Machine Learning and Pattern Recognition

Gilson Antonio Giraldi

LECTURE NOTES - GRADUATE COURSE
GRADUATE PROGRAM IN NANOBIOSYSTEMS 2021
UFRJ-FIOCRUZ-INMETRO-LNCC

PETRÓPOLIS, RJ - BRASIL
FEBRUARY 2021

To my Family

Maria Thereza

Gabriel and André

October 1, 2021

Preface

Machine learning is a mature research area encompassing methods in computer science, mathematics and statistics. On the other hand, a pattern is a discernible regularity in the world. Both machine learning and patters go together in the sense that we always want intelligent softwares to automate tasks involving pattern recognition (images analysis, computer-aided diagnoses in medicine, for instance). In the context of learning theory, the solution to accomplish this is to allow computers to learn from training data. Specifically, we have machines (algorithms) with internal parameters. The learning process corresponds to the parameters adaptation, obtained in a optimization process that uses the training samples to formalize the concept of learning from experience.

There are a plenty of books and an ocean of scientific paper in machine learning, covering theoretical aspects as well as applications. The following lecture notes cover an introduction to this field for graduate students. The main goal is to organize elements in computer science, mathematics, probability and learning theory to set the foundations for machine learning and pattern recognition that can be covered in a graduate course.

The theory of computation can help the tasks of algorithm analysis. For instance, concepts in computational complexity can be used to quantify resources required to solve a given problem. On the other hand, mathematical concepts in linear algebra and probability are fundamental for data representation and manipulation. Calculus in several variables and optimization theory are the tools to fine-tune internal parameters of machines, like neural networks, for instance. At the end of each chapter, there is a list of exercises that applies the concepts to reinforce machine learning aspects. Moreover, to help readers not familiar with foundations in mathematics, we let the first chapters to review set theory, linear algebra, calculus, and probability theory.

Contents

List of Figures

Chapter 1

Introduction

Nowadays, the idea of learning from data is present in almost every aspect of modern technology. In these applications, and everywhere in this monograph, learning theory means methodologies that enable computational systems to solve complex problems by adaptively improving their performance using data samples of the involved space. Learning processes are in the heart of machine learning techniques. These methods fulfill the ever increasing requirement for automated methods for data analysis, which is a demand due to the enormous amounts of data in digital form.

Machine learning is thus related to statistical learning, the theory that explores ways of estimating functional dependency from a given collection of data [7]. Pattern recognition belongs to this general statistical problem. Patterns can be described as consistent and recurring characteristics or traits that help in the identification of a phenomenon or problem, and may serve as indicators for predicting its future behavior [8]. From a statistical viewpoint, pattern recognition is one of the simplest models of statistical - or inductive - inference [9].

Statistical inference has more than 200 years, including names like Gauss and Laplace. However, the systematic analysis of this field started only in the late 1920s. By that time, an important question to be investigated was how to find a reliable method of inference, that means, to solve the problem: *Given a collection of empirical data originating from some functional dependency, infer this dependency* [7].

The analysis of methods of statistical inference began with the remarkable

works of Fisher (unified framework of parametric statistics) and the theoretical results of Glivenko and Cantelli (convergence of the empirical distribution to the actual one) and Kolmogorov (the asynptotically rate of that convergence). These events determined two approaches to statistical inference: The particular (*parametric*) inference and the *general* inference [7].

The parametric inference aims to create statistical methods for solving particular problems. Regression analysis is a know technique in this class. This discussion is in the context of parametric statistics which conjectures that data samples can be drawn from a probability distribution with a fixed set of parameters that models the corresponding population [10]. On the other hand, the general inference aims to find one induction method that can be applied for any statistical inference problem. Learning machines, like Perceptron [11] and SVM [7, 12] are nice examples in this area.

Both statistical learning and machine learning are data dependent. However, statistical learning models are yielded based on the assumption that data has certain regularity, such as normality and independence. Consequently, we can process data using probabilistic methods [13]. Machine learning models, in general, do not explicitly consider such assumptions and in most of the cases ignore them [14]. However, this distinction is not rigid. We can adopt a probabilistic view for machine learning and use the tools of probability theory to model related problems involving uncertainty, for instance [15]. In machine learning, uncertainty comes in the analysis of machine predictions, data characteristics, choice of the best model given some problem, etc.

The machines learn through a process named training. Specifically, the machine has internal parameters and training involves providing a learning algorithm and (training) data to automatically adapts the machine parameters to solve the focused problem (classification, regression, etc.). Mathematically, the machine parameters are also degrees of freedom of an objective (loss) function which is iteratively optimized during the training process. In each iteration, the learning rule inside the learning algorithm updates the machine parameters towards a critical point of the objective function (point where the gradient vanishes). In this way, machine learning methods can be divided into four main types: supervised, unsupervised, semi-supervised, and reinforcement learning

In the supervised or predictive learning approaches, the data is composed by

a set of labeled samples and the goal is to learn a map from the samples to the corresponding labels. In this monograph, the samples are represented by points in $\mathbb{R}^n$. The set of pairs formed by the samples and corresponding labels is called the training set. The loss function incorporates an error term to measure how close is the machine response to the target labels. The labels can be a categorical or nominal variable that takes values from some finite set. In this case, the problem is known as classification or pattern recognition. Moreover, if the desired output is a real-valued scalar variable then we are in face with a regression problem [15].

The second category of machine learning algorithms, the descriptive or unsupervised learning algorithms, only receive data samples (points in $\mathbb{R}^n$ in this monograph), and the aim is to perform knowledge discovery. This is an ill-defined problem, since we do not know the target patterns. The construction of the loss function and its error term are also more trick because, unlike supervised learning, it is not given the desired prediction to be compared with the machine output.

Semi-supervised learning, the third class in our taxonomy, encompasses machine learning techniques that use labeled and unlabeled data for training [15]. Reinforcement learning [16], the last type of machine learning is our taxonomy, has as its essence to learn through interaction. It is easier to think with agents instead of an algorithms. So, the agent interacts with its environment and learns the way to alter its own behavior in response to rewards received [16].

However, before bringing our data into a machine learning model, we must prepare the data to ensure that it is clean, consistent, and accurate [17]. Roughly, data preparation (or preprocessing) methodologies involves the following steps [1]: (i) Data collection; (ii) Data exploration by looking for outliers, exceptions, and missing information; (iii) Cleaning, composed by stages for removing incorrect and inconsistent samples; (iv) Formatting data, which encompasses normalization and more general transformations; (v) Feature engineering, which refers to the process of designing methods for feature computation from the raw data for compact representation; (vi) Data augmentation; (vii) Splitting data into training, validation and test sets.

Depending on the methodology and application, we can bypass some of these steps. For instance, feature engineering may be replaced by feature learning. In this case, a machine learning algorithm (like a convolutional neural network - CNN [18]) could be used for feature extraction [19, 20]. Consequently, after

formatting data, we can go to steps (vi)-(vii) to train the CNN to generate features from data. Moreover, other data management tasks like as error detection and data integration could be automated by using a machine learning model [21].

These lecture notes cover fundamental aspects in machine learning and statistical learning for graduate students. From the above explanation, we notice that the machine/statistical learning foundations branch into computer science, mathematics, statistics and probability theory.

The theory of computation can help the tasks of algorithm analysis. For instance, concepts in computational complexity can be used to quantify resources required to solve a given problem. On the other hand, mathematical concepts in linear algebra and statistics are fundamental for data representation and manipulation. Calculus in several variables and optimization theory are tools applied to fine-tune internal parameters of machines. In this way, the remainder of the material is organized as follows.

- Chapter 2: Set Theory, Induction, and Rates of Growth
  In order to allow students to travel safe in this avenue, it is necessary to offer some background in discrete mathematics. Set theory gives the official language to formalize concepts in data science involving data organization. This chapter also aims to give the student the fundamental of mathematical induction, which is used to prove results involving integer numbers. Algorithms in machine learning execute too many floating-point operations during training process. We need to quantify the amount of such operations in order to analyze computational complexity in machine/statistical learning. Big-O, Ω, and Big-Theta notations characterizes rates of growth, to measure the complexity of algorithms.

- Chapter 3: Boolean Expressions, Normal Forms and Satisfiability
  This Chapter presents Boolean expressions to set the background to study learning machines as computational models. For instance, the universality of neural networks will be demonstrated inside the context of Boolean expressions.

- Chapter 4: Linear Algebra and Matrices.
  Training data must be represented in some space (Euclidean, non-linear, etc.) which, in turn, is parameterized through coordinates in some $\mathbb{R}^n$.

Hence, basic data transformations involve linear operators and matrices, which are developed in this chapter.

- Chapter 5: Calculus of Several Variables and Optimization.
  The heart of the learning algorithm is an optimization technique that, in general, uses elements of calculus of several variables. Analytic geometry is also revised in this chapter in order to summarize geometric concepts, like hyperplanes and hypersurfaces implicitly represented, that are useful to understand results of learning processes.

- Chapter 6: Probability and Information Theory
  Tools in probability theory give ways to design machines that can learn from data and to perform their analyzes. Also, available databases are obtained by sampling processes that need probability elements to get some measure of reliability. Consequently, this chapter aims to offers basic principles on random signals, their representations by stochastic models and some elements of classical information theory.

- Chapter 7: Data Preparation and Patterns Representation
  This chapter discusses steps for preprocessing data to convert it into an appropriate format for data mining through machine learning approaches. Hence, operations like cleaning, dimensionality reduction, feature extraction, and normalization are discussed in this chapter. The text focuses in data whose attributes (variables) are measured in numerical values (continuous or discrete). The Chapter presents a geometric data model whose main assumptions are that the data points, or samples of the database, lye in a hypersurface $\mathcal{M} \subset \mathbb{R}^m$ and appear according to some probability density function (*pdf*).

- Chapter 8: Data and Learning Theory
  In this chapter the concept of learning on the basis of examples is formalized. The geometric-based model for data presented in the previous chapter is used to discuss some aspects of the learning problem. Learning machines encapsulates functional spaces due to their internal parameters (weights). Formal aspects related to data synthesis, classification, regression and probability density estimation are discussed within this viewpoint. Special attention is given to the training process and the steepest descent approaches to

optimize the Loss function, with respect to the machine parameters, during the training stage. Curse of dimensionality, overfitting, and overtraining are described in this context. The circuit theory is developed in order to demonstrate the universality of the neural networks model that will be developed in the next chapter. The chapter ends with some considerations about manifold learning and machine learning, steered by the geometric-based data model. Aspects regarding shallow versus deep neural networks are also considered,

- Chapter 11: Computing with Neural Networks
  This chapter develops introductory material in the field of neural networks. it starts with the first artificial neuron model, named perceptron, following by the multilayer perceptron architecture. The universality of the computational paradigm of neural networks is also presented in this chapter through the answer to the question: What kind of functions can be computed by neural networks? The development is made in the context of discrete mathematics using the circuit theory of chapter 8. Aspects regarding shallow versus deep neural networks are also considered, followed by the presentation of the back-propagation algorithm machinery.

- Chapter 10: Statistical Learning Theory
  In this chapter we discuss some aspects os statistical learning theory related to the Principal Component Analysis (PCA), Support Vector Machine (**SVM**), Discriminant Principal Components Analysis (**sec:DPCA**), Linear Discriminate Analysis (**LDA**) and kernel methods. The goal is to set a framework to discuss dimensionality reduction, discriminant analysis, classification and reconstruction. The material to be presented follows the references [7, 22, 11].

- Appendix A: Convolutions and Linear Processes
  Develops some elements in convolution theory and Fourier analysis for two-dimensional signal processing.

Chapter 2

Set Theory

In this Chapter we review some basic ideas in set theory. This theory offers an efficient background for formalization and demonstration of important results in the theory of computation. Besides, the notion of cardinality, to be develop in what follows is a fundamental one for computability and complexity theories. Thus, we start with the notion of sets and usual set operations (sections 2.2 and 2.1). Then, in section 2.3, we develop the concepts of relations and functions followed by set sequences in section 2.4. Cardinality is discussed in section 2.5 followed by some concepts about topological spaces in section 2.6. Mathematical induction, an important tool to prove results in discrete mathematics, is presented in section 2.7. Some notions in quantification of resources to solve problems through computers are presented in section 2.8 and discussed using the concept of limit of a sequence in 2.9. Finally, a list of exercises is proposed.

2.1 Definitions

A *set* is composed by objects which are its *elements*. If an object x belongs to a set A, we formally write [23, 24]:

$$x \in A, \tag{2.1}$$

otherwise, we say that:

$$x \notin A, \tag{2.2}$$

which means, the element x does not belongs to the set A.

We can define a set A by explicitly give all its elements or through a *rule* which defines its elements. For instance, in these examples:

$$A = \{1, 2, 3, 4, 5\}, \tag{2.3}$$

$$B = \{x; \quad x \quad is \quad vowel\}, \tag{2.4}$$

we give explicitly the elements of the set A but the set B is defined just by a property of its elements.

There are two very special sets. The empty set, denoted by $\emptyset$, which has no elements, and the *university* set, which can be defined as:

$$Universe = \{x; \quad x \quad is \quad object\}. \tag{2.5}$$

Observe that in the definition of the $Universe$ set, the term *object* is used in the general sense; that is, anything we can image is an element of the $Universe$ set.

An important relation between sets is the *inclusion* one:

$$A \subset B \Leftrightarrow \quad a \in A \quad then \quad a \in B. \tag{2.6}$$

Definition 1 *Given two sets A, B we say that: $A = B$ if and only if $A \subset B$ and $B \subset A$.*

Theorem 1 *The following properties can be easily demonstrated:*

1) Reflexive: $A \subset A, \quad \forall A$,

2) Anti-Symmetric: if $A \subset B$ and $B \subset A$, then $A = B$,

3) Transitive: if $A \subset B$ and $B \subset C$, then $A \subset C$.

Prof: Exercise.

Given a set X, we denote by $P(X)$ the set of the *parts* of X, which is given by:

$$P(X) = \{A; \quad A \subset X\} \tag{2.7}$$

As an example, let us consider the set $P(A)$ where $A = \{a, b, c\}$. According the definition above, $P(A)$ in this case is given by:

$$P(A) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, A\}. \tag{2.8}$$

Observe that $\emptyset$ is a subset of A and, consequently, $\emptyset \in P(A)$. In fact, it can be proved that:

Property: $\emptyset \subset A$, $\forall A$. (Exercise)

2.2 Set Operations

Given two sets A, B we can define the following operations:

1) $A \cup B = \{x; \quad x \in A \quad or \quad x \in B\}$.

2) $A \cap B = \{x; \quad x \in A \quad and \quad x \in B\}$.

2) $A - B = \{x; \quad x \in A \quad and \quad x \notin B\}$.

3) Cartesian Product: $A \times B = \{(a, b); \quad a \in A \quad and \quad b \in B\}$.

The set $A - B$ is also called the complement of B with respect to A:

$$A - B = C_A B.$$

The following properties can be demonstrated (Exercise):

a) $A \cup B = B \cup A$.

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

c) $A \cup B = A \Leftrightarrow B \subset A$.

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

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

f) If $B \subset E$ then $C_E(C_E B) = B$

g) $A \subset B \Leftrightarrow C_E B \subset C_E A$

h) $C_E(A \cup B) = C_E A \cap C_E B$

Many other properties can be found in [24, 23]. Also, the references [23, 25] offer interesting discussions about the "Paradoxes" of the traditional Set Theory. Moreover, the reference [26] is another interesting material, with a less formal

language, that can be used as an introduction to the philosophical aspects of Mathematics in general and set theory in particular.

2.3 Relations and Functions

A function $f : A \to B$ has four elements:

a) Domain: the set A.

b) Codomain: set B

c) A rule such that for each $x \in A$ we can find **the only** element $c \in B$ such that $f(x) = c$.

d) Range or Image Set: $Im = \{b \in B; \quad \exists x \in A \quad such \quad that \quad b = f(x)\}$ .The image set is also denoted by $f(A)$.

Given two sets A, B we call a relation any subset $\Delta \subset A \times B$. A special relation is the graph of a function f, defined as:

$$G(f) = \{(x, y) \in A \times B; \quad y = f(x)\}. \tag{2.9}$$

The more elementary properties of a function $f : A \to B$ are:

a)Surjective (onto): if $f(A) = B$.

b) Injective (one-to-one): $f(a) = f(b) \Rightarrow a = b$.

c) Bijective: If f is injective and surjective.

Another important concept is the inverse function. If f is bijective than we can define the function f^{-1}, called **inverse function**, as follows:

$$f^{-1} : B \to A; \quad f^{-1}(y) = x \Leftrightarrow f(x) = y. \tag{2.10}$$

Some times it is interesting to extend the concept of inverse function for inverse relations. Given a function $f : A \to B$, not necessarily bijective, we can always define the inverse image, through the relation,

$$f^{-1}(B) = \{x \in A; \quad f(x) \in B\}. \tag{2.11}$$

The following properties are given as exercises:

1) $f(X \cup Y) = f(X) \cup f(Y)$.

2)$f(X \cap Y) \subset f(X) \cap f(Y)$.

3)$f^{-1}(X \cup Y) = f^{-1}(X) \cup f^{-1}(Y)$.

4)$f^{-1}(\emptyset) = \emptyset$.

See [24],[23]. for a more complete list of properties.

2.3.1 Composition of functions

Given two functions $f : A \to B$ and $g : B \to C$, such that the domain of g is the codomain of f, we can define the composite function $g \circ f : A \to C$ as follows:

$$g \circ f(x) = g(f(x)), \quad \forall x \in A. \tag{2.12}$$

The composition is associative. In fact, given three functions f, g and h we can show from the above definition that:

$$g \circ (f \circ h) = (g \circ f) \circ h. \tag{2.13}$$

From these definitions, it is possible to show the following property: The set $\Phi = \{f : A \to A; \quad f \quad is \quad bijective\}$ is a group (see Definition 2) respect to the composition operation (Exercise).

Definition 2 *Group Definition*

A group is a pair $(G, \cdot)$ where G is a non-empy set and:

$$\cdot : G \times G \longmapsto G$$

is a mapping, also called an operation [27], with the following properties:

(1) Associative. If $g, h, k \in G$, then:

$$g \cdot (h \cdot k) = (g \cdot h) \cdot k.$$

(2) Unity element. There is an element $e \in G$, named unit element, such that:

$$e \cdot g = g \cdot e = g,$$

for all $g \in G$.

(3) Inverse. For any $g \in G$ there exists an alement $g^{-1} \in G$, called inverse, that satisfies:

$$g^{-1} \cdot g = g \cdot g^{-1} = e.$$

■

2.4 Sequences of Sets

Let L be a set which elements we will call indexes. Thus, given a set X, we call a sequence of elements of X, with indexes in L, a function $f : L \to X$. If we have a sequence of sets A_λ then we usually write the sequence as $(A_\lambda)_{\lambda \in L}$

Now, we can extend the operations of union and intersection for sequences:

$$\bigcup_{\lambda \in L} A_\lambda = \{x; \quad \exists \lambda \quad such \quad that \quad x \in A_\lambda\}, \tag{2.14}$$

$$\bigcap_{\lambda \in L} A_\lambda = \{x; \quad x \in A_\lambda, \quad \forall \lambda\}. \tag{2.15}$$

2.5 Cardinality

For finite sets, the cardinality of a set is its number of elements. For infinite sets, we can say that the cardinality is a measure that compares the size of sets. We can show that two finite sets A and B have the same number of elements by constructing a bijective function $f : A \to B$. Such approach, to compare the size of sets through bijections (*mappings*) can be used for both finite and infinite sets. Therefore, we can state that:

Definition 3 *(i) Two sets A and B have the same cardinality if there is a bijective function $f : A \to B$.*

(ii) The cardinality of a set X is less than or equal to the cardinality of a set Y if there is a injective function $f : X \to Y$.

We denote the cardinality of a set A by $card(A)$. So, the relationships (i) and (ii) are denoted by $card(A) = card(B)$ and $card(X) \leq card(Y)$, respectively. Besides, we say that $card(X) < card(Y)$ if $card(X) \leq card(Y)$ and $card(X) \neq card(Y)$.

A set that has the same cardinality of the set of natural numbers is said to be **countably infinite** or **enumerable**. The term **countable** refers to sets that are either finite or enumerable.

The cardinality of a set is also called a **cardinal number**. When working with cardinal numbers, some amazing facts happen. For example, we can show that the

cardinality of the Cartesian product $\mathbf{N} \times \mathbf{N}$ is the same of the $\mathbf{N}$, as well as that the cardinality $\mathbf{N}$ is the same of the set $\mathbf{Z}$ of all integers (Exercise).

In this section we followed more or less the presentation found in [28]. The interested reader will find interesting comments about the history and theory behind this field of mathematics in [29].

2.6 Topological Spaces

A $topology$ over a set E is a set $\Theta \subset P(E)$ such that [30]:

e1) Let I an index set. If $O_i \in \Theta$, then the union $O = \bigcup_{i \in I} O_i$ is such that $O \in \Theta$;

e2) $O_1, O_2 \in \Theta \Rightarrow O = O_1 \cap O_2$ is such that $O \in \Theta$;

e3) $E \in \Theta$.

The par (E, Θ) é called a *topological space* . The elements of E are called $points$ and the elements of Θ are called open sets. A subset $A \subset E$ is called **closed** if its complement is open.

We must remember that:

$$\bigcup_{i \in \emptyset} O_i = \emptyset,$$

and so, the empty set $\emptyset$ belongs to Θ.

Given a point $p \in E$, we call a **neighborhood** of p any set V_p that contains an open set O such that $p \in O$. A topological space is (E, Θ) is **separated** or **Hausdorff** if every two distinct points have disjoint neighborhoods; that is, if given any distinct points $p \neq q$, there are neighborhoods V_p and V_q such that $V_p \cap V_q = \emptyset$.

It is possible to define the concept of continuity in topological spaces. Given two topological spaces E_1, E_2 and a function $f : E_1 \to E_2$ we say that f is

continuous in a point $p \in E_1$ if $\forall V_{f(p)}$ there is a neighborhood V_p such that $f(V_p) \subset V_{f(p)}$.

2.7 Peano's Axioms and mathematical Induction

Given the set $\mathbf{N}$ of undefined objects called *natural numbers*, and a function $s : \mathbf{N} \to \mathbf{N}$,where $s(n)$ is called the successor of n. The function s satisfies the following axioms [24]:

P1) The function s in injective.

P2) $\mathbf{N} - s(\mathbf{N})$ has only one element.

P3. Induction Principle: If $X \subset \mathbf{N}$ is a subset such that $0 \in X$ and, for all $n \in X$ we have $s(n) \in X$, then $X = \mathbf{N}$.

The Induction Principle can be also stated in another way which is more suitable in practice. Before to re-write it, let us consider any property P concerning natural numbers as a function:

$$P : \mathbf{N} \to \{0, 1\},$$

such that $P(n) = 1$ if P is true for a given n and $P(n) = 0$ otherwise. So, we can re-write the axiom P3) in the following way:

Induction Principle: Let us consider P as a property related to natural numbers. If $P(0) = 1$, and starting from the hypothesis that $P(k) = 1$, it is possible to show that $P(k+1) = 1$ also, then we can conclude that $P(n) = 1$, $\forall n \in \mathbf{N}$.

Demonstration by Mathematical Induction: It is any demonstration in which the Induction Principle is applied.In this case, the demonstration follows the next steps:

(a) Show that $P(0) = 1$,

(b) As the *inductive hypothesis*, we assume that $P(k) = 1$. Then, using this hypothesis we try to prove that $P(k+1) = 1$ also.

(c) From (a), (b), and by the Induction Principle we conclude that $P(n) = 1$, $\forall n \in \mathbf{N}$.

Example: Prove by mathematical induction that:

$$(a-1)\left(1 + a + a^2 + ... + a^n\right) = a^{n+1} - 1.$$

2.8 Rates of Growth

Now, we turn to the question: What can be *efficiently* computed? This is a fundamental question in complexity theory [31, 32, 33], a field of computer science. To deal with this issue, we offer in this section a mathematical background to quantify resources to solve problems through computers. It is important to have in mind that our viewpoint is always independent of specific architectures or physical devices. Therefore, we must consider abstract elements that allow to get fundamental relations with practical consequences. The following definitions are very important to achieve this goal.

We will focus on functions $f : \mathbf{N} \to \mathbf{N}$ although, it may occurs that a finite number of points in the image set is negative. Therefore, let $f, g : \mathbf{N} \to \mathbf{N}$. Then:

Definition 4 *We say that* $f(n) = O(g(n))$ *if there are numbers* c *and* n_0, *such that* $f(n) \leq cg(n)$, *for all* $n \geq n_0$

Definition 5 *We say that* $f(n) = \Omega(g(n))$ *if there are numbers* $c > 0$ *and* n_0, *such that* $f(n) \geq cg(n)$, *for all* $n \geq n_0$.

Definition 6 ***Definition:*** *We say that* $f(n) = \Theta(g(n))$ *if* $f(n) = O(g(n))$ *and* $f(n) = \Omega(g(n))$.

The Figure 2.1 helps to understand the first definition, also named *big-O* notation.

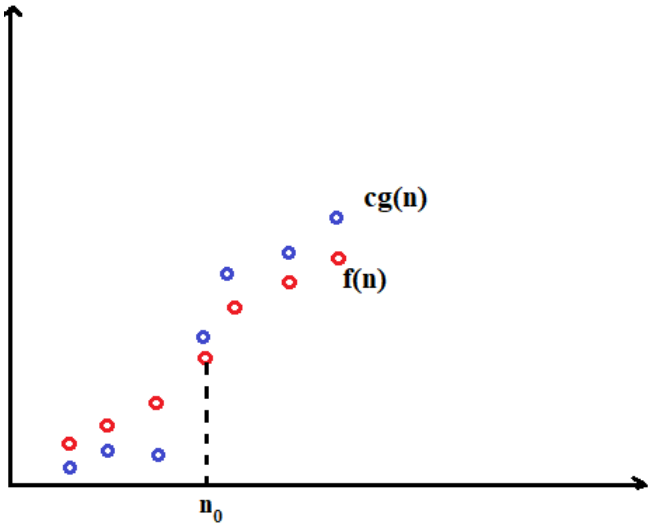


Figure 2.1: Representation of the fact that $f(n) = O(g(n))$.

Application: Calculate the number of floating-point operations to calculate the inner product in $\mathbb{R}^n$.

Solution: Let $\boldsymbol{u} = (u_1, u_2, \ldots u_n)^T \in \mathbb{R}^n$ and $\boldsymbol{v} = (v_1, v_2, \ldots v_n)^T \in \mathbb{R}^n$. Then:

$$\boldsymbol{u} \cdot \boldsymbol{v} = u_1 v_1 + u_2 v_2 + \ldots + u_n v_n,$$

so, we need to calculate $'n'$ products and $'n-1'$ additions. Consequently, the total number N of floating-point operations is represented by the funtion $f : \mathbb{N} \longrightarrow \mathbb{N}$, computed as:

$$f(n) = n + n - 1 = 2n - 1.$$

What is the computational time (CPU time) to perform this operation? Obviously, it depends from the computer architecture. Let us suppose that the CPU time to compute a single floating-point operation in your computer is $\bar{t}$. Then, the CPU time to calculate the inner product in $\mathbb{R}^n$ is given by the function:

$$T : \mathbb{N} \longrightarrow \mathbb{N}$$

$$T(n) = \bar{t}\,(2n - 1)\,.$$

Consequently:

$$T(n) = O(n),$$

independent of the type of the hardware of your computer. To demonstrate this fact, it is just a matter of setting $c = 2\bar{t}$ and $n_0 = 0$ since:

$$2\bar{t}n \geq \bar{t}(2n-1), \forall n \geq n_0.$$

2.9 Limit and Rates of Growth

For a function $f : \mathbb{N} \longrightarrow \mathbb{N}$ the expression:

$$\lim_{n\to\infty} f(n) = L < \infty,$$

means: For every $\varepsilon > 0$ there exists a positive number K such that

$$|f(n) - L| < \varepsilon \; whenever \; n > K.$$

Consequently:

$$L - \varepsilon < f(n) < \varepsilon + L, \; whenever \; n > K.$$

So, if $f(n) = h(n)/g(n)$ with $h : \mathbb{N} \longrightarrow \mathbb{N}$ and $g : \mathbb{N} \longrightarrow \mathbb{N} - \{0\}$, we can write:

$$L - \varepsilon < \frac{h(n)}{g(n)} < \varepsilon + L, \quad whenever \; n > K. \tag{2.16}$$

So, since $h(n) \geq 0$ and $g(n) > 0$ for all $n \in \mathbb{N}$, expression (2.16) allows to affirm that:

$$h(n) < (\varepsilon + L)\, g(n), \quad whenever \; n > K, \tag{2.17}$$

and:

$$h(n) > (L - \varepsilon)\, g(n), \quad whenever \; n > K, \tag{2.18}$$

Assuming $L > 0$, inequality (2.17) is equivalent to say that $h(n) = O(g(n))$ because there are constants $c_1 = \varepsilon + L$ and $n_0 = K$ such that $h(n) \leq c_1 g(n)$, for all $n \geq n_0$.

Besides, if $L > 0$ we can chose $\varepsilon > 0$ such that $(L - \varepsilon) > 0$. Then, expression (2.18) means that $h(n) = \Omega(g(n))$ since there are constants $c_2 = L - \varepsilon > 0$ and $n_0 = K$ such that $h(n) \geq c_2 g(n)$, for all $n \geq n_0$. Consequently, in this case, $h(n) = \Theta(g(n))$

2.10 Exercises

1. Proof Theorem 1.

2. Proof that $\emptyset \subset A$, $\forall A$.

3. Demonstrate the following properties:

   a) $A \cup B = B \cup A$.

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

   c) $A \cup B = A \Leftrightarrow B \subset A$.

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

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

   f) If $B \subset E$ then $C_E(C_E B) = B$

   g) $A \subset B \Leftrightarrow C_E B \subset C_E A$

   h) $C_E(A \cup B) = C_E A \cap C_E B$

4. Prove the following properties:

   a) $f(X \cup Y) = f(X) \cup f(Y)$.

   b)$f(X \cap Y) \subset f(X) \cap f(Y)$.

   c)$f^{-1}(X \cup Y) = f^{-1}(X) \cup f^{-1}(Y)$.

   d)$f^{-1}(\emptyset) = \emptyset$.

5. Show that the set $\Phi = \{f : A \to A; \quad f \quad is \quad bijective\}$ is a group respect to the composition operation (see Definition 2).

6. Prove that topological space is separated or Hausdorff if, and only if , the intersection of all closed neighborhoods of any point x is the set $\{x\}$.

7. What happens if we state the axiom e2) in the form:

   $O = \bigcap_{i \in I} O_i$ is such that $O \in \Theta$?

8. What is the usual topology of the $\mathbb{R}^n$?

9. Show that $card(\mathbf{N} \times \mathbf{N}) = card(\mathbf{N})$ and that $card(\mathbf{N}) = card(\mathbf{Z})$, where $\mathbf{Z}$ is the set of integers. Find the explicit form for the corresponding bijection.

10. Show that the cardinality of the real numbers is different from the cardinality of $\mathbf{N}$, $card(\mathbb{R}) \neq card(\mathbf{N})$.

11. Prove that the union of two countable sets is countable. Generalize this result for a union of a finite number of countable sets.

12. Show that, for a finite set A, $card(P(A)) = 2^{card(A)}$.

13. Prove that the set of finite subsets of a countable set is countable.

14. Discuss the statement: If $S \subset \mathbf{N}$ then S is countable. Can you prove it?

15. Is it true that:

$$\bigcup_{j=1}^{\infty} \left(\bigcap_{i=1}^{\infty} A_{ij} \right) = \bigcap_{i=1}^{\infty} \left(\bigcup_{j=1}^{\infty} A_{ij} \right),$$

    for any family $(A_{ij})_{(i,j) \in \mathbf{N} \times \mathbf{N}}$ of sets?

16. Let us consider the family $A_n \subset \mathbb{R}$ of intervals, defined by:

$$A_n = [0, \frac{1}{n}), \quad n \in \mathbf{N}$$

    Using the usual topology in $\mathbb{R}$, show that

$$\bigcap_{n=1}^{\infty} A_n = \{0\}.$$

17. A function $f : E_1 \to E_2$ is a continuous function in the topological space E_1 if it is continuous in every point $p \in E_1$. Show that it is equivalent to say that f is continuous if given any open set $B \subset E_2$ then the set $A = f^{-1}(B)$ is an open set of E_1.

18. Using demonstration by mathematical induction to prove that the following properties are true for all $n \in \mathbb{N}$.

    (a)
    $$1 + 3 + 5 + \ldots + (2n - 1) = n^2,$$

    (b)
    $$2 + 7 + 12 + \ldots + (5n - 3) = \frac{n}{2}(5n - 1),$$

    (c)
    $$\frac{1}{1 \cdot 2 \cdot 3} + \frac{1}{2 \cdot 3 \cdot 4} + \frac{1}{3 \cdot 4 \cdot 5} + \ldots + \frac{1}{n \cdot (n+1) \cdot (n+2)} = \frac{n(n+3)}{4 \cdot (n+1) \cdot (n+2)}$$

19. A widely used structure in computing is the Binary Trees (see [34]). Demonstrate, using the induction principle, the following property: In a binary tree with $n > 0$ nodes, the number of left and right empty sub-trees is $n + 1$.

20. Demonstrate or show that it is false

    (a) n^3 is $O(0.001n^3)$,

    (b) $25n^4 - 2900n^3 + 10^{100}n^2 + 3$ is $O(n^4)$,

    (c) 2^{n+100} is $O(2^n)$,

    (d) $\log_2 n$ is $O(\sqrt{n})$,

    (e) 2^n is $O\left(n^k\right)$, for k big enough.

21. Find the computational complexity of an algorithm to calculate the product of two matrices.

22. Find the computational complexity to solve a linear system by the method of determinants.

23. Prove that $f(n) = O(g(n))$ if and only if $g(n) = \Omega(f(n))$.

24. Prove that $f(n) = \Theta(g(n))$ if and only if $g(n) = \Theta(f(n))$.

25. Suppose that $f(n)$ is a k degree polynomial. Show that $f(n)$ is $O\left(n^l\right)$, for any $l \geq k$.

26. Show that $\log n$ is $O\left(n^k\right)$ for any $k > 0$.

27. Show that $n^k = O\left(n^{\log n}\right)$, for any k, but $n^{\log n}$ never is $O\left(n^k\right)$.

28. Show that c^n is $\Omega\left(n^{\log n}\right)$, for any $c > 1$, but $n^{\log n}$ never is $\Omega(c^n)$.

29. Suppose that $c(n) = O(f(n))$ and $g(n) = O(h(n))$. Show that $c(n) \cdot g(n) = O(f(n) \cdot h(n))$.

Chapter 3

Boolean Expressions, Normal Forms and Satisfiability

3.1 Introduction

In this Chapter we consider basic elements of Boolean expressions. These expressions formalize the basic operations that are combined in modern computers hardware to process digital data. From a theoretical viewpoint, Boolean expressions define a background to study learning machines as computational models. We start with the truth tables and algebra of propositions (sections 3.2 and 3.3). Then, we define normal forms and give a discussion about the Satisfiability problem (sections 3.4 and 3.5).

3.2 Truth Tables and Boolean Expressions

A variable x such that $x \in \{0, 1\}$ (true 1 and false 0) is called a Boolean variable. Given Boolean variables $x, y, ...$, they can be put together to form Boolean expressions through the operators of conjunction $(\wedge)$, disjunction $(\vee)$, negation $(\sim)$, exclusive *or* (XOR), implication $(\Rightarrow)$, and bi–implication $(\Leftrightarrow)$ [35]. These operators are defined by the standard Truth Tables 3.1.

Besides, the complete syntax includes parentheses to solve ambiguities. More-

	$\sim$
0	1
1	0

(a)

$\wedge$	0	1
0	0	0
1	0	1

(b)

$\vee$	0	1
0	0	1
1	1	1

(c)

XOR	0	1
0	0	1
1	1	0

(d)

Table 3.1: (a) Truth table for negation. (b) Truth table for conjunction (AND operator). (c) Truth table for disjunction (OR operator). (d) Truth table for XOR operator.

$\Rightarrow$	0	1
0	1	1
1	0	1

(a)

$\Leftrightarrow$	0	1
0	1	0
1	0	1

(b)

Table 3.2: (a) Truth table for implication. (b) Bi-implication and its truth table.

over, as a common convention it is assumed that the operators bind according to their relative priority. The priorities are, with the highest first: $\sim, \wedge, \vee, \Leftrightarrow, \Rightarrow$. So, for example,

$$\sim x_1 \wedge x_2 \vee x_3 \Rightarrow x_4 = (((\sim x_1) \wedge x_2) \vee x_3) \Rightarrow x_4. \tag{3.1}$$

A Boolean expression with variables $x_1, x_2, ..., x_n$ produces for each assignment of truth values ($0, 1$ values) to the variables itself a truth value according to the standard Truth Tables given in (3.1)-(3.2). If we denote the set of truth values by $B = \{0, 1\}$, then we can think of a Boolean expression with variables $x_1, x_2, ..., x_n$ as a function:

$$F : B^n \to B,$$

where $B^n = B \times B \times B ... \times B$ (n-times). As an example, let us rewrite expression (3.1) as:

$$F : B^4 \to B,$$

$$F(x_1, x_2, x_3, x_4) = \sim x_1 \wedge x_2 \vee x_3 \Rightarrow x_4. \tag{3.2}$$

Two Boolean expression $F_1, F_2 : B^n \to B$ are said to be equal if:

$$F_1(x_1, x_2, ..., x_n) = F_2(x_1, x_2, ..., x_n), \quad \forall (x_1, x_2, ..., x_n) \in B^n. \tag{3.3}$$

Tautology: A Boolean expression $F : B^n \to B$ is a tautology if $F(x_1, x_2, ..., x_n) = 1$ for all $(x_1, x_2, ..., x_n) \in B^n$.

Satisfiable: A Boolean expression $F : B^n \to B$ is satisfiable if there is at least one $(x_1, x_2, ..., x_n) \in B^n$ such that $F(x_1, x_2, ..., x_n) = 1$.

3.3 Algebra of Propositions

There are some algebraic laws that Boolean expressions obey, some of which are analogous to laws satisfied by the real numbers. These relationships can be proved by using the Truth Tables (3.1)-(3.2). The corresponding algebra is very much useful to simplify Boolean expressions. Specifically, we have the following laws [29, 33]:

1) **Idempotent laws**

$$p \wedge p = p$$

$$p \vee p = p$$

2) **Commutative**

$$p \wedge q = q \wedge p$$

$$p \vee q = q \vee p$$

$$p \Leftrightarrow q = q \Leftrightarrow p$$

3) **Associative**

$$(p \wedge q) \wedge r = p \wedge (q \wedge r)$$

$$(p \vee q) \vee r = p \vee (q \vee r)$$

4) **Absorption laws**

$$p \wedge (p \vee q) = p$$

$$p \vee (p \wedge q) = p$$

5) **Distributive**

$$p \wedge (q \vee r) = (p \wedge q) \vee (p \wedge r)$$

$$p \vee (q \wedge r) = (p \vee q) \wedge (p \vee r)$$

6) **Involution law**

$$\sim (\sim p) = p$$

7) **De Morgan 's Laws**

$$\sim (p \vee q) = (\sim p) \wedge (\sim q)$$

$$\sim (p \wedge q) = (\sim p) \vee (\sim q)$$

8) **Complement Laws**

$$p \vee \sim p = 1,$$

$$p \wedge \sim p = 0,$$

3.4 Normal Forms

A Boolean expression is in Disjunctive Normal Form (**DNF**) if it consists of a disjunction of conjunctions of variables and negations of variables; that is [35, 33]:

$$F(x_1, x_2, ..., x_n) = \left(t_1^1 \wedge t_2^1 \wedge ... \wedge t_{k_1}^1\right) \vee ... \vee \left(t_1^m \wedge t_2^m \wedge ... \wedge t_{k_m}^m\right), \quad (3.4)$$

where t_j^i is either a variable x_l or a negation of a variable $\sim x_l$.

Expression (3.4) can be rewritten as:

$$\bigvee_{j=1}^{m} \left(\bigwedge_{i=1}^{k_j} t_i^j \right). \quad (3.5)$$

As an example, consider the expression:

$$(x \wedge \sim y) \vee (\sim x \wedge y), \quad (3.6)$$

which is equal to the XOR gate (Exercise).

Similarly, a Conjunctive Normal Form (**CNF**) is an expression that can be written as:

$$\bigwedge_{j=1}^{m} \left(\bigvee_{i=1}^{k_j} t_i^j \right), \quad (3.7)$$

where, like in the previous definition, t_j^i is either a variable x_l or a negation of a variable $\sim x_l$.

Theorem 2 *Any Boolean expression is equal to an expression in CNF and an expression in DNF. (Exercise. See also [33])*

3.5 Satisfiability of Boolean expressions

From the viewpoint of applications, an important point is the satisfiability problem. In general, it is hard to determine whether a Boolean expression is satisfiable; that is, given a Boolean expression $F : B^n \to B$, find the set:

$$S = \{(x_1, x_2, ..., x_n) \in B^n; \quad F(x_1, x_2, ..., x_n) = 1\}.$$

There is a famous theorem, due to Cook, which made the hardness of the satisfiability problem precisely:

Theorem 3 *Satisfiability of Boolean expressions is NP-complete.*

The NP-complete complexity class is a special subset of the NP problems [33, 31, 32]. In a pragmatic viewpoint, we can say that a NP-Complete problem is the one for which the only known deterministic algorithms to solve it run in exponential time. No polynomial time **deterministic** algorithms are known for any of the NP-Complete problems yet.

It is important to observe that, if an expression is in DNF form, then the satisfiability is decidable in polynomial time but for DNFs the tautology check is hard. Besides, the conversion between CNFs and DNFs is exponential. To exemplify this fact, let us consider the following CNF example over the variables $x_0^1, x_0^2, ..., x_0^n, x_1^1, x_1^2, ..., x_1^n$:

$$\left(x_0^1 \vee x_1^1\right) \wedge \left(x_0^2 \vee x_1^2\right) \wedge ... \wedge \left(x_0^n \vee x_1^n\right). \tag{3.8}$$

Following the rules of section 3.3, it is easy to show that this expression can be put in the following DNF form (Exercise):

$$\begin{aligned} &\left(x_0^1 \wedge x_0^2 \wedge ... \wedge x_0^{n-1} \wedge x_0^n\right) \vee \qquad (3.9)\\ &\left(x_0^1 \wedge x_0^2 \wedge ... \wedge x_0^{n-1} \wedge x_1^n\right) \vee \\ &.... \\ &\left(x_1^1 \wedge x_1^2 \wedge ... \wedge x_1^{n-1} \wedge x_0^n\right) \vee \\ &\left(x_1^1 \wedge x_1^2 \wedge ... \wedge x_1^{n-1} \wedge x_1^n\right) \qquad (3.10) \end{aligned}$$

We shall observe that, whereas expression (3.8) has size proportional to n, the corresponding DNF expression has size proportional to $n2^n$. Due to the practical

and theoretical relevance of the satisfiability problem, other normal forms were proposed in order to address its hardness.

3.6 Exercises

1. Show how all operators of Tables (3.1)-(3.2) can be encoded using only $\sim, \wedge$.

2. Prove the De Morgan's and distributive laws.

3. For each of the following formulas tell whether it is (i) satisfiable, (ii) a tautology, (iii) unsatisfiable.

   (a) $(p \vee q) \Rightarrow p$

   (b) $p \wedge \sim q$

   (c) $\sim (p \Rightarrow q) \Rightarrow (p \wedge \sim q)$

4. Is it possible to say whether F is satisfiable from the fact that $\sim F$ is a tautology?

5. Prove that, given two Boolean expressions F_1 and F_2, then $F_1 = F_2$ if and only if the expression $F_1 \Leftrightarrow F_2$ is a tautology.

6. Use the laws of section 3.3 to simplify the expression (3.6).

7. Demonstrate Theorem 2 of section 3.4.

8. By using the propositional laws of section 3.3 show that expressions (3.8) can be rewritten in the form (3.9).

9. Find CNF and DNF versions for each one of the following Boolean expression:

   (a) $(p \wedge (q \vee r)) \vee (q \wedge (p \vee r))$

   (b) $\sim p \vee (p \wedge \sim q) \wedge (r \vee (\sim p \wedge q))$

   (c) $p \Rightarrow (q \Leftrightarrow r)$

Chapter 4

Linear Algebra and Matrices

When representing data points in $\mathbb{R}^n$, the study of data transformations (discrete Fourier, Cosine transform, etc.), as well as some aspects of discrete linear systems theory (convolution, for instance), can be presented using concepts in matrix theory and linear operations.

In this Chapter, we review these mathematical elements. We start with basic notions in linear algebra (section 4.1). See [36] for details in this area. Next, in section 4.2, we revise matrix theory. Our presentation is steered by image processing requirements. Images constitute an important type of data considering the large amount of acquisition instruments and applications of image processing techniques (medical imaging, geoscience, remote sensing, etc.). Additional material in convolution theory and Fourier analysis is presented in Appendix A. Finally, we propose some exercises.

4.1 Background in Linear Algebra

Let the N-dimensional Euclidean vector space $\mathbb{R}^N$ (or $\mathbb{C}^N$) composed by n-uplas:

$$\mathbf{v} = \begin{pmatrix} v(0) \\ v(1) \\ . \\ . \\ . \\ v(N-1) \end{pmatrix}. \tag{4.1}$$

and B a basis of $\mathbb{R}^N$ given by N linearly independent vectors:

$$B = \left\{ \mathbf{u}_i \in \mathbb{R}^N; \quad , \quad i = 0, 1, ..., N-1 \right\}. \tag{4.2}$$

Therefore, we know that any vector $\mathbf{v} \in \mathbb{R}^N$ can be written as a linear combination of elements in B, that means:

$$\mathbf{v} = \sum_{i=0}^{N-1} \alpha_i \mathbf{u}_i. \tag{4.3}$$

We call the array composed by the coefficients α_i the representation of $\mathbf{v}$ in the basis B, which we indicate by:

$$[\mathbf{v}]_B = \begin{pmatrix} \alpha(0) \\ \alpha(1) \\ . \\ . \\ . \\ \alpha(N-1) \end{pmatrix}. \tag{4.4}$$

Moreover, we know that such representation is unique. If B is the canonical basis given by:

$$B = \left\{ \begin{pmatrix} 1 \\ 0 \\ . \\ . \\ . \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ . \\ . \\ . \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ . \\ . \\ 0 \end{pmatrix}, \cdots, \begin{pmatrix} 0 \\ 0 \\ . \\ . \\ . \\ 1 \end{pmatrix} \right\}, \tag{4.5}$$

them we have $[\mathbf{v}]_B \equiv \mathbf{v}$. In this case, we will drop the subscript "B" in expression (4.4).

In this context, we can consider linear a operators $T : \mathbb{R}^N \to \mathbb{R}^N$, which are functions that satisfy the property:

$$T(\alpha_1 \mathbf{u}_1 + \alpha_2 \mathbf{u}_2) = \alpha_1 T(\mathbf{u}_1) + \alpha_2 T(\mathbf{u}_2), \tag{4.6}$$

$\alpha_1, \alpha_2 \in \mathbb{R}$ and $\mathbf{u}_1, \mathbf{u}_2 \in \mathbb{R}^N$. We can show that T can be represented by a matrix $\mathbf{A}$, in the sense that, given a vector $\mathbf{v} \in \mathbb{R}^n$ we can write:

$$T(\mathbf{v}) = A \cdot \mathbf{v}. \tag{4.7}$$

We can show that the matrix A is the representation of T in the canonical basis, denoted by $[T] = A$.

4.2 Matrix Theory

In this text, we focus on matrix theory results useful for image processing. A straightforward representation for digital images is a intensity matrix $\mathbf{A}$. So, we review in this chapter some fundamental results in matrix theory that are important for processing digital images.

In what follows, we will focus on traditional matrices where each element is a real number. The corresponding theory is applied for grey-level image processing. The extension for color images is straightforward and is performed by considering multidimensional matrices where each element is a vector $(R, G, B) \in \mathbb{R}^3$.

As usual in matrix theory, we are going to apply the following notations/definitions (see section 2.7 of [37] for details):

1. The transpose: $\mathbf{A}^T$
2. Inverse: $\mathbf{A}^{-1}$
3. Conjugate: $\mathbf{A}^*$
4. Complex Numbers Set: $\mathbf{C}$
5. Determinant of a square matrix $\mathbf{A}$: $\det(\mathbf{A})$
5. A matrix $\mathbf{A}$ is Nonsingular if: $\det(\mathbf{A}) \neq 0$,

Basic references for this chapter are [37],[5],[38].

4.2.1 Matrix Definition

A matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$ is a two-dimensional array $\mathbf{A} = \{a(m,n) \in \mathbb{R}; \quad 1 \leq m \leq M, \quad 1 \leq n \leq N\}$ represented by:

$$\mathbf{A} = \begin{pmatrix} a(1,1) & a(1,2) & \cdots & a(1,N) \\ & & \cdots & \\ & & \cdots & \\ a(M,1) & a(M,2) & \cdots & a(M,N) \end{pmatrix} \tag{4.8}$$

4.2.2 Row and Columns Representation

Given a matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$, we can represent it through a row vector, by performing the operation:

$$\mathbf{x} = (a(1,1), a(1,2), \cdots, a(1,N), a(2,1), a(2,2), \cdots, a(2,N) \cdots, a(M,1), \cdots a(M,N)). \tag{4.9}$$

Analogously, the column representation is obtained by the column by column stacking:

$$\mathbf{y} = (a(1,1), a(2,1), \cdots, a(M,1), a(1,2), a(2,2), \cdots, a(M,2) \cdots, a(1,N), \cdots a(M,N))^T. \tag{4.10}$$

4.2.3 Transposition and Conjugation Rules

It can be proved the following properties:

1. $(\mathbf{A}^*)^T = \left(\mathbf{A}^T\right)^*$
2. $(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T$
3. $(\mathbf{A}^{-1})^T = \left(\mathbf{A}^T\right)^{-1}$
4. $(\mathbf{AB})^* = \mathbf{B}^*\mathbf{A}^*$

4.2.4 Orthogonal, Unitary and Hermitian Matrices

A matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$ is orthogonal if:

$$\mathbf{A}^{-1} = \mathbf{A}^T. \tag{4.11}$$

On the other hand, a matrix $\mathbf{A} \in \mathbf{C}^{N\times N}$ is unitary if:

$$\mathbf{A}^{-1} = \mathbf{A}^{*T}. \tag{4.12}$$

Hermitian matrix $\mathbf{A} \in \mathbf{C}^{N\times N}$ is the one that satisfies:

$$\mathbf{A} = \mathbf{A}^{*T}. \tag{4.13}$$

In the specific case of real matrices $\mathbf{A} \in \mathbb{R}^{N\times N}$, we call Symmetic if it satisfies:

$$\mathbf{A} = \mathbf{A}^T. \tag{4.14}$$

4.2.5 Positive Definiteness

Given a $\mathbf{A} \in \mathbf{C}^{N\times N}$ Hermitian matrix, we say that [38]:

1. $\mathbf{A}$ is positive semidefinite if $\mathbf{x}^T\mathbf{A}\mathbf{x} \geq \mathbf{0}$, $\forall \mathbf{x} \in \mathbb{R}^N$ and $\mathbf{x} \neq \mathbf{0}$,
2. $\mathbf{A}$ is positive definite if $\mathbf{x}^T\mathbf{A}\mathbf{x} > \mathbf{0}$, $\forall \mathbf{x} \in \mathbb{R}^N$ and $\mathbf{x} \neq \mathbf{0}$,
3. $\mathbf{A}$ is negative semidefinite if $\mathbf{x}^T\mathbf{A}\mathbf{x} \leq \mathbf{0}$, $\forall \mathbf{x} \in \mathbb{R}^N$ and $\mathbf{x} \neq \mathbf{0}$,
4. $\mathbf{A}$ is negative definite if $\mathbf{x}^T\mathbf{A}\mathbf{x} < \mathbf{0}$, $\forall \mathbf{x} \in \mathbb{R}^N$ and $\mathbf{x} \neq \mathbf{0}$,

4.2.6 Diagonal Representation

Given a Hermitian matrix $\mathbf{A} \in \mathbf{C}^{N \times N}$, there exists an unitary matrix Φ such that:

$$\Phi^{*T} \mathbf{A} \Phi = \mathbf{\Lambda}, \tag{4.15}$$

where $\mathbf{\Lambda}$ is a diagonal matrix composed by the eigenvalues of $\mathbf{A}$, that means:

$$\mathbf{A}\phi_k = \lambda_k \phi_k, \quad k = 1, 2, \cdots, N, \tag{4.16}$$

where $\{\lambda_k\}, \{\phi_k\}$ are the eigenvalues and eigenvectors, respectively, of $\mathbf{A}$.

4.2.7 Block Matrices and Kronecker Product

A block matrix $\mathbf{A}$ is a two dimensional array of matrices, that means, a matrix whose elements are matrices themselves:

$$\mathbf{A} = \begin{bmatrix} A_{1,1} & . & . & . & A_{1,N} \\ . & & & & . \\ . & & & & . \\ . & & & & . \\ A_{M,1} & & & & A_{M,N} \end{bmatrix}, \tag{4.17}$$

where each element $A_{m,n} \in \mathbb{R}^{K \times L}$.

Given two matrices $\mathbf{A} \in \mathbb{R}^{M_1 \times M_2}$ and $\mathbf{B} \in \mathbb{R}^{N_1 \times N_2}$, then theier Kronecker product is defined by the following block matrix:

$$\mathbf{A} \otimes \mathbf{B} = \{a_{m,n} \cdot \mathbf{B}\} = \begin{bmatrix} a_{1,1} \cdot \mathbf{B} & . & . & . & a_{1,M_2} \cdot \mathbf{B} \\ . & & & & . \\ . & & & & . \\ . & & & & . \\ a_{M_1,1} \cdot \mathbf{B} & & & & a_{M_1,M_2} \cdot \mathbf{B} \end{bmatrix}. \tag{4.18}$$

There are several interesting properties for Kronecker product, which are stated on the Table 2.7, page 30, of the reference [5]. The exercises will remember the most important ones for image processing. For instance, the property

$$(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = (\mathbf{AC} \otimes \mathbf{BD}), \tag{4.19}$$

is useful to demonstrate that the Kronecker product of orthogonal matrices is an orthogonal matrix. Besides, we can establish the relationship between Kronecker product and Tensor product of linear operators in order to simplify the prof of properties, like the one stated on expression (4.19). See the material of "aula4.pdf" for details about this topic as well as the exercises therein.

4.3 Exercises

1. Prove the following properties for Kronecker product:

   (a) $(A + B) \otimes C = A \otimes C + B \otimes C$.

   (b) $(A \otimes B)^T = A^T \otimes B^T$.

   (c) $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$.

2. Given a square matrix $A \in \mathbb{R}^{N \times N}$ we define its trace ($Tr(A)$) as: $Tr(A) = \sum_{i=1}^{N} a_{ii}$. Show that for square real matrices A and B:

   (a) $Tr(A + B) = Tr(A) + Tr(B)$.

   (b) $Tr(AB) = Tr(BA)$.

3. Let a generalized convolution between the arrays $\mathbf{x}(m, n) \in \mathbb{R}^{M \times N}$ and $\mathbf{h}(m, n)$, defined by:

$$\mathbf{y}(m, n) = \sum_{s,w} \mathbf{x}(s, w)\, h(m, n; s, w). \tag{4.20}$$

   Show that, if we represent $\mathbf{x}(m, n)$ and $\mathbf{h}(m, n)$ as column vectors x, H, respectively, like in expression (4.10), we can write expression (4.20) as the product:

$$y = Hx, \tag{4.21}$$

where H is an $N \times N$ block matrix of basic dimension $M \times M$.

4. Find the eigenvalues and eigenvectors (see section 4.2.6) of the symmetric matrix:

$$\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \tag{4.22}$$

5. Show that a symmetric positive defined matrix has eigenvalues $\lambda_k > 0$.

6. Show that, the following subsets of $\mathbb{R}^{N \times N}$ are closed under inversion (see [38]): (a) positive definite matrices; (b) nonsingular symmetric matrices.

7. Show that, for square matrices $\mathbf{A}, \mathbf{U}$ and $\mathbf{B}$, the operation:

$$\mathbf{V} = \mathbf{A}\mathbf{U}\mathbf{B}^T, \tag{4.23}$$

can be represented as:

$$v = (\mathbf{A} \otimes \mathbf{B})\, u, \tag{4.24}$$

where v, u are column representations of $\mathbf{V}$ and $\mathbf{U}$ and $\otimes$ means the Kronecker product defined on section 4.2.7.

