir, Farid Mokhati, and Hassina Seridi-Bouchelaghem. Do we need specific quality models for multi-agent systems?: Toward using the iso/iec 25010 quality model for mas. 08 2014.

[Qia99] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999.

[RN10] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 3rd edition, 2010.

[Ros58] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65-6:386–408, 1958.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.