8. Given a column vector $\mathbf{u} \in \mathbb{R}^N$ show that the matrix $\mathbf{u} \cdot \mathbf{u}^T$ is positive semidefinite.

Chapter 5

Analytic Geometry and Differential Calculus

In this material we focus on real functions of several variables, partial derivatives and optimization, as well as some concepts in analytical and differential geometry. The goal is to set some elements behind the learning, its geometric interpretation, and the known backpropagation algorithm. In the end of this Chapter there is a list of proposed exercises. Additional material can be found in [39, 40]

5.1 Parametric Representation of Curves and Surfaces

Let us consider the Cartesian coordinate system xyz. A parametric representation of a curve is a mathematical expression like:

$$P(t) = (x(t), y(t), z(t)), \quad t \in [a, b], \tag{5.1}$$

where $a, b \in \mathbb{R}$.

If we have a parameter space with two variables, say $u, v \in \mathbb{R}$, then we can build a parametric representation of a surface with an expression in the form:

$$S(u, v) = (x(u, v), y(u, v), z(u, v)). \tag{5.2}$$

- Example 1: Given two distinct points $\boldsymbol{p}_1, \boldsymbol{p}_2 \in \mathbb{R}^3$ the line passing through these points can be generated through the parametric expression:

$$\boldsymbol{x} = \boldsymbol{p}_1 + t\boldsymbol{a}, \tag{5.3}$$

where the vector $\boldsymbol{a}$ is given by $\boldsymbol{a} = \boldsymbol{p}_2 - \boldsymbol{p}_1$, as shown in Figure 5.1.

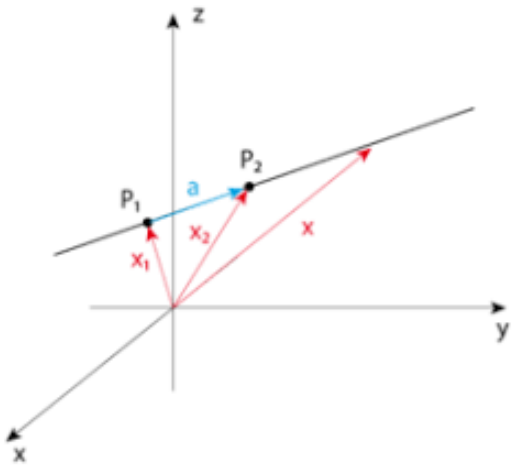


Figure 5.1: Line in $\mathbb{R}^3$ passing through points $\boldsymbol{p}_1, \boldsymbol{p}_2$.

- Example 2: Given three noncolinear points $\boldsymbol{p}_i = (x_i, y_i, z_i)$, $i = 1, 2, 3$, the parametric equation of the plane containing these points can be obtained through the following steps:

  1. Build vectors $\boldsymbol{v}_1 = \boldsymbol{p}_2 - \boldsymbol{p}_1$ and $\boldsymbol{v}_2 = \boldsymbol{p}_3 - \boldsymbol{p}_1$;
  2. Build the parametric representation of the plane as:

$$\boldsymbol{p} = \boldsymbol{p}_1 + \alpha\boldsymbol{v}_1 + \beta\boldsymbol{v}_2, \tag{5.4}$$

  where $\boldsymbol{p} = (x, y, z)^T$ is a generic point and $\alpha, \beta \in \mathbb{R}$ are the parameters of the representation.

- Example 3: The expression:

$$c(\theta) = (a\cos(\theta), b\sin(\theta), \theta), \quad \theta \in [0, 2\pi], \tag{5.5}$$

is a representation of a circular helix. If $a > 0, b > 0$ then the Figure 5.2 shows the shape of this curves.

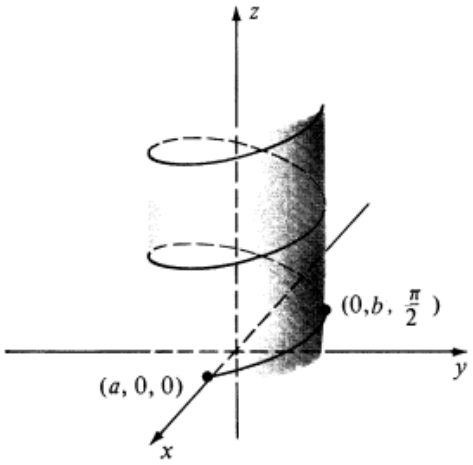


Figure 5.2: Circular helix example, computed through expression (5.5).

5.2 Implicit Representation

A surface in $\mathbb{R}^3$ can be represented through a set $\mathcal{M}$ of points that satisfies some equation:

$$\mathcal{M} = \left\{(x, y, z) \in \mathbb{R}^3; \quad f(x, y, z) = 0\right\}. \tag{5.6}$$

The equation $f(x, y, z) = 0$ is named the implicit representation of the surface S that geometrically represents the set $\mathcal{M}$.

For instance, in the case of the plane $\pi \subset \mathbb{R}^3$ containing the noncolinear points $\boldsymbol{p}_i = (x_i, y_i, z_i)$, $i = 1, 2, 3$, we can build its implicit representation by taking into account that every point $\boldsymbol{p} = (x, y, z)^T$ that belongs to π satisfies:

$$(\boldsymbol{p} - \boldsymbol{p}_1) \cdot \boldsymbol{n} = 0, \tag{5.7}$$

where $\boldsymbol{n}$ is a vector orthogonal to the plane, and '$\cdot$' represents the inner product which, given two vectors $\boldsymbol{a} = (a_1, a_2, a_3)^T$ and $\boldsymbol{b} = (b_1, b_2, b_3)^T$ is computed by:

$$\boldsymbol{a} \cdot \boldsymbol{b} = \sum_{i=1}^{3} a_i b_i = \|\boldsymbol{a}\| \, \|\boldsymbol{b}\| \cos\theta, \tag{5.8}$$

where θ is the smallest angle between the two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, as pictured in Figure 5.3.

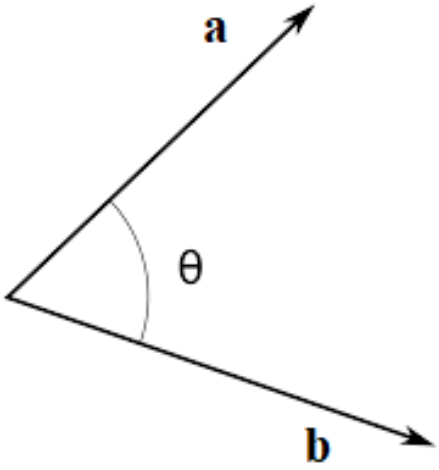


Figure 5.3: Smallest angle between two vectors that appears in equation (5.3).

On the other hand, if we proceed likewise in the section 5.1 and compute the vectors $\boldsymbol{v}_1 = \boldsymbol{p}_2 - \boldsymbol{p}_1$ and $\boldsymbol{v}_2 = \boldsymbol{p}_3 - \boldsymbol{p}_1$ then, the vector $\boldsymbol{n}$ can be calculated by the cross product $\boldsymbol{v}_1 \times \boldsymbol{v}_2$ defined by:

$$\boldsymbol{v}_1 \times \boldsymbol{v}_2 = \begin{vmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ (x_2 - x_1) & (y_2 - y_1) & (z_2 - z_1) \\ (x_3 - x_1) & (y_3 - y_1) & (z_3 - z_1) \end{vmatrix} =$$

$$\begin{vmatrix} (y_2 - y_1) & (z_2 - z_1) \\ (y_3 - y_1) & (z_3 - z_1) \end{vmatrix} \boldsymbol{i} - \begin{vmatrix} (x_2 - x_1) & (z_2 - z_1) \\ (x_3 - x_1) & (z_3 - z_1) \end{vmatrix} \boldsymbol{j} + \begin{vmatrix} (x_2 - x_1) & (y_2 - y_1) \\ (x_3 - x_1) & (y_3 - y_1) \end{vmatrix} \boldsymbol{k}, \tag{5.9}$$

where the unity vectors $\boldsymbol{i} = (1, 0, 0)^T$, $\boldsymbol{j} = (0, 1, 0)^T$, and $\boldsymbol{k} = (0, 0, 1)^T$ form the canonical basis of $\mathbb{R}^3$ (see section 4.1). Consequently, using expressions (5.7) and (5.8) we get:

$$(\boldsymbol{p} - \boldsymbol{p}_1) \cdot (\boldsymbol{v}_1 \times \boldsymbol{v}_2) = 0. \tag{5.10}$$

If $\boldsymbol{v}_1 \times \boldsymbol{v}_2 = (a, b, c)^T$ we can rewrite equation (5.10) as:

$$\left[\begin{pmatrix} x \\ y \\ z \end{pmatrix} - \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix}\right] \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix} = 0,$$

which generates:

$$ax + by + cz + d = 0, \tag{5.11}$$

where $d = -ax_1 - by_1 - cz_1$.

An analogous development can be performed to generate the implicit representation of a line in $\mathbb{R}^2$. Moreover, the generalization of definition (5.6) for $\mathbb{R}^n$ is straightforward. In this case, we say that the obtained hypersurface has dimension $n-1$ and we write $\mathcal{M}^{n-1}$ to represent this fact:

$$\mathcal{M}^{n-1} = \{(x_1, x_2, x_3, \ldots, x_n) \in \mathbb{R}^n; \quad f(x_1, x_2, x_3, \ldots, x_n) = 0\}. \tag{5.12}$$

In general, the dimension of the surface is related to the minimum number of parameters that we need to represent it.

If we consider:

$$f(x_1, x_2, x_3, \ldots, x_n) = \sum_{i=1}^{n} x_i^2 - r^2,$$

then, the equation:

$$f(x_1, x_2, x_3, \ldots, x_n) = 0 \Longleftrightarrow \sum_{i=1}^{n} x_i^2 - r^2 = 0, \tag{5.13}$$

defines a hypersphere in $\mathbb{R}^n$.

Sometimes, we can compute the parametric representation using the implicit one. For instance, in the case of the hypersphere in equation (5.13), it is direct that we can parameterize it using $n-1$ parameters through the expressions:

$$S^+(x_1, x_2, x_3, \ldots, x_{n-1}) = \left(x_1, x_2, x_3, \ldots, x_{n-1}, +\sqrt{r^2 - \sum_{i=1}^{n-1} x_i^2}\right), \tag{5.14}$$

$$S^{-}\left(x_1, x_2, x_3, \ldots, x_{n-1}\right) = \left(x_1, x_2, x_3, \ldots, x_{n-1}, -\sqrt{r^2 - \sum_{i=1}^{n-1} x_i^2}\right). \quad (5.15)$$

5.3 Calculus in One Variable

Given the set of real numbers $\mathbb{R}$ and a continuous function $f : \mathbb{R} \longrightarrow \mathbb{R}$, we say that this function is differentiable in a point $a \in \mathbb{R}$ if the limite:

$$\lim_{\Delta x \to 0} \frac{f\left(a + \Delta x\right) - f\left(a\right)}{\Delta x} \equiv \acute{f}\left(a\right), \quad (5.16)$$

exists and is finite. In this case, $\acute{f}\left(a\right)$ is called the derivative of f respect to the variable x in the point $x = a$. Sometimes, we also use the notation:

$$\frac{df}{dx}\left(a\right) \equiv \acute{f}\left(a\right). \quad (5.17)$$

The geometric interpretation of the $\acute{f}\left(a\right)$ is the following: If $y\left(x\right) = rx + s$ is the line tangent to the graphic of the function f in the point $\left(a, f\left(a\right)\right)$, then $r = \acute{f}\left(a\right)$. Figure 5.4 pictures this fact in the case where $f\left(x\right) = x^2 + 1$. Firstly, we shall compute the limite (5.33) for this polynomial function:

$$\lim_{\Delta x \to 0} \frac{f\left(a + \Delta x\right) - f\left(a\right)}{\Delta x} = \lim_{\Delta x \to 0} \frac{\left[\left(a + \Delta x\right)^2 + 1\right] - \left(a^2 + 1\right)}{\Delta x} = 2a \quad (5.18)$$

Therefore, the angular coefficient of the line is $r = 2a$. The linear coefficient (s) is obtained using the fact that the line and the function f share the point $\left(a, a^2 + 1\right) \in \mathbb{R}^2$. Therefore:

$$a^2 + 1 = 2a \cdot a + s \quad \Longrightarrow \quad s = 1 - a^2. \quad (5.19)$$

The above development can be applied for more general functions. For instance, for a polynomial function $f\left(x\right) = x^n$, $n \in \mathbb{N}^*$, we can use the definition (5.33) to demonstrate that:

$$\acute{f}\left(x\right) = nx^{n-1}, \quad (5.20)$$

and, consequently, we can compute the tangent line everywere in the graph of f. Others formulas to compute derivatives of functions $f : \mathbb{R} \longrightarrow \mathbb{R}$ are given in [39, 40].

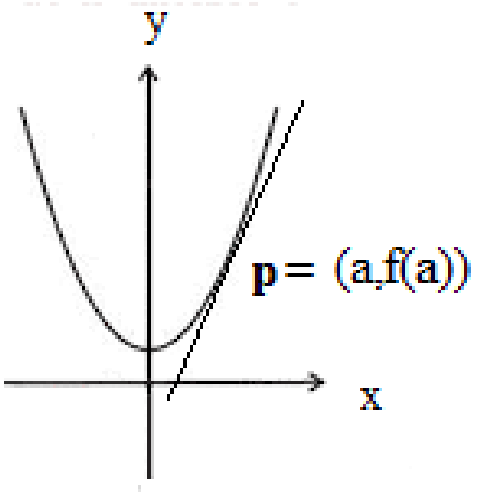


Figure 5.4: Tangent line and derivative.

5.4 Several Variables and Partial Derivatives

Now, we consider functions with several independent variables, that means $f : \mathbb{R}^n \longrightarrow \mathbb{R}$. For instance, let us take the function:

$$f : \mathbb{R}^2 \longrightarrow \mathbb{R}; \quad f(x, y) = x^2 + y^2, \tag{5.21}$$

whose graphical representation is a paraboloid embedded in the $\mathbb{R}^3$, as shown in Figure 5.5. This figure also pictures the trace of the surface in the yz-plane ($z = y^2$) and the trace in the xz-plane, given by equation $z = x^2$.

In this case, the concept of derivative in a point $(a, b) \in \mathbb{R}^2$ must take into account that we may have derivative respect to the variable 'x' and derivative respect to the variable 'y', formally defined by:

$$\frac{\partial f}{\partial x}(a, b) = \lim_{\Delta x \to 0} \frac{f(a + \Delta x, b) - f(a, b)}{\Delta x}, \tag{5.22}$$

$$\frac{\partial f}{\partial y}(a, b) = \lim_{\Delta y \to 0} \frac{f(a, b + \Delta y) - f(a, b)}{\Delta y}, \tag{5.23}$$

respectively. For the function f in expression (5.21) they can be computed as

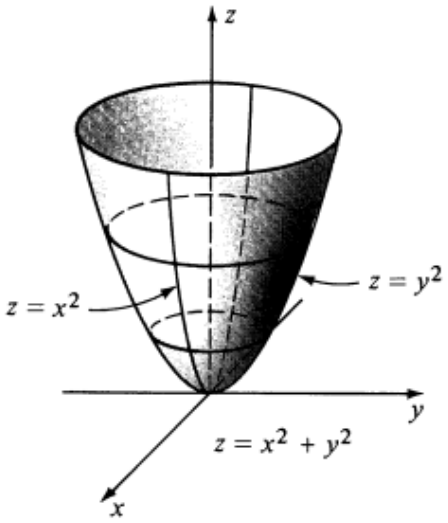


Figure 5.5: Paraboloid and its traces in the yz-plane and xz-planes.

follows:

$$\frac{\partial f}{\partial x}(a,b) = \lim_{\Delta x \to 0} \frac{(a+\Delta x)^2 + b^2 - (a^2+b^2)}{\Delta x} = \lim_{\Delta x \to 0} \frac{2a\Delta x + (\Delta x)^2}{\Delta x} = 2a, \tag{5.24}$$

with an analogous development for expression (5.23), that allows to obtain:

$$\frac{\partial f}{\partial y}(a,b) = 2b. \tag{5.25}$$

The function defined by expressions (5.22) and (5.23) are named partial derivatives of f with respect to x and y, respectively, also denoted by f_x and f_y. From expressions (5.22)-(5.23) we shall notice that, when deriving f with respect to variable x we must consider the other variable as a constant during the derivation process (analogous for the derivative with respect to variable y). It is an useful rule that allows to apply the formulas for derivatives of single valued functions to obtain partial derivatives of functions with several variables.

In the general case of a function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, the partial derivative of f with respect fo variable x_i in the point $(x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ is defined by:

$$\frac{\partial f}{\partial x_i}(x_1, x_2, \ldots, x_n) =$$

$$= \lim_{\Delta x_i \to 0} \frac{f(x_1, x_2, \ldots, x_i + \Delta x_i, \ldots, x_n) - f(x_1, x_2, \ldots, x_n)}{\Delta x_i}, \; i = 1, 2, \ldots, n. \tag{5.26}$$

If the functions $\partial f / \partial x_i$ in expression (5.26) are well defined, we say that f is a differentiable function. Moreover, we can go ahead with the concept of partial derivatives and define expressions:

$$\frac{\partial f^2}{\partial x_i \partial x_j}(x_1, x_2, \ldots, x_n), \quad i, j = 1, 2, \ldots, n, \tag{5.27}$$

$$\frac{\partial f^2}{\partial x_i \partial x_j \partial x_k}(x_1, x_2, \ldots, x_n), \quad i, j, k = 1, 2, \ldots, n, \tag{5.28}$$

$$etc.$$

If functions defined by expression (5.26) are continuous, we say that $f \in \mathcal{C}^1(\mathbb{R}^n)$. If the same happens for equations (5.27) we say that $f \in \mathcal{C}^2(\mathbb{R}^n)$, and so on.

5.5 Composite Functions and Chain Rule

We consider a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$, denoted by:

$$w = f(x_1, x_2, \ldots, x_n). \tag{5.29}$$

Given another coordinate system $z_j = z_j(x_1, x_2, \ldots, x_n)$ with inverse $x_i = x_i(z_1, \cdots, z_n)$, we can rewrite w as:

$$w = f(x_1(z_1, \cdots, z_n), \cdots, x_n(z_1, \cdots, z_n)) \equiv g(z_1, \cdots, z_n)$$

Chain Rule: We can prove that [40]:

$$\frac{\partial f}{\partial z_i} = \sum_{j=1}^{n} \frac{\partial f}{\partial x_j} \frac{\partial x_j}{\partial z_i}, \quad i = 1, 2, \cdots, n. \tag{5.30}$$

5.6 Gradient and Directional Derivative

With the concept of partial derivative, we can go ahead and define the gradient of a function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ in the point $\mathbf{p} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ as:

$$\nabla f\left(x_1, x_2, \ldots, x_n\right) =$$

$$\left(\frac{\partial f}{\partial x_1}\left(x_1, x_2, \ldots, x_n\right), \frac{\partial f}{\partial x_2}\left(x_1, x_2, \ldots, x_n\right), \ldots, \frac{\partial f}{\partial x_n}\left(x_1, x_2, \ldots, x_n\right)\right). \tag{5.31}$$

To simplify notation, it is usual to omit the function arguments and just write:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right). \tag{5.32}$$

To interpret the gradient, we shall consider a vector $\boldsymbol{v} \in \mathbb{R}^n$ and a line $\boldsymbol{r} : \mathbb{R} \longrightarrow \mathbb{R}^n$; such that $\boldsymbol{r}\left(t\right) = \boldsymbol{p}_0 + t\boldsymbol{v}$, where $\boldsymbol{p}_0 \in \mathbb{R}^n$ is a fixed point. Then, we can compute the composition $f\left(\boldsymbol{r}\left(t\right)\right) = g\left(t\right)$, where $g : \mathbb{R} \longrightarrow \mathbb{R}^n$ is a differentiable function with just one dependent variable t.

In this context, the directional derivative of the multivariate differentiable function f along the vector $\boldsymbol{v}$, at the point $\boldsymbol{p}_0$, is defined as:

$$\lim_{t \to 0} \frac{g\left(t\right) - g\left(0\right)}{t} = \frac{dg}{dt}\left(0\right). \tag{5.33}$$

We can use the Chain Rule (5.30) to calculate:

$$\frac{dg}{dt}\left(t\right) = \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}\frac{dx_i}{dt}\left(t\right) = \nabla f\left(x_1\left(t\right), x_2\left(t\right), \ldots, x_n\left(t\right)\right) \cdot \boldsymbol{v}, \tag{5.34}$$

with '$\cdot$' representing the scalar product defined in expression (5.8). So, expression (5.33) becomes:

$$\frac{dg}{dt}\left(0\right) = \nabla f\left(\boldsymbol{p}_0\right) \cdot \boldsymbol{v}, \tag{5.35}$$

where $\nabla f\left(\boldsymbol{p}_0\right)$ denotes the gradient computed in the point $\boldsymbol{p}_0$. In general, we rather prefer the notation:

$$\frac{\partial f}{\partial \boldsymbol{v}}\left(\boldsymbol{p}_0\right) = \nabla f\left(\boldsymbol{p}_0\right) \cdot \boldsymbol{v}. \tag{5.36}$$

So, expression (5.36) shows that the directional derivative $\frac{\partial f}{\partial \boldsymbol{v}}\left(\boldsymbol{p}_0\right)$ is maximum, in absolute value, if ∇f is parallel to $\boldsymbol{v}$ in $\boldsymbol{p}_0$. If we take $\boldsymbol{u} \in \mathbb{R}^n$ such that

$\|\boldsymbol{u}\| = 1$ then:

$$\frac{\partial f}{\partial \boldsymbol{u}}(\boldsymbol{p}_0) = \|\nabla f(\boldsymbol{p}_0)\| \, \|\boldsymbol{u}\| \cos\theta = \|\nabla f(\boldsymbol{p}_0)\| \cos\theta, \tag{5.37}$$

where θ is the smallest angle between the two vectors $\nabla f(\boldsymbol{p}_0)$ and $\boldsymbol{u}$, as represented in Figure 5.3. From expression (5.37) we can notice that, if $\boldsymbol{u}$ is parallel to $\nabla f(\boldsymbol{p}_0)$ then $\nabla f(\boldsymbol{p}_0)$ gives the direction of greatest increase of the function f at the point $(\boldsymbol{p}_0)$.

5.7 Regular Surfaces and Differentiable Manifolds

Let $D \subset \mathbb{R}^2$ and a surface $\mathcal{M} \subset \mathbb{R}^3$, represented by the function S in expression (5.2). In this case $S : D \subset \mathbb{R}^2 \longrightarrow \mathbb{R}^3$ and $\mathcal{M}$ is named a regular surface if S is a differentiable function; that means, if $\partial S/\partial u$ and $\partial S/\partial v$ are continuous functions in D. According to section 5.4, we can write $S \in \mathcal{C}^1(D)$.

These concepts can be generalized to hypersurfaces immersed in $\mathbb{R}^n$. In this case, the hypersurface $\mathcal{M}^{n-1}$ is represented through a function $S : D \subset \mathbb{R}^{n-1} \longrightarrow \mathbb{R}^n$ that also satisfies $S \in \mathcal{C}^1(D)$. However, such definition has a limitation because it needs a global parametrization S of the hypersurface $\mathcal{M}^{n-1}$. For example, it is a known fact in differential geometry that the two-dimensional sphere does not admit a global parametrization. To surpass this issue, we must think about local parametrizations of the surface, as described next.

A differentiable manifold of dimension d is a set, denoted in Figure 5.6 as $\mathcal{M}^d$ and a family of one-to-one functions $\{\varphi_\alpha\}_{\alpha \in I}$, with I an index set, $\varphi_\alpha : U_\alpha \subset \mathbb{R}^d \rightarrow \mathcal{M}^d$ where U_α is an open set of $\mathbb{R}^d$, such that [41]:

1) $\cup_{\alpha \in I} \varphi_\alpha(U_\alpha) = \mathcal{M}^d$.

2) For every $\alpha, \beta \in I$, with $\varphi_\alpha(U_\alpha) \cap \varphi_\beta(U_\beta) = W \neq \emptyset$, the sets $\varphi_\alpha^{-1}(W)$ and $\varphi_\beta^{-1}(W)$ are open sets in $\mathbb{R}^d$ and the chart transition $\varphi_\beta^{-1} \circ \varphi_\alpha : \varphi_\alpha^{-1}(W) \rightarrow \varphi_\beta^{-1}(W)$ are differentiable functions.

3) The family $\{(U_\alpha, \varphi_\alpha)\}$ is maximal respect to properties (1) and (2).

The properties (1) and (2) define the differential structure of $\mathcal{M}^d$. They allow to generate a natural topology over $\mathcal{M}^d$: a set $A \subset \mathcal{M}^d$ is an open set of $\mathcal{M}^d$ if $\varphi_\alpha^{-1}(A \cap \varphi_\alpha(U_\alpha))$ is an open set of $\mathbb{R}^d$, $\forall \alpha$.

Let $\mathbf{p} \in \varphi_\alpha(U_\alpha)$ and $\varphi_\alpha^{-1}(\mathbf{p}) = (x_1(\mathbf{p}), ..., x_n(\mathbf{p}))$. So, $\varphi_\alpha(U_\alpha)$ is called a coordinate neighborhood and the pair $(U_\alpha, \varphi_\alpha)$ a local parametrization or system

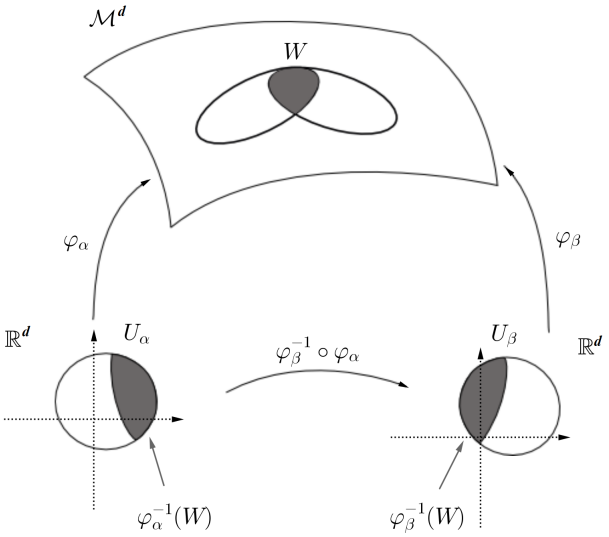


Figure 5.6: Coordinates change and differentiable manifold elements.

of coordinates for $\mathcal{M}^d$ in $\mathbf{p}$. The Figure 5.6 pictures the main elements of items (1)-(3) representing also the change of coordinate systems in the item (2). If $\varphi_\beta^{-1} \circ \varphi_\alpha \in C^k$, with $k \geq 1$, then we say that $\mathcal{M}^d$ is a C^k-differentiable manifold, or simply C^k-manifold. If $k = \infty$, $\mathcal{M}^d$ is called a smooth manifold. Besides, we say that $\mathcal{N}$ is a submanifold of $\mathcal{M}^d$ if $\mathcal{N} \subset \mathcal{M}^d$ and $\mathcal{N}$ is also a differentiable manifold.

Let $\mathcal{M}^d$ be a C^k-manifold of dimension d with local coordinates $\varphi : U \subset \mathbb{R}^d \to \mathcal{M}^d$, at a point $\mathbf{p} = \varphi(\mathbf{x})$. A tangent vector $\mathbf{v}$ to $\mathcal{M}^d$ at $\mathbf{p}$ can be expressed in the local coordinates $\mathbf{x} = (x_1, ..., x_n)$ as:

$$\mathbf{v} = \sum_{i=1}^{d} \left(v_i \frac{\partial \varphi}{\partial x_i} \right). \tag{5.38}$$

where the vectors

$$B = \left\{ \frac{\partial \varphi}{\partial x_1}, ..., \frac{\partial \varphi}{\partial x_m} \right\}, \tag{5.39}$$

are defined by the local coordinates.

The set of all tangent vectors to $\mathcal{M}^d$ at $\mathbf{p}$ is called the tangent space to $\mathcal{M}^d$ at $\mathbf{p}$, and is denoted by $T_{\mathbf{p}}\left(\mathcal{M}^d\right)$. The vectors in the set (5.39) determine an natural basis for $T_{\mathbf{p}}\left(\mathcal{M}^d\right)$. The collection of all tangent spaces to $\mathcal{M}^d$ is the tangent

bundle of the differentiable manifold $\mathcal{M}^d$:

$$TM = \bigcup_{\mathbf{p} \in \mathcal{M}^d} T_{\mathbf{p}} \left(\mathcal{M}^d \right). \tag{5.40}$$

A Riemannian manifold is a manifold $\mathcal{M}^d$ equipped with an inner product in each point $\mathbf{p}$ (bilinear, symmetric and positive definite form in the tangent space $T_{\mathbf{p}} \left(\mathcal{M}^d \right)$) that varies smoothly from point to point.

A geodesic in a Riemannian manifold $\mathcal{M}^d$ is a differentiable curve $\alpha : I \subset \mathbb{R} \rightarrow \mathcal{M}^d$ that is the shortest path between any two points $\mathbf{p}_1 = \alpha \left(t_1 \right)$ and $\mathbf{p}_2 = \alpha \left(t_2 \right)$ [42]. With this concept, we can define the geodesic distance between the points $\mathbf{p}_1$ and $\mathbf{p}_2$ as:

$$d_{\mathcal{M}^d} \left(\boldsymbol{p}_1, \boldsymbol{p}_2 \right) = \int_{t_1}^{t_2} \sqrt{\left\langle \frac{d\alpha}{dt}, \frac{d\alpha}{dt} \right\rangle} dt. \tag{5.41}$$

We denote by $\alpha(s, \mathbf{q}, \dfrac{\mathbf{v}}{\|\mathbf{v}\|})$ the geodesic, parameterized by the arc length s, that pass to $\mathbf{q}$ at $s = 0$ with unitary tangent vector $\alpha\prime \left(0 \right) = \dfrac{\mathbf{v}}{\|\mathbf{v}\|}$. Existence and uniqueness for geodesics can be demonstrated in a Riemannian manifold which allows to define the exponential map as follows.

5.8 Optimization of Scalar Fields

Given a scalar field (function) $f : \mathbb{R}^n \rightarrow \mathbb{R}$ we say that a point $\mathbf{q} \in \mathbb{R}^n$ is a critical point of f if $\nabla f \left(\mathbf{q} \right) = \mathbf{0}$; that means:

$$\frac{\partial f}{\partial x_1} \left(\mathbf{q} \right) = \frac{\partial f}{\partial x_2} \left(\mathbf{q} \right) = \ldots = \frac{\partial f}{\partial x_n} \left(\mathbf{q} \right) = 0. \tag{5.42}$$

Also, we can compute the Hessian $H_{\mathbf{x}} f \left(\mathbf{q} \right)$ as:

$$\left[H_{\mathbf{x}} f \left(\mathbf{q} \right) \right]_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j} \left(\mathbf{q} \right), \quad 1 \leq i, j \leq n, \tag{5.43}$$

and the dimension d of the null space of the matrix $\left[H_{\mathbf{x}} f \left(\mathbf{q} \right) \right]$ is called the corank of $H_{\mathbf{x}} f$ at $\mathbf{q} \in \mathbb{R}^n$. The rank of $H_{\mathbf{x}} f$ at $\mathbf{q} \in \mathbb{R}^n$ is given by: $rank \left(H_{\mathbf{x}} f \mathbf{q} \right) = n - d$.

We say that f has a nondegenerate critical point at $\mathbf{q} \in \mathbb{R}^n$ if $\nabla f \left(\mathbf{q} \right) = \mathbf{0}$ and if $H_{\mathbf{x}} f \left(\mathbf{q} \right)$, defined by expression (8.20), has rank equal to n (non-singular

matrix). On the other hand, if $H_{\mathbf{x}}f(\mathbf{q})$ is degenerate we say the point $\mathbf{q} \in \mathbb{R}^n$ that satisfies $\nabla f(\mathbf{q}) = \mathbf{0}$ is a degenerate critical point (corank $d > 0$).

From the Taylor Series around a critical point $\mathbf{q} \in \mathbb{R}^n$:

$$f(\boldsymbol{x}) = f(\mathbf{q}) + \frac{1}{2!}(\boldsymbol{x} - \mathbf{q})^T H_{\mathbf{x}}f(\mathbf{q})(\boldsymbol{x} - \mathbf{q}) + O\left(||\boldsymbol{x} - \mathbf{q}||^2\right). \tag{5.44}$$

we have tree possibilities:

1. Local minimum: $\nabla f(\mathbf{q}) = \mathbf{0}$ and $H_{\mathbf{x}}f(\mathbf{q})$ positive definite
2. Local maximum: $\nabla f(\mathbf{q}) = \mathbf{0}$ and $H_{\mathbf{x}}f(\mathbf{q})$ negative definite
3. Seddle point: $\nabla f(\mathbf{q}) = \mathbf{0}$ and $H_{\mathbf{x}}f(\mathbf{q})$ non-singular but with positive and negative eigenvalues
4. Degenerate point: $\nabla f(\mathbf{q}) = \mathbf{0}$ and $H_{\mathbf{x}}f(\mathbf{q})$ singular

Optimization theory for differentiable functions deals with the problem of efficiently finding the critical points and to classify them according to the cases (1)-(4) above. The situation is simplified if we have a differentiable scalar field $f : D \subset \mathbb{R}^n \to \mathbb{R}$ that is also a convex function; that means $H_{\mathbf{x}}f(\mathbf{x})$ is positive definite in the domain D.

The most usual algorithm in this area is the steepest descent one [43], which is based in the fact that the gradient defines the direction of greatest increase of the function f, as discussed in section 5.6. Consequently, the vector '$-\nabla f(\boldsymbol{x})$' points in the direction of greatest decrease of the function f in the point $\boldsymbol{x} \in D$. Hence, starting from a point $\boldsymbol{p} \in D$ we can iteratively approximate the critical point through the procedure in the Algorithm 1.

5.9 Optimization with Constraints

In machine learning application sometimes we need to optimize a function $w = f(x_1, x_2, \ldots, x_n)$, that means, to find the minimum and maximum values of the function, subject to some constraints in the form $g_i(x_1, x_2, \ldots, x_n) = k_i$, $i = 1, 2, \ldots N$. Hence, we consider the general problem:

$$\min\ w = f(x_1, x_2, \ldots, x_n) \tag{5.45}$$

Algorithmus 1 Steepest Descent Algorithm

Input: $\delta, \varepsilon > 0$,

Initialization: $\mathbf{x}_0 \in D$, $k = 0$;

$\boldsymbol{p} \leftarrow \mathbf{x}_0$;,

while $(\|\nabla f(\boldsymbol{p})\| < \varepsilon)$ **do**

$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \delta \nabla f(\boldsymbol{p})$,

$\boldsymbol{p} \leftarrow \mathbf{x}_{k+1}$,

$k \leftarrow k + 1$,

end while

$\boldsymbol{x}^{\star}_{opt} \leftarrow \boldsymbol{p}$,

Output: Solution $\boldsymbol{x}^{\star}_{opt}$

subject to:

$$g_j(x_1, x_2, \ldots, x_n) = k_j, \ j = 1, 2, \ldots m. \tag{5.46}$$

The solution of problem (5.45)-(5.46) is obtained through the next steps:

1. Build the Lagrangian function [39, 40]:

$$L(x_1, x_2, \ldots, x_n, \lambda_1, \lambda_2, \ldots, \lambda_m) =$$

$$= f(x_1, x_2, \ldots, x_n) - \lambda_j \left(g_j(x_1, x_2, \ldots, x_n) - k_j\right), \tag{5.47}$$

where the parameters λ_j are called the Lagrange multipliers.

2. Solve $\nabla L = (0, 0, \ldots, 0)$; that means, solve expressions:

$$\frac{\partial L}{\partial x_i}(x_1, x_2, \ldots, x_n, \lambda_1, \lambda_2, \ldots, \lambda_m) = 0, \quad i = 1, 2, \ldots, n, \tag{5.48}$$

$$\frac{\partial L}{\partial \lambda_j}(x_1, x_2, \ldots, x_n, \lambda_1, \lambda_2, \ldots, \lambda_m) = 0, \quad j = 1, 2 \ldots, m. \tag{5.49}$$

3. Take the solutions $\left(\overline{\boldsymbol{x}}, \overline{\boldsymbol{\lambda}}\right)$ of system (5.48)-(5.49) and set $\boldsymbol{\lambda} = \overline{\boldsymbol{\lambda}}$ into expression (5.47) to obtain the function $F(\boldsymbol{x}) \equiv L\left(\boldsymbol{x}; \overline{\boldsymbol{\lambda}}.\right)$

4. Calculte the Hessian $H_{\mathbf{x}}F(\overline{\boldsymbol{x}})$ given by:

$$[H_{\mathbf{x}}F(\overline{\boldsymbol{x}})]_{ij} = \frac{\partial^2 F}{\partial x_i \partial x_j}(\overline{\boldsymbol{x}}), \quad 1 \leq i, j \leq N, \tag{5.50}$$

5. Compute the eigenvalues of the Hessian $H_{\mathbf{x}}F(\overline{\boldsymbol{x}})$ and use them to classify $\overline{\boldsymbol{x}}$ according to the cases (1)-(4) of section 5.8.

5.10 Exercises

1. Consider the paraboloid $z = x^2 + y^2$ in $\mathbb{R}^3$. Show that it is a differentiable manifold of dimension 2.

2. Show that a sphere in $\mathbb{R}^n$ is a differentiable manifold of dimension $n-1$.

3. Compute an equation for the plane in $\mathbb{R}^3$ that contains the points $\boldsymbol{p}_1 = (1,0,0)$, $\boldsymbol{p}_2 = (0,1,1)$, $\boldsymbol{p}_3 = (-2,0,3)$.

4. Find equations for the tangent plane and the normal line to the graph of the given equations at the indicated point.

   (a) $4x^2 - y^3 + 3z^2 = 10$, point $\boldsymbol{p} = (2,-3,1)$

   (b) $z = 2\exp(-x)\cos(y)$,point $\boldsymbol{p} = (0,\pi/3,1)$

5. Demonstrate that the tangent plane to the given quadric surface at the point $\boldsymbol{p}_0 = (x_0, y_0, z_0)$ may be written in the indicated form:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1, \quad \frac{xx_0}{a^2} + \frac{yy_0}{b^2} + \frac{zz_0}{c^2} = 1.$$

6. Find the points on the hyperboloid $x^2 - 2y^2 - 4z^2 = 16$ at which the tangent plane is parallel to the plane $4x - 2y - 4z = 5$.

7. Prove that every normal line to a sphere passes through the center of the sphere. Given the sphere S^2, implicitly defined by $x^2 + y^2 + z^2 = 1$,

compute its parametric representation in Cartesian coordinates. Using this representation, find the unity normal vector field to S^2.

8. Two surfaces are said to be orthogonal at a point of intersection $\boldsymbol{p} = (x, y, z)$ if their normal lines at $\boldsymbol{p}$ are orthogonal. Show that the graphs of $F(x, y, z) = 0$ and $G(x, y, z) = 0$ (where F and G have partial derivatives) are orthogonal at $\boldsymbol{p}$ if and only if:

$$F_x G_x + F_y G_y + F_z G_z = 0$$

9. Consider the functions bellow and compute their critical points. Classify the critical points according to: local minimum, local maximum, seddle point, degenerate point.

   (a) $f(x, y) = x^2 + 2xy + 3y^2$

   (b) $f(x, y) = x^2 - y^2$

   (c) $f(x, y) = \cos(x) + \cos(y)$

   (d) $f(x, y) = \exp(x) \sin(y)$

10. Use Lagrange multipliers to find the local extrema of the given function f subject to the stated constraints:

    (a) $f(x, y) = y^2 - 4xy + 4x^2; \quad x^2 + y^2 = 1,$

    (b) $f(x, y) = 2x^2 + xy - y^2 + y; \quad 2x + 3y = 1.$

11. Use the Chain Rule to find $\partial w/\partial x$ and $\partial w/\partial y$:

    (a) $w = u \sin(v)$, $u = x^2 + y^2$, $v = xy$.

    (b) $w = uv + v^2$,$u = x \sin(y)$, $v = y \sin(x)$.

12. Use the Chain Rule to find $\partial r/\partial u$, $\partial r/\partial v$, and $\partial r/\partial t$:

    (a) $r = x \ln(y)$, $x = 3u + vt$, $y = uvt$.

    (b) $r = vw^2 \cos(z)$, $w = u^2 vt$, $z = ut^2$.

13. Consider the linear system $Ax = b$, where $A \in \mathbb{R}^{n \times m}$, $b \in \mathbb{R}^m$ and $n > m$. Use the optimization theory of section 5.8 to solve the problem:

$$\arg \min_{x \in \mathbb{R}^m} \|Ax - b\|_2^2 . \tag{5.51}$$

14. Given a set of points in $\mathbb{R}^2$, build and solve a least squares problem to calculate the line that best fits the data. Can you generalize your approach to $\mathbb{R}^m$?

15. Consider linear combinations of fixed nonlinear functions $\phi_j : \mathbb{R}^m \longrightarrow \mathbb{R}$ of the input vector $\boldsymbol{x} \in \mathbb{R}^m$, of the form:

$$y(\boldsymbol{x}) = w_0 + \sum_{j=1}^{M} w_j \phi_j(\boldsymbol{x}) , \tag{5.52}$$

    where ϕ_j are known as basis functions and $w_i \in \mathbb{R}$ are parameters of the model. In this exercise, the basis functions are defined by:

$$\phi_j(\boldsymbol{x}) = \sigma\left(\frac{\left\| \boldsymbol{x} - \boldsymbol{\mu}_j \right\|_2^2}{s_j} \right) , \tag{5.53}$$

    where $\sigma(a)$ is the logistic sigmoid function:

$$\sigma(a) = \frac{1}{1 + \exp(-a)} . \tag{5.54}$$

    Given a set composed by the points $(y_i, \boldsymbol{x}_i) \in \mathbb{R}^{1+m}$, $i = 1, 2, ..., N$ generalize the least square criterion to perform nonlinear regression with expression (5.52) through the error function:

$$E(\boldsymbol{w}) = \frac{1}{N} \sum_{n=1}^{N} \left(y_n - \boldsymbol{w}^T \begin{pmatrix} \phi_0(\boldsymbol{x}_n) \\ \phi_1(\boldsymbol{x}_n) \\ \vdots \\ \phi_M(\boldsymbol{x}_n) \end{pmatrix} \right)^2 ,$$

where $\phi_0(\boldsymbol{x}) = 1$.

16. Take a discretization of the function $y = x^2$ in the interval $[2, 3]$ to obtain a set $D = \{(x_i, y_i) \in \mathbb{R}^2;\ y_i = x_i^2,\ i = 1, 2, \ldots, n\}$. Compute the principal components $\boldsymbol{p}_1, \boldsymbol{p}_2$ for the set D. Show that $\boldsymbol{p}_1$ is almost tangent and $\boldsymbol{p}_2$ is almost orthonormal to the corresponding curve in $\mathbb{R}^2$.

17. Regularized least squares. Consider the error function:

$$E(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N}\left(y_n - \boldsymbol{w}^T\begin{pmatrix}\phi_1(\boldsymbol{x}_n)\\ \phi_2(\boldsymbol{x}_n)\\ \vdots \\ \phi_M(\boldsymbol{x}_n)\end{pmatrix}\right)^2 + \lambda \boldsymbol{w}^T\boldsymbol{w}. \tag{5.55}$$

Use the optimization theory of section 5.8 to solve the problem:

$$\arg\min_{\boldsymbol{w}\in\mathbb{R}^M} E(\boldsymbol{w}).$$

18. Multiple outputs: If we have a set composed by the points $(\boldsymbol{y}_i, \boldsymbol{x}_i) \in \mathbb{R}^{n+m}$, $i = 1, 2, ..., N$ we can generalize the model (5.52) through the expression:

$$\boldsymbol{y}(\boldsymbol{x}) = W\begin{pmatrix}\phi_0(\boldsymbol{x})\\ \phi_1(\boldsymbol{x})\\ \vdots \\ \phi_M(\boldsymbol{x})\end{pmatrix}, \tag{5.56}$$

where $\phi_0(\boldsymbol{x}) = 1$ and $W \in \mathbb{R}^{n\times m}$. Define an error function and minimize it with respect to W.

19. Take a clound of $N = 20$ points around the origem in $\mathbb{R}^2$. Mark half of them with label '1' and half with label '-1' to generate a set:

$$D = \left\{(x_{1i}, x_{2i}, l_i) \in \mathbb{R}^2 \times \{-1, 1\},\quad i = 1, 2, \ldots, N = 20\right\}. \tag{5.57}$$

Consider the expression:

$$z(x_j) = Tanh\left(w_0 + \sum_{i=1}^{2} w_i x_{ij}\right), \tag{5.58}$$

where $Tanh$ is the hyperbolic tangent function (https://en.wikipedia.org/wiki/Hyperbolic_functions) defined as:

$$Tanh\left(x\right)=\frac{\exp\left(x\right)-\exp\left(-x\right)}{\exp\left(x\right)+\exp\left(-x\right)}, \tag{5.59}$$

Study the **sequential gradient descent** described in page 144 of reference [44] and design an algorithm to compute a separating line for D; that means, to seek for a line $cy=ax+b$ such that:

$$cy_i-\left(ax_i+b\right)>0,\ if\quad l_i=1$$

$$cy_i-\left(ax_i+b\right)<0,\ if\quad l_i=-1.$$

Solution. Take the error function:

$$E\left(a,b,c\right)=\frac{1}{N}\sum_{n=1}^{N}E_n\left(a,b,c\right), \tag{5.60}$$

where $\boldsymbol{z}=(a,b,c)\in\mathbb{R}^3$ is the parameter vector and:

$$E_n\left(\boldsymbol{z}\right)=\left[Tanh\left(cy_i-\left(ax_i+b\right)\right)-l_i\right]^2. \tag{5.61}$$

Optimize function (5.60) by implementing the sequential gradient descent scheme of Algorithm 2.

20. Take a discretization with $N=20$ points of the sphere $S^2=\left\{\boldsymbol{x}\in\mathbb{R}^3;\left\|\boldsymbol{x}\right\|_2^2=1\right\}$ nearby the point $\boldsymbol{p}=\left(1/\sqrt{3},1/\sqrt{3},1/\sqrt{3}\right)$ to get a set $D=\left\{\boldsymbol{x}_i,i=1,2,\ldots N=20\right\}\in S^2$. Apply the theory of exercise 5 to approximate the surface S^2 nearby the point $\boldsymbol{p}$ through a function $y:\mathbb{R}^2\longrightarrow\mathbb{R}$ of the form (5.52). Compute the tangent space $T_p\left(S^2\right)$.

Algorithmus 2 Sequential Gradient Descent Algorithm

Input: $\delta, \varepsilon > 0$,

Initialization: $\mathbf{z}_0 \in \mathbb{R}^3$, $k = 0$;

$\boldsymbol{p} \leftarrow \mathbf{z}_0$;,

while $(\|\nabla E(\boldsymbol{p})\| \geq \varepsilon)$ **do**

for $n = 1, ..., N$ **do**

$\boldsymbol{z}_{k+1} = \boldsymbol{p} - \delta \nabla E_n(\boldsymbol{p})$,

$\boldsymbol{p} \leftarrow \mathbf{z}_{k+1}$,

$k \leftarrow k + 1$,

end for

$k \leftarrow 0$,

end while

$\boldsymbol{z}^{\star}_{opt} \leftarrow \boldsymbol{p}$,

Output: Solution $\boldsymbol{z}^{\star}_{opt}$

Chapter 6

Stochastic Representation of Digital Signals

Stochastic models describe a signal as a member of an ensemble, which can be characterized by its mean and covariance functions. This allows development of algorithms that are useful for an entire class or an ensemble of signals rather than for a single one. Fall in this category the covariance models, the $1-D$ models (signal is considered a vector) and $2-D$ models (signals are represented by matrices) and Gaussian models [5, 37].

In this text, we focus on basic principles on random signals, their representations by stochastic models and some elements of classical information theory.

Before continuing, let us start with some fundamental definitions.

Sample Space E is an exhaustive list of all the possible outcomes of an experiment. Any subset of E is called an **event**.

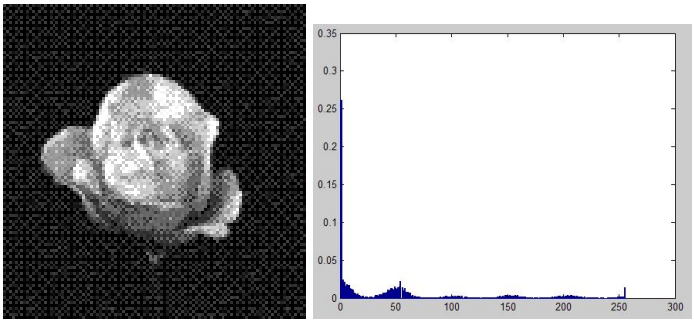
A **random variable** x is a function $x : E \to \mathbb{R}$. For discrete random variable, we have some probability distribution $p(x = i) = p_i$. If x is a continuous random variable, then there is a **probability density function**, also named **pdf**, $f : \mathbb{R} \to \mathbb{R}$, such that, the probability of the event given by $a \leq x \leq b$ is:

$$p\left(a \leq x \leq b\right) = \int_a^b f\left(x\right) dx. \tag{6.1}$$

As a consequence, a *pdf* must satisfies:

$$p\left(x_{\min} \leq x \leq x_{\max}\right) = \int_{x_{\min}}^{x_{\max}} f\left(x\right) dx = 1. \tag{6.2}$$

For instance, in image processing, it is a common practice to interpret the relative frequency of occurrence of the gray levels as probabilities. The **histogram of an image** is the graph obtained by plotting the gray level intensities in the x axis and the relative frequencies in the y axis.



(a) (b)

Figure 6.1: (a) Gray level image. (b) Histogram of the image.

6.1 Random Signals

Let a vector of N random variables:

$$\mathbf{u} = \begin{pmatrix} u\left(0\right) \\ u\left(1\right) \\ . \\ . \\ . \\ u\left(N-1\right) \end{pmatrix} \in \mathbb{R}^N, \tag{6.3}$$

where $u\left(i\right)$ is a real discrete random variable, $i = 0, 1, 2, ..., N-1$.

In this case, we can define its mean vector as [5]:

$$\mathbf{E}(\mathbf{u}) = \mu = \begin{pmatrix} \mu(0) \\ \mu(1) \\ . \\ . \\ . \\ \mu(N-1) \end{pmatrix} \in \mathbb{R}^N. \tag{6.4}$$

If $u(i)$ are random variable taking values in the set $\{\pi_0, \pi_1, ..., \pi_{M-1}\}$, then each mean $\mu(i)$ can be computed by an expression like:

$$\mu(i) = \sum_{j=0}^{M-1} \pi_j p_j^i, \tag{6.5}$$

with p_j^i satisfying:

$$0 \leq p_j^i \leq 1, \tag{6.6}$$

$$\sum_{j=0}^{M-1} p_j^i = 1, \quad i = 0, 1, ..., N-1. \tag{6.7}$$

The weights p_j^i are the probabilities of value π_j *in the position* i of the array. For instance, for digital image applications, each p_j^i will be the probability of intensity π_j in the pixel i. We say that p_j^i is the probability distribution associated to the problem.

Besides, we can define the covariance matrix by [5]:

$$\mathbf{Cov}[\mathbf{u}] = E\left((\mathbf{u} - \mu)(\mathbf{u} - \mu)^T\right) \tag{6.8}$$

We can extend these definitions for a two-dimensional $\{u(m,n)\}$ array of random variables, also called discrete random field. In this case, we have:

$$\mathbf{u} = \begin{pmatrix} u(1,1) & u(1,2) & \cdots & u(1,N) \\ & & \cdots & \\ & & \cdots & \\ u(M,1) & u(M,2) & \cdots & u(M,N) \end{pmatrix},$$

where each $u(m,n)$ is a random variable. Likewise in expression (6.4), we can define the mean functions for the discrete random field $\mathbf{u}$ as**:**

$$E(\mathbf{u}) = \begin{pmatrix} \mu(1,1) & \mu(1,2) & \cdots & \mu(1,N) \\ & & \cdots & \\ & & \cdots & \\ \mu(N,1) & \mu(N,2) & \cdots & \mu(N,N) \end{pmatrix}, \tag{6.9}$$

where $\mu(m,n) = E(u(m,n))$.

Besides, analogously to expression (6.8), we can define the covariance function as:

$$\mathbf{Cov}[\mathbf{u}] = \{Cov(m,n;s,t), \quad 1 \leq m,n,s,t \leq N\}, \tag{6.10}$$

such that:

$$Cov(m,n;s,t) = E((u(m,n) - \mu(m,n)) \cdot (u^*(s,t) - \mu^*(s,t))) \tag{6.11}$$

A considerable simplification occurs for stationary process, that means, the ones for which $\mu(m,n) = cons\tan t$.

6.2 Elements of Probability Theory

Given two events $A, B \subset E$, we can define the (conditional) probability that event A will occur given that event B has occurred by:

$$p(A|B) = \frac{p(A \cap B)}{p(B)}. \tag{6.12}$$

where $p(A|B)$ is the usual notation for conditional probability and "$\cap$" means intersection. Expression (6.12) is called the Baye's Rule and is the starting point for Bayesian approaches. The essence of this theory is to provide a mathematical rule explaining how you should change your existing beliefs in the light of new evidence. An interesting sub-class of random processes in this area is the Markov ones defined as follows.

Simply, a Markov process is the one whose future probabilities are determined by its most recent values. More precisely, A sequence $u(n)$ is called Markov-s or sth-order Markov if the conditional probability of $u(n)$ given the entire past is equal to the conditional probability of $u(n)$ given only the the last s observations:

$$p(u(n)|u(n-1), u(n-2), ...u(1)) = p(u(n)|u(n-1), u(n-2), ...u(n-s)). \tag{6.13}$$

6.2.1 Orthogonality and Independence

Two random variables x and y are called independent if and only if their joint probability density function is a product of their marginal densities:

$$p(x = \alpha, y = \beta) = p(x = \alpha)\, p(y = \beta). \tag{6.14}$$

On the other hand, x and y are said to be orthogonal if:

$$E(xy^*) = 0, \tag{6.15}$$

and are called uncorrelated if:

$$E(xy^*) = E(x) \cdot E(y^*). \tag{6.16}$$

Expression (6.16) implies that (Exercise):

$$E((x - \mu_x)(y - \mu_y)^*) = 0. \tag{6.17}$$

6.2.2 Mean Square Estimate

Let $\{y(n), \quad n = 1, 2, \cdots, N\}$ be a real random sequence that represents N observations of an unknown real random variable x. It is desired to find the optimum mean square estimate $\widehat{x}$ of x, from the observations $y(n)$, such that the mean square error:

$$\sigma^2 = E\left[(x - \widehat{x})^2\right] \tag{6.18}$$

is minimized. It can be shown that the solution is the conditional mean of x given $y(n), \quad n = 1, 2, \cdots, N$:

$$\widehat{x} = E(x|y) = E[x|y(1), \cdots, y(N)] = \int_{-\infty}^{+\infty} \xi \cdot p_{x|y}(\xi)\, d\xi, \tag{6.19}$$

where $p_{x|y}(\xi)$ is the conditional probability density of x given the observation array $\{y(n), \quad n = 1, 2, \cdots, N\}$.

6.3 Concepts in Information Theory

Information theory deals with measurement and transmission of information through a channel. A fundamental work in this area is the Shannon's Information Theory (see [45], Chapter 11), which provides many useful tools that are based on measuring information in terms of the complexity of structures needed to encode a given piece of information.

Shannon's theory solves two central problems for classical information:

(1) How much can a message be compressed; i.e., how redundant is the information? (*The noiseless coding theorem*).

(2) At what rate can we communicate reliably over a noisy channel; i.e., how much redundancy must be incorporated into a message to protect against errors? (*The noisy channel coding theorem*).

In this theory, the information and the transmission channel are formulated in a probabilistic point of view. In what follows, we review the development of Shannon's Theory presented in [45], pp. 537. In this reference, a central definition is an $\epsilon - typical$ sequence.

6.3.1 Classical Information Theory

Firstly, we must discuss Shannon entropy and its relevance to classical information.

A message is a string of n letters chosen from an alphabet of W letters:

$$A = \{a_1, a_2, ..., a_W\}$$

Let us suppose a prior probability distribution p :

$$\begin{aligned} p(a_i) &= p_i, \\ \sum_{i=1}^{W} p(a_i) &= 1 \end{aligned} \tag{6.20}$$

For example, the simplest case is for a binary alphabet where $p(1) = p$ and $p(0) = 1 - p$; $1 \leq p \leq 1$.

For n very large, the law of large numbers tells us that typical strings will contain (in the binary case) about $n(1-p)$ 0's and about np 1's. The number of distinct strings of this form is given by the binomial coefficient $\begin{pmatrix} n \\ np \end{pmatrix}$. From Stirling approximation [46] we know that $\log(n!) = n\log n - n + O(\log n)$. Thus, we approximate the binomial coefficient by (see [46], for more details):

$$\log \begin{pmatrix} n \\ np \end{pmatrix} \simeq nH(p), \tag{6.21}$$

where:

$$H(p) = -p\log p - (1-p)\log(1-p) \tag{6.22}$$

is the entropy function (observe that log's have base 2).

Thus, from equation (6.21) we can see that the number of typical strings is of order $2^{nH(p)}$.

Furthermore, the entropy H has the following properties:

(a) $0 \leq H(p) \leq 1$, if $1 \leq p \leq 1$;

(b) $H(p) = 1$ only if $p = \frac{1}{2}$.

Thus, from property (a) we find that:

$$2^{nH(p)} < 2^n \quad if \quad p \neq \frac{1}{2} \tag{6.23}$$

that is, we do not need a codeword for every n-letter sequence, but only for the typical ones. In another way, we can compress the information in a shorter string.

This can be generalized:

$$\frac{n!}{\prod_x (np(x))!} \simeq 2^{nH(X)} \tag{6.24}$$

$$H(X) = \sum_x p(x)(-\log p(x)) \tag{6.25}$$

where $X : A \to \mathbb{R}$ is a random variable with probability distribution $p(x)$.

Such result points out to a compression scheme. To accomplishes this, we need also to formulate these results more precisely. This is done in the next section.

6.3.2 Compression Problem

Firstly, we must formulate what is a compression scheme. Let us suppose that $X_1, X_2, X_3, ..., X_n$ is a *independent and identically distributed* classical information source over some finite alphabet; that is, the expectations and variances are such that $E(X_1) = E(X_2) = ... = E(X_n) \equiv E(X)$ and $D(X_1) = D(X_2) = ... = D(X_n) \equiv D(X)$, where X represents any of the random variables, and expression (6.26) holds.

A *Compression Scheme of Rate* R, denoted by $C^n(x)$, maps possible sequences $x = (x_1, x_2, ..., x_n)$ to a bit string of length nR. The matching *decompression scheme* D^n takes the nR compressed bits and maps them back to a string of n letters. This operation is denoted by $D^n(C^n(x))$. A compression-decompression scheme is said to be reliable if the probability that $D^n(C^n(x)) = x$ approaches to one as $n \to \infty$.

A fundamental result in this theory is the *Shannon's Noiseless Channel Coding Theorem*:

Suppose that $\{X_i\}$ *are independent and identically distributed random variables that define an information source with entropy* $H(X)$. *Suppose* $R >$

$H(X)$. *Then there exists a reliable compression scheme of rate* R *for the source. Conversely, if* $R < H(X)$ *then any compression scheme will not be reliable.*

In [45], pp. 537, this theorem is demonstrated following the development given bellow.

Let a particular $n-$message: $x_1, x_2, ..., x_n$.

So, by the assumption of statistically independent random variables:

$$P(x_1, x_2, ..., x_n) = p(x_1) \cdot p(x_2) \cdot ... \cdot p(x_n) \tag{6.26}$$

Thus, typically, we expect:

$$P(x_1, x_2, ..., x_n) \approx p^{np} (1-p)^{(1-p)n}. \tag{6.27}$$

So:

$$\frac{-1}{n} \log P(x_1, x_2, ..., x_n) \approx \langle -\log(p(x)) \rangle \equiv H(X), \tag{6.28}$$

in the sense that, for any $\epsilon > 0$ and for n large enough we have

$$H(X) - \epsilon \leq \frac{-1}{n} \log P(x_1, x_2, ..., x_n) \leq H(X) + \epsilon \tag{6.29}$$

Thus:

$$2^{-n(H(X)-\epsilon)} \geq P(x_1, x_2, ..., x_n) \geq 2^{-n(H(X)+\epsilon)} \tag{6.30}$$

A useful equivalent reformulation of this expression is:

$$\left| \frac{-1}{n} \log P(x_1, x_2, ..., x_n) - H(X) \right| \leq \epsilon. \tag{6.31}$$

A sequence that satisfies this property is called $\epsilon - typical$.

In [45], pp. 537, the Shannon's Noiseless Channel Coding Theorem is demonstrated using the following properties about $\epsilon - typical$ sequences.

Property 1: Fix $\epsilon > 0$. Then, for any $\delta > 0$, for sufficiently large n, the probability that a sequence is $\epsilon - typical$ is at least $1 - \delta$.

Property 2: For any fixed $\epsilon > 0$ and $\delta > 0$, for sufficiently large n, the number $|T(n, \epsilon)|$ of $\epsilon - typical$ sequences satisfies:

$$(1-\delta) 2^{n(H(X)-\epsilon)} \leq |T(n, \epsilon)| \leq 2^{n(H(X)+\epsilon)}.$$

Property 3: Let $S(n)$ be a collection with at most 2^{nR} sequences from the source, where $R < H(X)$ is fixed. Then, for any $\delta > 0$ and for sufficiently large n,

$$\sum_{x \in S(n)} p(x) \leq \delta.$$

6.4 Exercises

1. Let x be a continuous random variable with distribution:

$$f(x) = \begin{cases} \frac{x}{6} + k, & if\ 0 \leq x \leq 3, \\ 0 & otherwise. \end{cases} \tag{6.32}$$

   (i) Calculate k; (ii) Find $p(1 \leq x \leq 2)$; (ii) Calculate the rth-moment defined by:

$$\int_{\mathbb{R}} x^r f(x)\, dx.$$

2. Given two discrete random variables x and y, show that $E(x+y) = E(x) + E(y)$.

3. Let x be a random variable with mean μ and standard deviation $\sigma > 0$. Define the random variable:

$$x^* = \frac{x - \mu}{\sigma},$$

   named standardized random variable corresponding to x. Show that $E(x^*) = 0$ and $Var(x^*) = 1$.

4. In the exercises bellow, x and y are discrete random variables with distributions f and g, respectively, and joint distribution h.

(a) Show that $f(x_i) = \sum_j h(x_i, y_j)$and $g(y_j) = \sum_i h(x_i, y_j)$

(b) Let x and y be random variables on the same sample space S with $y = \Phi(x)$. Show that $E(y) = \sum_i \Phi(x_i) f(x_i)$.

(c) Let x be a random variable and k a real constant. Show that: (i) $E(kx) = kE(x)$; (ii) $E(x + k) = E(x) + k$; (iii) $Var(x + k) = Var(x)$; (iv) $Var(kx) = k^2 Var(x)$.

(d) If we define:

$$Cov(x, y) = \sum_{ij} (x_i - E(x))(y_j - E(y)) h(x_i, y_j),$$

then show that:

$$Cov(x, y) = \sum_{ij} x_i y_j h(x_i, y_j) - E(x) E(y).$$

(e) If x and y are independent random variables demonstrate that: (a) $Var(x + y) = Var(x) + Var(y)$; (b) $Cov(x, y) = 0$.

5. Given a image database, how to find the probabilities p_j^i in expression (6.5)?

6. Choose an image database and find the probability distribution p_j^i and the mean vector defined by expression (6.4). Then compute the covariance matrix, defined from expression (6.8).

7. Prove expressions (6.21) and (6.24).

8. Show that, the maximum of the Shannon entropy, subjected to the constraint $\sum_{i=1}^{W} p_i = 1$ is the uniform distribution.

9. Suppose that each pixel value of a digital image is uniformly quantized to B bits. It is desired to extract the first most significant bits in order to get a more compact image representation. Based on the *Shannon's Noiseless Channel Coding Theorem* in section 6.3.2, show that $B.H$, where H is the associated Shannon entropy, is the number of visually significant bits.

10. Demonstrate expression (6.17).

11. Show that, the optimum mean square estimate $\widehat{x}$ for x, from the observation $y(n)$ is given by expression (6.19).

12. From the results and definitions of section 6.2.1, interpret the covariance matrix given by expression (6.8).

Chapter 7

Data Preparation and Patterns Representation

Many areas such as pattern recognition, computer vision, signal processing and medical image analysis, require the managing of huge data sets with a large number of features or dimensions. Behind such *big data* we have medical imaging, scientific data produced by numerical simulations and acquisition systems (earth-orbiting satellites, genomics experiments) as well as complex systems from Nature, Internet, economy/finance, among other areas in society [47]. However, after acquiring vast amounts of data we must extract useful information. This task has proven extremely challenging requiring methodologies that blend traditional statistics and machine learning methods into sophisticated algorithms for information discovery in large volumes of data. So, we are in the context of data mining, that can be defined as 'the process of automatically discovering useful information in large data repositories' [1].

The overall process of converting raw data into useful information is named knowledge discovery. Its main steps are represented in Figure 7.1. From this figure we see that, before applying some data mining procedure, input data must be preprocessed to convert it into an appropriate format for subsequent analysis. For example, in the case of data mining through traditional neural network techniques, the preprocessing step roughly involves to embed the raw data into a suitable numerical space, named feature space. We call feature, any numerical value that can

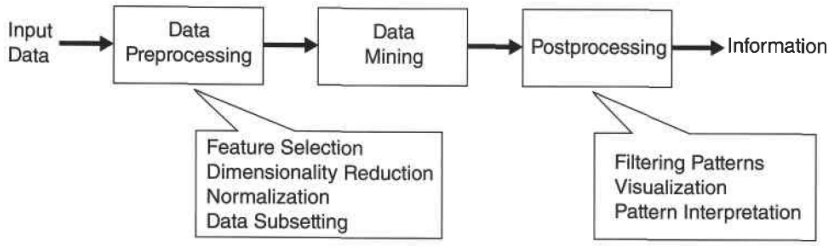


Figure 7.1: Main stages for knowledge discovery (Source [1])

be derived from the input data. Consequently, a feature space is represented using coordinates in $\mathbb{R}^m$, where each coordinate axis holds a specific feature.

In this context, the whole preprocessing, or data preparation pipeline includes the following steps: (i) Data collection; (ii) Data Exploration by looking for outliers, finding exceptions, and missing infomation; (iii) Cleaning by remotion of incorrect and inconsistent samples; (iv) Formatting data, which may include normalization, discretization and quantization; (v) Feature extraction and dimensionality reduction for compact representation; (vi) Data augmentation; (vii) Splitting data into training and evaluation sets.

In this avenue, some specific issues for development of data mining algorithm are: (a) Scalability; that means, efficient solutions in terms of allocation of computational resources (main memory and CPU)); (b) High dimensionality of the input data; (c) Heterogeneous and complex data.

Data mining tasks are generally divided into two major categories [1]:

- Predictive tasks. The objective of these tasks is to predict the value of a particular attribute (variable) based on the values of other attributes Examples: Classification and regression.

- Descriptive tasks. To derive patterns (correlations, trends, clusters, trajectories, anomalies, etc.) that summarize the underlying relationships in data.

In this text, we focus in data whose attributes (variables) are measured in numerical values (continuous or discrete). For instance, the high of individuals in a population is a continuous variable while their ages is a discrete attribute. In this sense, we might have an array of variable associated with each individual. Therefore, the raw data set can be viewed as a collection of objects (or samples) that

can be represented by points/vectors in $\mathbb{R}^m$, matrices, or graphs. For example the image histogram can be seen as a point in $\mathbb{R}^m$, the image itself is mathematically a matriz. Moreover, given a RGB image with a hand gesture, we can perform the hand segmentation for key points extraction to represent the hand region with a graph whose nodes are the key points and edges connect adjacent nodes.

In this text, the terms database and dataset are used as synonyms to reffer to a set of samples taken from some population or acquired through measurements. We will use two kinds of representations for a database D:

Point/Vector Representation: In this case, each sample of the dataset D is a point/vector with m attributes. Therefore, if the database has M samples, then it is a set:

$$D = \{\mathbf{x}_1, \mathbf{x}_2, \cdot, \cdot, \cdot, \mathbf{x}_M\} \subset \mathbb{R}^m. \tag{7.1}$$

The elements of D can be assembled in a $M \times m$ data matrix that is composed of M lines $\mathbf{x}_1^T, \mathbf{x}_2^T, \cdot, \cdot, \cdot, \mathbf{x}_M^T$ and m columns. In machine learning applications this matrix is also named training set matrix. Moreover, each element $\mathbf{x}_i \in D$ is a point in $\mathbb{R}^m$ and, consequently, we can represent S as a graph, where the nodes are the samples $\mathbf{x}_i$ and the edges are connections indicating neighboring relationships. For instance, each node $\mathbf{x}_i$ is connected with its K nearest neighbors.

Matrix Representation: In this case, we have a dataset D where the data samples variables are arranged in a $N \times M$ array A. If most of the $A(i, j)$ are null, we say that we have a sparse matrix. Some data mining algorithms work well only for sparse data. Moreover, in the case of image processing each entry (i, j) of A is named a pixel and we say that the spatial resolution of A is its dimension ($N \times M$). The dimensionality of A is its number of attributes, in this case, this number is given by $N \cdot M$. In general $N \cdot M$ is very large which brings specific challenges for development of efficient algorithms to deal with such high-dimensional data.

Moreover, data attributes may have relationships that involve order in time or space, which demands the following classification.

Time Ordered Data: In this data type, the elements $\mathbf{x}_i$ in the set D are sorted following relationships that involve order in time. Hence, we think the dataset D as a sequence $\mathbf{x}_i$, $i = 1, 2, \cdot \cdot \cdot, M$, where the index i is related to the time that the corresponding sample was recorded. A video record is very usual example of such data type. A subclass of this category is the *Time Series Data* characterized

by high temporal autocorrelation; i.e., we expect that $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ are close with respect to some similarity measure in $\mathbb{R}^m$.

Spatial Ordered Data: The objects are represented through variable associated to points regularly or irregularly distributed in space. For example, numerical simulations in science and engineering generate data sets that are the result of discretizations of physical fields, like pressure and temperature, using a two- or three-dimensional grid or mesh. If fields are stationary (time independent) and we have a regular grid, the obtained data can be arranged using a matrix.

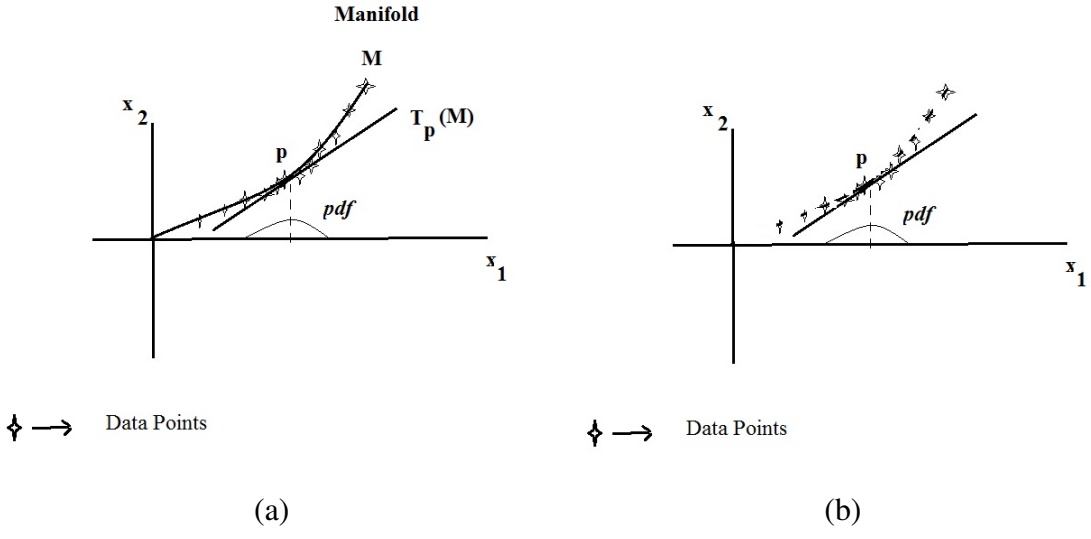
In all the considered data types the acquisition process is not perfect and there may be problems due to human error, limitations of measuring devices, noise and missing data. Moreover, the focused phenomenon may have outliers that are characterized by attribute values that are unusual with respect to the typical values observed. Even, redundancy is another problem not related to acquisition or distribution of the attribute values. Roughly, whenever the same kind of information is repeated in a subset of samples of the database, we can say we have data redundancy [48].

Therefore, we must address the issue of which preprocessing steps should be applied to make the data useful for the data mining step represented in Figure 7.1. The next sections deal with this problem, considering cleanning (section), aggregation (7.3), sampling (7.4), dimensionality reduction (7.5), feature extraction and selection (7.6), discretization and quantization (7.7), measures of similarity and dissimilarity (7.8), and normalization or standardization (section 7.9).

7.1 Data Model and Learning

In this monography we use a data model which main elements are pictured on Figure 7.2.

The main assumption in this data model is that the data points, or samples, lye in a differentiable manifold $\mathcal{M}$ (see section 5.7). In Figure 7.2.(a) the corresponding geometry is represented by a curve (a smooth curve is a differentiable manifold of dimension $n = 1$). Given a point $\mathbf{p} \in \mathcal{M}$, the tangent space $T_{\mathbf{p}}\mathcal{M}$ is the Euclidean space that spans all the vectors that are tangent to $\mathcal{M}$ in $\mathbf{p}$ (see section 5.7). Data points appear according to some probability density function (*pdf*), which is depicted bellow the tangent space in the Figure 7.2.



(a) (b)

Figure 7.2: (a) Data model elements: Original coordinate system (x_1, x_2), manifold ($\mathcal{M}$), tangent space at a point $\mathbf{p} \in \mathcal{M}$, probability density function (*pdf*) . (b) Samples and *pdf* of the data.

However, the input for learning algorithms are just the data points, which are depicted alone in Figure 7.2.(b). The *pdf*, also represented in Figure 7.2.(b), in general, is not known in advance. In general the dimension of the original data point space is very large which requires some dimensionality reduction technique to get a more compact representation and to discard redundancies. So, from a geometric viewpoint, it is equivalent to assumed that the input can be essentially described on a low-dimensional manifold, embedded in some high-dimensional space. In the case of Figure 7.2 the curve represents an one-dimensional manifold embedded in a two-dimensional space.

From the data model of Figure 7.2 some issues arises:

1. Where is the place of dirty data corrupted by acquisition errors in this model?

2. What is the geometric meaning of the *pdf* in Figure 7.2?

3. How to recover the underlying low-dimensional manifold from the samples?

4. How to compute the *pdf* from the samples?

The section 7.2 deals with data cleaning and answers the question (1). If we have a dataset that is representative of some population and we could visualize it over the manifold data, like in Figure 7.2, we could observe some regions with low density of samples while others are densely populated. This phenomenon is explained through the *pdf* behind the data. For example, let us suppose a dataset that is drawn through a multivariate d-dimensional Gaussian distribution given by:

$$G_d\left(\mathbf{x}, \Sigma\right) = \left[\left(2\pi\right)^{d/2} \left|\Sigma\right|^{1/2}\right]^{-1} \exp\left(-\frac{1}{2}\left(\mathbf{x}-\mu\right)^T \Sigma^{-1}\left(\mathbf{x}-\mu\right)\right), \qquad (7.2)$$

where Σ is the covariance matrix and μ is the mean vector. Figure 7.3 pictures a representation of such setup for $d = 2$. We shall notice that sample points are more concentrated nearby the center of the points cloud, which is the position of the mean vector μ. Far from this point, the samples are spacely distributed due to the shape of the Gaussian *pdf*. In this case, the data manifold is the Cartesian plane. In a more general situation, the data manifold is an hypersurface $\mathcal{M}^d$ and the population samples are points over it, drawn from the general *pdf* given by expression (7.2).

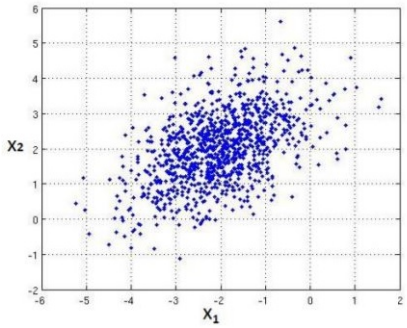


Figure 7.3: Samples drawn from a Gaussian distribution in $\mathbb{R}^2$

The issue (3) can be addressed by manifold learning techniques as well as with machine learning methods. We recognize manifold learning as a background to data analysis that is based on the assumed geometric structure of high-dimensional databases [49, 50]. Manifold learning is the starting point for the so called *geometric data analysis*, which is a combination of differentiable manifold elements and data representation techniques, for extracting meaningful information from

data [51, 52, 53, 54]. It can be implemented through machine and/or statistical learning methods as we will discuss in Chapters 8 and 10, respectively. Problem (4) will be formalized in the context of machine learning in Chapter 8.

7.2 Data Cleanning

The processes involved in data acquisition and collection often introduce errors in data like:

- Noise: It is a random process inherent to the acquisition stage. Geometrically, noise generates perturbations in the sample points around the manifold represented in Figure 7.2;
- Outliers: observations that belong to $\mathcal{M}$ but are far from the region that represents the most expected patterns;
- Incorrect or inconsistent samples: points that lie an abnormal distance from the data manifold $\mathcal{M}$
- Missing values: regions of the data manifold without samples in the database;
- Typos: errors in the collection process;
- Replicated entries: happens also in the collection process or due to limitations in the precision of the acquisition instruments;

Designers of machine learning algorithms must be aware about these issues to be succesfull. They are responsible for dirty data which is the most common barrier faced by researchers in machine learning and data science. Obviously, a visualization of the dataset like the ones in Figures 7.2 and 7.3 could be enough to identify corrupted points. However, such visualization of the dataset is not possible in practice due to the high dimension of raw data. Therefore, we must seek for more general tools and also to have in mind the necessity of techniques for error repairing.

In this way, Figure 7.4 shows a typical workflow for data cleaning. A dirty dataset offers the input for error detecntion and error repair algorithms. However,

it is often necessary to pass raw data to a discovery process, may be assisted by domain experts to model patterns, probability distributions, and other metadata in order to augment the error detection.

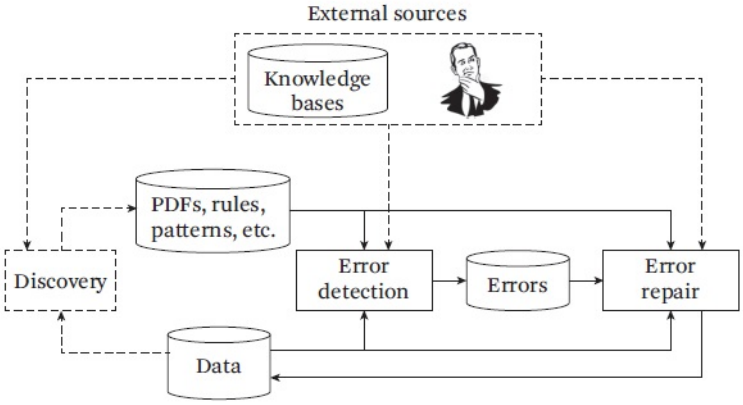


Figure 7.4: Typical workflow with the main data cleaning steps. Source [2].

Given a dirty dataset, the error detection step finds part of the data that does not conform to the expected patterns, and declares this subset to contain errors. Finally, given the errors detected, the error repair step is applied to the dirty dataset. There are many uncertainties in the data cleaning process because, in general, we do not know the *pdf* of patterns of corrupted and clean subsets of a database.

In the case of outlier detection techniques, the geometric view of Figure 7.2 offers a way to think about this problem in terms of statistics-based, distance-based and model-based methods. The former focuses in the stochastic elements of the data model represented in Figure 7.2. Thus, statistics-based techniques assume that the clean data points would appear in high probability regions of the *pdf*, while outliers would occur in low probability regions [55].

Statistics-based outlier detection methods can be assembled into two categories. The first one uses hypothesis testing methods, such as the Grubbs Test [56] and the Tietjen-Moore Test [57]. In the case of Grubbs Test, the method uses the observed data to calculate a test statistic, which is used to determine whether the null hypothesis H_0 (there is no outlier in the dataset) should be rejected against the alternative hypothesis H_a (there is one outlier in the dataset). Hence, if we have a dataset D like in expression (7.1), and we want to test the samples with

respect to the attribute j then the first step is to compute the mean:

$$\overline{\mu}_j = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_{i,j}, \tag{7.3}$$

and the standard deviation:

$$\sigma_j = \sqrt{\frac{\sum_{i=1}^{N}\left(x_{i,j} - \overline{\mu}_j\right)^2}{N}}. \tag{7.4}$$

Next, we shall compute the score:

$$G_j = \max_{i \in \{1,2,\ldots,M\}} \frac{\left|x_{i,j} - \overline{\mu}_j\right|}{\sigma_j}, \tag{7.5}$$

where $\overline{\boldsymbol{x}}$ is the global mean and σ is the standard deviation of the samples computed above.

Following, the null hypothesis of no outliers is rejected if:

$$G_j > \frac{N-1}{\sqrt{N}} \sqrt{\frac{t^2_{\frac{\alpha}{2N},N-2,j}}{N-2+t^2_{\frac{\alpha}{2N},N-2,j}}}, \tag{7.6}$$

where $t^2_{\frac{\alpha}{2N},N-2,j}$is the upper critical value of the t-distribution with $N-2$ degrees of freedom and a significance level of $\alpha/2N$ for the variable j (see [56] for details).

The second category of statistics-based outlier detection techniques is composed by parametric approaches that start by fitting a distribution or inferring a *pdf* based on the observed data. For instance, if we assume that the attribute j of the samples in the dataset D follows the univariate normal distribution, given by expression (7.2) with $d = 1$, then fitting the data samples under the Gaussian distribution essentially means computing the mean and the standard deviation through expressions (7.3) and (7.4). Then, a simple way to identify outliers is to compute a z-score, with respect to the attribute j, for every point in D, which is defined as:

$$Zscore_{i,j} = \frac{\left|x_{i,j} - \overline{x}_j\right|}{\sigma_j}. \tag{7.7}$$

and seek for data points $\boldsymbol{x}_i$ that have a z-score greater than a threshold T, for example $T = 3\sigma_j$. The samples that satisfies $Zscore_j > T$ are considered to be outliers.

In the multivariate test, we must compute the global mean:

$$\boldsymbol{\mu} = \frac{1}{M} \sum_{i=1}^{M} \boldsymbol{x}_i, \tag{7.8}$$

and the covariance matrix associated to the data:

$$R = \sum_{i=1}^{M} (\boldsymbol{x}_i - \boldsymbol{\mu}) (\boldsymbol{x}_i - \boldsymbol{\mu})^T . \tag{7.9}$$

Now, we need to measure the distance between a particular data point $\boldsymbol{x}_i$ to the mean using the Mahalanobis distance, defined by expression [58]:

$$d(\boldsymbol{x}_i, \boldsymbol{\mu}) = \sqrt{(\boldsymbol{x}_i - \boldsymbol{\mu})^T R^{-1} (\boldsymbol{x}_i - \boldsymbol{\mu})^T}, \tag{7.10}$$

For multivariate Gaussian data, the distribution of the squared Mahalanobis distance is known to be chi-squared with the dimension of the data. Hence, the adopted rule for identifying the outliers is selecting the threshold such that 95% of the dataset is smaller than that threshold distance [2, 59]. However, the problem with such approach is that the classical estimates of the mean μ and covariance matrix are influenced by the presence of outliers. So, we must apply estimates of centrality and covariance matrix that are resistant against the influence of outlying observations, giving rise to a robust Mahalanobis distances, as performed in [59].

Parametric approaches for fitting outliers detection have the drawback of assuming the data to follow an underlying distribution. In contrast, nonparametric techniques infer the distribution from the data itself. Mainly we find two types of techniques in this category [2]: histogram-based techniques and kernel density-based techniques.

Histogram-based methods depend on the construction of the data histogram. To explain this, let us suppose that we have a dataset with the high (y) of M individuals of a population, represented as:

$$D = \{y_1, y_2, \cdot, \cdot, \cdot, y_M\} \subset \mathbb{R}. \tag{7.11}$$

If we are going to build the histogram of D to get information about the shape of the probability distribution of the real attribute y, we need to define the number of bins as well as the intervals for each bin. Formally, if the high in D falls in the interval $y_{\min} \leq y \leq y_{\max}$, we should divide the range $Y = [y_{\min}, y_{\max}]$ into

intervals $b_i = [\xi_i, \xi_{i+1})$, $i = 0, 1, \ldots, Z-1$, with $\xi_0 = y_{\min}$ and $\xi_Z = y_{\max}$. Each interval b_i is named a bin of the histogram. The set $\{\xi_0, \xi_1, \cdot, \cdot, \cdot, \xi_Z\} \subset [y_{\min}, y_{\max}]$ is a discretization of the interval Y. Next, we shall count the absolute frequence of each bin b_i with respect to the set D in expression (7.11); that means:

$$C(i) = \sum_{j=1}^{Z-1} \chi(y_j, b_i), \quad i = 0, 1, \ldots, Z-1. \tag{7.12}$$

where:

$$\chi(y_j, b_i) = \begin{cases} 1, & if\ y_j \in b_i, \\ 0, & otherwise. \end{cases} \tag{7.13}$$

This process dependes on some steps, that are application dependent. Specifically, given the data set D in equation (7.11):

1. Search D do find out the range $Y = [y_{\min}, y_{\max}]$;
2. Choose the discretization $\{\xi_0, \xi_1, \cdot, \cdot, \cdot, \xi_Z\} \subset [y_{\min}, y_{\max}]$. This step defines the intervals $b_i = [\xi_i, \xi_{i+1})$;
3. Compute the absolute frequences through the expressions (7.12)-(7.13);
4. Assemble the frequences in an array $H = (C(0), C(1), \ldots, C(Z-1))$.

Anyway, once the histogram is computed, we can analyse the relative frequency distribution ($C(i)/M$, $i = 0, 1, \ldots, Z-1$) and consider as outliers the samples that count for bins with with very low frequency. Moreover, we can comput a kind of z-score and perform like before.

There are usually two ways to generalize histogram-based approaches to deal with multivariate data: (1) For each attribute compute an one-dimensional histogram and an outlier score, and then combine the scores into an overall outlier score for every data point; (2) Subdivide every dimension and count the number of data points belonging to each multidimensional cell, and consider outliers the data points that belong to cells with very low frequency. Other approaches in for outlier detection are

Distance-Based outlier detection methods often define a distance between data points that is used to stablish a criterion to characterize regular behavior. Thinking

through the data model in Figure 7.2 we are considering the maninfod $\mathcal{M}$ as a Riemannian one and using the metric to compute arch lengths through expression (5.41). In this way, a normal data point should be close to many other sample of the dataset, and data points that considerably deviate from such normal behavior are declared outliers. Distance-based outlier detection methods can be further divided into global or local methods depending on the reference population used when determining whether a point is an outlier. A global distance-based outlier detection method determines whether a point is an outlier based on the distance between that data point and all other data points in the dataset. On the other hand, a local method considers the distance between a point and its neighborhood points when determining outliers [2].

Model-based outlier detection techniques first learn a classifier model from a set of labeled data points, and then apply the trained classifier to a test data point to determine whether it is an outlier [2].

7.3 Aggregation

The data matrix commented in the introduction of this chapter is an example of aggregation (combining two or more objects into a single structure). Formally, given the dataset D in expression 7.15, we generate the matrix:

$$X = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_{M-1}^T \\ \mathbf{x}_M^T \end{bmatrix}, \tag{7.14}$$

Certainly, the graph composed by nodes $\mathbf{x}_i \in S$ connected by edges indicating neighboring relationships is another example of aggregation.

In this line, let us consider a dataset composed by K video records (like Youtube). Firstly, we shall taking in mind that mathematically, each video is a time ordered data composed by a number of images, also named frames, with some spatial resolution, say, $N \times M$. To simplify the explanation, we suppose that each

video has F frames. Hence, each video is a generalized matrix $V \in \mathbb{R}^{M \times N \times F}$. Consequently, the dataset D in this case can be represented as:

$$D = \{V_1, V_2, \cdot, \cdot, \cdot, V_K\} \subset \mathbb{R}^{M \times N \times F}. \tag{7.15}$$

Hence, an specific element of V_i is acessed by $V_i\,(i_1, i_2, i_3)$ where $1 \leq i_1 \leq M$, $1 \leq i_2 \leq N$, $1 \leq i_3 \leq F$. Going ahead, the dataset D in expression (7.15) can be aggregated into a higher dimensional matrix $\Psi \in \mathbb{R}^{M \times N \times F \times K}$ where $\Psi\,(i_1, i_2, i_3, i_4)$ is the intensity value of the pixel (i_1, i_2), in the frame i_3 of video i_4.

7.4 Sampling

It is common that the set D in expression (7.1) has a cardinality M very high. Therefore, it is too expensive or time consuming to process all the data. So, we must select a subset of D to be analyzed. This process is named sampling.

Also, we need to perform sampling to train machine learning algorithms. For example, to train and validate a neural network, we should divide the dataset D into two subsets, say D_1 and D_2, such that $D_1 \cap D_2 = \emptyset$. The former is used for training and the set D_2 to validate the model.

In general, the choice of elements to build the subsets D_1 and D_2 is randomly implemented, using a uniform distribution. The determination of the proper size of D_1 and D_2 depends on a statistical and methodical approaches. For example, in natural science applications, we shall notice that the acquisition process that generates the dataset D performs a sampling of a continuous process. So, the cardinality of D must be enough to catch the distribution of samples in the focused phenomenon. However, a larger number M of samples is necessary to obtain an acceptable performance when increasing dimensionality. Specifically, the number M grows exponentially with the value of m [60]. Such property can be verified through the way decision tree works. In this technique each feature is represented by an one-dimensional interval that needs to be partitioned into regular B intervals during tree construction. The intersections of these intervals generate cells that should receive a class label that depends on the samples inside it. Therefore, we will need B^m samples to avoid unlabeled cells. This phenomenon is referred as the curse of dimensionality (see also [44]).

7.5 Dimensionality Reduction

In general, applications of data mining techniques for pattern recognition and signal processing require the managing of huge data sets with a large number of features or dimensions. Therefore, dimensionality reduction may be necessary in order to discard redundancy and simplify further operations. The most know technique in this subject is the Principal Components Analysis (PCA) [61]. Given the dataset D in expression (7.1), the output of PCA technique is linear transformation that projects the original data into a reduced space $\mathbb{R}^{m\prime}$, where $m\prime < m$.

The PCA approach needs that the dataset is represented as arrays in $\mathbb{R}^m$. If data samples are matrices, then, we must convert them into vectors before apply PCA. This process is not unique. Such ambiguity do not respect the spatial order of the data. Consequenlty, multilinear representations using traditional and generalized matrices (like the ones in expression (7.15)), also called tensors, have been proposed to address this issue.

Tensor methods offer an unified framework because vectors and matrices are first and second order tensors, respectively. Besides, colored images can be represented as third order tensors and so on. The applications of multilinear data representation include face image analysis [62], face recognition under multiple viewpoints and varying lighting conditions, digital number recognition, video content representation and retrieval, face transfer that maps video recorded performances of one individual to facial animations of another one, gait recognition [63], geoscience and remote sensing data mining, visualization and computer graphics techniques based on tensor decomposition (see [64] and references therein).

Moreover, considering the data model in section 7.1, we agree that geometry and topology are natural tools to handle large, high-dimensional databases because we can encode in a geometric concept (like a differentiable manifold) notions related to similarity, vectors and tensors while the topology allows to synthesize knowledge about connectivity to understand how data is organized on different levels [65, 49]. In this scenario, manifold learning methods and tensor fields in manifolds offer new ways of mining data spaces for information retrieval [66]. Manifold learning methods look for a compact data representation by recovering the data geometry and topology [67]. The manifold structure offers resources to build vector and tensor fields which can be applied to encode data features.

7.6 Feature Extraction and Selection

As already explanined, in machine learning area, we call feature any numerical value that can be derived from the input data. In the case of image processing, each histogram (expression 7.12) entrie is also a feature and the histogram can be seen as a feature vector. Besides, the acquisition of information to build the feature space also can be obtained through texture descriptors, wavelets analysis and image transforms as well as convolutional neural networks (CNNs) [68, 69, 19].

However, once computed the features from the data set, there is need of selecting the most important discriminant features for pattern recognition tasks, for instance, classification ones [22]. Discriminant analysis techniques, that in the literature are known as discriminant functions, address this type of problem [70]. Traditionally, discriminant functions have been implemented using Fisher's linear discriminant analysis (LDA) and its variants [22, 71]). The LDA finds a projection transformation that maximizes the Fisher criterion for finding (at most) number of groups - 1 meaningful discriminant directions, that, in general, are different from the original features directions [22]. Besides, inspired in the LDA criterion, it is proposed in [72, 73] the Fisher discriminability criterion (FDC) that ranks the features according to the ratio of the between-class scatter over the within-class scatter.

On the other hand, another category encompasses techniques that receive as input a set of features and select, among the input features, the most discriminant ones to solve some recognition problem. From the viewpoint of this work, an important class of methods in this category is the embedded one which is composed by approaches that take into account the biases of classifiers by integrating discriminant analysis with the classifier construction [74]. Among embedded techniques, we have random forests [75], support vector machine (SVM) based approaches [76, 77], Zhu and Martinez method [72] and Fisher discriminant criterion [73].

Others important categories of feature selection methods are filter and wrapper approaches [78]. The former include methods that select/weight the features without the use of learning algorithms. Among filters methods, we have mutual information (MI) [79], t-statistics [80], Pearson's correlation [81], Minimum redundancy maximum relevance (MRMR) [82], correlation-based feature selection

(CFS) [83], Fisher criterion [84], Relief and Relieff [85]. These methods are easily adapted for data sets with high dimensional space. They are faster if compared with models in the wrapper and embedded classes because they do not use the feedback of classifiers. However, this is also their weakness in the sense that it is hard to quantify feature importance without the help of a separating surface defined by a learner.

Wrapper techniques are more efficient than filters models because they use the predictive accuracy of a classifier to select features [76]. However, such advantage has the price of being computationally involved mainly in high dimensional feature spaces [76, 86] due to the way wrapper methods work: (i) Searching a subset of features. (ii) Next, check the selected subset of feature using the performance of the classifier. These two stages are repeated until reach highest accuracy which is responsible for the high computational cost of the algorithms.

7.7 Discretization and Quantization

Some data visualization algorithms require that the data be in the binary form. For example, if we have discretized (through numerical simulation or a sampling process) the temperature field in a plate using a regular $N_1 \times N_2$ grid, then the discrete temprature field can be represented by a matrix $T \in \mathbb{R}^{N_1 \times N_2}$. Each entry of T is a real value. If we want to visualize this matriz as a color map, we have two possibilities: (a) Search T do find out the range $\mathcal{T} = [T_{\min}, T_{\max}]$, choose a discretization $\{\xi_0, \xi_1, \cdot, \cdot, \cdot, \xi_Z\} \subset [y_{\min}, y_{\max}]$ to define the intervals $b_i = [\xi_i, \xi_{i+1})$, and represent each interval $b_i = [\xi_i, \xi_{i+1})$ with a pre-defined color; (b) Perform quantization of the temperatura field.

For case (b), we first represent the intensities in matrix T through an array: $\{f(n), \quad 0 \leq n \leq N_1 \cdot N_2\}$, where each value $f(n)$ is a real variable. Te heart of the process to get the quantization is a quantizer.

The input of a quantizer is the sampled function $f(n) \in \Re$, and the output is the sequence $f^*(n) \in \{r_1, r_2, \cdots, r_L\}$. The values r_i, $i = 1, 2, ..., L$, are called the reconstruction levels.

So, let us consider a continuous variable u, defined in the interval $a \leq u \leq b$. The process of mapping a continuous variable u in a variable u^* which takes values from the finite set $\{r_1, r_2, \cdots, r_L\} \subset \Re$ is called **quantization**.

In the quantization process, we must define a set of transition levels $\{t_k, \quad k = 1, 2, ..., L+1\}$, with $t_1 = a$ and $t_{L+1} = b$, such that $r_k \in [t_k, t_{k+1})$. Then, the discrete variable u^* can be defined as follows:

$$u^*(u) = r_k, \quad if \quad u \in [t_k, t_{k+1}). \tag{7.16}$$

Such mapping can be represented by the staircase function of Figure 7.5, which also pictures the quantization error.

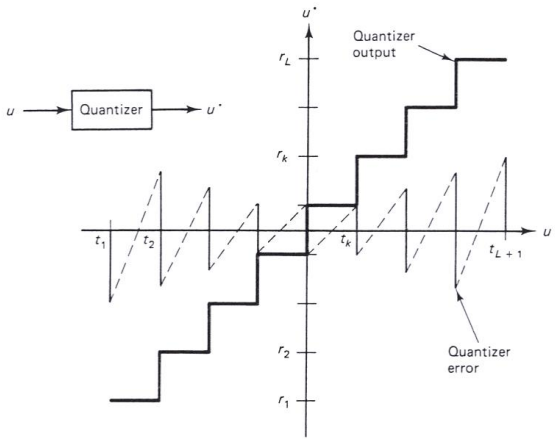


Figure 7.5: Quantization and the corresponding error.

For instance, if $0 \leq u \leq 10$ and the samples are uniformly quantized to 256 levels, then, the transition t_k and reconstruction r_k levels are, respectively, given by:

$$t_k = \frac{10\,(k-1)}{256}, \quad k = 1, 2, \cdots, 257 \tag{7.17}$$

$$r_k = t_k + \frac{5}{256}, \quad k = 1, 2, \cdots, 256. \tag{7.18}$$

In this case, the interval $q = t_k - t_{k-1} = r_k - r_{k-1}$ is constant for different values of k and it is called the quantization interval.

Obviously, in this process there is lost of information. So, a good quantizer is one which represents the original signal with minimum loss or distortion. Therefore, some optimization criterium must be considered in order to design a suitable

quantizer, which may be more efficient than the simple choice given by expressions like (7.17)-(7.18). This subject has a plenty of techniques, involving optimization as well as machine leaning algorithms [87, 88].

7.8 Measures of Similarity and Dissimilarity

Based on the deta model we may have similarity measures based on gemetric or statistics/probabilistic elements. Geometric-based similarity measures take more or less the concepts related to the manifold metric to define criteria do compare samples (see [89] and references therein). For instance, in the case of $\mathbb{R}^m$ the usual Euclidean distance is a simple possibility to compare two samples $\boldsymbol{x} \in \mathbb{R}^m$ and $\boldsymbol{y} \in \mathbb{R}^m$ through the expression:

$$d_e\left(\boldsymbol{x}, \boldsymbol{y}\right) = \sqrt{\left(\boldsymbol{x} - \boldsymbol{y}\right)^T \left(\boldsymbol{x} - \boldsymbol{y}\right)}. \tag{7.19}$$

We can also use other distance definitions in $\mathbb{R}^m$, like the p-norm and max-norm distances, defined respectively by:

$$d_p\left(\boldsymbol{x}, \boldsymbol{y}\right) = \left(\sum_{i=1}^{m} \left|x_i - y_i\right|^p\right)^{1/p}, \tag{7.20}$$

$$d_{max}\left(\boldsymbol{x}, \boldsymbol{y}\right) = \max\left\{\left|x_i - y_i\right|,\ i = 1, 2, \ldots, m\right\}. \tag{7.21}$$

One step further allows to incorporate statitical elements in the Euclidean distance to generate the Mahalanobis distance. So, given the data set D in expression (7.1), its mean vector $\boldsymbol{\mu}$ and covariance matrix R defined as:

$$\mu = \frac{1}{N} \sum_{i=1}^{M} \mathbf{x}_i, \tag{7.22}$$

$$R = \sum_{j=1}^{M} \left(\mathbf{x}_j - \boldsymbol{\mu}\right) \left(\mathbf{x}_j - \boldsymbol{\mu}\right)^T, \tag{7.23}$$

the Mahalanobis distance from a generic sample $\boldsymbol{x}$ do the mean vector $\boldsymbol{\mu}$ is given by:

$$d_{Mah}\left(\boldsymbol{x}, \boldsymbol{\mu}\right) = \sqrt{\left(\boldsymbol{x} - \boldsymbol{\mu}\right)^T R^{-1} \left(\boldsymbol{x} - \boldsymbol{\mu}\right)}, \tag{7.24}$$

Given two samples $\boldsymbol{x}, \boldsymbol{y} \in D$ we can also compute a similarity measure between them by using the formalism behind Mahalanobis distance:

$$d_{Mah}\left(\boldsymbol{x}, \boldsymbol{y}\right) = \sqrt{\left(\boldsymbol{x}-\boldsymbol{y}\right)^{T} R^{-1} \left(\boldsymbol{x}-\boldsymbol{y}\right)}, \tag{7.25}$$

Expression (7.19) can be generalized to the matrix space $\mathbb{R}^{n\times m}$, where $m, n \in \mathbb{N}$, generating the Frobenius distance [38]. Going ahead, we can consider generalized matrix spaces $\mathbb{R}^{n_1\times n_2\times \ldots \times n_k}$, $k, n_1, n_2, \ldots, n_k \in \mathbb{N}$ and also compute Frobenius distance between two elements $X, Y \in \mathbb{R}^{n_1\times n_2\times \ldots \times n_k}$, also named tensors [90].

All these definitions allow to compare samples in order do identify repetions of the same point in the dataset. We can demonstrate that the above definitions follows the metric axioms. Moreover, behind expressions (7.19)-(7.21) we have a norm function as follows:

- Euclidean norm:

$$Norm_{e}\left(\boldsymbol{x}\right) = \sqrt{\left(\boldsymbol{x}\right)^{T} \left(\boldsymbol{x}\right)}, \tag{7.26}$$

- P-norm

$$Norm_{p}\left(\boldsymbol{x}\right) = \left(\sum_{i=1}^{m} \left|x_i\right|^{p}\right)^{1/p}, \tag{7.27}$$

- Max-norm:

$$d_{max}\left(\boldsymbol{x}\right) = \max\left\{\left|x_i\right|,\ i = 1, 2, \ldots, m\right\}. \tag{7.28}$$

It is important to highlight that we can use similarities measures that do not define metric spaces. For instance, to comapare two image x and y we can use the measure of structural similarity (SSIM) given by [91]:

$$SSIM(x, y) = \frac{\left(2\mu_x\mu_y + c_1\right)\left(2\sigma_{xy} + c_2\right)}{\left(\mu_x^2 + \mu_y^2 + c_1\right)\left(\sigma_x^2 + \sigma_y^2 + c_2\right)}, \tag{7.29}$$

where μ_x and μ_y are the average intensities of images x and y, respectively; σ_x^2 and σ_y^2 are the corresponding intensity variances; σ_{xy} is the covariance of the intensities of images x and y; c_1 and c_2 are control variables that prevent division by 0.

It can be shown that $SSIM(x, y) \in [0, 1]$. Other possibilities for image similarity measures can be found in [92].

Having a similarity function $d : \mathbb{R}^m \times \mathbb{R}^m \longrightarrow \mathbb{R}$ we can generate a dissimilarity measure $diss(\boldsymbol{x}, \boldsymbol{y})$ that, in general, is thinking in normalized form $diss(\boldsymbol{x}, \boldsymbol{y}) \in [0, 1]$. For instance, diven the dataset D and the Eucliean metric (7.19), we can define the dissimilarity measure in D as:

$$diss : D \times D \longrightarrow [0, 1],$$

$$diss(\boldsymbol{x}, \boldsymbol{y}) = 1 - \frac{d_e(\boldsymbol{x}, \boldsymbol{y})}{d_{max}}, \tag{7.30}$$

where $d_{max} = \max\{d_e(\boldsymbol{x}, \boldsymbol{y}),\ \boldsymbol{x}, \boldsymbol{y} \in D\}$.

Mahalanobis distance is a simple example of the combination of geometric and statistics elements. For a further step, we can consider a stochastic model of a population given by a random vector $\boldsymbol{x}$ and a probability density function f . So, we can compute the mean vector and the covariance matrix following expressions (6.4) and (6.8), in order to compute the distance between two samples $\boldsymbol{x}, \boldsymbol{y}$ of the pupulation.

7.9 Normalization and Standardization

Let us return to the dataset D in equation (7.1). We can find a boundin box $\mathcal{B} \subset \mathbb{R}^m$ such that $D \subset \mathcal{B}$. If we normalize D such that its normalized version $\widehat{D} \subset [0, 1]^m$, where $[0, 1]^m$ is a unitary cube in $\mathbb{R}^m$, we could design algorithms all machine learning algorithms to work in $[0, 1]^m$ and, apply the inverse of the normalization process to get the output in the original $\mathcal{B}$ set. This is one reason to use normalization in pattern recognition applications.

The other one is related to the specific properties of data itself. As an example in fluid dynamics, consider the analysis of two-dimensional flows simulated simulated by a technique based on particles with positions $\mathbf{q}_i \in \mathbb{R}^2$ and velocities $\dot{\mathbf{q}}_i \in E^2$, with E^2 being the two-dimensional Euclidean space. Therefore, the configuration space, denoted by $\mathcal{C}$, is defined by the set:

$$\mathcal{C} = \{(\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_M) \in D \times D \times \cdots \times D; \quad i \neq j \Longleftrightarrow \mathbf{q}_i \neq \mathbf{q}_j\}, \tag{7.31}$$

where $\mathbf{q}_i = (q_{i1}, q_{i2}) \in \mathbb{R}^2$, $i = 1, 2, \cdots, M$, are the position of the particles. On the other hand, the phase space is define by the positions $\mathbf{q}_i$ and the (linear)

momentum $\mathbf{p}_i$ of particles computed by $\mathbf{p}_i = m_i\dot{\mathbf{q}}_i \in E^2$. Consequently, the phase space is given by the set:

$$\mathcal{F} = \{((\mathbf{q}_1, \mathbf{p}_1), (\mathbf{q}_2, \mathbf{p}_2), \cdots, (\mathbf{q}_M, \mathbf{p}_M)) \in (D \times E^2) \times (D \times E^2) \times \cdots \times (D \times E^2); \quad i \neq j \Longleftrightarrow \mathbf{q}_i \neq \mathbf{q}_j\}. \tag{7.32}$$

Additionally, we can define the momentum space, denoted by $\mathcal{P}$:

$$\mathcal{P} = \left\{(\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_M) \in E^2 \times E^2 \times \cdots \times E^2\right\}, \tag{7.33}$$

The phase space $\mathcal{F}$ contains the state of the system, given by the position and momentum of particles. Hence, it is more complete than the other two spaces because $\mathcal{C}$, as well as $\mathcal{P}$, keeps only part of the state information. However, $\mathcal{F}$ might suffer from redundancy more than $\mathcal{C}$ and $\mathcal{P}$. So, it is not clear in advance the more efficient space to compute summaries. Therefore, we shall develop a methodology that works with anyone of the spaces cited above.

In order to go ahead, we need the definition of metrics in the spaces defined by expressions (7.31)-(7.33). However, the geometric structure of the $\mathcal{F}$ space is not known in advance due to the viscous and boundary effects that enters the simulation. Therefore, we just suppose $\mathcal{P}$, $\mathcal{C}$ immersed in a high dimensional Euclidean space $\mathbb{E}$, and, consequently, $\mathcal{F}$ immersed in $\mathbb{E} \times \mathbb{E}$. Hence, the distance functions in $\mathcal{P}$, $\mathcal{C}$ and $\mathcal{F}$ are inherited from the corresponding environment spaces. So, we consider the following metrics:

$$d_E : \mathbb{E} \times \mathbb{E} \to \mathbb{R}^+, \quad d_E(\mathbf{q}, \mathbf{r}) = \left[\sum_{i=1}^{M} \left((q_{i1} - r_{i1})^2 + (q_{i2} - r_{i2})^2\right)\right]^{1/2} \tag{7.34}$$

$$d_{Cmax} : \mathcal{C} \times \mathcal{C} \to \mathbb{R}^+, \quad d_{Cmax}(\mathbf{q}, \mathbf{r}) = max\left\{\left((q_{i1} - r_{i1})^2 + (q_{i2} - r_{i2})^2\right)^{1/2}, i = 1, 2, \cdots, M\right\}, \tag{7.35}$$

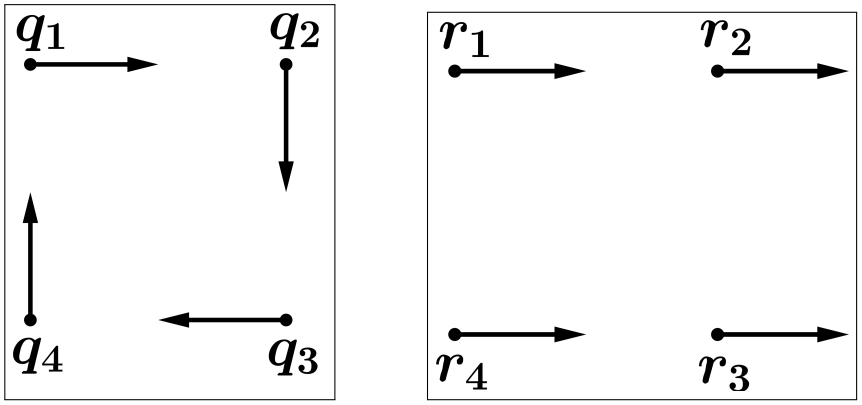
$$d_{Pmax} : \mathcal{P} \times \mathcal{P} \to \mathbb{R}^+, \quad d_{Pmax}(\mathbf{p}, \mathbf{w}) = max\left\{\left((p_{i1} - w_{i1})^2 + (p_{i2} - w_{i2})^2\right)^{1/2}, i = 1, 2, \cdots, M\right\}, \tag{7.36}$$

$$d_{Fmax} : \mathcal{F} \times \mathcal{F} \to \mathbb{R}^+, \quad d_{Fmax}((\mathbf{q}, \mathbf{p}), (\mathbf{z}, \mathbf{w})) = max\left\{d_E(\mathbf{q}, \mathbf{z}), d_E(\mathbf{p}, \mathbf{w})\right\}. \tag{7.37}$$

$$d_{EF} : \mathcal{F} \times \mathcal{F} \to \mathbb{R}^{+}, \quad d_{EF}\left((\mathbf{q},\mathbf{p}),(\mathbf{z},\mathbf{w}),\alpha,\beta\right) = \left[\alpha \sum_{i=1}^{M} \left((q_{i1}-z_{i1})^2 + (q_{i2}-z_{i2})^2\right) + \beta \sum_{i=1}^{M} \left((p_{i1}-w_{i1})^2 + (p_{i2}-w_{i2})^2\right)\right]^{1/2}, \tag{7.38}$$

where $\alpha, \beta \in \{0, 1\}$.

The definitions (7.34) and (7.35) are motivated by the fact that the configuration of the fluid volume in the domain D, at a time t, is defined by the position of the particles in that instant. For instance, the Figure 7.6 pictures two particle systems with equal configurations $\mathbf{q} = (\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \mathbf{q}_4)$ and $\mathbf{r} = (\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3, \mathbf{r}_4)$, but different momentum fields at simulation time t. In this case, we have $\mathbf{q}_i = \mathbf{r}_i$, $i = 1, 2, 3, 4$; so the distance from $\mathbf{q}$ to $\mathbf{r}$, given by expressions (7.34) and (7.35) are: $d_E(\mathbf{q}, \mathbf{r}) = d_{Cmax}(\mathbf{q}, \mathbf{r}) = 0$. However, if we take d_{Pmax} in equation (7.36) the distance of the momentum fields will not be null (the same for d_{Fmax} and d_{EF} above). Obviously, the configuration changes, according to the system dynamics, which involves the velocity and, consequently, the linear momentum. So, in the next simulation step, say $t + \delta t$, the position of the particles follow different paths in the domain D and the distances (7.34) and (7.35) get it.



(a) (b)

Figure 7.6: (a) Particles configuration and momentum field at time t. (b) The same configuration of particles but different momenta.

So, from the dynamical point of view, the metric defined by expressions (7.37) and (7.38) are more complete. However, these metrics involve two spaces (posi-

tion and momentum) and, in terms of data analysis, their inputs are composed by two data sets defined by the position $\mathbf{q}$ and momentum vector $\mathbf{p}$ of the system. Hence, the metrics d_{Fmax} and d_{EF} for $(\alpha, \beta) = (1, 1)$ are sensitive to the position and momentum distribution in the phase space. Specifically, the set that is spread out over a larger range may dominate in equation (7.37) which implies that we can not decide whether a given distance value is significant or not. A solution to remedy this problem is to center each data set respect to the corresponding mean and, next, to perform spatial normalization by using the standard deviations for avoiding that variables with larger variances dominate the corresponding deviations from the mean. Formally, we replace the Euclidean distance (7.34) by a simplified version of the Mahalanobis distance [22], computed by the following procedure.

In the phase space, the result of N steps of simulation is a sequence of numerical frames $(\mathbf{q}(t_j), \mathbf{p}(t_j))$, $j = 0, 1, 2, \cdots, N$, where:

$$\mathbf{q}(t_j) = \left(\mathbf{q}_1^j, \mathbf{q}_2^j, \cdots, \mathbf{q}_M^j\right) \equiv \mathbf{q}^j, \quad j = 0, 1, 2, \cdots, N, \tag{7.39}$$

$$\mathbf{p}(t_j) = \left(\mathbf{p}_1^j, \mathbf{p}_2^j, \cdots, \mathbf{p}_M^j\right) \equiv \mathbf{p}^j, \quad j = 0, 1, 2, \cdots, N, \tag{7.40}$$

with $\mathbf{q}_i^j = \left(q_{i,1}^j, q_{i,2}^j\right) \in \mathbb{R}^2$ being the position of the particle $\mathbf{q}_i$ at simulation time $t = t_j$ and $\mathbf{p}_i^j = \left(p_{i,1}^j, p_{i,2}^j\right)$ is the particle momentum at the same time.

1) Compute the position and momentum means:

$$\overline{\mathbf{q}} = \frac{1}{N}\sum_{j=0}^{N} \mathbf{q}^j, \tag{7.41}$$

$$\overline{\mathbf{p}} = \frac{1}{N}\sum_{j=0}^{N} \mathbf{p}^j, \tag{7.42}$$

2) Centering data:

$$\widetilde{\mathbf{q}}^j = \mathbf{q}^j - \overline{\mathbf{q}} = \left(\widetilde{q}_{11}^j, \widetilde{q}_{12}^j; \widetilde{q}_{21}^j, \widetilde{q}_{22}^j; \widetilde{q}_{31}^j, \widetilde{q}_{32}^j; \cdots; \widetilde{q}_{M1}^j, \widetilde{q}_{M2}^j\right), \tag{7.43}$$

$$\widetilde{\mathbf{p}}^j = \mathbf{p}^j - \overline{\mathbf{p}} = \left(\widetilde{p}_{11}^j, \widetilde{p}_{12}^j; \widetilde{p}_{21}^j, \widetilde{p}_{22}^j; \widetilde{p}_{31}^j, \widetilde{p}_{32}^j; \cdots; \widetilde{p}_{M1}^j, \widetilde{p}_{M2}^j\right), \tag{7.44}$$

for $j = 0, 1, 2, \cdots, N$.

3) Compute Scatter vectors:

$$\widetilde{\Psi}_{ik} = \left(\frac{1}{N} \sum_{j=1}^{N} |\widetilde{q}_{ik}^{j}|^2 \right)^{1/2}, \; k = 1, 2; \; i = 1, 2, \cdots, M, \tag{7.45}$$

$$\widetilde{\Upsilon}_{ik} = \left(\frac{1}{N} \sum_{j=1}^{N} |\widetilde{p}_{ik}^{j}|^2 \right)^{1/2}, \; k = 1, 2; \; i = 1, 2, \cdots, M. \tag{7.46}$$

4) Normalization:

$$\hat{\mathbf{q}}^{j} = \left(\frac{\widetilde{q}_{11}^{j}}{\widetilde{\Psi}_{11}}, \frac{\widetilde{q}_{12}^{j}}{\widetilde{\Psi}_{12}}; \frac{\widetilde{q}_{21}^{j}}{\widetilde{\Psi}_{21}}, \frac{\widetilde{q}_{22}^{j}}{\widetilde{\Psi}_{22}}; \cdots; \frac{\widetilde{q}_{M1}^{j}}{\widetilde{\Psi}_{M1}}, \frac{\widetilde{q}_{M2}^{j}}{\widetilde{\Psi}_{M2}} \right), \tag{7.47}$$

$$\hat{\mathbf{p}}^{j} = \left(\frac{\widetilde{p}_{11}^{j}}{\widetilde{\Upsilon}_{11}}, \frac{\widetilde{p}_{12}^{j}}{\widetilde{\Upsilon}_{12}}; \frac{\widetilde{p}_{21}^{j}}{\widetilde{\Upsilon}_{21}}, \frac{\widetilde{p}_{22}^{j}}{\widetilde{\Upsilon}_{22}}; \cdots; \frac{\widetilde{p}_{M1}^{j}}{\widetilde{\Upsilon}_{M1}}, \frac{\widetilde{p}_{M2}^{j}}{\widetilde{\Upsilon}_{M2}} \right), \tag{7.48}$$

5) Finally, the computation of the distance from $(\mathbf{q}, \mathbf{p}) \in \mathcal{F}$ to $(\mathbf{z}, \mathbf{w}) \in \mathcal{F}$ is performed by:

$$d_{Fmax}\left((\mathbf{q}, \mathbf{p}), (\mathbf{z}, \mathbf{w})\right) = d_{Fmax}\left((\widehat{\mathbf{q}}, \widehat{\mathbf{p}}), (\hat{\mathbf{z}}, \widehat{\mathbf{w}})\right) = max\left\{d_E\left(\widehat{\mathbf{q}}, \widehat{\mathbf{z}}\right), d_E\left(\widehat{\mathbf{p}}, \widehat{\mathbf{w}}\right)\right\}. \tag{7.49}$$

or:

$$d_{EF}\left((\mathbf{q}, \mathbf{p}), (\mathbf{z}, \mathbf{w}), 1, 1\right) = d_{EF}\left((\widehat{\mathbf{q}}, \widehat{\mathbf{p}}), (\hat{\mathbf{z}}, \widehat{\mathbf{w}}), 1, 1\right), \tag{7.50}$$

where d_{Fmax} is defined by expression (7.37) and d_{EF} through equation (7.38).

From the above considerations, we shall consider four possibilities: (i) The phase space; (ii) Normalized phase space; (iii) The momentum space; (iv) Only the configuration space.

Chapter 8

Data and Learning Theory

We can embed the learning problem (*the problem of seeking for the desired dependence on the basis of empirical data [7]*) into the data model of section 7.1. In this line, the *empirical data* is just composed by samples of the underlying geometry. Also, data samples may be associated to labels indicating categories (or classes). Moreover, to each sample we can associate a real value corresponding to some continuous attribute of the target phenomenon. In this context, we can consider the following viewpoints to the learning problem.

1. Given a data set $\mathbb{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\} \subset \mathbb{R}^m$, estimate a function $f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$, a parametric representation of the data manifold $\mathcal{M}$ that satisfies $\mathbb{D} \subset \mathcal{M}$;

2. In the case of labeled data; for instance, $\mathbb{D} = \{(\mathbf{x}_1, l_1), (\mathbf{x}_2, l_2), \ldots, (\mathbf{x}_M, l_M)\} \subset \mathbb{R}^m \times \{0, 1\}$, estimate a function $f : \mathbb{R}^m \longrightarrow \{0, 1\}$ such that $f(\mathbf{x}_i) = l_i$, $i = 1, 2, \ldots, M$.

3. If $\mathbb{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_M, y_M)\} \subset \mathbb{R}^m \times \mathbb{R}$, solve the regression problem: compute a function $f : \mathbb{R}^m \longrightarrow \mathbb{R}$ such that $f(\mathbf{x}_i) = y_i$, $i = 1, 2, \ldots, M$.

4. Estimate the probability density function (pdf) to get stochastic dependencies in the data $D = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\} \subset \mathbb{R}^m$. In this case, we seek for a random vector $\boldsymbol{\omega} \in \mathbb{R}^s$, and a probability density function $f : \mathbb{R}^s \longrightarrow [0, 1]$

such that, the probability of an event $\mathcal{E} \subset \mathcal{M}$ is:

$$p(\mathcal{E}) = \int_R f(\boldsymbol{\omega})\, d\boldsymbol{\omega}, \tag{8.1}$$

where $R \subset \mathbb{R}^s$ parameterizes the event $\mathcal{E} \subset \mathcal{M}$.

In the items (1)-(3), we are estimating *functional dependencies* while in the item (4) we are computing a *stochastic dependency* through the dataset $\mathbb{D}$.

8.1 Learning From Examples: Mathematical Perspective

Items (1)-(4) of the introduction of this Chapter formalize the general model of learning from examples in the context of machine learning. The basic elements of this model are:

- The dataset $\mathbb{D} \subset \mathbb{R}^m$, named input set,
- The learning machine $\mathbb{M}$,
- The training stage.

From a mathematical viewpoint, the learning machine encapsulates a functional space through its inner parameters. Therefore, besides the data space, represented by $\mathbb{R}^m$, we have a parameter space, say $\mathbb{R}^k$. So, the machine $\mathbb{M}$ is in fact:

- A family of functions $\mathbb{M} = \mathbb{M}(\boldsymbol{y}; \boldsymbol{z})$, where $(\boldsymbol{y}, \boldsymbol{z}) \in \mathbb{R}^n \times \mathbb{R}^k$ in case (1),
- A family of functions $\mathbb{M} = \mathbb{M}(\boldsymbol{x}; \boldsymbol{z})$, where $(\boldsymbol{x}, \boldsymbol{z}) \in \mathbb{R}^m \times \mathbb{R}^k$ in cases (2)-(3).

The elements z_i of the parameter vector $\boldsymbol{z} = (z_1, z_2, \ldots, z_k)$ are sometimes named weights of the machine (in the case of neural networks, for example). The training stage is the process of seeking for the best $\boldsymbol{z}_{opt} \in \mathbb{R}^k$ such that $\mathbb{M}(\boldsymbol{x}; \boldsymbol{z}_{opt}) \simeq f(\boldsymbol{x})$, where f is the functional dependency in items (1)-(3). Something analogous can be defined by case (4).

A learning process is implemented by choosing the learning machine and by defining an appropriate training strategy. Considering the viewpoint above, the learning process can be viewed as the process of seeking for an appropriate function, from a given set of functions encapsulated by the machine model. Let us formalize this idea.

Our discussion focus firstly on the cases (2)-(3) for simplicity. Once we have chosen the machine model $\mathbb{M} = \mathbb{M}(\boldsymbol{x}; \boldsymbol{z})$, we can implement the learning process by minimizing an error function:

$$Err(\boldsymbol{z}; \mathbb{D}) = \frac{1}{|\mathbb{D}|} \sum_{i=1}^{|\mathbb{D}|} L(\mathbb{M}(\boldsymbol{x}_i; \boldsymbol{z}) - \boldsymbol{d}_i), \tag{8.2}$$

with respect to $\boldsymbol{z}$, where $\boldsymbol{d}_i$ is a generic vector representing the desired outputs for cases (2)-(3), and L is a suitable metric or similarity measure in the output space. In practice, the problem may be ill-posed and we need some regularization terms, represented by functions $R_\tau = R_\tau(\boldsymbol{z}; \mathbb{D})$, $\tau = 1, 2, \ldots, K$ as well as constraints in the form $g_j(\boldsymbol{z}; \mathbb{D}) = k_j$, $j = 1, 2, \ldots N$. So, we build the Loss function:

$$Loss(\boldsymbol{z}; \mathbb{D}) = Err(\boldsymbol{z}; \mathbb{D}) + \sum_{\tau=1}^{K} \varsigma_\tau R_\tau(\boldsymbol{z}; \mathbb{D}), \tag{8.3}$$

and the training process output is the solution of the problem:

$$\min_{\boldsymbol{z} \in \mathbb{R}^k} Loss(\boldsymbol{z}; \mathbb{D}) = \min_{\boldsymbol{z} \in \mathbb{R}^k} \left[Err(\boldsymbol{z}; \mathbb{D}) + \sum_{\tau=1}^{K} \varsigma_\tau R_\tau(\boldsymbol{z}; \mathbb{D}) \right], \tag{8.4}$$

subject to:

$$g_j(\boldsymbol{z}; \mathbb{D}) = k_j, \ j = 1, 2, \ldots N. \tag{8.5}$$

To deal with the case (4) of the introduction of this Chapter, we develop the presentation in discrete probability spaces following [93], page 146, and properties stated in [94]. So, returning to the data model of section 7.1, the samples in the set $\mathbb{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\} \subset \mathbb{R}^m$ are assumed to be drawn independently from a true but unknown data distribution $p_{data}(\boldsymbol{x})$. In this case, the machine model $\mathbb{M} = \mathbb{M}(\boldsymbol{x}; \boldsymbol{z})$ represents a parametric family of probability functions over the dataset $\mathbb{D}$. To simplify further expressions, we write $\mathbb{M}(\boldsymbol{x}; \boldsymbol{z}) = p_{model}(\boldsymbol{x}; \boldsymbol{z})$.

So, the goal is to obtain a machine $\mathbb{M}$ such that $\mathbb{M}(\boldsymbol{x}_i;\boldsymbol{z}) = p_{model}(\boldsymbol{x}_i;\boldsymbol{z}) \simeq p_{data}(\boldsymbol{x}_i)$, $i = 1, 2, \ldots, M$. Now, the error function is replaced by the cross entropy of p_{data} and p_{model}:

$$H(\boldsymbol{z}) = -\sum_{i=1}^{M} p_{data}(\boldsymbol{x}_i;\boldsymbol{z}) \log p_{model}(\boldsymbol{x}_i;\boldsymbol{z}) \equiv E_{\sim p_{data}}(\log p_{model}(\boldsymbol{x};\boldsymbol{z})), \tag{8.6}$$

and the optimization problem is substituted by the maximum likelihood estimation problem, given by:

$$\boldsymbol{z}_{opt} = \arg\max_{\boldsymbol{z}\in\mathbb{R}^k} E_{\sim p_{data}}(\log p_{model}(\boldsymbol{x};\boldsymbol{z})). \tag{8.7}$$

One reason to consider the maximum likelihood estimation comes from the fact that, the Kullback-Leibler divergence from p_{data} to p_{model}, given by:

$$D_{KL}(p_{data}||p_{model}) = \sum_{i=1}^{M} p_{data}(\boldsymbol{x}_i) \log\left(\frac{p_{data}(\boldsymbol{x}_i)}{p_{model}(\boldsymbol{x}_i;\boldsymbol{z})}\right), \tag{8.8}$$

measures how p_{model} is different from the reference probability p_{data} [93, 94]. We can re-write expression (8.8) as:

$$D_{KL}(p_{data}||p_{model}) = \sum_{i=1}^{M} p_{data}(\boldsymbol{x}_i) \log p_{data}(\boldsymbol{x}_i) - \sum_{i=1}^{M} p_{data}(\boldsymbol{x}_i) \log p_{model}(\boldsymbol{x}_i;\boldsymbol{z}). \tag{8.9}$$

If we consider the problem:

$$\boldsymbol{z}^{\star}_{opt} = \arg\max_{\boldsymbol{z}\in\mathbb{R}^k} D_{KL}(p_{data}||p_{model}), \tag{8.10}$$

we shall notice that the first term in expression (8.9) does not depend on the model parameter $\boldsymbol{z}$. Consequently, optimization problem (8.10) with respect to $\boldsymbol{z}$ can be simplified to:

$$\boldsymbol{z}^{\star}_{opt} = \arg\max_{\boldsymbol{z}\in\mathbb{R}^k} \left(-\sum_{i=1}^{M} p_{data}(\boldsymbol{x}_i) \log p_{model}(\boldsymbol{x}_i;\boldsymbol{z})\right). \tag{8.11}$$

By comparing problems (8.7) and (8.11) it is clear that they are equivalent and, consequently, $\boldsymbol{z}_{opt} = \boldsymbol{z}^{\star}_{opt}$. Likewise in the expressions (8.4)-(8.5), we can add regularization terms and constraints to the problem (8.7).

8.2 What Do Machines Learn?

The problems of minimizing the Loss functional (8.4), subject to the constraints (8.5), and maximum likelihood estimation (equation (8.7)), on the basis of empirical data $\mathbb{D}$, formalizes four basic machine learning problems, that can be written by rephrasing items (1)-(4) above as follows:

1. Data Synthesis (generative model): find a function $f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$, that reproduce the data;

2. Pattern Recognition: After training, the machine is able to carries out the correct classification of data samples out of the training set $\mathbb{D}$.

3. Regression Estimation: Estimating the functional dependence based on the empirical data; that means, estimate $f : \mathbb{R}^m \longrightarrow \mathbb{R}$ such that $f(\mathbf{x}_i) = y_i$, $i = 1, 2, \ldots, M$.

4. Density Estimation: The problem of estimating $p_{model}(\boldsymbol{x}_i; \boldsymbol{z}_{opt})$ through p_{data} by solving problem (8.7) or, equivalently, by minimizing Kullback–Leibler divergence respect to the parameter vector $\boldsymbol{z}$, given by the problem (8.10).

In case (1) we say that the machine have learned a pattern, in the sense that, after training, the machine is able to reproduce samples of the manifold $\mathcal{M}$. In the case (2) we say that the machine learned to recognize a pattern, because it is able to classify each sample $\boldsymbol{x} \in \mathcal{M}$ according to the corresponding category. In items (3) the machine learns a functional dependence while in the case (4) it have learned a stochastic dependence.

Also, the scenarios (1) and (2) above helps to describe three types of machine learning:

- Supervised learning (Case 2): The input set encompasses examples and target responses;

- Unsupervised learning (Case 1): The machine learns patterns from untagged data;

- Semi-Supervised Learning: Only a subset of the input set $\mathbb{D}$ contains samples that are accompanied with the desired response.

Moreover, we shall consider two more types of machine learning strategies:

- Reinforcement learning: The algorithm gets told when the answer is wrong, but does not get told how to correct it. There is a *monitor* that scores the answer, but does not suggest improvements [3].
- Evolutionary learning: Learning algorithms based on biological evolution.

8.3 Types of Learning Machines

What kind of machine are we considering in this monograph? The Chapter 11 will describe the neural networks, that compose the first machine learning model used in this monograph. Formally, these models can be cast into the computational model formalized by the circuit theory. The statistical learning techniques are described in Chapter 10. They allows to desing statistical classiifiers, for instance.

8.3.1 Circuit Theory

Regarding the circuit theory, we start with a didactic development found in [45], page 129. So, we may say that a circuit is made up *wires* and gates. The wires carry information around and the gates (functions) perform simple computational tasks. The simplest gates are the logic operators presented on Chapter 3 and the corresponding functions are defined by their truth Tables (section 3.2). The Figure 8.1 shows the elementary gates and their graphical representation. Also, we can add to the set of elementary gates the **identity** gate (just a wire) and the **FANOUT** one, which replaces a bit with two copies of itself.

In Figure 8.2 it is pictured an example of a circuit that outputs the result of the Boolean expression $x \oplus y = XOR\,(x, y)$ (see Table 3.1.(d)) together with a carry bit set to 1 if x and y are both 1, or 0 otherwise (check it!).

Formally, let G be a set of gates that map several bits to one bit. For each $n, m \geq 0$, a circuit $C^{G}_{(n,m)}$, or simply $C_{n,m}$, if there is no ambiguity over the set G, is a directed, acyclic graph with a list of n input nodes (with no incoming edges), a list of m output nodes (with no outgoing edges), and a gate in G computing each non-input/non-output node.

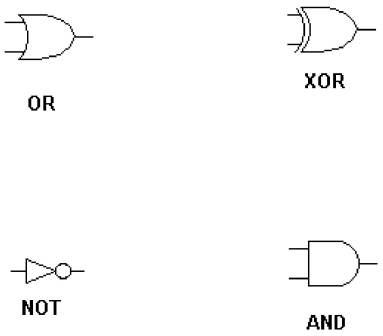


Figure 8.1: Basic Gates.

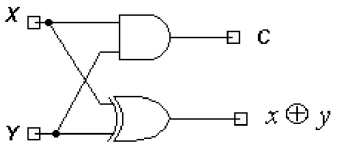


Figure 8.2: Circuit example.

Given a binary input string $(x_1, x_2, ..., x_n) \in \{0,1\}^n$, we label each input node x_i or $\sim x_i$ $(NOT\,(x_i))$. For every other node v with n predecessors $(y_1, y_2, ..., y_n)$, we recursively assign a value $g\,(y_1, y_2, ..., y_n)$, where g is the gate that calculates node v. Given an input $\boldsymbol{x} \in \{0,1\}^n$, the circuit $C^G_{(n,m)}$ outputs the value $C^G_{(n,m)}\,(\boldsymbol{x}) \in \{0,1\}^m$ given by the list of output nodes.

The **size of a circuit** $C_{n,m}$ is its number of nodes and the **depth of** C_n is the length of the longest path from any input node to an output node.

We say that a circuit $C^G_{(n,m)}$ computes a Boolean function $F : \{0,1\}^n \longrightarrow \{0,1\}^m$ if $C^G_{(n,m)}\,(\boldsymbol{x}) = F\,(\boldsymbol{x})$, for all $\boldsymbol{x} \in \{0,1\}^n$. A **circuit family** is a list of circuits $C = (C_{1,m}, C_{2,m}, ..., C_{n,m}, ...)$ where $C_{i,m}$ has i binary inputs and m binary outputs. The family C computes a family of Boolean functions $(f_1, f_2, ..., f_n, ...)$, where f_i is the Boolean function computed by circuit $C_{i,m}$.

We say that the family C has size complexity $s\,(n)$ and depth complexity $d\,(n)$

if for all $n \geq 0$ circuit C_n has size at most $s(n)$ and depth at most $d(n)$. Size and depth are important complexity descriptions of circuits that respectively characterize the *computational resources* and the number of steps needed to compute a family of Boolean functions. Important classes of circuits are the following ones.

Let G be the set of elementary gates represented in Figure 8.1 and $CG(n,m)$ the class of functions that can be computed by these circuits. We also consider the set of Boolean functions $\mathcal{F}(n,m) = \{F : \{0,1\}^n \longrightarrow \{0,1\}^m\}$.

Theorem 1.8: Given a Boolean expression $F : \{0,1\}^n \longrightarrow \{0,1\}^m$, there is a circuit $C^G_{(n,m)}$ such that $C^G_{(n,m)}(x_1, x_2, ..., x_n) = F(x_1, x_2, ..., x_n)$ for all $(x_1, x_2, ..., x_n) \in B^n$. As a consequence, it is straightforward that $CG(n,m) = \mathcal{F}(n,m)$.

Proof: We are going to show the demonstration for $m = 1$ and the geral case is let as exercise. The demonstration is performed using induction in n. Therefore, following the main steps of mathematical induction we shoud:

- Show that:

  (a) The property is true for $n = 1$;

- Next, assume that:

  (b.1) Hypothesis: Property is true for $n = k$;

- Then, we should demonstrate:

  (b.2) Thesis: Property is true for $n = k + 1$.

Using the basic gates of Figure 8.1 it is more or less straightforward to show that $CG(1,1) = \mathcal{F}(1,1)$, which solves the item (a).

To proof the thesis in item (b.2) the trick is to notice that any function $F : \{0,1\}^{k+1} \longrightarrow \{0,1\}$ can be written as:

$$F(x_0, x_1, ..., x_k) = (F_0 \wedge \sim x_0)\, XOR\, (F_1 \wedge x_0) \tag{8.12}$$

where:

$$F_0(x_1, x_2, \ldots, x_k) = F(0, x_1, x_2, \ldots, x_k), \tag{8.13}$$

$$F_1(x_1, x_2, \ldots, x_k) = F(1, x_1, x_2, \ldots, x_k), \tag{8.14}$$

Now, using the induction hypothesis, we can affirm that there are circuits $\widetilde{C}^{G}_{(k,1)}$ and $C^{G}_{(k,1)}$ that computes F_0 and F_1, respectively. Using this fact, to complete the demonstration of the thesis (item (b.2)), it is just a matter of noticing that expression (8.12) can be converted into a circuit $C^{G}_{(k+1,1)}$, as shown in Figure 8.3.

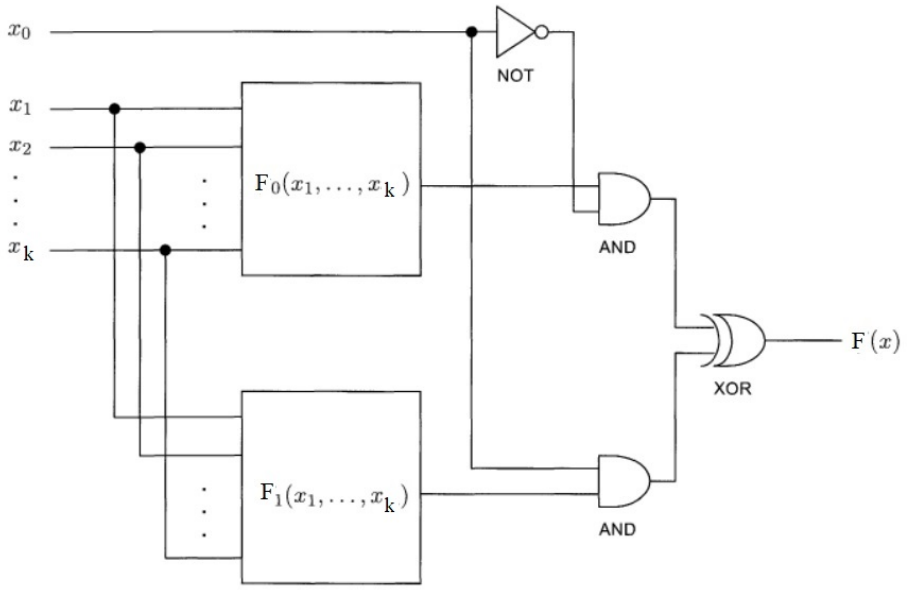


Figure 8.3: Circuit $C^{G}_{(n+1,1)}$ representing expression (8.12) to compute an arbitrary function $F : \{0,1\}^{k+1} \longrightarrow \{0,1\}$.

Therefore, from the fact that the property is true for $n = 1$ and items (b.1) and (b.2), we can affirm that, given a Boolean function $F : \{0,1\}^{n} \longrightarrow \{0,1\}$, there is a circuit $C^{G}_{(n,1)}$ such that $C^{G}_{(n,1)}(x_1, x_2, ..., x_n) = F(x_1, x_2, ..., x_n)$ for all $(x_1, x_2, ..., x_n) \in B^n$. Therefore $\mathcal{F}(n,1) \subset CG(n,1)$. Conversely, by the definition of circuits in the set $CG(n,1)$ it is obvious that $CG(n,1) \subset \mathcal{F}(n,1)$. So, $CG(n,1) = \mathcal{F}(n,1)$ which completes the proof for $m = 1$.

8.3.2 Threshold Circuits

A weighted threshold gate (or simply, a threshold gate) with threshold $\Delta \in \mathbb{Q}$ and a vector of weights $\mathbf{w} = (w_1, w_2, ..., w_n) \subset \mathbb{Q}^n$ is a Boolean function denoted by $Th^{n,\Delta}_{\mathbf{w}}$ and defined as follows [95]:

$$Th_{\mathbf{w}}^{n,\Delta} : \{0,1\}^n \rightarrow \{0,1\},$$

$$Th_{\mathbf{w}}^{n,\Delta}(x_1, x_2, ..., x_n) = 1, \quad if \quad \sum_{i=1}^{n} w_i x_i \geq \Delta, \tag{8.15}$$

$$Th_{\mathbf{w}}^{n,\Delta}(x_1, x_2, ..., x_n) = 0, \quad otherwise. \tag{8.16}$$

A threshold circuit, denoted by $TC_{(\boldsymbol{w},m)}^{n,\Delta}$, is a circuit over the set of threshold gates, with input $\mathbf{x} \in B^n$ and output $\mathbf{y} \in B^m$. Hence, from the definition of threshold gates (expression (8.15)-(8.16)) a threshold circuit is also a function $TC_{(\boldsymbol{w},m)}^{n,\Delta} : \{0,1\}^n \longrightarrow \{0,1\}^m$. The class of functions that can be computed by threshold circuits $TC_{(\boldsymbol{w},m)}^{n,\Delta}$ will be denoted as $\mathcal{TF}_{(\boldsymbol{w},m)}^{n,\Delta}$. An important theorem about threshold circuits is the following one:

Theorem 4 *Given a Boolean expression $F : B^n \longrightarrow B^m$, where $B = \{0,1\}$, there is a weighted threshold circuit $TC_{(\boldsymbol{w},m)}^{n,\Delta}$ such that $TC_{(\boldsymbol{w},m)}^{n,\Delta}(\mathbf{x}) = F(\mathbf{x})$ for all $\mathbf{x} \in B^n$.*

Proof: To demonstration this theorem it is just a matter of showing that the basic gates of Figure 8.1 can be computed by threshold circuits. Hence, the proof is a straightforward consequence of Theorem 1.8.

8.4 Optimizing the Loss Function

The process of seeking for the solution of the problem (8.4)-(8.5) happens during the training process. So, we consider a training set $\mathbb{D}_{tr}$, that might be a subset of the whole database (see section 8.5). Hence, for training, the expressions (8.4)-(8.5) are re-written by substituting $\mathbb{D}$ to $\mathbb{D}_{tr}$. The new expressions are used to compose the Lagrangian to be optimized. According to section 5.9, this problem can be solved through the following steps:

1. Build the Lagrangian function:

$$\mathcal{L}(z_1, z_2, \ldots, z_k, \lambda_1, \lambda_2, \ldots, \lambda_N) = Loss(\boldsymbol{z}; \mathbb{D}_{tr}) - \sum_{j=1}^{N} \lambda_j \left(g_j(\boldsymbol{z}; \mathbb{D}_{tr}) - k_j\right), \tag{8.17}$$

where the parameters λ_j are called the Lagrange multipliers, that are assembled in the array $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_N)$.

2. Solve $\nabla\mathcal{L} = (0, 0, \ldots, 0)$; that means:

$$\frac{\partial\mathcal{L}}{\partial z_i}(z_1, z_2, \ldots, z_k, \lambda_1, \lambda_2, \ldots, \lambda_N) = 0, \quad i = 1, 2, \ldots, k, \tag{8.18}$$

$$\frac{\partial\mathcal{L}}{\partial \lambda_j}(z_1, z_2, \ldots, z_k, \lambda_1, \lambda_2, \ldots, \lambda_N) = 0, \quad j = 1, 2 \ldots, N. \tag{8.19}$$

3. Take the solutions $\left(\overline{\boldsymbol{z}}, \overline{\boldsymbol{\lambda}}\right)$ of system (8.18)-(8.19) and set $\boldsymbol{\lambda} = \overline{\boldsymbol{\lambda}}$ into expression (8.17) to obtain the function $F(\boldsymbol{z}) \equiv \mathcal{L}\left(\boldsymbol{z}; \overline{\boldsymbol{\lambda}}\right)$

4. Calculate the Hessian $H_{\mathbf{z}}F(\overline{\boldsymbol{z}})$:

$$\left[H_{\mathbf{z}}F(\overline{\boldsymbol{z}})\right]_{ij} = \frac{\partial^2 F}{\partial z_i \partial z_j}(\overline{\boldsymbol{z}}), \quad 1 \leq i, j \leq N, \tag{8.20}$$

5. Compute the eigenvalues of $H_{\mathbf{z}}F(\overline{\boldsymbol{z}})$ and use them to classify $\overline{\boldsymbol{z}}$ according to the cases (1)-(4) of section 5.8.

However, in machine learning applications, in general, expressions (8.18)-(8.19) do not have exact solutions. Therefore, we shoud adapt the steepest descent procedure (Algorithm 1) in order to build an iterative algorithm to approximate the solution. Other approaches depends on application of penalty methods [96] that are out of the scope of this monograph.

For simplicity, in this presentation we assume that the feasible region $\mathcal{R} \subset \mathbb{R}^k$, given by the solution of equations (8.5) is known. Consequently, we can restrict the Loss function to $\mathcal{R}$, obtaining $\mathcal{L}_{\mathcal{R}}(\cdot, \mathbb{D}_{tr}) : \mathcal{R} \longrightarrow \mathbb{R}$. The Algorithm 3 summarizes the numerical optimization process.

The Loss function in the Lagrangian (8.17) encompasses the error function (8.2) that uses the whole data set in each update of the parameter vector $\boldsymbol{z}$. Such strategy, named batch mode, has disadvantage in terms of computational cost if M is too high since the complexity to update the parameter vector $\boldsymbol{z}$ in expression (8.21) is $O(M)$. To avoid this problem we shall notice that the gradient of the error function (8.2) is an expectation that could be estimated using a small set of samples, called minibatch. That is the key idea of stochastic gradient.

Algorithmus 3 Training Through Optimization of Lagrangian (8.17)

1: Input: Training set $\mathbb{D}_{tr}$, $\eta_1, \eta_2, \varepsilon > 0$,

2: Initialization: $\boldsymbol{z}_0 \in \mathcal{R}$, $\boldsymbol{\lambda}_0 \in \mathbb{R}^N$, $t = 0$;.

3: **while** $(\|\nabla_z \mathcal{L}_{\mathcal{R}}(\mathbf{z}_t, \boldsymbol{\lambda}_t, \mathbb{D}_{tr})\| \geq \varepsilon)$ **do**

4: Learning rule. Update parameter vector:

$$\boldsymbol{z}_{t+1} = \boldsymbol{z}_t - \eta_1 \nabla_z \mathcal{L}_{\mathcal{R}}(\mathbf{z}_t, \boldsymbol{\lambda}_t, \mathbb{D}_{tr}), \tag{8.21}$$

5: Verify if $\boldsymbol{z}_{t+1} \in \mathcal{R}$. If no, reduce η_1 and return to previous step;

6: Update Lagrange multipliers vector:

$$\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t - \eta_2 \nabla_\lambda \mathcal{L}_{\mathcal{R}}(\mathbf{z}_{t+1}, \boldsymbol{\lambda}_t, \mathbb{D}_{tr}), \tag{8.22}$$

7: $t \leftarrow t + 1$,

end while

8: $\left(\boldsymbol{z}^\star_{opt}, \lambda^\star_{opt}\right) \leftarrow (\mathbf{z}_t, \boldsymbol{\lambda}_t)$,

9: Output: Solution $\left(\boldsymbol{z}^\star_{opt}, \lambda^\star_{opt}\right)$

Formally, in each step of the main loop in Algorithm 3 we sample a minibatch $\mathbb{B} = \{\widehat{\mathbf{x}}_1, \widehat{\mathbf{x}}_2, \ldots, \widehat{\mathbf{x}}_\varsigma\} \subset \mathbb{D}_{tr}$, usually using an uniform distribution, and restrict the Loss function to the data points in $\mathbb{B}$.

For simplicity, we do not consider restrictions in this case. So, the Lagrangian is the function (8.3) with $\mathbb{D}$ substituted by the batch $\mathbb{B}$; that means:

$$\mathcal{L}(\boldsymbol{z}; \mathbb{B}) \equiv \frac{1}{|\mathbb{B}|} \sum_{i=1}^{|\mathbb{B}|} L\left(\mathbb{M}(\boldsymbol{x}_i; \boldsymbol{z}) - \boldsymbol{d}_i\right) + \sum_{\tau=1}^{K} \varsigma_\tau R_\tau(\boldsymbol{z}; \mathbb{B}), \tag{8.23}$$

where R_τ and $\boldsymbol{d}_i$ follow the previous definitions.

The optimization of the Lagrangian is performed through the stochastic gradient descent (SGD) procedure, given by the Algorithm 4. The first term in expression (8.25), is an approximation of the gradient of the error function Err, restricted to the minibatch $\mathbb{B}$:

$$\nabla_z \left[Err(\boldsymbol{z}; \mathbb{B})\right] = \nabla_z \left[\frac{1}{|\mathbb{B}|} \sum_{i=1}^{|\mathbb{B}|} L\left(\mathbb{M}(\boldsymbol{x}_i; \boldsymbol{z}) - \boldsymbol{d}_i\right)\right]. \tag{8.24}$$

We shall notice that the stop criterion in line 4 of Algorithm 4 uses the Lagrangian $\mathcal{L}$ computed using the minibatch set $\mathbb{B}$ (expression (8.3) with $\mathbb{D}$ replaced to $\mathbb{B}$). Libraries like TensorFlow and Keras (see [97] for a list of machine learning libraries) have implemented solutions based on SGD techniques, like the Algorithm 4. Also, each turn that the whole set $\mathbb{D}_{tr}$ passes the Algorithm 3 (or Algorithm 4) is called epoch. The strategy to initialize the parameter vector (definition of $\boldsymbol{z}_0$) in Algorithms 3 and 4 depends on the kind of learning machine (see Chapter 11). The initialization of the Lagrangian multipliers in Algorithm 3 depends also to mathematical considerations about the constraints involved.

The gradient (or stochastic gradient) descent is the driving force behind the learning rule in expressions (8.21) and (8.26) in the training that is an optimization process: the parameter values are adapting in order to minimize the Loss (or Lagrangian in Algorithm 8.17) function. However, with such procedure there are no guarantees that we end up at a global minimum. In fact, we my stop in a point $\boldsymbol{z}$ in the parameter space that is a local minimum; that means, just lower than those close to it (see section 5.8). To mitigate this problem we can try out several different initializations to obtain different networks. Then, we choose the one that offer the minimum error. Other schemes, like momentum in the case of neural network machines, will be discussed in Chapter 11.

Algorithmus 4 Minibatch Stochastic Gradient Descent Training

Input: Training set $\mathbb{D}_{tr}$, minibatch size ς, $\eta, \varepsilon > 0$,

Initialization: $\boldsymbol{z}_0 \in \mathbb{R}^k$, $t = 0$,

$\boldsymbol{p} \leftarrow \mathbf{z}_0$;,

while $(\|\nabla_z \mathcal{L}(\boldsymbol{p}; \mathbb{B})\| \geq \varepsilon)$ **do**

Sample minibatch $\mathbb{B} = \{\widehat{\mathbf{x}}_1, \widehat{\mathbf{x}}_2, \ldots, \widehat{\mathbf{x}}_\varsigma\} \subset \mathbb{D}_{tr}$,

Build the Loss function by substituting $\mathbb{D}$ to $\mathbb{B}$ in expression (8.3). Set the Lagrangian as the obtained Loss function, given by equation (8.23).

Compute gradient:

$$\nabla_z \mathcal{L}(\boldsymbol{p}; \mathbb{B}) = \nabla_z \left[\frac{1}{|\mathbb{B}|} \sum_{i=1}^{|\mathbb{B}|} L\left(\mathbb{M}(\boldsymbol{x}_i; \boldsymbol{p}) - \boldsymbol{d}_i\right) \right] + \nabla_z \left[\sum_{\tau=1}^{K} \varsigma_\tau R_\tau(\boldsymbol{p}; \mathbb{B}) \right] \tag{8.25}$$

Learning rule. Update parameter vector:

$$\boldsymbol{z}_{t+1} = \boldsymbol{z}_t - \eta \nabla_z \mathcal{L}(\boldsymbol{p}; \mathbb{B}), \tag{8.26}$$

$\boldsymbol{p} \leftarrow \boldsymbol{z}_{t+1}$;

$t \leftarrow t + 1$,

end while

$\boldsymbol{z}^\star_{opt} \leftarrow \boldsymbol{p}.$,

Output: Solution $\boldsymbol{z}^\star_{opt}$

8.5 Train, Validation, and Test Stages

Let us consider the case of classification problem where we have a labeled data set:

$$\mathbb{D} = \{(\mathbf{x}_1, l_1), (\mathbf{x}_2, l_2), \ldots, (\mathbf{x}_M, l_M)\} \subset \mathbb{R}^m \times \{0, 1\}, \tag{8.27}$$

Before proceeding, we must divide the dataset $\mathbb{D}$ in expression (8.27) into three disjoint subsets: training $\mathbb{D}_{tr}$, validation $\mathbb{D}_{val}$, and test $\mathbb{D}_{te}$ sets. This process is represented in Figure 8.4. To implement this process we randomly choose data points in $\mathbb{D}$, with an uniform distribution, using a proportion like $50 : 25 : 25$ or $60 : 20 : 20$, for $\mathbb{D}_{tr}$, $\mathbb{D}_{val}$, and $\mathbb{D}_{te}$, respectively [3].

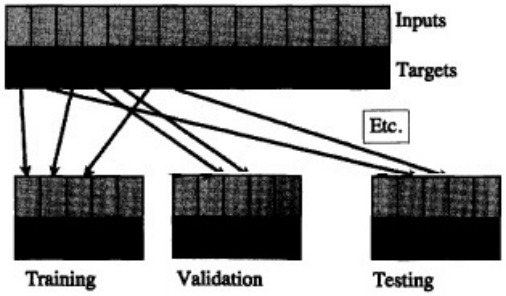


Figure 8.4: The labeled dataset is subdivided into three disjoint subsets for training testing, and validation of the learning machine (Source [3]).

Now, the training process can be performed using Algorithm 3 or 4 depending if we are going to use batch or minibatch training mode, respectively. Once we have completed one epoch in the training stage, we can use the validation set to analyze the behavior of the Loss function over it. The idea is to check if problems, like the phenomenon of overfitting, are happening, as we will see in section 8.7.

If everything is going well; that means, if the Loss is decreasing along the epochs for both the sets $\mathbb{D}_{tr}$ and $\mathbb{D}_{val}$, then it is expected that the stopping criterion in the $while$ loop of Algorithms 3 or 4 will be eventually achieved and, consequently, we will stop the training (and validation) steps. Hence, we can go to the test stage using the data points in the set $\mathbb{D}_{te}$.

In the test stage, we keep the parameter vector $\boldsymbol{z}^{\star}_{opt}$ unchanged! We just pass the data samples in the set $\mathbb{D}_{te}$ to the machine and compute some efficiency measure to check the automatic classification (see section 8.6).

The above process is a naive one because: (a) It is subject to local minima; (b) The partition of the dataset into training $\mathbb{D}_{tr}$, validation $\mathbb{D}_{val}$, and test $\mathbb{D}_{te}$ sets is performed through a random process. Consequently, we must use more than one machine instantiation to smooth performance variations through some kind of average.

Obviously, we can use several different initializations for different partitions to train different networks. Then, we can choose the one that offer the minimum error. Other possibility, statistically more reliable, is to perform leave-one-out, multi-fold cross-validation. In this strategy, the dataset $\mathbb{D} - \mathbb{D}_{te}$ is randomly partitioned into K subsets. Then, we chose one subset to be used as a validation set and the learning machine is trained on all of the remaining ones, as shown in Figure 8.5. In the next step, the same K subsets are taken but a different subset is left out for validation and the other ones are used to train another machine. We repeat this process until all the K subsets are used for validation. After this process we obtain K machines $\mathbb{M}_i$, $i = 1, 2, ..., K$ where each one was trained and validated using different sets with different initializations. Finally, the machine that yields the lowest validation error is used in the test stage, with the corresponding test set, as represented in Figure 8.5.

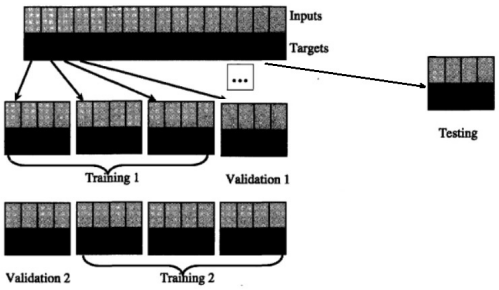


Figure 8.5: K-fold cross-validation algorithm representation (Source [3]).

8.6 Performance Measures

Before applying a classifier to solve day life tasks, we must consider evaluation measures to analyze its efficiency. For this goal, we use the confusion matrix

(C) to $N - class$ classification problems, where the entry $C_{i,j}$ is the number of observations known to be in group i and predicted to be in group j, $i, j = 1, 2, \ldots, N$ [98]. So, we shall define the following new quantities [99]:

- $TP_i \equiv C_{i,i}$ is equal to the number of observations known to be in group i and predicted to be in the correct class;
- $FN_i \equiv \sum_{j=1, j\neq i}^{N} C_{i,j}$, is equal to the number of observations known to be in group i but misclassified in some class $j \neq i$; that means, false negative of class i;
- $FP_i \equiv \sum_{k=1, k\neq i}^{N} C_{k,i}$, is equal to the number of observations known to be in group $k \neq i$ but misclassified in class i; that means, false positive of class i.

With these definitions, we can compute the following evaluation measures [100]:

Macro Precision of class i: Denoted by $P(i)$, and computed by:

$$P(i) = \frac{TP_i}{TP_i + FP_i}. \tag{8.28}$$

Macro Recall of class i: Denoted by $R(i)$, computed by:

$$R(i) = \frac{TP_i}{TP_i + FN_i}. \tag{8.29}$$

Macro Precision of the Classifier:

$$P_{macro} = \frac{\sum_{i=1}^{N} P(i)}{N}. \tag{8.30}$$

Macro Recall of the Classifier:

$$R_{macro} = \frac{\sum_{i=1}^{N} R(i)}{N}. \tag{8.31}$$

Macro F1-Score:

$$F1_{Score} = 2\frac{P_{macro} \cdot R_{macro}}{P_{macro} + R_{macro}}, \tag{8.32}$$

Multi-Class Accuracy or Recognition Rate:

$$Accuracy = \frac{1}{M}\sum_{i=1}^{N} TP_i, \tag{8.33}$$

where M is the number of samples used.

Averaged Error Rate:

$$AER = \frac{\sum_{i,j=1, i\neq j}^{N} C_{i,j}}{M}. \tag{8.34}$$

From expression (8.28), we can say that the precision of class i is the fraction of classifications delivered by the learner ($TP_i + FP_i$) that really belong to class i. Recall in expression (8.29) is the fraction between the events of class i detected and the number of elements of this class ($TP_i + FN_i$). The multi-class accuracy (expression (8.33)) and average error rate (equation (8.34)) are complementary, in the sense that:

$$Accuracy + AER = 1. \tag{8.35}$$

In words, accuracy focus only on the successful predictions while AER captures information about how well the model handles negative results (misclassifications).

8.7 Curse of Dimensionality, Overffiting, and Over-training

In the development of previous sections, important points are the number M of samples in the database $\mathbb{D}$, the original data space dimension m and the dimension n of the data manifold $\mathcal{M}$. In section 7.4, we discuss the curse of dimensionality problem considering the number M versus the dimension n and, from that discussion, we have concluded that M grows exponentially with the value of n (curse of dimensionality) to catch the distribution of samples in the focused phenomenon. Hence, we must consider dimensionality reduction techniques to reduce further pattern recognition steps.

To clarify the way that the curse of dimensionality affects the training stage, we shall consider a parametric representation $f : \mathbb{R}^n \longrightarrow \mathcal{M}$ of the

data manifold yielded by a generative machine learning model (section 8.2). Hence, the samples $\mathbb{D}$ could be represented through the image set $f^{-1}(\mathbb{D}) = \{f^{-1}(\mathbf{x}_1), f^{-1}(\mathbf{x}_2), \ldots, f^{-1}(\mathbf{x}_M)\} \subset \mathbb{R}^n$. Such representation is acceptable from the mathematical viewpoint but, for pattern recognition purposes, it present problems. For example, the distortions in the pair-wise distance map $d(f^{-1}(\mathbf{x}_i), f^{-1}(\mathbf{x}_j))$ may cause problems for a classifier to distinguish samples from different classes.

A simpler version of such problem is obtained by considering a representation based on the tangent space of the hypersurface corresponding to the map $\mathbf{x} = f(\xi_1, \xi_2, \ldots, \xi_n)$, in an appropriate point $\overline{\mathbf{p}} \in \mathcal{M}$, like in Figure 8.6.

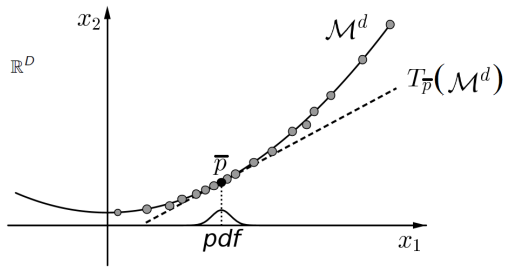


Figure 8.6: Tangent space at a point $\overline{p}$ of the data manifold $\mathcal{M}$.

Such representation could be written as:

$$\mathbf{x}_i \simeq \overline{\mathbf{p}} + \sum_{j=1}^{n} w_{i,j} \frac{\partial f}{\partial \xi_j}(\overline{\mathbf{p}}). \tag{8.36}$$

where the coefficient vectors $\boldsymbol{w}_i = (w_{i1}, w_{i2}, \ldots, w_{in}) \in \mathbb{R}^n$ should be properly computed. The PCA technique is a way to compute such representation as we will see in Chapter 10.

Anyway, as usual in pattern recognition applications, let us assume that the patterns of interest can be properly represented using the coefficient vectors $\boldsymbol{w}_i$, $i = 1, 2, ..., M$, in expression (8.36), also called feature vectors. Each direction:

$$\overline{\mathbf{q}}_j \equiv \frac{\partial f}{\partial \xi_j}, \quad j = 1, 2, .., n, \tag{8.37}$$

is named a component of the representation. Expression (8.36) can be considered

a projection of sample $\mathbf{x}_i$ into the affine space:

$$Aff_{\overline{\mathbf{p}}}(\mathcal{M}) = \left\{ \mathbf{x} \in \mathbb{R}^m;\ \mathbf{x} = \overline{\mathbf{p}} + \sum_{j=1}^{n} w_j \frac{\partial f}{\partial \xi_j}(\overline{\mathbf{p}}) \right\}, \tag{8.38}$$

which is the tangent hyperplane to $\mathcal{M}$ that contains $\overline{\mathbf{p}}$.

However, for orthogonal projections, two distinct points $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{D}$ with feature vectors $\boldsymbol{w}_i$ and $\boldsymbol{w}_j$ satisfies $d(\boldsymbol{w}_i, \boldsymbol{w}_j) \leq d(\boldsymbol{x}_i, \boldsymbol{x}_j)$, which is a side effect of this projection that may impairs pattern recognition tasks, like classification, if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ belong to different classes.

Moreover, from the viewpoint of a classifier, there is a trade-off between pattern recognition and the number of components $\overline{\mathbf{q}}_j$ to project samples. Specifically, data suffers from noise and redundancy but the representation given by expression (8.36) does not take into account procedures to reduce these defects, and, consequently, using all the n components in $\boldsymbol{w}_i$ may be not efficient. Let us understand this problem step by step.

If we use few components of the feature space, say $\frac{\partial f}{\partial \xi_1}, \frac{\partial f}{\partial \xi_2}, \ldots, \frac{\partial f}{\partial \xi_{n'}}$, with $n' \ll n$, we are projecting the data into a very low dimensional space. Consequently, we could lose too many details to allow the classifier to properly distinguish sample groups. So, we must increase the number of components aiming to improve the classifier performance. Hence, we must take $\frac{\partial f}{\partial \xi_1}, \frac{\partial f}{\partial \xi_2}, \cdots, \frac{\partial f}{\partial \xi_{n'+b'}}$ components, with $b' > 1$, in order to get a new subspace spanned by the $n' + b'$ components. However, this process also increases the complexity of the decision boundary and more samples may be necessary for improving the training process of the classifier (curse of dimensionality again). Moreover, we can add noise and redundancy in the data representation when increasing the subspace dimension. Now, we may be facing overfitting; that means, subspace dimension becomes larger enough so as to be able to learn the underlying differences among the within-class data and/or unimportant details (like noise) for samples classification.

Returning to the machine learning context, a fundamental point is to identify curse of dimensionality and overfitting along the optimization of the Loss function. In this case, besides the dimension of the space used to represent the data, we have another degree of freedom: the dimension k of the parameter space. Following [101], the concept of overfitting in this case is related to the dimension

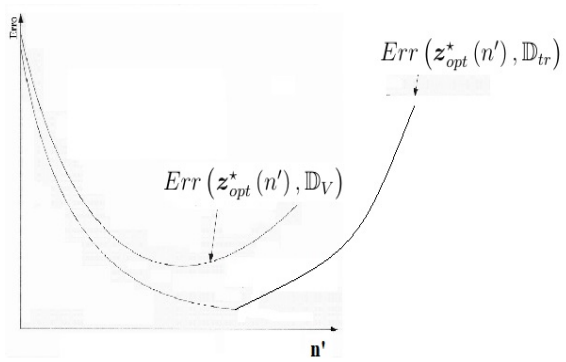
k. Specifically, if the dimension of the parameter space is too large, the machine tends to learn particular details, like noise and artifacts, of the population.

To identify curse of dimensionality along the training, we shall remember that in section 8.5 we build a partition of the data set $\mathbb{D}$ containing the disjoint subsets $\mathbb{D}_{tr}$ and $\mathbb{D}_{val}$. The former was used to optimize the objective function in Algorithms 3 and 4, in the training stage. The set $\mathbb{D}_{val}$ is going to be used in the validation step. Consider Algorithm 3 for simplicity, and compute $\boldsymbol{z}^{\star}_{opt}$ for $n' = 1$. Perform analogously for $n' = 2$, and so on. We shall notice that, along this process, we obtain a sequence $\boldsymbol{z}^{\star}_{opt}(n')$, $n' = 1, 2, ...$

We can plot the points $\left(n', Err\left(\boldsymbol{z}^{\star}_{opt}(n'), \mathbb{D}_{tr}\right)\right)$ and $\left(n', Err\left(\boldsymbol{z}^{\star}_{opt}(n'), \mathbb{D}_{val}\right)\right)$ to obtain two curves in the Cartesian plane, where Err is defined in expression (8.2). In practice, if curse of dimensionality happens then training and validation error curves go up when increasing number n' of features (subspace dimension), as observed in Figure 8.7. In this case, we should set n' with a value d before the point where all the errors start going up.

On the other hand, to verify overfitting we must proceed in another way, still considering Algorithm 3, with $n' = d$. After each iteration t of Algorithm 3, we plot the point $(t, Err(\boldsymbol{z}_t, \mathbb{D}_{tr}))$ as well as the point $(t, Err(\boldsymbol{z}_t, \mathbb{D}_{val}))$, where the error function Err is defined in expression (8.2). Therefore, we can show the training and validation errors in the same plot, as a function of iteration time t, like in Figure 8.8. Overfitting can be characterized by the behavior observed in Figures 8.8.(a)-(b). We notice that both $Err(\boldsymbol{z}_t, \mathbb{D}_{tr})$ and $Err(\boldsymbol{z}_t, \mathbb{D}_{val})$ decrease, achieving low values, until some iteration $t = T$. After that, the error $Err(\boldsymbol{z}_t, \mathbb{D}_{tr})$ remains a decreasing function with respect to T but the validation error $Err(\boldsymbol{z}_t, \mathbb{D}_{val})$ increases generating a large gap between the training and test errors [101], as represented in Figure 8.8.(a). This phenomenon indicates that the machine loses generalization capabilities; that means, ability to correctly classify samples that were not used in the training stage. Figure 8.8.(b) shows the training and validation errors, but in the $\boldsymbol{z} \times Error$ space. We expect an analogous behavior for the two plots. In fact, we shall notice that the Algorithm 3 is steered by the $-\nabla_{\boldsymbol{z}} Err(\boldsymbol{z}_t, \mathbb{D}_{tr})$ which points toward the direction of training error decreasing. However, in this case, walking along this direction implies in increasing of the validation error.

Moreover, Figure 8.8.(a) can be analyzed using another viewpoint related to



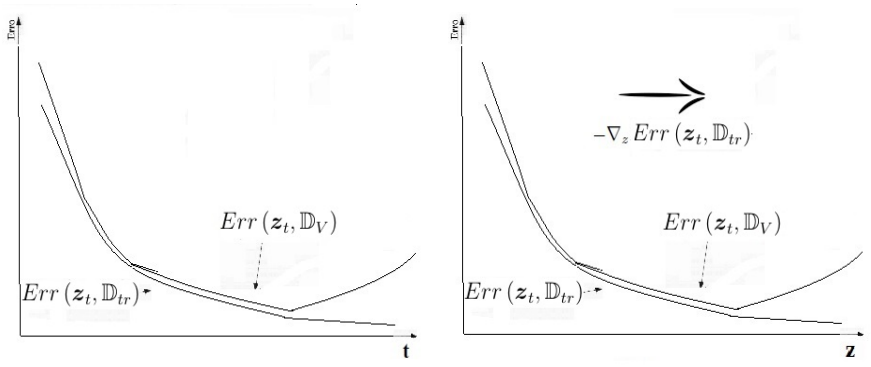
(a)

Figure 8.7: Error curves indicating curse of dimensionality.

the cardinality M of the dataset $\mathbb{D}$ as well as the number of iteration of the training algorithms. If M is too high then the parameters computed along the training process may adapt unimportant details affecting the generalization of the machine. The same phenomenon may happen if we impose an excessive number of iterations. In both cases, $M \gg 0$ and too high number of epochs, generating the behavior shown in Figure 8.8.(a), characterizes the phenomenon known as overtraining. Keeping the training for $t > T$ becomes over and the machine loses generalization capabilities. For both overfitting and overtraining, if the error $Err\left(\boldsymbol{z}_t, \mathbb{D}_{tr}\right)$ when $t = T$ is low enough, we can use the value T as a criterion to stop the learning algorithm. Otherwise, we must stop training, re-design the learning algorithm and try again.

8.8 Machine versus Manifold Learning

From a theoretical viewpoint, manifold learning is based on the assumption that the database samples (or their features) lie on a low-dimensional manifold $\mathcal{M}$ embedded in a high-dimensional space [102]. So, behind manifold learning techniques there is the data model which main elements are pictured on Figure 7.2. Moreover, in the case of Riemannian manifold learning (RML) techniques [103, 104], there is also the assumption that the low-dimensional manifold is a Riemannian one; that is, it is equipped with an inner product that varies smoothly



(a) (b)

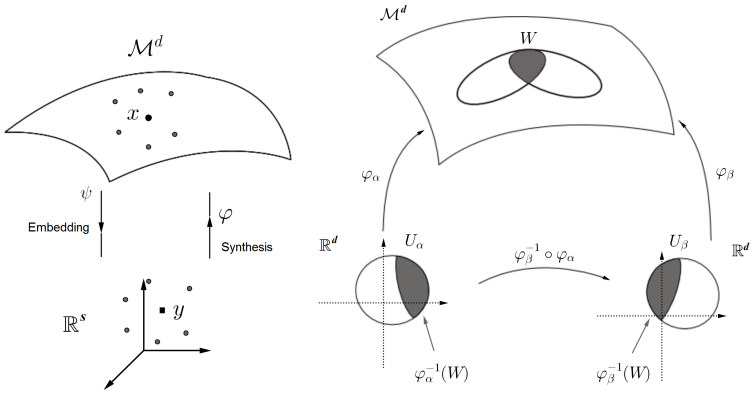
Figure 8.8: (a) Typical occurrence of overfitting during training. (b) Overfitting in the parameter space.

from point to point [105, 106].

Therefore, we need to learn the underlying intrinsic manifold geometry in order to address the problem of dimensionality reduction. Thus, instead of seeking for an optimum linear subspace, like performed for linear techniques [107], the manifold learning methods try to discover an embedding procedure that describes the intrinsic similarities of the data [108]. In order to implement this solution, manifold learning approaches take the samples of a database $\mathbb{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_M\} \subset \mathbb{R}^D$ and perform the following steps [50, 104, 109]: (a) Recover the data topology; (b) Determination of the manifold dimension d; (c) Construction of a neighborhood system; (d) Computing the embedding or local parameterizations associated to the neighborhood system. The former is a global map $\psi : \mathcal{M}^d \to \mathbb{R}^s$, $d \leq s \leq D$, while the latter is a family of local coordinate systems $\{(U_\alpha, \varphi_\alpha)\}_{\alpha \in I}$, where I is an index set, $U_\alpha \subset \mathbb{R}^d$ and $\varphi_\alpha : U_\alpha \to \mathcal{M}$.

The foundations of embedding approaches lie in the Whitney Theorem [110] which assures that since $\mathcal{M}^d \subset \mathbb{R}^D$ then $D \geq 2d + 1$. Therefore, if we compute an one-to-one smooth map $\psi : \mathcal{M}^d \to \mathbb{R}^s$ that preserves the differential structure of $\mathcal{M}^d$, called here embedding, such that $(2d + 1) \leq s < D$ then we perform dimensionality reduction in the sense that the embedding ψ allows to represent each data point using less coordinates than the original data representation. This process is pictured in Figure 8.9.(a). On the other had, Figure 8.9.(b) represents the

approach based on local parameterizations, that has the mathematical foundation in the theory of differentiable manifolds presented in section 5.7.



(a) (b)

Figure 8.9: (a) Manifold $\mathcal{M}^d \subset \mathbb{R}^D$ and the embedding in the $\mathbb{R}^s$ with synthetic data $\mathbf{x}$ in the manifold $\mathcal{M}^d \subset \mathbb{R}^D$ computed from arbitrary $\mathbf{y} \in \mathbb{R}^s$. (b) Learning the manifold structure by building local parameterizations.

Returning to the embedding approach, once we have the map $\psi : \mathcal{M}^d \to \mathbb{R}^s$, we can think about its inverse to perform the synthesis process; that means, to compute new samples in the manifold $\mathcal{M}^d \subset \mathbb{R}^D$, as represented in Figure 8.9.(a). In the case of the differentiable manifold viewpoint, the local parameterizations allow to perform synthesis of new samples directly.

The Locally Linear Embedding (LLE) follow the first approach and will be summarized as a didactic example in the manifold learning subject [4]. The LLE method works through the scheme of Figure 8.10.

In this method, given the dataset $\mathbb{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_M\} \subset \mathbb{R}^D$, for each data point $\mathbf{x}_i \in \mathbb{D}$ it is computed the K nearest neighbors (KNN) [3] represented with red color in the Figure 8.10. Then, a weighting vector $w_{i,j}$ is computed for each $\mathbf{x}_i$ by solving the optimization problem:

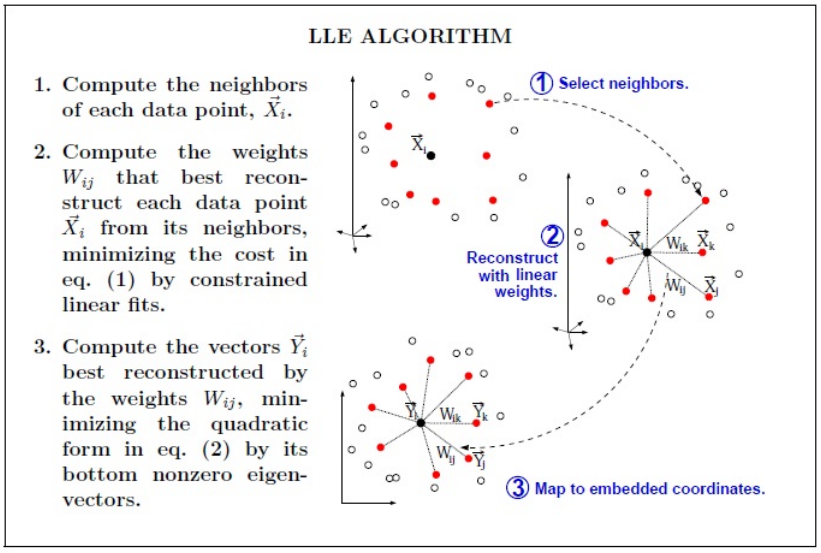


Figure 8.10: Basic steps of Locally Linear Embedding (LLE) algorithm (Reprinted from [4]).

$$\min_{(w_{i1},w_{i2},\cdots,w_{iK})} \left\| \mathbf{x}_i - \sum_{j=1}^{K(i)} w_{ij}\mathbf{x}_j \right\|^2, \tag{8.39}$$
$$subject \quad to: \sum_{j=1}^{K(i)} w_{ij} = 1.$$

where $K(i) = KNN(\mathbf{x}_i)$.

Expression (8.39) implies that we are looking for a point, closer to the point $\mathbf{x}_i$, in the convex hull of the polygon with vertices in the $KNN(\mathbf{x}_i)$ set. Finally, if we assume that the underlying data geometry can be embedded in $\mathbb{R}^s$, with $s < D$, then we can assign a feature vector $\mathbf{y}_i \in \mathbb{R}^s$ to each $\mathbf{x}_i$ by solving the expression:

$$\min_{\mathbf{y}_i} \left\| \mathbf{y}_i - \sum_{j=1}^{K(i)} w_{ij}\mathbf{y}_j \right\|^2, \tag{8.40}$$

Observe that in this case we are not computing a compact local representation of the manifold through the tangent space, like in equation 8.38. Instead, the

low dimensional output $\mathbf{y}_i$ represents the s coordinates on the manifold since the map φ follows the relation $\varphi(\boldsymbol{y}_i) = \boldsymbol{x}_i$, as shown in Figure 8.9. The objective functions in the optimization problems (8.40) can be arranged in the following unified way:

$$F(\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_M) = \sum_{i=1}^{M} \left(\mathbf{y}_i - \sum_{j=1}^{K(i)} w_{ij}\mathbf{y}_j \right)^T \left(\mathbf{y}_i - \sum_{m=1}^{K(i)} w_{im}\mathbf{y}_m \right) =$$

$$= \sum_{i=1}^{M} \left(\mathbf{y}_i^T\mathbf{y}_i - \sum_{m=1}^{K(i)} w_{im}\mathbf{y}_i^T\mathbf{y}_m - \sum_{j=1}^{K(i)} w_{ij}\mathbf{y}_j^T\mathbf{y}_i + \sum_{j,m=1}^{K(i)} w_{ij}w_{im}\mathbf{y}_j^T\mathbf{y}_m \right)$$

$$= \sum_{i,j=1}^{M} \left(\eta_{ij} - W_{ij} - W_{ji} + \sum_{k=1}^{N} w_{ki}w_{kj} \right) \mathbf{y}_i^T\mathbf{y}_j$$

$$= \sum_{i,j=1}^{M} J_{ij}\mathbf{y}_i^T\mathbf{y}_j,$$

where J is symmetric, and positive semidefinite. Additional restrictions over the objective function in the optimization problem (8.40):

- Invariant to translation of the outputs $\mathbf{y}_i$. This is achieved by requiring that:

$$\sum_{i=1}^{N} \mathbf{y}_i = 0, \tag{8.41}$$

- Invariant to rotation of the outputs $\mathbf{y}_i$ by imposing that the outputs have covariance matrix satisfying:

$$\frac{1}{N} \sum_{i=1}^{N} \mathbf{y}_i\mathbf{y}_i^T = I. \tag{8.42}$$

The solution of problem (8.40) subject to constraints (8.41)-(8.42) completes the LLE embedding construction. We shall observe that it is not obvious how to implement the synthesis in the LLE approach. Also, and more important for this monograph, we must discuss whether we can compute an embedding $\psi : \mathcal{M}^d \rightarrow \mathbb{R}^s$ using a machine learning model; that means, by training a learning machine

$\mathbb{M}$ that computes the target map. If the machine encapsulates a smooth functional space then the data manifold will be smooth also.

To solve this issue using the tools already presented, let us consider that we have two machines $\mathbb{M}_1 : \mathbb{R}^D \times \mathbb{R}^{k_1} \longrightarrow \mathbb{R}^s$ and $\mathbb{M}_2 : \mathbb{R}^s \times \mathbb{R}^{k_2} \longrightarrow \mathbb{R}^D$. Also, since each machine represents a family of functions, we can combine them and impose that:

$$\mathbb{M}_2 \left(\mathbb{M}_1 \left(\boldsymbol{x}, \boldsymbol{\omega} \right), \boldsymbol{z} \right) \simeq \boldsymbol{x}. \tag{8.43}$$

Therefore, from the machine learning viewpoint we have a new machine given by the composition of $\mathbb{M}_1$ and $\mathbb{M}_2$ such that the output of $\mathbb{M}_1$ is the input for $\mathbb{M}_2$ that, in its time, performs reconstruction of the input samples. However, how to assure that the constraint (8.43) can be satisfied? It can be performed through the training process if we minimize the reconstruction error, quantified by the mean square error (MSE), given by:

$$MSE = \sum_{i=1}^{M} \left\| \boldsymbol{x}_i - \mathbb{M}_2 \left(\mathbb{M}_1 \left(\boldsymbol{x}_i, \boldsymbol{\omega} \right), \boldsymbol{z} \right) \right\|_2^2. \tag{8.44}$$

After training, if everything is fine, we obtain optimum parameter vectors $\boldsymbol{\omega}_{opt}^{\star} \in \mathbb{R}^{k_1}$ and $\boldsymbol{z}_{opt}^{\star} \in \mathbb{R}^{k_2}$ such that the machine $\mathbb{M}_1 \left(\boldsymbol{x}, \boldsymbol{\omega}_{opt}^{\star} \right)$ computes an embeeding $\psi : \mathcal{M}^d \rightarrow \mathbb{R}^s$ while the machine $\mathbb{M}_2 \left(\boldsymbol{y}, \boldsymbol{z}_{opt}^{\star} \right)$ corresponds to the inverse map $\psi^{-1} : \mathbb{R}^s \rightarrow \mathcal{M}^d$ and could be used to perform the synthesis. According to section 8.2, this is an example of unsupervised learning since the we do not have the pairs $(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathbb{R}^D \times \mathbb{R}^s$ to train a machine to compute the function ψ that calculates $\psi \left(\boldsymbol{x}_i \right) = \boldsymbol{y}_i$. The autoencoder technique [111] is an instance of the model in expression (8.43).

Another model in the unsupervised learning context is the generative adversarial network (GUN), that follows another approach [112]. In this case there are also two machines, named the generator $\mathbb{G} : \mathbb{R}^d \times \mathbb{R}^{k_1} \longrightarrow \mathbb{R}^D$ and the discriminator $\mathbb{D} : \mathbb{R}^D \times \mathbb{R}^{k_2} \longrightarrow [0, 1]$. The generator receives as input a vector $\boldsymbol{\xi} \in \mathbb{R}^d$ drawn from a probability distribution function $p_{\boldsymbol{\xi}} \left(\boldsymbol{\xi} \right)$ and produces a sample $\boldsymbol{x}_{z_1} = \mathbb{G} \left(\boldsymbol{\xi}, \boldsymbol{z}_1 \right)$. The discriminator $\mathbb{D} \left(\boldsymbol{x}, \boldsymbol{z}_2 \right)$ outputs a single scalar that represents the probability that $\boldsymbol{x}$ comes from the probability distribution function of the data ($p_{data} \left(x \right)$) rather than from the probability distribution function associated to the generator $p_g \left(\boldsymbol{x} \right)$. The training stage involves the objective function:

$$V(\mathbb{D},\mathbb{G}) = \mathbb{E}_{x\sim p_{data}(x)}\left[\log\left(\mathbb{D}\left(x\right)\right)\right] + \mathbb{E}_{z\sim p_z(z)}\left[\log\left(1-\mathbb{D}\left(\mathbb{G}\left(z\right)\right)\right)\right], \quad (8.45)$$

that is optimized by solving the problem:

$$\min_{\mathbb{G}}\max_{\mathbb{D}} V(\mathbb{D},\mathbb{G}), \quad (8.46)$$

to seek for the optimum $(\boldsymbol{z}_1^\star, \boldsymbol{z}_2^\star) \in \mathbb{R}^{k_1} \times \mathbb{R}^{k_2}$. We shall notice that, after training the generator works as a global parametrization of the data manifold.

8.9 Exercises

1. Demonstrate that: Given a Boolean expression $F : \{0,1\}^n \longrightarrow \{0,1\}^m$, there is a circuit $C^G_{(n,m)}$ such that $C^G_{(n,m)}(x_1, x_2, ..., x_n) = F(x_1, x_2, ..., x_n)$ for all $(x_1, x_2, ..., x_n) \in B^n$. As a consequence, it is straightforward that $CG(n,m) = \mathcal{F}(n,m)$.

2. Complete the formalization of the proof for Theorem 4.

3. Generalize exercises 1 and 2 by considering functions $F : \mathbb{Q}^n \longrightarrow \mathbb{Q}$ where $\mathbb{Q}$ is the set of rational numbers.

4. Another limitation of the LLE method described in section 8.8 is the fact that, given a new sample $\widehat{\boldsymbol{x}} \notin \mathbb{D}$, where $\mathbb{D} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_M\} \subset \mathbb{R}^D$ is the input set, it is not obvious how to compute its projection in $\mathbb{R}^s$. Discuss the following approach: (a) Compute the LLE projections $\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_M$; (b) Build another input set $\mathbb{S} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i), \quad i = 1, 2, \ldots, M\}$, (c) Train a machine $\mathbb{M}$ to compute the map $\mathbb{M}(\boldsymbol{x}_i) = \boldsymbol{y}_i$; (c) Apply the machine $\mathbb{M}$ to calculate the projection of $\widehat{\boldsymbol{x}}$. Explicit the Loss function for your solution.

5. Develop a circuit that computes the operation $x - y$ (subtraction), for $x, y \in \{0,1\}$. The circuit must have x, y as inputs and two outputs, one of them used to send the signal of the operation, say, 0 for "$+$" and 1 for "$-$".

6. Develop threshold circuits to implement the following operations:

   (a) $x \oplus y$, for $x, y \in \{0, 1\}$.

   (b) Product: $x.y$, where $x, y \in \{0, 1\}$.

7. Develop circuits to implement the following operations:

   (a) $x.y \oplus z.w$, $x, y, z, w \in \{0, 1\}$.

   (b) Sum of two natural numbers m and n that can be represented by K bits.

   (c) Given a pair $(x, y) \in \{0, 1\}^2$, gives as output the pair (y, x) .

8. Simplify the following expression and give the corresponding circuit: $\sim ((a \wedge \sim b) \vee c) \wedge ((a \vee c) \wedge \sim c)$.

Chapter 9

Computing with Neural Networks

The field of neural networks (NN) is a mature research area being a branch of the machine learning area. Since the first artificial neuron model, named perceptron [11], the paradigm of neural computing has been applied almost everywhere in science and technology, attracting researches from a wide variety of backgrounds.

In this area, we want intelligent softwares to automate tasks involving pattern recognition (images analysis, computer-aided diagnoses in medicine, for instance), learn data patterns of the input set in order to generate new samples with some variations (data synthesis), regression and probability density function estimation. In neural networks, like in all machine learning algorithm, the solution to accomplish these tasks is to allow computers to learn from training data. From the viewpoint of chapter 8, neural networks constitute a class of learning machines. Each neural network model encapsulates a functional space through its inner parameters. So, neural networks are machines with internal parameters. The training process performs parameter adaptation, in a optimization process that uses the training samples to formalize the concept of learning from experience, as presented in chapter 8. The learning process can be supervised, unsupervised or semi-supervised, or based on reinforcement strategies, as summarized in section 8.2.

From the viewpoint of computational models, neural networks use a different approach to problem solving than the conventional computers. In conventional computers there is a program, a set of instructions to be followed in order to solve a problem. On the other hand, neural networks process information in a similar

way the human brain does. The network is composed of a large number of highly interconnected processing elements (neurons) working in parallel. There is no a main program. Instead, the neurons are arranged in layers and computation is performed layer-by-layer until the final output.

The neural computation process can be organized in a graph, or circuit (section 8.3), whose nodes (neurons) are arranged in layers. With theoretical and hardware developments, nowadays we can deal with deep graphs, representing neural nets with many layers, generating the deep learning field. There are a plenty of books and an ocean of scientific paper in neural networks, covering aspects in computer science, mathematics, as well as applications.

9.1 Perceptron Model

The first logical neuron was developed by W. S. McCulloch and W.A. Pitts in 1943 [11]. It describes the fundamental functions and structures of a neural cell reporting that a neuron will fire an impulse only if a threshold value is exceeded.

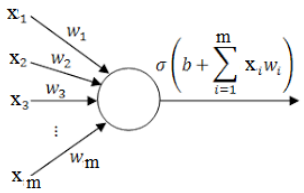


Figure 9.1: McCulloch-Pitts neuron model.

Figure 9.1 shows the basic elements of McCulloch-Pitts model: $x_i \in \mathbb{R}$, $i = 1, 2, ..., m$ are the inputs, that can be arranged in the input vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_m) \in \mathbb{R}^m$. The coefficients $w_i \in \mathbb{R}$ and the bias b are machine parameters, usually named weights in the neural network literature. The combination $\sum_{i=1}^{m} w_i x_i + b$ is named the induced local field. The real value y is the output, and σ is the *activation function* that computes the value y in the output. A

simple choice for σ is the signal function $\sigma(.) = sgn(.)$ defined by:

$$\sigma\left(\sum_{i=1}^{m} w\left(i\right) x\left(i\right) + b\right) = \begin{cases} 1, & if\ \sum_{i=1}^{m} w\left(i\right) x\left(i\right) + b > 0 \\ -1, & otherwise. \end{cases} . \tag{9.1}$$

But the McCulloch-Pitts neuron did not have a mechanisms for *learning*. Based on biological evidences, D.O. Hebb suggested a rule to adapt the weights, which is interpreted as learning rule for the system [11]. This biological inspired procedure can be expressed in the following manner:

$$w_i^{new} = w_i^{old} + \Delta w_i; \quad \Delta w_i = \eta(y^{desired} - y)x_i, \tag{9.2}$$

where w^{new} and w^{old} are adapted and previous weights respectively, η is a real parameter to control the rate of learning and $y^{desired}$ is the desired (know) output. This *learning rule* plus the elements of Figure 9.1 is called the perceptron model for a neuron.

Then, the learning occurs through training, or exposure to a know set of input/output data. A usual trick in machine learning literature is to augment the input vector and the weights vector as follows:

$$\boldsymbol{x} \equiv (1, x_1, x_2, \ldots, x_m) \in \mathbb{R}^{1+m}, \tag{9.3}$$

$$\boldsymbol{w} \equiv (b, w_1, w_2, \ldots, w_m) \in \mathbb{R}^{1+m}, \tag{9.4}$$

respectively, in order to re-write the argument of the activation function σ in Figure 9.1 as:

$$b + \sum_{i=1}^{m} x_i w_i = \boldsymbol{x} \cdot \boldsymbol{w}. \tag{9.5}$$

In this way, the input set can be written as:

$$\mathbb{D} = \left\{\left(\boldsymbol{x}_1, l_1\right), \left(\boldsymbol{x}_2, l_2\right), \cdot, \cdot, \cdot, \left(\boldsymbol{x}_M, l_M\right)\right\} \subset \mathbb{R}^{1+m} \times \{0, 1\}, \tag{9.6}$$

where, according to expression (9.3), $\boldsymbol{x}_i = (1, x_{i,1}, x_{i,2}, \ldots, x_{i,m})$. The training algorithm iteratively adjusts the connection weights and bias assembled in the extended weight vector (9.4) analogous to synapses in biological nervous. These connection weights and bias store the knowledge necessary to solve specific problems. Figure 9.2 shows the main steps of the perceptron algorithm, where in this figure we must convert $\boldsymbol{x}\left(n\right)$ to $\boldsymbol{x}_n$ and $\boldsymbol{w}\left(n\right)$ to $\boldsymbol{w}_n$ to be consistent with the notation of this monograph.

TABLE 3.2 Summary of the Perceptron Convergence Algorithm

Variables and Parameters:

$\mathbf{x}(n)$ = $(m+1)$-by-1 input vector
$= [+1, x_1(n), x_2(n), \ldots, x_m(n)]^T$

$\mathbf{w}(n)$ = $(m+1)$-by-1 weight vector
$= [b(n), w_1(n), w_2(n), \ldots, w_m(n)]^T$

$b(n)$ = bias

$y(n)$ = actual response (quantized)

$d(n)$ = desired response

η = learning-rate parameter, a positive constant less than unity

1. *Initialization.* Set $\mathbf{w}(0) = \mathbf{0}$. Then perform the following computations for time step $n = 1, 2, \ldots$.

2. *Activation.* At time step n, activate the perceptron by applying continuous-valued input vector $\mathbf{x}(n)$ and desired response $d(n)$.

3. *Computation of Actual Response.* Compute the actual response of the perceptron:

$$y(n) = \text{sgn}[\mathbf{w}^T(n)\mathbf{x}(n)]$$

where $\text{sgn}(\cdot)$ is the signum function.

4. *Adaptation of Weight Vector.* Update the weight vector of the perceptron:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)$$

where

$$d(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \\ -1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \end{cases}$$

5. *Continuation.* Increment time step n by one and go back to step 2.

Figure 9.2: Perceptron procedure: model and learning rule.

9.2 Convergence of Perceptron Learning Rule

In this section we are going to use a simplified version of the perceptron learning rule (expression (9.2)) given by:

$$\boldsymbol{w}^{new} = \boldsymbol{w}^{old} + (d - y)\boldsymbol{x},$$

where $d \in \{1, -1\}$ represent the desired label for the sample $\boldsymbol{x}$ and $y \in \{1, -1\}$ denotes the perceptron result. The development bellow follows section 3.5 of reference [11].

Consider the set:

$$F = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_M\} \subset \mathbb{R}^{m+1},$$

where $\boldsymbol{x}_i = (1, x_{i1}, x_{i2}, \ldots, x_{im})$, $i = 0, 1, \ldots, M$.

We are supposing that $F = F^+ \cup F^-$and that there is a unit vector $\boldsymbol{w}^\star = (b^\star, w_1^\star, w_2^\star, \ldots, w_m^\star)$, which partitions up the set F, and a small positive fixed number δ such that:

$$\boldsymbol{w}^\star \cdot \boldsymbol{x} = \sum_{j=0}^{m} w_j x_j = b^\star + \sum_{j=1}^{m} w_j^\star x_j > \delta, \quad if \quad \boldsymbol{x} \in F^+, \tag{9.7}$$

$$\boldsymbol{w}^\star \cdot \boldsymbol{x} = \sum_{j=0}^{m} w_j x_j = b^\star + \sum_{j=1}^{m} w_j^\star x_j < -\delta, \quad if \quad \boldsymbol{x} \in F^-. \tag{9.8}$$

We can simplify the development if we define $\mathbb{F} = F^+ \cup (-F^-)$, where $\boldsymbol{x} \in -F^- \Longrightarrow -\boldsymbol{x} \in F^-$. Therefore:

$$\boldsymbol{w}^\star \cdot \boldsymbol{x} > \delta, \quad if \quad \boldsymbol{x} \in \mathbb{F}.$$

We shall define:

$$G(\boldsymbol{w}) = \frac{\boldsymbol{w}^\star \cdot \boldsymbol{w}}{\|\boldsymbol{w}\|}, \tag{9.9}$$

which satisfies $-1 \leq G(\boldsymbol{w}) \leq 1$ because $\boldsymbol{w}^\star \cdot \boldsymbol{w} = \|\boldsymbol{w}\| \cos\theta$. We randomly initialize $\boldsymbol{w} = \boldsymbol{w}_0$. Then if take $\boldsymbol{x}_0 \in \mathbb{F}$ and $\boldsymbol{w}_0 \cdot \boldsymbol{x}_0 < 0$ we must perform

$$\boldsymbol{w}_1 = \boldsymbol{w}_0 + (d_0 - y_0)\, \boldsymbol{x}_0.$$

Now, we take $\boldsymbol{x}_1 \in \mathbb{F}$. If $\boldsymbol{w}_1 \cdot \boldsymbol{x}_1 < 0$ then we perform again:

$$\boldsymbol{w}_2 = \boldsymbol{w}_1 + (d_1 - y_1)\, \boldsymbol{x}_1,$$

and, in this case:

$$\boldsymbol{w}^\star \cdot \boldsymbol{w}_2 = \boldsymbol{w}^\star \cdot (\boldsymbol{w}_1 + (d_1 - y_1)\, \boldsymbol{x}_1) = \boldsymbol{w}^\star \cdot (\boldsymbol{w}_0 + (d_0 - y_0)\, \boldsymbol{x}_0 + (d_1 - y_1)\, \boldsymbol{x}_1)$$

$$= \boldsymbol{w}^\star \cdot \boldsymbol{w}_0 + \boldsymbol{w}^\star \cdot (d_0 - y_0)\, \boldsymbol{x}_0 + \boldsymbol{w}^\star \cdot (d_1 - y_1)\, \boldsymbol{x}_1 =$$

$$= \boldsymbol{w}^\star \cdot \boldsymbol{w}_0 + (d_0 - y_0)\,(\boldsymbol{w}^\star \cdot \boldsymbol{x}_0) + (d_1 - y_1)\,(\boldsymbol{w}^\star \cdot \boldsymbol{x}_1) >$$

$$= \boldsymbol{w}^\star \cdot \boldsymbol{w}_0 + (d_0 - y_0)\,\delta + \eta\,(d_1 - y_1)\,\delta.$$

So, if

$$\boldsymbol{w}^\star \cdot \boldsymbol{w}_0 > 0 \quad and \quad 1 \leq d_i - y_i \leq 2 \tag{9.10}$$

then:

$$\boldsymbol{w}^\star \cdot \boldsymbol{w}_2 > (d_0 - y_0)\,\delta + (d_1 - y_1)\,\delta \geq 2\delta.$$

By induction, we can show that:

$$\boldsymbol{w}^\star \cdot \boldsymbol{w}_n > n\delta,$$

if conditions like (9.10) holds.

Considering the denominator of (9.9), we have:

$$\boldsymbol{w}_1 \cdot \boldsymbol{w}_1 = (\boldsymbol{w}_0 + (d_0 - y_0)\,\boldsymbol{x}_0) \cdot (\boldsymbol{w}_0 + (d_0 - y_0)\,\boldsymbol{x}_0) =$$

$$\boldsymbol{w}_0 \cdot \boldsymbol{w}_0 + 2\,(d_0 - y_0)\,\boldsymbol{x}_0 \cdot \boldsymbol{w}_0 + (d_0 - y_0)^2\,\boldsymbol{x}_0 \cdot \boldsymbol{x}_0 < \boldsymbol{w}_0 \cdot \boldsymbol{w}_0 + 4\boldsymbol{x}_0 \cdot \boldsymbol{x}_0 \tag{9.11}$$

If $\boldsymbol{w}_1 \cdot \boldsymbol{x}_1 < 0$ then:

$$\boldsymbol{w}_2 \cdot \boldsymbol{w}_2 = (\boldsymbol{w}_1 + (d_1 - y_1)\,\boldsymbol{x}_1) \cdot (\boldsymbol{w}_1 + (d_1 - y_1)\,\boldsymbol{x}_1)$$

$$= \boldsymbol{w}_1 \cdot \boldsymbol{w}_1 + 2\,(d_1 - y_1)\,\boldsymbol{w}_1 \cdot \boldsymbol{x}_1 + (d_1 - y_1)^2\,\boldsymbol{x}_1 \cdot \boldsymbol{x}_1$$

$$< \boldsymbol{w}_1 \cdot \boldsymbol{w}_1 + 4\boldsymbol{x}_1 \cdot \boldsymbol{x}_1. \tag{9.12}$$

By substituting the result (9.11) into expression (9.12) we get:

$$\boldsymbol{w}_2 \cdot \boldsymbol{w}_2 < \boldsymbol{w}_0 \cdot \boldsymbol{w}_0 + 4\boldsymbol{x}_0 \cdot \boldsymbol{x}_0 + 4\boldsymbol{x}_1 \cdot \boldsymbol{x}_1 < \boldsymbol{w}_0 \cdot \boldsymbol{w}_0 + 4 \cdot 2 \cdot \max\left\{\boldsymbol{x}_0 \cdot \boldsymbol{x}_0, \boldsymbol{x}_1 \cdot \boldsymbol{x}_1\right\}.$$

By induction:

$$\|\boldsymbol{w}_n\|^2 < \boldsymbol{w}_0 \cdot \boldsymbol{w}_0 + 4n \max\left\{\boldsymbol{x}\left(i\right) \cdot \boldsymbol{x}\left(i\right),\ i = 0, 1, 2, \ldots, M\right\} \Longrightarrow \|\boldsymbol{w}\left(n\right)\| < \left(\boldsymbol{w}_0 \cdot \boldsymbol{w}_0 + 4n\Pi\right)^{1/2},$$

where $\Pi = \max\left\{\boldsymbol{x}\left(i\right) \cdot \boldsymbol{x}\left(i\right),\ i = 0, 1, 2, \ldots, M\right\}$.

Consequently:

$$G\left(\boldsymbol{w}\left(n\right)\right) = \frac{\boldsymbol{w}^{\star} \cdot \boldsymbol{w}\left(n\right)}{\|\boldsymbol{w}\left(n\right)\|} > \frac{n\delta}{\left(\boldsymbol{w}_0 \cdot \boldsymbol{w}_0 + 4n\Pi\right)^{1/2}} \approx \frac{n\delta}{\left(4n\Pi\right)^{1/2}},$$

for $n \gg 0$. Hence:

$$1 \geq \frac{n\delta}{\left(4n\Pi\right)^{1/2}} = \frac{n\delta}{2\left(n\right)^{1/2}\left(\Pi\right)^{1/2}} = \frac{\left(n\right)^{1/2}\delta}{2\left(\Pi\right)^{1/2}},$$

and:

$$n \leq \frac{4\Pi}{\left(\delta\right)^2}. \tag{9.13}$$

Consequently, expression (9.13) assures that the simplified perceptron model always converge after a finite number of updates of the vector of weights $\boldsymbol{w}$.

9.3 Multilayer Perceptron

In terms of dataflow, we can think a neural network as a directed, acyclic graph, whose nodes are neurons (perceptrons), arranged in layers, with weights associated with the edges [113]. As an example, let us consider the learning machine $\mathbb{M} = \mathbb{M}\left(\boldsymbol{x}; \boldsymbol{w}\right)$ represented in Figure 9.3, where:

- The input vector $\boldsymbol{x} = \left(x_1, x_2, \ldots, x_m\right)^T \in \mathbb{R}^m$ is a generic data sample,
- The first layer is the input one, with index $j = 0$, containing $n_0 = m$ neurons, with the goal to distribute information to the computing layers,
- The layer $j = 1$ is the hidden layer, positioned between the input and output layers and formed by n_1 neurons,
- Synaptic weights $w_{jk}^{(i)}$, associated to the connection of the neuron j of layer $i - 1$ to the neuron k of layer i,

- Bias $b_j^{(i)}$ associated to the perceptron j of the layer i,

- Output layer, with index $i = 2$, containing n_2 neurons,

Let us introduce the notation: $\left\{w_{jk}^{(i)}\right\}_{1\leq k\leq n_i}^{1\leq j\leq n_{i-1}}$, $\boldsymbol{b}^{(i)} = \left(b_1^{(i)}, b_2^{(i)}, \ldots, b_{n_i}^{(i)}\right)$, where $i = 1, 2$ in the case of the neural network of Figure 9.3. Consequently, the machine weights can be assembled in the array $\boldsymbol{w} = \left(\left\{w_{jk}^{(1)}\right\}_{1\leq k\leq n_1}^{1\leq j\leq n_0}; \left\{w_{jk}^{(2)}\right\}_{1\leq k\leq n_2}^{1\leq j\leq n_1}; \boldsymbol{b}^{(1)}; \boldsymbol{b}^{(2)}\right)^T$.

Hence, the parameter space of the machine model has dimension $k = n_0 \cdot n_1 + n_1 \cdot n_2 + n_1 + n_2$ and the learning machine encapsulates a space composed by functions $f : \mathbb{R}^m \times \mathbb{R}^k \longrightarrow \mathbb{R}^{n_2}$.

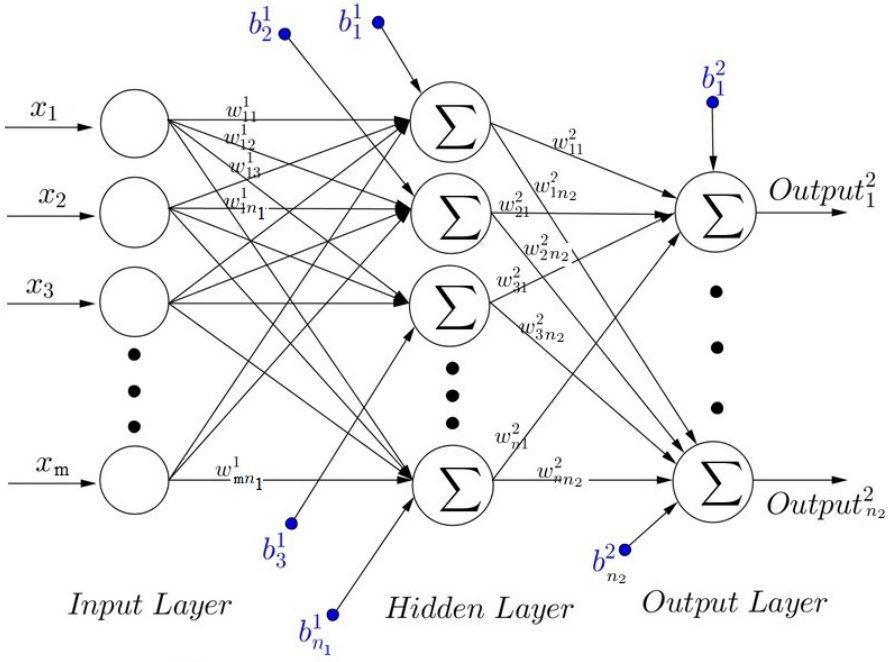


Figure 9.3: Traditional multilayer perceptron (MLP) neural network.

A neural network with only one hidden layer is also named a shallow neural network, like the one in Figure 9.3. On the other hand, deep neural networks, like the one in Figure 9.4, have more then one hidden layer. In the more general case, a deep neural network has:

- The input layer ($j = 0$), with $n_0 = m$ neurons,

- Hidden layers ($j = 1, 2, \ldots, L-1$) where the layer j has n_j neurons, $j = 1, 2, \ldots, L-1$,
- Output layer ($j = L$) with n_L neurons.

Consequently, the machine model $\mathbb{M} = \mathbb{M}(\boldsymbol{x}; \boldsymbol{w})$ can be represented by a function family $f : \mathbb{R}^m \times \mathbb{R}^k \longrightarrow \mathbb{R}^{n_L}$,where:

$$k = \sum_{i=0}^{L-1} n_i n_{i+1} + \sum_{i=1}^{L} n_i, \tag{9.14}$$

where $n_0 = m$.

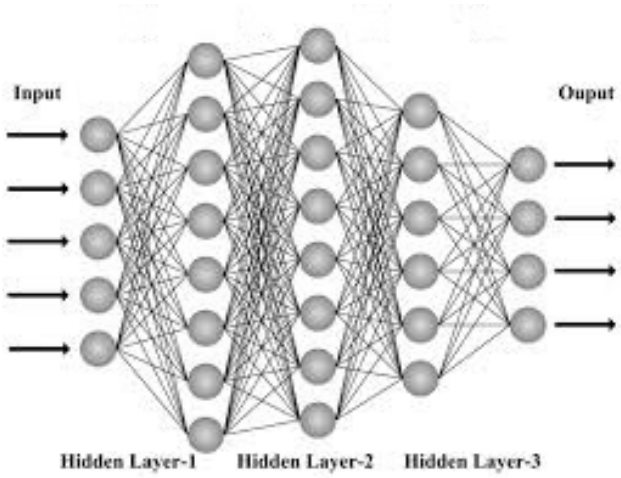


Figure 9.4: Deep multilayer perceptron neural network.

The MLP is also named fully connected neural network in the sense that all the nodes from the layer '$i-1$' are connected to the nodes of layer i. We must be careful about this definition because, if a weight $w_{jk}^{(i)}$ is null, the effect is the same as if there was no connection between neuron j of layer $i-1$ and neuron k of layer i.

9.4 Activation Functions

The nonlinear capabilities of neural networks are the consequence of application of nonlinear activation functions. If we desire the output to be in the range $[0, 1]$,

one possibility for activation functions is the Logistic (or sigmoid) function.

$$f\left(z;\alpha\right)=\frac{1}{1+\exp\left(-\alpha z\right)}. \tag{9.15}$$

Derivative respect to 'z':

$$\frac{df}{dz}=-\left(1+\exp\left(-\alpha z\right)\right)^{-2}\left(-\alpha\exp\left(-\alpha z\right)\right)=\frac{\alpha\exp\left(-\alpha z\right)}{\left(1+\exp\left(-\alpha z\right)\right)^{2}}.$$

If we compare these expressions we see that:

$$\frac{df}{dz}=\left(\alpha\exp\left(-\alpha z\right)\right)\left(f\left(z\right)\right)^{2}. \tag{9.16}$$

However, from expression (9.15) we get:

$$\left(1+\exp\left(-\alpha z\right)\right)f\left(z\right)=1,$$

consequently:

$$\exp\left(-\alpha z\right)=\frac{1}{f\left(z\right)}-1.$$

By substituting this result into equation (9.16) we obtain:

$$\frac{df}{dz}=\alpha\left(\frac{1}{f\left(z\right)}-1\right)\left(f\left(z\right)\right)^{2}=\alpha\left(1-f\left(z\right)\right)f\left(z\right),$$

which can be inserted into back-propagation results (see section 9.7).

However, sigmoid function suffers some drawbacks due to the fact its image set is bounded in the interval $(0,1)$. Hence, given a regression problem whose target function if $f:\mathbb{R}^{m}\longrightarrow[0,y_{\max}]\subset\mathbb{R}$ we shall normalize the image set to $[0,1]$ and, consequently, the combination of normalization and sigmoid will bind a large range of input to a small range of output [114]. Such procedure could generate small gradient values during the training process, resulting is premature convergence to a poor solution or inability of deep models to learn patterns on a given dataset due to the fact that the machine propagates only small gradient values from the output layer to the input one (see section 9.7), a phenomenon known as vanishing gradient problem [115, 113]

To overcome limitation of sigmoid to output only a small range of positive values, but keeping the benefits of sigmoidal functions, the hyperbolic tangent ($tanh$) function is introduced:

$$\tanh(x) = \frac{1 - \exp(-x)}{1 + \exp(-x)}, \tag{9.17}$$

that can produce zero-centered results. However, $\tanh$ also undergoes the vanishing gradient problem.

The rectified linear unit (ReLU) function, proposed in [116] is defined as:

$$f(x) = \max\{0, x\}. \tag{9.18}$$

Since its proposal, the ReLU has been one of the most widely used activation function in deep architectures [114]. We shall notice that ReLU function is continuous, but its derivative is not defined for $x = 0$. Moreover, it is not-bounded and not zero-centered. The computational cost of the evaluation of ReLU is cheap due to its definition. Also, it can introduces sparsity because all negative input values are pushed to zero.

In deep architectures, the ReLU function is generally used within the hidden layers with another activation functions in the output unities. In this way, the literature has reported better performance and generalization than the sigmoid or $\tanh$ alone [114]. A variation of ReLU is the parametric ReLU function:

$$f(x; a) = \begin{cases} ax & if\ x \leq 0 \\ & \\ x & otherwise \end{cases}, \tag{9.19}$$

where $a > 0$. The special case of $a = 0.01$ is named Leaky ReLU function. Other activation functions can be found in [117]

In multiclass problems, given an imput sample $\boldsymbol{x}$, the neural network must outputs a vector $\boldsymbol{y} = (y_1, y_2, \ldots, y_K)$ where K is the number of classes. So, let us assembly the induced local fields of the output layer in the form $\boldsymbol{z} = W\boldsymbol{h} + \boldsymbol{b}$. If we consider a probabilistic interpretation of feedforward outputs of neural networks then the set of outputs should have the form of a probability distribution over class labels, conditioned on the input; that means, $\widehat{Out}_i = p(c = i|\boldsymbol{x})$ [118]. In this case, it is also required that $\widehat{Out}_i \geq 0$ and $\sum_{i=1}^{K} Out_i = 1$. To enforce such

stochastic constraints, it is suggested in [118] the normalized exponential output:

$$Out_i = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)}, i = 1, 2, ..., K \tag{9.20}$$

which can be arranged in the softmax activation function [118, 93]:

$$softmax(\boldsymbol{z}) = \left(\frac{\exp(z_1)}{\sum_{j=1}^{K} \exp(z_j)}, \frac{\exp(z_2)}{\sum_{j=1}^{K} \exp(z_j)}, \ldots, \frac{\exp(z_K)}{\sum_{j=1}^{K} \exp(z_j)} \right). \tag{9.21}$$

We shall remark that the success of an activation function may be conditioned to the loss function used. For instance, the function L in expression (8.2) is named cost function. In classification problems, quadratic cost function defined by:

$$L(\boldsymbol{x}, \boldsymbol{d}; \boldsymbol{w}) = \|\mathbb{M}(\boldsymbol{x}; \boldsymbol{w}) - \boldsymbol{d}\|_2^2, \tag{9.22}$$

is commonly used, where $\boldsymbol{x}$ denotes the input, $\boldsymbol{d}$ the desired output and $\boldsymbol{w}$ the machine parameters (weights vector). The corresponding loss functions (expression (8.3)) is given by:

$$Loss(\boldsymbol{w}; \mathbb{D}) = \frac{1}{|\mathbb{D}|} \sum_{i=1}^{|\mathbb{D}|} \|\mathbb{M}(\boldsymbol{x}_i; \boldsymbol{w}) - \boldsymbol{d}\|_2^2 + \sum_{\tau=1}^{K} \varsigma_\tau R_\tau(\boldsymbol{z}; \mathbb{D}). \tag{9.23}$$

The negative log-likelihood cost function is another possibility in this case, defined by:

$$L(\boldsymbol{x}, d) = -\ln p(d|\boldsymbol{x}), \tag{9.24}$$

where in this expression d is the class label of sample $\boldsymbol{x}$.

See [119] for a survey in cost function for machine learning applications. It is observed in [120] that the negative log-likelihood cost function coupled with softmax outputs worked much better for classification problems than the quadratic cost for feedforward neural networks.

9.5 Neural Networks and Universality

The main question in this section is: What kind of functions can be computed by neural networks?

To present the solution step-by-step, we start by considering only Boolean functions $F : B^n \longrightarrow B^m$, where $B = \{0, 1\}$. So, we return to Theorem 4, which assures that for every Boolean function $F : B^n \longrightarrow B^m$, there is a weighted threshold circuit $TC^{n,\Delta}_{(\boldsymbol{w},m)}$ such that $TC^{n,\Delta}_{(\boldsymbol{w},m)}(\boldsymbol{x}) = F(\boldsymbol{x})$ for all $\boldsymbol{x} \in B^n$. Moreover, we should notice that a simple change in the neuron model:

$$y = \chi(\sum_{i=1}^{n} w_i x_i - b), \tag{9.25}$$

where:

$$\chi(\sum_{i=1}^{n} w_i x_i - b) = 1,\ if\ \sum_{i=1}^{n} w_i x_i > b, \tag{9.26}$$

$$\chi(\sum_{i=1}^{n} w_i x_i - b) = 0,\ otherwise, \tag{9.27}$$

allows to identify neurons with threshold gates, defined by expressions (8.16), if we restrict $(x_1, x_2, \ldots, x_n) \in B^n$ and $(w_1, w_2, \ldots, w_n) \in \mathbb{Q}^n$. In this way, neurons are threshold gates $T^{n,\Delta}_{\boldsymbol{w}} : B^n \longrightarrow \{0, 1\}$. Consequently, a neural network becomes a weighted threshold circuit $TC^{n,\Delta}_{(\boldsymbol{w},m)}$. So, a straightforward consequence of Theorem 4 is that

$$TC^{n}_{(m)} \supset \mathcal{F}(n, m), \tag{9.28}$$

where $TC^{n}_{(m)}$ is the class of functions that can be computable by threshold circuits and $\mathcal{F}(n, m)$ is the class of Boolean functions $F : B^n \longrightarrow B^m$.

Therefore, we can affirm that given a Boolean functions $F : B^n \longrightarrow B^m$ there is a neural network machine $\mathbb{M} = \mathbb{M}(\boldsymbol{x}; \boldsymbol{w}^*)$, where $\boldsymbol{w}^*$ is a fixed parameter vector in $\mathbb{R}^k$, such that $\mathbb{M}(\boldsymbol{x}; \boldsymbol{w}^*) = F(\boldsymbol{x})$, for all $\boldsymbol{x} \in B^m$.

To generalize this result for function $f : \mathbb{Q}^n \longrightarrow \mathbb{Q}^m$, such that:

$$f(q_1, q_2, \ldots, q_n) = (f_1(q_1, q_2, \ldots, q_n), f_2(q_1, q_2, \ldots, q_n), \ldots, f_m(q_1, q_2, \ldots, q_n)), \tag{9.29}$$

we shall use the following elements:

- A bijective function $\varrho : \mathbb{Z} \longrightarrow \{0, 1\}^{k+1}$, with $k \in \mathbb{N}$ large enough, such that If $z \in \mathbb{Z}$ then $\varrho(z) = (c_0, c_1, \ldots, c_k)$ is the only binary sequence in $\{0, 1\}^{k+1}$ that satisfies:

$$z = s(c_k)\left(\sum_{j=1}^{k} c_{j-1} 2^{j-1}\right), \tag{9.30}$$

where $s(1) = +1$ and $s(0) = -1$.

- Given $q \in \mathbb{Q}$, $\exists\,(a, b) \in \mathbb{Z} \times \mathbb{Z}_*$ such that:

$$q = \frac{a}{b}.$$

Therefore, a function $f : \mathbb{Q}^n \longrightarrow \mathbb{Q}^m$, with $f = f(q_1, q_2, \ldots, q_n)$, can be written as $g : \mathbb{Z}^{2n} \longrightarrow \left(B^{k+1}\right)^{2m}$, such that

$$f(q_1, q_2, \ldots, q_n) = g((a_1, b_1), (a_2, b_2), \ldots, (a_m, b_m)), \tag{9.31}$$

where $(a_i, b_i) \in \mathbb{Z} \times \mathbb{Z}_*$ and $q_i = a_i/b_i$, $i = 1, 2, \ldots, n$. Moreover, we compose functions g and ϱ as element-wise:

$$g((a_1, b_1), (a_2, b_2), \ldots, (a_m, b_m)) =$$

$$= g\left(\left(\varrho^{-1}(\boldsymbol{\alpha}_1), \varrho^{-1}(\boldsymbol{\beta}_1)\right), \left(\varrho^{-1}(\boldsymbol{\alpha}_2), \varrho^{-1}(\boldsymbol{\beta}_2)\right), \ldots, \left(\varrho^{-1}(\boldsymbol{\alpha}_n), \varrho^{-1}(\boldsymbol{\beta}_n)\right)\right) \tag{9.32}$$

where $\boldsymbol{\alpha}_i \in \{0, 1\}^{k+1}$ and $\boldsymbol{\beta}_i \in \{0, 1\}^{k+1}$ are the binary representations of a_i and b_i in $\{0, 1\}^{k+1}$, respectively. If we put together expressions (9.29),(9.31), and (9.32) we can build a function $g \circ \varrho^{-1} : \left(\{0, 1\}^{k+1} \times \{0, 1\}^{k+1}\right)^n \longrightarrow \mathbb{Q}^m$ that satisfies:

$$f(q_1, q_2, \ldots, q_n) =$$

$$= g\left(\left(\varrho^{-1}(\boldsymbol{\alpha}_1), \varrho^{-1}(\boldsymbol{\beta}_1)\right), \left(\varrho^{-1}(\boldsymbol{\alpha}_2), \varrho^{-1}(\boldsymbol{\beta}_2)\right), \ldots, \left(\varrho^{-1}(\boldsymbol{\alpha}_n), \varrho^{-1}(\boldsymbol{\beta}_n)\right)\right) \equiv$$

$$\equiv g \circ \varrho^{-1}(\boldsymbol{\alpha}_1, \boldsymbol{\beta}_1; \boldsymbol{\alpha}_2, \boldsymbol{\beta}_2; \ldots, \boldsymbol{\alpha}_n, \boldsymbol{\beta}_n). \tag{9.33}$$

The next step, let as exercise, is to deal with the codomain $\mathbb{Q}^m$ because the network output should be in the set $\left(\{0, 1\}^{k+1} \times \{0, 1\}^{k+1}\right)^m$. To perform this task we shall define the function:

$$\varrho \circ g \circ \varrho^{-1}(\boldsymbol{\alpha}_1, \boldsymbol{\beta}_1; \boldsymbol{\alpha}_2, \boldsymbol{\beta}_2; \ldots, \boldsymbol{\alpha}_n, \boldsymbol{\beta}_n) = \varrho \circ f(q_1, q_2, \ldots, q_n), \tag{9.34}$$

and work with it.

9.6 Shallow versus Deep Neural Networks

Owing the DNF and CNF normal forms defined in expressions (3.4) and 3.7, respectively, we can discuss equivalence of shallow and deep neural networks from the viewpoint of computational models.

We will demonstrate for the particular case $m = 1$ and let the generalization for $m > 1$ as exercise. Specifically, let a neural network computed by the threshold circuit $TC^{n,\Delta}_{(\boldsymbol{w},1)}$. Hence, from DNF normal form (expression (3.4)) the Boolean function $TC^{n,\Delta}_{(\boldsymbol{w},1)} : B^n \longrightarrow \{0,1\}$can be written as:

$$TC^{n,\Delta}_{(\boldsymbol{w},1)}(x_1, x_2, ..., x_n) = \left(t^1_1 \wedge t^1_2 \wedge ... \wedge t^1_{k_1}\right) \vee ... \vee \left(t^m_1 \wedge t^m_2 \wedge ... \wedge t^m_{k_m}\right), \tag{9.35}$$

where t^i_j is either a variable x_l or a negation of a variable $\sim x_l$. We shall introduce the notation $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ and $\sim \boldsymbol{x} = (\sim x_1, \sim x_2, \ldots, \sim x_n)$.

Now, we shall notice that expressions $\left(t^i_1 \wedge t^i_2 \wedge ... \wedge t^i_{k_i}\right)$ that appear in the DNF form can be computed by the threshold gate:

$$Th^{2n,\Delta}_{\boldsymbol{w}} : \{0,1\}^{2n} \rightarrow \{0,1\},$$

$$Th^{2n,k_i}_{\boldsymbol{w}}(\boldsymbol{x}, \sim \boldsymbol{x}) = 1, \quad if \quad \sum_{l=1}^{n} w_l x_l + \sum_{l=1}^{n} w_{l+k_l}(\sim x_l) \geq k_i, \tag{9.36}$$

$$Th^{2n,k_i}_{\boldsymbol{w}}(\boldsymbol{x}, \sim \boldsymbol{x}) = 0, \quad otherwise, \tag{9.37}$$

where $w_l = 1$ if there exists $j \in \{1, 2, \ldots, k_i\}$ such that $t^i_j = x_l$ and $w_l = 0$ otherwise, for $l = 1, 2, \ldots, n$. Also, $w_{l+n} = 1$ if there exists $j \in \{1, 2, \ldots, k_i\}$ that satisfies $t^i_j = \sim x_l$ and $w_{l+k_i} = 0$ otherwise, for $l = 1, 2, \ldots, n$.

Moreover, expression in the right-hand side of equation (9.35) can be represented as:

$$Th^{m,\Delta}_{\boldsymbol{w}} : \{0,1\}^{m} \rightarrow \{0,1\},$$

$$Th^{m,\Delta}_{\boldsymbol{w}}(y_1, y_2, ..., y_m) = 1, \quad if \quad \sum_{j=1}^{m} w_j y_j \geq \Delta,$$

$$Th^{m,\Delta}_{\boldsymbol{w}}(y_1, y_2, ..., y_m) = 0, \quad otherwise,$$

where $\Delta = 1$, $\boldsymbol{w} = (1, 1, 1, \ldots, 1)$ and $y_i = \left(t_1^i \wedge t_2^i \wedge ... \wedge t_{k_i}^i\right)$, $i = 1, 2, \ldots, m$. Therefore, it is straightforward to write the Boolean expression as the neural network of Figure 9.5.We see a shallow architecture however, the number m of gates in the hidden layer may be proportional to $n2^n$ as discussed in section 3.5.

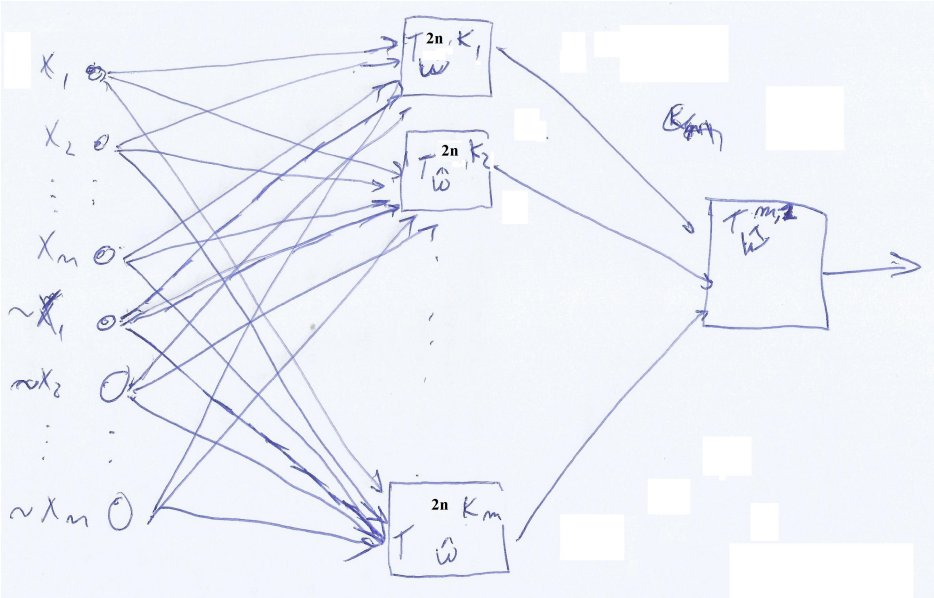


Figure 9.5: Shallow neural network computing expression (9.35).

An analogous result can be obtained using the CNF form of the Boolean function $TC^{n,\Delta}_{(\boldsymbol{w},1)} : B^n \longrightarrow \{0, 1\}$ (exercise). The threshold circuit in Figure 9.5 represents a shallow neural network with only one hidden layer. Given a Boolean function $F : B^n \longrightarrow B^1$ we can design a deeper neural network, with more than one hidden layer. The above development shows that we always can convert the deep architecture into a shallow one. However, the conversion is not efficient in the sense that the number gates in the hidden layer may growths exponentially with respect to n. Obviously, an analogous result can be obtained for the general case with $m > 1$.

9.7 Back-Propagation in Deep MLP Architectures

In this section we present the Back-Propagation algorithm whose goal is to efficiently compute the gradient with respect to the network weights, in order to

implement the gradient descent procedures in Algorithms 3 and 4. In this development, we suppose the labeled input set:

$$D = \{(\boldsymbol{x}_1, d_1), \ldots, (\boldsymbol{x}_N, d_N)\} \subset \mathbb{R}^{m+1}, \tag{9.38}$$

where $\boldsymbol{x}_i = (x_{1i}, x_{2i}, \ldots, x_{mi}) \in \mathbb{R}^m$, $i = 1, \ldots, N$ and $d_i \in \mathbb{R}$ is the desired response. Moreover, the objective function is given by the expression (8.23) with $\boldsymbol{z}$ replaced by $\boldsymbol{w}$ and $\mathbb{M}(\boldsymbol{x}_i; \boldsymbol{w}) \equiv o(\boldsymbol{x}_i; \boldsymbol{w})$:

$$\mathcal{L}(\boldsymbol{w}; \mathbb{B}) \equiv \frac{1}{|\mathbb{B}|} \sum_{i=1}^{|\mathbb{B}|} L(o(\boldsymbol{x}_i; \boldsymbol{w}) - d_i) + \sum_{\tau=1}^{K} \varsigma_\tau R_\tau(\boldsymbol{w}; \mathbb{B}), \tag{9.39}$$

where R_τ and d_i follows the definitions of section 8.1. Since the iteration t is implicit in expression (8.25), we discard this variable in the development.

Let us consider a MLP with $L \geq 1$ hidden layers [113] defined by the following elements.

1. Input Layer:

$$I = \left\{1, 2, \ldots, p^{(0)}\right\} \subset \mathbb{N}.$$

2. First Hidden Layer:

$$H^{(1)} = \left\{p^{(0)} + 1, p^{(0)} + 2, \ldots, p^{(1)}\right\} \subset \mathbb{N}.$$

3. Remaining Hidden Layers:

$$H^{(n)} = \left\{p^{(n-1)} + 1, p^{(n-1)} + 2, \ldots, p^{(n)}\right\} \subset \mathbb{N}, \quad 2 \leq n \leq L.$$

4. Output Layer:

$$Out = \left\{p^{(L)} + 1, p^{(L)} + 2, \ldots, p^{(Out)}\right\}.$$

5. Synaptic weight connecting the output of neuron i to the input of neuron j : w_{ij}

6. Bias associated to the neuron j : b_j

7. Network weights vector: $\boldsymbol{w} = (\{w_{ij}\}, \{b_j\})$

8. Activation function (see section 9.4)

In this development, we are going to consider $p^{(Out)} = p^{(L)} + 1$; that means, only one output node, corresponding to regression or classification network, as shown in Figure 9.6.

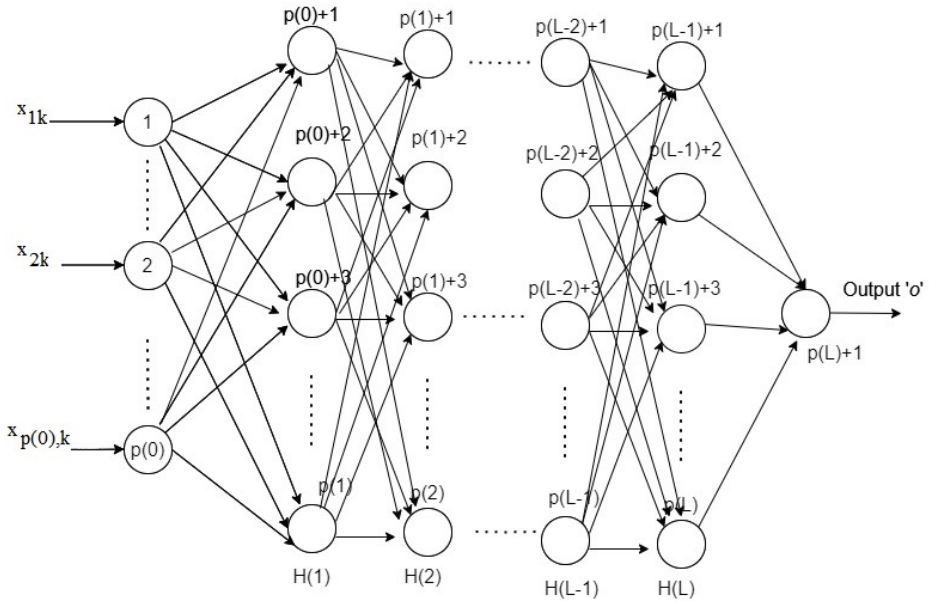


Figure 9.6: Deep MLP scheme for classification or regression with the sample $\boldsymbol{x}_k$ as input.

Hence, we shall define important network quantities as follows.

1. Induced local field at neuron $j \in H^{(1)}$ at input sample k:

$$net_j(k) = \sum_{i \in I} w_{ij} x_{ik} + b_j$$

2. Induced local field at neuron $j \in H^{(n)}$ due to input sample k:

$$net_j(k) = \sum_{i \in H^{(n-1)}} w_{ij} o_i(k) + b_j,$$

where $o_i(k) = f(net_i(k))$.

3. Induced local field due to input sample k at neuron $j \in Out$:

$$net_j(k) = \sum_{i \in H^{(L)}} w_{ij} o_i(k) + b_j. \tag{9.40}$$

4. Output of neuron $j \in Out$ due to input sample k:

$$o_j(k) = f(net_j(k)), \tag{9.41}$$

where f is the chosen activation function.

5. Loss function:

$$\mathcal{L}(\boldsymbol{w}; \mathbb{B}) \equiv \frac{1}{|\mathbb{B}|} \sum_{k=1}^{|\mathbb{B}|} \sum_{j \in Out} (o_j(k) - d_j(k))^2 + \sum_{\tau=1}^{K} \varsigma_\tau R_\tau(\boldsymbol{w}; \mathbb{B}), \tag{9.42}$$

Now, we can explicit the learning rule of Algorithm 4 using the above elements.

1. Update network weights. Following expression (8.25), we must use a gradient based method:

$$w_{ij} \leftarrow w_{ij} + \eta \left(-\frac{\partial \mathcal{L}(\boldsymbol{w}; \mathbb{B})}{\partial w_{ij}} \right),$$

$$b_j \leftarrow b_j + \eta \left(-\frac{\partial \mathcal{L}(\boldsymbol{w}; \mathbb{B})}{\partial b_j} \right),$$

which can be re-written as follows to simplify the notation:

$$w_{ij} \leftarrow w_{ij} + \eta \left(-\frac{\partial \mathcal{L}}{\partial w_{ij}} \right)_{\boldsymbol{w}=\boldsymbol{w}(t)},$$

$$b_j \leftarrow b_j + \eta \left(-\frac{\partial \mathcal{L}}{\partial b_j} \right)_{\boldsymbol{w}=\boldsymbol{w}(t)}.$$

where:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \sum_{k=1}^{|\mathbb{B}|} \sum_{\xi=p(0)+1}^{p(L)+1} \frac{\partial \mathcal{L}}{\partial net_\xi(k)} \frac{\partial net_\xi(k)}{\partial w_{ij}},$$

$$\frac{\partial \mathcal{L}}{\partial b_j} = \sum_{k=1}^{|\mathbb{B}|} \sum_{\xi=p(0)+1}^{p(L)+1} \frac{\partial \mathcal{L}}{\partial net_\xi(k)} \frac{\partial net_\xi}{\partial b_j}.$$

2. Cases to consider:

   (a) If $j \in H^{(1)}$we have:

$$\frac{\partial net_j(k)}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\left(\sum_{\xi \in I} w_{\xi j} x_{\xi k} + b_j\right) = x_{ik},$$

$$\frac{\partial net_j(k)}{\partial b_j} = 1.$$

   (b) If $j \in H^{(n)}$, $2 \leq n \leq L$, we have:

$$\frac{\partial net_j(k)}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\left(\sum_{\xi \in H^{(n-1)}} w_{\xi j} o_\xi(k) + b_j\right) = o_i(k),\ i \in H^{(n-1)}.$$

$$\frac{\partial net_j(k)}{\partial b_j} = 1.$$

   (c) If $j \in Out$:

$$\frac{\partial net_j(k)}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\left(\sum_{\xi \in H^{(L)}} w_{\xi j} o_\xi(k) + b_j\right) = o_i(k),\ i \in H^{(L)}.$$

$$\frac{\partial net_j(k)}{\partial b_j} = 1.$$

3. Let δ_j be:

$$\delta_j = -\frac{\partial \mathcal{L}}{\partial net_j}.$$

4. Then:

   (a) $j \in Out$:

$$\delta_j(k) = -\frac{\partial \mathcal{L}}{\partial net_j(k)}$$

$$= -\frac{\partial \mathcal{L}}{\partial o_j(k)} \frac{\partial o_j(k)}{\partial net_j(k)}$$

$$= \frac{2}{|\mathbb{B}|} f'(net_j(k)) \left[(d_j(k) - f(net_j(k))) \right]. \tag{9.43}$$

(b) $j \in H^{(L)}$:

$$\delta_j(k) = -\frac{\partial \mathcal{L}}{\partial net_j(k)}$$

$$= -f'(net_j(k)) \frac{\partial \mathcal{L}}{\partial o_j(k)}$$

$$= -f'(net_j(k)) \left[\sum_{\xi \in Out} \frac{\partial \mathcal{L}}{\partial net_\xi(k)} \frac{\partial net_\xi(k)}{\partial o_j(k)} \right]$$

$$= -f'(net_j(k)) \left[\sum_{\xi \in Out} \frac{\partial \mathcal{L}}{\partial net_\xi(k)} w_{j\xi} \right]. \tag{9.44}$$

(c) $j \in H^{(n)},\ 1 \leq n \leq L-1$:

$$\delta_j(k) = -\frac{\partial \mathcal{L}}{\partial net_j(k)} = -f'(net_j(k)) \frac{\partial \mathcal{L}}{\partial o_j(k)}$$

$$= -f'(net_j(k)) \left[\sum_{\xi \in H^{(\nu \geq n+1)}} \frac{\partial \mathcal{L}}{\partial net_\xi(k)} \frac{\partial net_\xi(k)}{\partial o_j(k)} \right]$$

5. Putting all together through back-propagation:

(a) $j \in Out$:

$$w_{ij} \longleftarrow w_{ij} + \eta \left(-\frac{\partial \mathcal{L}}{\partial w_{ij}} \right) = w_{ij} + \eta \sum_{k=1}^{|\mathbb{B}|} \left(-\frac{\partial \mathcal{L}}{\partial net_j(k)} \frac{\partial net_j(k)}{\partial w_{ij}} \right), \tag{9.45}$$

$$= w_{ij} + \eta \sum_{k=1}^{|\mathbb{B}|} \left(\frac{2}{|\mathbb{B}|} f'(net_j(k)) \left[(d_j(k) - f(net_j(k))) \right] \right) o_i(k), \quad i \in H^{(L)}. \tag{9.46}$$

(b) $j \in H^{(L)}$:

$$w_{ij} \longleftarrow w_{ij} + \eta \left(-\frac{\partial \mathcal{L}}{\partial w_{ij}} \right) = w_{ij} + \eta \sum_{k=1}^{|\mathbb{B}|} \left(-\frac{\partial \mathcal{L}}{\partial net_j\,(k)} \frac{\partial net_j\,(k)}{\partial w_{ij}} \right), \tag{9.47}$$

$$= w_{ij} + \eta \sum_{k=1}^{|\mathbb{B}|} \left(-f'(net_j\,(k)) \left[\sum_{\xi \in Out} \frac{\partial \mathcal{L}}{\partial net_\xi\,(k)} w_{j\xi} \right] \right) o_i\,(k) \tag{9.48}$$

$$= w_{ij} + \eta \sum_{k=1}^{|\mathbb{B}|} \delta_j\,(k)\, o_i\,(k)\,. \quad i \in H^{(L-1)}$$

(c) $j \in H^{(n)},\ 1 \leq n \leq L-1$:

$$w_{ij} \longleftarrow w_{ij} + \eta \left(-\frac{\partial \mathcal{L}}{\partial w_{ij}} \right) = w_{ij} + \eta \sum_{k=1}^{|\mathbb{B}|} \left(-\frac{\partial \mathcal{L}}{\partial net_j\,(k)} \frac{\partial net_j\,(k)}{\partial w_{ij}} \right)$$

$$= w_{ij} + \eta \sum_{k=1}^{|\mathbb{B}|} \left(-f'(net_j\,(k)) \left[\sum_{\xi \in H^{(\nu \geq n+1)}} \frac{\partial \mathcal{L}}{\partial net_\xi\,(k)} \frac{\partial net_\xi\,(k)}{\partial o_j\,(k)} \right] \right) o_i\,(k)$$

$$= w_{ij} + \eta \sum_{k=1}^{|\mathbb{B}|} \delta_j\,(k)\, o_i\,(k)\,, \quad i \in H^{(n-1)},$$

where $H^{(\nu \geq n+1)} = \cup_{v=n+1}^{Out} H^{(\nu)}$ and $H^{(0)} = I$ (with analogous expressions for b^j (exercise)).

If we consider the Logistic function 9.15 as activation function then, since:

$$o_j\,(k) = f\,(net_j\,(k))\,,$$

then:

$$f'(net_j\,(k)) = \alpha\,(1 - f\,(net_j\,(k)))\, f\,(net_j\,(k)) = \alpha\,(1 - o_j\,(k))\, o_j\,(k)\,,$$

consequently, we can compute the derivative $f'(net_j\,(k))$ by taking only the output of the neural network.

9.8 Steps of Back-propagation Algorithm

At each iteration t perform:

1. Forward-Backward Computation:

   (a) Forward Computation: For each $x_k \in \mathbb{B}$, pass it to the network and keep the induced local fields and outputs $net_j(k)$ and $o_j(k)$, $j \in I \cup \left(\cup_{v=1}^{L} H^{(\nu)}\right) \cup Out$;

   (b) Backward Computation:

   (c) $j \in Out$:

$$\Delta_{ij}^{k} = \left(\frac{2}{|\mathbb{B}|} f'(net_j(k)) \left[(d_j(k) - f(net_j(k)))\right]\right) o_i(k), \quad i \in H^{(L)}, \tag{9.49}$$

   (d) $j \in H^{(n)},\ n = L, L-1, \ldots, 1$:

$$\Delta_{ij}^{k} = \left(-f'(net_j(k)) \left[\sum_{\xi \in H^{(\nu \geq n+1)}} \frac{\partial \mathcal{L}}{\partial net_\xi(k)} \frac{\partial net_\xi(k)}{\partial o_j(k)}\right]\right) o_i(k), \quad i \in H^{(n-1)} \tag{9.50}$$

   We must notice that, for $n = L$ we need the derivatives

$$\frac{\partial \mathcal{L}}{\partial net_\xi(k)}, \quad \xi \in Out,$$

   that were computed according to expression (9.43). Also, for $n = L-1$ we need the derivatives:

$$\frac{\partial \mathcal{L}}{\partial net_\xi(k)}, \quad \xi \in H^{(L)},$$

   that were compute in the previous step according to equation (9.44).

2. Compute the derivatives respect to the weights:

$$-\frac{\partial \mathcal{L}}{\partial w_{ij}} = \sum_{k=1}^{|\mathbb{B}|} \Delta_{ij}^{k},$$

3. Update weights:

$$w_{ij} \longleftarrow w_{ij} + \eta \left(-\frac{\partial \mathcal{L}}{\partial w_{ij}} \right)$$

9.9 Momentum Term and Rate of Learning

At iteration t of the training stage, consider the rule:

$$w_{ij}(t+1) - w_{ij}(t) = \beta \left(w_{ij}(t) - w_{ij}(t-1) \right) + \eta \left(-\frac{\partial \mathcal{L}(t)}{\partial w_{ij}(t)} \right), \qquad (9.51)$$

where β is called the momentum constant.

If we denote: $\Delta w_{ij}(t) \equiv w_{ij}(t+1) - w_{ij}(t)$ and introduce this notation in expression (9.51) we obtain:

$$\Delta w_{ij}(t) = \beta \Delta w_{ij}(t-1) + \eta \left(-\frac{\partial \mathcal{L}(t)}{\partial w_{ij}(t)} \right).$$

Therefore, if $t = 3$, we get:

$$\Delta w_{ij}(3) = \beta \Delta w_{ij}(2) + \eta \left(-\frac{\partial \mathcal{L}(3)}{\partial w_{ij}(3)} \right)$$

$$= \beta \left[\beta \Delta w_{ij}(1) + \eta \left(-\frac{\partial \mathcal{L}(2)}{\partial w_{ij}(2)} \right) \right] + \eta \left(-\frac{\partial \mathcal{L}(3)}{\partial w_{ij}(3)} \right)$$

$$= \beta \left[\beta \left(\beta \Delta w_{ij}(0) + \eta \left(-\frac{\partial \mathcal{L}(1)}{\partial w_{ij}(1)} \right) \right) + \eta \left(-\frac{\partial \mathcal{L}(2)}{\partial w_{ij}(2)} \right) \right] + \eta \left(-\frac{\partial \mathcal{L}(3)}{\partial w_{ij}(3)} \right)$$

$$= \beta^3 \Delta w_{ij}(0) + \beta^2 \eta \left(-\frac{\partial \mathcal{L}(1)}{\partial w_{ij}(1)} \right) + \beta \eta \left(-\frac{\partial \mathcal{L}(2)}{\partial w_{ij}(2)} \right) + \eta \left(-\frac{\partial \mathcal{L}(3)}{\partial w_{ij}(3)} \right)$$

$$= \beta^3 \Delta w_{ij}(0) - \eta \sum_{\xi=1}^{3} \beta^{3-\xi} \frac{\partial \mathcal{L}(\xi)}{\partial w_{ij}(\xi)}$$

By induction we can show that:

$$\Delta w_{ij}(t) = \beta^t \Delta w_{ij}(0) - \eta \sum_{\xi=1}^{t} \beta^{t-\xi} \frac{\partial \mathcal{L}(\xi)}{\partial w_{ij}(\xi)}, \tag{9.52}$$

or, using the definition of $\Delta w_{ij}(t)$, we obtain:

$$w_{ij}(t+1) = w_{ij}(t) + \beta^t \Delta w_{ij}(0) - \eta \sum_{\xi=1}^{t} \beta^{t-\xi} \frac{\partial \mathcal{L}(\xi)}{\partial w_{ij}(\xi)}. \tag{9.53}$$

This time series deserves some considerations. Firstly, in order to avoid convergence problems, we should restrict the momentum constant β in the interval $0 \leq \beta < 1$. If we set $\beta = 0$ then the back-propagation algorithm works without momentum. Also, if the partial derivatives in expression (9.52) presents the same algebraic sign on consecutive iterations then the exponentially weighted sum increases and, consequently, the weight w_{ij} in expression (9.53) is adjusted by a large amount. Hence, in this case, we can say that the inclusion of momentum induces an acceleration of the convergence to a critical point of the loss function [113].

On the other hand, if the derivative in relation (9.53) interchanges positive and negative algebraic signs then the exponentially weighted sum shrinks in magnitude and the weight w_{ij} undergoes a little change [113]. So, in this scenario, the momentum has a stabilizing effect in regions where the sign of the derivative oscillates.

9.10 Weights and Bias Computation and Back-propagation

When implementing the learning machine in Algorithms 4 and 3 through neural networks we apply the back-propagation technique to efficiently compute the gradient respect to the machine parameters. So, we have an iterative algorithm that, update weights (and bias) at each iteration t. The input data, given in expression (9.38), must be analysed and processed following the theory described in Chapter 7. Specifically, after Data collection; Data Exploration by looking for outliers,

finding exceptions, and missing infomation; Cleaning by remotion of incorrect and inconsistent samples; we can perform Data Normalization and start training our neural network as follows

1. Data Normalization: If we apply the Logistic function (equation (9.15)) as activation function then we must normalize the input data to fit the interval $[0, 1]$. This can be achieved therough the operations:

$$\boldsymbol{x}_{\min} = (\min\{x_{1i},\ i = 1, 2, \ldots, N\}, , \ldots, \min\{x_{mi},\ i = 1, 2, \ldots, N\}) \in \mathbb{R}^m,$$

$$\boldsymbol{x}_{\max} = (\max\{x_{1i},\ i = 1, 2, \ldots, N\}, , \ldots, \max\{x_{mi},\ i = 1, 2, \ldots, N\}) \in \mathbb{R}^m,$$

$$\tilde{x}_{uv} = \frac{x_{uv} - x_{u,\min}}{x_{u,\max} - x_{u,\min}},\ u = 1, 2, \ldots, m,\ v = 1, 2, \ldots, N. \tag{9.54}$$

   In the case of the hyperbolic tangent activation function (expresson (9.17)), the normalization procedure should have image in the range$[-1, 1]$. To perform this, we shall compute:

$$\tilde{x}_{uv} = -1 + 2\left(\frac{x_{uv} - x_{u,\min}}{x_{u,\max} - x_{u,\min}}\right),\ u = 1, 2, \ldots, m,\ v = 1, 2, \ldots, N. \tag{9.55}$$

   For others possibilities for data normalization see the section 7.9 and the batch normalization technique [121].

2. Initialization: Weights and bias of the perceptrons in the neural network must be initialized in order to instantiate the iterative process. We have the following possibilities in this step:

   (a) For each layer l Initialize the biases to be zero $\left(b_j^{(0)} = 0\right)$ and the weights $w_{ij}^{(0)}$as:

$$w_{ij}^{(0)} \sim U\left[-\frac{1}{\sqrt{n_{l-1}}}, \frac{1}{\sqrt{n_{l-1}}}\right], \tag{9.56}$$

where $U[-a, a]$ is the uniform distribution in the range $(-a, a)$ and n_{l-1} is the number of neurons of the layer $l-1$, which, following Figure 9.4, is given by [120]:

$$n_{l-1} = p(l-1) - (p(l-2)+1). \tag{9.57}$$

(b) For each layer l Initialize the biases to be zero $\left(b_j^{(0)} = 0\right)$ and the weights $w_{ij}^{(0)}$ as [120]:

$$w_{ij}^{(0)} \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_{l-1}+n_l}}, \frac{\sqrt{6}}{\sqrt{n_{l-1}+n_l}}\right], \tag{9.58}$$

where U is an uniform distributions likewise in the previous item, $n_{l-1}$is computed by expression (9.57), and $n_l = p(l) - (p(l-1)+1)$ according to Figure 9.4 also.

3. At each iteration t of Algorithm 4 perform:

4. Forward-Backward Computation:

(a) Forward Computation: For each $x_k \in \mathbb{B}$, pass it to the network and keep the induced local fields and outputs $net_j(k)$ and $o_j(k)$, $j \in \cup_{v=0}^{Out} H^{(\nu)}$;

(b) Backward Computation:

i. 1.2.1) $j \in Out$:

$$\Delta_{ij}^k = \left(\frac{2}{|\mathbb{B}|} f'(net_j(k))\left[(d_j(k) - f(net_j(k)))\right]\right) o_i(k), \quad i \in H^{(L)}, \tag{9.59}$$

ii. $j \in H^{(n)},\ n = L, L-1, \ldots, 1$:

$$\Delta_{ij}^k = \left(-f'(net_j(k))\left[\sum_{\xi \in H^{(\nu \geq n+1)}} \frac{\partial \mathcal{L}}{\partial net_\xi(k)} \frac{\partial net_\xi(k)}{\partial o_j(k)}\right]\right) o_i(k), \quad i \in H^{(n-1)} \tag{9.60}$$

We must notice that, for $n = L$ we need the derivatives

$$\frac{\partial \mathcal{L}}{\partial net_{\xi}(k)}, \quad \xi \in Out,$$

that were computed according to expression (9.43). Also, for $n = L - 1$ we need the derivatives:

$$\frac{\partial \mathcal{L}}{\partial net_{\xi}(k)}, \quad \xi \in L,$$

that were compute in the previous step according to equation (9.44).

5. Compute the derivatives respect to the weights:

$$-\frac{\partial \mathcal{L}}{\partial w_{ij}} = \sum_{k=1}^{|\mathbb{B}|} \Delta_{ij}^{k},$$

6. Update weights:

$$w_{ij} \longleftarrow w_{ij} + \eta \left(-\frac{\partial \mathcal{L}}{\partial w_{ij}} \right)$$

9.11 Exercises

1. Pereceptron [113, 11]:

   (a) Generate a database $S \subset \mathbb{R}^2 \times \{+1, -1\}$, with $|S| = N$ composed by two classes C_1 and C_2.

   (b) Partitioning S into disjoint sets: training $\mathbb{D}_{tr}$, validation $\mathbb{D}_{val}$, and test $\mathbb{D}_{te}$ sets.

   (c) Implement the perceptron model (Figure 9.2) for classification in $\mathbb{R}^2 \times \{+1, -1\}$ and perform training, validation and test steps. In the latter, compute the accuracy given by expression (8.33). Show some graphical configurations of the line that partitions the pattern space together with the final solution.

2. Take the FEI database [122, 123].

(a) Define the classes C_1 and C_2 with $|C_1| = N_1$ and $|C_2| = N_2$ and $N = N_1 + N_2$.

(b) Convert each image to gray scale

(c) Feature extraction: Compute the relative frequency histogram, denoted by $\boldsymbol{x}$, for each image. Generate the feature space $S = \{(\boldsymbol{x}_i, d_i), \quad i = 1, 2, ..., N\} \subset \mathbb{R}^{256} \times \{+1, -1\}$.

(d) Partitioning S into disjoint sets: training $\mathbb{D}_{tr}$, validation $\mathbb{D}_{val}$, and test $\mathbb{D}_{te}$ sets.

(e) Train a perceptron model and compute the accuracy after training

3. Show that perceptron can not compute the XOR gate.

4. Generalize the proof in section 9.2 for the learning rule in expression (9.2) with $0 < \eta < 1$.

5. Study the influence of the different types of activation functions of section 9.4 in the perceptron model.

6. According to [124], *Overtraining refers to training a network past the point where it generalizes best*. Hence, consider the function used in [125]:

$$f(x) = -\cos(x) + \nu, \quad 0 \leq x < \pi,$$

$$f(x) = \cos(3(x - \pi)) + \nu,$$

where ν is a uniformly distributed random variable in the interval $[-0.25, 0.25]$. Take a discretization of that function to generate the dataset $\mathbb{D} = \{(x_i, f(x_i)), i = 1, 2, \ldots N\}$ and subdivide the set $\mathbb{D}$ into $\mathbb{D}_{tr}$, $\mathbb{D}_{val}$, ans $\mathbb{D}_{te}$. Design a Loss function and train a MLP for data interpolation aiming to identify overtraining. Use 1-fold cross-validation for simplicity.

7. Take a discretization of the sphere $S^2 = \left\{\boldsymbol{x} \in \mathbb{R}^3; \|\boldsymbol{x}\|_2^2 = 1\right\}$ nearby the point $\boldsymbol{p} = \left(1/\sqrt{3}, 1/\sqrt{3}, 1/\sqrt{3}\right)$ to get a set $\mathbb{D} = \{(x_i, y_i, z_i), i = 1, 2, \ldots N\} \subset S^2$. Subdivide the set $\mathbb{D}$ into $\mathbb{D}_{tr}$, $\mathbb{D}_{val}$, and $\mathbb{D}_{te}$. Build a Loss function and train a MLP architecture to compute the

implicit function $z = \sqrt{1 - x^2 - y^2}$ nearby the point $\boldsymbol{p}$. For simplicity, use the 1-fold cross validation (Figure 8.5) for training the network. Take care about the activation function and try more than one type.

8. Return to exercise 7 and include new points perturbed with Gaussian noise to the training set $\mathbb{D}_{tr}$ to generate a new set $\widehat{\mathbb{D}}_{tr}$. Keep the validation set $\mathbb{D}_{val}$ and test set $\mathbb{D}_{te}$ unchanged and re-train the model from the begining. Keep the same kind of Loss function. Plot the training and validation curves to identify overffiting or overtraining.

9. The concept of overfitting can also be understood by thinking about polynomial interpolation tasks (see Fig. 1 of reference [126]). In this case, the polynomial coefficients are the degree of fredom, while in a neural network the weights control the model flexibility. Return to problem 6 and train a network with the goal of producing overfitting.

10. Consider the sets $\widehat{\mathbb{D}}_{tr}$ and $\mathbb{D}_{val}$ of exercise 6. Train a larger MLP architecture (increase the number of neurons and/or layers). Plot the training and validation curves to identify overffiting.

11. Take the FEI image database and compute the PCA for dimensionality reduction. Use the procedure explained to compute Figure 8.7 to identify curse of dimensionality using the principal components sorted following the decreasing order of the eigenvalues, say $\overline{\mathbf{p}}_1, \overline{\mathbf{p}}_2, ..., \overline{\mathbf{p}}_n$. Hence, buind a network whose input layer has n nodes and the output layer just one node. Train this network in the PCA subspace spanned by $\overline{\mathbf{p}}_1$ and plot the point $\left(1, Err\left(\boldsymbol{z}^{\star}_{opt}\left(1\right), \mathbb{D}_{tr}\right)\right)$. Then, repeate this procedure for components $\overline{\mathbf{p}}_1$, $\overline{\mathbf{p}}_2$, and so on.

12. Generate a set of points in $\mathbb{R}^2$ with two classes and with the same spatial distribution observed in Figure 1 of reference [126]. Train a perceptron and a MLP whose corresponding decision boudary is the dashed curve. Study the overfitting problem in this case and compare the performance of both classifiers using the recognition rate given by expression (8.33).

13. Consider the dimension d identified in exercise 11. Train the classifier using 3-fold cross-validation but, in this case, start the training process using

the weights already obtained in exercise 11. Then, compute the confusion matrix (section 8.6) , the precision-recall and Receiver operating characteristics (ROC) curves (see reference [127]).

14. Transfer learning reffers to the process of using what has been learned in one setting (or domain) to improve generalizations in another setting [93] . So, chose another regular surface and repeat the steps of exercise 7 but using the weights already computed there to initialize the neural network.

15. Data Augmentation is a suite of methods to add new samples to the input set or enhance the quality of training datasets in order to improve the model generalization [128]. Consider the PCA subspace with dimension d identified in exercise 11. Take the principal component $\overline{\mathbf{p}}_j$ and perform data augmentation in the PCA space through the expression:

$$I\left(i,j\right) = \widehat{\mathbf{x}} + \delta_{i,j} \cdot \overline{\mathbf{p}}_j, \tag{9.61}$$

where $\widehat{\mathbf{x}}$ is the global mean:

$$\widehat{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i, \tag{9.62}$$

and $\delta_{i,j} \in \{\pm i \cdot \lambda_j^{0.5},\ i = 0,1,2,3\}$, and λ_j is the eigenvalue associated to $\overline{\mathbf{p}}_j$

Chapter 10

Statistical Learning Theory

The main goal of this chapter is to present fundamental aspects in supervised and unsupervised statistical learning for image analysis. The basic pipeline that steers our presentation is: (a) Dimensionality reduction; (b) Discriminant analysis; (c) Choose a learning method to compute a separating hypersurface, that is, to solve the classification problem; (d) Reconstruction problem, that means, to consider how good a low dimensional representation will look like.

As already pointed out in the introduction (Chapter 1), both statistical learning and machine learning are data dependent. However, statistical learning models are yielded based on the assumption that data has certain regularity, such as linearity, normality and independence. Consequently, we can process data using methods steered by those beliefs [13]. Machine learning models, in general, do not explicitly consider such assumptions and in most of the cases ignore them [14].

Like machine learning, statistical learning also explores ways of estimating functional dependency from a given collection of data [22]. It covers important topics in classical statistics such as discriminant analysis, regression methods, and the density estimation problem [129, 130, 131]. Moreover, pattern recognition and classification can be seen from the viewpoint of the general statistical problem of function estimation from empirical data. Therefore, they fit very well in the statistical learning framework [132, 61].

The analysis of methods of statistical inference began with the remarkable works of Fisher (parametric statistics) and the theoretical results of Glivenko and Cantelli (convergence of the empirical distribution to the actual one) and Kol-

mogorov (the asynptotically rate of that convergence). These events determined two approaches to statistical inference: The particular (*parametric*) inference and the *general* inference [7].

The parametric inference aims to create statistical methods for solving particular problems. Regression analysis is a know technique in this class. On the other hand, the general inference aims to find one induction method that can be applied for any statistical inference problem. Learning machines, like perceptron (section 9.1), and statistical learning methods like support vector machine (SVM) [7, 12], and the linear discriminant analysis (LDA) are nice examples in this area.

When performing a database analysis a common practice is to consider each data point as a point in a n-dimensional space, where n is the number of involved variables. In general, n is very large, a known problem in image analysis, for example. Therefore, dimensionality reduction may be necessary in order to discard redundancy and simplify further operations. The most know technique in this subject is the Principal Components Analysis (PCA) [61]. The PCA technique is a way to implement the item (a) of our pipeline. Following this line, once performed dimensionality reduction through PCA, we must determine the most important discriminant PCA features for a pattern recognition goals, like classification (item (b)). Discriminant analysis techniques, like LDA, can be used to solve this task. Next, the fundamental point is which learning method to use to solve the classification problem in the item (c). Besides neural networks discussed in Chapter 11, and many other possibilities [44, 22, 3], we can choose the SVM model in this step. Then, the step (d) deals with the visualization of data that was projected in the reduced PCA space.

In what follows we first review PCA technique in section 10.1. Next, in section 10.3, we describe the SVM classifier. Then, we consider two discriminant analysis techniques: the discriminant principal component analysis (DPCA) and the LDA, summarized in sections 10.4 and 10.5, respectively. The kernel version of the PCA method is developed in section 10.6. Then, we consider classification and reconstruction from the viewpoint of SVM and LDA methods (section 10.7).

10.1 Principal Components Analysis

In this section we review results in principal components analysis (**PCA**). When applying PCA for dimensionality reduction before classification, it has been common practice to use components with the largest eigenvalues. Such idea can be justified if we clarify the main principles of the PCA analysis. For completeness, we will review some points of this theory in this section first.

Principal Component Analysis (PCA), also called Karhunen-Loeve or KL method, can be seen as a method for data compression or dimensionality reduction [133] (see [5], section 5.11 also). Thus, let us suppose that the data to be compressed consists of N measurements or data samples, from the n-dimensional space $\mathbb{R}^n$. Then, PCA searches for m orthonormal vectors in $\mathbb{R}^n$ the that can best represent the data, where $m \leq n$. Thus, let $S = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$ be the data set and its centroid given by the global mean:

$$\widehat{\mathbf{x}} = \frac{1}{N} \sum_{j=1}^{N} \mathbf{x}_j. \tag{10.1}$$

To address the issue of compression, we need a projection basis that satisfies a proper optimization criterion. Following [5], consider the operations in Figure 10.1 where:

$$\mathbf{u}_j = \mathbf{x}_j - \widehat{\mathbf{x}}. \tag{10.2}$$

The (centered) vector $\mathbf{u}_j$ is first transformed to a vector $\mathbf{v}_j$ by the transformation matrix A. Thus, we truncate $\mathbf{v}_j$ by choosing the first m elements of $\mathbf{v}_j$. The obtained vector $\mathbf{w}_j$ is just the transformation of $\mathbf{v}_j$ by I_m, that is a matrix with 1's along the first m diagonal elements and zeros elsewhere. Finally, $\mathbf{w}_j$ is transformed to $\mathbf{z}_j$ by the matrix B. Let the square error be defined as follows:

$$J_m = \frac{1}{N} \sum_{j=1}^{N} \|\mathbf{u}_j - \mathbf{z}_j\|^2 = \frac{1}{N} Tr \left[\sum_{j=1}^{N} (\mathbf{u}_j - \mathbf{z}_j) (\mathbf{u}_j - \mathbf{z}_j)^T \right], \tag{10.3}$$

where Tr means the trace of the matrix between the square brackets and the notation (T) means the transpose of a matrix. Following Figure 10.1, we observe that $\mathbf{z}_j = BI_mA\mathbf{u}_j$. Thus we can rewrite (10.3) as:

$$J_m = \frac{1}{N} Tr \left[\sum_{j=1}^{N} (\mathbf{u}_j - BI_mA\mathbf{u}_j) (\mathbf{u}_j - BI_mA\mathbf{u}_j)^T \right], \tag{10.4}$$

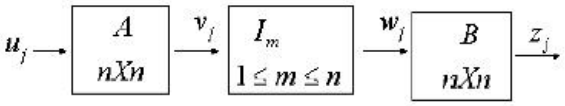


Figure 10.1: KL Transform formulation. Reprinted from [5].

which yields

$$J_m = \frac{1}{N} Tr \left[(I - BI_m A) \, R \, (I - BI_m A)^T \right], \tag{10.5}$$

where

$$R = \sum_{j=1}^{N} (\mathbf{x}_j - \widehat{\mathbf{x}})(\mathbf{x}_j - \widehat{\mathbf{x}})^T. \tag{10.6}$$

Following the literature, we call R the covariance matrix. We can now state the optimization problem by saying that we want to find the matrices A, B that minimizes J_m. The next theorem gives the solution for this problem.

Theorem 1: The error J_m in expression (10.5) is minimum when:

$$A = \Phi^T, \quad B = \Phi, \quad AB = BA = I, \tag{10.7}$$

where Φ is the matrix obtained by the orthonormalized eigenvectors of R arranged according to the decreasing order of its eigenvalues. *Proof.* See [5].

Therefore, from this result, if our aim is data compression than we must choose the components with the largest eigenvalues. Hence, we must choose the set of $m \leq n$ eigenvectors of R, which corresponds to the m largest eigenvalues. Such a set of eigenvectors that defines a new uncorrelated coordinate system for the centered training matrix $\Theta = [\mathbf{u}_1^T \mathbf{u}_2^T \cdots \mathbf{u}_N^T]$ is known as the principal components. In the context of face recognition, those $P_{pca} = [\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_m]$ components are frequently called eigenfaces [134]. The PCA methodology is summarized in the Algorithm 5.

After computing matrix P_{pca} we can project each centered sample $\mathbf{u}_i$ in the reduced PCA space through the expression:

$$\overline{\mathbf{u}}_i = (P_{pca})^T \mathbf{u}_i. \tag{10.10}$$

Algorithmus 5 Procedure to compute PCA matrix.

1: Input: $m \leq n$, dataset $S = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\} \subset \mathbb{R}^n$

2: Compute the global mean $\widehat{\mathbf{x}}$ of the input data through expression (10.2);

3: Centering input samples: $\mathbf{u}_i = \mathbf{x}_i - \widehat{\mathbf{x}}$;

4: Compute the covariance matrix:

$$R = \sum_{i=1}^{N} (\mathbf{u}_i)(\mathbf{u}_i)^T. \tag{10.8}$$

5: Compute orthonormalized eigenvectors of R arranged according to the decreasing order of its eigenvalues: $\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_n$

6: Output: Matrix formed by the $m \leq n$ principal eigenvectors

$$P_{pca} = [\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_m]. \tag{10.9}$$

10.2 Changing Criterion

However, we can change the *information measure* in order to get another criterion. This is performed in [135] by using the following idea. Let $\mathbf{u}$ be a n-dimensional random vector with a mixture of two normal distributions with means μ_1 and μ_2, mixing proportions of p and $(1-p)$, respectively, and a common covariance matrix Σ. Let Δ denote the Mahalanobis distance between the two sub-populations, and R be the covariance matrix of $\mathbf{u}$. Then, it can be shown that [135]:

$$R = p(1-p)\mathbf{d}\mathbf{d}^T + \Sigma, \tag{10.11}$$

where $\mathbf{d} = \mu_1 - \mu_2$.

In [135], instead of the optimization criterion of choosing the components that minimizes J_m, it is assumed that the effectiveness of using a set of variables can be measured by the Mahalanobis distance computed on the basis of these variables. This distance will be called the information contained in the variables.

So, let $\mathbf{a}_i, i = 1, ..., n$ be the eigenvectors of R with eigenvalues $\lambda_1, \lambda_1, , ..., \lambda_n$, not necessarily sorted in decreasing order. For a given $B_m = (\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_m)$, $m \leq n$ we denote Δ_m the distance between the sub-populations using $B_m\mathbf{u}$. For instance, for $m = 1$, we can write:

$$\mathbf{a}_1^T R \mathbf{a}_1 = p(1-p)\mathbf{a}_1^T \mathbf{d}\mathbf{d}^T \mathbf{a}_1 + \mathbf{a}_1^T \Sigma \mathbf{a}_1. \tag{10.12}$$

Therefore, using the fact that $R\mathbf{a}_1 = \lambda_1 \mathbf{a}_1$, the expression (10.12) becomes:

$$\lambda_1 = p(1-p)\left(\mathbf{a}_1^T \mathbf{d}\right)^2 + \mathbf{a}_1^T \Sigma \mathbf{a}_1. \tag{10.13}$$

So, we find that:

$$\mathbf{a}_1^T \Sigma \mathbf{a}_1 = \lambda_1 - p(1-p)\left(\mathbf{a}_1^T \mathbf{d}\right)^2. \tag{10.14}$$

But, the Mahalanobis distance computed in the variable defined by the component $\mathbf{a}_1$ is:

$$\Delta_1^2 = \mathbf{d}^T \mathbf{a}_1 \left(\mathbf{a}_1^T \Sigma \mathbf{a}_1\right) \mathbf{a}_1^T \mathbf{d} = \frac{\left(\mathbf{a}_1^T \mathbf{d}\right)^2}{\mathbf{a}_1^T \Sigma \mathbf{a}_1},$$

which can be rewritten as:

$$\Delta_1^2 = \frac{\left(\mathbf{a}_1^T \mathbf{d}\right)^2}{\lambda_1 - p(1-p)\left(\mathbf{a}_1^T \mathbf{d}\right)^2}, \tag{10.15}$$

by using the equation (10.14). Finally, if we divide the numerator and denominator of equation (10.15) by λ_1 we obtain:

$$\Delta_1^2 = \frac{\frac{\left(\mathbf{a}_1^T \mathbf{d}\right)^2}{\lambda_1}}{1 - p(1-p)\frac{\left(\mathbf{a}_1^T \mathbf{d}\right)^2}{\lambda_1}}., \tag{10.16}$$

This result can be generalized for $m > 1$ by the following result [135]:

$$\Delta_m = \frac{\sum_{i=0}^{n-1} \frac{\left(\mathbf{a}_i^T \mathbf{d}\right)^2}{\lambda_i}}{1 - p(1-p)\sum_{i=0}^{n-1} \frac{\left(\mathbf{a}_i^T \mathbf{d}\right)^2}{\lambda_i}}. \tag{10.17}$$

By now, the following consequence may be draw: The component with the largest amount of information is not necessarily the one with largest eigenvalue. This is because Δ_i is a monotonic function of $\left(\mathbf{a}_i^T \mathbf{d}\right)^2 / \lambda_i$ instead of λ_i (see Δ_1 in equation (10.16)). The best subset of m principal components is the one with the largest Δ_m.

The relationship between this criterion and the last one that selects principal components in decreasing order of eigenvalues is obtained by demonstrating that, the information is distributed in m (or less than m) principal components if Σ has m distinct eigenvalues. In another words, the use of traditional PCA is justified when the information is concentrated in a few principal components with a large sample size.

10.3 Support Vector Machines

In section 9.1 we have discussed the perceptron model, which in turn is a linear classifier. In this section we will present another type of linear classifier, the support vector machine (SVM), based on a separating hyperplane with optimality properties. So, given a dataset:

$$S = \{(\mathbf{x}_i, y_i)\,,\ \mathbf{x}_i \in \mathbb{R}^n, \quad y_i \in \{-1, 1\}\,,\ i = 1, 2, \ldots, m\}\,, \tag{10.18}$$

we say that the subset I for which $y = 1$ and the subset II for which $y = -1$ are separable by the hyperplane:

$$\mathbf{x} * \phi = c, \tag{10.19}$$

if there exists both a unit vector ϕ ($|\phi| = 1$) and a constant c such that the inequalities:

$$\mathbf{x}_i * \phi > c, \quad \mathbf{x}_i \in I, \tag{10.20}$$

$$\mathbf{x}_j * \phi < c, \quad \mathbf{x}_j \in II, \tag{10.21}$$

hold true ("$*$" denotes the usual inner product in $\mathbb{R}^n$). Besides, let us define for any unit vector ϕ the two values:

$$c_1(\phi) = \min_{\mathbf{x}_i \in I} (\mathbf{x}_i * \phi)\,, \tag{10.22}$$

$$c_2(\phi) = \max_{\mathbf{x}_j \in II} (\mathbf{x}_j * \phi)\,. \tag{10.23}$$

The Figure 10.2 represents the dataset and the hyperplanes defined by ϕ and the values c_1, c_2 defined in expressions (10.22)-(10.23):

In this figure the points P_1 and P_2 give the solutions of problems (10.22)-(10.23), respectively, and the planes π_1 and π_2 are defined by:

$$\mathbf{x} * \phi = c_1, \tag{10.24}$$

$$\mathbf{x} * \phi = c_2. \tag{10.25}$$

Now, let us consider the plane π, parallel to π_1,π_2, with the property:

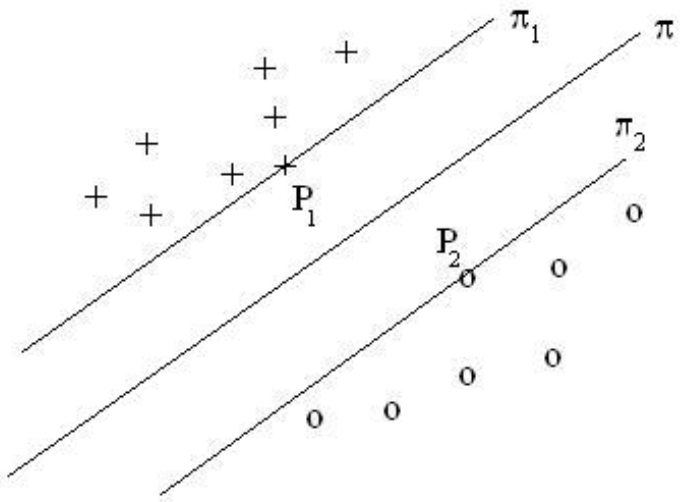


Figure 10.2: Separating hyperplane π and its offsets π_1, π_2.

$$d_\pi\left(P_1\right) = d_\pi\left(P_2\right), \tag{10.26}$$

where $d_\pi\left(P\right)$ means the Euclidean distance from a point P to a plane π. This plane is the hyperplane that separates the subsets with maximal margin. Expression (10.26) can be written as:

$$\left|\frac{P_1 * \phi - c}{|\phi|}\right| = \left|\frac{P_2 * \phi - c}{|\phi|}\right|. \tag{10.27}$$

If we suppose $P_1 * \phi - c \geq 0$ then we have $P_2 * \phi - c \leq 0$. So, by remembering that $|\phi| = 1$, the expression (10.27) becomes:

$$\left(P_1 * \phi - c\right) + \left(P_2 * \phi - c\right) = 0,$$

then, by using expressions (10.24)-(10.25) we finally obtain:

$$c = \frac{c_1\left(\phi\right) + c_2\left(\phi\right)}{2}. \tag{10.28}$$

Besides, let us call the $d_{\pi_1}\left(\pi_2\right)$ the distance between the planes π_1 and π_2, which can be computed through the distance between the point P_1 and the plane π_2, given by:

$$d_{\pi_1}(\pi_2) \equiv d_{\pi_2}(P_1) = \frac{(P_1 * \phi - c_2)}{|\phi|}, \tag{10.29}$$

By using expression (10.24) and the fact that $|\phi| = 1$, this equation becomes:ppp

$$d_{\pi_1}(\pi_2) = c_1 - c_2. \tag{10.30}$$

We call the *maximum margin hyperplane* or the *optimal hyperplane* the one, defined by the unit vector ϕ that maximizes the function:

$$\rho(\phi) = \frac{c_1(\phi) - c_2(\phi)}{2}, \tag{10.31}$$

$$|\phi| = 1. \tag{10.32}$$

The corresponding separating plane π has a constant c given by equation (10.28).

Now, let us consider another version of the optimization problem above. Let us consider a vector ψ such that $\psi/|\psi| = \phi$. So, equations (10.24)-(10.25) become:

$$\mathbf{x}_i * \psi > |\psi| c_1, \quad \mathbf{x}_i \in I, \tag{10.33}$$

$$\mathbf{x}_j * \psi < |\psi| c_2, \quad \mathbf{x}_j \in II. \tag{10.34}$$

Let us suppose that there is a constant b_0 such that $|\psi| c_1 \geq 1 - b_0$ and $|\psi| c_2 \leq -1 - b_0$. Then, we can rewrite expressions (10.33)-(10.34) as:

$$\mathbf{x}_i * \psi + b_0 \geq 1, \quad y_i = 1, \tag{10.35}$$

$$\mathbf{x}_j * \psi + b_0 \leq -1, \quad y_j = -1. \tag{10.36}$$

To understand the meaning of b_0 it is just a matter of using the fact that the equality in (10.35) holds true for P_1 and the equality in (10.36) is true for P_2. Therefore, it is straightforward to show that:

$$b_0 = -|\psi| \left(\frac{c_1(\phi) + c_2(\phi)}{2} \right) = -|\psi| c. \tag{10.37}$$

So, by substituting this equation in expressions (10.35)-(10.36) one obtains:

$$\mathbf{x}_i * \phi \geq c + \frac{1}{|\psi|}, \quad y_i = 1, \tag{10.38}$$

$$\mathbf{x}_j * \phi \leq c - \frac{1}{|\psi|}, \quad y_j = -1. \tag{10.39}$$

These expressions mean that we suppose that we can relax the constant c through the value $(1/\left|\psi\right|)$ without loosing the separating property. But, the vector ψ is not an unit one. Therefore the distance (10.29) can be obtained by:

$$d_{\pi_1}(\pi_2) = \frac{(1 - b_0) - (-b_0 - 1)}{|\psi|} = \frac{2}{|\psi|}. \tag{10.40}$$

In order to maximize this distance (and also maximize the function $\rho(\phi)$ in equation (10.31)) we must minimize the denominator in expression (10.40). So, we get an equivalent statement to define the optimal hyperplane: Find a vector ψ_0 and a constant (threshold) b_0 such that they satisfy the constraints:

$$\mathbf{x}_i * \psi_0 + b_0 \geq 1, \quad y_i = 1, \tag{10.41}$$

$$\mathbf{x}_j * \psi_0 + b_0 \leq -1, \quad y_j = -1. \tag{10.42}$$

and the vector ψ has the smallest norm:

$$|\psi| = \psi * \psi. \tag{10.43}$$

We shall simplify the notation by rewriting the constraints (10.41)-(10.42) in the equivalent form:

$$y_i (x_i * \psi_0 + b_0) \geq 1, \quad i = 1, 2, ..., m. \tag{10.44}$$

In order to solve the quadratic optimization problem stated above, it is used in [7] the Kuhn-Tucker theorem (Appendix B), which generalizes the Lagrange multipliers for convex optimization. The corresponding Lagrange function is:

$$L(\psi, b, \alpha) = \frac{1}{2}\psi * \psi - \sum_{i=1}^{m} \alpha_i \left(y_i \left((\mathbf{x}_i * \psi_0) + b_0\right) - 1\right), \tag{10.45}$$

where α_i are the Lagrange multipliers. Following the usual optimization theory (see sections 5.8 and 5.9), the minimum points of this functional must satisfy the conditions:

$$\frac{\partial L}{\partial \psi} = \psi - \sum_{i=1}^{m} y_i \alpha_i \mathbf{x}_i = 0, \tag{10.46}$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{m} y_i \alpha_i = 0. \tag{10.47}$$

If we substitute (10.46) into the functional (10.45) and take into account the result (10.47) we finally render the following objective function:

$$W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j (\mathbf{x}_i * \mathbf{x}_j). \tag{10.48}$$

We must maximize this expression in the non-negative quadrant $\alpha_i \geq 0$, $i = 1, 2, ..., m$, under the constraint (10.47) . In [7] it is demonstrate that the desired solution is given by:

$$\psi_0 = \sum_{i=1}^{m} y_i \alpha_i \mathbf{x}_i, \tag{10.49}$$

$$b_0 = \max_{|\phi|=1} \rho(\phi), \tag{10.50}$$

subject to:

$$\sum_{i=1}^{m} y_i \alpha_i = 0, \tag{10.51}$$

$$\alpha_i (y_i ((\mathbf{x}_i * \psi_0) + b_0) - 1) = 0, \quad i = 1, 2, ..., m, \tag{10.52}$$

$$\alpha_i \geq 0. \tag{10.53}$$

The expression (10.52) states the Kuhn-Tucker conditions. By observing these conditions one concludes that the nonzero values of α_i, $i = 1, 2, ..., m$, correspond only to the vectors x_i that satisfy the equality:

$$y_i ((\mathbf{x}_i * \psi_0) + b_0) = 1. \tag{10.54}$$

These vectors are the closest to the optimal hyperplane. They are called *support vectors*. The separating hyperplane can be written as:

$$\sum_{i=1}^{m} y_i \alpha_i^0 \left(\mathbf{x}_i * \mathbf{x}\right) + b_0 = 0, \tag{10.55}$$

where α_i^0, $i = 1, 2, ..., m$, satisfy the constraints (10.51)-(10.53). So, we can construct a decision function in the input space:

$$f\left(\mathbf{x}, \alpha_0\right) = sign\left(\sum_{i=1}^{m} y_i \alpha_i^0 \left(\mathbf{x}_i * \mathbf{x}\right) + b\right), \tag{10.56}$$

It is time to make a comparison between the perceptron model of section 9.1 and SVM. Firstly, both can be used to seek for a separating hyperplane to solve a binary classification problem. However, SVM does this task steered by an optimality condition that is satisfied by the maximum margin hyperplane (see also [6] for a comparison between perceptron and SVM). Now we describe two generalizations for the above approach.

10.3.1 Kernel Support Vector Machines

The foundations of kernel support vector machines (KSVM) belongs to the reproducing kernel Hilbert spaces and Mercer theory [136]. A remarkable result in this scenario is the Mercer theorem, which we summarized bellow in order to set the mathematical machinery that we are going to use in what follows [7]. So, let the space $\mathbb{R}^n$ and μ a finite measure in $\mathbb{R}^n$. We define also the function spaces $L_2\left(\mathbb{R}^n\right) = \{f : \mathbb{R}^n \longrightarrow \mathbb{R};\ |f|^2\ is\ \mu - integrable\}$ and $L_\infty\left(\mathbb{R}^n\right) = \{f : \mathbb{R}^n \longrightarrow \mathbb{R};\ \exists K > 0,\ |f\left(\mathbf{x}\right)| \leq K\}$. Before stating the fundamental theorem, we also need the following definition [137]:

Definition 6 (Positive definite functions). A symmetric function $h : \mathbb{R}^n \times \mathbb{R}^n \longrightarrow \mathbb{R}$ is positive definite if $\forall m \geq 1, \forall \left(a_1, \ldots, a_m\right) \in \mathbb{R}^m, \forall \left(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\right) \in \mathbb{R}^n \times \mathbb{R}^n \times \cdots \times \mathbb{R}^n$:

$$\sum_{i=1}^{m} \sum_{j=1}^{m} a_i a_j h\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) \geq 0. \tag{10.57}$$

Theorem 1 (Mercer): Suppose $k : \mathbb{R}^n \times \mathbb{R}^n \longrightarrow \mathbb{R}$ is a continuous symmetric positive definite function (kernel) such that $k \in L_\infty\left(\mathbb{R}^n \times \mathbb{R}^n\right)$. Under certain

conditions, the integral operator $T_k : L_2(\mathbb{R}^n) \longrightarrow L_2(\mathbb{R}^n)$:

$$(T_k f)(\mathbf{x}) = \int_{\mathbb{R}} k(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) d\mu(\mathbf{y}), \tag{10.58}$$

has a set of normalized eigenfunctions $\psi_j : \mathbb{R}^n \longrightarrow \mathbb{R}$, with associated eigenvalues $\lambda_j > 0$, sorted in non-increasing order, such that:

$$k(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{n_F} \lambda_j \psi_j(\mathbf{x}) \psi_j(\mathbf{y}), \tag{10.59}$$

where either $n_F \in \mathbb{N}$ or $n_F = \infty$.

If $n_F < +\infty$, expression (10.59) means that $k(\mathbf{x}, \mathbf{y})$ corresponds to a dot product in $\mathbb{R}^{N_F}$, i.e., $k(\mathbf{x}, \mathbf{y}) =< \Phi(\mathbf{x}), \Phi(\mathbf{y}) >$, with:

$$\Phi : \mathbb{R}^n \longrightarrow \mathbb{R}^{N_F},$$

$$\Phi(x) = \left(\sqrt{\lambda_1}\psi_1(x), \sqrt{\lambda_2}\psi_2(x), \ldots, \sqrt{\lambda_{N_F}}\psi_{N_F}(x) \right), \tag{10.60}$$

where $\psi_j : \mathbb{R}^n \longrightarrow \mathbb{R}$.

The Hilbert space $\mathbb{R}^{N_F}$ is called the feature space. If $n_F = \infty$ the complete theorem assures uniform convergence of the series that represents $k(\mathbf{x}, \mathbf{y})$. This implies that given $\varepsilon > 0$, there exists an $t \in \mathbb{N}$, such that, even if the range of Φ is infinite-dimensional, the kernel k can be approximated within accuracy ε as a dot product in $\mathbb{R}^t$:

$$\Phi_t(\mathbf{x}) = \left(\sqrt{\lambda_1}\psi_1(\mathbf{x}), \sqrt{\lambda_2}\psi_2(\mathbf{x}), \ldots, \sqrt{\lambda_t}\psi_t(\mathbf{x}) \right).$$

With these results, the KSVM generalizes the linear support vector machines through the kernel function k. Specifically, we apply the kernel trick and replace the inner product in expression (10.55) to its kernel version to write the hypersurface that separates positive from negative samples in the input space as [7]:

$$F(\mathbf{x}) \equiv \sum_{i=1}^{m} y_i \alpha_i k(\mathbf{x}_i, \mathbf{x}) + \widetilde{b} = 0, \tag{10.61}$$

where $\alpha_i \geq 0$, $i = 1, 2, \cdot, \cdot, \cdot, m$, are Lagrange multipliers in the quadratic optimization problem behind KSVM technique [7]. This problem is generated by

replacing the inner product in expression (10.48) to its kernel version. Hence, we maximize the expression:

$$W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}), \tag{10.62}$$

in the non-negative quadrant $\alpha_i \geq 0, i = 1, 2, ..., m$, under the constraint (10.47).

Likewise in the linear case, the samples $\mathbf{x}_i$ with $\alpha_i \neq 0$ are named support vectors. Let $\Phi(\mathbf{x}) = (z_1(\mathbf{x}), z_2(\mathbf{x}), z_3(\mathbf{x}), \ldots, z_{n_F}(\mathbf{x}))$ be the map in expression (10.60). Hence, since $k(\mathbf{x}_i, \mathbf{x}) =< \Phi(\mathbf{x}_i), \Phi(\mathbf{x}) >$, the separating surface in equation (10.61) can be written in the feature space $\mathbb{R}^{n_F}$ as:

$$\sum_{r=1}^{n_F} \omega_r z_r(\mathbf{x}) + \widetilde{b} = 0 \tag{10.63}$$

where $\omega_r = \sum_{i=1}^{m} y_i \alpha_i z_r(\mathbf{x}_i)$.

Therefore, we can generalize expression (10.56) by using the inner product defined by the kernel k :

$$f(x, \alpha_0) = sign\left(\sum_{i=1}^{M} y_i \alpha_i^0 k(\mathbf{x}_i * \mathbf{x}) + \widetilde{b}\right), \tag{10.64}$$

or, equivalently, we can use the linear decision function in the feature space Z :

$$f(x, \alpha_0) = sign\left[\sum_{i=1}^{M} y_i \alpha_i^0 \left(\sum_{r=1}^{n_F} \lambda_r z_r\left(\mathbf{x}^i\right) z_r(\mathbf{x})\right) + \widetilde{b}\right], \tag{10.65}$$

These expressions define the kernel SVM (KSVM) method [7, 12]. In summary, the method computes a separating hypersurface (Figure 10.3.(a)) in the input space through expression (10.61) which corresponds to seek for the hyperplane defined by equation (10.63), in the feature space, which separates positive and negative observations with the maximum margin, as represented in Figure 10.3.(b).

Some possibilities for the kernel k can be found in [136]. An usual choice for the kernel function k is the radial basis function (RBF), given by:

$$k(\mathbf{x}, \mathbf{y}) = \exp\left(\frac{||\mathbf{x} - \mathbf{y}||^2}{2\sigma^2}\right), \tag{10.66}$$

where $\sigma \in \mathbb{R}$.

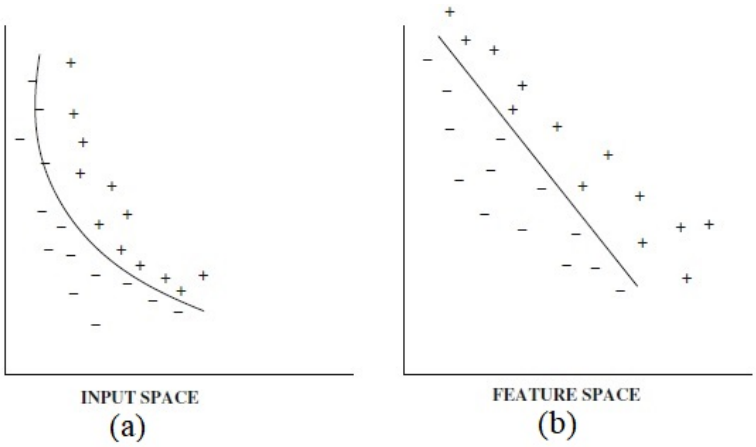


Figure 10.3: (a) Decision boundary in the input space. (b) Separating hypersurface in the feature space. Reprinted from [6].

10.3.2 General Non-separable Case

Sometimes the subsets may be non-separable, that means, we can not find a small constant δ such that conditions (9.7)-(9.8) hold true. An important property of the SVM is its extendibility to non-separable case through slack variables ξ_i that simulate pushing outlier samples toward the correct group [7]; that means:

$$\langle \mathbf{x}_i + \xi_i \phi, \phi \rangle + b \geq 1 \Longleftrightarrow \langle \mathbf{x}_i, \phi \rangle + b \geq 1 - \xi_i \langle \phi, \phi \rangle , \tag{10.67}$$

$$\langle \mathbf{x}_i - \xi_i \phi, \phi \rangle + b \leq -1 \Longleftrightarrow \langle \mathbf{x}_i, \phi \rangle + b \leq -1 + \xi_i \langle \phi, \phi \rangle . \tag{10.68}$$

The norm $\langle \phi, \phi \rangle$ can be incorporated in the slack variables values. Besides, we can use the fact that $y_i \in \{-1, 1\}$ to summarize expressions (10.67)-(10.68) as:

$$y_i \left(\langle \mathbf{x}_i, \phi \rangle + b \right) \geq 1 - \xi_i. \tag{10.69}$$

Moreover, one must incorporate the slack variables in the objective function:

$$\tau(\phi, \xi, C) = \frac{1}{2} ||\phi||^2 + C \sum_{i=1}^{M} \xi_i, \tag{10.70}$$

where C is a relaxation constant. The minimization of function (10.70) subject to constraints (10.69) gives the non-separable linear SVM, also called C-SVM in [7, 138]. To perform this task we must follow the same steps of section 10.3, which we let as exercise.

We shall notice that it is not possible to guarantee that these C-SVM machines possess all the nice properties of the SVM machine defined constructed in section 10.3.

10.4 Discriminant Principal Components Analysis

We shall remember that *feature* is any numerical value that can be derived from the input data. Given a feature space, a key question is "how can we determine (or compute) the most important discriminant features for a pattern recognition task, like classification?" Discriminant analysis techniques address this question, which is very known in the context of PCA.

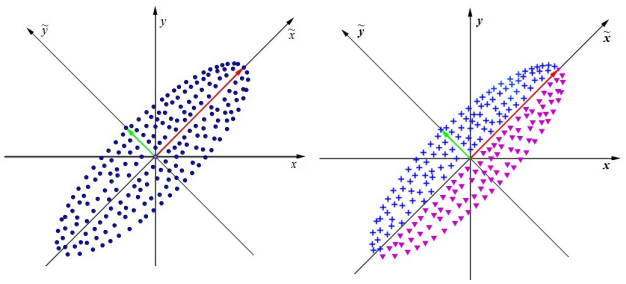
The Figure 10.6 is a simple example that pictures the limitation of PCA for discriminant features extraction. Both Figures 10.6.(a) and 10.6.(b) represent the same data set. Figure 10.6.(a) just shows the PCA directions ($\widetilde{\mathbf{x}}$ and $\widetilde{\mathbf{y}}$) and the distribution of the samples over the space. However, in Figure 10.6.(b) we distinguish two patterns: plus $(+)$ and triangle $(\blacktriangledown)$. We observe that the principal PCA direction $\widetilde{\mathbf{x}}$ can not discriminate samples of the considered groups because the projection of the data points over direction $\widetilde{\mathbf{x}}$ will mix the patterns in the corresponding one-dimensional subspace.

In general, Fisher's linear discriminant analysis (LDA) is used to identify the most important linear directions for separating sample groups rather than PCA [22]. This method, as well as the weighted pairwise variant of the well-known multi-class Fisher criterion introduced in [139] has the limitation of finding number of groups - 1 meaningful discriminant directions (see section 10.5).

In [140] it is proposed the DPCA technique, based on the idea of using the discriminant weights obtained by separating hyperplanes to select among the principal components the most discriminant ones.

The original DPCA is implemented taking as input a labeled training set:

$$X = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2) \ldots (\mathbf{x}_M, y_M)\}. \tag{10.71}$$



(a) (b)

Figure 10.4: (a) Scatter plot and PCA directions. (b) The same population but distinguishing patterns plus $(+)$ and triangle $(\blacktriangledown)$.

Firstly, for discarding redundancies, the PCA transformation matrix $P_{pca} = [\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_m]$ is computed and each zero mean data vector $\mathbf{u}_i$ is projected generating a vector $\overline{\mathbf{u}}_i = (P_{pca})^T \widetilde{\mathbf{x}}_i$. Afterwards, the obtained $M \times m'$ data matrix and their corresponding labels are used as input to calculate the separating hyperplane. In the following we focus on the SVM technique, although any other linear classifier could be used.

Since DPCA assumes only two classes to separate, there are only one discriminant vector $\phi_{svm} = (w_1, w_2, \cdots, w_m)$ given by the SVM hyperplane. If we multiply the $M \times m$ most expressive features matrix by the $m \times 1$ discriminant SVM vector:

$$\begin{aligned} c_1 &= \overline{\mathbf{x}}_{11} w_1 + \overline{\mathbf{x}}_{12} w_2 + ... + \overline{\mathbf{x}}_{1m} w_m, \\ c_2 &= \overline{\mathbf{x}}_{21} w_1 + \overline{\mathbf{x}}_{22} w_2 + ... + \overline{\mathbf{x}}_{2m} w_m, \\ &... \\ c_M &= \overline{\mathbf{x}}_{N1} w_1 + \overline{\mathbf{x}}_{N2} w_2 + ... + \overline{\mathbf{x}}_{Nm} w_m. \end{aligned} \tag{10.72}$$

we get the most discriminant feature $c_i \in \mathbb{R}$ of each one of the m-dimensional vectors $\overline{\mathbf{x}}_i$. Therefore, we can determine the discriminant contribution of each feature by investigating the weights $[w_1, w_2, ..., w_m]$. In fact, weights that are estimated to be 0 or approximately 0 have negligible contribution on the discriminant scores c_i described in equation (10.72), indicating that the corresponding features are not significant to separate the sample groups. In contrast, largest weights (in absolute values) indicate that the corresponding features contribute more to the

discriminant score and consequently are important to characterize the differences between the groups.

Therefore, instead of sorting these features by selecting the corresponding principal components in decreasing order of eigenvalues, as PCA does, DPCA selects as the most important features for classification the ones with the highest discriminant weights, that is, $|w_1| \geq |w_2| \geq ... \geq |w_m|$.

10.5 Linear Discriminant Analysis (LDA)

The primary purpose of LDA is to compute discriminant directions to separate samples of distinct groups by maximizing their between-class separability while minimizing their within-class variability. Its main objective is to find a projection matrix W_{lda} that maximizes the Fisher's criterion:

$$W_{lda} = \arg\max_{W} \frac{\left|W^T S_b W\right|}{\left|W^T S_w W\right|} \tag{10.73}$$

where S_b and S_w are the between-class and within-class matrices, respectively, which are defined as:

$$S_b = \sum_{i=1}^{g} N_i (\overline{\mathbf{x}}_i - \overline{\mathbf{x}})(\overline{\mathbf{x}}_i - \overline{\mathbf{x}})^T \quad and \quad S_w = \sum_{i=1}^{g} \sum_{j=1}^{N_i} (\mathbf{x}_{i,j} - \overline{\mathbf{x}}_i)(\mathbf{x}_{i,j} - \overline{\mathbf{x}}_i)^T \tag{10.74}$$

with N_i representing the number of training pattern from class 'i', $\mathbf{x}_{i,j}$ is the m-dimensional pattern j from class 'i', g is the total number of classes. Each sample group 'i' has a class mean, which is denote as $\overline{\mathbf{x}}_i$, where:

$$\overline{\mathbf{x}}_i = \frac{1}{N_i} \sum_{j=1}^{N_i} \mathbf{x}_{i,j} \tag{10.75}$$

and the grand mean vector $\overline{\mathbf{x}}$, equivalent to expression (10.2), is given by:

$$\overline{\mathbf{x}} = \frac{1}{N} \sum_{i=1}^{g} N_i \overline{\mathbf{x}}_i = \frac{1}{N} \sum_{i=1}^{g} \sum_{j=1}^{N_i} \mathbf{x}_{i,j} \tag{10.76}$$

where N is total number of samples, that is, $N = N_1 + N_2 + ... + N_g$.

It can be demonstrated that the Fisher criterion is maximized when the projection matrix W_{lda} is the solution of the eigensystem problem, [71, 132]:

$$S_b W - S_w W \Lambda = 0. \tag{10.77}$$

So, by multiplying both sides of equation 10.77 by S_w^{-1}, we obtain:

$$\begin{aligned} S_w^{-1} S_b W - S_w^{-1} S_w W \Lambda &= 0, \\ S_w^{-1} S_b W - W \Lambda &= 0, \\ (S_w^{-1} S_b) W &= W \Lambda. \end{aligned} \tag{10.78}$$

and, consequently, W_{lda} is composed the $g-1$ eigenvectors of $S_w^{-1} S_b$ with nonzero eigenvalues, [132]. In the case of a two-class problem, the LDA projection matrix is in fact the leading eigenvector w^{lda} of $S_w^{-1} S_b$, as represented in Figure 10.5.

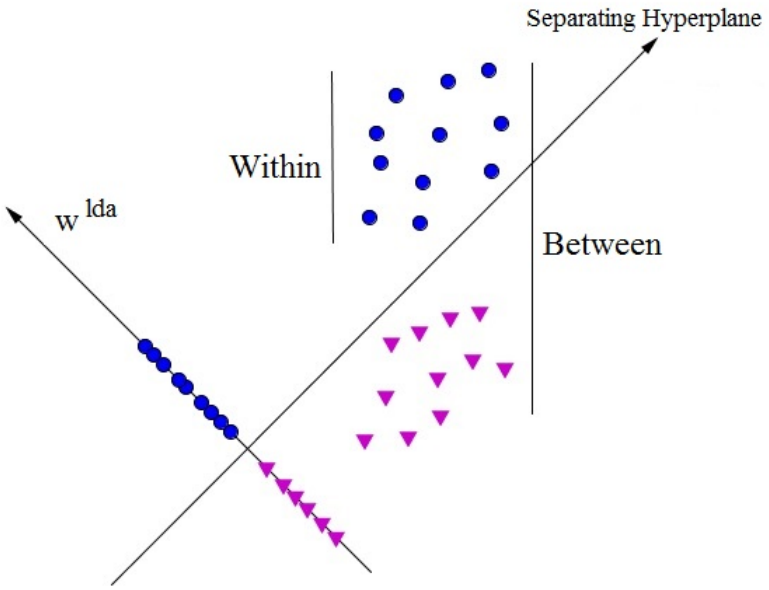


Figure 10.5: Representation of separating hyperplane generated by LDA

The matrix S_w may be singular when there are a limited number of total training observations N compared to the dimension of the feature space m [71]. So, the performance of the standard LDA can be seriously degraded due to the fact that

it is necessary to invert the S_w matrix to find the LDA subspace. In order to deal with such situations we can use a regularized version of the LDA approach called Maximum Uncertainty LDA (MLDA), [71] which replaces S_w by the matrix:

$$S_w^* = S_p^*(N-g) \tag{10.79}$$

with $S_p^* = \Phi\Lambda^*\Phi^T$, where Φ is composed by the eigenvectors of matrix:

$$S_p = \frac{S_w}{(N-g)}, \tag{10.80}$$

the diagonal matrix Λ^* is formed by:

$$\Lambda^* = \begin{bmatrix} \max\{\lambda_1, \overline{\lambda}\} & 0 & \dots & 0 \\ 0 & \max\{\lambda_2, \overline{\lambda}\} & \dots & 0 \\ . & . & . & . \\ 0 & 0 & \dots & \max\{\lambda_m, \overline{\lambda}\} \end{bmatrix} \tag{10.81}$$

where, each λ_i, is an eigenvalue of S_p and $\overline{\lambda}$ is computed by:

$$\overline{\lambda} = \frac{1}{m}\sum_{i=1}^{m}\lambda_i. \tag{10.82}$$

The output of the MLDA is the projection matrix W^{mlda} computed by replacing S_w with S_w^* in the Fisher criterion and by solving the corresponding optimization problem:

$$W^{mlda} = \arg\max_W \frac{|W^T S_b W|}{|W^T S_w^* W|}. \tag{10.83}$$

The Algorithm 6, proposed in [71], summarizes the MLDA procedure.

10.6 Kernel PCA

Consider the kernel $k : \mathbb{R}^n \times \mathbb{R}^n \longrightarrow \mathbb{R}$ and the corresponding map $\Phi : \mathbb{R}^n \longrightarrow \mathbb{R}^{N_F}$ of Mercer theorem (section 10.3.1). Also, we consider the dataset $S = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_m\} \subset \mathbb{R}^n$. So, we can project S in the feature space to compose the set $\Phi(S) = \{\Phi(\boldsymbol{x}_1), \Phi(\boldsymbol{x}_2), \dots, \Phi(\boldsymbol{x}_m)\} \subset \mathbb{R}^{N_F}$.

Algorithmus 6 Procedure for MLDA.

1: Find the Φ eigenvectors and Λ eigenvalues of S_p, where $S_p = \frac{S_w}{(N-g)}$

2: Calculate the average eigenvalue $\overline{\lambda}$ using expression (10.82);

3: Form a new matrix of eigenvalues through:

$\Lambda^* = diag[max\{\lambda_1, \overline{\lambda}\}, ..., max\{\lambda_n, \overline{\lambda}\}]$;

4: Compute the modified within-class scatter matrix

$$S_w^* = S_p^*(N-g) = (\Phi\Lambda^*\Phi^T)(N-g). \tag{10.84}$$

5: Compute W^{mlda} by solving the problem (10.83).

The kernel version of the PCA, named KPCA technique, is developed based on the following assumptions

1) Samples in the feature space have zero mean:

$$\frac{1}{m}\sum_{i=1}^{m} \Phi(\boldsymbol{x}_i) = 0, \tag{10.85}$$

2) The covariance matrix R in the feature space:

$$R = \frac{1}{m}\sum_{i=1}^{m} \Phi(\boldsymbol{x}_i)\,\Phi(\boldsymbol{x}_i)^T, \tag{10.86}$$

has eigenvectors $\boldsymbol{v}_k$, $k = 1, 2, \ldots, N_F$ that can be expressed as a linear combinations of projected samples:

$$\boldsymbol{v}_k = \sum_{j=1}^{m} a_{kj}\Phi(\boldsymbol{x}_j), \tag{10.87}$$

3) The matrix $K = \{k(\boldsymbol{x}_i, \boldsymbol{x}_j)\}_{1\leq i,j\leq m}$, named Gram matrix, is non-singular.

With these assumptions in mind, we follow the main ideas of linear PCA (section 10.1) but considering the projected dataset $\Phi(S)$. Hence, the eigenvalue-eigenvector equation $R\boldsymbol{v}_k = \lambda_k v_k$ is central in the development. Due to expression (10.86) and the assumption 2 , this equation can be rewritten as:

$$\frac{1}{m}\sum_{i=1}^{m} \Phi(\boldsymbol{x}_i)\,\Phi(\boldsymbol{x}_i)^T \left(\sum_{j=1}^{m} a_{kj}\Phi(\boldsymbol{x}_j)\right) = \lambda_k \left(\sum_{j=1}^{m} a_{kj}\Phi(\boldsymbol{x}_j)\right). \tag{10.88}$$

The remaining text is constructed based on the reference [44]. So, we apply the key step that consists in rewrite equation (10.88) using the kernel function k computed in the pairs $(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Since $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \Phi(\boldsymbol{x}_i)^T \Phi(\boldsymbol{x}_i)$, this is done by multiplying both sides of expression (10.88) by $\Phi(\boldsymbol{x}_l)^T$:

$$\frac{1}{m}\sum_{i=1}^{m} \Phi(\boldsymbol{x}_l)^T \Phi(\boldsymbol{x}_i) \Phi(\boldsymbol{x}_i)^T \left(\sum_{j=1}^{m} a_{kj}\Phi(\boldsymbol{x}_j)\right) = \lambda_k \left(\sum_{j=1}^{m} a_{kj}\Phi(\boldsymbol{x}_l)^T \Phi(\boldsymbol{x}_j)\right), \tag{10.89}$$

that is equivalent to:

$$\frac{1}{m}\sum_{i=1}^{m} k(\boldsymbol{x}_l, \boldsymbol{x}_i) \left(\sum_{j=1}^{m} a_{kj} k(\boldsymbol{x}_i, \boldsymbol{x}_j)\right) = \lambda_k \left(\sum_{j=1}^{m} a_{kj} k(\boldsymbol{x}_l, \boldsymbol{x}_j)\right), \tag{10.90}$$

which can be re-written in matrix form as:

$$K^2 \boldsymbol{a}_k = m\lambda_k K \boldsymbol{a}_k, \tag{10.91}$$

where $K = \{k(\boldsymbol{x}_i, \boldsymbol{x}_j)\}_{1 \leq i,j \leq m}$ and $\boldsymbol{a}_k = \begin{pmatrix} a_{k1} & a_{k2} & \dots & a_{km} \end{pmatrix}^T$.

From the assumption 3 above we finally obtain:

$$K \boldsymbol{a}_k = m\lambda_k \boldsymbol{a}_k. \tag{10.92}$$

Now, we shall return to 10.87 and impose the normalization constraint to the eigenvectors $\boldsymbol{v}_i$:

$$\begin{aligned} 1 = \boldsymbol{v}_i^T \boldsymbol{v}_i &= \sum_{j=1}^{m}\sum_{l=1}^{m} a_{ij}\Phi^T(\boldsymbol{x}_j)\, a_{il}\Phi(\boldsymbol{x}_l) = \sum_{j=1}^{m}\sum_{l=1}^{m} a_{ij} a_{il} k(\boldsymbol{x}_j, \boldsymbol{x}_l) \\ &= \boldsymbol{a}_i^T K \boldsymbol{a}_i = m\lambda_i \boldsymbol{a}_i^T \boldsymbol{a}_i \end{aligned} \tag{10.93}$$

So, a projected point $\Phi(\boldsymbol{x})$ is represented in the KPCA basis of the normalized eigenvectors as:

$$\Phi(\boldsymbol{x}) = \sum_{i=1}^{N_F} b_i \boldsymbol{v}_i, \tag{10.94}$$

where, each component b_i can be computed by:

$$b_i = \Phi^T(\boldsymbol{x})\, \boldsymbol{v}_i = \sum_{j=1}^{m} a_{ij}\Phi^T(\boldsymbol{x})\, \Phi(\boldsymbol{x}_j) = \sum_{j=1}^{m} a_{ij} k(\boldsymbol{x}, \boldsymbol{x}_j) \tag{10.95}$$

Before continuing, we shall notice the number of non-null eigenvalues does not exceed the number of samples. However, the number of nonlinear principal components can exceed the dimension n of the input samples.

The development of KPCA in the general case, where the assumption (1) is not verified, can be found in [44]. Basically, likewise in the linear PCA, we centralize the projected samples:

$$\widetilde{\Phi}\left(\boldsymbol{x}_i\right) = \Phi\left(\boldsymbol{x}_i\right) - \frac{1}{m}\sum_{i=1}^{m}\Phi\left(\boldsymbol{x}_i\right), \tag{10.96}$$

and compute the Gram matrix elements $\widetilde{k}\left(\boldsymbol{x}_s, \boldsymbol{x}_t\right)$ as:

$$\widetilde{k}\left(\boldsymbol{x}_s, \boldsymbol{x}_t\right) = \widetilde{\Phi}^T\left(\boldsymbol{x}_s\right)\widetilde{\Phi}\left(\boldsymbol{x}_t\right)$$

$$= \left(\Phi\left(\boldsymbol{x}_s\right) - \frac{1}{m}\sum_{i=1}^{m}\Phi\left(\boldsymbol{x}_i\right)\right)^T \left(\Phi\left(\boldsymbol{x}_t\right) - \frac{1}{m}\sum_{i=1}^{m}\Phi\left(\boldsymbol{x}_i\right)\right)^T. \tag{10.97}$$

After some algebra, we can show that the Gram matrix $\widetilde{K}$ can be computed by:

$$\widetilde{K} = K - 1_m K - K 1_m + 1_m K 1_m, \tag{10.98}$$

where $1_m \in \mathbb{R}^{m\times m}$ is the matrix in which every element is equal to $1/m$. We let the details to obtain expression (10.98) as exercise (see also [44]). Expression (10.98) shows that we can evaluate $\widetilde{K}$ using only the kernel function. Once $\widetilde{K}$ we determine the eigenvalues and eigenvectors by solving expression 10.92 with K replaced by $\widetilde{K}$. So, we must perform the following steps:

1. Solve the eigenvalue-eigenvector equation:

$$\widetilde{K}\widetilde{\boldsymbol{a}}_k = m\widetilde{\lambda}_k\widetilde{\boldsymbol{a}}_k. \tag{10.99}$$

2. Nomalization:

$$1 = \widetilde{\boldsymbol{v}}_i^T\widetilde{\boldsymbol{v}}_i = \widetilde{\boldsymbol{a}}_i^T\widetilde{K}\widetilde{\boldsymbol{a}}_i = m\widetilde{\lambda}_i\widetilde{\boldsymbol{a}}_i^T\widetilde{\boldsymbol{a}}. \tag{10.100}$$

3. Representation in the normalized KPCA basis:

$$\widetilde{\Phi}\left(\boldsymbol{x}\right) = \sum_{i=1}^{N_F}\widetilde{b}_i\tilde{\boldsymbol{v}}_i, \tag{10.101}$$

where:

$$\widetilde{b}_i = \widetilde{\Phi}^T(\boldsymbol{x})\,\widetilde{\boldsymbol{v}}_i = \sum_{j=1}^{m} \widetilde{a}_{ij}\widetilde{\Phi}^T(\boldsymbol{x})\,\widetilde{\Phi}(\boldsymbol{x}_j) = \sum_{j=1}^{m} \widetilde{a}_{ij}\widetilde{k}(\boldsymbol{x}, \boldsymbol{x}_j)\,. \qquad (10.102)$$

Once expression (10.101) is computed we shall remember equation (10.96) are write:

$$\Phi(\boldsymbol{x}) = \frac{1}{m}\sum_{i=1}^{m} \Phi(\boldsymbol{x}_i) + \sum_{i=1}^{N_F} \widetilde{b}_i \tilde{\boldsymbol{v}}_i \qquad (10.103)$$

10.7 Classification versus Reconstruction

In the context of image processing it is worthwhile to analyze how good a low dimensional representation will look like by visualizing the projected data in the original space. So, returning to the equation (10.10) given a projected centered data $\overline{\mathbf{u}}$, this is formally performed by the operation:

$$\mathbf{x} = P_{pca}\overline{\mathbf{u}} + \widehat{\mathbf{x}}. \qquad (10.104)$$

The operation (10.104) is named reconstruction. The vector $\mathbf{x}$ obtained belongs to the original space and, consequently, its patterns can be recognized through visualization. From the viewpoint of feature extraction, reconstruction and classification are two different problems. In the former, one seeks for a subspace able to represent features well perceivable on the original data space. On the other hand, classification depends on finding out a projection that emphasizes discriminative patterns, sometimes related to details in the samples and, consequently, less useful for data visualization.

More specifically, since PCA explains the covariance structure of all the data its most expressive components, [141], that is, the first principal components with the largest eigenvalues, do not necessarily represent important discriminant directions to separate sample groups, [142]. The Figure 10.6 is a simple example that pictures this fact. Both Figures 10.6.(a) and 10.6.(b) represent the same data set. Figure 10.6.(a) just shows the PCA directions ($\widetilde{x}$ and $\widetilde{y}$) and the distribution of the samples over the space. However, in Figure 10.6.(b) we distinguish two patterns: plus $(+)$ and minus $(\blacktriangledown)$. We observe that the principal PCA direction $\widetilde{x}$

can not discriminate samples of the considered groups because the projection of the dataset over $\widetilde{x}$ will mix the different patterns observed in the figure. On the other hand, the PCA direction $\widetilde{y}$ is efficient for classification allowing to separate samples groups. So, we say that $\widetilde{y}$ is the most discriminant direction to recognize patterns plus $(+)$ and triangle $(\blacktriangledown)$.

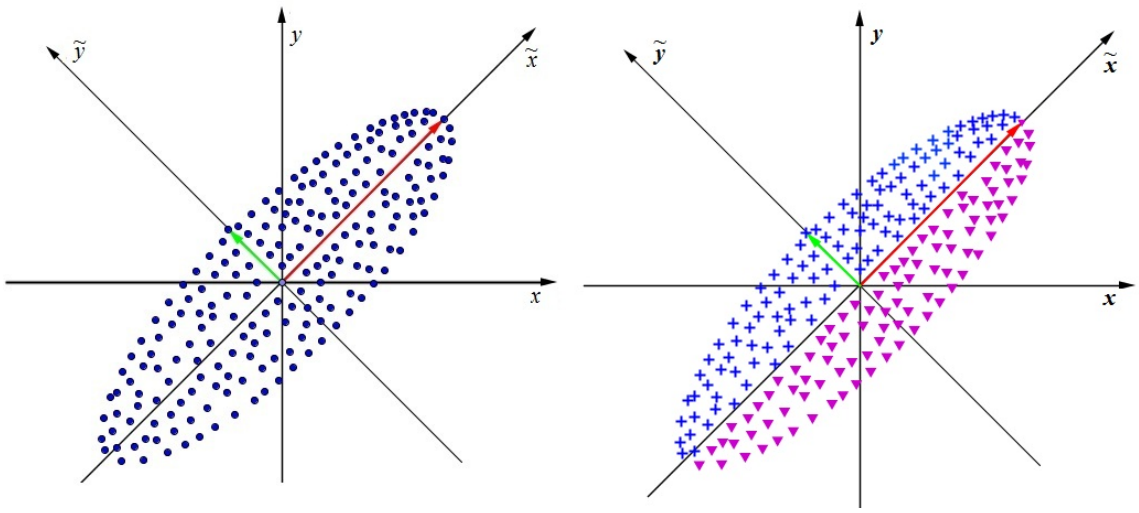


Figure 10.6: (a) Scatter plot and PCA directions. (b) The same population but distinguishing patterns plus $(+)$ and triangle $(\blacktriangledown)$.

Give a feature space, not necessarily PCA, the LDA (section 10.5) technique can be used to compute meaningful discriminant directions. Before continuing, it is important to highlight that the solution of the LDA optimization problem (equation 10.73) is, in general, gives directions different from the coordinate directions of the original feature space. If we pay attention in the mathematical formulation of LDA and PCA we notice that the former works by maximizing the between-class separability while minimizing their within-class variability, that means, the method tries to collapse the classes in single points as separated as possible in order to get a subspace efficient for classification. In a different way, the optimization criterion behind PCA can be written as: seek for the matrix P that minimizes the root mean squared error (RMSE) for the reconstruction process, computed by:

$$RMSE(k) = \sqrt{\frac{\sum_{j=1}^{M} ||P.I_k.P^T\mathbf{x}_j - \mathbf{x}_j||^2}{M}}, \tag{10.105}$$

where I_k is a truncated identity matrix that keeps the selected subspace with dimension k, $P = P_{pca}$, and $\|\cdot\|$ is the usual 2-norm. Hence, the projection matrix P_{pca} has the property of minimizing truncation error inserted when projecting the database samples in a reduced PCA subspace. We can show that this process is equivalent to seek for directions that maximize the variance of the project data. In fact, the distribution of eigenvalues of the covariance matrix R in expression (10.6) gives a measure of those variances.

In the case of image datasets we can visualize these facts by considering another version of the reconstruction problem yielded with the help of a separating hyperplane. Hence, given the direction $\mathbf{q}$ in the feature space, we can use the expression:

$$I = \widehat{\mathbf{O}} + \delta \cdot \mathbf{q}, \tag{10.106}$$

where $\delta \in \mathbb{R}$ is a parameter, and $\widehat{\mathbf{O}}$ is a point in the feature space. If $\mathbf{q}$ is a PCA direction, a common choice is $\delta \in \{\pm j \cdot \lambda^{0.5},\ j = 0, 1, 2, 3\}$, and λ is the eigenvalue associated to $\mathbf{q}$.

For instance, as described in section 10.3, the SVM method seeks to find a decision boundary that separates data into different classes as well as possible. The Figure 10.7 pictures a dataset composed by two classes. This figure represents the dataset, the PCA components (PCAx and PCAy) and the separating plane obtained by the SVM method with its orientation and position given by expression (10.49)-(10.50). In the representation of Figure 10.7, we assume that the $b_0 = 0$. If we set $\mathbf{q} = \psi_0$ and take a sequence $\delta_1, \delta_2, \cdots$, we generate an array of new data points $I_1, I_2, \cdots$ in the line defined by $\widehat{\mathbf{O}}$ and the vector ψ_0 in the PCA space. So, expression (10.104) allows to visualize these samples in the original image space through the computation of the sequence:

$$\mathbf{x}_i = P_{pca} I_i + \widehat{\mathbf{x}}, \quad i = 1, 2, ... \tag{10.107}$$

An analogous operation can be performed using any other hyperplane in the PCA space. In this line, in [143] authors use an hyperplane with the orientation given by the LDA technique and passing through origin of the PCA coordinate system (centroid position of the dataset), as represented in Figure 10.8. The LDA solution is a spectral matrix analysis of the data and is based on the assumption

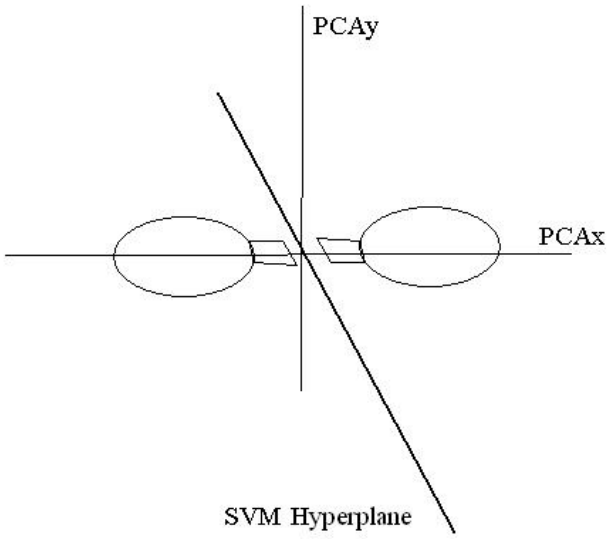


Figure 10.7: SVM separating hyperplane.

that each class can be represented by its distribution of data, that is, the corresponding mean vector (or class prototype) and covariance matrix (or spread of the sample group). In this sense, the hyperplane shown in Figure 10.8 can be used for classification, although it is expected to misclassify data points, specially nearby the frontier of the classes.

The description of the SVM solution, on the other hand, does not make any assumption on the distribution of the data, focusing on the observations that lie close to the opposite class, that is, on the observations that most count for classification. In other words SVM discriminative direction focuses on the data at the boundary of the classes, extracting group-differences that are less perceivable on the original image space. This is emphasized in Figure 10.7 which indicates that SVM is more robust to outliers, given a zoom into the subtleties of group differences.

However, according to the above discussion, if we apply expression (10.106) over the LDA and SVM discriminate directions (normal vector of the separating planes) and project the result back into the image domain we expect to observe a better reconstruction result for the LDA case. To clarify this fact, we shall notice that LDA problem (10.73), for binary databases, tries to collapse the two classes in single points as separated as possible. Therefore, the LDA discriminate direction takes into account all the data allowing to perform a more reliable reconstruction

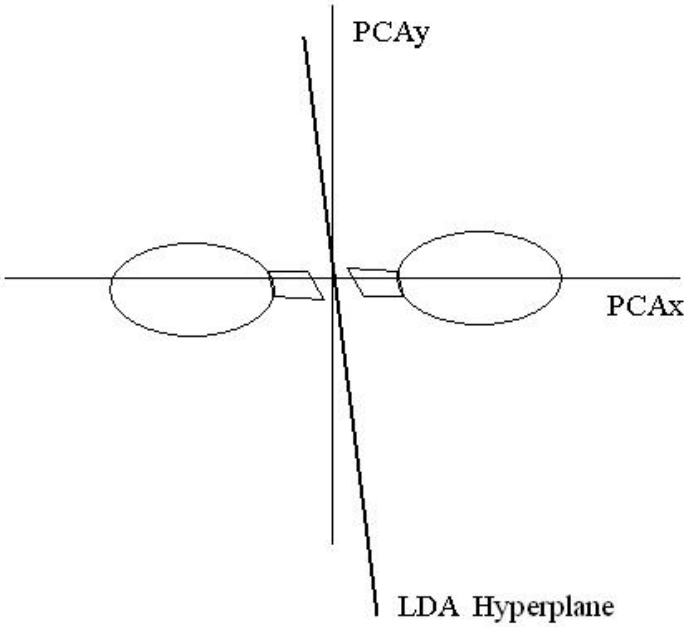


Figure 10.8: LDA separating hyperplane.

process through expression (10.106).

10.8 Exercises

1. Generate a dataset with two patterns in $\mathbb{R}^2$, with labels in the set $\{-1, 1\}$, separable by a line. Compute the linear SVM and find out the support vectors.

2. Consider a set of points $\mathbb{R}^2$ with two patterns like in Figure 10.9.

   Compute KSVM and try to find a three dimensional feature space where the projected patterns are linearly separable

3. Generate the Figure 1 of reference [143] for the FEI database.

4. Consider the database of exercise 3. Project the data in the PCA space. Apply SVM for binary classification. Evaluate the result using F1-Score (equation 8.32) and Accuracy (equation 8.33).

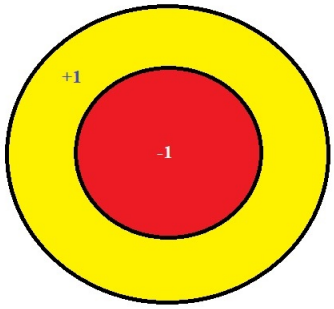


Figure 10.9: Two patterns in $\mathbb{R}^2$ with non-linear separating curve.

5. Reproduce exercise 4 for KSVM.

6. Compute a linear classifier for gender in the FEI database using the LDA technique.

7. Consider the FEI database and apply the KPCA. Compute a SVM classifier in the KPCA space .

Chapter 11

Deep Neural Networks

11.1 Tensor Algebra

In the traditional image processing literature, a tensor of order n is just a generalized matrix $\mathbf{X} \in \Re^{m_1 \times m_2 \times \ldots \times m_n}$ [144]. So, it becomes clear that there is an isomorphism between $\Re^{m_1 \times m_2 \times \ldots \times m_n}$ and $\Re^{m_1 \cdot m_2 \cdots m_n}$. Therefore, the notions of internal product and norm in $\Re^{m_1 \times m_2 \times \ldots \times m_n}$ are induced, in a natural manner, from the $\Re^{m_1 \cdot m_2 \cdots m_n}$ space as follows.

Definition 7 *The internal product between two tensors* $\mathbf{X} \in \Re^{m_1 \times m_2 \times \ldots \times m_n}$ *and* $\mathbf{Y} \in \Re^{m_1 \times m_2 \times \ldots \times m_n}$ *is defined by:*

$$\langle \mathbf{X}, \mathbf{Y} \rangle = \sum_{i_1=1,\ldots,i_n=1}^{m_1,\ldots,m_n} \mathbf{X}_{i_1,..,i_n} \mathbf{Y}_{i_1,..,i_n} \tag{11.1}$$

Definition 8 *The Frobenius norm of a tensor is given by the expression:*

$$\| \mathbf{X} \| = \sqrt{\langle \mathbf{X}, \mathbf{X} \rangle}, \tag{11.2}$$

,,

and the distance between tensors $\mathbf{X}$ *and* $\mathbf{Y}$ *is computed by:*

$$D(\mathbf{X}, \mathbf{Y}) = \| \mathbf{X} - \mathbf{Y} \| . \tag{11.3}$$

11.2 Convolution Operation

Since a digital image can be represented as a matrix of intensities, we can apply discrete operations in order to transform the input signal. The discrete convolution between the input signal $u(m,n)$ and the filter $h(m,n)$ is defined by:

$$v(m,n) = h(m,n) \circledast u(m,n) = \sum_{k=-\infty}^{+\infty} \sum_{l=-\infty}^{+\infty} h(k,l)\, u(m-k, n-l) \quad (11.4)$$

The mathematical properties behind these operations are discussed in the Appendix A. In practice, we implement these transformations through a discrete filter $h : W \to \mathbb{R}$, where:

$$W = \{-\xi, -\xi+1, \ldots -2, -1, 0, 1, 2, \ldots, \xi-1, \xi\}^2 . \quad (11.5)$$

The value $f = 2\xi+1$ is the filter size and matrix $\{h(m,n)\}_{-\xi \leq m,n \leq \xi}$ is named the filter mask. The discrete convolutions is computed by:

$$v(m,n) = \sum_{(k,l) \in W} h(k,l)\, u(m-k, n-l), \quad (11.6)$$

where $u(m,n)$ is the input image, $v(m,n)$ is the output image and $h(k,l)$ is the filter, also named the convolution kernel. Depending on the frequency response of the kernel $h(k,l)$ the filter may be a Low-Pass, High-Pass and Band-Pass.

11.2.1 Padding in Convolutions

The operation implemented through expression (11.6) has some issues for pixels nearby the boundary of the image. So, let us consider a image $u \in \mathbb{R}^{H \times W}$ and take the boundary pixels $\mathcal{B} = \{u(m,n);\ m = H \ or \ n = W\}$. The convolution (11.6) is defined only in positions (m,n) that satisfies:

$$|m-s| \geq \xi \ and \ |n-t| \geq \xi, \quad \forall (s,t) \in \mathcal{B}.$$

Consequently, the convolution in expression (11.6) will generate a new array $v \in \mathbb{R}^{(H-2\xi) \times (W-2\xi)}$ with resolution less than the input image. In terms of the

filter size, the resolution of the convolution output v is:

$$(H-2\xi)\times(W-2\xi)=\left(H-2\frac{(f-1)}{2}\right)\times\left(W-2\frac{(f-1)}{2}\right)$$
$$=(H-f+1)\times(W-f+1)\,. \tag{11.7}$$

In order to address this issue, we can use padding that consists of increasing the image resolution by adding $zeros$ all around the image boundary. Specifically, if we apply a padding p we get a new image $\widetilde{u}\in\mathbb{R}^{(H+2p)\times(W+2p)}$, such that:

$$\widetilde{u}\,(m,n)=u\,(m,n)\,,\ if\ \ (p+1)\leq m\leq(H+p)\,,\ \ (p+1)\leq m\leq(W+p)\,,$$

$$\widetilde{u}\,(m,n)=0,\ otherwise.$$

However, the idea is to use padding in order to get a output image v with the same dimensions of the input u. So, since the resolution of $\widetilde{u}$ is $(H+2p)\times(W+2p)$ we must apply expression (11.7) to calculate the dimension of the convolution $h\circledast\widetilde{u}$ in expression (11.6) that is given by:

$$((H+2p)-2\xi)\times((W+2p)-2\xi)=(H+2p-f+1)\times(W+2p-f+1)\,. \tag{11.8}$$

Consequently, in order to get an output $\widetilde{v}=h\circledast\widetilde{u}\in\mathbb{R}^{H\times W}$, we need a padding p that satisfies:

$$H=H+2p-f+1,$$

$$W=W+2p-f+1,$$

So, consequently:

$$p=\frac{f-1}{2}. \tag{11.9}$$

11.2.2 Strided Convolution

In this case, we chose a integer value s and compute the convolution only in the positions (m, n) that satisfies:

$$\xi + 1 \leq m \leq \alpha s,\ 0 \leq \alpha \leq \left\lfloor \frac{H}{s} \right\rfloor \ and\, H - \alpha s \geq \xi, \tag{11.10}$$

$$\xi + 1 \leq m \leq \beta s,\ 0 \leq \beta \leq \left\lfloor \frac{W}{s} \right\rfloor \ and\, W - \beta s \geq \xi. \tag{11.11}$$

As a consequently, if we apply padding p and stride s we get an output $\widetilde{v} \in \mathbb{R}^{\widetilde{H} \times \widetilde{W}}$ such that:

$$\widetilde{H} = \left\lfloor \frac{H + 2p - f}{s} + 1 \right\rfloor, \tag{11.12}$$

$$\widetilde{W} = \left\lfloor \frac{W + 2p - f}{s} + 1 \right\rfloor. \tag{11.13}$$

11.2.3 Convolution over Volumes in the CNN

Firstly, we shall remember a relationship between convolution (11.6) and the inner product in expression (11.1) given by:

$$\langle u, h \rangle (m, n) = \sum_{(k,l) \in W} h(k, l)\, u(m + k, n + l) = u(m, n) \circledast h(-m, -n) \tag{11.14}$$

where $u \in \mathbb{R}^{H \times W}$ is represents the input image and $h \in \mathbb{R}^{f \times f}$ represents the filter. The expression (11.14) allows to write the convolutions using inner products.

For instance, given a volume (tensor) $u \in \mathbb{R}^{W \times H \times D}$ with D even, the convolutions in a CNN layer follows:

1) Filter dimension must be such that the number of channels in the input and filter depth are the same; that means $\{h(m, n, s)\} \in \mathbb{R}^{f \times f \times D}$.

2) We must define the discrete filter $h : W \to \mathbb{R}$, where:

$$W = \{-\xi, \ldots, -1, 0, 1, \ldots, \xi\}^2 \times \left\{ \left\lfloor -\frac{D}{2} \right\rfloor, \ldots, -1, 0, 1, \ldots, \left\lfloor \frac{D}{2} \right\rfloor \right\} \tag{11.15}$$

with value $f = 2\xi + 1$ like before and matrix $\{h(m,n,s)\}_{(m,n,s)\in W}$ is named the filter mask. The discrete convolutions is computed by:

$$v(m,n) = \sum_{(k,l,t)\in W} h(k,l,t)\, u(m+k, n+l, \bar{s}+t) = u(m,n,\bar{s}) \circledast h(-m,-n,-t), \tag{11.16}$$

where $\bar{s} = median\{1, 2, \ldots, D\}$ (see Figure 11.1).

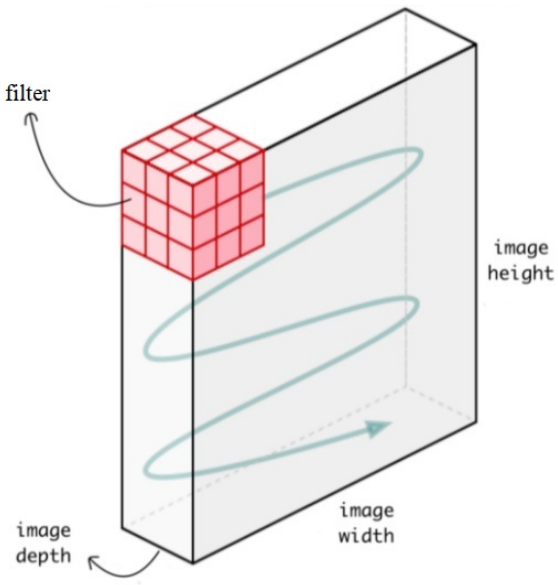


Figure 11.1: Representation of the convolution over a volume in the CNN.

Expression (11.14) opens the door for generalization of the convolution operation following the kernel trick (section 10.3.1). Specifically, given a kernel k, we can replace the inner product in expression (11.1) to its kernel generalization:

$$\langle u, h \rangle (m,n) \equiv k(u(m,n), h(-m,-n)), \tag{11.17}$$

For example, if the kernel k is the RBF one (expression (10.66)), the equation is computed by:

$$k(u(m,n), h(-m,-n)) = \exp\left(\frac{||\mathbf{u}(m,n) - \mathbf{h}(-m,-n)||^2}{2\sigma^2}\right), \tag{11.18}$$

where $\mathbf{u}(m,n)$ and $\mathbf{h}(-m,-n)$ are the arrays:

$$\mathbf{u}(m,n) = \{u(m+k,n+l)\}_{(k,l)\in W} \quad , \tag{11.19}$$

$$\mathbf{h}(-m,-n) = \{h(-k,-l)\}_{(k,l)\in W} \quad , \tag{11.20}$$

with the norm $||.||^2$ given by expression (11.2). In equations (11.19)-(11.20) we are supposing that the order in the set W is coerent with the convolution operation. Generalization of CNN architectures are obtained using the the kernel trick [145] aiming to learn how to approximate kernel feature map on training data.

11.2.4 Pooling Operation

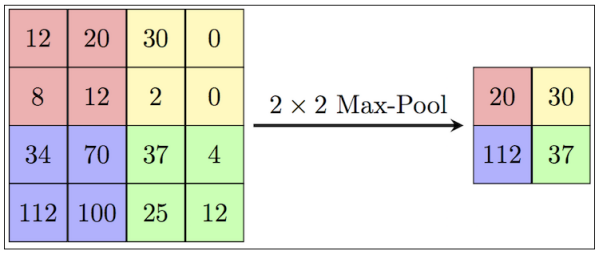
In order to reduce data dimension, a pooling operation can be appied. It performs subsampling of an input image $u \in \mathbb{R}^{H\times W}$. It uses stride s, following expressions (11.10)-(11.11) as well as padding p. So, the final image resolution is given by expressions (11.12)-(11.13). Its computation is based on the above expression:

$$\widetilde{u}(i,j) = pool\{u(m,n),\ (m,n) \in \mathcal{R}_{ij}\}, \tag{11.21}$$

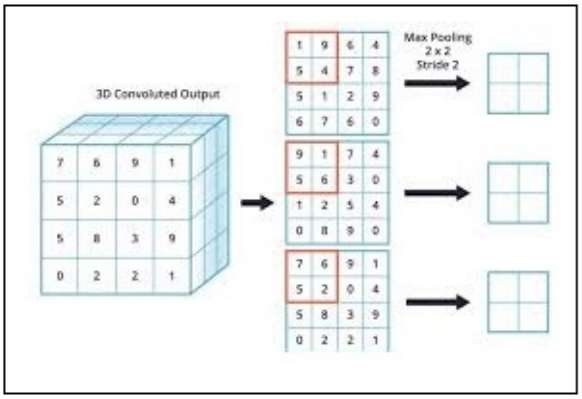
with $\mathcal{R}_{ij}$ being a neighborhood of pixel (i,j), with size $n_p \times n_p$ and $pool$ returns the a value ($\widetilde{u}(i,j)$) generated through the sub-image in $\mathcal{R}_{ij}$. Usually, $\widetilde{u}(i,j)$ is the maximum (max-pooling) or the mean value (average pooling) of the pixels inside $\mathcal{R}_{ij}$. Figure 11.2.(a) shows the application of max-pooling in the image 4×4, with stride $s = 2$ and padding $p = 0$. Figure 11.2.(b) performs this operation over a volume (tensor) by just applying max-pooling over each data plane separately. See [146] for a review about pooling operations.

11.3 Convolutional Neural Networks - CNN

A CNN can be seen as a special case of feedforward neural networks that has convolutions as one of their basic building blocks [93]. Pioneered by the work of LeCun et al. [147], CNNs are regarded as state-of-the-art in deep learning application to images [93].



(a)



(b)

Figure 11.2: (a) max-pooling with size 2×2 , stride $s = 2$, padding $p = 0$. (b) Application of max-pooling in a tensor: each data plane is processed independently.

A specific CNN architecture is represented in Figure 11.3. The first layer ($l = 0$), the input one, receives the image, here denoted by a tensor $\boldsymbol{z}^0 \in \mathbb{R}^{m_0 \times \eta_0 \times \beta_0}$, where $m_0 = \eta_0 = 100$ and $\beta_0 = 3$ in the Figure 11.3. The input image is processed by the CNN that is composed by two convolutional layers ($l = 1, 3$), formed by $n_1 = 50$ and $n_3 = 10$ filters, represented by real matrices $W_{k_l}^l \in \mathbb{R}^{3 \times 3 \times d_l}$, $k_l = 1, 2, \ldots, n_l$, $l = 1, 3$, where $n_l = 50$, $n_3 = 10$, d_l is the deph of each filter mask. In the case of Figure 11.3 $d_1 = 3$ and $d_3 = 50$.

In this example, we are setting the padding $p = 1$ and the stride $s = 1$ in expressions (11.12)-(11.13) for all convolution operations. Hence, the convolutional layer $l = 1$ outputs 50 filtered images, each one with size 100×100 (that is why the notation $100 \times 100 \times 50$ appears in the figure). Also, we have bias $\boldsymbol{b}_{k_l}^l = b_{k_l}^l \cdot \mathbf{1} \in \mathbb{R}^{m_l \times m_l}$, where $\mathbf{1}$ is the matrix with all entries igual to one, $b_{k_l}^l$ is a

scalar value, $l = 1, 3$, with $m_1 = 100$ and $m_3 = 50$, as we will see next, and .

The results of the convolutions are processed by activation functions $\varphi : \mathbb{R} \rightarrow [a, b]$ (see section 9.4), in a process that can be formally written as:

$$\boldsymbol{z}_{k_l}^l = \varphi\left(W_{k_l}^l \circledast \boldsymbol{z}^{l-1} + \boldsymbol{b}_{k_l}^l\right),\ k_l = 1, 2, \ldots, n_l,\ l = 1, 3, \tag{11.22}$$

where φ operates element-wise and '$\circledast$' denotes the convolution operation [93]. Each $\boldsymbol{z}_{k_l}^l$ generated in expression (11.22) is named a feature map.

Besides, the network includes a pooling layer ($l = 2$ in Figure 11.3) that performs subsampling, as follows:

$$z_{i,j,k_{l+1}}^{l+1} = pool\left\{\boldsymbol{z}_{m,n,k_l}^l,\ (m, n) \in \mathcal{R}_{ij}\right\},\ l = 1, 3,\ k_{l+1}, k_l = 1, 2, \ldots, n_l, \tag{11.23}$$

with $\mathcal{R}_{ij}$ being a neighborhood of pixel (i, j), with size 2×2 in our implementation, and $pool$ returns the maximum or the mean value inside $\mathcal{R}_{ij}$. In any case, the frames that enter pooling layer $l = 2$ are reduced to resolution 50×50, generating a block with size $50 \times 50 \times 50$ that is the input data to layer $l = 3$ that is a another convolution layer. The corresponding filters are tensor $W_{k_3}^3 \in \mathbb{R}^{3\times3\times d_3}$and $d_3 = 50$. Moreover, the bias $\boldsymbol{b}_{k_3}^3 \in \mathbb{R}^{50\times50}$ where $k_3 = 1, 2, \ldots, 10$. Then, it is applied the activation function, following expression (11.22) with $l = 3$. The feature maps generated enter the pooling layer that reduces their resolution and generate a tensor $\boldsymbol{z}^4 \in \mathbb{R}^{25\times25\times10}$.

Next, $\boldsymbol{z}^4$ is vectorized by a reshape operation that yield an array $\boldsymbol{v}^4 \in \mathbb{R}^{25\cdot25\cdot10}$ that is the input of a MLP attached to the last pooling layer. This MLP, also named fully connected neural network (FCNN) that performs the classification of the input data $\boldsymbol{z}^0$.

A general CNN architecture has as learnable parameters the filters $W_{k_l}^l \in \mathbb{R}^{s_l\times s_l\times d_l}$, bias $b_{k_l}^l \in \mathbb{R}$, with $l = 1, 2, \ldots, M$ and $k_l = 1, 2, \ldots, n_l$ as well as the MLP weights and bias arrays represented by $\left\{w_{jk}^{(i)}\right\}_{1\leq k\leq\varsigma_i}^{1\leq j\leq\varsigma_{i-1}}$, $\boldsymbol{b}^{(i)} = \left(b_1^{(i)}, b_2^{(i)}, \ldots, b_{n_i}^{(i)}\right)$, $i = 1, 2, \ldots L+1$, respectively (see section 9.3). These arrays encompass the parameters of the network, that can be assembled in only one array $\boldsymbol{\Theta}$, that should be obtained in the training step by solving the problem:

$$\boldsymbol{\Theta}^* = \arg\min_{\boldsymbol{\Theta}} Loss\left(\boldsymbol{\Theta}, \mathbb{D}_{tr}\right), \tag{11.24}$$

where $Loss$ is the loss in expression (8.3) and $\mathbb{D}_{tr}$ is the training set.

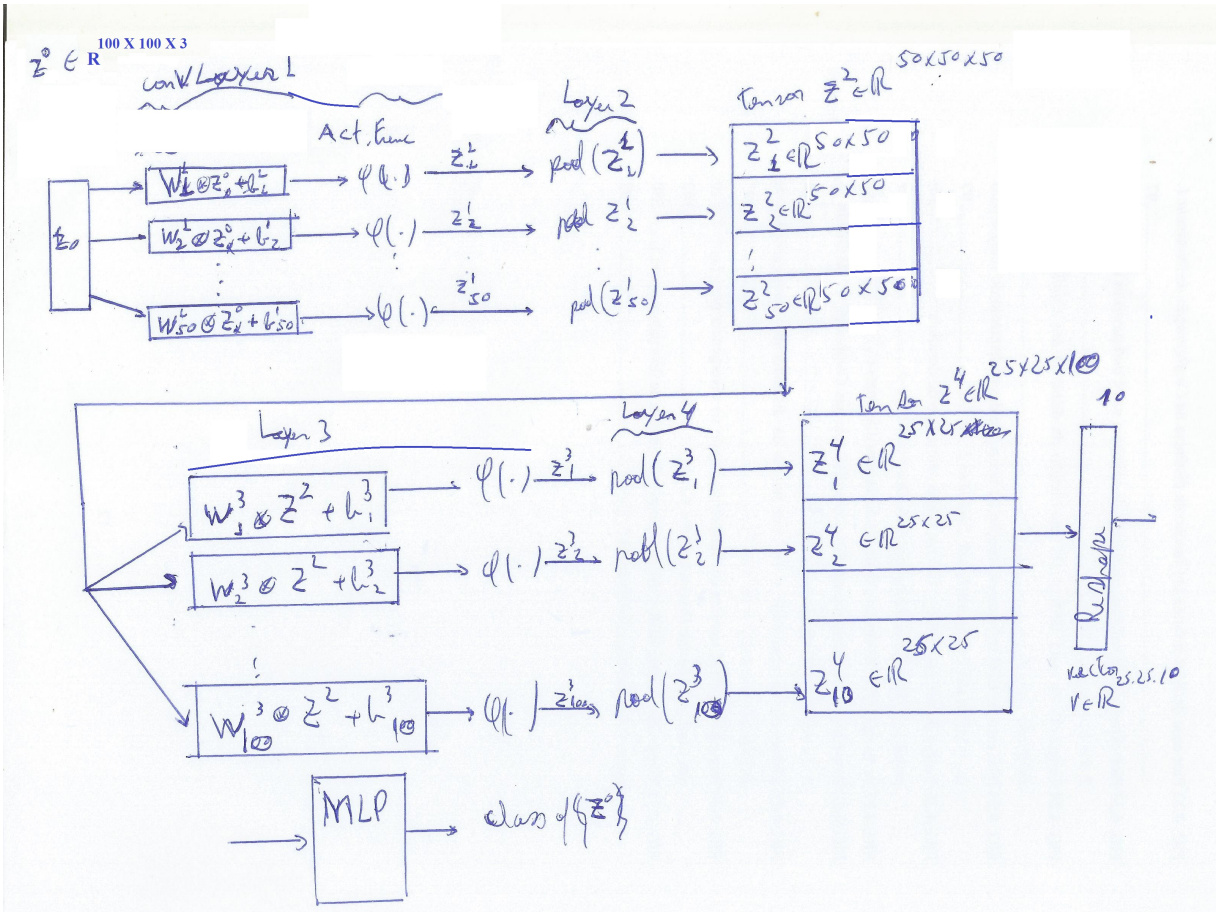


Figure 11.3: CNN architecture with two convolutional layers.

Given an input image, the computational cost of the network is dominated by the convolutional layers. Moreover, CNN parameters:

$$W = \left\{ \left(W_{k_l}^l, b_{k_l}^l\right)_{1 \le k_l \le n_l}^{1 \le l \le M} \right\} \cup \left\{ \left\{ w_{jk}^{(i)} \right\}_{1 \le k \le \varsigma_i}^{1 \le j \le \varsigma_{i-1}}, \boldsymbol{b}^{(i)} \right\}, \tag{11.25}$$

are obtained in the training process by using a gradient descent method, like Algorithms 3 and 4. The back-propagation algorithm is applied to efficiently calculate the gradient of expression (11.38), with respect to its parameters (see section 9.7). The computational cost of this process is defined by the backpropagation algorithm.

When a single input image $\boldsymbol{z}^0 \in \mathbb{R}^{m_0 \times \eta_0 \times \beta_0}$ passes to the network the compu-

tational complexity of all M convolutional layers is [148]:

$$O\left(\sum_{l=1}^{M} n_{l-1} \cdot s_l^2 \cdot n_l \cdot m_l \cdot \eta_l\right), \tag{11.26}$$

where n_{l-1} and n_l represent the number of filters of the convolutional layers $l-1$ and l, respectively; s_l denotes the the spatial size (length) of the filters in the l-th layer, and (m_l, η_l) gives the spatial resolution of the output feature map of the layer l.

Moreover, by considering Figure 9.6, the computational complexity of the FCNN (or MLP) component of the CNN is:

$$O\left(\sum_{l=1}^{L+1} p_{l-1}^2 \cdot p_l\right), \tag{11.27}$$

where we must remember that $p_0 = m_l \cdot \eta_l$.

So, if the training process lasts N_{epoch}, the forward computation steps encompasses a computational complexity of:

$$N_{epoch} \cdot N \cdot O\left(\sum_{l=1}^{M} n_{l-1} \cdot s_l^2 \cdot n_l \cdot m_l \cdot \eta_l + \sum_{l=1}^{L+1} p_{l-1}^2 \cdot p_l\right). \tag{11.28}$$

The complexity of the backpropagation algorithm (section 9.7) is linear in the cardinality of W; that means, it is $O(W)$ (see section 4.16 of [113]). The initialization of the network parameters (filters and FCNN weights and bias) can be performed through the procedures of section 9.10.

11.4 Generative Adversarial Network - GAN

Before enter in the GAN subject, we shall revise some elements in distribution theory. So, we let a probability distribution, or probability density function p over a data space χ:

$$p : \chi \longrightarrow [0, 1],$$

$$\int_{\chi} p(x)\, d\mu(x) = 1.$$

In this context, we can compute the following quantities:
Entropy:

$$E\left(p\right)=-\int p\left(x\right)\log\left(p\left(x\right)\right)d\mu\left(x\right).$$

Kullback-Leibler (KL) divergence [149]:

$$KL\left(p_r||p_\alpha\right)=\int\log\left(\frac{p_r\left(x\right)}{p_\alpha\left(x\right)}\right)p_r\left(x\right)d\mu\left(x\right)$$

$$=\int\log\left(p_r\left(x\right)\right)p_r\left(x\right)d\mu\left(x\right)-\int\log\left(p_\alpha\left(x\right)\right)p_r\left(x\right)d\mu\left(x\right)$$

Jensen-Shannon (JS) divergence [150]:

$$JS\left(p_r||p_g\right)=KL\left(p_r\left(x\right)||\left(\frac{p_r\left(x\right)+p_g\left(x\right)}{2}\right)\right)+KL\left(p_g\left(x\right)||\left(\frac{p_r\left(x\right)+p_g\left(x\right)}{2}\right)\right)$$

Therefore:

$$JS\left(p_r||p_g\right)=$$

$$\int\log\left(\frac{2p_r\left(x\right)}{p_r\left(x\right)+p_g\left(x\right)}\right)p_r\left(x\right)d\mu\left(x\right)+\int\log\left(\frac{2p_g\left(x\right)}{p_r\left(x\right)+p_g\left(x\right)}\right)p_g\left(x\right)d\mu\left(x\right)$$

$$=\log 4+\int\log\left(\frac{p_r\left(x\right)}{p_r\left(x\right)+p_g\left(x\right)}\right)p_r\left(x\right)d\mu\left(x\right)+\int\log\left(\frac{p_g\left(x\right)}{p_r\left(x\right)+p_g\left(x\right)}\right)p_g\left(x\right)d\mu\left(x\right)$$

$$=\log 4+\mathbb{E}_{x\sim p_r(x)}\left[\log\left(\frac{p_r\left(x\right)}{p_r\left(x\right)+p_g\left(x\right)}\right)\right]+\mathbb{E}_{x\sim p_g(x)}\left[\log\left(\frac{p_g\left(x\right)}{p_r\left(x\right)+p_g\left(x\right)}\right)\right] \tag{11.29}$$

11.4.1 GAN Overview

In [112] it is proposed an approach for training generative models through an adversarial process. In this framework two models (networks) are simultaneously

trained: a generative model G that learns the distribution of the original data, and a discriminative network D that evaluates the probability that a sample came from the training data rather than from G. The key idea of the training process is that G eventually maximizes the probability of D making a mistake. Figure 11.4 shows an overview of the training process in the GAN framework. The generator G receives random sample drawn from a distribution $p_z(\boldsymbol{z})$, where $\boldsymbol{z} \in Z \subset \mathbb{R}^k$ is named noise in Figure 11.4. So, we can write the generator as a function $G(\theta_g; \boldsymbol{z})$ where θ_g are the internal parameter of the corresponding network and $G(\theta_g; \boldsymbol{z}) \in \chi$, with χ being the data space. The discriminator is another network that computes a scalar function $D(\theta_d; \boldsymbol{x})$ where $\boldsymbol{x} \in \chi$ represents an object in the original data space. Also we suppose that we have a data generating distribution $p_{data}(\boldsymbol{x})$.

During the training process, the discriminator receives samples drawn from the distribution $p_{data}(\boldsymbol{x})$ as well as samples generated by the generator G, as shown in Figure 11.4. The parameter update is obtained through a minimax optimization process, described next.

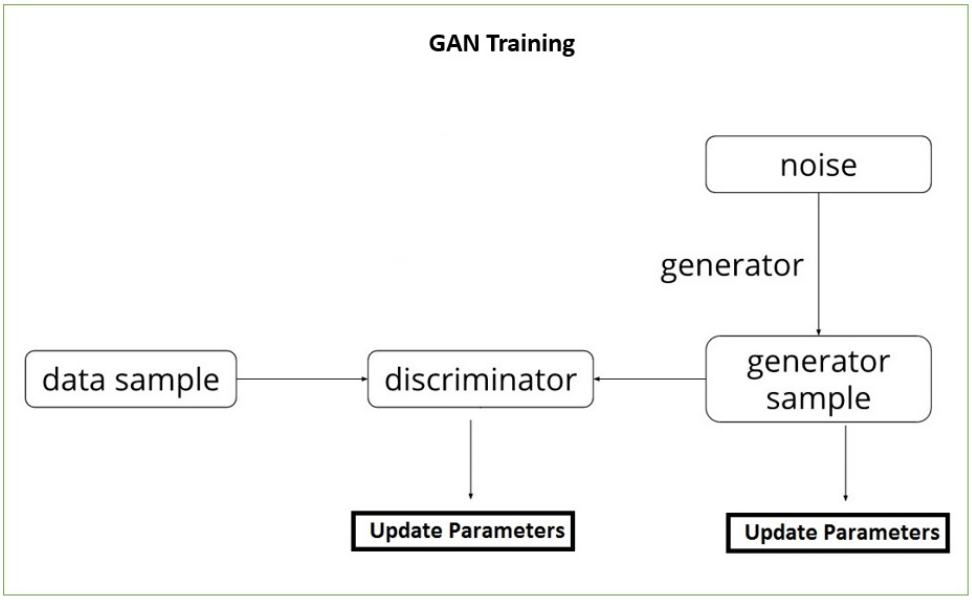


Figure 11.4: Representation of training process for GANs.

11.4.2 Original GAN Model

GAN formulation is based on the problem:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}(x)} \left(\log \left(D \left(\theta_d; \boldsymbol{x} \right) \right) \right) + \mathbb{E}_{z \sim p_z(z)} \left(\log \left(1 - D \left(G \left(\theta_g; z \right) \right) \right) \right) \right]. \tag{11.30}$$

To simplify notation, we follow [112] and re-write expression (11.30) as:

$$\min_{G} \max_{D} \left[\mathbb{E}_{x \sim p_{data}(x)} \left(\log \left(D \left(x \right) \right) \right) + \mathbb{E}_{z \sim p_z(z)} \left(\log \left(1 - D \left(G \left(z \right) \right) \right) \right) \right]. \tag{11.31}$$

Firstly, we define:

$$V \left(G, D \right) = \mathbb{E}_{x \sim p_{data}(x)} \left[\log \left(D \left(x \right) \right) \right] + \mathbb{E}_{z \sim p_z(z)} \left[\log \left(1 - D \left(G \left(z \right) \right) \right) \right], \tag{11.32}$$

or:

$$V \left(G, D \right) = \int_{\chi} p_{data} \left(x \right) \log \left(D \left(x \right) \right) dx + \int_{Z} p_z \left(z \right) \log \left(1 - D \left(G \left(z \right) \right) \right) dz, \tag{11.33}$$

Therefore, if G is known, we can write expression (11.33) as:

$$V \left(D \right) = \int_{\chi} p_{data} \left(x \right) \log \left(D \left(x \right) \right) dx + \int_{\chi} p_g \left(x \right) \log \left(1 - D \left(x \right) \right) dx$$

$$= \int_{\chi} \left[p_{data} \left(x \right) \log \left(D \left(x \right) \right) + p_g \left(x \right) \log \left(1 - D \left(x \right) \right) \right] dx.$$

Let us define the function:

$$F \left(D \left(x \right) \right) = p_{data} \left(x \right) \log \left(D \left(x \right) \right) + p_g \left(x \right) \log \left(1 - D \left(x \right) \right). \tag{11.34}$$

Hence, the optimum discriminator D is defined by the associated Euler-Lagrange equation:

$$\frac{\partial F}{\partial D} = p_{data} \left(x \right) \frac{1}{D \left(x \right)} - p_g \left(x \right) \frac{1}{\left(1 - D \left(x \right) \right)} = 0,$$

which renders:

$$D^{*}\left(x\right)=\frac{p_{data}\left(x\right)}{p_{data}\left(x\right)+p_{g}\left(x\right)} \tag{11.35}$$

By inserting this result in expression (11.32) we can obtain:

$$C\left(p_{g}\right)=\mathbb{E}_{x\sim p_{data}(x)}\left[\log\left(\frac{p_{data}\left(x\right)}{p_{data}\left(x\right)+p_{g}\left(x\right)}\right)\right]+\mathbb{E}_{x\sim p_{g}(x)}\left[\frac{p_{g}\left(x\right)}{p_{data}\left(x\right)+p_{g}\left(x\right)}\right], \tag{11.36}$$

which must be minimized in p_g. Comparing with expression (11.29) we notice that:

$$C\left(p_{g}\right)=-\log 4+JS\left(p_{data}||p_{g}\right). \tag{11.37}$$

Theorem 1: The global minimum of $C\left(p_{g}\right)$is achieved if and only if $p_{data}=p_{g}$ and, at this point, $C\left(p_{g}\right)=-\log 4$.

In fact, if $p_{data}=p_{g}$ in equation (11.36) we obtain $C\left(p_{g}\right)=-\log 4$ due to expression (11.37).

Besides, once $JS\left(p_{data}||p_{g}\right)\geq 0$ and $JS\left(p_{data}||p_{g}\right)=0$ if and only if $p_{data}=p_{g}$ we conclude that:

$$C\left(p_{g}\right)=-\log 4+JS\left(p_{data}||p_{g}\right)\geq-\log 4,\quad\forall p_{data},p_{g},$$

and $C\left(p_{g}\right)=-\log 4$ if and only if $p_{data}=p_{g}$.

The minibatch stochastic gradient descent training for GANs is shown in Figure 11.5

11.5 Convolutional Autoencoder - CAE

An autoencoder can be seen as a special case of feedforward neural network that is trained to reproduce its input at the output layer [93]. An autoencoder consists of two parts: encoder and decoder. The encoder is responsible for data codification, while the decoder must reconstruct the input data from the representation generated at the encoder output.

GAN Training Algorithm

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter.

for number of training iterations **do**
  for k steps **do**
    • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
    • Update the discriminator by ascending its stochastic gradient:

$$\theta^i{}_d = \theta_d^{i-1} + \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for
  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  • Update the generator by descending its stochastic gradient:

$$\theta^i{}_g = \theta_g^{i-1} - \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

https://arxiv.org/abs/1406.2661

Figure 11.5: Training procedure for GANs.

A convolutional autoencoder (CAE) is a deep unsupervised network that is built through CNNs, which, in turn, have convolutions as one of their basic building blocks, as described in section 11.3.

Figure 11.6 shows an example of CAE . The first layer ($l = 0$), the input one, receives the image, here denoted by $\boldsymbol{z}^0 \in \mathbb{R}^{m_0 \times m_0 \times 1}$, where $m_0 = 100$ in the Fig. 11.6. The input image is processed by the encoder that is composed by two convolutional layers ($l = 1, 3$), formed by $n_1 = 50$ and $n_3 = 100$ filters, represented by real matrices $W_{k_l}^l \in \mathbb{R}^{3 \times 3 \times d_l}$, $k_l = 1, 2, \ldots, n_l$, $l = 1, 3$, and $d_1 = 1$, $d_3 = 50$. In the CAE example of Figure 11.6, we are supposing padding $p = 1$ and stride $s = 1$ for all convolutions (see expressions (11.12)-(11.13)). Hence, the convolutional layer $l = 1$ outputs 50 filtered images, each one with

size 100×100 (that is why the notation $100 \times 100 \times 50$ appears in the figure). Also, we bias $b_{k_l}^l \in \mathbb{R}$, $l = 1, 3$. The results of the convolutions are processed by activation functions $\varphi : \mathbb{R} \to [a, b]$, likewise in expression (11.22). Besides, the encoder network includes a pooling layer ($l = 2$) that performs subsampling, following equation (11.23).

Hence, the feature maps that enter pooling layer $l = 2$ are reduced to resolution 50×50, generating a block with size $50 \times 50 \times 50$ that is the input data to layer $l = 3$. The feature maps yielded in the encoder output (layer $l = 3$) compose the so called latent space.

The decoder network is composed by layers $l = 4, 5, 6$, where $l = 4, 6$ are convolutional ones, followed by activation functions like in the encoder, in a processing chain formalized also through equation (11.22). The layer $l = 5$ encompasses upsampling operations $\boldsymbol{y}_{k_l}^l = upsampling\left(\boldsymbol{z}_{k_{l-1}}^{l-1}\right)$, where $\boldsymbol{y}_{k_l}^l \in \mathbb{R}^{2m_{k_{l-1}} \times 2m_{k_{l-1}}}$ because $\boldsymbol{z}_{k_{l-1}}^{l-1} \in \mathbb{R}^{m_{k_{l-1}} \times m_{k_{l-1}}}$ (see [151] for details).

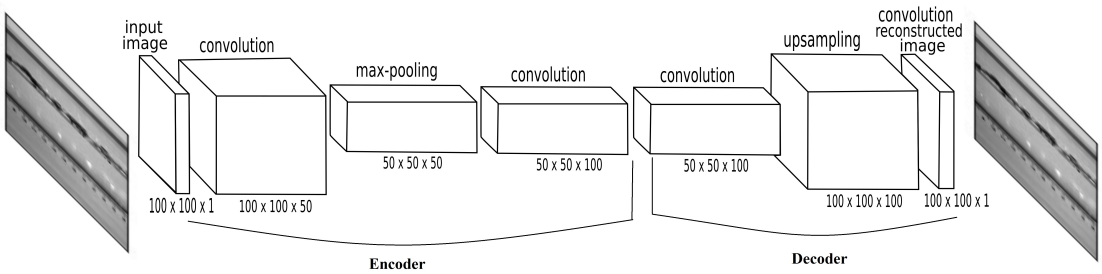


Figure 11.6: Components of CAE architecture: Input layer ($l = 0$), that receives the input image; convolutional layers ($l \in \{1, 3, 4, 6\}$), made of convolution kernels with size 3×3; pooling that reduces the dimensionality of the feature maps through expression (11.23); upsampling that increases image resolution.

A general CAE architecture includes filters $W_{k_l}^l \in \mathbb{R}^{s_l \times s_l \times d_l}$, and bias $b_{k_l}^l \in \mathbb{R}$, with $l = 1, 2, \ldots, 2M$ and $k_l = 1, 2, \ldots, n_l$. These matrices compose the parameters of the network, assembled in an array $\boldsymbol{\Theta}$, that should be obtained in the training step by solving the problem:

$$\boldsymbol{\Theta}^* = \arg\min_{\boldsymbol{\Theta}} \sum_{i=1}^{N} L\left(\boldsymbol{z}_i, \hat{\boldsymbol{z}}_i\right), \quad with, \quad L\left(\boldsymbol{z}, \hat{\boldsymbol{z}}\right) = \|\boldsymbol{z} - \hat{\boldsymbol{z}}\|_F^2, \tag{11.38}$$

where $\boldsymbol{z}_i$ represents an input image, $\hat{\boldsymbol{z}}_i$ is the reconstruction of $\boldsymbol{z}_i$ obtained in the decoder output.

Appendix A

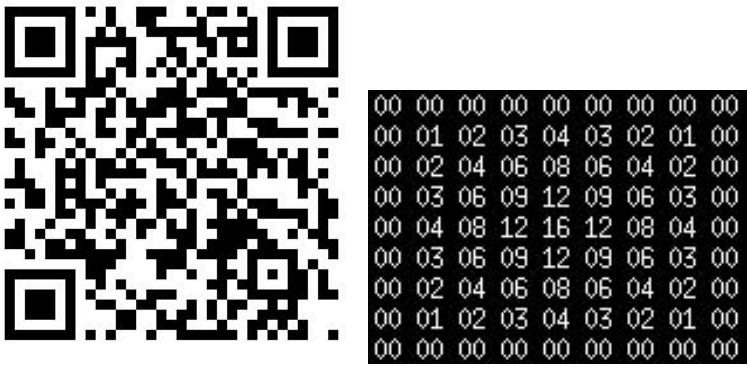
Convolutions and Linear Processes

In this text, we focus on mathematical concepts used in the study of two-dimensional linear processes. Digital images are generally the output of two-dimensional systems and, consequently, we need the mathematical concepts used in the study of such systems.

A digital image can be represented as a two dimensional matrix. Its elements are called **pixels** and to each pixel it is associated color channels or a gray-level value. In the case of color images, an usual color system is the RGB one. For full color displays we have 8 bits for each color channel. Therefore, we have 24 bits for color representation.

Grey-level images are, in general, represented by 8 bits. So, we can assign intensities $I \in \{0, 1, ..., 255\}$ to each pixel. Figure A.1.a is a example of a gray-level image and Figure A.1.b pictures the corresponding matrix of intensities.

In what follows, we will focus on gray-level images. The extension for color images is straightforward. Basic references for this chapter are [37] and Chapter 5 of reference [5]. We start with the continuous theory because a digital image can be seen as a discrete version of a continuous function (signal), that means, a function $f : D \subset \mathbb{R}^2 \longrightarrow \mathbb{R}$.



(a) (b)

Figure A.1: (a) Digital image visualization. (b) Intensities of the pixels in a digital image .

A.1 Two-Dimensional Signals and Digital Images

From the mathematical viewpoint, a continuous two-dimensional signal can be represented as a function:

$$\begin{aligned} f &: D \subset \Re^2 \to \Re; \\ (x,y) \quad \underset{\to}{f} \quad z &= f(x,y) \end{aligned} \tag{A.1}$$

where x, y are variable of interest (time , space, etc.). In the case of a continuous image, D would be the image domain and x, y cartesian coordinates to locate pixels (image points). In this case, usual operations to process the two-dimensional signal are the convolution and Fourier transform. The latter, and its inverse are given by:

$$\widehat{f}(\omega_1,\omega_2) = (\digamma f)(\omega_1,\omega_2) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\exp\left(-2\pi j\left(x\omega_1+y\omega_2\right)\right)dxdy, \tag{A.2}$$

$$f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \widehat{f}(\omega_1,\omega_2)\exp\left(2\pi j\left(x\omega_1+y\omega_2\right)\right)d\omega_1 d\omega_2. \tag{A.3}$$

The convolution operation is defined as:

$$g(x,y) = h(x,y) \circledast f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h(x-s, y-w)\, f(s,w)\, dsdw, \quad \text{(A.4)}$$

where h is the convolution kernel, also called **impulse response**. The elements of functional analysis behind this definition (functional space, inner product, etc.) can be found in [152].

In practice, we get a sampled image from the acquisition systems (digital camera, CT scanner, etc.). In this case, we have a two-dimensional array, which can be represented as a matrix, like in Figure A.1:

$$\mathbf{u}(m,n) = \begin{pmatrix} \mathbf{u}(0,0) & \mathbf{u}(0,1) & \cdots & \mathbf{u}(0,N-1) \\ & & \cdots & \\ & & \cdots & \\ \mathbf{u}(M-1,0) & \mathbf{u}(M-1,1) & \cdots & \mathbf{u}(M-1,N-1) \end{pmatrix} \quad \text{(A.5)}$$

Once a digital image can be represented as a matrix of intensities, we can apply discrete operations in order to transform the input signal. The discrete version of the convolution, applied to the input signal $\mathbf{u}(m,n)$, is defined by:

$$v(m,n) = h(m,n) \circledast \mathbf{u}(m,n) = \sum_{s=-\infty}^{+\infty}\sum_{w=-\infty}^{+\infty} h(m-s, n-w)\, f(s,w) \quad \text{(A.6)}$$

Likewise the one-dimensional case, we prove the convolution theorem, which is a very useful tool for linear filtering methods:

Theorem 1: The Fourier transform of the convolution of two functions is the product of their Fourier transforms, that means:

$$g(x,y) = h(x,y) \circledast f(x,y) \Leftrightarrow \widehat{g}(\omega_1,\omega_2) = \widehat{h}(\omega_1,\omega_2) \cdot \widehat{f}(\omega_1,\omega_2). \quad \text{(A.7)}$$

Besides another fundamental property is the following one:

Theorem 2: The inner product of two functions is equal to the inner product of their Fourier transforms, that is:

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h(x,y)\, f^*(x,y)\, dxdy = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \widehat{h}(\omega_1,\omega_2)\, \widehat{f}^*(\omega_1,\omega_2)\, d\omega_1 d\omega_2. \tag{A.8}$$

A trivial corolary of this result is the **Parseval energy conservation** formula, obtained when $h = f$:

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} |f(x,y)|^2\, dxdy = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \left|\widehat{f}(\omega_1,\omega_2)\right|^2 d\omega_1 d\omega_2. \tag{A.9}$$

Discrete counterparts of theorems 1 and 2 are obtained through the **Fourier transform of sequences** (arrays), defined as:

$$X(\omega_1,\omega_2) = \sum_{m=-\infty}^{+\infty}\sum_{n=-\infty}^{+\infty} x(m,n)\exp\left(-j\left(m\omega_1 + n\omega_2\right)\right), \quad -\pi \le \omega_1,\omega_2 \le \pi \tag{A.10}$$

$$x(m,n) = \frac{1}{4\pi^2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} X(\omega_1,\omega_2)\exp\left(j\left(m\omega_1 + n\omega_2\right)\right) d\omega_1 d\omega_2. \tag{A.11}$$

So, the following properties can be proved:

Theorem 3: The Fourier transform of the convolution of the two-dimensional sequences $h(m,n)$ and $x(m,n)$ is the product of their Fourier transforms, that means:

$$y(m,n) = h(m,n) \circledast x(m,n) \Leftrightarrow Y(\omega_1,\omega_2) = H(\omega_1,\omega_2)\cdot X(\omega_1,\omega_2). \tag{A.12}$$

Theorem 4: The inner product of the two-dimensional sequences $h(m,n)$ and $x(m,n)$ is equal to the inner product of their Fourier transforms, that is:

$$I = \sum_{m=-\infty}^{+\infty}\sum_{n=-\infty}^{+\infty} x(m,n)\, y^*(m,n) \Leftrightarrow I = \frac{1}{4\pi^2}\int_{-\pi}^{\pi}\int_{-\pi}^{\pi} X(\omega_1,\omega_2)\, Y^*(\omega_1,\omega_2)\, d\omega_1 d\omega_2. \tag{A.13}$$

Other important properties can be found in the Chapter 2 of reference [5] (see also the exercises bellow).

A.2 Exercises

1) Define the space $L^P(\Re^2)$ and its inner product. In the expression (A.13) we observe the factor $(1/4\pi^2)$. Why is it necessary?

2) Prove theorems 2, 3 and 4.

3) Prove the properties stated in the exercise 2.2, page 44, of the reference [5]?

4) Prove that, for two continuous signals h and f we have (multiplication property):

$$g(x,y) = h(x,y)\, f(x,y) \Leftrightarrow \widehat{g}(\omega_1,\omega_2) = \widehat{h}(\omega_1,\omega_2) \circledast \widehat{f}(\omega_1,\omega_2)$$

5) Take the standard bidimensional normal distribution (see page 32, reference [5]). Implement a low-pass filter; that means, its impulse response, based on the continuous (normalized) distribution.

6) Take a digital image and apply the filter obtained in the exercise 4. What happens?

7) Exercise 2.5, page 45, reference [5].

8) Would be possible to define low-pass, high-pass and high-pass filters for two-dimensional sequences, based on Fourier transform of sequences and the Theorem 3?

9) What happens with expressions (A.11), theorems 3 and 4, if we define the Fourier transform of sequences as:

$$X(\omega_1,\omega_2) = \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} x(m,n) \exp\left(j\left(m\omega_1 + n\omega_2\right)\right), \quad -\pi \leq \omega_1, \omega_2 \leq \pi \tag{A.14}$$

10) Prove the conjugation, separability, scaling and shifting properties for Fourier transform, stated on Table 2.3, page 17, of reference [5].

Appendix B

Kuhn-Tucker Theorem

Kuhn and Tucker generalized the Lagrange multipliers method by considering the so-called convex optimization problem, where one minimizes a convex objective function under convex constraints formed through inequalities [7]. A nonempty set $U \subset \mathbb{R}^n$ is called convex if, given any two points $\boldsymbol{x}, \boldsymbol{y} \in U$, we have:

$$\{\boldsymbol{z};\ \boldsymbol{z} = \alpha\boldsymbol{x} + (1-\alpha)\,\boldsymbol{y},\ 0 \leq \alpha \leq 1\} \subset U. \tag{B.1}$$

Moreover, a function $f : U \longrightarrow \mathbb{R}$ is convex if for any two points $\boldsymbol{x}, \boldsymbol{y} \in U$ the inequality:

$$f\left(\alpha\boldsymbol{x} + (1-\alpha)\,\boldsymbol{y}\right) \leq \alpha f\left(\boldsymbol{x}\right) + (1-\alpha)\, f\left(\boldsymbol{y}\right), \quad 0 \leq \alpha \leq 1,$$

holds true.

Now, let us consider the following problem:

$$\begin{aligned} &\min_{\boldsymbol{x}\in U} && f_0(\boldsymbol{x}) \\ &\text{subject to} && f_k(x) \leq 0, \quad k = 1, 2, \ldots, m, \end{aligned} \tag{B.2}$$

where $f_k : U \longrightarrow \mathbb{R}$, $k = 0, 1, 2, \ldots, m$, are convex functions and $U \subset \mathbb{R}^n$ is a nonempty convex set.

The solution of this problem is obtained by firstly considering the Lagrangian [7]:

$$L(\boldsymbol{x}, \lambda_0, \boldsymbol{\lambda}) = \sum_{k=0}^{m} \lambda_k f_k(\boldsymbol{x}), \tag{B.3}$$

where $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_m)$.

Theorem (Kuhn-Tucker). If $\boldsymbol{x}^* \in \mathbb{R}^n$ solves the optimization problem (B.2) then there exist Lagrange multipliers λ_0^* and $\boldsymbol{\lambda}^* = (\lambda_1^*, \lambda_2^*, \ldots, \lambda_m^*)$ such that:

1. $\lambda_k^* \geq 0, \quad k = 0, 1, \cdots, m$ and $\max\{\lambda_k^* \geq 0, \quad k = 0, 1, \cdots, m\} > 0$,
2. $\nabla_x L(x^*, \lambda_0^*, \boldsymbol{\lambda}^*) = 0$,
3. The Kuhn-Tucker conditions:

$$\lambda_k^* f_k(\boldsymbol{x}^*) = 0, \quad k = 0, 1, \cdots, m$$

If $\lambda_0^* \neq 0$ and conditions (1)-(3) hold true, then $\boldsymbol{x}^*$ is solution of the convex optimization problem given by expression (B.2). Moreover, sufficient conditions to guarantee that $\lambda_0^* \neq 0$ is that there exists $\boldsymbol{z} \in \mathbb{R}^n$ such that:

$$f_k(\boldsymbol{z}) < 0, \quad k = 1, \cdots, m,$$

named Slater conditions. In this case, we can choose $\lambda_0^* = 1$ and the Lagrangian in expression (B.3) takes the form:

$$L(\boldsymbol{x}, 1, \boldsymbol{\lambda}) = f_0(\boldsymbol{x}) + \sum_{k=1}^{m} \lambda_k f_k(\boldsymbol{x}), . \tag{B.4}$$

Bibliography

[1] P. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.

[2] Ihab F. Ilyas and Xu Chu. *Data Cleaning*. ACM, 2019.

[3] Stephen Marsland. *Machine Learning: An Algorithmic Perspective*. Chapman & Hall/CRC, 2009.

[4] Lawrence K. Saul and Sam T. Roweis. Think globally, fit locally: Unsupervised learning of low dimensional manifolds. *J. Mach. Learn. Res.*, 4:119–155, December 2003.

[5] Anil K. Jain. *Fundamentals of Digital Image Processing*. Prentice-Hall, Inc., 1989.

[6] M.N. Murty and R. Raghava. *Support Vector Machines and Perceptrons: Learning, Optimization, Classification, and Application to Social Networks*. SpringerBriefs in Computer Science. Springer International Publishing, 2016.

[7] Vladimir N. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, INC., 1998.

[8] Business Dictionary. *Pattern Definition*. http://www.businessdictionary.com/definition/pattern.html, 2018.

[9] David L. Dowe. Mml, hybrid bayesian network graphical models, statistical consistency, invariance and uniqueness. In Prasanta S. Bandyopadhyay and Malcolm R. Forster, editors, *Philosophy of Statistics*, volume 7 of

Handbook of the Philosophy of Science, pages 901 – 982. North-Holland, Amsterdam, 2011.

[10] S. Geisser. *Modes of Parametric Statistical Inference*. John Wiley & Sons, 2006.

[11] R. Beale and T. Jackson. *Neural Computing*. MIT Press, 1994.

[12] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[13] Sridhar Mahadevan. *How is Statistical Learning different from Machine Learning?* https://www.quora.com/How-is-Statistical-Learning-different-from-Machine-Learning.

[14] Perficient. *Machine Learning Vs. Statistical Learning*. https://blogs.perficient.com/2018/01/29/machine-learning-vs-statistical-learning/.

[15] Kevin P. Murphy. *Machine learning: A Probabilistic Perspective*. MIT Press, Cambridge, Mass. [u.a.], 2013.

[16] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. 2017. cite arxiv:1708.05866Comment: IEEE Signal Processing Magazine, Special Issue on Deep Learning for Image Understanding (arXiv extended version).

[17] KDnuggets^{TM} News. *Six Steps to Master Machine Learning with Data Preparation*. https://www.kdnuggets.com/2018/12/six-steps-master-machine-learning-data-preparation.html.

[18] PULKIT SHARMA. *A Comprehensive Tutorial to learn Convolutional Neural Networks from Scratch*. `https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/`.

[19] Anselmo Ferreira and Gilson Giraldi. Convolutional neural network approaches to granite tiles classification. *Expert Systems with Applications*, 84:1–11, 2017.

[20] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, and Gang Wang. Recent advances in convolutional neural networks. *CoRR*, abs/1512.07108, 2015.

[21] B. Laure, B. Angela, and M. Tova. Machine learning to data management: A round trip. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1735–1738, 2018.

[22] T. Hastie, R. Tibshirani, and J.H. Friedman. The elements of statistical learning. *Springer*, 2001.

[23] R. L. Vaught. *Set Theory: An Introduction*. Birkhauser, Boston, 1985.

[24] E. L. Lima. *Curso de Análise*, volume 2. Livros Técnicos e Científicos Editora S.A., 1985.

[25] B. Russell. *Bertrand Russell and the Origins of the Set-Theoretic Paradoxes*. Birkhauser Verlag Basel, 1992.

[26] S. F. Barker. *Filosofia da Matemática*. Prentice-Hall Inc., 1976.

[27] Serge Lang. *Algebra*. Addison-Wesley Pub. Company, Inc., Yale University, New Haven, 1984.

[28] T.A. Sudkamp. *Languages and Machines: An Introduction to the Theory of Computer Science*. Addison-Wesley Publishing Company, INC., 1988.

[29] R. Garnier and J. Taylor. *Discrete Mathematics for New Technology*. Adam Hilger, 1992.

[30] C. S. Honig. *Alicações da Topologia à Análise*. Editora Edgar Blucher, Ltda, SP-Brasil, 1976.

[31] H.R. Lewis and C.H. Papadimitriou. *Elements of the Theory of Computation*. Prentice-Hall, Second Edition., 1998.

[32] M.L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, INC., 1967.

[33] M.D. Davis and E.J. Weyuker. *Computability, complexity, and languages: fundamentals of theoretical computer science*. New York Academic Press, 1983.

[34] J.L. Szwarcfiter and L. Markenzon. *Estruturas de Dados e seus Algoritmos*. Editora LTC., 1994.

[35] Henrik Reif Andersen. An introduction to binary decision diagrams. Technical report, www.itu.dk/people/hra/bdd97.ps, 1997.

[36] K. HOFFMAN and R. KUNZE. *Linear algebra*. Prentice Hall, 1971.

[37] R.C. Gonzalez. *Digital Image Processing*. Reading Addison-Wesley, 1992.

[38] G. Golub and C. Von Loan. *Matrix Computation*. The Johns Hopkins Univ. Press, 1985.

[39] Earl W. Swokowski. *Calculus with analytic geometry*. Prindle Weber and Schmidt, Boston, 1975.

[40] L. Leithold. *The calculus, with analytic geometry*. Number v. 1 in The Calculus, with Analytic Geometry. Harper & Row, 1972.

[41] A.T. Fomenko and S.P. Novikov. Modern geometry: Methods and applications. 1992.

[42] M. do Carmo and P. Roitman. *Geometria diferencial de curvas e superfícies*. Textos Universitários. Sociedade Brasileira de Matemática, 2006.

[43] C. Chapra and R.P. Canale. *Numerical Methods for Engineers.* MacGraw-Hill International Editions, 1988.

[44] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[45] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press., December 2000.

[46] J. Preskill. Quantum computation - caltech course notes. Technical report, http://www.theory.caltech.edu/people/preskill/ph229/, 2001.

[47] Aboul Ella Hassanien, Ahmad Taher Azar, Vaclav Snasael, Janusz Kacprzyk, and Jemal H. Abawajy, editors. *Big Data in Complex Systems*, volume 9 of *Studies in Big Data*. Springer, 2015.

[48] Techopedia. *Data Redundancy*. https://www.techopedia.com/definition/18707/data-redundancy.

[49] Alexander Kuleshov and Alexander Bernstein. *Proc. 10th International Conference, St. Petersburg, Russia, July 21-24*, chapter Manifold Learning in Data Mining Tasks, pages 119–133. 2014.

[50] Yunqian Ma and Yun Fu. *Manifold Learning Theory and Applications*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 2011.

[51] J. Wang. *Geometric Structure of High-Dimensional Data and Dimensionality Reduction*. Springer Berlin Heidelberg, 2012.

[52] John A. Lee and Michel Verleysen. *Nonlinear Dimensionality Reduction*. Springer Publishing Company, Incorporated, 1st edition, 2007.

[53] Alan Julian Izenman. *Modern Multivariate Statistical Techniques: Regression, Classification, and Manifold Learning*. Springer Publishing Company, Incorporated, 1 edition, 2008.

[54] Edson C. Kitani, C. E. Thomaz, and G. A. Giraldi. Geometric Elements of Manifold Learning. Technical report, National Laboratory for Scientific Computing and FEI Universitary Center, 2011.

[55] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41:15:1–15:58, 2009.

[56] Frank E. Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11(1):1–21, 1969.

[57] Gary L. Tietjen and Roger H. Moore. Some grubbs-type statistics for the detection of several outliers. *Technometrics*, 14(3):583–597, 1972.

[58] P. C. Mahalanobis. On the generalized distance in statistics. In *Proceedings of the National Institute of Sciences (Calcutta)*, 1936.

[59] Rosa Elvira Lillo Rodríguez, Henry Laniado Rodas, and Elisa Cabana Garceran del Vall. Multivariate outlier detection based on a robust mahalanobis distance with shrinkage estimators. https://doi.org/10.1007/s00362-019-01148-1, 2017.

[60] Jiliang Tang, Salem Alelyani, and Huan Liu. Feature selection for classification: A review. In *Data Classification: Algorithms and Applications*, 2014.

[61] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, New York, 1990.

[62] Tiene A. Filisbino, Gilson A. Giraldi, and Carlos E. Thomaz. Comparing ranking methods for tensor components in multilinear and concurrent subspace analysis with applications in face images. *Int. J. Image Graphics*, 15(1), 2015.

[63] Haiping Lu, K.N. Plataniotis, and A.N. Venetsanopoulos. Mpca: Multilinear principal component analysis of tensor objects. *Neural Networks, IEEE Transactions on*, 19(1):18–39, Jan 2008.

[64] T. A. Filisbino, G. A. Giraldi, and C. E. Thomaz. Tensor fields for multilinear image representation and statistical learning models applications. In *2016 29th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials (SIBGRAPI-T)*, pages 24–37, Oct 2016.

[65] M Rasetti and E Merelli. The topological field theory of data: a program towards a novel strategy for data mining through data language. *Journal of Physics: Conference Series*, 626(1):012005, 2015.

[66] Junping Zhang, Hua Huang, and Jue Wang. Manifold learning for visualizing and analyzing high-dimensional data. *IEEE Intelligent Systems*, 25(4):54–61, 2010.

[67] John A. Lee and Michel Verleysen. *Nonlinear Dimensionality Reduction*. Springer Publishing Company, Incorporated, 1st edition, 2007.

[68] Nourhan Zayed and Heba A. Elnemr. Statistical analysis of haralick texture features to discriminate lung abnormalities. *Journal of Biomedical Imaging*, 2015:12:12–12:12, January 2015.

[69] R. M. Haralick, K. Shanmugam, and I. Dinstein. Texture features for image classification. *IEEE Transaction System Man Cybernet SMC-3*, 6:610–621, 1973.

[70] Richard Brereton. *Chemometrics for pattern recognition*. John Wiley & Sons, 2009.

[71] CarlosEduardo Thomaz, EdsonCaoru Kitani, and DuncanFyfe Gillies. A maximum uncertainty lda-based approach for limited sample size problems - with application to face recognition. *Journal of the Brazilian Computer Society*, 12(2):7–18, 2006.

[72] Manli Zhu and Aleix Martinez. Selecting principal components in a two-stage lda algorithm. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, 1:132–137, 2006.

[73] Anitha Sheela and Satya Prasad. Linear discriminant analysis f-ratio for optimization of tespar & mfcc features for speaker recongnition. *Journal of Multimedia*, 2007.

[74] Shuangge Ma and Jian Huang. Penalized feature selection and classification in bioinformatics. *Briefings in Bioinformatics*, 9(5):392–403, 2008.

[75] Ramón Díaz-Uriarte and Sara Alvarez de Andrés. Gene selection and classification of microarray data using random forest. *BMC Bioinformatics*, 7:3, 2006.

[76] Zena M. Hira and Duncan Fyfe Gillies. A review of feature selection and feature extraction methods applied on microarray data. *Adv. Bioinformatics*, pages 1–13, 2015.

[77] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46:389–422, 2002.

[78] I. Guyon, S. Gunn, M. Nikravesh, and L. A. Zadeh, editors. *Feature Extraction: Fundations and Applications*. 2006.

[79] N. Hoque, D.K. Bhattacharyya, and J.K. Kalita. Mifs-nd: A mutual information-based feature selection method. *Expert Systems with Applications*, 41(14):6371 – 6385, 2014.

[80] Nina Zhou and Lipo Wang. A modified t-test feature selection method and its application on the hapmap genotype data. *Genomics, Proteomics & Bioinformatics*, 5(3-4):242–249, 2007.

[81] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157–1182, 2003.

[82] CHRIS DING and HANCHUAN PENG. Minimum redundancy feature selection from microarray gene expression data. *Journal of Bioinformatics and Computational Biology*, 03(02):185–205, 2005.

[83] Mark A. Hall. Correlation-based feature selection for discrete and numeric class machine learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 359–366, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[84] T.A. Filisbino, ., G.A. Giraldi, and C.E. Thomaz. Approaches for multiclass discriminant analysis for ranking principal components. In *XII Workshop de Visao Computacional (WVC'16)*, Nov 2016.

[85] Igor Kononenko, Edvard Šimec, and Marko Robnik-Šikonja. Overcoming the myopia of inductive learning algorithms with relieff. *Applied Intelligence*, 7(1):39–55, 1997.

[86] Alan Jovic, Karla Brkic, and Nikola Bogunovic. A review of feature selection methods with applications. In *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1200–1205, May 2015.

[87] Z. Wu and J. Yu. Vector quantization: a review. *Frontiers Inf Technol Electronic Eng*, 20:507–524, 2019.

[88] Marius Arvinte, Sriram Vishwanath, and Ahmed H. Tewfik. Deep learning-based quantization of l-values for gray-coded modulation, 2019.

[89] A. Darvishi and H. Hassanpour. A geometric view of similarity measures in data mining. *IJE TRANSACTIONS C: Aspects*, 28(12):1728–1737, 2015.

[90] Haiping Lu, Konstantinos N. Plataniotis, and Anastasios N. Venetsanopoulos. A survey of multilinear subspace learning for tensor data. *Pattern Recogn.*, 44(7):1540–1551, July 2011.

[91] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Trans. on image processing*, 13(4):600–612, 2004.

[92] Mitchell H.B. *Image Fusion*, chapter Image Similarity Measures. 2010.

[93] I. Goodfellow, Y. Bengio, and A Courville. *Deep Learning*. MIT Press, 2016.

[94] Wikipedia. *Kullback-Leibler divergence*. https://en.wikipedia.org/wiki/Kullback-Leibler_divergence.

[95] John H. Reif and Stephen R. Tate. On threshold circuits and polynomial computation. *SIAM J. Comput.*, 21(5):896–908, 1992.

[96] Wikipedia. *Penalty method*. https://en.wikipedia.org/wiki/Penalty_method.

[97] Giraldi Gilson A. *Course Notes - Ano 2018*. https://www.lncc.br/∼gilson/NN-GB500/.

[98] Scikit-Learn. *Compute confusion matrix*. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html.

[99] Data-Science-Blog. *Performance Measures for Multi-Class Problems*. https://www.datascienceblog.net/post/machine-learning/performance-measures-multi-class-problems/.

[100] M. Hossin and M.N Sulaiman. A Review on Evaluation Metrics for Data Classification Evaluations. *International Journal of Data Mining & Knowledge Management Process (IJDKP)*, 5(2):1–11, November 2019.

[101] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern recognition*. Academic Press, 2003.

[102] Yair Goldberg, Alon Zakai, Dan Kushnir, and Ya'acov Ritov. Manifold learning: The price of normalization. *J. Mach. Learn. Res.*, 9:1909–1939, June 2008.

[103] Tong Lin and Hongbin Zha. Riemannian manifold learning. *IEEE Tran. on Patt. Analysis and Machine Intelligence*, 30(5):796–809, 2008.

[104] Gastão F. Miranda Jr., Gilson A. Giraldi, Carlos E. Thomaz, and Daniel Millan. Composition of local normal coordinates and polyhedral geometry in riemannian manifold learning. *IJNCR*, 5(2):37–68, 2015.

[105] S. Novikov B. Dubrovin, A. Fomenko. *Modern Geometry : Methods and Applications*. Springer,Verlag, 1990.

[106] M. P. Carmo. *Geometria Riemanniana*. Livros Técnicos e Científicos Editora S.A., 1979.

[107] John P. Cunningham and Zoubin Ghahramani. Linear dimensionality reduction: Survey, insights, and generalizations. *Journal of Machine Learning Research*, 16:2859–2900, 2015.

[108] Ke Sun and Stéphane Marchand-Maillet. An information geometry of statistical manifold learning. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 1–9, 2014.

[109] G. F. Miranda Jr., G. A. Giraldi, C. E. Thomaz, and R. D. Millan. Aprendizagem e síntese de variedades via coordenadas normais de riemann locais e baricentricas. In *Proc. of the ENIAC*, Fortaleza, Ceará, Brazil, 20th-24th October 2013.

[110] E.L. Lima. *Variedades diferenciáveis*. Monografias de matemática. Instituto Matemática Puro e Aplicada, Conselho Nacional de Pesquisas, 1973.

[111] Jun-Hai Zhai, Sufang Zhang, J. Chen, and Qiang He. Autoencoder and its various variants. *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 415–419, 2018.

[112] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2672–2680, 2014.

[113] Simon Haykin. *Neural Networks - A Comprehensive Foundation, Second Edition*. Prentice Hall, 2 edition, 1998.

[114] Leonid Datta. A survey on activation functions and their relation with xavier and he normal initialization. *CoRR*, abs/2004.06632, 2020.

[115] S. Kong and M. Takatsuka. Hexpo: A vanishing-proof activation function. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2562–2567, 2017.

[116] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814. Omnipress, 2010.

[117] Wikipedia. *Activation Function*. `https://en.wikipedia.org/wiki/Activation_function`, 2020.

[118] John S. Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In Franï¿½oise Fogelman-Souliï¿½ and Jeanny Hï¿½rault, editors, *NATO Neurocomputing*, volume 68 of *NATO ASI Series*, pages 227–236. Springer, 1989.

[119] Qi Wang, Yue Ma, Kun Zhao, and Yingjie Tian. A comprehensive survey of loss functions in machine learning. *Annals of Data Science*, 04 2020.

[120] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[121] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML'15: Proceedings of the 32nd International Conference on International Conference on Machine Learning*. JMLR.org, 2015.

[122] Carlos Eduardo Thomaz and Leo Leonel de Oliveira Junior. *FEI Face Image Database*. http://fei.edu.br/∼cet/frontalimages_manuallyaligned_part1.zip, 2005.

[123] Carlos Eduardo Thomaz and Leo Leonel de Oliveira Junior. *FEI Face Image Database*. http://fei.edu.br/∼cet/frontalimages_manuallyaligned_part2.zip, 2005.

[124] M L Astion, M H Wener, R G Thomas, G G Hunder, and D A Bloch. Overtraining in neural networks that interpret clinical data. *Clinical Chemistry*, 39(9):1998–2004, 11 2019.

[125] S. Lawrence and C. L. Giles. Overfitting and neural networks: conjugate gradient and backpropagation. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, volume 1, pages 114–119 vol.1, 2000.

[126] Jake Lever, Martin Krzywinski, and Naomi Altman. Points of significance: Model selection and overfitting. *Nature Methods*, 13(9):703–704, August 2016.

[127] Tom Fawcett. An introduction to roc analysis. *Pattern Recogn. Lett.*, 27(8):861–874, June 2006.

[128] Connor Shorten and T. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6:1–48, 2019.

[129] C.J. Huberty. *Applied Discriminant Analysis*. John Wiley & Sons, INC., 1994.

[130] A. Gelman and J. Hill. Data analysis using regression and multilevel/hierarchical models. 2007.

[131] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall/CRC, 1986.

[132] P.A. Devijver and J. Kittler. *Pattern Classification: A Statistical Approach*. Prentice-Hall, 1982.

[133] V.R. Algazi and D.J. Sakrison. On the optimality of karhunen-loeve expansion. *IEEE Trans. Information Theory*, pages 319–321, 1969.

[134] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3:71–86, 1991.

[135] Wei-Chien Chang. On using principal components before separating a mixture of two multivariate normal distributions. *Appl. Statist.*, 32(3):267–275, 1983.

[136] Thomas Hofmann, B. Scholkopf, and Alex Smola. Kernel methods in machine learning. *Annals of Statistics*, 36:1171–1220, 2008.

[137] Dino Sejdinovic and Arthur Gretton. *What is an RKHS?* http://www.gatsby.ucl.ac.uk/∼gretton/coursefiles/RKHS_Notes1.pdf, 2012.

[138] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012.

[139] Marco Loog, R. P. W. Duin, and R. Haeb-Umbach. Multiclass linear dimension reduction by weighted pairwise fisher criteria. *IEEE Trans. Patterns Anal. Mach Intell.*, 23(7):762–766, 2001.

[140] C. E. Thomaz and G. A. Giraldi. A new ranking method for principal components analysis and its application to face image analysis. *Image Vision Comput.*, 28(6):902–913, June 2010.

[141] D. Swets and J. Weng. Using discriminants eigenfeatures for image retrieval. *IEEE Trans. Patterns Anal. Mach Intell.*, 18(8):831–836, 1996.

[142] T.A. Filisbino, G.A. Giraldi, and C.E. Thomaz. Ranking methods for tensor components analysis and their application to face images. In *Graphics,*

Patterns and Images (SIBGRAPI), 2013 26th SIBGRAPI - Conference on, pages 312–319, Aug 2013.

[143] Carlos E. Thomaz and Gilson Antonio Giraldi. A new ranking method for principal components analysis and its application to face image analysis. *Image Vision Comput.*, 28(6):902–913, 2010.

[144] Demetri Terzopoulos M. Alex O. Vasilescu. Multilinear analysis of image ensembles: Tensorfaces. 447/460, 2002.

[145] J. Mairal, Piotr Koniusz, Z. Harchaoui, and C. Schmid. Convolutional kernel networks. In *NIPS*, 2014.

[146] Nadeem Akhtar and U Ragavendran. Interpretation of intelligence in cnn-pooling processes: A methodological survey. *Neural Computing and Applications*, pages 1–20, 2020.

[147] Y. Lecun, Y. Bengio, and G Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[148] K. He and J. Sun. Convolutional neural networks at constrained time cost. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5353–5360, 2015.

[149] Housam Khalifa Bashier Babiker and R. Goebel. Using kl-divergence to focus deep visual explanation. *ArXiv*, abs/1711.06431, 2017.

[150] Richard Connor, Franco Alberto Cardillo, Robert Moss, and Fausto Rabitti. Evaluation of jensen-shannon distance over sparse data. In Nieves Brisaboa, Oscar Pedreira, and Pavel Zezula, editors, *Similarity Search and Applications*, pages 163–168, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[151] Shaoxing Mo, Yinhao Zhu, Nicholas Zabaras, Xiaoqing Shi, and Jichun Wu. Deep convolutional encoder-decoder networks for uncertainty quantification of dynamic multiphase flow in heterogeneous media. *ArXiv*, 2019.

[152] Djairo G. de Figueiredo. *Análise de Fourier e Equações Diferenciais Parciais*. Edgard Blücher, 1977.